Supervised Preference Models: Data and Storage, Methods, and Tools for Application

by

István Papp

April 27, 2023

A Dissertation submitted in partial fulfilment of the Degree of Master of Philosophy

Department of Statistical Science
University College London

I, István Papp, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

In this thesis we present a variety of models commonly known as *pairwise comparisons*, *discrete choice* and *learning to rank* under one paradigm that we call preference models. We discuss these approaches together with the intention to show that these belong to the same family, and showing a unified notation to express these. We focus on supervised machine learning approaches to predict preferences, present existing approaches and identify gaps in the literature. We discuss reduction and aggregation, a key technique used in this field and identify that there are no existing guidelines for how to create probabilistic aggregations, which is a topic we begin exploring. We also identify that there are no machine learning interfaces in Python that can account well for hosting a variety of types of preference models and giving a seamless user experience when it comes to using commonly recurring concepts in preference models, specifically, reduction, aggregation and compositions of sequential decision making. Therefore, we present our idea of what such software should look like in Python and show the current state of the development of this package which we call *skpref*.

Impact statement

Supervised preference models have a wide range of applications, some of them include, understanding odds of sporting events, document retrieval, societal analysis of how people perceive certain events and what choices people make under certain conditions, recommender systems, product ranging decisions, congestion charge setting and many others.

Because of the wide impact of supervised preference models, research in the field has been scattered across many communities such as psychometrics, econometrics, statistics and machine learning. This research aims to bring them together via presenting a common mathematical formulation, building a common software toolkit for analysis and exposing ways to translate problem formulations via aggregation and reduction methods so that different approaches developed in these communities can be tried on the same problem.

We have built a new package in Python, available in GitHub, that could host all supervised preference models in a similar fashion to how *scikit-learn* hosts machine learning models today. We call this package *skpref* and believe that we have created a base package that could make a significant leap in the democratisation of preference models both in academia and in industry, with the help of some further development by the rest of the community. The *skpref* package aims to reduce the intellectual complexity that is associated with working with these models, introduces new best practices when it comes to software toolbox development for preference models and sets up the community to standardise the supervised preference modelling experience making it easier to spread new models and improvements discovered in the field. With potential academic and industry applications in the areas mentioned above.

As part of the research and the process of building *skpref* we encountered a lack of discussion about how to aggregate models in a probabilistic fashion. Therefore, we formalised some necessary conditions for a probabilistic pairwise to discrete choice aggregation logic and have proposed two methods which we have implemented in *skpref*, this enables researchers to understand the how learning a native probabilistic model in discrete choice compares to

using our proposed aggregation methods. We hope that proposing solutions in the pairwise comparison to discrete choice probabilistic aggregation space can serve as inspiration to eventually find ways to probabilistic aggregators from discrete choice to subset choice, which would be a significant leap for the community. Furthermore, we have detected a parallel between the pairwise to discrete choice probabilistic aggregation problem and pairwise coupling and have translated several pairwise coupling algorithms into what could be probabilistic pairwise to discrete choice aggregators. Pairwise coupling leads to computational efficiencies, whilst not compromising on prediction quality, such efficiencies are increasingly important in the world of big data. Detecting the similarity between the two problems and translating some of the solutions has, we hope, opened the door to such improvements also in discrete choice modelling.

Acknowledgements

I would like to thank dunnhumby for funding my studies and research as well as for being flexible, encouraging and kind during the process. I would like to especially thank: Rosie Prior, Giles Pavey, Dr. Richard Congram, Jason O'Sullivan and Richard Smith for allowing this project to happen.

I owe a great debt of gratitude to my supervisors Dr. Franz Király and Dr. Ioanna Manolopoulou whose patience, time, wisdom and guidance has shaped me into a better data scientist and researcher than I could have imagined at the start of the program. I want to especially thank Dr. Franz Király for playing a key part in my mathematical education, in helping me formulate the skpref package architecture and the mathematical set up for preference models. I want to also especially thank Dr. Ioanna Manolopoulou for the way in which she attended all of our meetings and always offered a highly valuable third pair of eyes on our work, and for increasing her role to becoming my primary supervisor to make sure that I get to finish my research in the final phase of my program. Thank you both for being kind mentors and for making our collaboration a joyful experience, which I will remember fondly.

I would also like to thank Pritha Gupta and Karlson Pfannschmidt for the many interesting conversations we had about using Python packages for preference models and for also helping shape our ideas on the skpref architecture.

To my family: Papp István, Papp Istvánné, Geöcze Mária, Rumiana Harizanova, Todor Belev, my father István and my mother Mina, my uncle Gábor, my brothers Rudolf and Richárd, and friends: Gabriela Martinez, Dr. Marios Michailidis, Robert Hurst, Philip Clarke and Maxime Beau. Thank you all for remaining supporting and understanding whilst I was juggling work and studies, for helping me not lose focus on and nurturing my well-being, remaining encouraging and for making sure that I have fun along the way.

Contents

Chapter 1

Introduction

1.1 Motivation

Every time people compare and relate alternatives to each other they express a preference. For example, when two sports teams play a game and one wins (one team scored more relative to the other) or they draw (they scored the same), when shoppers go to a store and buy some products but not others (shoppers express that they preferred the products purchased to those not purchased), when people watch movies and give them a star based rating (they express that movies that were given 5 stars are similar to each other but any one of them is better than movies given 4 stars), when runners compete (they are ranked from fastest to slowest runner).

The ability to mathematically map the relationship between alternatives, understand what drives these preference observations and the ability to predict future observations can be beneficial in several settings. Psychologists such as Thurstone (1927b) wanted to be able to map a mathematical distance between how gravely society perceives a crime, a wider adoption of this work could enable psychologists and sociologists to compare society across time and also to understand variation of attitudes within segments of society. Tversky (1972) wanted to understand what attributes of products influence the decisions of buying them mostly and how the introduction of certain new alternatives may change consumers' perceptions of existing alternatives. Answers to such questions can help retailers around the world plan the range of products they want to sell in a more informed manner. Statisticians such as Plackett (1975) wanted to improve on the way odds are calculated by betting agencies in horse races, finding

a more accurate way to calculate probabilities of certain outcomes is profitable to betting agencies. Today ranking techniques that are derivative of his original work are being used for document retrieval (Cao et al., 2007). Mathematicians such as Zermelo (1929) wanted to find an accurate way to rank chess players. There are over 50 million registered chess players on chess.com (chess.com, 2020), measures such as these help relate players to each other in this vast network and helps find match ups that are of similarly skilled players.

At the core of these problems lies the task of predicting correctly the outcome of preference relations. If we assume that something we observe can be predicted then it must be that there is some perhaps not immediately visible process that generates what we observe. We can begin to partially predict the outcome of a process by making simplified mathematical representations of it which we call models. Models attempt to capture some of the rules that might lead to the observed outcome. Even if the core objective of the researcher is exploratory, that is, the main purpose is not to make predictions but to enhance the understanding of the process that creates the observations, the way to have a measurable value of how much of the process the model explains is to see with what accuracy the model can make predictions. In this context, accuracy is defined by a mathematical measure of success that is used to evaluate how closely predictions resemble true observations. However, there is no single widely accepted universal measure of accuracy, the particular measure of accuracy used by the researcher depends on their particular objective of interest.

The simplifications that models use are also sometimes referred to as assumptions. When examining the assumptions of preference models, we can understand how they fundamentally differ from other well known approaches for modelling, such as classifiers and continuous models described in well known data science books like Hastie et al. (2009) and Bishop (2006). Conventional models would assume that preferences are expressed for each available alternative independently of what the other alternatives were. For example, under these models, a customer in a store would purchase Pepsi with the same likelihood if Coca Cola was also available or not. Preference models lift these assumptions so the alternatives presented have a direct impact on the preference expressed. For example, preference models would allow for the probability of buying a Pepsi not to be the same when Coca-Cola is also available. The ability to capture this complexity can provide more accurate models and a deeper understanding of how preferences might be generated and could lead to more accurate predictions.

In this thesis we present methods used for understanding what influences preferences and methods for predicting the preferences that will be expressed in the future, as well as methods which help us understand the probabilities with which some preferences might be expressed. Methods for predicting new preferences based on historically observed ones exist within the wider context of data science (see Donoho (2017)) which is a vast topic that covers areas

of statistical methods and computer science. Out of the six main divisions of what Donoho (2017) calls the field of "greater data science", we will cover three with a focus on how they apply to preferences: data representation/transformation, modelling and computing.

1.2 The outline of the thesis

Preferences are expressed by relating alternatives to each other. In **chapter 2** we unpack this statement in more detail, showing that preferences must have at least four components: a *set of alternatives* (e.g. laptops in a store), *a question* with which these alternatives can be compared (e.g. which laptop would you like to buy?), *a decision maker or decision process* that compares the alternatives and communicates how they relate to each other given that question, finally resulting in *a relation expressed* over the set of alternatives (e.g. the customer bought laptop A, indicating that they decided that for them laptop A was better than all the other laptops). There are different ways in which alternative relations can be expressed and this thesis centres around five commonly observed types: full orders (fully ranking all alternatives e.g. race results), partial orders (ranking alternatives with ties e.g. star based rating systems), subset choices (highlighting that a subset of alternatives are better than all alternatives not in the subset e.g. shoppers purchasing items in a store), discrete choices (choosing one alternative from many e.g. commuters deciding whether to drive, take the bus or ride a bike) and pairwise comparisons (choosing or tying two alternatives e.g. two teams playing against each other). We discuss what observations would look like for such data and the best practice for storing them computationally known as a normalised relational storage format (Harrington, 2016; Codd, 1970).

In **chapter 3** we begin to discuss how this data can be used to create accurate predictions for the outcome of future expressions of preference and to understand the process that might be generating the observed preferences by using supervised learning techniques Hastie et al. (2009), specifically discussing the methods that exist within supervised preference models. These models vary by the type of relation that they learn to predict, by the assumptions they make over the process that generated the observations, and by mathematical families to which they belong. We start by discussing the assumptions that these methods make and what potential complexities and shortcomings they might have, and then present the most popular and widely observed supervised preference models starting with the first known generalised linear model of Thurstone (1927a) until the more recent developments deep learning of Pfannschmidt et al. (2019a) We will show that there exists a large body of study for methods used to model full ranks, discrete choices and pairwise comparisons, however methods to

model partial orders and subset choices are much less explored. We follow with a discussion how to estimate preference models with numerical optimisation (Nocedal and Wright, 2006) and Majorisation-Minimisation techniques (Hunter and Lange, 2004).

Chapter 4 will present best practices for using preference models drawing from general data science best practices outlined in Hastie et al. (2009) and Bishop (2006). Starting with how to determine which model is the most successful at making accurate predictions and ways in which it is possible to improve their performance.

In **chapter 5** we explore ways to translate some relation types into others, for example the subset choice of {A, B} from {A, B, C} can be expressed as the following pairwise comparisons A is better than C and B is better than C. This technique is particularly useful when methods for modelling certain type of relations, such as subset choices have not been so well developed compared to other models such as pairwise comparison models. These transformations allow researchers to translate the problem into another type of problem (subset choice to pairwise comparison in this case) that might have a more robust or a larger variety of methods available. This approach sits in the wider context known as model reductions (Beygelzimer et al., 2005). Once a reduction has been applied, the researcher needs to find a way to express the predictions of the reduced model in the original problem space, for example if a researcher translates a subset choice into pairwise comparisons and used a model to predict these pairwise comparisons, they might still need to find a way to express the pairwise comparison predictions as a subset choice. We will refer to the process of translating back into the original space as aggregation.

In **chapter 6** we discuss the current state of availability of preference models in computational software, focusing on the Python language discussing packages such as `choix` (Maystre, 2020), `pylogit` (Brathwaite and Walker, 2018) and `cs-rank` (Pfannschmidt et al., 2019a) in the context of software best practices (Király et al., 2021) and the current state of the art data science package of `scikit-learn` (Pedregosa et al., 2011). We show that currently there is a lack of an equivalent package to `scikit-learn` for preference models. This means that researchers have to use packages of varying architectures many of which do not follow current best practice in the realm of data science software development. The disadvantage of not having such a package for preference modelling is that it may lead to lost time invested in having to learn many different ways of coding; modelling inaccuracies and a general lack of engagement with preference models in the scientific community. In this chapter we also present our own ideas for a software architecture (`skpref`) that we believe can bridge over these gaps (Papp et al., 2021).

Finally, in **chapter 7** we present further research questions that we have identified whilst

researching this topic.

1.3 Our contributions

The ideas we present in this thesis are: a mathematical abstraction that enables the discussion of different types of preference models, a methodological contribution of probabilistic aggregation, a software contribution and a documentation contribution, so these are scattered across different parts of the document.

- In chapter 2 we present a mathematical abstraction that can be used to discuss different types of preference models. We will focus the discussion on the five types we outlined above (full orders, partial orders, subset choice, discrete choice and pairwise comparisons).
- For the methodological contribution we present a formula for aggregating pairwise comparison probabilistic predictions into discrete choice probabilities. This can be found in section 5.3.2.
- For the software contribution we present the conceptualisation of a Python package that can host preference models in a way that improves the current infrastructures for many packages. This can be found in section 6.2. We also created a minimum viable product for this package to demonstrate some of the core ideas, which can be found in GitHub together with some documentation available on GitHub pages, which include some jupyter notebooks of worked examples from the package [1] [2], these can also be found in the appendix also (Papp et al., 2021). The aggregation solution we proposed in section 5.3.2 can also be found and used in `skpref`.
- Finally, we also consider this detailed review of the state of preference models a contribution in itself.

1.3.1 Probabilistic aggregations of pairwise comparisons to discrete choice

We have identified some straightforward approaches for predicting an outcome by aggregation, however, when it comes to predicting a probability of an outcome by aggregation we have found no research showing how to do this. We explore potential ways in which this can be done starting from the simplest possible case, which is that of aggregating probabilistically

pairwise comparisons into discrete choices. That is, given that we know the following pairwise probabilities: the probability with which A is preferred to B and the probability with which B is preferred to C and the probability with which A is preferred to C, how can we express the probability that A would be chosen from {A,B,C} in a discrete choice problem? This is an important question because currently researchers using aggregation techniques cannot predict the probability of an outcome, which is a disadvantage of the strategy to make reductions.

The problem of making probabilistic aggregations from pairwise comparisons to discrete choices does not have a unique solution, therefore, we begin by formalising mathematical conditions which should be necessary for a plausible probabilistic aggregation technique and propose some methods which satisfy these conditions and could be used. An analogous problem can be found in the literature for aggregating binary classifiers into multiclass classifiers, which is called pairwise coupling (Wu et al., 2004). We also propose ways in which pairwise coupling aggregating methods could be translated into a probabilistic aggregation from pairwise comparisons into discrete choices, though these have not been tested and remain as proposals.

1.3.2 Building a Python package tailored for preference models

To tackle the lack of a unified interface for preference models of comparable quality to scikit-learn, we have designed a package in Python with the working name of *skpref* that can host preference models for all five types of preference relations we mentioned (full order, partial order, subset choice, discrete choices, and pairwise comparisons) and considers that the best form of storing the data is in relational databases and that reduction is a common practice amongst researchers in the field. Note that a large part of the contributions of our research is not contained within this document but is the code in the GitHub repository of *skpref*. Building a machine learning interface around the assumption of a relational database, would be a new way of designing data science packages that parts with and in our opinion enhances the best practice set by scikit-learn, which is something we explore in the discussion in section 6.1.1. In section 6.2 we present a design and snapshots of the current state of the prototype for skpref.

1.4 Further gaps identified in the field as possible research questions

In **chapter 7** we present research questions and possible areas of work that researchers can contribute to in the future.

- Extending Bradley-Terry trees (Strobl et al., 2011) with boosting and random forests.
- Breaking down subset choices into two models that work together, one that predicts the number of items chosen in the subset and another that predicts each alternative that will be in the chosen set.
- Speeding up gradient descent on Bradley-Terry models using a normalised data set-up as an extension of the work presented by Kumar et al. (2015).

Chapter 2

Observation, mathematical expression and computational storage of preferences

2.1 What is a preference?

If one embarks on the journey of preference modelling, one also realises that it is surprisingly difficult to explain specifically and exhaustively what a preference is. Whilst there are key things in common for most works describing preferences, there seems to be no global definition of what a preference actually is. For the purposes of this report, a preference is something that must have the following components:

- *A set of alternatives*. This can be anything ranging from: products in a store, options of transportation, homes to rent and many more.
- *A question over the set of alternatives*, which can be: "Which products would you rather buy from this list of products?", "Which house would you rather rent", "Would you take a cab or a bus?", "Which team has won the match?", "Which box is the heaviest?" or "How would you rank these three things?".
- *A relation expressed over the set of alternatives for a given question*. This can have many forms, some of which will be explored in this thesis, for example to the question

"Which products would you rather buy?" the relation can be a choice in the form of a selection of a subset of the alternatives. Or to the question "Which Sushi was the best tasting?" a relation expressed could be a fully ranked order of all the Sushis that were tried from best to worst. A large list of different types of relations that can be expressed is captured by Öztürké et al. (2005), however even this 36 page long account of all the different relations that can be expressed claims to be non-exhaustive. In this thesis we will focus on just a few of these which we believe capture a large number of applications which we will present in detail in this chapter: pairwise comparisons, discrete choices, subset choices, partial orders and full orders.

- *A deciding process or choice maker*. Someone, something or a process that is presented the set of alternatives and generates a relation over this set of alternatives based on the question asked over the set of alternatives. Though preference models were initially developed by psychometricians to analyse the perception of alternatives by people (Thurstone, 1927c) and then further perfected by econometricians (Train, 2009), the application of the mathematical abstraction goes beyond human decisions and could be applied to anything where a relation is expressed over a set of alternatives. In the case of a person it could be someone expressing which flavour in a product they prefer, however, the choice-maker could also be a decision process based on a set of rules, for example a match defined by "which team has scored the most goals?" is a process that selects one winner from a set of two teams. These both have a set of alternatives over which a relation is expressed with regards to a question. As further examples, Zermelo (1929) and Plackett (1975) have studied models whose mathematical abstraction would count as preference models given the definition above, but they were in the context of chess matches and horse races respectively where the decision process was the match or the race. In a lot of historical literature these are rarely discussed together with human preferences, so we would like to underscore that the mathematical abstraction of preference models has applications that go beyond just the study of human preference.

Preferences can be described as stated or revealed. A stated preference is one where a choice-maker says what their expressed relation over a set of alternatives would be if they would have to hypothetically act upon a question over the set of alternatives. A revealed preference, formalised by Samuelson (1938), considers what relation the deciding processes or deciding person actually expressed over a set of alternatives when they acted upon a question. The different ways in which revealed choices can be categorised is summarised in Varian (2006). The discrepancy between stated and revealed preference is a field of study on its own. Some articles include: Urama and Hodge (2006); Lambooij et al. (2015); Engström and Forsell (2018); Vasanen (2012). Although in this thesis we will be mostly referencing

examples of revealed preferences, the modelling techniques we describe in the next chapter could be used for both.

2.2 Types of relations expressed over a set of alternatives

A relation expressed over a set of alternatives is a key component of preference, in this section we will examine five types of relations that can be expressed over a set of alternatives: pairwise comparisons, discrete choice, subset choice, partial order and full order. We will show an intuitive example of each of these, followed by mathematical notation that can be used to express relations in each of these categories. We will begin by introducing some general notation that will be used throughout this text, to help our discussion.

2.2.1 Notation for expressing relations over a set of alternatives

(i) We write $\mathbb{B}$ for the boolean domain, i.e., we write $\mathbb{B} := \{1, 0\}$ where 0 is called "False" and 1 is called "True".

(ii) For a set A, we write $\mathcal{P}(A)$ for the power-set of A.

(iii) For two random variables X and Y, we will use $X \perp\!\!\!\perp Y$ to denote X is independent from Y and $X \not\perp\!\!\!\perp Y$ to denote that X and Y are not independent.

(iv) For $x, y \in \mathbb{R}$, we will use $x \approx y$ to denote that x is approximately equal to y.

(v) We will use $\#A$ to denote the cardinality of the set A.

(vi) We will refer to a vector of k ones as $\mathbb{1}_k$.

(vii) For a discrete random variable Y and the values this random variable can potentially take y we will define the probability $P : Y, y \to \mathbb{R}_{[0,1]}$ of a certain outcome by $P(Y = y)$.

Definition 1. A (binary) relation on a set A is a function $r : A^2 \to \mathbb{B}$. For $s, t \in A$, we write $s \succeq_r t$ iff $r(s, t) = 1$. When an alternative is being compared to itself, the binary relation is always one $r(s, s) = 1$. For example, if someone said that they preferred a *latte* to *black coffee* and are indifferent between *black coffee* and a *tea* then $r(\text{latte}, \text{black coffee}) = 1$, $r(\text{black coffee}, \text{latte}) = 0$, $r(\text{black coffee}, \text{tea}) = 1$, $r(\text{tea}, \text{black coffee}) = 1$, $r(\text{tea}, \text{tea}) = 1$, $r(\text{black coffee}, \text{black coffee}) = 1$ and $r(\text{latte}, \text{latte}) = 1$.

■

(ix) For two alternatives a and b, we denote a is preferred to b as $a \succ b$, which means $r(a,b) = 1$ and $r(b,a) = 0$ and we denote indifference between a and b as $a \sim b$, which means that $r(a,b) = r(b,a) = 1$. We use this notation to express ranks as well, for example, saying that a is preferred to b which is preferred to c can be expressed as $a \succ b \succ c$. The implicit relations behind this notation is that $r(a,b) = 1$, $r(b,a) = 0$, $r(b,c) = 1$, $r(c,b) = 0$, $r(a,c) = 1$ and $r(c,a) = 0$.

Definition 2. Consider a set of alternatives A. A preference expressed over this set of alternatives would be **transitive** when the following is true: if $a \succ b$ and $b \succ c$ then $a \succ c \; \forall a, b, c \in A$. ■

2.2.2 Examples of types of preferences that can be expressed

In this section we will show examples of the five types of preference relations that can be expressed focusing on two components: the set of alternatives and the relation expressed over this set of alternatives.

Full order

A **full order** is a relation where every element in the set of alternatives is ranked. An example of a natural domain in which we can observe these are certain sports competitions that involve racing. These usually end up with a fully ranked result where each of the competitors have been given a position in which they finished the race. The terms full rank, full order and total order can be used interchangeably.

Consider table 2.1, a way of representing these observations, imagine that each row is a different race, the first column *drivers* shows the set of drivers in the race and the second column shows how they have finished the race. So in the first row, racer "a" has finished the race in 4th position and racer "c" has won.

Table 2.1: Example of full ranks

drivers	ranked drivers in race
{a, b, c, d}	$c \succ b \succ d \succ a$
{a, b, c, d}	$d \succ b \succ c \succ a$
{a, b, c}	$c \succ b \succ a$
{a, b, c, d}	$d \succ c \succ b \succ a$

Partial order

In a **partial order** rankings may tie, for example the top 5 items are ranked 1 to 5 and then the rest are tied but understood that they are all ranked inferior to the fifth ranked alternative. These can be observed in some election data, where voters might rank their top x choices and leave the rest unranked and also in systems where people give star-based ratings to alternatives, many items might have a $5/5$ star ranking and then others will get a $4/5$ star ranking. We know that those that have 5 stars are preferred to those that have 4 stars, however, we cannot know amongst those ranked 5 stars which one is the most preferred one. This can be seen in examples in table 2.2, where for each title viewed we have a number that indicates how many stars that viewer gave that title and each row is a different viewer.

Table 2.2: Example of star based rating system

movie ratings
{Star Trek Discovery: 5, Star Wars Return of the Jedi: 4, Travels with my father: 5}
{Star Trek Discovery: 1, Star Wars Return of the Jedi: 5, Travels with my father: 5}
{Star Trek Discovery: 5, Star Wars Return of the Jedi: 5}

Subset choice

In a **subset selection** it is possible to select more than one of the alternatives as preferred to the other not selected alternatives. For example, shopping in a grocery store, shoppers usually buy more than just one item. We know at the moment of purchase, that for that specific decision, the items purchased are preferred to those not purchased, however, we do not know any order of preference amongst those that have been purchased or amongst those that have not been purchased. This means that subset choices are a special case of partial orders, where there are only two ranks available, tied first amongst those that were chosen and tied last amongst those that were not. Examples of this can be seen in table 2.3. In the first row we see an example where everything was purchased, since here we cannot know which item is preferred between purchased ones this is the equivalent of all products tying. In the second row we can see that products “a” and “c” are preferred to “b” since “b” has not been selected whereas the other two have been.

Table 2.3: Example of shopping observations

items purchased	items in store
{a, b, c}	{a, b, c}
{a, c}	{a, b, c}
{a}	{a, d}
{d}	{b, d}

Discrete choice

In a **discrete choice** the deciding process or decision maker only chooses one alternative from the set of alternatives, which is a special case of subset choice selection, but instead of top x the decision makers or processes select the top 1. It's used widely for modelling choices made in transportation, where an individual can naturally be only taking one mode of transportation at a time (it is rare to see someone riding a bike whilst driving a car). Examples of this can be seen in table 2.4, where each row is a commuting decision, the first column is the mode of transport taken, and the second column is the available options.

Table 2.4: Example discrete choice in commuting data

transportation used	transportation available
bicycle	{train, bicycle}
train	{train, car, bicycle}
car	{train, car, bicycle}
train	{train, bicycle}

Pairwise comparison

For **Pairwise comparisons** only two alternatives are presented to decision makers or decision processes, examples could be football matches, where there are only two teams playing at a time and either one team wins or there's a draw.

In table 2.5 we show examples of what US college basketball matches could look like. In the first column we identify the winning team, and in the second column we identify the two teams that played. The defining characteristic of pairwise comparisons is that the number of alternatives presented is always two. In this case we don't show ties, since in basketball one team always has to win. In the case of a tie the **winning team** column could take values

either as an empty set or as the set of both teams.

Table 2.5: Example of pairwise comparison observations in basketball

winning team	teams that played
Virginia	{Purdue, Virginia}
Auburn	{Auburn, Kentucky}
MI State	{MI State, Duke}

2.2.3 Mathematical formulation that describes these five types of relations

In this section we will show a mathematical formulation that can be used to describe the above mentioned types of relations. First we introduce some additional notation, and then we go through each of the examples, showing how they fit into this notation.

(i) For the set A, let $M \in \mathbb{B}^{\#A \times \#A}$ be a $\#A \times \#A$ matrix whose ith row and jth column would contain $r(A_i, A_j)$ where $A_i, A_j \in A$. For example, for $A = \{a, b, c\}$,

$$M = \begin{bmatrix} r(a,a) & r(a,b) & r(a,c) \\ r(b,a) & r(b,b) & r(b,c) \\ r(c,a) & r(c,b) & r(c,c) \end{bmatrix}.$$

The type of relations that can be expressed by decision makers / processes that we will investigate in this thesis (full order, partial order, subset choice, discrete choice and pairwise comparisons) can be expressed by a collection of binary relations that are represented in this matrix format, we will proceed by going through each relation types in turn and show how this works, after a few more definitions.

(ii) For a specific property of a relation p (e.g. antisymmetric, fully-ordered), let $\mathcal{R}_p(A) \subseteq \mathbb{B}^{\#A \times \#A}$ denote the set of matrices that can be produced for a specific relation type on the set of alternatives A. Some of the more relevant properties in this thesis that p can take is shown in table 2.6

We may express a combination of properties using the following notation $\mathcal{R}_{a \vee b} = \mathcal{R}_a \cup \mathcal{R}_b$. For example $\mathcal{R}_{wo \vee as}$ is the combination of all antisymmetric and weak order relations in a set.

Table 2.6: The properties of relation sets in a choice modelling context

Property (p)	**name for elements of** $\mathcal{R}_p(A)$	**name for** $\mathcal{R}_p(A)$ **on** A	**symbol for** $\mathcal{R}_p(A)$
antisymmetric	Preference	N/A	$\mathcal{R}_{as}(A)$
is partial order	transitive preference	poset	$\mathcal{R}_{po}(A)$
is weak order	tied ranking	weakly ordered set	$\mathcal{R}_{wo}(A)$
weak order with two ranks	subset choice or discrete choice	sub-set or element	$\mathcal{R}_{ch}(A)$
is well-order	ranking	(totally) ordered set	$\mathcal{R}_{to}(A)$

Full order

If a relation set is antisymmetric then the binary relations $r()$ on the set A are such that $r(s,t) \neq r(t,s), \forall s,t \in A$ where $s \neq t$. What this means for sets of choices is that for every pair of items that has been presented to a choice maker, the choice maker knows which one from the two they prefer so there are no ties communicated.

Total orders are in $\mathcal{R}_{as}(A)$ where $\#A > 2$ with the added criteria that these must be transitive. If a full order were to be expressed over a set of alternatives $A = \{a, b, c\}$, then the domain $R_{to}(A)$ would have all the combinations of

$$\begin{bmatrix} r(a,a) & r(a,b) & r(a,c) \\ r(b,a) & r(b,b) & r(b,c) \\ r(c,a) & r(c,b) & r(c,c) \end{bmatrix} \in \mathcal{R}_{to}(\{a,b,c\}) \tag{2.1}$$

subject to $r(s,t) \neq r(t,s), \forall s,t \in \{a,b,c\}$ where $s \neq t$ and transitivity.

The following result $a \succ b \succ c \in R_{to}(A)$ would look like this:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

A way to detect fully ordered alternatives is that when a full ordered set is left-multiplied by the vector of ones then no two elements of the resultant vector contain the same rank and

such a multiplication retrieves the ranks of the alternatives.

$$(\mathbb{1}_3)^T \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} \text{rank of a}, & \text{rank of b}, & \text{rank c} \end{bmatrix} \tag{2.3}$$

Partial order

We can describe partial orders by $\mathcal{R}_{wo}$ which includes all the combinations where there is at least one tied rank. For a set of three alternatives there are seven possible combinations.

Where there are ties in the first position: $a \sim b \succ c$, $a \sim c \succ b$, $b \sim c \succ a$

ties in the second position: $a \succ b \sim c$, $b \succ a \sim c$, $c \succ a \sim b$

all tie: $a \sim b \sim c$.

These are represented the following way respectively

$$\mathcal{R}_{wo}(\{a,b,c\}) = \left\{ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right\} \tag{2.4}$$

.

When we left multiply these with $(\mathbb{1}_3)^T$ we get the modified ranking:

ties in first position: $\begin{bmatrix} 2 & 2 & 3 \end{bmatrix}$, $\begin{bmatrix} 2 & 3 & 2 \end{bmatrix}$, $\begin{bmatrix} 3 & 2 & 2 \end{bmatrix}$

ties in second position: $\begin{bmatrix} 1 & 3 & 3 \end{bmatrix}$, $\begin{bmatrix} 3 & 1 & 3 \end{bmatrix}$, $\begin{bmatrix} 3 & 3 & 1 \end{bmatrix}$

all tie: $\begin{bmatrix} 3 & 3 & 3 \end{bmatrix}$

To get a standard ranking, we would need to modify definition 1 such that ties are represented with a 0 rather than a 1. So when $b \sim c$ then $r(b,c) = 0$, however keeping the relation to oneself $r(a,a) = 1$. We do not see how we could achieve dense ranking with a similar, simple modification on definition 1.

Subset choice

The situation becomes mathematically trickier when expressing subset choices, since here the options of choosing nothing and everything become available. Consider a set of three alternatives $A = \{a, b, c\}$. When everything is chosen all the relations $r(x, y) = 1 \forall x, y \in A$, since neither individual element is preferred to the other. The result is exactly the same when none of the alternatives are chosen, since then also no alternative is expressed as preferred to any other. However, in practice for subset choice we need to be able to mathematically distinguish between the cases of everything being chosen vs. nothing being chosen. Using only the binary relation notation on the elements of the set of alternatives, it is impossible to distinguish between the cases of everything being chosen vs. nothing being chosen, because in both situations all alternatives tie.

The solution is to express binary relations not on the elements of the set of alternatives in A, but on the power-set of A. This way all potential subsets are treated like individual alternatives, and one of those will be chosen, with there being a clear distinction between nothing and everything being chosen. In these cases the set of all binary relations $\mathcal{R}_{ch}(\mathcal{P}(A))$ becomes a $2^{\#A} \times 2^{\#A}$ matrix. As an example, consider the case where there are two elements in the set $A = \{a, b\}$. The power set of A would be $\mathcal{P}(A) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$, and $\mathcal{R}(\mathcal{P}(A))$ would have matrices with the following elements:

$$\begin{bmatrix} r(\emptyset, \emptyset) & r(\emptyset, \{a\}) & r(\emptyset, \{b\}) & r(\emptyset, \{a, b\}) \\ r(\{a\}, \emptyset) & r(\{a\}, \{a\}) & r(\{a\}, \{b\}) & r(\{a\}, \{a, b\}) \\ r(\{b\}, \emptyset) & r(\{b\}, \{a\}) & r(\{b\}, \{b\}) & r(\{b\}, \{a, b\}) \\ r(\{a, b\}, \emptyset) & r(\{a, b\}, \{a\}) & r(\{a, b\}, \{b\}) & r(\{a, b\}, \{a, b\}) \end{bmatrix} \tag{2.5}$$

.

In this case $\{a, b\}$ being chosen as a subset would be expressed by the matrix

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

.

When these matrices are left multiplied by a vector of ones of the same length then the

resultant matrix would be the equivalent to a ranking where ties are given the lowest rank.

$$
\begin{aligned}
(\mathbb{1}_4)^T \times \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} &= \begin{bmatrix} 4 & 4 & 4 & 1 \end{bmatrix} \\
&= \Big[\text{rank of } \emptyset, \quad \text{rank of a}, \text{rank of b}, \text{rank of a and b}\Big]
\end{aligned}
\tag{2.7}
$$

.

We can see that subset choice is a special case of partial order where there are only two ranks.

Power-sets and subsets of power-sets can be also used as the input to $\mathcal{R}$ if responses are preferences or rankings between subsets of the alternatives for example preferring $\{a, b, c\}$ to $\{z, m, d\}$.

Using the power-set of the set of alternatives with partial orders Note that when we're working with partial orders there is only one case where all alternatives tie, so there is no mathematical requirement to use binary relations over power-sets to express relations within partial orders. However, this technique can be still used to gain more interesting insights into the relationship between alternatives such as finding complementarity between products. For example, if a person always drinks milk with coffee, but doesn't enjoy milk or coffee by themselves, then it might be observed for this person that $\{\text{Milk}, \text{Coffee}\} \succ \emptyset \succ \{\text{Milk}\} \sim \{\text{Coffee}\}$.

Discrete choice

In the case where one item is chosen from a set of alternatives there is only one element s of the set A for which the following holds $s \in A$ s.t. $r(s, t) = 1$ and $r(t, s) = 0, \forall t \in A, t \neq s$. In our example where a commuter making a choice between going via a bicycle, train or a car, each matrix in $\mathcal{R}_{ch}$ would contain the following information:

$$
\begin{bmatrix} r(\text{bicycle}, \text{bicycle}) & r(\text{bicycle}, \text{train}) & r(\text{bicycle}, \text{car}) \\ r(\text{train}, \text{bicycle}) & r(\text{train}, \text{train}) & r(\text{train}, \text{car}) \\ r(\text{car}, \text{bicycle}) & r(\text{car}, \text{train}) & r(\text{car}, \text{car}) \end{bmatrix}
\tag{2.8}
$$

and

$$\mathcal{R}_{ch}(\{\text{bicycle}, \text{train}, \text{car}\}) = \left\{ \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \right\} \tag{2.9}$$

Where the matrices would show the following information: a bicycle was chosen, a train was chosen and a car was chosen respectively. Here it is easy to see why discrete choices are a special case of partial orders. Consider the matrix that represents a train being chosen. If we left multiply this matrix by a vector of ones we will get the ranks for each of the alternatives revealing that discrete choices are partial orders where the chosen object has a rank 1 and the not chosen objects all tie in rank $\#A$.

$$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 3 \end{bmatrix} = \begin{bmatrix} \text{rank of bicycle}, & \text{rank of train}, & \text{rank of car} \end{bmatrix} \tag{2.10}$$

Sometimes in discrete choices choosing nothing is an option, in that case the option of choosing nothing is treated as an alternative itself and the set of available alternatives A can be redefined as $\{Q : Q \in \mathcal{P}(A), \#Q \leq 1\}$.

Pairwise Comparisons

Note that pairwise comparisons are defined by the fact that the number of alternatives in the list of alternatives presented is two. Therefore for all of the above mentioned cases, the number of alternatives presented must be greater than two, otherwise, if it is two, then it would be a pairwise comparison.

An example of pairwise comparison is a basketball match. In this case no ties are allowed and we can represent the relation expressed between two teams a and b as the antisymmetric relation set $\mathcal{R}_{as}(\{a, b\})$ as two different 2×2 matrices where each alternative is represented in a row and column and each element represents the binary relation of that row's and column's alternatives like so:

$$\mathcal{R}_{as}(\{a, b\}) = \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \right\} \tag{2.11}$$

Which is the representation of the following: either team b will win or team a will win.

Example of representing pairwise comparisons where ties are allowed For the weak ordered set, ties are allowed: $\exists s, t \in A$ such that $r(s,t) = 1$ and $r(t,s) = 1$. This could be a football match where teams are allowed to be tied. In the cases of teams a and b the relation set with weak orders is $\mathcal{R}_{wo}(\{a,b\}) = \left\{ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right\}$ representing the two teams playing a draw (if both are better than or equal to each other then by definition they must both be equal to each other). Combining it with $\mathcal{R}_{as}(\{a,b\})$ we can get the expressions to any result of a football match: either team b will win, team a will win, or they will play a draw. Recall that we denote by $\mathcal{R}_{wo\vee as}(A)$ as all the weak order and antisymmetric relations we can express on a set A,

$$\mathcal{R}_{wo\vee as}(\{a,b\}) = \left\{ \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right\} \tag{2.12}$$

.

2.2.4 Simplifying notation in special cases

In order to simplify notation on ordered and transitive cases we will express the matrices in $\mathcal{R}$ for what they correspond to. For example in the case of $\mathcal{R}_{wo\vee as}(\{a,b\})$ the matrix $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ will be also referred to as $\{b \succ a\}$, the matrix $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ will be referred to as $\{a \succ b\}$ and finally $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ as $\{a \sim b\}$ or $\{b \sim a\}$.

Definition 3. *The choice function*

To simplify referring to choice matrices we are introducing a choice function $c : \mathcal{P}(A) \times A \to \mathcal{R}_{ch}(\mathcal{P}(A))$ where A is a set of alternatives available and the first argument in $c()$ corresponds to the choices that have been made $A' \subseteq A$. For $a, b \in A$, the output of $c()$ is the element in $\mathcal{R}_{ch}(\mathcal{P}(A))$ such that:

$$r(a,b) = \begin{cases} 0 \text{ if } a \in A \setminus A' \text{ and } b \in A' \\ 1 \text{ otherwise} \end{cases}$$

.

For example from matrix 2.5 choosing a would be denoted by

$$c(\{a\}, \mathcal{P}(\{a, b\})) = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

■

2.3 Preference observations and preference data

In the previous section we have described the types of relations that can be expressed over a set of alternatives. Recording the relations expressed by decision makers or decision processes together with the set of alternatives (over which the relations are expressed) is the basic step of preference data collection. Preference data can also be made up of further information which we describe in this section. It is important to split these additional data sources from the start into three categories, because this type of information is very likely to be stored in different data tables, the exact reasons why are explored soon in section 2.4.1. Making this categorisation will also help us discuss how certain models need to treat information coming from these data sources differently, which is something we dive deeper into in section 3.3.1 specifically in lemma 3.3.2. For now, we present the three different categories:

- **Decision level data**, includes the set of alternatives presented to a decision maker, and what relations they have expressed over these alternatives, there may be other recordings that describe the circumstances of the decision, such as the temperature on a day someone went shopping. This generally contains information that varies by decision. Note that this is not data about the decision makers / processes, but rather data that describes the circumstances that are unique to each decision, whereas a decision maker / process can make several decisions under different circumstances, for example, shopping on different days.
- **Alternative level data**, are recordings that describe the alternatives, such as the price of a product. This generally contains information that varies by alternative, but for the same alternative would be constant across different decisions, for example, different shoppers facing the same price for the same product.

- **Decision maker / process level data** are recordings that describe the decision makers and processes, such as the age of a shopper. Once again, these are observations that vary by decision maker, but for the same decision maker they would be constant across different decisions. In this thesis we are not focusing on temporal elements, so perhaps this specific example is not technically correct since, people and people's circumstances change over time, over a long enough period of time, the age of a shopper will not be the same, the number of children a shopper has or their level of income might not be the same, however, for the purposes of this report we consider these to be constant in our data. For using data that is over such a long period of time that this would cause problems then this should be mitigated by considering temporal elements also, we leave this out of scope for this document.

To illustrate how this information might be relevant to estimation, imagine if we are trying to understand ice-cream and soup purchases. Choice-specific information might be able to shed a light on whether people in general might be less likely to choose ice cream over soup in colder weather, alternative level information might help confirm that cheaper ice-creams of the same flavour are more likely to get purchased, and finally choice maker level information could help investigating whether children are more likely to chose ice cream over soup than adults. Note that not all types of data are necessary to formulate a preference problem. The only type of data that is necessary is decision level data and within that only the set of alternatives presented and the relations expressed over these sets of alternatives are crucial. So, it is not the case that a researcher must have all the types of data available to them to work on preference problems.

All the information that has been recorded pertaining to an expressed relation over a set of alternatives is what we call an observation. A dataset is a collection of observations that is recorded somewhere (usually in a computer). We now describe the three types of information in more detail, showing some practical examples of these observations first, and then we follow it by introducing mathematical formalism around it.

2.3.1 Examples of preference data and observations

In this section we provide example datasets for each of the five different types of relations that can be expressed but with the additional decision level, alternative level and decision maker / process level data where applicable. We will also refer to some of the examples presented in this section throughout the report.

Bookmaker for Formula 1 races - full order

Consider data recorded about historical races where the drivers could be considered the alternatives, the rank in which they have crossed the finish line as the relation expressed over these alternatives and the race as the deciding process over the question "in what order did the drivers complete the race?". There might be further information available specific to the drivers and their vehicles (alternative level variables) and variables relevant to the decision process, such as weather, the track. These are shown in table 2.7. Note that in this example there is no decision maker / process level data.

Table 2.7: Example of race observations

Decision level data				
race id	**finishing position**	**drivers**	**weather**	**location**
1	$c \succ b \succ d \succ a$	{a, b, c, d}	sunny	Budapest
2	$d \succ b \succ c \succ a$	{a, b, c, d}	sunny	Monaco
3	$c \succ b \succ a$	{a, b, c}	sunny	Monza
4	$d \succ c \succ b \succ a$	{a, b, c, d}	rainy	Silverstone

Alternative level data	
driver	**car**
a	Ferrari
b	McLaren
c	Renault
d	McLaren

Star based rating systems - partial order

It is commonplace in recent services to capture user feedback by asking people to rate products from 1-5 stars. A video streaming service, Netflix, has used such a system until 2017, where movie watchers were submitting their ratings and a merchant platform Amazon also uses a 1-5 star product rating system and a food delivery platform, Deliveroo uses such a system for customers to rate restaurants on the platform. Users create a tied ranking, for example, all movies a user ranked at 5 stars are understood to be better than all movies they ranked 4 stars. However, there is no difference communicated between those that have 5 stars or those that have 4. In table 2.8 we show an example of decision level data - the ratings themselves, decision maker level data - information about the people making the ratings and alternative level data - information about the movies being rated.

Table 2.8: Example of star based rating system

Decision level data

person id	**movie ratings**
1	{Star Trek Discovery: 5, Star Wars Return of the Jedi: 5, Travels with my father: 5}
2	{Star Trek Discovery: 1, Star Wars Return of the Jedi: 2, Travels with my father: 5}
3	{Star Trek Discovery: 5, Star Wars Return of the Jedi: 5}

Decision maker / process level data

person	**age**	**sex**
1	30	male
2	25	female
3	45	male

Alternative level data

movie	**year of release**	**director**
Star Trek Discovery	2017	Bryan Fuller and Alex Kurtzman
Star Wars Return of the Jedi	1983	Richard Marquand
Travels with my father	2017	Jack Whitehall

Shopping in a grocery store - subset selection

Consider transactions in a grocery store. The alternatives could be considered to be the products that are sold in a store, the decision makers would be the shoppers visiting that store, and the relation expressed is a subset selection of items purchased. The decision level data is the transactional data for each basket where we see the products purchased and for each store we know the list of alternatives available. There might be further information: alternative level data and choice maker level data. Alternative level data in this case would be data about the products and would most likely consist of their attributes, such as the pack size and the price of the item. Choice maker level data in this case could be information the owner has that is specific to the customers, such as whether in general they buy yoghurt that is flavoured or not, or whether they tend to buy more or less expensive items. Table 2.9 shows an example of how shopping data might be represented as a subset choice in relational data format.

Table 2.9: Example of shopping observations

Decision level data

customer id	**items purchased**	**items in store**	**temperature at purchase time (C)**
1	{a, b, c}	{a, b, c}	22
2	{a, c}	{a, b, c}	15
3	{a}	{a, d}	24
4	{d}	{b, d}	10

Alternative level data

Product	**flavour**	**price (GBP)**
a	vanilla	2
b	chocolate	2.50
c	vanilla	1.20
d	natural	1

Decision maker / process level data

customer id	**age**
1	25
2	65
3	40
4	16

Taking public transport - discrete choice

Consider another popular example, which is the discrete choice of choosing methods of transportation. The choice makers are the commuters who decide to choose from the different forms of commuting available to them such as cars, cycling and public transport, which in turn are the set of alternatives. This is an interesting case, because here the set of alternatives available is defined by the commuters (e.g. do they have a car or not?) as well as by destination of travel, for example, whether the destination is accessible via public transport. In this case the set of alternatives are shown as decision maker / process level data not as decision level data. However, unless otherwise specified we will consider the set of alternatives available as decision level data. The type of relation expressed is choosing one alternative from several. Examples of alternative level data would contain the cost of each commuting option and the length of the journey. Examples of choice maker level data would contain income, fitness level, and examples of decision level data would contain time of day and weather. We show these together in table 2.10

Table 2.10: Example of commuting observations

Decision level data

commuter id	**transportation used**	**weather**
1	bicycle	sunny
2	train	sunny
2	car	sunny
3	train	cloudy

Decision maker / process level data

commuter	**income (high, med, low)**	**transportation available**
1	med	{train, bicycle}
2	high	{train, car, bicycle}
3	med	{train, bicycle}

Alternative level data

vehicle	**cost (GBP)**
bicycle	1
train	7
car	20

Basketball matches - pairwise comparisons

For pairwise comparisons consider basketball matches, where the set of alternatives are two teams playing a match and the deciding process is the match itself. The questions over which

the deciding process picks out one team over the other is "which team has scored the most points during the match?" and the relation expressed is one team being better than the other, although in football the relation expressed could also be both teams tie. Examples of alternative level data would contain information such as how many points per match does each team score prior to this game and the mean height of the players in the team. Examples of decision level data would be asking whether the match was held close to any of the team's home towns. Table 2.11 shows examples of what this type of data might look like.

Table 2.11: Example of pairwise comparison observations

Decision level data

team 1	**team 2**	**team 1 won (1/0)**	**location**
Virginia	Purdue	1	Richmond, Virginia
Kentucky	Auburn	0	Dallas, Texas
Duke	MI State	0	New York, New York

Alternative level data

team	**mean points scored in season**
Virginia	80
Kentucky	75
Duke	67
Purdue	96
Auburn	62
MI State	85

2.3.2 Expressing decision, alternative and decision maker / process level data mathematically

In this section we describe the mathematical formalism of expressing observations such as the ones in the examples we have shown earlier.

Decision level data

- **The observed alternatives**: One of the core components in preference models is the set of alternatives. Let the set of all possible alternatives that could be presented in a decision be defined as $\mathcal{A}$, this can be all the distinct items that are sold by a retailer, or all the teams that play in a league. For a decision i the decision maker or deciding process might not always be exposed to the full set of alternatives, for example basketball matches are only played between two teams, and not all products sold by a retailer can be found in all stores. For example, we can see in table 2.9, the universe of products would be $\mathcal{A} = \{a, b, c, d\}$, however, none of the stores had all these products ranged, and the actual set from which customers could choose was in fact a subset of $\mathcal{A}$. Since for each observation there might be a subset presented of all the alternatives, we denote the alternatives presented in observation i as $A_i \subseteq \mathcal{A}$. Note that the reason this doesn't fall under the alternative level data, is because the set of alternatives presented for each decision can change from decision to decision.

- **The observed relations expressed over the alternatives**: First we fix some property p on relations which are expressed over the set of alternatives as in definition 1. Then relations may be expressed over A_i with property p and these observations will be denoted by $Y_i \in \mathcal{R}_p(A_i)$, specifically for subset choice it is $Y_i \in \mathcal{R}_{ch}(\mathcal{P}(A_i))$. Y_i is typically the variable of interest that researchers want to learn more about: usually what it is influenced by and how it can be predicted, it is often referred to as the *ground truth*. The domain of Y for some property p is defined by $\mathcal{Y} = \mathcal{R}_p(\mathcal{P}(\mathcal{A}))$. We will sometimes also refer to an expressed relation over a set of alternatives as a decision.

- **Other variables observed on the decision level**

  - We also define X_i which takes values in some domain $\mathcal{X}$ that is not further specified. Usually, $\mathcal{X} \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}$; and X_i are interpreted as observed covariates. Even though in our examples we have defined some of these covariates by their description, for example in table 2.10, we said the weather was "sunny" (which is not a member of the real numbers), in practice these are often converted to a numerical representation, for example by one-hot encoding (Harris and Harris,

2010) which a method used to express observations such as the weather conditions in a way that is in the reals.

- Optionally, there are further observations $B_1, \ldots, B_N$, that vary in $\mathcal{B} = \mathcal{R}_{p'}(\mathcal{P}(\mathcal{A}))$ for some property p' not necessarily equal to p. The B_i are interpreted as a relation specifying how the alternatives were presented, for example a sequence or hierarchy of presentation or sometimes used as home team advantage.
- There may be further information available that indicates which decision maker / process is making this specific decision. We will denote the list of all possible decision makers / processes by $\mathcal{S}$ and the decision maker / process of observation i by $S_i \subseteq \mathcal{S}$. Using that definition allows for multiple decision makers / processes making a joint decision, however, for the sake of not over-complicating this report, we will be expressing everything in a way that assumes $S_i \in \mathcal{S}$. We can now begin to describe an observation (i) as something made up of $(A_i, S_i, B_i, X_i, Y_i)$, however, this is information that pertains only to decision level data, we will now move on with describing alternative level and decision maker level data mathematically also.

Table 2.12 shows an example of what decision level data might look like for shopping data.

Table 2.12: Example of decision level data

A_i **items in store**	S_i **customer id**	X_i **temperature at purchase time (C)**	Y_i **items purchased**
{a, b, c}	1	22	{a, b, c}
{a, b, c}	2	15	{a, c}
{a, d}	3	24	{a}
{b, d}	3	10	{d}

Alternative level data

Further data that may be observed include alternative level observations. We will denote information available on the alternatives by $\mathcal{G} \subseteq \mathbb{R}^q$ for some $q \in \mathbb{N}$. These are for alternative level data what X is in the decision level data, and similarly in some of our examples they might not be shown as real numbers, but, for practical purposes they will be converted to them, we show an example of how to do this for whether an alternative has a chocolate flavour or not in table 2.13. Contrary to X, there will be a slight difference in notation for $G \in \mathcal{G}$. Whereas in X, we denote X_i as the decision level covariates for decision i, we should not do the same thing for G, because it does not vary on the decision level. Therefore, information about a specific alternative $a \in \mathcal{A}$ will be denoted by G_a, this can contain information like the weight or the price of a product. We will denote all the information available about the alternatives in a decision by $G_{A_i} = [G_a : \forall a \in A_i]$ so where $A_1 = \{a, b\}$ then $G_{A_1} = [G_a, G_b]$. In table 2.13 we show an example of alternative level data where there are products $\mathcal{A} = \{a, b, c, d\}$ that are described by their prices and whether they are a chocolate flavour or not. Now we can add to the definition of our observation (i) the alternative level variables as $(A_i, S_i, B_i, X_i, G_{A_i}, Y_i)$.

Table 2.13: Example of alternative level data.

Lookup to A_i	G	
Product	**flavour is chocolate**	**price (GBP)**
a	0	2
b	1	2.50
c	0	1.20
d	0	1

Decision maker / process level data

Similarly to the alternative level data, the decision maker/process level data is keyed by the distinct decision makers / processes. We will denote by $C \in \mathbb{R}^d$ for $d \in \mathbb{N}$ their covariates. C is indexed by the elements of $\mathcal{S}$ so for observation i we will express C_{S_i} as the covariates of the decision maker / process observed in decision i. Finally the whole data comes together for observation (i) as $(A_i, S_i, B_i, X_i, G_{A_i}, C_{S_i}, Y_i)$.

Definition 4. Preference data

We define a dataset of $n \in \mathbb{N}$ observations as $D = \{(A_i, S_i, B_i, X_i, G_{A_i}, C_{S_i})\}, (i = 1, ..., n)$ where the domain of D is defined by $\mathcal{D} = (\mathcal{P}(\mathcal{A}) \times \mathcal{S} \times \mathcal{B} \times \mathcal{X} \times \mathcal{G} \times \mathcal{C})$. Note

that we keep $Y \in \mathcal{R}_p$ separate in this definition. This becomes useful since in most cases we will be describing actions where we are mapping the data to the ground truth, and keeping the ground truth separate from the data in our definitions aids simplicity in notation in later sections.

■

In table 2.14 we bring the previous two tables together and show also decision maker/process level data we have annotated an example of shopping data to show what each element in this table would correspond to in our notation. Observation 4 in this table can be generally expressed by $A_4, S_4, X_4, G_D, G_B, C_3, Y_4$. We didn't use any structural data B in this example. Note that C_3 is not a mistake, but is due to the fact that the decision maker in observation 4 is the decision maker number 3 from the decision makers id table $S_4 = 3$.

Table 2.14: Example of shopping observations and how they relate to the notation.

Decision level data

A_i	S_i	X_i	Y_i
items in store	**customer id**	**temperature at purchase time (C)**	**items purchased**
{a, b, c}	1	22	{a, b, c}
{a, b, c}	2	15	{a, c}
{a, d}	3	24	{a}
{b, d}	3	10	{d}

Alternative level data

Lookup to A_i	G	
Product	**flavour is chocolate**	**price (GBP)**
a	0	2
b	1	2.50
c	0	1.20
d	0	1

Decision maker / process level data

Lookup to S_i	C
customer id	**age**
1	25
2	65
3	40

2.4 Preference datasets and dataset storage

In the earlier section we have described what components constitute an observation and preference data. Here we discuss the best practice for storing them computationally. We show some fictional and real examples of preference data to further clarify the concept behind the different types of relations that can be expressed and to show how all the different components that create a preference observation interact. The purpose of this section is to further solidify the reader's understanding in the different types of relations that can be expressed and to show examples of how to best store preference data.

2.4.1 Relational data formats

A Kaggle survey showed that "the majority of learning tasks faced by data scientists involve relational data" (Abo-Khamis et al., 2020). The most likely format in which preference models are stored is also in a relational database. This is because a relational database helps to maintain the data integrity, reduces data redundancy, and thereby makes it easy to implement security methods.

A popular way of evaluating whether a database is robust is through the ACID test (Haerder and Reuter, 1983), which are a set of principles that ensure that the data stored in an architecture is resistant to external issues such as power failures. Jatana et al. (2012) discuss that whilst relational databases pass this test, many other forms of data storage do not pass the ACID test which could compromise the integrity of the data. On the other hand, they also point out the drawbacks of relational databases, which is that they are less computationally efficient at handling large datasets than some of the alternative solutions.

The literature on relational databases is very extensive and the goal of this section is to only superficially discuss two possible data structures that might be used for storing preference data. We will contrast the pooled data structure vs. the relational database structure to argue that the relational database structure is superior for when it comes to storage efficiency and maintaining and updating records. This argument will be also a central theme to ideas posited in section 6.1.3.

We will refer to a pooled dataset structure as one where all the information is contained in one table only. Consider the following example of Table 2.15 football data.

This type of data storage is considered poor practice because it has information that is duplicated on the team budget level. We can see that the information for Southend Utd and

Table 2.15: pooled data

Team1	Team2	Matchday	Weather	Winner	Team1 budget	Team2 budget
Southend Utd	AFC Wimbledon	1	Rainy	Southend Utd	£6.46m	£3.89m
Bury	Oxford Utd	1	Sunny	Oxford Utd	£6.53m	£6.64m
Blackpool	Rochdale	1	Cloudy	Rochdale	£3.33m	£3.69m
Southend Utd	Bury	2	Snowy	Southend Utd	£6.46m	£6.53m

Bury's budgets are repeated. This in the literature is referred to as data redundancy, repeating groups or nonsimple domains (columns of a dataset are sometimes referred to as domains). Harrington (2016) outlines a variety of issues that such a data storage set up can cause:

- The insertion anomaly: it is impossible to store budget information about a team unless that team has a recorded match.
- The deletion anomaly: if the record of the encounter between Blackpool and Rochdale gets removed from this table the budget information on these two teams are lost.
- The update anomaly: if it turns out that the budget entered for Bury is erroneous, then the data maintainer will have to *correctly* update the number every time it appears, over two columns in this example, the second element of the Team 1 budget and the fourth element of the Team 2 budget. Having to do this over many places increases the chance of making a mistake in data storage, and at the very least increases the time it takes to update a table.

To deal with some of these problems the best practice today is considered to be a relational database, which was first published in the 70's by Codd, an IBM researcher and the first prototype, was released by Oracle in 1979 and it featured SQL, which is why many relational databases today are still set up as SQL tables (Lance Ashdown, 1993; Codd, 1970). One of the fundamental principles of it is to fix the above mentioned anomalies by a process Codd called normalisation. This technique rests on creating tables that contain unique identifiers for each row called primary keys and to push this key through from each parent table to a child table. The primary key of a parent table in a child table is sometimes referred to as a foreign key. A child table is one that can have repeating groups of the parent table, these can be represented graphically as a tree where the parent table is at the top. Normalisation has been described as follows.

For table 2.15 this would mean breaking the pooled table into a *team* table and a *result* table, where in the results table there would be only information which vary by the specific match (for example, *Weather* varies by the specific match) and in the team table all the information which is specific to the team (for example budget depends only on the team). As we can see this would split out the data based on the groups of information we called out earlier, in this case we have *decision level information* (Table 2.16) and *alternative level information* (Table 2.17). This not only solves the anomalies listed above, but it also occupies less space in computational storage as some column values are not duplicated as many times. For example, the budget for Bury is stored recorded once, not twice as it was in table 2.15.

Table 2.16: Decision level table

Team1	**Team2**	**Matchday**	**Weather**	**Winner**
Southend Utd	AFC Wimbledon	1	Rainy	Southend Utd
Bury	Oxford Utd	1	Sunny	Oxford Utd
Blackpool	Rochdale	1	Cloudy	Rochdale
Southend Utd	Bury	2	Snowy	Southend Utd

Table 2.17: Alternative level table

Team	**total_budget**
Southend Utd.	£6.46m
Bury	£6.53m
Blackpool	£3.33m
AFC Wimbledon	£3.89m
Oxford Utd.	£6.64m
Rochdale	£3.69m

Furthermore, it is possible in this set up to also store the information about which table is a parent table, child table and what is the field that links them to each other. For example, the alternative level table is the parent table of the decision level table and its primary key *Team* links to the columns *Team1* and *Team2* in the child table. The place where this information is usually stored is called a data dictionary.

2.5 Motivating datasets

Here we will describe the datasets that could be used for research experimentation in preference models. We will refer to some of these datasets when describing research questions.

2.5.1 Swissmetro dataset - discrete choice

The swissmetro dataset (Bierlaire et al., 2001; Antonini et al., 2007) tracks 470 respondents on which transportation alternative they have taken during the month of March 1998. There are 3 options in general: train, car and swissmetro.

2.5.2 dunnhumby The Complete Journey - subset selection

This dataset by dunnhumby; Venkatesan Raj (2020) contains 2500 frequent shoppers' purchase data, which is a subset choice dataset. The data contain all their purchases from many different categories. An important element is that transactions are recorded in long format, which means that each product purchased in a basket is represented by one row with a unique basket code and product code and rows are unique in product id and basket id, the set is just under 2.6 million rows long.

It is set up in a relational database structure, there are separate tables for promotional campaign data, coupons that have been sent out for the products and redeemed by customers or not, customer level information, product level information, and finally the transactional data. We provide more information about these in the appendix 9.4.1

2.5.3 Kaggle NCAA - pairwise comparison

Every March in the USA there is an elimination based basketball tournament played by the best performing college teams, which is also often referred to as March madness (NCAA, 2020). An example of this data can be found in the Kaggle (2019) National Collegiate Athletic Association (NCAA) men's competition. The purpose of the competition is to use data that is available before the March Madness tournament begins to predict the results of March Madness pairings. The data contains detailed results of the season and the tournament which contain the statistics accumulated in a match by each team, such as rebounds, field goals attempted, field goals made, blocks, steals etcetera. This data is available between 2003 and 2018. We also have access to the seeding of the teams, which are supposed to reflect the competence of them and the Massey Ordinals, which is a compilation of team ranking. More details about the NCAA tables can be found in appendix 9.4.2.

Chapter 3

Supervised preference models: assumptions, methods and estimation

Once the type of data we outlined in the previous chapter has been stored, it can be used to infer the process that generates the observed preferences, predict future preferences or predict the probability with which a certain preference might occur in the future. In this chapter we discuss supervised preference models, the theoretical assumptions that they're built upon and how they can make predictions of unseen preferences or begin to offer an explanation on what generates these preferences.

This chapter contains a definition of the modelling task that supervised preference models are trying to achieve. There can be three ways in which supervised learning preference models can be categorised: by the relations expressed in the tasks they're trying to solve, by the assumptions that guide the modelling method or by the mathematical approach of the method. We examine several modelling methods outlined in table 3.7 in the light of each of these three categorisations. We give an explanation on how supervised learning preference models are adjusted and estimated using observed data that we have outlined in the previous chapter and examples of how supervised preference models have been and can be applied in practice.

3.1 Supervised learning and the preference learning task

Supervised learning models belong to the family of machine learning models. Supervised learning problems have "a set of variables that might be denoted as *inputs*, which are measured or preset. These have influence on one or more *outputs* ... the goal is to use the inputs to predict the values of the outputs" (Hastie et al., 2009). Let's deconstruct these statements for preference models. In the simplest setting we may observe alternatives (A_i) and a type of relation (p) expressed over the alternatives where the expression is denoted by Y_i. The objective of supervised learning preference models is for a given type of relation (p), predict $Y_i \in \mathcal{R}_p(A_i)$ or $Y_i \in \mathcal{R}_p(\mathcal{P}(A_i))$ in the case of subset choices. In the case of preference models with no covariates this is achieved by creating a function $f : \mathcal{A}^k \to \mathcal{R}_p(\mathcal{A}^k)$ or $f : \mathcal{A}^k \to \mathcal{R}_p(\mathcal{P}(\mathcal{A}^k))$ in the case of subset choice. In this set up the **input** is A_i and the **output** is Y_i.

The inputs may be augmented with additional data that we have described in the previous chapter. Recall the different elements: decision maker / process identifier (S_i) and decision maker / process level information (C_{S_i}), information about the structure in which alternatives have been presented (e.g. home team advantage, captured by B_i), decision level variables (X_i), and alternative level variables (G_{A_i}). In our observation i of $(A_i, S_i, B_i, X_i, G_{A_i}, C_{S_i}, Y_i)$ the measured inputs are considered $(A_i, S_i, B_i, X_i, G_{A_i}, C_{S_i})$ which influence the output Y_i and these can also be used to predict Y_i. With these augmentations, the function in the previous paragraph can be expressed as $f : \mathcal{D} \to \mathcal{R}_p(\mathcal{A}^k)$ or $f : \mathcal{D} \to \mathcal{R}_p(\mathcal{P}(\mathcal{A}^k))$ in the case of subset choice, recall $\mathcal{D}$ from definition 4. In this set up the **input** is $(A_i, S_i, B_i, X_i, G_{A_i}, C_{S_i})$ and the **output** is Y_i.

Consider table 2.14, where the subset choice example table has been annotated with these symbols. Assume that we want to predict what items will be purchased by each customer (Y_i). We could employ a supervised learning model whose **outputs** would be for each customer a set of items that they are most likely to purchase. In general we denote these by Y_i, in the case of a subset preference model, this set of items would be always a subset of the available items in the store, i.e. the alternatives presented, which in turn would serve as one of the **inputs** into the model. The outputs of these models are often referred to as the ground truth, and are best described as *what* we're trying to predict.

We will use the definition of a supervised learning task as defined by Király et al. (2021). According to that, there are three components to a supervised learning task:

Definition 5. *Supervised Preference Learning Task*

- The data specification, which we have discussed at length thus far in the report, but the crucial part is that there are feature-label pairs. Features are the observations we defined as decision level observations $\mathcal{X}$, alternative level information $\mathcal{G}$, decision maker / process level information $\mathcal{C}$ and the set of alternatives offered for a choice $\mathcal{A}$. These are paired with a relation expressed with property p such that $Y \in \mathcal{R}_p(A)$ forming the following observations for

$$N \in \mathbb{N} : \{(A_1, S_1, B_1, X_1, G_{A_1}, C_{S_1}, Y_1), \ldots, (A_N, S_N, B_N, X_N, G_{A_N}, C_{S_N}, Y_N)\}$$

- We define **probabilistic learning** as estimating a function that can predict the result of the preferences $f : \mathcal{D} \to Distr(\mathcal{R}_p(\mathcal{A}^k))$ for some $k \in \mathbb{N}$ where $Distr(\mathcal{R}_p(\mathcal{A}^k))$ is a distribution of probabilities for each of the possible binary relation sets on $\mathcal{A}^k$.

  Non-probabilistic learning is to find a function that can accurately predict the result of the choices. $f : \mathcal{D} \to \mathcal{R}_p(\mathcal{A}^k)$ for some $k \in \mathbb{N}$. This function f "may depend on the values of the feature-label pairs ... but does not have access to [their distribution]" (Király et al., 2021).

- The final part of the supervised learning task is a definition of success which is a way of measuring how correctly the learning task has managed to use the observations (A, B, S, X, G, C) to predict Y. A function $f : \mathcal{D} \to \mathcal{R}_p(\mathcal{A}^k)$ is considered to be performing well if the expected generalisation loss $\mathbb{E}[L(g(A, B, S, X, G, C), Y))]$ is low for some loss function (L), specified by the researcher, in our case, $L : \mathcal{R}_p(\mathcal{A}^k) \times \mathcal{R}_p(\mathcal{A}^k) \to \mathbb{R}$ (Király et al., 2021).

■

There are two aspects to probabilistic learning. One is as discussed above when it comes to prediction, to capture the fact that we are always uncertain about what relation will be expressed so we express a probability to capture the uncertainty of our estimates of each potential outcome. The other aspect is that it might be the case that decision makers / processes themselves might not make the same decision under the same circumstances, so there's a natural volatility in the way the data is generated. Probabilistic learning aims to also capture this volatility. In section 3.2 we discuss behavioural hypotheses based on the probability of decision makers / processes making a certain decision.

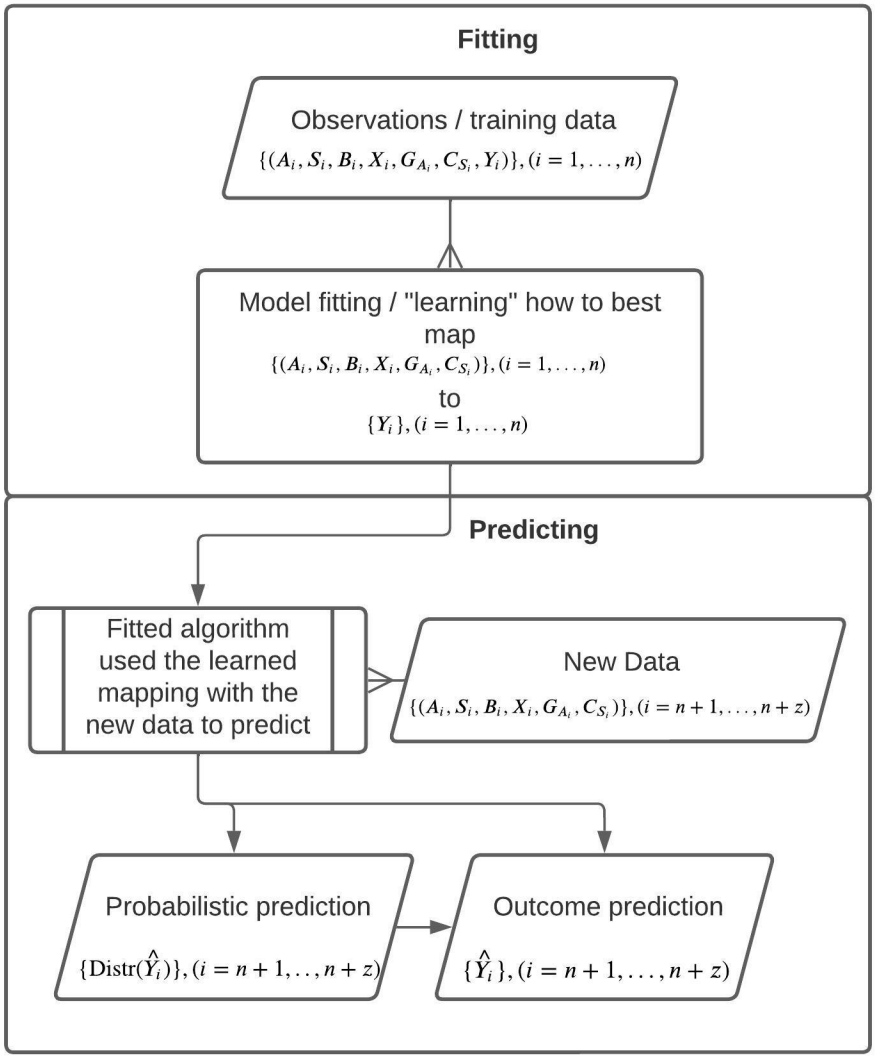
Supervised learning tasks are solved by supervised learning algorithms, also known as supervised learners. The way supervised learners map the inputs to the outputs is through the interaction of inputs with fixed real numbers called parameters. Parameters interact with

the input variables to control how much each influences the prediction of the output. The nature of the interaction between model inputs and parameters often depend on the model assumptions which will be discussed in the next section.

Parameters are estimated by the supervised learner based on examples of inputs and outputs that it is presented with, for example historical football matches where the teams playing and information about the teams playing (for example, how many goals they score in an average game) are the inputs and the result of the match are the outputs. Each supervised learner has an intrinsic loss function, which determines how well a specific set of parameters interact with the inputs map to the outputs. A supervised learner then adjusts the parameters such that it fits the outputs best according to the loss function. Note that this may or may not be the same loss function the researcher uses to evaluate success as in point three in definition 5. The process of finding the best parameters is often referred to as *fitting* or *training*. This will be discussed in more detail in section 3.4. Once these parameters are learned, they can be used to *predict* the outcome of previously unobserved decisions. To continue with our football example, suppose the supervised learner now figured out how to interact its parameters with the information provided about the teams playing, such that using these it can reconstruct the observed results with the least possible error according to its loss function. The model, which contains the rules on how to interact the parameters with the inputs, can now be used for predicting which team will win a match in the future by providing the information used about the teams in the fitting process but updated for future matches.

Figure 3.1 shows a flow diagram of modelling. How the data is consumed by the model fitting process and then given new observations the model applies the parameters learned in the fitting process to make predictions.

Figure 3.1: A simple flow diagram for a preference model task. In the upper part we have the fitting side of the process where the box on top denotes the data of $n \in \mathbb{N}$ observations that has been captured and is being fed into the supervised learning algorithm, which is a process that learns how to map the data it receives to the ground truth values (Y). On the predicting side, suppose that we now have k more observations, however, we do not have a recorded ground truth value for them. Since during the fitting, the supervised learning model has come up with a way to map the input observations to the ground truth, it uses this method to predict the ground truth values for the new input observations. The predictions can be probabilistic or a point estimate of the value the ground truth will take. We denote the predicted ground truth values by $\hat{Y}$.



We now begin to see that a supervised learner contains several components: parameters, a function that interacts parameters with the inputs to generate an output, a fitting method that calculates these parameters and a predicting method which applies the model to the inputs to generate the outputs. To capture all of this information, supervised learners have been recently described by Király et al. (2021) in a novel framework of "scientific typing" *scitype* for short which is "an abstract mathematical object based on the set of operations that we usually perform with them". A supervised learner can be described by scitype notation as:

Table 3.1: Scitype notation for describing supervised learners adapted from Király et al. (2021). We use notation from definition 4 for defining the data and table 2.6 for defining the properties of the relations we will also refer to k it will be a natural number $k \in \mathbb{N}$. Where the definitions for mathobject and paramobject as defined by Király et al. (2021) is "types of abstract representation of parameters and model objects respectively".

class type	$\mathrm{SupervisedPreferenceLearner}$		
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_p(\mathcal{A}^k) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_p(\mathcal{A}^k)$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_p(\mathcal{A}^k))$

3.1.1 Defining tasks by the relation expressed

The most common way to classify supervised learning tasks of preference models is by classifying them by the type of relation that it learns to map to. For example, the models that are used to learn pairwise comparisons are called pairwise comparison models. Similarly the literature models learn how to map inputs to discrete choices are called discrete choice models, and it is the same idea for subset choice models, partial rank and full rank models. The type of relation that is the output of each model is fixed. More formally, we show each of the types of models using scitype notation, where we can see that the difference between these model categorisations is the type of relation that is being mapped to. We will also use notation from definition 4 for defining the data and table 2.6 for defining the properties of the relations we will also refer to k it will be a natural number $k \in \mathbb{N}$. For saying that a scitype is a member of another general family of scitypes we use the following notation $\mathrm{SupervisedRankingModel} \leftarrow \mathrm{SupervisedPreferenceLearner}$, by which we mean that supervised ranking models are a type of supervised preference learners.

Table 3.2: Scitype notation for describing supervised ranking models

class type	SupervisedRankingModel		$\leftarrow$ SupervisedPreferenceLearner
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_{to}(\mathcal{A}^k) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_{to}(\mathcal{A}^k)$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_{to}(\mathcal{A}^k))$

Table 3.3: Scitype notation for describing supervise partial order models

class type	SupervisedPartialOrderModel		$\leftarrow$ SupervisedPreferenceLearner
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_{wo}(\mathcal{A}^k) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_{wo}(\mathcal{A}^k)$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_{wo}(\mathcal{A}^k))$

Table 3.4: Scitype notation for describing subset choice

class type	SupervisedSubsetChoiceModel		$\leftarrow$ SupervisedPreferenceLearner
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_{ch}(\mathcal{P}(\mathcal{A}^k)) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_{ch}(\mathcal{P}(\mathcal{A}^k))$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_{ch}(\mathcal{P}(\mathcal{A}^k)))$

Table 3.5: Scitype notation for describing supervised choice models

class type	SupervisedChoiceModel		$\leftarrow$ SupervisedPreferenceLearner
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_{ch}(\mathcal{A}^k) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_{ch}(\mathcal{A}^k)$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_{ch}(\mathcal{A}^k))$

Table 3.6: Scitype notation for describing supervised pairwise comparison models

class type	$\mathrm{SupervisedPairwiseComparisonModel}$		$\leftarrow \mathrm{SupervisedPreferenceLearner}$
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$\mathcal{D} \times \mathcal{R}_{wo \vee as}(\mathcal{A}^2) \rightarrow$ model
	predict	:	$\mathcal{D} \times$ model $\rightarrow \mathcal{R}_{wo \vee as}(\mathcal{A}^2)$
	predict probability (only probabilistic learners)	:	$\mathcal{D} \times$ model $\rightarrow Distr((R_{wo \vee as}(\mathcal{A}^2))$

3.2 Behavioural hypotheses in preference models

Learning tasks are accomplished via models, and each model is based on a series of behavioural hypotheses and assumptions. It is important to be familiar with the set of behavioural assumptions made by preference models so that we can understand the model's limitations and recognise situations in which certain models might not be applicable. Most of the behavioural hypotheses stem from economists who have researched discrete choice models. Therefore we will also illustrate these using discrete choice examples. Recall that in our formulation $\mathcal{A}$ is the set of all potential alternatives to choose from, $A_i \subseteq \mathcal{A}$ is the list of alternatives that has been available for making choice i and Y_i is the choice that has been made. We will express choices using the choice function from definition 3, $c(a, A_i)$, which returns the matrix equivalent to choosing alternative $a \in A_i$. For example, a can be a laptop and A_i are all the laptops in a specific store. The properties of all alternatives in A_i will be captured in the matrix G where we will denote the vector that represents properties of alternative a as G_a. For example, properties for laptops can be things such as price, RAM, storage capacity, etc. We will also make the assumption only for the purposes of explaining behavioural hypotheses that these properties are captured in such a way that a greater positive number is more desirable regarding that property, so when it comes to price it might be something like how good the price is.

As mentioned after our definition of a supervised learning task (definition 5), decision makers / processes themselves might not make the same decision under the same circumstances and different decision makers might make different decisions under the same circumstances, so there's a natural volatility in the way the data is generated. This means that for a decision maker and a specific decision, instead of a fixed outcome with 100% certainty there is a discrete probability distribution encompassing the likelihood of choosing each of the different options available. We will proceed to explore studies on how the probability of making

a choice of a specific alternative $P(Y_i = c(a, A_i))$ changes as A_i changes by the addition or removal of one alternative with specific properties in relation to a, for example what is the probability of purchasing a DELL laptop when there are no Macbooks in the store? And what is it when there are some Macbooks in the store? This in the literature is also known as context effects. The naming refers to how the context in which an alternative is presented influences the likelihood of that alternative being selected.

3.2.1 Independence from All Alternatives (IAA)

The first assumption we will discuss is the one that removes context effects altogether, by stating choices are made independently of all alternatives that are offered in the same decision. For example, under this assumption for a discrete choice the probability of buying a product is the same when there are 5 options to choose from or whether it's the only product in the store $P(Y_i = c(a, A_i)) = P(Y_j = c(a, A_j))\forall A_i$ and A_j. In these set-ups the probabilities of choosing the alternatives usually do not add up to one as discrete choice models would not be working under these assumptions. IAA predictions would be usually the output of $\#A_i$ binary models each predicting whether an alternative in A_i is chosen or not.

3.2.2 Independence from Irrelevant Alternatives

The most classical behavioural assumption for which many models have been developed is called the independence from irrelevant alternatives (IIA), dating back to Arrow (1951) which states that the odds between choices stay the same regardless of what alternatives are available (Ray, 1973).

Definition 6. IIA states that the ratio of probabilities between two alternatives does not depend on any other alternative Train (2009):

$$\frac{P(Y_i = c(a, A_i))}{P(Y_i = c(b, A_i))} = \frac{P(Y_j = c(a, A_j))}{P(Y_j = c(b, A_j))} \forall A_i, A_j \text{ s.t. } \{a, b\} \subseteq A_i \text{ and } \{a, b\} \subseteq A_j.$$

■

Note that models that follow independence from all alternatives also follow independence from irrelevant alternatives since $P(Y_i = c(a, A_i)) = P(Y_j = c(a, A_j))$ and $P(Y_i = c(b, A_i)) = P(Y_j = c(b, A_j))$.

There has been plenty of criticism of IIA, the most often cited one is by Debreu (1960) and summarised in Train (2009) in the blue bus red bus problem. This criticism sets up a commuter that is faced with two different choices. Assume that their first choice set is $A_1 = \{\texttt{car},\ \texttt{blue bus}\}$ and here the probability that they choose the car or the blue bus is 50% each $P(Y_1 = c(\texttt{car}, A_1)) = P(Y_1 = c(\texttt{blue bus}, A_1)) = \frac{1}{2}$. If IIA were to hold, the following ratio would be the same for all possible set of alternatives, in this case with A_1 it is $\frac{P(Y_1=c(\texttt{car},A_1))}{P(Y_1=c(\texttt{blue bus},A_1))} = 1$. Now suppose that for their second choice set a red bus is also introduced, which follows the same route and schedule as the blue bus $A_2 = \{\texttt{car},\ \texttt{blue bus},\ \texttt{red bus}\}$. It would make sense for the commuter to not care whether they take the red bus or blue bus so $P(Y_2 = c(\texttt{red bus}, A_2)) = P(Y_2 = c(\texttt{blue bus}, A_2))$, or $\frac{P(Y_2=c(\texttt{red bus},A_2))}{P(Y_2=c(\texttt{blue bus},A_2))} = 1$. However, according to IIA now the odds of choosing the blue bus or the car must also remain the same as in A_1 meaning that for A_2 it should be the case that $\frac{P(Y_2=c(\texttt{car},A_2))}{P(Y_2=c(\texttt{blue bus},A_2))} = 1$. The only solution for which both the odds described for A_2 are true is when $P(Y_2 = c(\texttt{red bus}, A_2)) = P(Y_2 = c(\texttt{blue bus}, A_2)) = P(Y_2 = c(\texttt{car}, A_2)) = \frac{1}{3}$. However, in real life one would expect that the probabilities would look more like this $P(Y_2 = c(\texttt{car}, A_2)) = \frac{1}{2}$, $P(Y_2 = c(\texttt{red bus}, A_2)) = P(Y_1 = c(\texttt{blue bus}, A_2)) = \frac{1}{4}$. That is, we would not expect that the commuters' preference of taking a car vs taking a bus changes because now a bus of a different colour serving the same route at the same times becomes available.

This highlights the importance of how alternatives are defined when using preference models that are based on the IIA assumption. The only way a researcher using models that follow IIA doesn't introduce biases in their task is by making sure that alternatives such as the blue bus and the red bus are defined as the same alternative. Thus researchers using these models need to be extra careful and spend plenty of time on defining the alternatives correctly. If in doubt, researchers can use statistical tests for the validity of the IIA assumption over a dataset. Some methods and an estimation for their power is described by Fry and Harris (1998).

3.2.3 Regularity

Definition 7. The regularity assumption states that

$$P(Y_i = c(a, A)) \geq P(Y_i = c(a, B)) \forall A \subseteq B.$$

■

Regularity states that any given alternative can have no larger probability of being chosen when there are more alternatives available. For example, an alternative a has to have at least as much or higher chance of being chosen from the set $\{a, b, c\}$ than from the set $\{a, b, c, d\}$. A statistical test for regularity is described by Gruca (1990).

3.2.4 Dependence on special alternatives

In the following sections we will present studies that have shown that these assumptions break down in practice with the introduction of some special alternatives. These alternatives are: substitutes (substitute effect), asymmetrically dominated alternatives (attraction effect) and compromise alternatives (compromise effect).

Substitute effect

One of the ways in which independence from irrelevant alternatives could break down is by the substitution effect. The substitution effect was first posited by Tversky (1972). It postulates that when a new alternative is presented in an existing group of alternatives it will decrease the probability of the choice of a similar alternative proportionally more than a different alternative. For example, if a store was selling Heinz Ketchup and Heinz Mustard and we introduced a new product, Hellman's Mustard, then this would impact the likelihood of Heinz Mustard being purchased much more than that of Heinz Ketchup, because presumably more people would switch from Heinz Mustard to Hellman's Mustard than from Heinz Ketchup to Hellman's Mustard. Such an outcome would violate the IIA assumption.

More generally consider two sets of alternatives $A = \{a, b\}$ and $B = \{a, b, c\}$ where alternative b is very similar to c but a is quite different from both, based on IIA, the following assumption would hold:

$$\begin{aligned} P(Y_i = c(c, \{a, c\})) &= \frac{P(Y_i = c(c, B))}{P(Y_i = c(c, B)) + P(Y_i = c(a, B))} \\ P(Y_i = c(b, \{a, b\})) &= \frac{P(Y_i = c(b, B))}{P(Y_i = c(b, B)) + P(Y_i = c(a, B))} \end{aligned} \tag{3.1}$$

In the published paper (Tversky, 1972), there is no proof of how we arrive from definition 6 to equation 3.1, nor are there any references to a proof, however this is an essential component of the substitution effect argument and understanding. So in this thesis there is a worked proof in appendix 9.1.1.

Tversky (1972) working on the substitution effect has found instances where it could be measured in the special case where alternative c is very similar to b we observe:

$$P(Y_i = c(c, \{a, c\})) > \frac{P(Y_i = c(c, B))}{P(Y_i = c(c, B)) + P(Y_i = c(a, B))}$$

$$P(Y_i = c(b, \{a, b\})) > \frac{P(Y_i = c(b, B))}{P(Y_i = c(b, B)) + P(Y_i = c(a, B))}$$

.

This suggests a break in the IIA implying that the probability of an alternative being chosen is proportionally greater when there isn't another alternative that is very similar to it

$$\frac{P(Y_i = c(c, \{a, c\}))}{P(Y_i = c(a, \{a, c\}))} > \frac{P(Y_i = c(c, B))}{P(Y_i = c(a, B))}$$

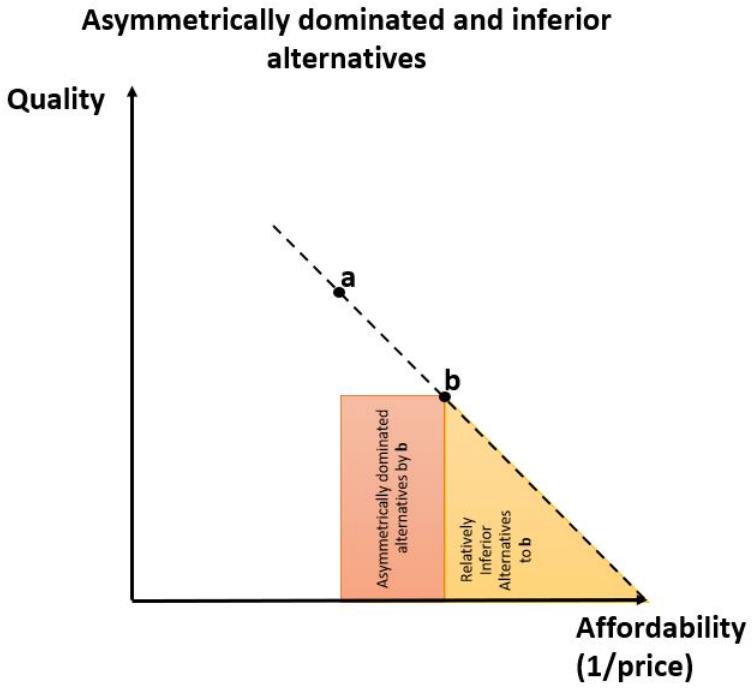
.

In other words, the introduction of a new alternative could affect disproportionately more similar alternatives than less similar alternatives and when researchers believe that such an effect might be taking place models that use IIA as an assumption might not be suitable tools.

Attraction effect

Another proposition in which IIA might break down is the attraction effect. The attraction effect considers alternatives that are asymmetrically dominated. An asymmetrically dominated alternative is inferior regarding all measurable aspects to one alternative in the decision set, but not inferior to all alternatives. Note that in this case what is deemed to be a "superior" attribute of an item needs some commonly agreed upon criteria by researchers, such as "lower price is a superior attribute to higher price" or "a larger pack is superior to a smaller pack". The attraction effect states that "adding an [asymmetrically dominated] alternative can increase the probability of choosing the item that dominates it" (Huber et al., 1982). For a graphical explanation please refer to figure 3.2.

Figure 3.2: This figure is based on the one shown by Huber et al. (1982). Imagine that there are two attributes by which alternatives a and b are captured. In this graph, these are "Quality" and "Affordability". We can see that alternative a is better in Quality than alternative b, which in turn is better in Affordability. Alternatives that would be asymmetrically dominated by alternative b would be ones that are in the red rectangle, where b is better in terms of both Quality and Affordability. The yellow shaded triangle shows an area where alternatives that would be called relatively inferior alternatives to b would be represented. Which is where the trade off between Quality and Affordability is better when switching from an alternative in the yellow triangle to b.



The attraction effect would predict the contrary to the substitution effect, such that when alternative b dominates alternative c in every aspect (for example if we have two aspects of affordability and quality then b is both more affordable and higher quality than c, that is c would be in the red rectangle in figure 3.2), however alternative a is not better than c in every aspect, for example a might be higher quality but less affordable than c then:

$$\frac{P(Y_i = c(b, \{a, b\}))}{P(Y_i = c(a, \{a, b\}))} < \frac{P(Y_i = c(b, B))}{P(Y_i = c(a, B))}$$

The paper goes farther than that, claiming that it might even happen that regularity is violated, $P(Y_i = c(b, \{a, b\})) < P(Y_i = c(b, B))$. This observation contradicts almost completely the substitution effect in the special case where an alternative is asymmetrically

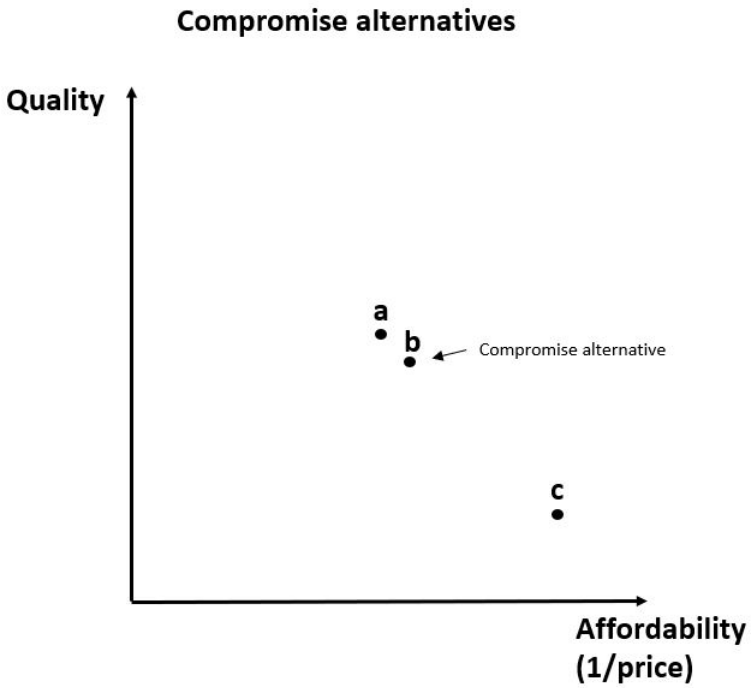
dominated, and the paper highlights that it is important to attempt to separate the confounding of the substitution and attraction effects (Huber et al., 1982).

It's not so easy to find examples of asymmetrically dominated alternatives in practice, but according to Huber et al. (1982) the effect can also be observed for what they call relatively inferior alternatives. That is alternatives that are not inferior to another alternative but have relatively worse trade-offs and are closer to the target alternative (the one that is supposed to gain from the attraction effect) in all variables. We can observe these products in real life all the time. At movie theatres the difference in size between a large and a medium popcorn is usually very large, whereas the difference in price is usually very small. The idea is that having the medium sized pop-corn option boosts sales of large popcorns by prompting people to trade up, whereas more people would opt for small sized popcorn if the medium size was not an option.

Compromise effect

Finally, the compromise effect would state that alternatives can become more attractive when they are "compromise alternatives in the choice set" regarding the attributes of the alternatives in the choice set: "If a decision maker is uncertain which of the two attributes is more important, a selection of a compromise alternative that can be seen as combining both attributes might be easiest to justify" (Simonson, 1989).

Figure 3.3: In this choice set where only two attributes are important, "Quality" and "Affordability", alternative b is the compromise alternative, which is defined by there being an alternative, in this case c, which is distant to the other alternatives but not inferior to any (as per the definition in figure 3.2). The alternative closest to the distant alternative, in this case b becomes the compromise alternative. This graph is based on the one presented in the Simonson (1989) paper.



The compromise effect suggests that an alternative that is too different in its features can actually be seen as less attractive e.g. an alternative that has by far the best price but also by far the worst quality or vice versa. For example if the initial set of alternatives was $\{a, b\}$ where a is slightly better quality than b at a slightly worse price, suppose that a new alternative c is being introduced such that c is far better priced than b (and therefore far better priced than a) but is also far lower quality than b (and therefore also far lower quality than a), then people might switch from a to b in a disproportionate way such that IIA breaks down in the following manner:

$$\frac{P(Y_i = c(b, \{a, b\}))}{P(Y_i = c(a, \{a, b\}))} < \frac{P(Y_i = c(b, \{a, b, c\}))}{P(Y_i = c(a, \{a, b, c\}))}$$

.

This might also be taken to the extent to violate regularity $P(Y_i = c(b, \{a, b\})) <$

$P(Y_i = c(b, \{a, b, c\}))$.

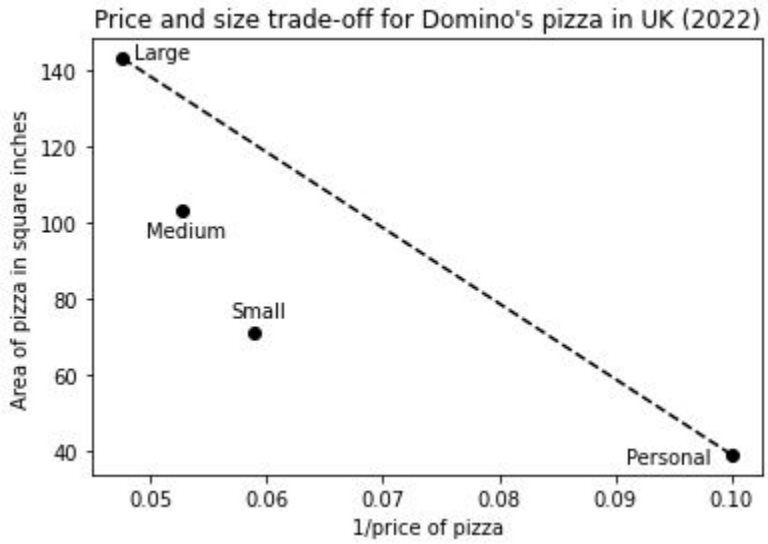
An example of the compromise effect could be observing people making a decision on buying a company car. For example, if given the choice of buying a Ford (£17k) or a Mercedes (£160k) might generate the odds of 2:1 in favour of Ford. A potentially observed compromise effect might be that if a Bugatti (£7.3m) was also introduced as a potential alternative then the odds between the Ford and the Mercedes might change to 1:1 indicating that people in this case might trade up from the Ford to the Mercedes seeing it as a compromise alternative between the Ford and the Bugatti.

Simonson and Tversky (1992) generalised the substitute, attraction and compromise effects in what they called contrast and extremeness aversion hypotheses.

An real life example of where the attraction compromise effects might be happening

A candidate for a real life example of the compromise and attraction effect could be Domino's pizza in London UK in 2022 (Dominos, 2022). Looking at figure 3.4 for the compromise effect we can see that the Personal size option is much farther away from the other options such that it could be making the Small size a compromise. We can also see that given the trade-off between the Large and the Personal pizza the Medium and Small sized pizzas are relatively inferior alternatives to Large which could be driving a behaviour of people trading up. For example, a shopper who would want to buy a medium sized would be clearly tempted to spend £2 to trade up to having much more, the difference in price between a large and a medium pizza is roughly 10% whereas the difference in size is roughly 40%.

Figure 3.4: This figure shows the trade-off between size and price for Domino's pizza UK based on prices in London UK in 2022 (Dominos, 2022). It could be argued that the Large, Medium and Small options are much closer together than the Small and Personal options, making the Small option a likely compromise alternative. We can also see that the Personal to Small and Personal to Medium options are worse trade-offs than the Personal to Large trade-off, making the Medium and Small options relatively inferior alternatives to the Large option.



The broader impact on society from these effects

The studies presented above for the substitution, attraction and compromise effects show how variations on alternatives offered have significant societal impact as they begin to quantify how much power there is in deciding the choice set for decision makers. Political scientists have long studied the power of agenda setting as pointed out by Schattschneider (1960) "the definition of alternatives is the supreme instrument of power". Understanding the behavioural effects of offering alternatives is important not just for political contexts but also how offering commercial alternatives impact members of the public. The abovementioned studies show that people's choices can be manipulated by defining the alternatives, in a way that goes beyond the straightforward exclusion of someone's "most preferred" alternative. Awareness from the general public of these effects could lead to decisions that are more influenced by the individual's independent consideration of what might be best for them.

3.2.5 Dependence on previous behaviour

Simonson and Tversky (1992) noted that "If a consumer habitually purchases the same brand category, ... context effects are unlikely to play a major role. In contrast when people are uncertain about the values of options they are more likely to use context in determining the best buy". The implication of this is that having chosen an alternative previously, should start dampening context effects, which is another way to refer to the substitute, attraction and compromise effects.

Riefer et al. (2017) have found through an extensive study on how people are likely to explore v.s. exploit different options, that having chosen something previously indeed increases the likelihood of choosing it again. This effect is called Consistency Maximising in psychology literature.

The above mentioned studies are showing examples of situations in which previous behaviour reinforces the same future behaviour, however one can also think of situations where previous behaviour deters the same behaviour. For example, if someone is going to the cinema every week, the fact that they have seen a movie the previous week would make the likelihood of seeing a different movie much higher.

To capture how previous behaviour impacts future behaviour, a researcher would need to have temporal identifier in their data to show the sequence of decisions made. As we are trying to keep the temporal element to a minimum in this thesis we will not dive much deeper into this branch, other than mention one model in a later section.

3.2.6 Dependence on other decision maker's behaviour

We will leave a detailed examination of this hypothesis as outside the scope for this document. This field of study starts from acknowledging that sometimes decision makers/processes can observe what other decision makers/processes have done in a similar situation. This observation can in turn influence a decision they are about to make. For example, someone's decision to drop out of school might be influenced by the decisions they have observed from others also doing the same thing (Brock and Durlauf, 2001) or trading algorithms in the market where one algorithm might change its behaviour having observed the behaviour of another algorithm or at least the outcome of the behaviour of that algorithm on the market (Borch, 2021).

To capture this effect a researcher would need additional data to what we've described in

this set up, which should be information about which decision maker / process has observed which other decision maker's / process' decision.

3.3 Methods in supervised learning preference models

In this section we will discuss supervised learning preference models. We will present them in relation to the mathematical modelling family to which they belong, the learning task they solve which looks at the type of relation expressed that needs to be learned and we will mention which behavioural hypothesis each model follows. Most of the research in this MPhil was focused on methods that belong to the family of generalised linear models, so these models will be explained in more detail than some of the others also presented in this section. Table 3.7 shows a summary of the models discussed in the subsequent sections and their classification according to the three taxonomies we found being used in the literature.

Table 3.7: Models based on mathematical family, task domain and behavioural hypothesis, we haven't worked out fully yet the mathematical family of some models, we coined those as "to be researched" for now.

Model	**Mathematical family**	**Modelling task domain**	**Behavioural hypothesis**
Classifiers	Generalised Linear Models	Classification	Independence from All Alternatives
Thurstone	Generalised Linear Models	Pairwise comparisons	Independence from Irrelevant Alternatives
General Logit Type	Generalised Linear Models	Discrete choice	Independence from Irrelevant Alternatives
Luce	Generalised Linear Models	Discrete choice	Independence from Irrelevant Alternatives
Zermelo-Bradley-Terry	Generalised Linear Models	Pairwise comparisons	Independence from Irrelevant Alternatives
Elimination by aspects model	To be researched	Discrete choice	Dependence on Special Alternatives
Coherency Driven Model	To be researched	Discrete choice	Dependence on previous behaviour
Plackett-Luce and variants	Generalised Linear Models	Full rank	Independence from Irrelevant alternatives
Support vector machines	Support Vector Machines	Pairwise comparisons	Not applicable
FATE / FETA	Neural Networks	Subset choice	Dependence on Special Alternatives
Bradley-Terry trees	Ensemble Learning	Pairwise comparisons	Independence from Irrelevant Alternatives
Nested Logit Models	Generalised Extreme Value Models	Discrete choice	Dependence on Special Alternatives

3.3.1 Generalised Linear Models

The concept of Generalised Linear Models (GLM) was popularised by Nelder and Wedderburn (1972) who realised that a modelling framework can be derived of which linear regression, logistic regression and Poisson regressions are special cases. Later this family of models was extensively explored in a book published by McCullagh and Nelder (1989). All the content in this subsection (3.3.1) has been reproduced from this book unless otherwise indicated.

GLMs have three components:

1. the observed data ($y \in \mathbb{R}^n$ where n is the number of observations) which is assumed to

be "the realisation of a random variable whose components are independently distributed with means μ" (McCullagh and Nelder, 1989).

2. a linear predictor ($T\beta$ where $\beta \in \mathbb{R}^d$ is a model parameter where d is the number of covariates (columns) in T, and $T \in \mathbb{R}^{n \times d}$ note that $T\beta \in \mathbb{R}^n$) that corresponds to y
3. a link function $g(\cdot)$ which creates the correspondence between y and $T\beta$ such that $E(y) = g^{-1}(T\beta)$.

A distribution belongs to the family of exponential distributions when its probability density function f_D can be written of the form $f_D(y;\theta,\phi) = \exp\left(\frac{y\theta - b(\theta)}{a(\phi)} + j(y,\phi)\right)$. For some functions $a(\cdot), b(\cdot), j(\cdot)$ and θ is known as the canonical parameter of the distribution when ϕ is known.

For example, for a normal distribution:

$$\begin{aligned} f_Y(y;\theta,\phi) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right) \\ &= \exp\left(\frac{y\mu - \frac{\mu^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2)\right)\right) \end{aligned} \tag{3.2}$$

where $\theta = \mu, \phi = \sigma^2, a(\phi) = \phi, b(\theta) = \frac{\theta^2}{2}, j(y,\phi) = -\frac{1}{2}\left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2)\right)$

In GLM we want to obtain the canonical parameters that maximises functions expressed like above, except it will be different functions for different models. In practice the log of these functions is used for finding the maximum, more on this will be in section 3.4. By solving for $E\left(\frac{d\log(f_Y)}{d\theta}\right) = 0$ we can obtain that $E(y) = b'(\theta)$. Therefore $b'(\theta)$ is the inverse of the link function, $b'(\theta) = g^{-1}(\cdot)$.

In cases when $y \in \mathbb{B}$, it is often represented by 0 and 1, where 0 corresponds to False ($\bot$) and 1 corresponds to True ($\top$). To ensure that a prediction of a GLM is restricted to the domain $[0,1]$ a cumulative probability density function is often used as $b'(\cdot)$. Common link functions used are the quantile function of the normal distribution and the logit function, also known as the probit and logistic regressions respectively. Note that which link function to use if often up to the discretion of the researcher and is often referred to as a model assumption.

Independence from all alternatives

The most basic preference GLM set up uses the independence from all alternatives assumption. Suppose there are two alternatives a and b in a pairwise comparison. In the most basic set up

we would have a model that says $P(Y_i = \{a \succ b\}) = g^{-1}(X_i\beta + G_a\gamma + C_{S_i}\kappa + \alpha)$. Where as mentioned in section 2.3 X, G and C are decision level, alternative level (note G_a are the alternative level variables for alternative a) and decision maker /process level covariates respectively and β, γ and κ are parameter vectors and α is a constant that might capture inherent advantages a would have that are not captured by the other parameters. When g^{-1} is the logit function, then this problem is formulated as a simple logistic regression where alternative a either gets chosen or not.

Such a formulation would follow the independence from all alternatives assumption. It can be seen that there is nothing in the equation that captures information about the other alternatives that a is being compared to, meaning that absence or presence of other alternatives does not impact the probability of a being chosen. To further illustrate, suppose that now we now have the comparison of a and c the formula still remains $P(Y_i = \{a \succ c\}) = g^{-1}(X_i\beta + G_a\gamma + C_{S_i}\kappa + \alpha) = P(Y_i = \{a \succ b\})$.

Independence from irrelevant alternatives

The simplest way to extend GLM that follow independence from all alternatives to ones that follow independence from irrelevant alternatives and regularity is through the assumption that each alternative in $\mathcal{A}$ has an unobserved strength / attractiveness to it which can be parameterised. Assume this parameter for each alternative $a \in \mathcal{A}$ to be $\lambda_a \in \mathbb{R}$.

Generalised linear choice models learn the vector of $\lambda \in \mathbb{R}^{\#\mathcal{A}}$ which assumes that every alternative's strength parameter is a position on the one dimensional real line, meaning that all these models assume that the alternatives can be represented in a fully ordered set from strongest to weakest.

This leads to the idea that the global preference of these alternatives are transitive. If these models were made to predict the alternative chosen in a decision (as a point estimate not a probabilistic estimate) they would assume that decision makers will always choose the alternative with the strongest parameter. Therefore, in such a discrete choice set up, the only way a decision maker would change their choice stemming from an increase in the alternatives available is if a globally stronger alternative is presented. By presenting additional globally inferior alternatives, if these models were made to predict a single alternative to be chosen, the decision maker would not be predicted to change their decision. This is a property of IIA and often used as an intuitive explanation of the concept.

The simplest GLM that follows IIA is a pairwise comparison assuming that $P(Y_i = \{a \succ$

$b\}) = g^{-1}(\lambda_a - \lambda_b)$. We can see here that now the probability of choosing any alternative depends on the strength parameter of the other alternative also.

These can be further augmented with a constant and covariates, mixing in elements from the classical GLM $P(Y_i = \{a \succ b\}) = g^{-1}(\lambda_a - \lambda_b + X_i\beta + G_{A_i}\gamma + C_{S_i}\kappa + \alpha)$ where α is a constant term often associated with a having some inherent advantage over and above it's skill parameter, such as the home team advantage in sports (Tutz, 1986; Fahrmeir and Tutz, 1994).

Thurstone Models Thurstone (1927c) proposed the first model that the literature widely recognises as the first pairwise comparison model ($A_i \in \mathcal{A}^2$) where in the above set-up he defined g^{-1} as the CDF of the normal distribution. The model is of the form $f : A \rightarrow Distr(Y)$.

The model assumes that for observation i the perceived attractiveness of alternative a is described by the random variable $\theta_{i,a}$ generated from a Normal distribution $\theta_{i,a} \sim \mathcal{N}(\lambda_a, \sigma_a)$ where $a \in \mathcal{A}$. The model is derived based on the assumption that when $\theta_{i,b} < \theta_{i,a}$ alternative a will be indicated as preferred by observer S_i. Therefore the probability that alternative a is preferred to alternative b can be expressed by $P(Y_i = \{a \succ b\})) = P(\theta_{i,b} < \theta_{i,a})$ where $\{b, a\} = A_i$.

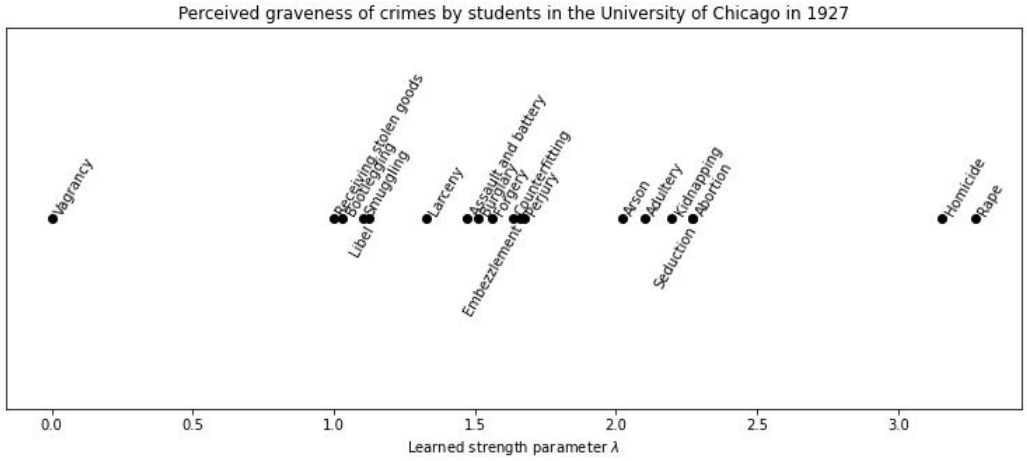
When reinterpreting this probability as a new random variable $\theta_{i,b} - \theta_{i,a}$ with mean $\lambda_b - \lambda_a$ and variance $\sigma_b^2 + \sigma_a^2 - 2COV(\theta_{i,b}, \theta_{i,a})$, then $P(\theta_b - \theta_a < 0)$ yields the general form of the Thurstone model $P(Y_i = \{a \succ b\}) = \Phi\left(\frac{\lambda_a - \lambda_b}{\sqrt{\sigma_b^2 + \sigma_a^2 - 2COV(\theta_{i,b}, \theta_{i,a})}}\right)$ where $\Phi()$ is the CDF of the Normal distribution.

Assume that in an experiment people are asked which product they preferred. Rather than substitutability and complementarity correlation between $\theta_{i,b}$ and $\theta_{i,a}$ reveals correlation between tastes. A positive correlation between $\theta_{i,b}$ and $\theta_{i,a}$ would indicate that it can be observed that as people perceive product b to be more attractive they would also perceive product a to be more attractive, this could be because they are similar products, for example people who like Coca-Cola might also like Pepsi or because they are complementary, e.g. people who choose to buy burgers might also choose to buy burger buns. Both these instances have a positive correlation between $\theta_{i,b}$ and $\theta_{i,a}$ and it could either mean the item is substitutable or complementary. A negative correlation would suggest the opposite, for example it could be hypothesised that some vegan and meat products could have such a relationship, for example people who like tofu burgers might not like beef burgers.

The simplest case of the Thurstone model also known as case V assumes the denominator is 1 and can be presented as Probit regression with just the two variables of the unobserved strength parameters $E(Y_i) = \Phi(\lambda_a - \lambda_b)$ in accordance with the definition of a GLM. Another also commonly used variant is the case III which assumes that the random variables are independent of each other, thus $COV(\theta_b, \theta_a) = 0$ which would remove the ability to account for correlations between tastes like mentioned in the previous paragraph (Thurstone, 1927a; Takane, 1981; Maystre, 2018). Cases II and IV are not used very often and the literature on them is scarce. Case II assumes that the decision is made by a group and case IV implements constraints on σ_a and σ_b.

With this type of model we can get estimates for the whole vector of parameters λ and produce a global ranking of the alternatives in $\mathcal{A}$, in fact, Thurstone (1927b) used this in an experiment to determine how students perceived the graveness of several offences. He then plotted this on a scale he called the "psychological continuum".

Figure 3.5: Research by Thurstone (1927b) similar to how it is graphically presented by Maystre (2018) showing the perceived seriousness of a crime by plotting the learned parameters λ. Note that this research is based on the law nearly 100 years ago and some things on here are no longer considered a crime, which in itself makes this an interesting study about how societal norms evolve over time. We expand on the applications of preference models further in section 3.5



Random Utility and Logit type models Whilst Thurstone was a scholar of the field of psychology, the field of choice models has been most evolved by econometricians. Economists approach the topic of preference and choice from the slightly different perspective of utility maximisation.

> "In Victorian days, philosophers and economists talked blithely about 'utility' as an indicator of a person's overall well-being. Utility was thought of as a numeric measure of a person's happiness. Given this idea, it was natural to think of consumers making choices so as to maximize their utility, that is, to make themselves as happy as possible" (Varian, 2010).

Since much of the preference and choice model literature is coming from the economics side it is often the case that it refers to decision makers solely as people not considering cases where it could also be decision processes such as a football game determining the outcome. This does not mean that models that belong to the logit-type branch of the GLM family cannot be used in cases where there are decision processes rather than decision makers, it just so happens that most contributors to the field were mostly concerned about choices that people make.

This probabilistic model is derived based on the utility maximisation principle, the probability of selecting one alternative in the pairwise comparison case changes to $P(U_t > U_s)$ where $t, s \in A_i$ and U_k is the utility of the observer for alternative k. The utility of an observer for choice i and product k is expressed as $U_{i,k} = \lambda_k + X_i\beta + G_k\gamma + C_{S_i}\kappa$ where λ_k as previously is the latent strength of preference for option k, it is also described as the mean impact of factors that affect utility but are not included by the other components (Train, 2009). Components that might affect utility but are not included in the formula are referred to as unobserved components / portions of utility in the literature and are often denoted by ϵ. So in economics literature the utility above would be often written as $U_{i,k} = \lambda_k + X_i\beta + G_k\gamma + C_{S_i}\kappa + \epsilon_{i,k}$, to represent the fact that there might be other components not captured by the rest of the equation that can impact a person's utility. The economics side of the literature often uses distributional assumptions on this unobserved (sometimes called error) term to derive models. A detailed exposition of the econometric models is presented by Train (2009).

In the Thurstone model θ could be interpreted as the utility, since this is a random variable. Marschak (1960) has coined the term Random Utility Models in the field of economics, so often relevant models in this field would be referred to under that term. A detailed history of how discrete choice models evolved in the field of economics has been given by McFadden (2001) in his Nobel Prize lecture.

The general form of the logit type models has been developed in the domain of discrete choices which means that the alternatives considered are $A_i \in \mathcal{A}^{\geq 3}$ and $Y_i \in \mathcal{R}_{ch}(A_i)$. The

most general form that we have come across during our research so far has been captured by Brathwaite and Walker (2018) who proposed the following model class and called them logit-type models for $d \in \mathbb{N}$ a function $f : \mathbb{R}^d \to \mathbb{R}$:

$$P(Y_i = c(t, A_i)) = \frac{\exp[\lambda_t + f(X_i\beta, G_t\gamma, C_{S_i}\kappa, z_t)]}{\sum_{\{j;\mathcal{A}_j \in A_i\}} \exp[\lambda_j + f(X_i\beta, G_j\gamma, C_{S_i}\kappa, z_j)]} \tag{3.3}$$

where $t \in A_i$ and z is a column vector of shape parameters mainly to enable changing the symmetry property of the logistic equation, for example by creating a skewed logistic equation, also known as a scobit model (Nagler, 1994). Most formulations of these probabilities have symmetric probability distributions around 0.5 and the primary area of research for Brathwaite and Walker (2018) was to create asymmetric models, but in doing so they ended up proposing a general case for the models we will discuss subsequently.

The expression shown in this probability is also known as the softmax function.

Definition 8. For some $k \in \mathbb{N}$, $softmax : \mathbb{R}^k \to \mathbb{R}^k$

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \text{ for } i = 1, ..., k \text{ and } z \in \mathbb{R}^k$$

■

McCullagh and Nelder (1989) have shown that models that use the softmax function to determine the probability of an outcome, such as the one in this case, are part of the GLM family.

The economics branch of the literature and the statistics Generalised Linear Models branch of the literature present how they arrive to the logit model slightly differently. The GLM branch following the formulation of Nelder and Wedderburn (1972) would say that the model assumption would be that the link function is the logistic function. Economists such as Train (2009), present this as an assumption on the distribution of the unobserved variables (ϵ), which they say is a Gumbel distribution and from there it is also possible to derive that the probability formulation itself then should follow the logistic formulation.

With the Logit-type formulation we can show that these always follow IIA, the proof is short enough to include it in this section, and can be also found in Train (2009).

Lemma 3.3.1. *Given definition 6 models formulated like equation 3.3 have IIA as a property.*

Proof. In this lemma we show that IIA is a property of the logit-type models. We will simplify

some notation, let $V_{i,t} = \lambda_t + f(X_i\beta, G_t\gamma, C_{S_i}\kappa, z_t)$ and $P_{i,t} = P(Y_i = c(t, A_i))$. For alternatives $a, b \in A_i$:

$$\frac{P_{i,a}}{P_{i,b}} = \frac{\frac{e^{V_{i,a}}}{\sum_j e^{V_{i,j}}}}{\frac{e^{V_{i,b}}}{\sum_j e^{V_{i,j}}}}$$
$$= \frac{e^{V_{i,a}}}{e^{V_{i,b}}} = e^{V_{i,a} - V_{i,b}}$$

. This only depends on the observables in a and b, therefore it does not matter how large A_i becomes, this ratio will always stay the same. ■

In the following sections we will present some special cases of Logit-Type models known as Luce, Zermelo-Bradley-Terry models.

The Luce model in its most classical form doesn't consider covariates only the parameters λ (Luce, 1959).

$$P(Y_i = c(t, A_i)) = \frac{e^{\lambda_t}}{\sum_{\{j \in A_i\}} e^{\lambda_j}} = softmax(\lambda)_t.$$

The augmentation with covariates would be

$$P(Y_i = c(t, A_i)) = \frac{e^{\lambda_t + G_t\gamma^T}}{\sum_{j \in A_i} e^{\lambda_j + G_j\gamma^T}} = softmax(\lambda + G\gamma^T)_t.$$

This is an equivalent formulation to the Multinomial Logit model, with the added latent strength features for each alternative λ. Note that $X_i\beta^T$ is not included here, this is due to the fact that the majority of the literature that has been so far digested in this MPhil has concerned itself with using only the alternative level variables (in our notation denoted by G) as inputs to make predictions for preferences, ignoring decision level variables (X) and many with the exception of the approach outlined by Strobl et al. (2011) in section 3.3.7 do not talk about decision maker level variables (denoted by C). This could omit significant explanatory variables in models. For example, when two teams play a match on a rainy day, it rains equally for both teams, but one team could play better in the rain than another. When people walk into a store on a cold day it's a cold day for all products being considered, but a cold day could increase the likelihood for purchasing soup more than purchasing ice cream. Rain could impact someone's decision to cycle to work differently than their decision to drive to work. Practitioners today should focus on including these variables also, not only variables that describe the alternatives.

The reason why an explicit discussion about this is necessary is because the way alter-

native level and non-alternative level covariates (decision level and decision maker / process level) need to be used in generalised linear choice models is different. Say the decision level variables are $X \in \mathbb{R}^{n\times k}$ for some $n, k \in \mathbb{N}$ where n is the number of observations and k is the number of covariates. It is possible that someone familiar with how covariates work in classical GLM described in section 3.3.1 might be primed to include decision level variables to interact with model parameters $\beta \in \mathbb{R}^k$. However with that specification $softmax(\lambda + G\gamma^T)_t = softmax(\lambda + G\gamma^T + X_i\beta^T)_t$.

Lemma 3.3.2. *If $X \in \mathbb{R}^{n\times k}$ and $\beta \in \mathbb{R}^k$ then $softmax(\lambda + G\gamma)_t = softmax(\lambda + G\gamma + X_i\beta)_t$.*

Proof.

$$\begin{aligned} softmax(\lambda + G\gamma + X_i\beta^T))_t &= \frac{e^{\lambda_t + G_t\gamma + X_i\beta}}{\sum_{j\in A_i} e^{\lambda_j + G_j\gamma + X_i\beta}} \\ &= \frac{e^{X_i\beta} e^{\lambda_t + G_t\gamma}}{e^{X_i\beta}\sum_{j\in A_i} e^{\lambda_j + G_j\gamma}} \\ &= \frac{e^{\lambda_t + G_t\gamma}}{\sum_{j\in A_i} e^{\lambda_j + G_j\gamma}} = softmax(\lambda + G\gamma) \end{aligned}$$

■

So, covariates that vary on the decision level and not on the alternative level make no impact in the probability of these decisions when included in a way that would perhaps feel most intuitive at first sight, such that there is one parameter for each covariate. Yet there is a large use for these variables, as mentioned in some of the examples, when choosing whether to buy for lunch a sandwich or a soup might be strongly influenced by the weather, a colder day might increase the propensity of buying soup. So how can these be captured in logit-type models? The answer is to learn separate coefficients for each alternative. Statistically, this is creating interaction variables between the coefficients and a one-hot encoded feature for each alternative. Learning a single parameter vector β here would suggest that a sunny day would impact the probability of buying a sandwich by the same amount as buying a soup, which is not the intention behind the example. Therefore β needs to be redefined as $\boldsymbol{\beta} \in \mathbb{R}^{m\times k}$ where m is the number of alternatives and k is the number of covariates, that is, the width of each X_i. This way the matrix $\boldsymbol{\beta}$ can capture the fact that a cold rainy day impacts the purchasing of soup in a different way than the purchasing of ice-cream. Now β_j will refer to the parameter vector $\boldsymbol{\beta}$ for alternative j. So to include these parameters the equation

becomes

$$P(Y_i = c(t, A_i)) = \frac{e^{\lambda_t + G_t\gamma + X_i\beta_t^T}}{\sum_{j \in A_i} e^{\lambda_j + G_j\gamma + X_i\beta_j^T}} = softmax(\lambda + G\gamma + X_i\boldsymbol{\beta}^T)_t$$

.

Similarly if decision maker / process data is being used with covariates that don't vary across alternatives, we arrive to the same effect as in lemma 3.3.2. So for these models the covariates also need to be defined on a decision maker / process level and alternative level, that is when including the variables $C_{S_i} \in \mathbb{R}^{s \times h}$ (s unique decision makers / processes with h descriptive variables each), the covariates need to be defined as $\kappa \in \mathbb{R}^{m \times h}$ where m is the number of distinct alternatives. So including all terms in the Luce model would become:

$$P(Y_i = c(t, A_i)) = \frac{e^{\lambda_t + G_t\gamma + X_i\beta_t^T + C_{S_i}\kappa_t^T}}{\sum_{j \in A_i} e^{\lambda_j + G_j\gamma^T + X_i\beta_j^T + C_h\kappa_j^T}} = softmax(\lambda + G\gamma^T + X_i\boldsymbol{\beta}^T + C_{S_i}\kappa^T)_t$$

The additional complexity that comes with having to vary the decision level and the decision maker / process level parameters for each alternative could be an explanation for why many publications would only use alternative level information in their formulation. However, when more than alternative level variables are being used the correct set up is as described above. An example where this set up is clearly defined can be found in Gillen et al. (2015). A longer discussion about implementing different types of parameters can be found in Schauberger and Tutz (2019).

We can prove that the Luce and by extension the Logit-type formulation always gives transitive preferences.

Lemma 3.3.3. *The Luce formulation with covariates produces transitive preferences.*

Proof. In this formulation $a \succ b$ if $P(Y_i = c(a, A_i)) > P(Y_i = c(b, A_i))$ which means $P(Y_i = c(a, A_i)) - P(Y_i = c(b, A_i)) > 0$. To prove the lemma we need to show that if

1. $P(Y_i = c(a, A_i)) - P(Y_i = c(b, A_i)) > 0$ and
2. $P(Y_i = c(b, A_i)) - P(Y_i = c(c, A_i)) > 0$ then
3. $P(Y_i = c(a, A_i)) - P(Y_i = c(c, A_i)) > 0$.

We can prove this using the concept that when $x, y \in \mathbb{R}_{>0}$ then $x + y > 0$. We proceed by adding the first two equations together yielding

$$\begin{aligned} &\text{equation } 1 + \text{ equation } 2 > 0 \\ P(Y_i = c(a, A_i)) - P(Y_i = c(b, A_i)) + P(Y_i = c(b, A_i)) - P(Y_i = c(c, A_i)) &> 0 \\ P(Y_i = c(a, A_i)) - P(Y_i = c(c, A_i)) &> 0 \end{aligned}$$

■

The Zermelo-Bradley-Terry model has been invented several times independently by researchers. Most people know it just as the Bradley-Terry model. It is the pairwise comparison version ($A_i \in \mathcal{A}^2$) of the Luce model. For $t, s \in A_i$

$$P(Y_i = \{t \succ s\}) = \frac{e^{\lambda_t}}{\sum_{j \in A_i} e^{\lambda_j}} = \frac{e^{\lambda_t}}{e^{\lambda_s} + e^{\lambda_t}} = \frac{1}{1 + e^{\lambda_s - \lambda_t}} = \frac{e^{\lambda_t - \lambda_s}}{1 + e^{\lambda_t - \lambda_s}} = g(\lambda_t - \lambda_s)$$

where $g()$ is the sigmoid function.

Zermelo (1929) proposed this model to rank chess players and then Bradley and Terry (1952) did the same thing but as an intended alternative to the Thurstone model. Luce later generalised Bradley and Terry's formulation to several items in the choice set. It has been also noted that the Bradley Terry model bears a relationship to the Élő (1978) model, which derives a strength parameter for each player for games played by two players or two teams (original domain is chess) and updates differently based on the skill difference between players. For example, a player's rating increases less when winning against a lower ranked player than against a higher ranked player. The relationship between the Bradley Terry and Élő model is such that the Élő model can be thought of as an update step in the Bradley Terry model (Coulom, 2007; Király and Qian, 2017). These models can be used in any Pairwise Comparison setting that doesn't allow for ties. Basketball data or karate matches are a good application of this model.

Interactions between different alternatives in pairwise comparisons can be often represented as a graph, where the alternatives are the nodes and a comparison between the alternatives would be an edge. In figure 3.6 we can see an example of a fully connected graph

and in figure 3.7 we can see an example when a graph is not fully connected.

Figure 3.6: An example of a fully connected graph. Here imagine that we are representing the following information: in our data at least once, a has been compared to b, b has been compared to c and d (for Bradley-Terry models this would be on two separate observations), and c has been compared to e. Note that in a fully connected graph not all nodes need to be connected to each other, so even though a has never been directly compared to c or e, this is still a fully connected graph, because from any node in the graph we can get to any other node via the edges.

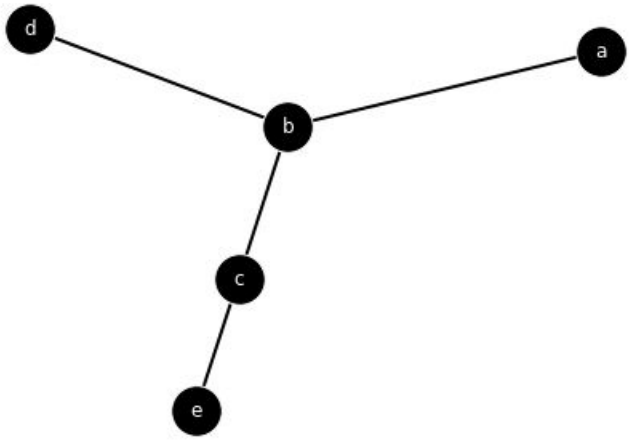


Figure 3.7: An example of a graph that is not fully connected. This a graph of the same comparisons as above, with the difference of one comparison never happening, in this example b has never been compared to c. Here we can see that there is no way of getting to every node starting from any node. In such cases Bradley-Terry would not yield reliable estimates for the strength parameters of these alternatives.

For the Zermelo-Bradley-Terry model to work there needs to be a fully connected graph between all the alternatives, fully connected meaning that from any alternative there should be a way to get to any other alternative by following the edges and the nodes (Ford, 1957). If the graph is not fully connected that means that there are clusters of alternatives both having no elements that have never been compared to each other, which would indicate that the set of alternatives is poorly defined.

The Bradley-Terry model is well studied to cater for sports prediction problems. In one variant it is possible to add a home team advantage or topic preference to the Bradley Terry model by adding in a constant in the equation α such that for $t, s \in A_i$, $P(Y_i = \{t \succ s\}) = g(\alpha + \lambda_t - \lambda_s)$(Tutz, 1986; Fahrmeir and Tutz, 1994). Note that this only works for a specific structural assumption in the data that the alternative presented on the left hand side of the preference inequality is always the home team.

It is also sometimes augmented with covariates $P(Y_i = g(\alpha + \lambda_t - \lambda_s + (G_t - G_s)\gamma^T + X_i(\beta_t - \beta_s)^T + C_{S_i}(\kappa_t - \kappa_s)^T)$.

Rao and Kupper (1967) have created a version that allows for ties which helps with sports where ties are allowed such as football or in a simple choice model where someone cannot decide which alternative is better. They proposed that there should be a parameter η that generates a tie when $|\lambda_t - \lambda_s| < \eta$ and incorporated it into the model the following way:

$$P(Y_i = \{t \succ s\}) = \frac{e^{\lambda_t}}{e^{\lambda_t} + e^{\lambda_s + \eta}}$$
$$P(Y_i = \{s \succ t\}) = \frac{e^{\lambda_s}}{e^{\lambda_t + \eta} + e^{\lambda_s}}$$
$$P(Y_i = \{t \sim s\}) = \frac{(e^{\lambda_s + \lambda_t})(e^{2\eta} - 1)}{e^{\lambda_s + \eta} + e^{\lambda_t + \eta}}$$

. When η is 0 then we have the Bradley-Terry model, so this is a generalisation of the Bradley-Terry model.

Incorporating temporal elements into preference models is out of scope for this report. However for more information we recommend starting with Cattelan et al. (2013) who have implemented a dynamic version of the Bradley Terry model also where they "model the evolution in time of the abilities in home and away matches of each team through an exponentially weighted moving average process". There might be further research opportunities into models like this, since there might be some opportunities to incorporate more advanced temporal methods other than ARIMA into these models.

The Plackett-Luce model is used for solving fully ranked data, that is $Y_i \in \mathcal{R}_{to}$. Plackett (1975) has studied the permutations of rankings in horse races in which he has proposed a model where the probability of a permutation is

$$P(Y_i = \{\mathcal{A}_1 \succ \mathcal{A}_2 \succ \ldots \succ \mathcal{A}_m\}) = \prod_{r=1;\mathcal{A}_r \in A_i}^{m} \frac{e^{\lambda_r}}{\sum_{j=r;\mathcal{A}_j \in A_i}^{m} e^{\lambda_j}}$$

.

"This can be seen as m-1 independent choices made using Luce's model, iteratively, over the remaining alternatives" (Maystre, 2018).

The Plackett-Luce model can be also parameterised as:

$$P(Y_i = \{\mathcal{A}_1 \succ \mathcal{A}_2 \succ \ldots \succ \mathcal{A}_m\}) = \prod_{r=1;\mathcal{A}_r \in A_i}^{m} \frac{e^{\lambda_r + G_r\gamma^T + X_i\beta_r^T + C_{S_i}\kappa_r^T}}{\sum_{j=r;\mathcal{A}_j \in A_i}^{m} e^{\lambda_j + G_j\gamma^T + X_i\beta_j^T + C_{S_i}\kappa_j^T}}.$$

Most of the fully ranked models have been developed by scholars who are identified as members of the learning to rank field. Fully ranked models, which in this thesis are referred to as models that use a fully ranked set of alternatives to learn how to predict future rankings are referred to as *listwise* approaches in the learning to rank literature. Other versions of listwise approaches including SVM, MAP, PermuRank, AdaRank, ListMLE, SoftRank, AppRank can be found in Liu (2011).

Dependence on previous behaviour

The Coherency Driven model (Hornsby and Love, 2020) accounts for shifting preferences of certain attributes of products. It departs from the observations in $G_a \in \mathbb{R}^m$ for m number of attributes. Each decision maker S_i has a preference p over each attribute of an alternative so that $p \in \mathbb{R}^{d \times m}$ where d is the number of unique decision makers / processes. For example if the first element in G_a is the price of the item, then $p_{1,1}$ is the preferred price of decision maker 1 and each decision maker has an attention vector $w \in \mathbb{R}^{d \times m}$ which measures how important it is that the attribute of the product be close to their preferred number.

The utility function of an alternative a for decision maker S_i is described as

$$U(a)_i = -\zeta \left(w_{S_i}^T (G_a - p_{S_i})^2\right)^{\frac{1}{2}}$$

where ζ is a constant defined by the researcher. This utility then gets passed into the softmax

function, similar to the Luce choice axiom $P(Y_i = c(a, A_i)) = softmax(U(A_i))_a$.

The key differences in this model compared to the aforementioned preference models are the concepts of the attention and preference parameter vectors and that parameters are updated sequentially for each choice. That is, if there are 100 observed choices for a decision maker / process then there will be 100 updates to the attention and preference parameter vectors. If these are saved, then changes in preferences and importance given to those preferences can be tracked. This also means that this model classifies as a simple reinforcement learning algorithm, as these are described by Sutton and Barto (2018).

3.3.2 Generalised Extreme Value Models

Generalised Extreme Value Models are used to allow for more complex choice structures where IIA can break down. The most popular implementation is the Nested Logit Models, which is the only one we will describe in this document.

"The generalised extreme value (GEV) distribution proposed by Jenkinson (1955) encompasses the three standard extreme value distributions: Fretchet, Weibull and Gumbel" (Bali, 2003).

Recall that the logit formulation can be derived by assuming that the unobserved part of the utility formulation follows a Gumbel distribution. Generalised Extreme Value models are derived from making the assumption that the unobserved parts in a model are jointly distributed as a GEV, of which the Gumbel distribution is a special case (Train, 2009).

Nested Logit Models partition the set of alternatives faced by a decision maker / process into nests where within the nest the following holds:

- For any two alternatives within the nests IIA holds.
- For any two alternatives in different nests IIA doesn't hold.

The following has been taken from Train (2009). Let the alternatives be partitioned into K non-overlapping subsets denoted $B_1, B_2, \ldots, B_K$. Usually these are defined by the researcher to be groups of similar alternatives, for example, one could consider food aisles such as yoghurts or meats to be a subset. Note that $B_1 \cup B_2 \cup, ..., \cup B_K = \mathcal{A}$, such that $B_i \cap B_j = \emptyset \forall i, j$, therefore, iterating over all subsets, will iterate over all alternatives. Once these subset are defined, the probability of choosing alternative a can be defined by the

following components. Suppose that the variables impacting preference for choice i are defined as $V_{i,a} = \lambda_a + G_a\gamma + X_i\beta_a^T + C_{S_i}\kappa_a^T$, suppose that alternative a is in nest k and we defined a measure of independence in the unobserved utility (in this case $\epsilon_{i,a}$ where $U_{i,a,} = V_{i,a,} + \epsilon_{i,a}$) among the alternatives in nest k as $\nu \in \mathbb{R}^K$. The probability of choosing alternative a according to the nested logit model is

$$P(Y_i = c(a, A_i)) = \frac{e^{\frac{V_{i,a}}{\nu_k}} \left(\sum_{j \in B_k} e^{\frac{V_{i,j}}{\nu_k}} \right)^{\nu_k - 1}}{\sum_{l=1}^{K} \left(\sum_{j \in B_l} e^{\frac{V_{i,j}}{\nu_l}} \right)^{\nu_l}}$$

.

A higher value of ν_k means greater independence and less correlation between the nests. When $\nu_k = 1$ we obtain the logit-type model.

In his book Train (2009) describes the process that McFadden (1978) used to derive GEV and shows practical application of nested logit models. One of these applications is in modelling transport choices, where Forinash and Koppelman (1993) have created nests for buses, cars and trains and show an improvement compared to multinomial logit models.

3.3.3 Elimination by Aspects

This discrete choice model is an early response to IIA. It formulates the choice process based on the attributes of the alternatives G. The formulation of this problem is such that it recursively eliminates alternatives based on the attributes they have and how much utility a the decision maker might attain from that attribute. For example, in the case of food delivery, a customer at a certain moment might have a preference for a type of cuisine like Indian, and then eliminates all alternatives (in this case restaurants) that aren't in this category. Then a person might want to order only from restaurants that have more than 4.5/5 stars from other user's ratings, so eliminates all alternatives that don't have this attribute. The process goes on until only one alternative remains.

The model assumes that each attribute of an item has a utility associated to it so there is a function $u : \mathcal{G} \to \mathbb{R}$ that takes each value of G_a and assigns an importance to it so we get functions such as $u_1(h)$ which is the utility of the first attribute in G being equal to h. If the first attribute is the colour of an item then $G_{a,1}$ denotes the colour of product a and will take values such as "red", "green"... etc . Let's define the list of distinct utilities from a set of alternatives A_i as U_{A_i}, this means that U_{A_i} contains all the unique utility numbers that

could be calculated from the alternatives that are in A_i. Then the probability of selecting an alternative a from a list of alternatives A_i is

$$P(Y_i = c(a, A_i)) = \frac{\sum_{z=1}^{\#G_a} u_z(G_{a,z}) P(Y_i = c(a, \{m \in A_i : G_{m,z} = G_{a,z}\}))}{\sum U_{A_i}}$$

This formula has a recursion until $\{m \in A_i : G_{m,z} = G_{a,z}\} = a$. In order to save space we will not go into a more detailed explanation here, however an elaborate example has been presented in the original manuscript (Tversky, 1972). The Nested Logit model approaches the Elimination by Aspects model as $\nu_k \to 0$ (Train, 2009).

3.3.4 Multiclass and binary classification

In the following sections we will describe Support Vector Machines, Neural Networks and Tree based models. These have an origin in what is known as multiclass classification. Multiclass models have a set up where the ground truths $Y \in \mathcal{A}$ can take one value of a finite set. Multiclass classification is well researched and we will not enter in much more detail about how exactly they work in this thesis, however, more information can be found in Aly (2005).

In preference modelling discrete choice can be thought of as the closest relative to multiclass problem. The key difference is that in multiclass prediction the set of different values that the ground truth can take is constant across all observations $A_i = \mathcal{A}$, whereas in discrete choice it can vary by observation $A_i \subseteq \mathcal{A}$. Consequently there can be ways of expressing discrete choices as multiclass tasks which will be examined in section 4.3.

Binary classification is a special case of multiclass classification where the ground truth can be of a specific class or not $Y \in \{0, 1\}$, which is related to the pairwise comparison task, where 0 can be one alternative and 1 the other, however, in binary classification, the alternatives are also constant across all observations in the data.

In their original formulation, multiclass classification sets up a problem to follow the independence from all alternatives approach. Some models however have been adapted to break from this assumption which we will present in the next sections.

3.3.5 Support Vector Machines

Support Vector Machines originally introduced by Cortes and Vapnik (1995) are non-probabilistic classifiers. This means that they do not define a probability for an outcome. Therefore, Support Vector Machines cannot be defined in terms of the behavioural hypotheses that have been presented in this chapter, since they are all descriptions rooted in how the probabilities of certain outcomes change.

Evgeniou et al. (2005) have introduced the idea of using support vector machines on pairwise comparison data and it was further explored by Gupta (2019).

The original formulation uses only alternative level variables and states that

$$Y_i = \begin{cases} \{j \succ s\} \text{ if } G_j\gamma \geq G_s\gamma \\ \{s \succ j\} \text{ if } G_s\gamma \geq G_j\gamma \end{cases}$$

So if it was to make a prediction $\hat{Y}_i$ rather than predicting a probability it would predict an outcome

$$\hat{Y}_i = \begin{cases} \{j \succ s\} \text{ if } (G_j - G_s)\gamma \geq 0 \\ \{s \succ j\} \text{ if } (G_j - G_s)\gamma \leq 0 \end{cases}$$

These can be augmented with the decision level and decision maker / process level variables as follows

$$Y_i = \begin{cases} \{j \succ s\} \text{ if } G_j\gamma + X_i\beta_j^T + C_{S_i}\kappa_j^T \geq G_s\gamma + X_i\beta_s^T + C_{S_i}\kappa_s^T \\ \{s \succ j\} \text{ if } \beta G_s\gamma + X_i\beta_s + C_{S_i}\kappa_s^T \geq G_j\gamma + X_i\beta_j + C_{S_i}\kappa_j^T \end{cases}$$

So if it was to make a prediction $\hat{Y}_i$ rather than predicting a probability it would predict an outcome

$$\hat{Y}_i = \begin{cases} \{j \succ s\} \text{ if } (G_j - G_s)\gamma + X_i(\beta_j - \beta_s)^T + C_{S_i}(\kappa_j - \kappa_s)^T \geq 0 \\ \{s \succ j\} \text{ if } (G_j - G_s)\gamma + X_i(\beta_j - \beta_s)^T + C_{S_i}(\kappa_j - \kappa_s)^T \leq 0 \end{cases}$$

It is very important in this model that there is at least some minimal variation between the covariates since when $(G_j - G_s)\gamma + X_i(\beta_j - \beta_s)^T + C_{S_i}(\kappa_j - \kappa_s)^T = 0$ the outcome is undefined.

Though this was not in the literature reviewed, one possible extension could be to explore

cases where the outcomes are close to zero, and classifying these as a draw. The following definition could be used to create ties for some $c \in \mathbb{R}_{\geq 0}$:

$$\hat{Y}_i = \begin{cases} \{j \succ s\} \text{ if } (G_j - G_s)\gamma + X_i(\beta_j - \beta_s)^T + C_{S_i}(\kappa_j - \kappa_s)^T \geq c \\ \{s \succ j\} \text{ if } (G_j - G_s)\gamma + X_i(\beta_j - \beta_s)^T + C_{S_i}(\kappa_j - \kappa_s)^T \leq -c \end{cases}$$

It can be shown that these models also produce transitive preferences similar to how it's done in proof 3.3.3.

Although the original formulation of SVMs are non-probabilistic, Platt et al. (1999) offers a method to map probabilities onto Support Vector Machines by proposing to learn $P(Y_i) = \frac{1}{1+e^{af+b}}$ where f is the unthresholded output of an SVM and a and b are parameters to be learned. Currently using this technique in choice models has not been explored much.

3.3.6 Neural Networks

We will present an adaptation of the description of neural networks found in Hastie et al. (2009). Neural Networks, create a network of linear models that interact in one or several hidden layers and is usually represented by a network diagram.

Generalised Linear Models can be thought of a one layer neural network and conversely neural networks can be thought of as a combination of linear models. This combination of linear models form what is referred to as layers. In the first layer a number of linear models, specified by the researcher are created from the inputs (covariates). Each linear model is referred to as a node in the next layer. Further layers may be created with linear models from the nodes. The number of layers specified and nodes in each layer is together referred to as the architecture of the neural network.

Recall that G_{A_i} are all the covariates for the alternatives involved in decision i and C_{S_i} be all the covariates for the decision maker / process involved in decision i and X_i are the decision level covariates.

A potential vanilla set up for a neural network could be in the context of subset choice models being treated as an m-class classification. Suppose there are m number of alternative level variables that is $\#G_{A_i} = m$ and let $q = \#X_i$ the number of decision level covariates, furthermore let $r = \#C_{S_i}$ be the number of covariates used for characterising a decision maker / process. We will denote $[X_i, G_{A_i}, C_{S_i}]$ as a stacked vector of dimensions $1 \times (q + m + r)$.

The nodes (or neurons) are defined by the researcher where $Z_j \in \mathbb{R}$ is node j and let $s = \#Z$. Each node is the result of an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, similar to the link function in generalised linear models.

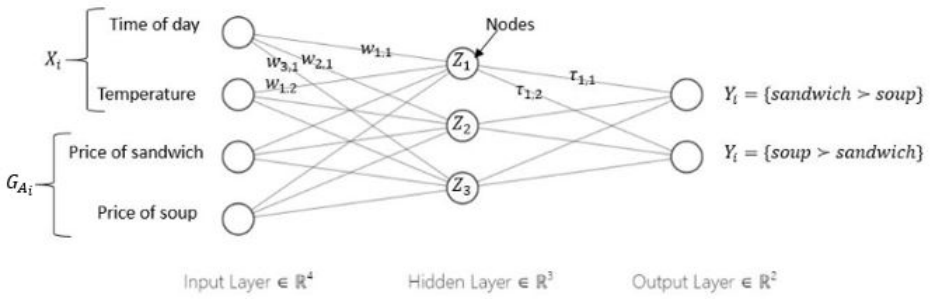
The nodes of a neural network are linear combinations of the inputs and the modelled target is a function of the linear combinations of the nodes such that for weight vectors $w \in \mathbb{R}^{s\times(q+m+r)}$ and $\tau \in \mathbb{R}^{s\times t}$ where t is the number of potential outcomes.

$$\begin{aligned} Z_j &= \sigma([X_i, G_{A_i}, C_{S_i}]w_j^T) && j = 1, ..., s \\ T_k &= \alpha_k + Z\tau_k^T && k = 1, ..., t \\ P(Y_i)_k &= g_k(T) && k = 1, ..., t \end{aligned}$$

where w and τ are learned parameters. The activation function σ is usually chosen to be the sigmoid function and $g()$ is usually the softmax. α_k is a constant learned that is a bias term for each potential outcome.

To illustrate, consider the pairwise comparison of whether to buy a soup or a sandwich for a meal. If we believe that the decision level variables that can impact this are: time of day and the temperature and the alternative level variables are the price of a sandwich and the price of soup, then we would draw the network diagram displayed in figure 3.8.

Figure 3.8: An example of a neural network diagram



From the image it can be appreciated how flexible a structure Neural Networks can provide. It can learn different impacts from the same decision level feature so in this case the temperature and time of day can naturally impact the probability of purchasing soup and sandwich differently so there is no need of specifying different number of parameters for the alternative level and observation level covariates. Neural networks can also be used in ways that allow for more flexible behavioural assumptions than IIA as we will see in the FATE

and FETA models soon. Neural Networks are a powerful tool and theories exist that for any function there can be a Neural Network that can approximate that function with a reasonable margin of error, this is often referred to as the universal approximation theorem (Hornik et al., 1989; Cybenko, 1989; Wang et al., 2020). A practical application of a neural network that predicts pairwise comparisons can be found in Price et al. (1995)

ListNet

ListNet (Cao et al., 2007) is a version of the Plackett-Luce algorithm which is described as:

$$P(Y_i = \{\mathcal{A}_1 \succ \mathcal{A}_2 \succ \ldots \succ \mathcal{A}_m\}) = \prod_{r=1;\mathcal{A}_r \in A_i}^{m} \frac{e^{f(X_i, G_r; \lambda_r \beta, \gamma)}}{\sum_{j=r;\mathcal{A}_j \in A_i}^{m} e^{f(X_i, G_j; \lambda_j \beta \gamma)}}$$

Where the the linear terms are replaced by functions (f), which are neural networks.

First Evaluate Then Aggregate

First Evaluate Then Aggregate (FETA) and First Aggregate Then Evaluate (FATE) (Pfannschmidt et al., 2019a, 2018) are some of the newest techniques in the field.

Recall that in section 3.3.1 we introduced the concept of utility. For this section it would help to generalise the concept of utility as a function $U : \mathbb{R}^h \to \mathbb{R}$, where $h \in \mathbb{N}$. The real values that U maps to correspond to how much value a specific alternative generates for a decision maker / process. The term $\mathbb{R}^h$ is a vector that describes each alternative which we have presented as G.

In FETA the utility of an item is defined by a function $U_0 : \mathcal{G} \to \mathbb{R}_{[0,1]}$. Furthermore a pairwise utility is also defined as $U_1 : \mathcal{G}^2 \to \mathbb{R}_{[0,1]}$. The pairwise utility function $U_1(G_a, G_b)$ for $a, b \in \mathcal{A}$ is interpreted as the utility function of alternative a in the presence of alternative b. This allows for breaking from IIA and making the model able to give different importance to an alternative depending on what other alternatives it is presented with.

Then the utility of choosing object a from A_i is defined as

$$U_{i,a} = U_0(G_a) + \frac{1}{\#A_i - 1} \sum_{j \in A_i \setminus \{a\}} U_1(G_a, G_j)$$

The predicted choice is $\hat{Y}_i = c(\{a \in A_i : U_{i,a} > t\}, A_i)$ where t is a threshold that is set by the researcher. Note, that in this algorithm the predicted choice can be a subset if there are several alternatives that are above the specified threshold.

This allows the model to capture preferences that are dependent on special alternatives. The variation is known as the FETA variation, because "an alternative is first evaluated in each sub-context and these evaluations are then aggregated" (Pfannschmidt et al., 2019a). There is an elaborate example showing how this works in the publication (Pfannschmidt et al., 2019a).

First Aggregate Then Evaluate

FATE uses a function $\phi : \mathcal{G} \to \mathcal{Z}$ where $\mathcal{Z} \in \mathbb{R}^m$ to create m dimensional embeddings of each alternative. Recall that here G contains the alternative level covariates. Let's define $C(a)_i = A_i \setminus \{a\}$ and $\mu_{C(a),i} = \frac{1}{\#C(a)_i} \sum_{x \in C(a)_i} \phi(G_x)$, this function captures the context in which a appears in observation i. For $t \in A_i$ we can generate a vector containing the scores $\mu_{C(t),i}$. By defining a utility function $U : \mathcal{G} \times \mathcal{Z} \to \mathbb{R}$ that maps the embeddings to a real number, the prediction becomes $\hat{Y}_i = c(\{a \in A_i : U(G_a, \mu_{C(a),i}) > t\}, A_i)$ for some threshold t. Note, that in this algorithm the predicted choice can be a subset if there are several alternatives that are above the specified threshold.

An example of how this works and an image of the neural network architecture can be seen in Pfannschmidt et al. (2019a).

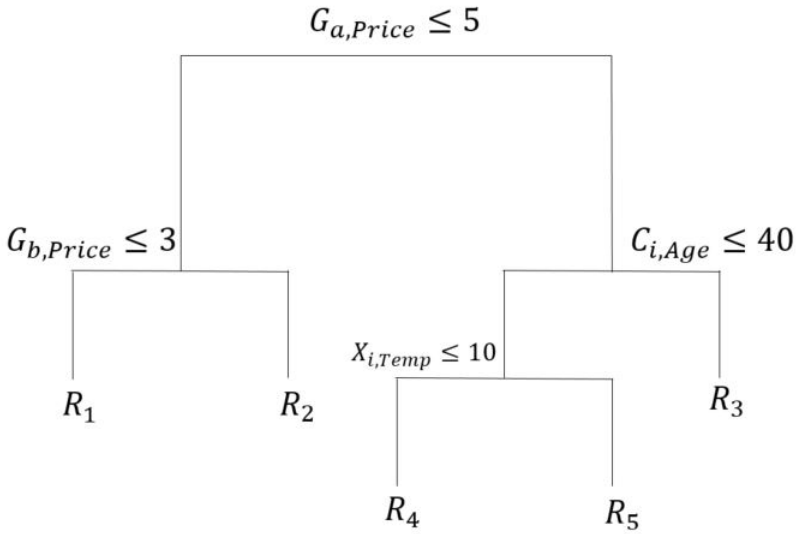
Both FATE and FETA have been constructed with using the vector G_{A_i} and from the sources consumed in this thesis it is not clear how to integrate other covariates such as the decision level variables (X_i) or the decision maker / process level variables C_{S_i}.

3.3.7 Tree based models

"Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one" (Hastie et al., 2009). Based on some input features, in our case it can be G, X or C, decision trees learn some rules on how to split the dependent variable up into k different regions $R_1, \ldots, R_k$ where $R_j \subset Y$, such that at each decision node only one input feature is being considered and the rule is whether this feature is greater or smaller than a threshold t, which is the parameter that is learned by the process. For example, a rule on G can be, "does the alternative cost more than $5?" then in the region of

this decision where this is true, we would have all the corresponding ground truth observations Y where there was an alternative that cost more than \$5. A visual example for a decision tree is in figure 3.9.

Figure 3.9: An example of what a decision tree that is trying to establish $P(Y_i = \{a \succ b\})$ using the prices of products a and b, the age of the decision maker and the temperature outside. In the regions $R_1, ..., R_5$ the frequency with which $Y_i = \{a \succ b\}$ can be observed and used to make predictions.



The splits in a tree are called branches and the end regions of the trees are called leaves. Predictions are made by looking at the information available in the leaves, for example the most popular decision in the leaf is what would be predicted for all cases where following the tree splits of the covariates would lead to that leaf.

Bradley Terry and Plackett-Luce trees Strobl et al. (2011) have introduced a tree based Bradley-Terry model, where the decision nodes are split in such a way as to maximise the coherency of which item is preferred in each region. The original idea is to make the splits using data about the decision makers / processes. This can be useful for showing how preferences are different between different types of people and can be used to produce some insightful results. Turner et al. (2020) have made a Plackett-Luce tree using the same principle as Strobl et al. (2011).

Since the nodes are all Bradley-Terry or Plackett-Luce models, each leaf follows the IIA assumption, however, similar to Nested Logit models, there is no reason why IIA would be

followed across different leaves. One thing that should be mentioned and examined when using these models is whether the alternatives still form a fully connected graph within each region. If this is not the case then the regions are too granular and some levels from the decision tree might need to be removed.

An example of Bradley Terry trees can be found in Strobl et al. (2011). The data used was of people who were asked to make pairwise comparisons of attractiveness of some of the participants in Germany's top model TV contest. The results show how tastes for the most attractive model vary by older vs. younger viewers, those who watches the show vs. those that haven't and men vs. women.

3.4 Methods of model estimation

In the earlier sections we have introduced models that define probabilistic and non-probabilistic processes which use parameters and some observed inputs to create predictions. In this section we will examine how those parameters are estimated in the fitting process by using the data. First we need to introduce more detail in the Supervised Machine Learning concept, which is a more detailed distinction of two processes we introduced at the start of this section, fitting also known as model estimation and prediction.

Definition 9. Fitting and Prediction

Fitting is the process of estimating the parameters in models. Let the set of all parameters be defined as $\Psi \in \mathbb{R}^d$ for some $d \in \mathbb{N}$, which is the number of parameters. *Hyperparameters* (shown in more detail in section 4.2) are fixed real numbers defined by the researcher to modify the fitting process let the set of all hyperparameters for some $w \in \mathbb{N}$ be defined by $\mathcal{H} \in \mathbb{R}^w$. Using definition 4 for data and p for a relational property (see table 2.6) and $k \in \mathbb{N}$, model fitting can be defined by $M^{(f)} : \mathcal{D} \times \mathcal{R}_p(\mathcal{A}^k) \times \mathcal{H} \to \Psi$. For a specific dataset of $n \in \mathbb{N}$ observations, $D \in \mathcal{D}^n$, $Y \in \mathcal{R}_p^n$ and hyperparameters $H \in \mathcal{H}$, let the learned parameters be denoted by $M^{(f)}(D, Y; H) = \hat{\Psi}$.

Prediction is the process of using the fitted parameters to predict future observations of the data. $M^{(p)} : \mathcal{D} \times \Psi \to \mathcal{R}_p(\mathcal{A}^k)$. Where the data to predict has $z \in \mathbb{N}$ observations, $D \in \mathcal{D}^z$, we refer to the predicted values of the ground truth as $M^{(p)}(D; \hat{\Psi}) = \hat{Y}$.

■

From the equations in definition 9 it can be seen that there's a conditionality between

fitting and predicting. The predicting process uses the fitted parameters for prediction. Therefore, prediction should be only made once the parameters have been fitted. The reason we say "should" and not "can only be" is because technically it is possible to define any parameters randomly or manually for predicting, but this is likely to perform very poorly, and a necessary objective of Supervised Learning is to be at least more accurate than making random predictions. The remaining of this section is about methods of fitting.

3.4.1 Optimisation based approaches

Maximum Likelihood

In this section we will describe the approaches for estimating the generalised linear models that we have described in section 3.3.1 using maximum likelihood. Suppose that we have a set of observations $Y_i \in \mathcal{R}_{ch}^N$ where $N \in \mathbb{N}$ is the number of observations. If we assumed that Y_i is generated by a logit-type model as described in section 3.3.1 we would have the probability of observing a specific outcome as a function of the parameters Ψ as $P(Y_i) = f(D_i; \Psi)$ where $f : \mathcal{D} \times \mathbb{R}^d \to \mathbb{R}_{[0,1]}$, for some $d \in \mathbb{N}$, which is the number of parameters.

If we fix some value for the parameters, we can ask the question what is the probability or likelihood of having observed the sequence of Y_i under these assumed values for the parameters with the data we observed? Under maximum likelihood our objective is to find the values for parameters $\hat{\Psi}$ for which the probability of having observed all the data is the highest out of any possible values that the parameters in Ψ could have.

Definition 10. Adapted from Wasserman (2013).

Let $Y_1, \ldots, Y_N$ be identically and independently distributed with a probability density function $f(D_i; \Psi)$ where Ψ are parameters of the function. The likelihood function is defined by

$$\mathcal{L}(\Psi) = \prod_{i=1}^{n} f(D_i; \Psi)$$

. The log-likelihood function is defined by $\ell(\Psi) = \log \mathcal{L}(\Psi)$ The maximum likelihood estimators denoted by $\hat{\Psi}$ are the values that maximise $\mathcal{L}(\Psi)$. ■

The log-likelihood is often used because working with summation is simpler than with multiplication, and the log function is monotonically increasing, therefore a value that maximised the log of a function, would also maximise the function itself. Sometimes the solution

to finding $\hat{\Psi}$ is closed form, however when the answer is not closed form, such as usually is the case with logit-type models, the solution is obtained via numerical optimisation (Hastie et al., 2009).

In this thesis we will not go through every likelihood function of every model that has been presented in this chapter. However, we will provide the likelihood function for a vanilla Bradley-Terry model. Let

$$\mathbb{I}(Y_i = \{a \succ b\}) = \begin{cases} 1 \text{ if } Y_i = \{a \succ b\} \\ 0 \text{ otherwise} \end{cases},$$

$$\text{let } g(x) = \frac{1}{1+e^x},$$

then

$$\begin{aligned} \ell(\lambda)_{\text{Bradley-Terry}} = \sum_{i=1}^{N} \Big(& \mathbb{I}(Y_i = \{A_{i,1} \succ A_{i,2}\}) \ln(g(\lambda_{A_{i,1}} - \lambda_{A_{i,2}})) + \\ & \mathbb{I}(Y_i = \{A_{i,2} \succ A_{i,1}\}) \ln(1 - g(\lambda_{A_{i,1}} - \lambda_{A_{i,2}})) \Big) \end{aligned} \tag{3.4}$$

where $A_i \in \mathcal{A}^2$ and $A_{i,1}$ and $A_{i,2}$ are the first and second elements of A_i respectively, which might map to different alternatives for each row of i and are not to be interpreted here as two constant alternatives named '1' and '2'.

Unconstrained Numerical optimisation

First of all, it should be pointed that most of numerical optimisation is expressed in terms of minimising functions, rather than maximising. We have shown earlier that maximum likelihood requires maximising a function. It is not uncommon in machine learning literature for this to be expressed as a minimisation problem of the negative of the log likelihood. Therefore our task would be shown in numerical optimisation literature as such $\min_{\Psi} -\ell(\Psi)$. This is also sometimes referred to as the objective function.

> "In unconstrained optimisation we minimise an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is $\min_x f(x)$ where $x \in \mathbb{R}^n$ is a real vector with $n \geq 1$ components and $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function" (Nocedal and Wright, 2006).

In numerical optimisation global and local optimas are terms often referred to. A global

maximum point is the highest point in the entirety of the function

$$\mathcal{L}(\Psi^*) > \mathcal{L}(\Psi) \forall \Psi \neq \Psi^*$$

. A local maximum point, is one where the function is at a maximum within a neighbourhood, but it isn't necessarily the global maximum point. For some $k \in \mathbb{R}$;

$$\mathcal{L}(\tilde{\Psi}) > \mathcal{L}(\Psi) \forall \Psi \in \mathbb{R}_{[\tilde{\Psi}-k, \tilde{\Psi}+k]} \setminus \tilde{\Psi}$$

.

Note that mathematically a local minimum point is defined by a point where the gradient of the likelihood function is 0, but has a positive second derivative, whereas a local maximum point is defined by the gradient being zero and the second derivative being negative. By these definitions all global maximum / minimum points are also local maximum minimum points. It is impossible to know what the global maximum and minimum point is without finding all the local maxima and minima of a function. In general, this is highly impractical therefore most optimisation methodologies are said to only find local maximum or minimum points.

There are two search strategies for finding the parameters that minimise a function. One is called a line search and the other trust regions. In this thesis we will focus on line search methods.

> "In the line search strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along (p_k) can be found approximately by solving the following one dimensional minimisation problem to find a step length α:
>
> $$\min_{\alpha>0} f(x_k + \alpha p_k)$$
>
> "(Nocedal and Wright, 2006).

There are several schools of thought about how to define α. A popular approach can be simply setting it to 1 and not changing though sometimes this doesn't work. For the following equations we will present strategies assuming $\alpha = 1$. Amongst other popular methods is running a separate line search method for updating α, often to satisfy one of three conditions Armijo (1966), Goldstein (1967) or Wolfe (1969).

The basic idea behind line search methods is that we initialise the parameters of the functions with random numbers and then see the gradient of the likelihood function at that point. We then use the gradient to update the initialised parameters in the direction where the negative log likelihood function is going down, which in other words is going up in the

likelihood function.

This minimisation usually relies on a quadratic approximation of the equation obtained by the first three components of a Taylor series approximation.

$$f_{k+1} = f(x_k + p) \approx f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p$$

Where ∇f_k is the derivative of the function f_k and $\nabla^2 f_k$ is the Hessian matrix in the case of what is known as Newton methods and it is a positive definite approximation to the Hessian (B_k) for approaches known as quasi-Newton. In practice quasi-Newton methods are used often because there is no guarantee that the Hessian would be a positive definite matrix. When the Hessian is not positive definite then $f(x + p_k)$ would not be defined. In a case where the Hessian is positive definite then for the Newton method the step is defined by $p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k$ and the update to the parameters is $x_{k+1} = x_k + \alpha_k p_k$.

The most intuitive and simplest to implement line search rule is one called the method of steepest descent and is also known as batched gradient descent. It is a method that moves along $p_k = -\nabla f_k$ in each iteration. The benefits of this solution is that it is simpler because it doesn't need to calculate the Hessian. The hindrance of this method is that it can be slower than some of the other methods used. In the case of the Bradley-Terry model with no covariates for which the loss function was shown in 3.4 this would be the equivalent of making the following update rule to the parameter of alternative a, full calculation of how to arrive here shown in appendix 9.1.2.

$$\lambda_a^{\text{new}} = \lambda_a^{\text{old}} + \sum_{k=1}^{N} \sum_{b \in A_k} \mathbb{I}(Y_k = \{a \succ b\}) \frac{e^{\lambda_b^{\text{old}}}}{e^{\lambda_a^{\text{old}}} + e^{\lambda_b^{\text{old}}}} + \mathbb{I}(Y_k = \{b \succ a\}) \frac{-e^{\lambda_a^{\text{old}}}}{e^{\lambda_a^{\text{old}}} + e^{\lambda_b^{\text{old}}}}. \quad (3.5)$$

So now that we are updating the parameters of the function x, a question is when to stop the process. The closer we get to the minimum point of a continuous function, by definition there are diminishing returns of each iteration; we pay the fixed cost of an iteration for every update, but since at minima $\nabla f_k \to 0$ the updates are also $p_k^N \to 0$. There often comes a usually subjective point where it is no longer worth it to make a new iteration for a relatively small adjustment in parameters, which would not impact predictions by this model to an extent that a researcher would deem significant. Setting this point is referred to as the *tolerance* $\epsilon \in \mathbb{R}_{>0}$ and the stopping criteria for an algorithm usually is $||\nabla f_k|| < \epsilon$, however, since stopping criteria is subjective, there can be other stopping criteria also, such as setting a computational running time.

The BFGS algorithm A method that has been implemented for several preference models by researchers is the quasi-Newton method developed by Broyden (Broyden, 1970), Fletcher (Fletcher, 1970), Goldfarb (Goldfarb, 1970) and Shanno (Shanno, 1970) known as the BFGS method which is defined by:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{3.6}$$

where $s_k = x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$ and B_k is a positive definite approximation to the Hessian. In the case of this update the new step is defined by $p_k^N = -(B_k)^{-1} \nabla f_k$ (Nocedal and Wright, 2006).

MM algorithms

MM stands for majorise/minimise when a function is being minimised and minorise/maximise when a function is being maximised. MM algorithms are useful for making it possible to maximise or minimise non-differentiable functions, and also have the property of speeding up the learning process by avoiding matrix inversions.

Definition 11. "For a specific value of the parameter θ denoted by $\theta^{(m)}$; the function $g(\theta|\theta^{(m)})$ is said to majorise a real valued function $f(\theta)$ at the point $\theta^{(m)}$ provided the following two conditions:

$$\text{domination for all } \theta : g(\theta|\theta^{(m)}) \geq f(\theta) \forall \theta$$

$$\text{tangency at } \theta^{(m)} : g(\theta^{(m)}|\theta^{(m)}) = f(\theta^{(m)})$$

The function is said to minorise if $-g(\theta|\theta^{(m)})$ majorises $-f(\theta)$ at $\theta^{(m)}$ In a majorise-minimise MM algorithm, we minimise the majorising function $g(\theta^{(m)}|\theta^{(m)})$ rather than the actual function $f(\theta)$"

(Hunter and Lange, 2004). ■

Hunter et al. (2004) have invented MM algorithms for Bradley-Terry models with home team advantage and with the Rao-Kupper variation using ties.

The log likelihood of the Bradley-Terry model for latent strength parameter vector λ can be expressed as

$$\ell(\lambda) = \sum_{i=1}^{m} \sum_{j=1}^{m} [\omega_{ij} \ln(\lambda_i) - \omega_{ij} \ln(\lambda_i + \lambda_j)],$$

where ω_{ij} denotes the number of times alternative i was preferred to alternative j, that is

$$\omega_{ij} = \sum_{k=1}^{\#Y} \mathbb{I}(Y_k = \{i \succ j\})$$

. The objective is to find the parameter vector λ that maximises the function $\ell(\lambda)$. From equation 3.5 we can see that when using previously mentioned gradient based numerical optimisation techniques such as the method of steepest descent, for updating a parameter in the Bradley Terry model λ_i, we need the parameters of the other alternative also λ_j. When using MM algorithms for estimating Bradley-Terry it is possible to separate the update parameters which yields faster conversion. Hunter et al. (2004) propose the minorising function at iteration of the update k, that allows separating these parameters.

$$g_k(\lambda) = \sum_{i=1}^{m}\sum_{j=1}^{m} \omega_{ij} \left[\ln \lambda_i - \frac{\lambda_i + \lambda_j}{\lambda_i^{(k)} + \lambda_j^{(k)}} - \ln(\lambda_i^{(k)} + \lambda_j^{(k)}) + 1 \right]. \tag{3.7}$$

Where $\lambda_i^{(k)}$ and $\lambda_j^{(k)}$ are known real values of the current iteration. The fact that this formulation satisfies the MM condition is based on the observation that $-\ln(\lambda_i + \lambda_j)$ is convex and based on the supporting hyperplane property of $f(x) \geq f(y) + \frac{df(y)}{dy}(x - y)$ where f is a convex function:

$$-\ln x \geq 1 - \ln y - (x/y) \text{ with equality if and only if } x = y$$

Substituting in this formulation $x = \lambda_i + \lambda_j$ and $y = \lambda_i^{(k)} + \lambda_j^{(k)}$ yields

$$-\ln(\lambda_i + \lambda_j) \geq 1 - \ln(\lambda_i^{(k)} + \lambda_j^{(k)}) - \frac{\lambda_i + \lambda_j}{\lambda_i^{(k)} + \lambda_j^{(k)}}$$

meaning that $g_k(\lambda)$ will always be below $\ell(\lambda)$ but being the same at the points $\lambda_i = \lambda_i^{(k)}$ and $\lambda_j = \lambda_j^{(k)}$. Therefore this must mean that any new values for λ such that $g_k(\lambda) \geq g_k(\lambda^{(k)})$ translates to $\ell(\lambda) \geq \ell(\lambda^{(k)})$.

If we let $W_i = \sum_j w_{i,j}$ denote the number of times alternative i has been preferred to any other alternative, and $N_{ij} = \omega_{ij} + \omega ji$ which is the number of times i and j have been compared. Function $g_k(\lambda)$ is maximised by the iteration:

$$\lambda_i^{(k+1)} = W_i \left[\sum_{i \neq j} \frac{N_{ij}}{\lambda_i^{(k)} + \lambda_j^{(k)}} \right].$$

Since for some parameters we already have an updated strength parameter, this equation can be written in what is referred to as a cyclical form

$$\lambda_i^{(k+1)} = W_i \left[\sum_{j<i} \frac{N_{ij}}{\lambda_i^{(k)} + \lambda_j^{(k+1)}} + \sum_{j>i} \frac{N_{ij}}{\lambda_i^{(k)} + \lambda_j^{(k)}} \right].$$

"One feature of the function [in equation 3.7] that makes it easier to maximise than the original log-likelihood is the fact that it separates the components of the parameter vector [λ]" (Hunter et al., 2004).

With the same spirit of separating the components in their work, Hunter et al. (2004) derive the MM algorithm for Bradley-Terry with ties, home advantage and Plackett-Luce.

Numerical Optimisation for a Normalised database set up

As we have discussed earlier the proper set up for data in preference models is a relational database. This is specifically convenient due to the fact that researchers would often need to store information on the alternative and decision maker levels also. Most of the numerical optimisation literature is based on the assumption of having a pooled data table on which optimisation is performed. This often leads to duplicating the same observation in creating a table that is much wider than it needs to be. Recently researchers have been proposing ways to tackle this. Kumar et al. (2015) have been looking at how to leverage the relational database structure to speed up estimation of generalised linear models. They have proposed a new technique called factorised learning, which on batched gradient descent (steepest descent) shows an improvement in processing speed, without a decline in model accuracy. Whilst there is no algorithmic description on how factorised learning works in the published writings, there is an R package developed by the authors of these papers called *santoku* from which it should be possible to figure out how it works exactly (Kumar, 2016).

3.4.2 Other model estimation methods

During the course of this MPhil there has been a more detailed investigation of linear models, in the following sections we will give a brief description on how some of the other algorithms mentioned in this document are estimated.

Support vector machines

The basic idea behind fitting support vector machines is to find a hyperplane that separates the observations in such a way that the normed distance between the hyperplane (which constitutes the border between the different classes of observations) and the nearest observations to the hyperplane from each class are maximised. In practice finding a perfect boundary is impossible some observations will cross over, these are called slack variables in the literature.

Support vector machines are solved via a quadratic programming solution deploying Lagrange multipliers for constrained optimisation.

Neural Networks

"With the softmax activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood", this is achieved via a technique called back-propagation (Hastie et al., 2009). Note that this is only the case if the last layer of the neural network is the log-loss, which is the relevant case for the preference models we have discussed. There is a lot more to neural network optimisation, which we will leave out of scope for this report though all of them use forward and back-propagation, we will mention that most recently there has been some research by Baydin et al. (2022) on how to optimise neural networks without using back-propagation, which could yield significant computational improvements.

Tree based algorithms

Finding the right partition in decision trees is done usually in a greedy way to optimise a metric for each split. By "a greedy way" what is meant that all possible splits are tried and the one that is best at optimising the metric is used. The metrics to optimise are usually the sum of squares for regression and the gini index or the cross-entropy deviance for classification (Hastie et al., 2009).

It is a little different for Bradley-Terry trees. The method described by (Strobl et al., 2011) is:

1. Fit a BT model to the paired comparisons of all subjects in the current subsample starting with the full sample.
2. Assess the stability of the BT model parameters with respect to each available covariate.

3. If there is significant instability, split the sample along the covariate with the strongest instability and use the cutpoint with the highest improvement of the model fit.
4. Repeat Steps 1-3 recursively in the resulting subsamples until there are no more significant instabilities (or the subsample is too small)

Details on how this is exactly done can be found in the original paper.

Bayesian inference

Bayesian methods can be used to express uncertainty about the parameters learned by the models. Based on the definition in Hastie et al. (2009) for a classic Bradley-Terry model it would define the sampling as $P(Y|\lambda)$ and would define a prior distribution $P(\lambda)$ reflecting the researcher's knowledge about the strength parameters λ and the posterior distribution is computed as

$$P(\lambda|Y) = \frac{P(Y|\lambda)P(\lambda)}{\int P(Y|\lambda)P(\lambda)d\lambda}$$

which represents the researcher's knowledge about λ after seeing the data. Sampling from this posterior distribution can help express a range of likely true values for the strength parameters.

Bayesian inference has not been researched in this MPhil. However, there are a couple of Bayesian inference models for Bradley Terry that can be found.

> "[S]everal authors have proposed to perform Bayesian inference for (generalized) Bradley-Terry models (Adams, 2005; Gormley and Murphy, 2008; Guiver and Snelson, 2009). The resulting posterior density is typically not tractable and needs to be approximated. An expectation-propagation method was developed by Guiver and Snelson (2009); this yields an approximation of the posterior which can be computed quickly and might be suitable for very large-scale applications" (Caron and Doucet, 2012).

3.5 Examples of applications of supervised preference models

So far we have discussed certain preference models are defined and estimated. There is a wide range of questions preference models can solve, some of them are explored here to give the reader an idea of the breadth of the domain these models have applications in and the nature of questions that one might want to answer with preference models.

What odds should be given to someone who wants to bet on an order in a race? Imagine a odds maker at the horse races and some clients who would not like to only place a bet on the winning horse, but also on the fact that a horse might end up in the top three. For the odds maker to give odds where they can expect some profit, at first they need to know the true odds of permutations in the field. This is the initial question which led to the now popular Plackett-Luce model (Plackett, 1975), with which the probabilities of permutations can be estimated.

What is the the most relevant to least relevant document for this search query? Search engines provide a ranking of documents for a query from most relevant to least relevant. Sometimes to support the improvement of these ranking there are some examples where people have manually ranked the relevance of documents to certain search queries. These human created ranks are often used to learn ways of ranking previously unseen documents for previously unseen queries (Cao et al., 2007) and full rank models are being used for these sorts of tasks.

How do people perceive different crimes, and how much worse do they think some crimes are than others? Sometimes it's not only a rank that we would like to know but to also have a concept of the distance between ranks, the difference between the first and second ranked objects might be much larger than the gap between the second and the third. Being able to deduce these differences can bring understanding of certain social issues into new dimensions like the perceived seriousness of a crime as we have seen in figure 3.5 (Thurstone, 1927b).

Recommender systems Recommender systems are a popular way for businesses to keep their customers engaged. Take for example a case where people may have ranked some movies in a partial order and based on this we want to recommend to them other movies to watch. Making such recommendations is a very important source of revenue to many services and retailers as good recommendations can drive sales and engagement for businesses (Pathak et al., 2010). The problem to recommend movies to people became famous when Netflix offered a $1 million prize to the people who could beat their recommender systems; the competition began in 2006 and in 2009 someone has finally passed the threshold for the award (Netfix, 2006).

How will this product perform when launched in a different store and how will it affect other products' sales? Imagine an owner of two grocery stores is selling a slightly different combination of yoghurt items in the two stores. One of the products which is sold in store 1 but not in store 2 is performing very well. The store owner begins to wonder whether they should sell this item also in store 2, but is also concerned about how the sales of the other products in that store might be affected. Being able to foresee how shoppers' preferences might change due to the introduction of a new product in a store is key for knowing how many different items to stock and to begin to understand what the impact will be on profits. Models such as FATE and FETA can have an interesting application in estimating how dynamics between preferences of an alternative might change under different ranges.

How much can car usage be reduced by increasing congestion charge? Many cities charge drivers an environmental fee called the congestion charge. The idea is to incentivise car owners to take public transport more often. This also helps traffic in the roads. Whether a person chooses to drive or take public transport, is a discrete choice. Being able to predict how an increase in the congestion charge will affect the preference for driving is a key component for successfully setting the right rate. One way in which this has been done in the past is by building a discrete choice model in which the congestion charge is a covariate (Brathwaite and Walker, 2018). An accurate model will be able to predict how much car usage might change as a result of changing the congestion charge.

Which team will win the basketball match? It is customary in many social circles during March Madness to create predictions of which teams will go through each round, this is also referred to as a bracket (Rodrigues, 2019). It is also becoming a tradition in some data science communities to participate in a Kaggle (2019) competition which asks for the competitors to predict for each possible combination of teams the probability with which each would win and then the person or team that gets the lowest log loss on the predicted probabilities vs. the actual outcome of the games wins the competition (submissions of predictions must be given before the games are played). An added excitement to this competition is that no one to date has ever predicted the perfect bracket (Benzie, 2019). This probably has to do with the fact that there are 2^{63}, that is 9.2 quintillion different possible brackets and the NCAA estimates that someone who knows about basketball might have a 1 in 120.2 billion chance of getting the right bracket (Wilco, 2020). Pairwise comparison models would be a natural option for predicting such problems.

Chapter 4

Preference models in machine learning pipelines

In chapter 3 we have presented various models that can be used for solving the same machine learning task, for example, a Bradley-Terry decision tree vs. a Nested Logit model. It is important to be able to assess which model is performing best on predicting the problem. As such, the first thing that we will discuss in this chapter is what is called an evaluation metric which is a numeric, measurable indication to assess which algorithm completed the preference task more adequately. In this section we will describe the main components to machine learning pipelines. A machine learning pipeline is a series of steps taken to ensure that machine learning algorithms improve their accuracy. Some of the key components of the machine learning pipeline discussed in this thesis are: removing and transformation of covariates, tuning of hyperparameters and composition of several models.

4.1 Evaluating the accuracy of preference models

Suppose we have a preference task of predicting which team will win a basketball match and we have used a Thurstone and a Bradley-Terry model to make predictions and now we would like to know which one is more accurate. Evaluation of the model predictions in machine learning refers to understanding how well the models have completed the task and it is the final component of a supervised learning task as described in definition 5. Evaluating a model's predictive accuracy is not the only form of evaluating models, for example one can

also consider other aspects such as interpretability, however in this thesis we only consider evaluation of predictive accuracy and will refer to this as model evaluation.

4.1.1 Evaluation metrics

Evaluation metrics are functions that take predictions of a model ($\hat{Y}_i$) and the ground truth (Y_i) and return a real number $f_{\text{eval}} : \mathcal{R}_p^2 \to \mathbb{R}$ where p is some property for the relation expressed as indicated in table 2.6. Evaluation metrics also exist for probabilistic predictions where the metric would be $f_{\text{eval}} : \mathbb{R}_{[0,1]} \times \mathcal{R}_p \to \mathbb{R}$, where the first element in the domain is the probability estimation of observing the second element. Depending on the measure, a larger number can mean that the predictions are more accurate, in which case they are often referred to as *accuracy metrics* or it can mean that the model is less accurate in which case they're referred to as *error metrics*. It is not uncommon for this to be the same as the loss or maximum likelihood function of the models that have been learned, however, there are many other functions that can be used for evaluation. We will discuss some of the more common ones observed in research involving types of preference models, however, it should be noted that this is not an exhaustive list, and often it can be found that researchers define their own tailored metrics.

We break down evaluation metrics into two groups, those that are adequate for evaluating: pairwise comparisons, discrete choices and subset choices, we will refer to these as choice based evaluation methods; and those that are adequate for evaluating partial and full orders, to which we will refer to as rank based evaluation methods.

4.1.2 Evaluation metrics based on binary classifiers adapted for choice based evaluation methods

In this subsection we provide popular evaluation metrics used for binary classification and multiclass classification evaluation that could be adapted for choice based evaluation methods. In the next section we will be referencing research conducted in the supervised preference models space that have used some of these metrics for evaluation. In binary classification $Y_i \in \{0, 1\}$ and for probabilistic models the prediction of the outcome $\hat{Y}_i \in \mathbb{R}_{[0,1]}$, whereas for non-probabilistic models it is $\hat{Y}_i \in \{0, 1\}$. Popular binary classification evaluation metrics are referred to in table 9.1. For many of these metrics there is a threshold $t \in \mathbb{R}_{(0,1)}$ which is used to decide whether something belongs to the class or not; it is usually set to 0.5, although different choices could also be used, specifically when there are imbalanced classes, that is,

there are many more observations of one class than there is of another. When a model is not probabilistic then mathematically all these measures except for the cross entropy loss still work with any arbitrary threshold.

The core principle behind binary classification is that an event either happens or it does not happen. It is possible to frame pairwise comparison, discrete choice and subset choice tasks as binary classification tasks by saying that in these preference tasks each alternative is either chosen or not chosen. The key difference between framing one of these preference tasks as a classification task is that where one of these preference tasks would see only one observation where a choice is made from the set of alternatives A_i in observation i, a binary classification task would see $\#A_i$ observations for observation i, since each alternative in A_i can be thought of as if it was chosen or not chosen. The reason why preference models are used instead of binary classification models is that the latter would consider the decision to choose each alternative from A_i as independent from what the alternatives are in A_i, whereas as we have seen in the previous chapter, preference models inherently consider how the alternatives available for a decision would impact the likelihood for choosing either one of the available alternatives. However, since the preference predictions of pairwise comparisons, discrete choices and subset choices can be framed as classification predictions, classification evaluation metrics become accessible to researchers for measuring how well each of these preference tasks has performed, which is how they are in general evaluated.

In binary choice there are only two events. Something is either true or not, and the dependent variable takes on the values 1 and 0 respectively for these two cases. A true positive in binary classification is referred to as predicting 1 when the observed ground truth was also 1, a true negative would be predicting 0 when the observed ground truth was also 0, a false positive is predicting 1 when the observed ground truth was 0 and a false negative is predicting 0 when the observed ground truth was 1.

This metric can be used directly with pairwise comparisons, when defining the problem in a binary classification setting as we have shown in table 2.11 (reproduced below for convenience).

Repeated table 2.11: Example of pairwise comparison observations where the ground truth **team 1 won (1/0)** is shown as a binary classifier.

team 1	**team 1**	**team 1 won (1/0)**	**location**
Virginia	Purdue	1	Richmond, Virginia
Kentucky	Auburn	0	Dallas, Texas
Duke	MI State	0	New York, New York

For a vector of predictions $\hat{y}$ and ground truth values y, the true positives in a binary classification would be defined as

$$tp(\hat{y}, y) = \sum_{i=1}^{\#y} \begin{cases} 1, & \text{if } y_i = 1 \text{ and } \hat{y}_i = 1 \\ 0, & \text{otherwise} \end{cases}.$$

For true negatives (tn) the condition for summing 1 would change to $y_i = 0$ and $\hat{y}_i = 0$, for false positives (fp) it is $y_i = 0$ and $\hat{y}_i = 1$ and for false negatives (fn) it is $y_i = 1$ and $\hat{y}_i = 0$.

These measures by themselves are not usually used for evaluating model predictions, however, they are key building blocks in popular metrics, such as:

- Accuracy, which is the proportion of total observations correctly classified, used by Király and Qian (2017) and Coulom (2008) to evaluate predictions on premier league matches. The formula for it is

$$f_{\text{accuracy}}(\hat{y}, y) = \frac{tp(\hat{y}, y) + tn(\hat{y}, y)}{tp(\hat{y}, y) + fp(\hat{y}, y) + tn(\hat{y}, y) + fn(\hat{y}, y)}$$

  .

- Precision: proportion of alternatives identified as preferred by the model being actually preferred. an example usage is McHale and Morton (2011) who have analysed tennis player's likelihood of winning a match,

$$f_{\text{precision}}(\hat{y}, y) = \frac{tp(\hat{y}, y)}{tp(\hat{y}, y) + fp(\hat{y}, y)}$$

  .

- Recall: proportion of all preferred alternatives have been identified correctly by the model.

$$f_{\text{recall}}(\hat{y}, y) = \frac{tp(\hat{y}, y)}{tp(\hat{y}, y) + fn(\hat{y}, y)}$$

  .

- F1 score: harmonic mean of the precision and recall,

$$f_{F_1}(\hat{y}, y) = \frac{2 \times f_{\text{precision}}(\hat{y}, y) \times f_{\text{recall}}(\hat{y}, y)}{f_{\text{precision}}(\hat{y}, y) + f_{\text{recall}}(\hat{y}, y)}$$

  .

- F-beta-score

$$f_{F_\beta}(\hat{y}, y, \beta) = \frac{(1+\beta^2) \times f_{\text{precision}}(\hat{y}, y) \times f_{\text{recall}}(\hat{y}, y)}{\beta^2 f_{\text{precision}}(\hat{y}, y) + f_{\text{recall}}(\hat{y}, y)}$$

  , where β is a parameter defined by the researcher to trade off precision and recall, when $\beta > 1$ recall is weighed higher than precision and when $\beta < 1$ precision is weighed higher than recall. A balanced weighting is when $\beta = 1$, which is the same as the F_1-score.

Suppose now that there are more than two outcomes, for example as it is in the famous *iris* dataset (Fisher, 1936), where sepal and petal length and width measurements are given for three species of flowers in the iris genus and machine learning algorithms can be trained to predict which type of flower an observation belongs to based on its measurements. This type of problem is known as multiclass classification. The way the above mentioned four metrics are translated for multiclass classification is by treating each class as if it were a binary prediction. For example, if the three classes were the *setosa*, *versicolor* and *virginica* species of the iris genus, then we can treat each class as its own binary classification problem by saying whether the plant in the observation belongs to that class or not. If the algorithm correctly predicts that the plant in a given observation belongs to the class *setosa* then that is a true positive, whereas correctly predicting that it doesn't belong to this class is a true negative. This way, for each observation and each class there is a true positive, true negative, false positive and false negative calculation, which means that each class will have its own accuracy precision, recall and F_1-score calculation. For multiclass classification, when one of these metrics, say accuracy, is being reported, the calculation is the mean accuracy across the classes.

To illustrate how this calculation would work on a subset choice, we will illustrate by a case where there are three alternatives for every choice $\{a, b, c\}$. For this example, we will represent each alternative as a binary choice that is a is either chosen (1) or not chosen (0). Suppose that there is an algorithm that predicts a subset choice and we also convert its predictions into binary format. We now take three example observations of true subset choices $y = [\{a, b\}, \{c, b\}, \{b\}]$ and their respective predictions $\hat{y} = [\{a\}, \{c, b\}, \{c\}]$. In table 4.1 we show the categorisation of each prediction.

Based on the formula above we could calculate that the accuracy of a is 1, the accuracy for b is $\frac{1}{3}$ and the accuracy on c is $\frac{2}{3}$. The average accuracy for these predictions would then be $\frac{2}{3}$. However, there is a much quicker way of getting to this result than having to convert every alternative into a binary choice.

Table 4.1: Showing the true positive / negative and false positive / negative counts for an example in subset choice, by using the same technique as multiclass classification uses. In the table we denote $(a \in y) \in \{1, 0\}$ to say that a was (1) or was not (0) in the true subset choice, and $(a \in \hat{y}) \in \{1, 0\}$ to show that a was (1) or was not (0) in the predicted subset choice.

y	$\hat{y}$	$a \in y$	$a \in \hat{y}$	**type for** a	$b \in y$	$b \in \hat{y}$	**type for** b	$c \in y$	$c \in \hat{y}$	**type for** c
$\{a, b\}$	$\{a\}$	1	1	True Positive	1	0	False Negative	0	0	True Negative
$\{c, b\}$	$\{c, b\}$	0	0	True Negative	1	1	True Positive	1	1	True Positive
$\{b\}$	$\{c\}$	0	0	True Negative	1	0	False Negative	0	1	False Positive

For observation i, recall that A_i is the set of available alternatives, we will refer to $y_i \subseteq A_i$ as choices and $\hat{y}_i \subseteq A_i$ as the estimated choices. We will further denote $w_i = A_i \setminus y_i$ as the alternatives not chosen and $\hat{w} = A_i \setminus \hat{y}_i$ as the predicted alternatives not chosen. With these new definitions for the metrics of true positives, false negatives, false positives and true negatives become:

- True positives: number of alternatives that a model predicted would be chosen and were actually chosen $f_{TP}(\hat{y}, y) = \sum_i \#(y_i \cap \hat{y}_i)$.
- True negatives: number of alternatives that a model predicted would be not chosen and were actually not chosen. $f_{TN}(\hat{w}, w) = \sum_i \#(w_i \cap \hat{w}_i)$.
- False positives: number of alternatives that a model predicted would be chosen and were actually not chosen. $f_{FP}(\hat{y}, w) = \sum_i \#(w_i \cap \hat{y}_i)$.
- False negatives: number of alternatives that a model predicted would be not chosen and were actually chosen. $f_{FN}(\hat{w}, y) = \sum_i \#(y_i \cap \hat{w}_i)$.

We can now see that substituting these formulas into the original formula for accuracy gives us the same score that we got by projecting each alternative as if it were a binary choice, calculating the accuracy for each one of them and then averaging the accuracy across the alternatives, however, this is much simpler, and the operation is likely to be also computationally more efficient. Table 4.2 shows how to what the y and w would be in our running example.

Working it out we can see that the total true positives are 3, the total true negatives are 3, the total false positives are 1 and the total false negatives are 2, inserting this into the accuracy metric we get $\frac{3+3}{3+1+3+2} = \frac{2}{3}$, same as when we mapped onto binary cases.

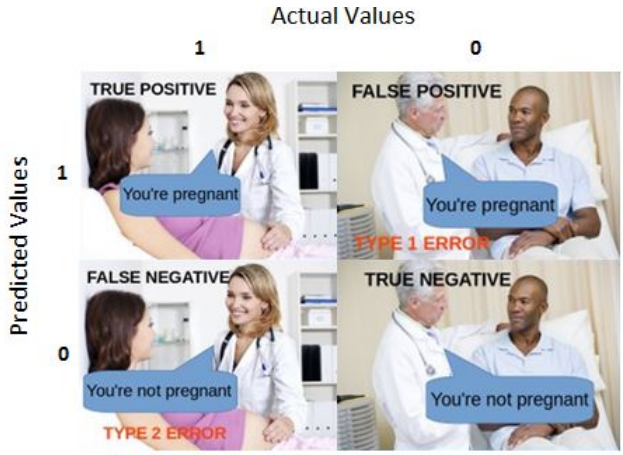
True / False negatives and True / False positives are usually captured in a **confusion matrix**. A confusion matrix can be quite useful for going further than binary comparisons.

Table 4.2: Recreating table 4.1 with examples for w and $\hat{w}$

y	$\hat{y}$	w	$\hat{w}$
$\{a, b\}$	$\{a\}$	$\{c\}$	$\{b, c\}$
$\{c, b\}$	$\{c, b\}$	$\{a\}$	$\{a\}$
$\{b\}$	$\{c\}$	$\{a, c\}$	$\{a, b\}$

The idea is that if the researcher has c different classes to predict then they create a $c \times c$ matrix where each row represents a predicted class and each column represents the true class. The numbers in the confusion matrix represent the times something was predicted v.s. what it actually was. The row-wise sum thus adds up to the total number of times that class was predicted, the column sum adds up to the number of times that class was observed and all the matrix's values add up to the number of predictions made. A popular example of a confusion matrix can be found in figure 4.1.

Figure 4.1: Popular example to show the confusion matrix in the binary case the values reduce to true positives, false positives, true negatives and false negatives (Narkhede, 2020)



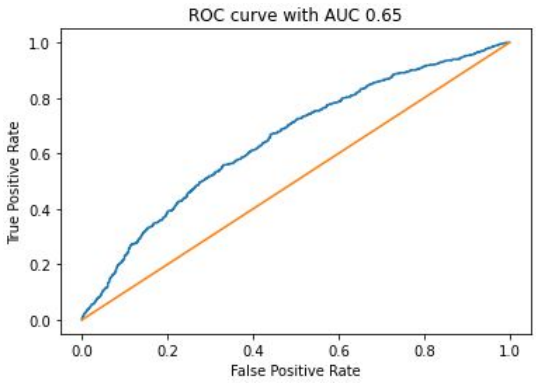
The Area Under the Receiver Operating Characteristic (ROC) Curve, often abbreviated to AUC is also a popular metric it is defined by the recall (also often referred to as the true positive rate) and the false positive rates. It is designed to evaluate probabilistic models by plotting recall and the false positive rate against each other. Where $y \in \{0, 1\}^n, \hat{y} \in \mathbb{R}^n_{\in[0,1]}$ and t is a threshold set by the researcher such that $t \in \mathbb{R}_{\in[0,1]}$ The probabilistic definition of these variables are:

- True positives: $f_{TP}(\hat{y}, y, t) = \frac{1}{n} \sum_{i=1}^{n} [[\hat{y}_i \geq t \cap y_i = 1]]$

- False negatives: $f_{FN}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i < t \cap y_i = 1]]$
- False positives: $f_{FP}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i \geq t \cap y_i = 0]]$
- True negatives: $f_{FP}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i < t \cap y_i = 0]]$
- Recall: $f_{RE}(\hat{y}, y, t) = \frac{f_{TP}(\hat{y},y,t)}{f_{TP}(\hat{y},y,t)+f_{FN}(\hat{y},y,t)}$
- False positive rate: $f_{RE}(\hat{y}, y, t) = \frac{f_{FP}(\hat{y},y,t)}{f_{FP}(\hat{y},y,t)+f_{TN}(\hat{y},y,t)}$

The AUC can be only calculated on probabilistic models, because it works by modifying the probability threshold t, which has no effect on non-probabilistic models. The ROC curve plots the recall $f_{RE}(\hat{y}, y, t)$ and false positive rate $f_{FP}(\hat{y}, y, t)$ for varying t. When t is set to 0, then both recall and false positive rates are 1, since everything is classified as a positive, so there will be no false negatives and no true negatives either, making the numerator and denominator of both equations the same. When it is set to 1 then both these metrics will become zero, because there will be no false positives and both metrics would have zero in the numerator. Usually in computer science application an artificial threshold is created that is beyond 1 to enable easy counting of the cases that are at 1 or below and to avoid trouble with the edge case of a prediction being exactly 1, for example in a popular Python package scikit-learn this is resolved by taking the highest prediction value and $+1$ is added to it see solution here (Pedregosa et al., 2011). The $f_{FP}(\hat{y}, y, t) = f_{RE}(\hat{y}, y, t)$ line between these two points represents completely random predictions. The area underneath the ROC curve is an accuracy measure often used. A summary of this metric can be found in Google (2020), whilst more details and mathematical analysis can be found in Faraggi and Reiser (2002) and Fawcett (2006). An example of what the ROC curve looks like can be found in figure 4.2.

Figure 4.2: Example of an Receiver Operating Characteristic curve



The ROC curve can also be used to improve the thresholds that separate classes, providing

a visual aid on the trade-off between the two accuracy metrics. Analytically, one could define the best threshold as the point that is closest to the (0,1) co-ordinate in the graph (top left corner), an example of this method is discussed in Hoo et al. (2017). If the points are sparse then one can use interpolation techniques to find the theoretical point on the curve that might be closest to the (0,1) co-ordinate, usually interpolations are linear, however other methods have also been proposed such as smoothing (Maxion and Roberts, 2004).

The most straightforward application of the ROC curve would be in pairwise comparisons, which can be framed as a binary classification task see for example table 2.11, replicated above in the confusion matrix discussion. If we learn the probability of team 1 winning we can plot the ROC curve to evaluate predictions.

However, the ROC curve need not be limited to only pairwise comparisons, there have been some suggestions by researchers to extend the ROC to multi-class classification analysis, such as evaluating ROC curves for each possible class, created weighted averages of ROC curves across each class (Fawcett, 2006) or calculating a volume under the curve measure using a multi-dimensional ROC curve (Ferri et al., 2003). Multi-class accuracy metrics can be translated directly to discrete choice predictions, and to subset choice predictions also if each element in the power-set of the available alternatives was treated as a discrete choice. In the work by Guo et al. (2018) there's an example of how to use the AUC.

4.1.3 Evaluation metrics for pairwise comparison models

All the binary classifier evaluation metrics can be applied to pairwise comparison models. For all the measures except cross entropy loss this is quite straightforward from their definitions. Király and Qian (2017) have used the cross-entropy loss for evaluation. To see how this works with pairwise comparisons, imagine that we have the true observations and a probabilistic prediction for them $P(\hat{Y}_i = \{A_{i,1} \succ A_{i,2}\})$ in table 4.3. The way to calculate cross entropy loss would be to convert Y_i into 1 if Alternative 1 is selected and 0 otherwise. If the probabilistic predictions are defined as $P(\hat{Y}_i = \{A_{i,1} \succ A_{i,2}\})$ they can be used as they are, if they're defined as $P(\hat{Y}_i = \{A_{i,2} \succ A_{i,1}\})$ then they should be converted to $1 - P(\hat{Y}_i = \{A_{i,2} \succ A_{i,1}\}) = P(\hat{Y}_i = \{A_{i,1} \succ A_{i,2}\})$. Table 4.3 will become table 4.4, and then we can use the cross entropy loss as per usual on the columns Y_i and $\hat{Y}_i$.

One can use common classifier evaluation metrics when working with pairwise comparisons, as the problem is easily translated to a binary classification, for example, Kang and Kim (2015) used the confusion matrix to evaluate predictions coming from a Bradley Terry model. This form of evaluation can be particularly useful when ties are also a possible result

Table 4.3: Pairwise comparison predictions

Alternative 1	**Alternative 2**	Y_i	$\hat{Y}_i$
A	B	$\{A \succ B\}$	$P(Y_1 = \{A \succ B\}) = 0.8$
C	D	$\{C \succ D\}$	$P(Y_2 = \{D \succ C\}) = 0.9$

Table 4.4: Pairwise comparison predictions converted to calculate cross entropy loss

Alternative 1	**Alternative 2**	Y_i	$\hat{Y}_i$
A	B	1	0.8
C	D	1	0.1

to observe, since then the problem is more analogous to a c class classification.

4.1.4 Evaluation metrics for discrete choice

In discrete choice by definition the task is to predict the one alternative that will be chosen from a set of alternatives. However, for this task, as the number of alternatives increase, metrics based on true negative and false negative counts get inflated. This is because in the worst case scenario for each decision the number of true negatives will be $\#A_i - 2$ and the false negatives will be 1, that is if the chosen object was by mistake classified as a negative. We can see from the equations in table 9.1 that as $\#A_i \to \infty$, $f_{FP} \to 0$ and $f_{Spec} \to 1$, making all measures that depend on these calculations less informative, pointing to the fact that non-probabilistic models might become hard to compare in a discrete choice setting where there are many different alternatives.

When trying to find evaluation metrics for discrete choice data it is best practice to follow methods suggested by researchers who have studied the evaluation of classification with imbalanced classes. Imbalanced classes in classification is referred to observations where there are many more cases of a certain class than others, for example in fraud detection, there are much less people committing fraud than law abiding citizens, therefore, if a model just predicted that everyone will be law abiding might have a 99.9% accuracy. Another more reasonable model might have a similarly high accuracy in which case it becomes very hard to compare them. In discrete choice, the more alternatives are presented per choice the more the imbalanced class problem is present for evaluation.

Works such as Bekkar et al. (2013) offer a variety of calculations that helps deal with imbalanced classes and therefore are also relevant for evaluating discrete choices where there are a large number of alternatives. They show that the F_1 score is not sensitive to class imbalances, and suggest further metrics such as the G-means measure (see table 9.1), which is the metric they have found most studies using.

For probabilistic models in discrete choice, the mean absolute error gives a much more interpretable result, which is equivalent to measuring the average likelihood error for each prediction. The root mean squared error and the cross entropy loss are also options that can be considered.

4.1.5 Evaluation metrics for subset choice

If in general the size of the subsets selected are much smaller than the number of potential alternatives available or they are a large proportion of the potential alternatives available, then an evaluation method that works on imbalanced classes might be the best option. When the subset selection is done in such a way that classes seem balanced then there is perhaps more choices available such as the ones described in section 4.1.2.

4.1.6 Rank based evaluation methods

Evaluating ranks at cut-off points

Ranking methods use evaluation techniques that are not directly linked to the ones discussed so far. A list of them can be found in table 4.5.

Ranking models are often evaluated at several cut-offs points of the ranks. That is partially to account for the fact that in many cases getting the top ranks right is more important than getting the bottom ranks right, so when evaluating ranking data it would be common to see breakdowns of the evaluation metric at top 5, 10, 20, 30 etcetera. Example of this way of evaluating can be found in Schäfer and Hüllermeier (2015).

For explaining the following metrics it would be useful to recall the property of our definition of the matrix representation of choices, Y that when left multiplied by a vector of ones, we can obtain the rank of the items, see equation 2.3, so

$$\mathbb{1}^T_{\#A_i}\hat{Y}_i = [\text{Predicted rank of } A_{i,1}, \text{Predicted rank of } A_{i,2}, ...].$$

We will use $R_i = \mathbb{1}^T_{\#A_i} Y_i$ to represent the observed rank for choice i as a way to save some notation and make equations clearer in this section. We will use $\hat{R}_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$ for predicted ranks. Also we'd like to issue a warning to not confuse R with $\mathcal{R}$ where the former is the rank vector as we just described and the latter is the domain of the relation matrices as described in table 2.6.

Correlation metrics

Hüllermeier et al. (2008) have discussed various ways to evaluate rankings with a focus on: Spearman correlation, Kendall's tau and voting theory.

Spearman correlation (Spearman, 1904) is defined as:

$$f_{Spearman}(R, \hat{R}) = \frac{\sum_i (R_i - \bar{R})(\hat{R}_i - \bar{\hat{R}})}{\sqrt{\sum_i (R_i - \bar{R})^2} \sqrt{\sum_i (\hat{R}_i - \bar{\hat{R}})^2}},$$

where $\bar{R} = \frac{\sum_i R_i}{\#R}$ is the mean. For evaluating two rankings coming from full order preference models, the Spearman correlation can be used like so $f_{Spearman}(R_i, \hat{R}_i)$, and for evaluating a series of rankings it is possible to take the mean Spearman correlation across the predictions to then have an evaluation metric across all predictions. The Spearman correlation works for evaluating full and partial orders alike and it can also be used for comparing full orders with partial orders.

Kendall's Tau is a metric that counts the number of concordant pairs in a ranking vs. the number of discordant pairs. A concordant pair is one where both rankings agree that one of the alternatives is ranked higher than the other, a discordant pair is the opposite where the two rankings disagree. The measure is a correlation metric, so it runs between -1 and 1, where 1 means that the two rankings agree completely and -1 means that the two rankings are perfect inverses of each other. Then this score is calculated across each decision, and it is finally averaged across all observations. Examples of rankings being evaluated using Kendall's Tau can be found in (Cheng et al., 2010a).

To understand Kendall's Tau, we first need to explain what concordant and discordant pairs are. Suppose that we have two different rankings for items $\{a, b, c\}$ being $B = [1, 2, 3]$ and $Z = [2, 3, 1]$, where the first element in the ranking ranks a, the second ranks b and the third ranks c, exactly like R_i does. A pair of observations between B and Z is said to be

concordant if $\text{sgn}(B_j - B_i) = \text{sgn}(Z_j - Z_i)$, where $j < i$ and $\text{sgn} : \mathbb{R} \to \{-1, 0, 1\}$ is a function where

$$\text{sgn}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{otherwise} \end{cases}.$$

When a pair of observations is not concordant then we call it discordant. With this definition using our example we would say that the pairs of B_1, B_2 and Z_1, Z_2 are concordant, B_1, B_3 and Z_1, Z_3 are discordant, and B_2, B_3 and Z_2, Z_3 are also discordant. Therefore, we would say that the vectors B and Z have one concordant and two discordant pairs.

Let the function $\text{conc} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{N}$ for some $d \in \mathbb{N}$ count the number of concordant pairs in two vectors be defined as

$$\text{conc}(Z, B) = \sum_{j<i} \begin{cases} 1, & \text{if } \text{sgn}(Z_j - Z_i)\text{sgn}(B_j - B_i) > 0 \\ 0, & \text{otherwise} \end{cases}.$$

Let's define a similar function that counts the number of discordant pairs $\text{disc} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{N}$ for some $d \in \mathbb{N}$ be defined as

$$\text{disc}(Z, B) = \sum_{j<i} \begin{cases} 1, & \text{if } \text{sgn}(Z_j - Z_i)\text{sgn}(B_j - B_i) < 0 \\ 0, & \text{otherwise} \end{cases}.$$

The evaluation metric can be expressed as:

$$f_{K\tau}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{2(\text{conc}(R_i, \hat{R}_i) - \text{disc}(R_i, \hat{R}_i))}{n(n-1)}$$

.

Kendall's Tau can also be though of as the interaction between two measures called Correctness f_{CR} (also known as the gamma rank correlation (Cheng et al., 2010b)) and Completeness f_{CP} (Adam et al., 2020).

$$f_{CR}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{conc}(R_i, \hat{R}_i) - \text{disc}(R_i, \hat{R}_i)}{\text{conc}(R_i, \hat{R}_i) + \text{disc}(R_i, \hat{R}_i)}$$

$$f_{CP}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{2(\text{conc}(R_i, \hat{R}_i) + \text{disc}(R_i, \hat{R}_i))}{n(n-1)}$$

which yields Kendall's tau being $f_{CR}(y, \hat{y}) f_{CP}(y, \hat{y})$.

Kendall's tau can also be extended to ties (Adler, 1957) and can be used like the Pearson correlation to compare partial orders to each other and full orders to partial orders. This variation is also known as Tau-b. For T_i being all the ties in R_i and $\hat{T}_i$ being all the ties in $\hat{R}_i$. The formula looks like this:

$$f_{K\tau}(Y, \hat{Y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{(\text{conc}(R_i, \hat{R}_i) - \text{disc}(R_i, \hat{R}_i))}{\sqrt{(0.5n(n-1) - T_i)(0.5n(n-1) - \hat{T}_i)}}$$

The Spearman correlation and Kendall's Tau produce very similar results, and can be used interchangeably. Kendall's Tau has been shown to often produce smaller coefficients of correlation and whilst more complicated to calculate it could produce more reliable confidence intervals as its distribution approaches normality faster than the Spearman correlation (Colwell and Gillett, 1982).

Normalized Discounted Cumulative Gain (NDCG)

For partial orders, Järvelin and Kekäläinen (2002) have examined ways in which to compare a ranking to a partial order, some of the measures in their document such as the **Normalized Discounted Cumulative Gain (NDCG)** has been used by researchers such as Liu et al. (2009). The way metrics such as NDCG work is that for each ranked item the ground truth is expressed over a scale of relevance. In the case of Netflix movies this would be the equivalent of comparing a full order ranking of movies with the 1-5 star rating from users. Let the vector of ratings a user gives to the alternatives in A_i be captured by $K_i^{(r)} \in \mathbb{Z}^{\#A_i}$, so $K_i^{(r)} = [\text{rating user gave to alternative in } A_{i,1}, \text{rating user gave to alternative in } A_{i,2}, ...]$. NDCG is usually calculated by looking at the ratings given to those alternatives that the algorithm has predicted up to the k^{th} rank. But notice that the elements in $K_i^{(r)}$ are not sorted by the rank in $\hat{R}_i$.

To define the NDCG function we need to first define the **Discounted Cumulative Gain (DCG)** function, which is the following for a vector of $k \in \mathbb{N}$ real numbers $V \in \mathbb{R}^{\geq k}$:

$$f_{DCG}(V, k) = \sum_{p=1}^{k} \frac{2^{V_p} - 1}{\log(1 + p)}.$$

Next we define a sorting function (sort) that takes a vector of unique natural numbers of $d \in \mathbb{N}$ elements whose minimum value will be 1 and its elements cover all the natural numbers between 1 and d, we will call this the arrangement vector. The sort function also takes another vector whose elements are to be rearranged that we will call the target vector. The output of the sort function is a new vector that is a rearrangement of the target vector such that the first position in the target vector will be placed in the value of the first position in the arrangement vector. For example, if the target vector is $[a, b, c]$ and the arrangement vector is $[3, 1, 2]$ then the output of $\mathrm{sort}([a, b, c], [3, 1, 2]) = [b, c, a]$. Using this sort function we can create the vector that results from $\mathrm{sort}(K_i^{(r)}, \hat{R}_i)$ which for its first position will contain the rating given by a decision maker / process for the alternative that was ranked first by a supervised ranking algorithm, for its second position it will contain the rating given by a decision maker / process for the alternative that was ranked second by the supervised ranking algorithm and so on.

Now if we wanted to calculate the DCG for an observation where there are four alternatives $A_i = \{a, b, c, d\}$ and the user ratings $K_i^{(r)} = [5, 5, 1, 3]$ and the ranking prediction is $\hat{R}_i = [3, 1, 4, 2]$, if the researcher defines $k = 3$, then:

$$f_{DCG}(\mathrm{sort}(K_i^{(r)}, \hat{R}_i), k) = \frac{2^5}{\log(2)} + \frac{2^3}{\log(3)} + \frac{2^5}{\log(4)}.$$

We can see that this method would give a higher result to a ranking that is closer to the user's rank, for example ranking alternative a in position 2 instead of position 3 would improve this metric because $\frac{2^5}{\log(3)} + \frac{2^3}{\log(4)} > \frac{2^3}{\log(3)} + \frac{2^5}{\log(4)}$.

The NDCG calculation is normalised where the normalisation term is such that for each observation the best possible permutation is given a score of 1 and the worst possible permutation a score of 0. In our example above, the best possible permutation at $k = 3$, that is the permutation that would give us the highest score on DCG, would be achieved by getting the 3 highest numbers from $K_i^{(r)}$ ordered by highest to lowest. We define the $\mathrm{sortdes}$ function that sorts a vector of real numbers from highest to lowest. Similarly, we would need to know the lowest possible score we can get from the DCG and for this we need a function $\mathrm{sortasc}$ that sorts a vector of real numbers from lowest to highest numbers. Running from our previous example $\mathrm{sortdes}([5, 5, 1, 3]) = [5, 5, 3, 1]$ and $\mathrm{sortasc}([5, 5, 1, 3]) = [1, 3, 5, 5]$. Therefore $f_{DCG}(\mathrm{sortdes}(K_i^{(r)}), k)$ would give the highest possible DCG at point k and $f_{DCG}(\mathrm{sortasc}(K_i^{(r)}), k)$ will give the lowest possible DCG at point k.

Now we can define NDCG as:

$$f_{NDCG}(\hat{Y}, K^{(r)}, k) = \frac{1}{n}\sum_{i=1}^{n}\frac{f_{DCG}(\text{sort}(K_i^{(r)}, \hat{R}_i), k) - f_{DCG}(\text{sortasc}(K_i^{(r)}), k)}{f_{DCG}(\text{sortdes}(K_i^{(r)}), k) - f_{DCG}(\text{sortasc}(K_i^{(r)}), k)},$$

.

In the table below we show some of the formulas we have discussed in this section as a recap.

Table 4.5: Commonly used evaluation metric for ranking

loss/accuracy function name	**equation**
Normalised discounted cumulative gain	$f_{NDCG}(\hat{R}, K^{(r)}, k) = \frac{1}{n}\sum_{i=1}^{n}\frac{f_{DCG}(\text{sort}(K_i^{(r)}, \hat{R}_i), k) - f_{DCG}(\text{sortasc}(K_i^{(r)}), k)}{f_{DCG}(\text{sortdes}(K_i^{(r)}), k) - f_{DCG}(\text{sortasc}(K_i^{(r)}), k)}$
Spearman correlation	$f_{Spearman}(R, \hat{R}) = \frac{\sum_i (R_i - \bar{R})(\hat{R}_i - \bar{\hat{R}})}{\sqrt{\sum_i (R_i - \bar{R})^2}\sqrt{\sum_i (\hat{R}_i - \bar{\hat{R}})^2}}$
Kendall's Tau correlation	$f_{K\tau}(R, \hat{R}) = \frac{1}{n}\sum_{i=1}^{n}\frac{2(\text{conc}(R_i, \hat{R}_i) - \text{disc}(R_i, \hat{R}_i))}{n(n-1)}$
Correctness	$f_{CR}(R, \hat{R}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{conc}(R_i, \hat{R}_i) - \text{disc}(R_i, \hat{R}_i)}{\text{conc}(R_i, \hat{R}_i) + \text{disc}(R_i, \hat{R}_i)}$
Completeness	$f_{CP}(R, \hat{R}) = \frac{1}{n}\sum_{i=1}^{n}\frac{2(\text{conc}(R_i, \hat{R}_i) + \text{disc}(R_i, \hat{R}_i))}{n(n-1)}$

4.1.7 Evaluation data

Consider the task of predicting basketball matches for the NCAA on the described data in 2.5.3. Suppose we have a Thurstone model and a Bradley-Terry model and want to know which is most accurate and we chose a loss metric fit for pairwise comparisons, say the F1 score. The next key point to mention in the machine learning pipeline is what data to use to measure the accuracy on. The purpose of the task would be to predict matches that haven't happened yet. Suppose we have matches played between 2003 and 2018 (as in 2.5.3). We can train the algorithms on the 2003 - 2015 matches and make them predict the 2016-2018 matches. This sort of out of sample prediction to then check which model performance is also called model validation.

We have discussed in section 3.4 that during algorithm fitting or training, models adjust their parameters to the data that has been provided to them, this data is also often referred to as training data. This means that measuring performance on the very data for which the prediction has been explicitly optimised can cause a false sense of accuracy. The effect of having a model which fits the training data very well, but doesn't predict correctly observations that have been unseen in the training data set is called *overfitting*, models that overfit are also described as models that do not *generalise* well. To avoid falsely believing that a model generalises well, the simplest correct strategy is to split the data the researcher has at hand into data that model is trained on, also known as the training data $T \subset D_V$ (recall notation for data in definition 4) and data that the performance of the model is evaluated on also known as validation or test sets $V \subset D_V$ such that $T \cap V = \emptyset$ and $T \cup V = D_V$. These concepts are well captured across many machine learning books, the writing here has been mainly informed by Bishop (2006) and Hastie et al. (2009). We will denote the ground truth values of these datasets as T_Y and V_Y respectively. Depending on the task the split might be done in two ways:

(i) There might be cases where we are trying to predict the decisions that will be made in the future based on decisions made in the past. Suppose that the data is

$$D = \{(A_1, S_1, B_1, X_1, G_{A_1}, C_{S_1}, Y_1,), \ldots, (A_N, S_N, B_N, X_N, G_{A_N}, C_{S_N}, Y_N)\},$$

where N is the number of observations, such that the observations are sorted in an increasing time scale where the Nth observation is the most recent one and the first observation is the most distant one. Furthermore, these could be bundled, for example in the cases of concurrent matches played (matches played on the same day) or round based competitions (matches may be played across different days but each player / team

only plays once in a round). To accommodate for such cases we specify a time-based vector Ξ which contains the first observation of each round, for example $\Xi = [1, 5, 20]$ would mean that the first round starts at observation one, the second at the fifth and the third at the twentieth. When the data has a temporal split we will in general split our training and test sets the following way:

$$T := \{(A_1, S_1, B_1, X_1, G_{A_1}, C_{S_1}), \ldots, (A_{k-1}, S_{k-1}, B_{k-1}, X_{k-1}, G_{A_{k-1}}, C_{S_{k-1}})\}$$

$$T_Y := \{Y_1, \ldots, Y_{k-1})\}$$

and

$$V := \{(A_k, S_k, B_k, X_k, G_{A_k}, C_{S_k}), \ldots, (A_N, S_N, B_N, X_N, G_{A_N}, C_{S_N})\}$$

$$V_Y := \{Y_k, \ldots, Y_N\}$$

where $k \in \Xi$ such that $\frac{k}{N} = \alpha$. Often k is set in a way that gets the researcher closest to $\alpha = 0.8$, that is 80% of the data is used as training and 20% as validation.

When splitting based on time, a key assumption is that the data is generated by the same process throughout the observations in D, that is, V and T have been created by the same underlying mechanism. When the process that generates the data changes over time, it is known as data drift, which when present these sorts of train/validation splits become unreliable (Hoens et al., 2012). Note that this is splitting the observations on the decision level, alternative level and decision maker / process level tables are not split in validation.

(ii) In case of predicting choices of independent choice makers one can uniformly sample the whole data without replacement to obtain:

$$T := \{(A_1, S_1, B_1, X_1, G_{A_1}, C_{S_1}), \ldots, (A_k, S_k, B_k, X_k, G_{A_k}, C_{S_k})\}$$

$$T_Y := \{Y_1, \ldots, Y_k\}$$

and

$$V := \{(A_{1^*}, S_{1^*}, B_{1^*}, X_{1^*}, G_{A_{1^*}}, C_{S_{1^*}}), \ldots, (A_{k^*}, S_{k^*}, B_{k^*}, X_{k^*}, G_{A_{k^*}}, C_{S_{k^*}})\}$$

$$V_Y := \{Y_{1^*}, \ldots, Y_{k^*}\}$$

s.t. $\frac{k}{k+k^*} = \alpha$ and $N = k + k^*$.

Making a training and validation split is good, however, the best practice is a technique called k-fold cross-validation (Stone, 1974). It is the method of partitioning the data into k chunks. Imagine that $k = 5$ then the training will happen on 4 chunks and predicting on the one that is out of sample and once these predictions have been made we can measure the accuracy on the holdout sample. Then the model is trained again holding out a different chunk, and so on 5 times until every chunk has been held out once. When the results are recorded this gives 5 different validation results, which can be very useful to examine the stability of the validation results. It can also be used to increase the number of validation points by pooling together all the holdout chunks and then measuring the accuracy on all of it, this way all of the data has been used to validate and to train but in such a way that the validation points were always held out.

There is a third split that needs to happen when reporting results called a test split, this is best described after explaining regularisation, so we will come back to this point in the next section.

4.2 Methods for mitigating overfitting risks that become more prominent in the era of big data

Sometime in the first decade of the 2000's the capacity to gather and store data has started growing rapidly, this is often referred to in the literature as big data. "5 exabytes (10^{18}) of data were created by humans until 2003. [In 2013] this amount of information is created in two days. In 2012, digital world of data was expanded to 2.72 zettabytes (10^{21})" (Sagiroglu and Sinanc, 2013). This brings many opportunities to define models with covariates that bear more relevance to the data, however it also brings new challenges, specifically for overfitting. In his book, Bishop (2006) shows an example of overfitting via too many variables. He generates 10 points for the model $y = sin(x)$ for the range $x \in \mathbb{R}_{[0,2\pi]}$ and tries to fit a linear regression by taking M polynomials of x. When the number of polynomials which in this case are covariates reach 9, the model fits perfectly on the curve since there are only 10 degrees of freedom, so the covariates can be trained to fit the points perfectly. However, it can be clearly seen that this model would perform terribly for another held out set of points.

4.2.1 Regularisation

Bishop also notes that as the number of covariates increased so did the magnitude of the parameters. Therefore to avoid having to decide which covariates to drop, the solution might be changing the loss function in such a way that large magnitudes for the covariates are penalised. The concept of regularisation is employed across almost all machine learning algorithms. To take as an example, in linear models there are two widely used ways to penalise large coefficient values, they are called L1 and L2 regularisation.

L1 regularisation also known as Lasso regression (Tibshirani, 1996, 2011) adds the sum of the absolute parameters to the loss function, which has the effect of setting some parameters completely to zero. For a parameter $\alpha \in \mathbb{R}$, chosen by the researcher

$$\ell(\lambda, \beta, \gamma, \kappa; \alpha)_{L_1} = \ell(\lambda, \beta, \gamma, \kappa) + \alpha \sum (|\beta| + |\gamma| + |\lambda| + |\kappa|)$$

.

The parameter α is called a hyperparameter. In general hyperparameters are parameters defined by the researcher before the model fitting begins, rather than a model parameter, which is something that is adjusted in the fitting process.

L2 regularisation is also known as Ridge regression (Hoerl and Kennard, 1970) adds the sum of the squares of the parameters into the loss function, which has an effect of shrinking the parameters towards zero but never setting it actually to zero.

$$\ell(\lambda, \beta, \gamma, \kappa; \alpha)_{L_2} = \ell(\lambda, \beta, \gamma, \kappa) + \alpha(\beta^T\beta + \gamma^T\gamma + \lambda^T\lambda + \kappa^T\kappa)$$

.

In a Bradley-Terry model without covariates L2 regularisation would look like the following (Chen et al., 2019; Maystre, 2019):

$$\ell(\lambda, \alpha)_{\text{Bradley-Terry L2}} = \ell(\lambda)_{\text{Bradley-Terry}} + \alpha\lambda^T\lambda$$

and L1 regularisation would look like the following:

$$\ell(\lambda, \alpha)_{\text{Bradley-Terry L1}} = \ell(\lambda)_{\text{Bradley-Terry}} + \alpha \sum_{a<b} |\lambda_a - \lambda_b|,$$

where $a, b \in \mathcal{A}$ and by $a < b$ we mean that each unique pair is summed only once. Further details about how to use Lasso in Bradley-Terry can be found in (Schauberger and Tutz, 2019).

4.2.2 Feature selection

We explained that regularisation is a way of mitigating overfitting. Another way is feature selection. Feature selection is the process of removing covariates from the model to further avoid overfitting. There are many methods for doing this in machine learning, we will discuss a couple of examples here, but this section is not meant to be an exhaustive exposition of approaches for feature selection.

As we mentioned in L1 regularisation some coefficients are set to zero by this method, which means that it inherently does feature selection. Another proposal was to train models using all possible subsets of the covariates and examine with cross-validation which subset performs best (John et al., 1994). These methods are also known as greedy or brute force methods in machine learning, meaning that all possibilities are tried and the best one is picked rather than finding the best method by some more sophisticated techniques that might get to the result faster. For example, if there are k covariates then this method would run the algorithm $2^k - 1$ times (-1 for assuming that a version with no covariates is not run).

A less greedy variant is recursive feature elimination (Kohavi et al., 1997; Guyon et al., 2002), where the idea is that the covariate that contributes the least amount to the scoring is dropped from the model, until a predefined top t number of alternatives are reached. These methods only run $k - t$ times.

4.3 Transformation

Transformation can be used to increase or reduce the complexity of the data. They can be used on covariates and the dependent variable. We show some examples on increasing the complexity. In most of machine learning transformation of the covariates is a well known and widely used technique, however, in preference modelling transforming the dependent variables, is also widely used and we will discuss this in detail in the next chapter. First, in section 4.3.1, we will give a short presentation about the different ways in which transforming covariates can be done. Note that this is not an exhaustive list of techniques for feature transformations, rather it is meant to mention some of the approaches that might be commonly encountered.

4.3.1 Transforming covariates

We will express transformers that are used on covariates as a function t is that is used to create a different expression of a series of numbers, $t : \mathbb{R}^{n\times j} \to \mathbb{R}^{n\times k}$ for some $n, k, j \in \mathbb{N}$.

Standardising is very important when regularising. Imagine if there are two covariates in the dataset one is measured in meters and the other in millimetres, imagine that in a linear model these covariates have the same impact. Then the parameter for one of these covariates would be a thousand times larger than the other, this can cause problems when regularising, since the process punishes large parameters. It is therefore important to have all the covariates on a comparable scale when using this technique.

One way to standardise a covariate is to scale it to a 0 mean and 1 standard deviation. Let x be a column of the matrix X and $n = \#x$, then $t_{\text{standardiser}} : \mathbb{R}^n \to \mathbb{R}^n$ and

$$t_{\text{standardiser},i}(x) = \frac{\left(x_i - \frac{\sum_i (x_i)}{n}\right)}{\sqrt{\frac{\sum_i \left(x_i - \frac{\sum_i (x_i)}{n}\right)^2}{n}}}$$

There are other types of transformations, such as **min-max scaling**, which can be used to stretch out differences of observations within a covariate of a small range. $t_{\text{min-max}} : \mathbb{R}^n \to \mathbb{R}^n_{[0,1]}$

$$t_{\text{min-max},i}(x) = \frac{x_i - \inf(x)}{\sup(x) - \inf(x)}$$

where $\inf$ is the infimum and $\sup$ is the supremum of the vector x. This type of transformation is very sensitive to outliers and therefore it should be considered when the covariate has a very low standard deviation and there are no outliers present. This type of transformation is also very sensitive to generalising, since it is driven by the most extreme values which might be very different in the training and test sets.

When there are outliers in the data the **robust-scaler** is also an option, which standardises based on the median and the interquartile range. $t_{\text{robust-scaler}} : \mathbb{R}^n \to \mathbb{R}^n$

$$t_{\text{robust-scaler},i}(x) = \frac{x_i - \text{median}(x)}{Q(x, 75) - Q(x, 25)}$$

where $Q(x, p) : \mathbb{R}^n \times \mathbb{R}_{[0,100]}$ is a function that returns the p percentile of the numbers observed in the vector x. The median and the inter-quartile range are less sensitive to outliers than the mean and the standard deviation, which is why this scaler might be best to use when

outliers are present and cannot be dropped or capped.

Transforming the data into a distribution that looks as close to a normal distribution as possible can improve model performance, as many models assume that features are normally distributed. In this case a transformer like Box-Cox (Box and Cox, 1964) is often used.

Polynomials When using generalised linear models, since the underlying relationship is linear it is a common technique to take polynomials to capture non-linearity. $t_{\text{polynomial}} : \mathbb{R}^n \to \mathbb{R}^{n \times p}$ for some $p \in \mathbb{N}$

$$t_{\text{polynomial}}(x, p) = [x, x^2, \ldots, x^p]$$

Where $x^p = [x_1^p, \ldots, x_n^p]$

Another popular technique is to take the log transformation of features to capture non-linearity.

These specific transformations mostly apply to linear models and is unnecessary in other techniques such as tree-based models (like random forests or gradient boosted models) or neural networks, which due to their structures can already capture non-linearities. Support vector machines can also have kernels which transform the data into a non-linear space, one of them is the polynomial kernel, though details about support vector machine kernels will not be covered in this thesis.

There are also transformations that decrease the number of variables such as taking the **principal components** of several covariates more details on how these work can be found in Wold et al. (1987).

4.4 Auto ML / Tuning / Searching

The idea behind tuning is to select the transformations, hyperparameters and features that minimise the loss function on a hold out set or on cross validation. The terms AutoML, Tuning and Searching are often used interchangeably for this process.

Auto ML and tuning was designed for finding the optimal hyperparameters. Suppose that our data consisted of inputs D which has domain $\mathcal{D}$, which we will not specify further here, but definition 4 can be used for further details, and ground truth values $Y \in \mathcal{R}_p$ for some property p. Let these be split up into training and validation sets such that $T \cup V = D$ and $V_Y \cup T_Y = Y$. Let the set of all model parameters be denoted by $\Psi \in \mathbb{R}^d$ for some $d \in \mathbb{N}$

and the set of all hyperparameters be H which take the domain $\mathcal{H}$ also not further specified here. Let a model M have a training process $M^{(f)} : \mathcal{D} \times \mathcal{R}_p \times \mathcal{H} \rightarrow \Psi$ and prediction process $M^{(p)} : \mathcal{D} \times \Psi \rightarrow \mathcal{R}_p$. For a loss function or accuracy metric f the optimal set of hyperparameters are defined by:

$$\hat{H} = \begin{cases} \text{argmin}_H \, f(V_Y, M^{(p)}(V, M^{(f)}(T, T_Y; H))) \text{ if using loss function} \\ \text{argmax}_H \, f(V_Y, M^{(p)}(V, M^{(f)}(T, T_Y; H))) \text{ if using accuracy metric} \end{cases}$$

There are many ways of searching through the space of all the potential combinations in H. The greediest method is called grid-search (originally called full factorial design), which looks through all options in a set of combinations provided by the researcher (Kempthorne, 1952). An alternative to grid-search is random-search (Bergstra and Bengio, 2012), which looks over random elements of the set of combinations for H. There are more targeted methods also such as Bayesian optimisation by Bergstra et al. (2013). A more exhaustive list of techniques can be found in Feurer and Hutter (2019).

The need for a test set when trialling different hyperparameters

We can now conclude the rationale behind the third evaluation set. When results on model accuracy is being reported, it is best practice to do so on a test set that has never been used for validation or cross validation. The idea is to take a chunk of the data (usually 20-30%) at the very beginning and only to use it when the final accuracy is being reported on the model. The argument behind it is essentially the same as the argument for not using the train-set to evaluate models. Since hyperparameters are not being learned by models, often researchers try different settings of them, usually using some form of Auto ML search, to see which one gives the best prediction on the validation set. That means that the validation set is being used by the researcher to learn the hyperparameter. Therefore, researchers cannot fully believe that results on the validation set are reflective of the models' actual performance since those are the results that were used to "learn" the hyperparameters. So best practice is to take a test set at the start that is not being used for validation at all and use it at the end to assess the accuracy of the models.

4.5 Composition of supervised preference models

"Model composition is combining separate models into an integrated composite model" (Petty and Weisel, 2019). We would like to split the possible modelling compositions into two different types of compositions. One type of composition is where the researcher uses several predictions for the same outcome stemming from several models, for example, having a Thurstone model and a Bradley-Terry model predicting the same pairwise comparison task and averaging their predictions. We call this type of composition *consensus composition* since the technique tries to reach a consensus for a prediction from several different algorithms expressing a different view for the same task.

The second technique is where the original task is split into different sub tasks and an algorithm is learned for each sub task and together the algorithms create a prediction for the original task. As an example, imagine that we want to predict a subset choice, but we do not know any models that can accomplish this task. One way of doing this would be to have one model predict the number of alternatives that will be selected by a decision maker and a second model that recursively predicts a discrete choice until the number of alternatives predicted by the first model is reached. We call this type of composition *anatomising composition* since the original task is deconstructed into several tasks which are then used together.

4.5.1 Anatomising composition

Since "human actions are naturally compositional" (Kato et al., 2018) it is likely that preferences generated by people such as items selected in a shopping basket is not the result of a single decision but the composition of several smaller decisions.

Consider the following two competing hypotheses for how people might shop the yoghurt category:

- **Hypothesis 1**: first a person decides how many different items they want to buy, then they iteratively purchase their favourite, followed by their second favourite and so on.
- **Hypothesis 2**: people in the store would consider all the possible combinations of products they can buy and then select the one that they like best.

It is possible to get some idea on which of these hypotheses works best by using anatomising composition. The process is to learn models that reflect both approaches and then see which one works best for predicting unseen data.

Learning a model for hypothesis one To express this more mathematically, suppose that we capture the number of different yoghurt a person has bought in shop i by $Y_i^{(q)} \in \mathbb{N}$ and the yoghurt purchased as $Y_i \in \mathcal{R}_{ch}$. Let $k \in \mathbb{N}$ and Q be a model that predicts number of alternatives likely to be purchased $Q : \mathcal{A}^k \times \mathcal{X} \times \mathcal{G} \times \mathcal{C} \rightarrow \mathbb{N}$ and B the scoring function of a Luce model with covariates such that $B : \mathcal{A}^k \times \mathcal{X} \times \mathcal{G} \times \mathcal{C} \rightarrow Distr(\mathcal{R}_{ch}(\mathcal{A}^k))$. Suppose that there is also a function T that takes the predictions of the Luce model, ranks the different alternatives and returns the top $t \in \mathbb{N}$ highest ranked alternatives. That is $T : Distr(\mathcal{R}_{ch}(\mathcal{A}^k)) \times t \rightarrow \mathcal{A}^t$. In a composition, the researcher trains two models Q and B and interacts them in function T such that the prediction of the alternative selected $\hat{Y_i}^{(subset)}$ becomes $\hat{Y_i}^{(subset)} = T(B(A_i, X_i, G_i, C_{S_i}), Q(A_i, X_i, G_i, C_{S_i}))$.

For the second hypothesis the researcher can fit a reduction to a Luce model where the alternatives are the powerset of the distinct alternatives and use this to predict the subset purchased by the shopper.

Anatomising composition allows us to gather further evidence supporting one of these hypotheses by executing both these strategies and checking on a validation set how well they can predict new observations.

Some of these types of compositions have been explored in hurdle models, which is a specific interaction of two models, one is to decide whether the decision maker will participate in making a decision (for example a shopper can choose not to purchase anything) and if they do decide to participate then they decide what action to take. Therefore overcoming two hurdles. It is possible to create hurdle models that only use one function therefore it is not necessary to go through the computational expense of running two models. Applications of hurdle models can be found in Burton and Rigby (2009) and von Haefen et al. (2005).

4.5.2 Consensus composition

Researchers can also make several models that have been trained independently of each other to predict the outcome of the same event. When the researcher has several predictions for the same event from different sources they need to find a way to reach a consensus prediction from the many predictions, which is why we call this type of approach consensus composition. The final prediction still is a composition of many models, but unlike in anatomising composition, where the prediction task is subdivided into smaller tasks and each model is then passing information to the next model, in consensus composition there are many models predicting outcomes for the same event and now an agreement needs to be reached from the many

predictions.

The most common method of consensus composition of models is to calculate the mean or modal prediction between the different models, however there are other more sophisticated techniques such as bootstrapping, bagging, stacking and bumping. These are well explained mathematically in Chapter 8 of Hastie et al. (2009), so in this thesis there will not be a full exposition of these topics but there will be some intuitive description.

Bootstrapping is the technique of sampling the available data with replacement such that the bootstrapped samples contain the same number of observations as the original dataset. So if the original dataset contains 50 observations then each bootstrapped dataset would contain 50 observations which were sampled from the original dataset with replacement. Then a model is fit on the bootstrapped samples. The idea is that by creating many bootstrapped datasets and then fitting a model on each of them, it is possible to observe some natural variation in the outputs of the models. It can be used to calculate confidence intervals on the parameters of a model, such as the λ vector in a Bradley-Terry model, by observing the different values for each parameter that has been learned. Bootstrapping can also be used to predict the same unseen sample by the models learned in the different bootstrapped samples to get a range of predictions for the unseen observation. Other applications of bootstrapping are bagging, which is a composition that uses the average prediction of models learned on bootstrapped sample as its final prediction.

> "Bumping uses bootstrap sampling to move randomly through model space. For problems where fitting method finds many local minima, bumping can help the method to avoid getting stuck in poor solutions. As in bagging, we draw bootstrap samples and fit a model to each. But rather than average the predictions, we choose the model estimated from a bootstrap sample that best fits the training data" (Hastie et al., 2009).

It is also frequently done that researchers fit several models on the same dataset, for example we can fit a Bradley-Terry and a Thurstone model on pairwise comparisons and then a third model is created from these two. The simplest way of doing so is by averaging their predictions. More complex ways of interacting the models can be done through stacking, where a third model is trained using the predictions of the Bradley-Terry and Thurstone models as the inputs and the ground truth is the same as it was for the original models, that is a new model that learns the best way of weighting the Bradley-Terry and Thurstone models for prediction. There is no limit to how many input models stacking can have. These predictions can also be layered such that several models are trained to weigh the initial models in a neural network-like architecture. For example, a linear model and a tree based model both trained using the outputs of the Bradley-Terry and Thurstone models and then finally another model that combines the predictions of the linear model and tree based model. This

is known as the StackNet approach (Michailidis, 2017).

4.6 Machine Learning pipelines

The process of doing some of these jointly together with a model: transformation, composition and feature selection is called a machine learning pipeline.

> "*Full model selection* (Escalante et al., 2009) was the first attempt to automatically build a complete ML pipeline by simultaneously selecting a preprocessing, feature selection and classification algorithm while tuning the hyperparameters of each method ... an ML pipeline ... is a sequential combination of various algorithms that transforms a feature vector ... into a target value." (Zöller and Huber, 2019).

That is, a machine learning pipeline behaves like a machine learning model itself, and it is also a type of composition, in fact the kind of model compositions we have described in section 4.5 can be included in the machine learning pipeline.

Though pipelines can get quite complex, their practicality is that once defined they behave exactly like any preference model would. When compared with figure 3.1 the inputs and outputs are the same. Therefore pipelines can be used just like models and can be evaluated just as easily as one model is being evaluated against another. More recently there have been efforts to automate the pipeline creation process such as Drori et al. (2018).

Chapter 5

Transforming tasks, reduction and aggregation techniques

In the previous chapter we have discussed how machine learning pipelines work for solving supervised preference tasks. So far, we have presented methods like discrete choice models as approaches that solve discrete choice tasks. However, in preference supervised learning, the domain of techniques used (e.g. all discrete choice models), does not have to be constrained or defined by the domain of the observed task (e.g. all discrete choice observations). For example, if a researcher observes a discrete choice, it does not mean that they have to use a discrete choice task to model it. It is possible to transform a discrete choice observation into multiple pairwise comparison observations and then train a pairwise comparison model on the newly transformed observations. In this chapter we will discuss how it is possible to transform one supervised preference task into a different one to enable researchers to be more flexible, accurate or efficient in their modelling approaches.

Preference relations can be expressed as special cases of each other. Consider a discrete choice where we observe person i making a choice between whether they will take a bicycle, car or a bus to work. If we observe that they chose a bus as a discrete choice from the set $\{$bicycle, car, bus$\}$ we can also say that person i has preferred the bus to a bicycle and also preferred the bus to a car. More generally, a discrete choice observation i where the set of available alternatives is A_i can be expressed as $\#A_i - 1$ pairwise comparisons where the chosen alternative is said to be preferred to all alternatives that weren't chosen. It could be tempting to augment the information available by a further $\binom{\#A_i - 1}{2}$ pairwise comparisons where the alternatives that have not been chosen are said to tie. For this case in our example

we would be also saying that whilst person i preferred a bus to a bicycle and a bus to a car they seemed indifferent between the bicycle and the car. This, however, would be incorrect as we might observe the same person preferring one alternative over the other if the choice to take a bus were removed.

We can see that this type of transformation on the dependent variable changes the modelling task itself, in the case of this example from a discrete choice to a pairwise comparison. This might be beneficial for the following reasons:

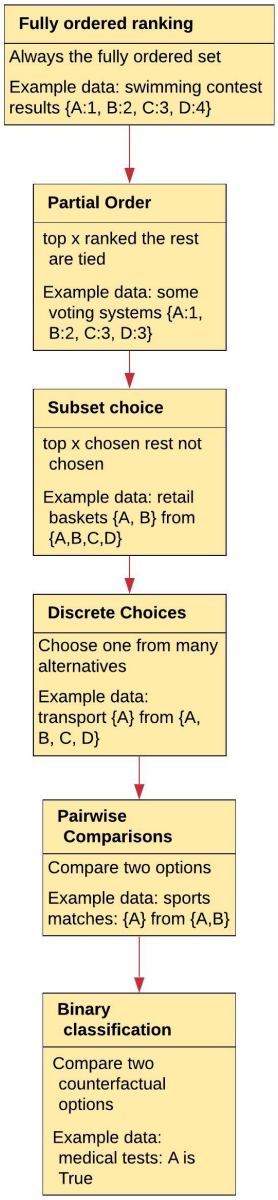
- The researcher believes that the observations are generated by a process that makes several decisions on a different task, for example even though the final observation is a discrete choice the process that generates those observations might be based on several pairwise comparisons.
- The researcher believes that models designed for solving other tasks are more robust, for example, there might be significantly fewer robust and well established subset choices models than pairwise comparison models, it might be beneficial to translate a subset choice task into a pairwise comparison task to have more variety of models to test by the researcher.
- There may be computational efficiencies in using transformed tasks.

Whichever case it is, the researcher should always validate their approach out of sample validation as we have outlined in the previous chapter.

These are transformations on the expressed relations and follow a hierarchy which we will present. When transforming a relation that is higher up in the hierarchy into a relation that is lower down the hierarchy then we say that we are creating reductions. The example we presented of expressing a discrete choice as pairwise comparisons is an example of a reduction. However, when the purpose of using preference models is to predict (rather than attempt to explain the process that generates the observations) the predictions made by a reduced task needs to then be transformed again into the original level. We call the transformation that moves up the hierarchy an aggregation. In section 5.1 we discuss ways to create reductions of expressed relations and in section 5.2 we discuss ways to aggregate relations.

5.1 Reduction of the dependent variable

Figure 5.1: The relation hierarchy



As we have seen in chapter 3 we found there was less research into models that are designed for subset choice than pairwise comparisons or discrete choice. In the case when a researcher needs to solve a task like subset choice, that in its original form has seen much less research conducted than another task say discrete choice the researcher faces two choices:

> "[t]he first is to solve it independently from other tasks. The second is to reduce it to a task that has already been thoroughly analyzed, automatically transferring existing theory and algorithms from well-understood tasks to new tasks" (Beygelzimer et al., 2005).

For example, a researcher could reduce a subset choice into a discrete choice problem, leveraging all the existing knowledge in discrete choice modelling, instead of relying on work done in the much less explored area of subset choice modelling. By doing this the researcher can "transfer existing theory and algorithms from well-understood tasks to new tasks" as per the quote above.

There is a philosophical similarity between anatomising composition and reduction and aggregation which is to transform the problem into a different one that has better defined models. The difference between anatomising composition and model reduction is that reduction refers to changing one task into a simpler task, whilst anatomising composition changes a task into several other tasks that can work together to estimate the original task, but the root of the idea is the same, the two can be used together but it doesn't necessarily has to be so.

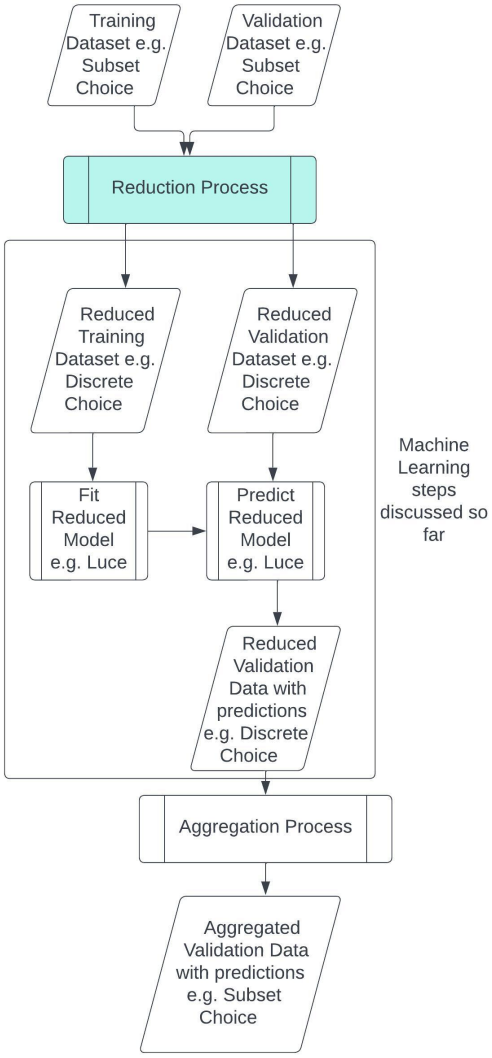
We present the relations that can be expressed on alternatives in the form of a hierarchy discussing in detail how they relate to each other and how reductions and aggregations can be made between them. The relations expressed over alternatives, and the modelling tasks follow a hierarchy where lower levels are special cases of the upper levels. We will refer to relations higher in this hierarchy as the parent and those lower

as the child. Note that what qualifies a relation to be higher in the hierarchy is not its mathematical family, but the amount of information it contains. Normally, mathematically, full orders would be a special case of partial orders. However, in this case, consider a full order of 10 alternatives; it would be very easy to convert a full order into a partial order if we knew the cut off points where alternatives should tie. For example, we could say we're interested in the top 5 alternatives ranked and then everything else can tie after them. Knowing a full order it would be very easy to translate it into such a partial order. However, if we are given the partial order translation and now asked to recover the full order from this, it would be much more difficult to do. Since the hierarchy we present is focused on transforming the relations observed, in this case we put the full order on the top of the hierarchy because it can be easily converted into any of the other relations that we have discussed in this thesis, however, from the other relations there is no way to recover it by only looking at the expressed relation itself.

At the top of the hierarchy there is a full order or full rank. Partial order is a special case of full ranks, where alternatives can tie. A subset choice is a special case of partial order where there are only two ranks. Discrete choice is a special case of subset choice where there is only one top ranked alternative and everything else ties in the second rank. Pairwise comparisons are a special case of discrete choice where there is only one alternative that has the second rank (and one alternative that has the first rank). Finally, binary classification is a special case of pairwise comparisons where only the same two alternatives are available for every decision. This is visualised in the hierarchy in figure 5.1. Each level can be expressed in the form of one of its child nodes by going down in the hierarchy, and so models of a lower level can be used to solve problems of a higher level, for example, discrete choice model types can be used for subset choice tasks. We will call *native models* where the model types and the model task are on the same level in the hierarchy such as using a Plackett-Luce model for fully ranked data. We will now show how each of the subsequent levels in the hierarchy are a special cases of their parent relations and how to convert parent relations into child relations.

In diagram 5.2 we show a work flow of using reduction and aggregation. In this section we focus on reduction processes.

Figure 5.2: A flow diagram for a preference model task with reduction and aggregation. The researcher starts with data that is split into training and validation on the LHS of the diagram. Suppose that the problem is a subset choice task, and the researcher cannot find any subset choice models that they are able to implement with the tools they have at their disposal, but they are able to solve discrete choice problems. Then the researcher would like to transform the subset choice problem into a discrete choice problem via a reduction method. The reduction method will take the subset choice problem as its input and will return a reduced representation e.g. a discrete choice in this example as the output. Then the researcher can learn a discrete choice model using the output of the reduction process. The discrete choice model would create prediction that are also discrete choice, however, the researcher needs the prediction to be on the subset choice level. The aggregation process solves the problem of how to represent the reduced prediction in the original subset choice level. In this section we discuss the reduction process (highlighted in the figure), in the next section we discuss the aggregation process.



5.1.1 Converting Full orders into reduced representations

As mentioned at the top of the relation hierarchy is the full rank. An example of a natural domain in which we can observe this task are certain sports competitions where each encounter creates a full order of the participants such as running, cycling, or car racing. In this section we explore how to convert full orders into representations that are lower in the relation hierarchy.

Converting full order into partial order

Consider the full order of Formula 1 racers of a particular race day in section 2.3.1. It would be possible to directly transform these observations into a specific case of partial order, for example, if the task was to identify the top three in order, this would be possible from the fully ordered list, and then everyone else can be represented as a tie in the same rank that is lower than three. We wouldn't have to do anything special when we know the full order to show these as a partial order if we are given the description of the partial order of interest. Table 5.1 is an adaptation from table 2.7 where the results are converted to a partial order where the top two finishers are identified and the bottom one also, perhaps in some sports might be candidates for promotion and relegation. Now it would be possible to consider using a partial order model on the column called *conversion to partial order*.

Table 5.1: Converting full orders into partial orders

race id	finishing position	drivers	conversion to partial order
1	[3, 2, 4, 1]	{a, b, c, d}	[2, 1, 3, 1]
2	[4, 2, 3, 1]	{a, b, c, d}	[3, 1, 2, 1]
3	[3, 2, 1]	{a, b, c}	[2, 1, 1]
4	[4, 3, 2, 1]	{a, b, c, d}	[3, 2, 1, 1]

Let's consider trying to do this the other way around. If we were given a partial order, e.g. the top three finishers of the race ranked and then every other racer at rank four we could not directly create a full order from this information. This is the reason why partial orders are a reduction of full orders, full orders contain more information such that it is possible to create partial orders from full orders but not the other way round. When we try to go the other way we need special rules and techniques to do so which are in the realms of aggregation. We will discuss aggregation in the next section.

The algorithm for converting full order into partial order is:

Algorithm 1: Converting full orders into partial orders: we start with the inputs of a vector of natural numbers which define the cut off points, for example $[1, 2, 3, 5]$. The algorithm assigns each rank between two cut off points as the lower rank. For example with the above defined cut off points, if the input rank is $[1, 2, 3, 4, 5]$ then the output would be $[1, 2, 3, 4, 4]$. Notice in this case the input cut off points jump from 3 to 5 indicating that everything between the ranks 3 and 5 should be given the rank 4.

Inputs:

- For a number of $d \in \mathbb{N}$ cutoff points a vector of $P \in \mathbb{N}^d$, such that $P_i \neq P_j \forall i, j$
- a full order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$ (recall that when we left multiply Y_i with a vector of ones we get the rank of the full order, see equation 2.3)

Outputs:

- A partial order $K_i \in \mathbb{N}^{\#A_i}$

Algorithm described by pseudo-code:

$K_i \leftarrow \mathbf{0}^{\#A_i}$ where $\mathbf{0}^j$ is a vector of j zeros
$p^{(-1)} \leftarrow 0$
for $p \in P$ **do**
  $l \leftarrow 1$
  $r \leftarrow \inf(\{j : j \in R_i; p^{(-1)} < j \leq p\})$
  for $y \in R_i$ **do**
    if $y \leq p$ *and* $K_i[l] = 0$ **then**
      $K_i[l] \leftarrow r$
    end
    $l \leftarrow l + 1$
  end
  $p^{(-1)} \leftarrow p$
end
return K_i

Converting full order into subset choices

Converting full order into a subset choice, is simpler since only a cutoff rank is necessary. Everything that is in a better rank than the cutoff is assumed to be chosen, everything that

is in a worse rank is assumed to be not chosen.

Algorithm 2: Converting full orders or partial orders into subset choice: For a cut off point in the ranking e.g. 3 everything that has a rank of less than or equal to 3 is assumed to be chosen in the subset. For example, if the full ranks are $[5, 2, 4, 3, 1]$ representing alternatives $\{a, b, c, d, e\}$ and the cut off point is 3, then we the subset choice would be $\{e, b, d\}$.

Inputs:

- a full order or partial order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$
- a cutoff point $p \in \mathbb{N}$
- a list of alternatives A_i

Outputs:

- A subset of alternatives $K_i \subseteq A_i$ that have been chosen.

Algorithm described by pseudo-code:

$K_i \leftarrow \emptyset$
$l \leftarrow 1$
for $y \in R_i$ **do**
  if $y \leq p$ **then**
    $K_i \leftarrow K_i \cup A_i[l]$
  end
  $l \leftarrow l + 1$
end
return $K_i \setminus \emptyset$

Converting full order into discrete choice

Discrete choices are simple enough that we won't describe it with pseudo code. To convert a full rank into a discrete choice the process is to take the top ranked alternative.

Converting full order into pairwise comparisons

There are two approaches that can be used for converting full orders into pairwise comparisons. One is what we will call the *full conversion approach* another is what we will call the *next down approach*. The next down approach expresses the full order into a vector of transitive

pairwise preferences. For example for the first observation in table 5.1 the conversion would be three observations $[\{D \succ B\}, \{B \succ A\}, \{A \succ C\}]$, whereas the full conversion approach makes a pairwise comparison of everything against everything else and would result in six observations $[\{D \succ B\}, \{D \succ A\}, \{D \succ C\}, \{B \succ A\}, \{B \succ C\}, \{A \succ C\}]$. The full version might be less desirable because it might end up creating too many observations and processing times might slow down, on the other hand it could also capture more information and enable more accurate models to be learned. In total for each fully ranked observation Y_i it would generate $\frac{\#A_i(\#A_i-1)}{2}$ observations, whereas the next down approach full only create $\#A_i - 1$ observations for each full rank.

Algorithm 3: Converting full orders into pairwise comparisons using the next down approach: starting with a full rank, it starts with the highest ranked alternative and creates an observation saying that this alternative is preferred to the next highest ranked alternative, then jumps to the second highest ranked alternative and creates an observation that says that the second highest ranked alternative is preferred to the third, then the third to the fourth and so on. For example, if the alternatives $\{a, b, c\}$ were ranked $[1, 3, 2]$ then the output would be $K_i = [\{a \succ c\}, \{c \succ b\}]$.

Inputs:

- a full order or partial order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$
- a list of alternatives A_i

Outputs:

- K_i a vector of $\#A_i - 1$ pairwise comparisons

Algorithm described by pseudo-code:

$K_i \leftarrow \emptyset$
$s \leftarrow argsort(R_i)$
$A_i^{(r)} \leftarrow A_i[s]$
for $l = \mathbb{N}_{[1,\#A_i-1]}$ **do**
  $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \succ A_i^{(r)}[l+1]\}$
end
return $K_i \setminus \emptyset$

Algorithm 4: Converting full orders into pairwise comparisons using the full conversion approach: starting with the most preferred alternative, create an observation that states that it is preferred to every other alternative that has been ranked worse. Do this for the second most preferred alternative and so on. For example when the alternatives are $\{a, b, c\}$ and the ranking is $[1, 3, 2]$ then $K_i = [\{a \succ c\}, \{a \succ b\}, \{b \succ c\}]$.

Inputs:

- a full order or partial order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$
- a list of alternatives A_i

Outputs:

- K_i a vector of $\frac{\#A_i(\#A_i-1)}{2}$ pairwise comparisons

Algorithm described by pseudo-code:

$K_i \leftarrow \emptyset$
$s \leftarrow argsort(R_i)$
$A_i^{(r)} \leftarrow A_i[s]$
for $l = \mathbb{N}_{[1,\#A_i-1]}$ **do**
  for $j \in A_i^{(r)}[l+1:]$ **do**
    $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \succ j\}$
  end
end
return $K_i \setminus \emptyset$

Modelling full orders by continuous models

It is also possible to reduce fully ranked models in a way that they are fit by a continuous model, such as linear regression. In continuous models the dependent variable is a member of the real $Y_i \in \mathbb{R}$. Since in fully ranked models each alternative contains a rank $\mathbb{1}^T Y_i \in \mathbb{N}^d$ for some $d \in \mathbb{N}$, the rank itself can be expressed as the dependent variable of a continuous model. For each observation this would generate $\#A_i$ observations, since each rank will now be a row.

5.1.2 Converting partial orders into reduced representations

Converting partial order into subset choice

In a subset selection it is possible to select more than one of the alternatives as preferred to the other not selected alternatives, recall the example in section 2.3.1, from all the yoghurt in a store only a subset, say for example, products A and B are purchased by a customer, it is not shown whether A is preferred to B, but it is understood that both A and B are preferred to anything else that was available as an alternative. Subset selection is a special case of partial ranking where the top x alternatives tie but are preferred to all other alternatives, which also tie. Essentially, it is the same as if we took a rank cut off point in the partial order and asked, which alternatives were ranked above this position and which were below. Table 5.2 is and adaptation from table 2.8 where we show how a partial order might be converted into a subset choice. Suppose that in this case we would like to create a subset of movies for which the user has created top ratings, so the cut off point would be the first rank, anything that hasn't been ranked first, is not selected in the subset.

Table 5.2: Example of converting a partial rank into a subset selection

person id	**movie ratings**
1	{Star Trek Discovery: 5, Star Wars Return of the Jedi: 5, Travels with my father: 5}
2	{Star Trek Discovery: 1, Star Wars Return of the Jedi: 2, Travels with my father: 5}
3	{Star Trek Discovery: 5, Star Wars Return of the Jedi: 5, Travels with my father: 4}
person id	**movie ratings converted in subset choice**
1	{Star Trek Discovery, Star Wars Return of the Jedi, Travels with my father}
2	{Travels with my father}
3	{Star Trek Discovery, Star Wars Return of the Jedi}

To save space on paper, the alternatives are not displayed in table 5.2 however, they would be {Star Trek Discovery, Star Wars Return of the Jedi, Travels with my father} for each observation. The algorithm for making this conversion is essentially the same as algorithm 2.

Converting partial order into discrete choice

For converting discrete choice, the approach is similar as in the full order, except there might be additional tie breaking rules when there is a joint first or the observations can be duplicated. For example, in the third decision in table 2.8 a discrete choice could be expressed as two observations, the first where *Star Trek Discovery* is chosen from {*Star Trek Discovery, Travels with my father*} and the second where *Star Wars Return of the Jedi* is chosen from {*Star*

Wars Return of the Jedi, Travels with my father}.

Converting partial order into pairwise comparisons

For converting into pairwise comparisons the full order algorithm needs to be modified as it now needs to account for the expression of indifference also, but the same two approaches next down approach and full conversion approach exist.

Algorithm 5: Converting partial orders into pairwise comparisons using the next down approach: order alternatives by rank and then if the rank between the first and second alternative is the same then they are stored as indifferent, if the first alternative is higher ranked then say it is preferred to the second then do the same between the second and the third alternatives, third and fourth and so on. For example, for a list of alternatives $\{a, b, c\}$ and ranking $[1, 2, 1]$ this reduction would yield $K_i = [\{a \sim c\}, \{c \succ b\}]$

Inputs:

- a partial order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$
- a list of alternatives A_i

Outputs:

- K_i a vector of $\#A_i - 1$ pairwise comparisons

Algorithm described by pseudo-code:

$K_i \leftarrow \emptyset$
$s \leftarrow argsort(R_i)$
$A_i^{(r)} \leftarrow A_i[s]$
$R_i^{(r)} \leftarrow R_i[s]$
for $l = \mathbb{N}_{[1,\#A_i-1]}$ **do**
  if $R_i^{(r)}[l] = R_i^{(r)}[l+1]$ **then**
    $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \sim A_i^{(r)}[l+1]\}$
  else
    $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \succ A_i^{(r)}[l+1]\}$
  end
end
return $K_i \setminus \emptyset$

Algorithm 6: Converting partial orders into pairwise comparisons using the full conversion approach: starting with the most preferred alternative, create an observation that states that it is preferred to every other alternative that has been ranked worse and is indifferent to every other alternative that has been ranked the same. Do this for the second highest ranked alternative and so on. For example, for a list of alternatives $\{a, b, c\}$ and ranking $[1, 2, 1]$ this reduction would yield $K_i = [\{a \sim c\}, \{a \succ b\}, \{c \succ b\}]$.

Inputs:

- a full order or partial order $R_i = \mathbb{1}^T_{\#A_i} \hat{Y}_i$
- a list of alternatives A_i

Outputs:

- K_i a vector of $\frac{\#A_i(\#A_i-1)}{2}$ pairwise comparisons

Algorithm described by pseudo-code:

$K_i \leftarrow \emptyset$
$s \leftarrow argsort(R_i)$
$A_i^{(r)} \leftarrow A_i[s]$
$R_i^{(r)} \leftarrow R_i[s]$
for $l = \mathbb{N}_{[1,\#A_i-1]}$ **do**
  $h \leftarrow 1$
  for $j \in A_i^{(r)}[l+1:]$ **do**
    if $R_i^{(r)}[l] = R_i^{(r)}[l+h]$ **then**
      $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \sim j\}$
    else
      $K_i \leftarrow K_i \cup \{A_i^{(r)}[l] \succ j\}$
    end
    $h \leftarrow h + 1$
  end
end
return $K_i \setminus \emptyset$

Reducing partial orders into continuous models

It is possible to convert partial orders into continuous models the same way as one might do with full orders.

5.1.3 Converting subset choices into reduced representations

Converting subset choice to discrete choice

A discrete choice is a special case of subset selection where the deciding process only chooses one alternative from the set of alternatives, which is a special case of subset choice selection, but instead of top x, the deciding process selects the top 1. We could also imagine how subset selections such as the items in stores in section 2.3.1 could be expressed as a discrete choice. One popular way would be to express the alternatives for the discrete choice not as the set of unique alternatives, but as the power set of the unique alternatives. This would treat all possible combinations as a discrete alternative, including the empty-set, which in this case would translate to not purchasing anything. For example, if the alternatives in subset selection are $\{a, b, c\}$ then to convert this into a discrete choice, the set of alternatives are transformed into $\{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$, where the combinations of alternatives are treated a different alternative in itself, i.e. now $\{a, c\}$ is treated as if it were an alternative completely independent of alternatives a and b. In many cases this can "blow up" the dataset with too many possible observations and makes model fitting a longer and computationally more expensive process, however, in some cases where the number of alternatives is not too many this could be useful (Train, 2009). In table 5.3 we convert table 2.9 into a discrete choice using the power set approach. Note that in this case we are not transforming the result Y_i, instead we are transforming the list of alternatives available A_i to $A_i^{(t)} = \mathcal{P}(A_i)$. So in the case of a Luce model what we are actually increasing is the number of alternative level parameters λ to be estimated. The transformation yields for each $\#A_i^{(t)} = 2^{\#A_i}$.

In the case where the alternatives are transformed into the power-set of alternatives, the researcher should also consider what to do with the alternative level covariates. These would need to be defined for each alternative, and it's possible that the alternative level table would have to be transformed also such that each combination in power-set is included. For example, in the case where shoppers are browsing items in stores, the price of the subset could be defined as the sum of the prices of the individual products, but for the pack size perhaps it would make more sense to include the mean pack size of the items in the subset rather than the sum.

The second method for coverting subset choice into discrete choice modifies both the set of alternatives and the observed subset selection. First of all, it removes from the set of alternatives everything that has not been chosen, and for the observed selection it loops through the individual alternatives and frame it as a discrete choice that has been selected from a set of alternatives not chosen, so for each row of observation we get several. The

Table 5.3: Converting subset selection into discrete choice by transforming the available alternatives into a power-set

customer id	items purchased	items in store	alternatives available transformed
1	{a, b, c}	{a, b, c}	{∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c},{a, b, c}}
2	{a, c}	{a, b, c}	{∅, {a}, {b}, {c}, {a,b}, {a,c}, {b,c},{a, b, c}}
3	{a}	{a, d}	{∅, {a}, {d}, {a,d}}
4	{d}	{b, d}	{∅, {b}, {d}, {b,d}}

downside is that observations where decision makers chose everything available gets lost in this set up when choosing nothing is not an alternative. We will refer to this method as the looped purchase method, we show this transformation in table 5.4.

Table 5.4: Expressing subset choices as discrete choices by the looped purchase method, $A_i^{(t)}$ $Y_i^{(t)}$ are the transformed presented alternatives and relations expressed respectively. We can see that what was formerly the observation for customer number 2 we now have two choices made, where in reality there was just one. However, we cannot express the observation of customer 1 in this case, as they have chosen everything.

customer id	items purchased	items in store	$A_i^{(t)}$	$Y_i^{(t)}$
2	{a, c}	{a, b, c}	{a, b}	{a}
2	{a, c}	{a, b, c}	{b, c}	{c}
3	{a}	{a, d}	{a, d}	{a}
4	{d}	{b, d}	{b, d}	{d}

Algorithm 7: Converting subset selection into discrete choice via the loop method: Remove all chosen alternatives from the set of available alternatives, then create observations that state that each chosen alternative was chosen from a set of alternatives that contained itself and all the not chosen alternatives. For example, if the original set of alternatives is $\{a, b, c, d\}$ and the subset choice was $\{a, b\}$ then the following observations would be created: a chosen from $\{a, c, d\}$, b chosen from $\{b, c, d\}$.

Inputs:

- a subset choice $K_i \subseteq A_i$
- a list of alternatives A_i

Outputs:

- tuples of alternatives available and a single alternative that has been chosen

Algorithm described by pseudo-code:

if $A_i = K_i$ **then**
  skip
else
  for $k \in K_i$ **do**
    $A_i^{(t)} \leftarrow A_i \setminus K_i$
    $A_i^{(t)} \leftarrow A_i^{(t)} \cup \{k\}$
    $K_i^{(t)} \leftarrow \{k\}$
    return $(A_i^{(t)}, K_i^{(t)})$
  end
end

Converting subset choice to pairwise comparisons

Converting a subset choice into a pairwise comparison could be done with ties or without ties, though when ties are incorporate the researcher needs to make the assumption that the decision maker / process would have no preference between the not-chosen alternatives and would also have no preference between the chosen alternatives, this assumption would be very strong most of the times. If the researcher does not want to use a model that incorporates ties then the solution is to loop through each alternative that has been chosen and saying that it is preferred to each alternative that has not been chosen.

Algorithm 8: Converting subset selection into pairwise comparisons: creates pairwise comparisons to state that every alternative that has been chosen in a subset is preferred to every other alternative that has not been chosen, optionally it also creates further observations to state that every alternative that has been chosen ties and every alternative that has not been chosen also ties. For example, if the set of alternatives are $\{a, b, c\}$ and the subset choice is $\{a, b\}$ then the outputs would be the following $(\{a, c\}, \{a \succ c\}), (\{b, c\}, \{b \succ c\})$.

Inputs:

- a subset choice $K_i \subseteq A_i$
- a list of alternatives A_i
- an indication whether ties should be generated or not

Outputs:

- tuples of a set of two alternatives and a relation between those two alternatives

Algorithm described by pseudo-code:

$j_{all} \leftarrow \emptyset$
$k_all \leftarrow \emptyset$
if $K_i = A_i$ **then**
  skip
else
  $A_i^{(t)} \leftarrow A_i \setminus K_i$
  for $k \in K_i$ **do**
    for $j \in A_i^{(t)}$ **do**
      `// everything that has been chosen is preferred to everything that has not been chosen`
      $A_i^{(t2)} \leftarrow \{j, k\}$
      $K_i^{(t)} \leftarrow \{k \succ j\}$
      return $(A_i^{(t2)}, K_i^{(t)})$
    end
    if *run with ties* **then**
      $k_{all} \leftarrow k_{all} \cup k$
      for $z \in K_i \setminus k_{all}$ **do**
        `// everything that has been chosen is indifferent to everything else that has been chosen`
        $A_i^{(t2)} \leftarrow \{k, z\}$
        $K_i^{(t)} \leftarrow \{k \sim z\}$
        return $(A_i^{(t2)}, K_i^{(t)})$
      end
    end
  end
  if *run with ties* **then**
    for $j \in A_i^{(t)}$ **do**
      $j_{all} \leftarrow j_{all} \cup j$
      for $h \in A_i^{(t)} \setminus j_{all}$ **do**
        `// everything that has not been chosen is indifferent to everything else that has not been chosen`
        $A_i^{(t2)} \leftarrow \{j, h\}$
        $K_i^{(t)} \leftarrow \{j \sim h\}$
        return $(A_i^{(t2)}, K_i^{(t)})$
      end
    end
  end
end

Reducing subset choices into binary classification problems

Subset choices can also be converted into binary classification problems. The supervised binary classification task is one where the learning function maps to the set $\{0, 1\}$. For subset choices it is possible to transform the data such that there is a row for each alternative available and Y_i becomes 1 when the alternative gets selected and 0 otherwise. Table 5.5 shows an example on what this might look like.

Table 5.5: Example of shopping observations transformed for fitting a classifier

customer id	items purchased	items in store	temperature at purchase time (C)
1	1	A	22
1	1	B	22
1	1	C	22
2	1	A	15
2	0	B	15
2	1	C	15
3	1	A	24
3	0	D	24
4	0	B	10
4	1	D	10

5.1.4 Converting discrete choices into reduced representations

Converting discrete choices into pairwise comparisons

The method for converting discrete choices into pairwise comparisons is the same as described in algorithm 8 and they can also be converted into binary classifiers the same way as subset choices also.

Converting discrete choices into multiclass classifications

In multiclass classification, the dependent variable is a member of a fixed list of alternatives $Y_i \in \mathcal{A}$, which makes it very similar to discrete choice. For discrete choices it is also possible to use a multiclass classifier, where the whole set of alternatives $\mathcal{A}$ is treated as the available classes for the multiclass classifier. Generally, the more classes there are the slower the

multiclass task runs. There might be cases where there are too many unique alternatives in the universe make reduction to multiclass infeasible, for example in the cases of retailers that have tens of thousands of products sold nationwide.

5.1.5 Converting pairwise comparisons into reduced representation

Finally for pairwise comparison there is only one level to which it can be reduced, which is binary classification. The supervised binary classification task is one where the learning function maps to the set $\{0, 1\}$. The interesting thing about reducing pairwise comparisons to binary classification is that it can be taken directly without needing to convert the data. All that needs to be fixed is the structure. There can be columns that refer to alternative 1 and alternative 2 and the dependent variable needs to be converted to 1 if alternative 1 is chosen and 0 if alternative 2 is chosen.

As an example, consider the pairwise comparison problem in section 2.3.1. These can also be expressed as a binary classification problem, where the dependent variable is the *team 1 won (1/0)* column from table 2.11. An example of a reduction would be instead of training a pairwise comparison model on this data, training a logistic regression that predicts whether *team 1* will win or not, using some features such as the difference of *mean points scored in season* between *team 1* and *team 2*, which is an input available in the team level table. Such a binary model would be a simplification because it would learn a relationship where for any case if the difference in mean points scored in the season is the same, then *team 1* always has the same probability of beating *team 2* **regardless which team team 1 is**. A supervised learning model designed for pairwise comparisons such as the Bradley-Terry model would have additional variation in its prediction based on which team *team 1* is and which team *team 2* is.

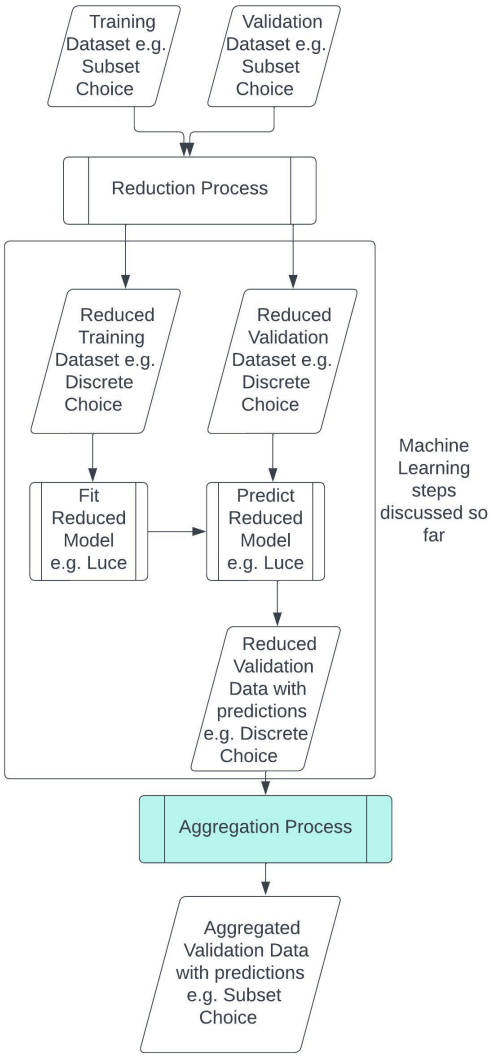
When ties are possible in the pairwise comparison then it is possible to convert the pairwise comparison task into a multiclass task where the options are $\{$alternative 1, alternative 2, tie$\}$. This works because these three options are always available for pairwise comparisons with ties and one of these will always be the outcome.

5.2 Aggregation of a reduced dependent variable

Aggregation is translating the outputs of a model into the output of a parent type. Aggregations might be observed after a reduction to transform the prediction of the reduced models

into the original form, this is shown in the chart in figure 5.3. However, this does not necessarily need to be the case. There have been experiments where researchers have formulated a problem as a pairwise comparison and gathered pairwise comparison data but then use trained pairwise comparison models to infer a full rank over all the alternatives available (Thurstone, 1927b). This section looks at how to create aggregations from model predictions.

Figure 5.3: A flow diagram for a preference model task with reduction and aggregation in this section we will explore the methods of aggregation, highlighted in blue.



There are two types of aggregation non-probabilistic and probabilistic. Non-probabilistic aggregation is where an estimated result for the ground truth of a parent class is given based on the probabilistic or non-probabilistic predictions of a child class. Probabilistic aggregation is when a probability distribution is expressed over all possible ground truth outcomes of a parent class based on the probabilistic or non-probabilistic predictions of a child class. Making probabilistic aggregations is less straightforward and is currently under-explored in the preference model literature.

In this subsection we describe ways to aggregate child model predictions into parent relation types, where a probabilistic approach is available we will mention it. We reiterate that this aggregation does not need to come after a reduction, it is possible to deliberately set up an experiment in a child class to then infer the parent class relations, which is why we keep the aggregation research separately from the reduction research in this thesis.

5.2.1 Aggregating from multiclass and binary classifiers

Aggregating binary classifiers to pairwise comparisons

For aggregating from binary classifiers to pairwise comparisons, recall that the reduction is formulated in terms of the binary outcome of whether *alternative in column 1 will be chosen vs the alternative in column 2: yes or no (1/0)*. When the prediction of the binary classifier is 1 or the probabilistic prediction is greater than 0.5 then the aggregation can be done by saying that the alternative in column 1 is preferred to the alternative in column 2. In the case of a probabilistic output the predicted probability can be interpreted as the probability of the team in the column 1 winning.

In the case of a pairwise comparison model with ties that has been reduced to and then predicted by a multilabel classification problem, when the classifier predicts *alternative 1* is chosen, it is interpreted as $Y_i = \{A_{i,1} \succ A_{i,2}\}$ where $A_{i,1}$ is the alternative present in the row i of the *alternative 1* column and $A_{i,2}$ is the one present in the *alternative 2* column. When the classifer predicts *alternative 2* then $Y_i = \{A_{i,2} \succ A_{i,1}\}$. When the classifier predicts *tie* then $Y_i = \{A_{i,1} \sim A_{i,2}\}$. In the case of probabilistic multilabel classifiers, it is possible to use the predicted probability for each class as the probabilities of either of the teams winning or them tying.

Aggregating binary and multiclass classifiers to discrete choices

Aggregating binary classifiers to discrete choice relations is possible when the binary classifier is probabilistic. In this case the researcher chooses the alternative with the highest predicted probability of being selected as the discrete choice selection. When trying to give a probabilistic output of the likelihood of a discrete choice being made, it is best to use a probabilistic multiclass classifier, which would express the sum of the probabilities of any of the alternatives being chosen as one, $\sum_{Y \in \mathcal{R}_c(\mathcal{A})} P(Y) = 1$. A binary classifier which for each alternative would give a likelihood of it being chosen independently of all other alternatives has no guarantees that its probabilistic predictions will sum to one across all the possibilities of the ground truth, therefore some form of normalisation making them sum to one might be required.

Aggregating binary and multiclass classifiers to subset choices

For subset selection, consider the case of the store owner problem in section 2.3.1. Suppose that we reduce the subset selection to a binary classification problem of the form where if a product gets purchased will be labelled with 1 and if a product doesn't get purchased will be labelled by 0. This would transform the decision level data of table 2.9 into table 5.5. A binary classifier on this problem, such as logistic regression, can create for each product in the store predictions of 1 or 0 whether that customer will purchase the product or not. These outputs then need to be aggregated back to being able to say which subset of products will be purchased. A simple way would be via a rule that every product that is predicted to be purchased with a probability higher than 0.5 during a visit will be in the basket. This probability threshold can be also adjusted by the researcher as a hyperparameter in ways that might improve the accuracy.

For expressing probabilities of different subsets, multiclass classifiers can be used, where each class is a member of the powerset of the list of alternatives, much like in the case of representing discrete choices as multiclass this approach also might generate too many classes to work in practice, since the number of classes generated in this case would become $2^{\#\mathcal{A}}$. Another option could be to treat the likelihood of selection of each option as an independent decision given by the probability of the binary classifier. Then where $P(Y_{i,a}^{(t)})$ is the binary probability of alternative a being chosen in decision i the $P(Y_i = c(\{a, b\}, A_i)) = P(Y_{i,a}^{(t)} = 1)P(Y_{i,b}^{(t)} = 1)$. However, the sum of all permutations might not add up to one in this case. Researchers might choose to normalise the permutations such that they sum to one, though it is unknown how this would impact the accuracy of predictions.

Aggregating continuous and multiclass models to ranking

Using binary classifiers as a reduction to partial orders and full orders would be a little unusual, instead it might be simpler to use a model that predicts continuous variables, and using the ranks as the ground truths. The output of these predictions can be used to create a full rank or a partial rank. If it is of interest to show the probability of certain rankings then it might be best to train a multiclass model where each permutation is modelled as a class. This might not always be computationally feasible though since the number of ranking and especially partial ranking permutations increase with the number of alternatives by the factorial of the number of alternatives available ($\#\mathcal{A}!$).

5.2.2 Aggregating from pairwise comparisons

Aggregating from pairwise comparisons to discrete choice for models that create transitive predictions

In transitive pairwise comparison models, such as the pairwise SVM, there will be one alternative that is always preferred to all other alternatives in each choice. When aggregating to a discrete choice, these would be the alternatives that are selected for the prediction.

Aggregating from pairwise comparisons to discrete choice for models that do not create transitive predictions

Recall that when generating pairwise comparisons, we create at least $\#A_i - 1$ pairwise comparison observations for each observation in the discrete choice relation. The discrete choice prediction would be described as $\hat{Y}_i^{(disc)}$, we use $\hat{Y}_i^{(pairwise)}$ to denote pairwise comparison predictions $\hat{Y}_i^{(disc)} \in \mathcal{R}_{ch}$ and $\hat{Y}_i^{(pairwise)} \in \mathcal{R}_{as}^{\#A_i - 1}$. We will denote the predicted outcome of the pairwise comparison of a and b in observation i by $\hat{Y}_{i,a,b}^{(pairwise)}$. Note that for our purposes $\hat{Y}_{i,a,b}^{(pairwise)} = \hat{Y}_{i,b,a}^{(pairwise)}$.

For models that do not create a transitive preference the voting rule developed by Knerr et al. (1990) and Friedman (1996) can be used. We can define this rule by using an indicator function $I : \mathcal{R} \times \mathcal{R} \to \{0, 1\}$ which takes two relations and returns 1 if they are equal and 0

otherwise :

$$\hat{Y}_i^{(disc)} = c\left(\underset{a \in A_i}{\text{argmax}}\left[\sum_{b \in A_i \setminus a;} I(\hat{Y}_{i,a,b}^{(pairwise)}, \{a \succ b\})\right], A_i\right)$$

Which is saying that the alternative in A_i that is the one that would be chosen most of the times from pairwise comparisons is the one that will be selected in a discrete choice.

Aggregating from pairwise comparisons to subset choices

We struggled to find much literature around aggregating from pairwise comparisons to subset choice. For predicting the output in transitive models it should be possible using a predetermined number of alternatives to be selected. For example, if a researcher is told up front that the subset selected is always the top 2 then the top 2 ranked alternatives for each choice would be selected in the aggregation. The top x number of alternatives to be selected can be fixed by the researcher as a hyperparameter or it can be estimated separately in a composition step, more about the latter in section 4.5.

Aggregating from pairwise comparisons to full and partial orders

Transitive models can be used to predict full and partial orders also. For full order, the transitive ranking is taken, for the partial order, extra rules might need to be defined to determine which pairs tie, or a pairwise comparison model that can handle ties can be used, such as the (Rao and Kupper, 1967) extension of the Bradley-Terry model.

Creating a full rank from pairwise comparisons is well known with the dedication of a whole sub-field in learning to rank where it is known as pairwise approach of ranking (Liu, 2011; Negahban et al., 2017; Wauthier et al., 2013; Hüllermeier et al., 2008). During the MPhil there was not a large focus on ranking from pairwise comparisons and there is much literature to be read here, we have so far not found any comprehensive guidance on how to aggregate pairwise comparison probabilities to the probability of a ranking.

5.2.3 Aggregating from discrete choice

Aggregating from discrete choice to subset choices

Amongst reduction techniques of converting subset choices into discrete choices we mentioned that there are two methods, the first one presented where subset choices are expressed as discrete choices in the power-set of the set of alternatives as shown in table 5.3 makes aggregation simple. It is possible to use any discrete choice model to make predictions for the subset, which is treated as an alternative chosen from the power set and a probabilistic discrete choice model would give probabilities for these also.

The question becomes trickier when aggregating from a subset selection that has been converted via the loop method (algorithm 7) the result of which is shown in table 5.4. The first challenge in predicting here is that once again the researcher does not know pre-emptively how many alternatives will be selected. This has to be predefined either as a hyperparameter or through a composite model (as in section 4.5). Once the number of selections has been defined then for predictions the researcher needs to loop through the set of alternatives eliminating one alternative in each loop to come up with the predictions for the subset selection. For example, suppose that the alternatives presented are $A_i = \{a, b, c, d\}$ and it is given that there will be two items selected. Then the researcher would go through a process of finding the alternative that has the highest probability of being chosen from that list $argmax_{k_1}(P(Y_i = c(k_1, A_i)))$ for $k_1 \in A_i$ then afterwards the loop is repeated excluding the variable just selected $argmax_{k_2}(P(Y_i = c(k_2, A_i \setminus k_1)))$ for $k_2 \in A_i \setminus k_1$.

Aggregating from discrete choice to full and partial orders

Many discrete choice models would create an implicit ranking of the alternatives, in the simplest case, the Luce model with no covariates the vector of strength parameters λ can be used for this, or when using covariates the utility $u(x)_i = \lambda_x + X_i\beta_x + G_x\gamma + C_{S_i}\kappa_x$ can be used to score up and rank all the alternatives for a specific choice. These rankings can be used to predict full orders and partial orders with the researcher defining threshold boundaries within which alternatives tie.

5.2.4 Aggregating from subset choice

We haven't identified many models that work on modelling subset choices, we have identified a few papers to read here such as Falmagne and Regenwetter (1996), Doignon et al. (2004) and Benson et al. (2018), the latter showing that learning the optimal subset is an NP-hard problem. Each of these are rooted in random utility models and the first two are strongly referencing internal full ranks of items, understanding these model might lead to the ability to map them for aggregation also.

5.2.5 Aggregating from partial order

At this time we still haven't covered models that work with partial orders. However, we have found some learning to rank papers that talk about learning to rank from data that contains ties such as Zhou et al. (2008) and Zhu and Klabjan (2020), however it is not yet clear to us whether these models always learn a full order despite the fact that the training data contains ties, or are also capable of predicting ties. If its the first then these models can be used as aggregators from partial orders. We would need to investigate more this topic to form conclusions on aggregations from partial orders.

In table 5.6 we summarise the aggregation approaches outlined in this section.

Table 5.6: Ways of aggregating

Aggregating from	**Aggregating to**	**Method for predicting the ground truth**	**Method for probabilistic prediction**
Binary	Pairwise comparisons	Most likely alternative wins pairwise comparison	Directly from probabilistic output
Binary	Discrete choice	Most likely alternative is chosen	Normalised probability outputs or using multiclass classification
Binary	Subset choice	Top k most likely alternatives are chosen or anything above a certain probability threhsold is	Normalised interactive probability outputs or using multiclass classification on powerset
Continuous model or multiclass for probabilistic prediction	Partial order	Alternatives ranked by buckets of probability thresholds	Using multiclass method for each class
Continuous model or multiclass for probabilistic prediction	Full order	Alternatives ranked by probabilities	Using multiclass method for each class
Pairwise comparisons	Discrete choice	Alternative with highest rank gets chosen	Unknown
Pairwise comparisons	Subset choice	Top k alternatives with highest ranks get chosen	Unknown
Pairwise comparisons	Partial order	Use internal ranking of transitive models with threhsold	Unknown
Pairwise comparisons	Full order	Transitive model's internal ranking	Unknown
Discrete choice	Subset choice	For a predefined number of alternatives it is possible to loop through it	Unknown
Discrete choice	Partial order	Can use internal ranking with thresholds	Unknown
Discrete choice	Full order	Can use internal ranking	Unknown
Subset choice	Partial order	Can use internal ranking with thresholds	Unknown
Subset choice	Full order	Can use internal ranking	Unknown
Partial order	Full order	Unknown, but more likely due to lack of research on our end	Unknown

5.3 Our contribution: Probabilistic aggregation of pairwise comparison predictions to discrete choice

5.3.1 Research question

There should be an investigation in the different ways in which it is possible to aggregate probabilistically from pairwise comparisons into discrete choices. The pairwise probabilistic aggregations problem in the simplest case of making a discrete choice from a set of three alternatives $\{a, b, c\}$ can be described in the following way (using notation from section 4.3): if we can obtain $P(Y_{i,a,b}^{(pairwise)} = \{a \succ b\})$, $P(Y_{i,a,c}^{(pairwise)} = \{a \succ c\})$ and $P(Y_{i,b,c}^{(pairwise)} = \{b \succ c\})$ how can we say what is $P(Y_i = c(a, \{a, b, c\})$?

As mentioned earlier, from a researchers perspective, the main purpose of reduction is to change the original task which may not have many robust solutions into a simpler one which is well studied. It is valid to ask why study the specific reduction-aggregation problem between pairwise comparisons to discrete choice given how advanced the field of discrete choice modelling is. There might be no or little methodological gains from reducing discrete choices into pairwise comparisons.

It is true that there is no need to make reductions due to lack of mathematical knowledge in discrete choices. However, we have not yet found a paper that talks about evaluating the accuracy and computational performance of reduced pairwise comparison vs. native level discrete choice models, whilst literature in close disciplines indicate there might be some gains to be had in computational performance. An analogous problem is that of pairwise coupling, also known as round-robin classification which is a method that first reduces multiclass classifiers into $\frac{c(c-1)}{2}$ (where $c \in \mathbb{N}$ is the number of classes) binary classifiers and then aggregates binary classifiers to multiclass classifiers (Wu et al., 2004; Fürnkranz, 2002). Hüllermeier et al. (2008) have shown that this approach is "superior in terms of computational efficiency, and at least competitive in terms of accuracy". Having a reference to whether there are any potential similar gains in reducing discrete choices to pairwise comparisons can help researchers either use techniques with higher performance or to have a warning to know to avoid this type of reduction all-together, which might save time by narrowing the number of different techniques to consider.

For human decisions, it is unlikely that there is an underlying process of reducing discrete choices into pairwise comparisons, because the number of pairwise comparisons in a list of alternatives (A) that can be made are $\frac{\#A(\#A-1)}{2}$ which escalates quickly for people to efficiently think that way. For example, when choosing from 10 options 45 different pairwise

comparisons would need to be made. It is unlikely that a human would internally go through 45 comparisons to solve the problem of choosing one item from ten. A machine however would be easily able to do this. We are living in the advent of artificial intelligence (AI) and many AI methods such as machine vision, voice assistants and self-driving cars make decisions and express preferences. Since these are mostly using black-box techniques which are "models [that] have a complex and opaque decision-making process that is difficult for humans to understand" (Wang and Lin, 2019), we often do not know what drives their decision making process, it could be that an AI making a discrete choice concludes that the best decision can be made when reducing the problem to pairwise comparisons. This is not something that a researcher can figure out from a first glance of looking at the AI's algorithm if this is generated by a black-box technique. More recently there has been significant effort dedicated to understanding how these machines make decisions (Guidotti et al., 2018). Opening ways for probabilistic aggregation from pairwise comparison models could be helpful in the future for testing the hypothesis on whether the decisions certain AI takes is based on a reduction to pairwise comparisons.

When it comes to aggregating probabilistically there is no one solution that works. This is because similarly to pairwise coupling, there are $\#A_i - 1$ free parameters satisfy to $\frac{\#A_i(\#A_i-1)}{2}$ constraints since the following needs to hold true: $\sum_{a \in A_i} P(Y_i = c(a, A_i)) = 1$ (Hastie and Tibshirani, 1997). To answer how to make probabilistic pairwise aggregations into discrete choice we suggest:

- Start to outline necessary (but not sufficient) conditions that all methods of probabilistic aggregation must follow for them to provide reasonable results.
- Propose new mathematical methods for aggregating.
- Translate the aggregation methods developed in the pairwise coupling field for pairwise comparison to discrete choice aggregation and then examine them to see if they meet the conditions which we deem necessary for an aggregator to work.
- From the methods that adhere to the necessary conditions for being a pairwise to discrete choice aggregator evaluate their performance on synthetic and real-life data.
- Evaluate the difference in performance compared to using discrete choice models.

5.3.2 General aggregation method from pairwise comparisons to discrete choice

Criteria for a feasible pairwise aggregator

We began forming some guidelines for what are the necessary requirements for a pairwise comparison to discrete choice probabilistic aggregator to be considered valid.

Definition 12. Conditions for pairwise to discrete choice aggregations

1. The probabilities of all the discrete choices sum to one: $\sum_{j \in A_i} P(Y_i = c(j, A_i)) = 1$
2. The probability of a discrete choice is the same constant across all alternatives, only if the pairwise probabilities of the alternatives are all 0.5 for $j \in A_i$; $P(Y_i = c(j, A_i)) = \frac{1}{\#A_i}$ if and only if $P(Y_i = \{a \succ b\}) = 0.5 \forall a, b \in A_i$ where $a \neq b$
3. If a is 100% preferred to b and c then a is 100% preferred from $\{a, b, c\}$ if $P(Y_i = \{a \succ b\}) = 1$ and $P(Y_i = \{a \succ c\}) = 1$ then $P(Y_i = c(a, \{a, b, c\})) = 1$

■

Our current proposed methods

We have also begun working on our own proposed methods for aggregating which adhere to the rules above.

A proposal to aggregate when using the Bradley-Terry model is to use the fact that it is a special case of the Luce model. To use simpler notation, let's define the utility of alternative x as $u(x)_i = \lambda_x + X_i\beta_x + G_x\gamma + C_{S_i}\kappa_x$. Suppose we have a decision with alternatives $A_i = \{a, b, c\}$, then implementing a Bradley-Terry model that has been learned on a pairwise reduction of the same process that generated this data, the pairwise comparison probabilities would yield

$$P(Y_i = \{a \succ b\}) = \frac{e^{u(a)_i}}{e^{u(a)_i} + e^{u(b)_i}}$$

We know that the general term for Luce's model for choosing a from A_i would be:

$$P(Y_i = c(a, A_i)) = \frac{e^{u(a)_i}}{e^{u(a)_i} + e^{u(b)_i} + e^{u(c)_i}}$$

The Bradley-Terry model would have learned all the parameters necessary to estimate the utility functions, so using these parameters directly with the Luce formulation would be for example a method that satisfies all three of the above conditions.

For probabilistic pairwise comparison models that aren't Bradley-Terry for example the Thurstone model another suggestion can be made using the following assumptions:

- Assume pairwise preferences are transitive
- Assume they are independent, that is $P(\{a \succ b\} \cap \{b \succ c\}) = P(\{a \succ b\})P(\{b \succ c\})$
- Assume that when $c(a, A_i)$ then $a \succ j \forall j \in A_i \setminus a$

We can use the following formulation

$$P(Y_i = c(a, A_i)) = \frac{\prod_{j \in A_i \setminus a} P(Y_i = \{a \succ j\})}{\sum_{k \in A_i} \prod_{j \in A_i \setminus k} P(Y_i = \{k \succ j\})}$$

which also follows the conditions outlined in definition 12. Note that the Bradley-Terry probabilities could also be used in this formulation, but they would give a different result than in the previous suggestion.

Exploring further methods from pairwise coupling to be used for probabilistic aggregation of pairwise comparisons to discrete choices

In this section we present existing techniques in the field of pairwise coupling but we translated the original formulas into forms that we believe would be applied for aggregating pairwise comparisons into discrete choices. Pairwise coupling is the technique for reducing a multiclass classification problem into a binary models of pairwise comparisons between all the alternatives in the multiclass classification problem and then these binary models are aggregated to create a multiclass classification. The methods for making such an aggregation outlined in Wu et al. (2004). The fact that pairwise comparison is related to binary choice and discrete choice is related to multiclass classification means that there might be techniques from the field of *pairwise coupling* that could be used for aggregating pairwise comparisons into discrete choices.

Below we list some of the techniques we have come across, formulated in terms of aggregating from pairwise comparisons to discrete choice, rather than their original form, which

was to aggregate from binary classifications to multiclass. It should be further investigated if these methods would adhere to the rules we described in definition 12 and then see how accurate they are compared to each other, but also compared to just using discrete choice models.

Hastie and Tibshirani (1997) note that probabilistically aggregating binary models into classifiers does not have an exact solution since we require that $\#A_i - 1$ free parameters satisfy $\frac{\#A_i(\#A_i-1)}{2}$ constraints since the following needs to hold true: $\sum_{a \in A_i} P(Y_i = c(a, A_i)) = 1$.

Using the voting rule is one way of estimating the discrete choice probability $\hat{P}(Y_i = c(a, A_i))$ where I is an indicator function $I : \mathcal{R} \times \mathcal{R} \to \{0, 1\}$ which takes two relations and returns 1 if they are equal and 0 otherwise:

$$\hat{P}(Y_i^{(discrete)} = c(a, A_i)) = \frac{2 \sum_{b \in A_i \setminus a} I(\hat{Y}_{i,a,b}^{(pairwise)}, \{a \succ b\})}{\#A_i(\#A_i - 1)}$$

Translating the approach outlined by Refregier and Vallet (1991) for pairwise coupling, in the space of preference models would suggest setting the following equality, which note is the definition of independence from irrelevant alternatives (see definition 6):

$$\frac{\hat{P}(Y_{i,b,a}^{(pairwise)} = \{a \succ b\})}{\hat{P}(Y_{i,b,a}^{(pairwise)} = \{b \succ a\})} = \frac{\hat{P}(Y_i^{(discrete)} = c(a, A_i))}{\hat{P}(Y_i^{(discrete)} = c(b, A_i))}$$

selecting any $\#A_i - 1$ estimated pairwise probabilities, conditioning on $\sum_{a \in A_i} \hat{P}(Y_i^{(discrete)} = c(a, A_i)) = 1$ and solving a set of linear equations. In this method the final probabilities estimated depend a lot on the $\#A_i - 1$ pairwise probabilities selected (Wu et al., 2004; Price et al., 1995).

Another pairwise coupling approach suggested by Price et al. (1995) would translate the following rule:

$$\hat{P}(Y_i^{(discrete)} = c(a, A_i)) = \frac{1}{\sum_{b \in A_i \setminus a} \frac{1}{\hat{P}(Y_{i,b,a}^{(pairwise)} = \{a \succ b\})} - (\#A_i - 2)}$$

And then normalising the resultant probabilities to $\sum_{a \in A_i} \hat{P}(Y_i = c(a, A_i)) = 1$.

The pairwise coupling method suggested by Hastie and Tibshirani (1997) in the space of preference models would define $\mu_{a,b} = \frac{\hat{P}(Y_i^{(discrete)} = c(a, A_i))}{\hat{P}(Y_i^{(discrete)} = c(a, A_i)) + \hat{P}(Y_i^{(discrete)} = c(b, A_i))}$ and propose

minimising the Kullback-Leibler distance between $\hat{P}(Y_{i,b,a}^{(pairwise)} = \{a \succ b\})$ and $\mu_{a,b}$. Which can be done via algorithm 9.

Algorithm 9: Minimising the KL distance for pairwise comparisons for aggregating into discrete choices

define $n_{a,b}$ as the number of times a has been compared to b in a pairwise comparison

Initialise randomly $\hat{P}(Y_i^{(discrete)} = c(a, A_i)) \forall a \in A_i$ with values from $\mathbb{R}_{[0,1]}$ and their corresponding $\mu_{a,b}$ for $b \in A_i \setminus a$

define $\alpha \in \mathbb{R}_{(0,1)}$

while $1 - \alpha < \frac{\sum_{b \in A_i \setminus a} \hat{P}(Y_{i,b,a}^{(pairwise)} = \{a \succ b\}) n_{a,b}}{\sum_{b \in A_i \setminus a} \mu_{a,b} n_{a,b}} < 1 + \alpha$ **do**

  for $a \in A_i$ **do**

    $\hat{P}(Y_i^{(discrete)} = c(a, A_i)) \leftarrow \frac{\hat{P}(Y_i^{(discrete)} = c(a,A_i)) \sum_{b \in A_i \setminus a} \hat{P}(Y_{i,b,a}^{(pairwise)} = \{a \succ b\}) n_{a,b}}{\sum_{b \in A_i \setminus a} \mu_{a,b} n_{a,b}}$

Normalise the probabilities such that $\sum_{a \in A_i} \hat{P}(Y_i^{(discrete)} = c(a, A_i)) = 1$

Finally the pairwise coupling approach from Wu et al. (2004) would be akin to proposing to solve the system

$$(Y_i^{(discrete)} = c(a, A_i)) = \sum_{b \in A_i \setminus a} \left(\frac{\hat{P}(Y_i^{(discrete)} = c(a,A_i)) + \hat{P}(Y_i^{(discrete)} = c(b,A_i))}{\#A_i - 1} \right) \hat{P}(Y_{i,a,b}^{(pairwise)} = \{a \succ b\}) \tag{5.1}$$

subject to $\sum_{a \in A_i} \hat{P}(Y_i^{(discrete)} = c(a, A_i)) = 1; \hat{P}(Y_i^{(discrete)} = c(a, A_i)) > 0 \forall a$ which is done by obtaining the unique global minimum to the convex problem:

Let $\hat{P}_{disc.}$ be the vector of discrete probabilities $\hat{P}_{discrete} = [\hat{P}(Y_i^{(discrete)} = c(a, A_i)) : a \in A_i]$

$$\underset{\hat{P}disc.}{\operatorname{argmin}} \sum a \in A_i \left(\sum_{b \in A_i \setminus a} \hat{P}(Y_{i,a,b}^{(pair.)} = \{b \succ a\}) P(Y_i^{(disc.)} = c(a, A_i)) \right.$$
$$\left. - \sum_{b \in A_i \setminus a} \hat{P}(Y_{i,a,b}^{(pair.)} = \{a \succ b\}) P(Y_i^{(disc.)} = c(b, A_i)) \right)^2 \tag{5.2}$$

Another approach they suggested would be the equivalent to minimising the following problem:

$$\underset{\hat{P}_{disc.}}{\operatorname{argmin}} \sum_{a \in A_i} \sum_{b \in A_i \setminus a} \Bigg(\hat{P}(Y_{i,a,b}^{(pair.)} = \{b \succ a\}) P(Y_i^{(disc.)} = c(a, A_i)) \\ - \hat{P}(Y_{i,a,b}^{(pair.)} = \{a \succ b\}) P(Y_i^{(disc.)} = c(b, A_i)) \Bigg)^2 \tag{5.3}$$

As we can see there are several methods that could be borrowed from pairwise coupling in terms of aggregating pairwise comparison models to discrete choice and investigating whether similar efficiency gains could be made from this aggregation as in the space of multiclass classification could be interesting future research.

Chapter 6

Preference model software and architectures

In this chapter we will focus on the Python open source software and how practitioners who would like to implement preference models can use it. We start by describing the current most successful data science package in Python called scikit-learn and note the lack of preference models in this infrastructure. Nevertheless, the approach that scikit-learn developers took to create an environment for data science software has much transferable knowledge to the space of preference models and the approach in general is the benchmark for best practice in data science software development in Python, so we examine some of its crucial components that should serve as transferable ideas to a preference model infrastructure. We also talk about the pandas package in Python which allows for the kinds of data sets we described for use with preference models to be stored and manipulated in Python.

Based on our learning from best practice from the scikit-learn package and crucial properties of preference models outlined in the earlier chapters of this thesis, such as reduction, aggregation and composition, we make an argument for what should be key components to software that hosts packages that can be used for preference models. With those criteria in mind we present software that is currently available in Python for solving supervised preference tasks and make the argument that none of them quite adhere to the principles we think would be necessary for a single Python repository.

We began a Python package currently hosted for further development in GitHub that could contain preference models in a similar way as the scikit-learn infrastructure hosts other

models currently and adheres to the principles we outline for preference model software in Python. In section 6.2 we present the key concepts in this package and how they address the principles we outlined, however a more detailed documentation of the package is available in GitHub pages.

6.1 Availability and design for using supervised preference models in open source software

Software allows researchers to apply machine learning techniques and statistical analysis.Using published software ensures consistency in the execution of methods across different researchers and establishes a common standard. This allows research to be much more reproducible, since the same approach used in a software should generate the same results on the same data. This makes research much more transparent in the community and easier to iterate improvements on as when an improvement is created within the same software, all researchers who use it can benefit from it and easily re-run their analysis to see if their conclusions hold with the new improvements also.

The most accessible and popular tools for machine learning are Python (Van Rossum and Drake, 2009) and R (R Core Team, 2020). Both are considered to be open source, an initiative that allows free licence access to software and depends on several contributors to tailor it. An exact definition of open source can be found in Perens et al. (1999). Both Python and R have a rich community of researchers contributing to them. Software contributed to Python and R are called packages and these are generally developed in the public domain where the codes are stored most of the time using the public repository called github (2008). This has a significant advantage, since it allows researchers full transparency in reviewing and contributing to the ecosystem of packages ensuring the highest possible quality of execution and learning from different areas of data science, for example computational efficiency and statistical robustness to be shared across communities faster. In this thesis there will be a focus on Python packages, although since a significant amount of research has gone into developing preference models in R these will also be mentioned.

6.1.1 Optimal design architecture for supervised preference models

Object oriented programming

State of the art machine learning software use a programming principle known as object oriented programming (Lutz, 2010; Meyer, 1997). We can explain object oriented programming by thinking back to our supervised learning tasks. A supervised learning model has parameters that are adjusted to fit the data according to an objective function, recall definition 5 for a summary. We can think of supervised learning models also as a recipe or set of instructions on how to adjust parameters when presented with a given dataset. The same supervised learning model will have the same set of instructions, however, when presented with different datasets, it will produce different results, since it will make different parameter adjustments. In object oriented programming, *classes* are the recipes. Once the class is applied for a specific dataset using a specific set of hyper-parameters, it becomes unique, the supervised learning model for that dataset with those hyper-parameters. This is called an instance of the class, which is also referred to as an object. In more general terms

> "The class is a software text. It is static; in other words, it exists independently of any execution. In contrast, an object derived from that class is a dynamically created data structure, existing only in the memory of a computer during the execution of a system" (Meyer, 1997).

Best practice amongst ML software

The most popular machine learning package in Python is called scikit-learn (Pedregosa et al., 2011). It "is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems" Pedregosa et al. (2011). The intention of the authors was to contribute a package that is: open source, efficient, has as few dependencies as possible (it's only dependencies are numpy (van der Walt et al., 2011) and scipy (Virtanen et al., 2020)). The authors have compared it to other existing packages such as MLPy, PyBrain, pymvpa, MDP. Compared to these machine learning libraries scikit-learn is the package that has the most stars, forks and watches on GitHub, which we might use as an indication of popularity of the package. We can see the breakdown of this in Table 6.1, so it is possible to say that scikit-learn has accomplished its mission to make a better interface than the ones the developers were benchmarking against. Albanese (2019) IDSIA/CogBotLab (2019) Hanke et al. (2019) Zito et al. (2019) Sonnenburg et al. (2019)

Due to the success and strong popularity of scikit-learn for a machine learning tool to be useful today, it would have to be as closely aligned with scikit-learn as possible, an example

Table 6.1: Python machine learning package popularity on GitHub as of 13-05-2019

package	scikitlearn	PyBrain	shogun	pymvpa	MDP	MlPy
stars	35,038	2,687	2,433	245	62	1
forks	17,119	785	926	35	30	2
watches	2,245	250	219	118	13	3

is XGBoost, which was written to integrate "naturally with language native data science pipelines such as scikit-learn" Chen and Guestrin (2016). The technique has become popular after yielding successful results for competitors on Kaggle (Chen and Guestrin, 2016). The package has 15,917 stars, 6,652 forks and 965 watches (XGBoost developers, 2019). sktime (Löning et al., 2019) is another package that has been recently written specifically for time series data, with an interface that is compatible with scikit-learn. Since scikit-learn has set such a dominant best practice in place for machine learning in Python, we also believe that a package that hosts preference models should be compatible with scikit-learn.

> "[In scikit-learn the] estimator interface is at the core of the library. It defines instantiation mechanisms of objects and exposes a fit method for learning a model from training data.
>
> ...
>
> Estimator initialization and actual learning are strictly separated, in a way that is similar to partial function application: an estimator is initialized from a set of named constant hyper-parameter values (e.g., the C constant in SVMs) and can be considered as a function that maps these values to actual learning algorithms. The constructor of an estimator does not see any actual data, nor does it perform any actual learning. All it does is attach the given parameters to the object.
>
> ...
>
> Actual learning is performed by the fit method. This method is called with training data (e.g., supplied as two arrays X_train and y_train in supervised learning estimators). Its task is to run a learning algorithm and to determine model-specific parameters from the training data and set these as attributes on the estimator object.
>
> ...
>
> The predictor interface extends the notion of an estimator by adding a predict method that takes an array X_test and produces predictions for X_test, based on the learned parameters of the estimator" Buitinck et al. (2013).

Király et al. (2021) have recently formalised the process of creating machine learning interfaces, which also encompasses the scikit-learn interface. This report will not go into the same level of detail of how machine learning interfaces work as the paper, however it will use some concepts developed there, specifically, "scientific typing" scitype for short which is "an abstract mathematical object based on the set of operations that we usually perform with

them". What we described in the previous paragraph is the equivalent of what Király et al. (2021) call a "supervised learner" generally:

class type	$SupervisedLearner$		
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$(\mathcal{X} \times \mathcal{Y}) \rightarrow$ model
	predict	:	$\mathcal{X} \times$ model $\rightarrow \mathcal{Y}$

Where $\mathcal{X} \in \mathbb{R}^x$ for some $x \in \mathbb{N}$ and $\mathcal{Y}$ is the dependent variable, in our case $\mathcal{Y} \in \mathcal{R}_p$ for some property p. Paramobject are parameter objects for the model, this would contain the hyperparameters of the model as well as the model parameters. Mathobject is an abstract representation of model objects. The state in this case refers to whether the model has already been fitted or not. In scikit-learn the predict method is conditioned such that it cannot run if the state of the model is not fitted.

Listing 6.1: An example of how to use the sklearn interface

```
# load the LogisticRegression object
from sklearn.linear_model import LogisticRegression
# Set up a Logistic regression with with
# regularisation parameter of 0.5
clf = LogisticRegression(C = 0.5)
# For a matrix of covariates X_train
# and its corresponding ground truths y_train fit a model
clf.fit(X_train, y_train)
# Use the model for prediction with unseen observations X_test
y_hat = clf.predict(X_test)
```

There are several benefits of object oriented programming, and this thesis will not cover all of them, but one more important one to mention is inheritance. Classes can inherit attributes and methods from each other. For example, all supervised learning methods have parameters and it might be of interest to query them. Scikit-learn does this by having a class called *BaseEstimator* which has a *get_params* method which fetches the parameters of a model. If there are two supervised learning models let's say logistic regression and random forest, the class for these will not have their own *get_params* method, they will inherit it from *BaseEstimator* and it will behave in exactly the same way in both. For a modelling package to be compatible with scikit-learn it needs to follow the signature methods from the scikit-learn estimators. For ease of integration it is also recommended by the scikit-learn contributing documentation that estimators inherit from the scikit-learn BaseEstimator and optionally a

few other methods such as the ClassifierMixin (Scikit learn online documentation, 2019).

Transformers as scitypes would be defined as

class type	$Transformer$		
params	paramlist	:	paramobject
state	model	:	mathobject
methods	fit	:	$(\mathcal{X}) \rightarrow$ model
	transform	:	$\mathcal{X}\times$ model $\rightarrow \mathbb{R}^{x\times y}$ for some $x, y \in \mathbb{N}$

Pipelines in scikit-learn are classes that string objects together in various steps, such as a transformation of the input variables and a predictive model. Pipelines also have a *fit* and *predict* methods. So once the user has an instance of a pipeline they can call the *fit* and *predict* methods just like they would for one model. More specifically, $Pipeline : (Transformer)^n \times (SupervisedLearner)^n \rightarrow SupervisedLearner$. The execution looks like figure 3.1, however in practice in could be much more complex. A good machine learning software will make it much easier for the researcher to create complicated model structures.

There are several reasons why scikit-learn falls short for solving preference tasks. The main reason is that it doesn't actually contain preference models, but rather contains binary, multiclass and continuous models, meaning that for solving a preference problem in scikit-learn the researcher must use a reduction.

Whilst the scikit-learn approach is powerful in its logical flow, its issue is that it forces a user to create a pooled dataset as the input data. If the database set up is relational, then the user is forced to merge together all the tables for modelling. The problem with this is that it:

- is a fixed repetitive work that needs to be done by each user of the package, which is at best a waste of time, at worst an additional source of human error
- relies on the users' pro-activity to remove the pooled dataset after modeling to be more memory efficient
- recall that creating reductions often generates a longer table, for example, when a subset choice gets reduced into discrete choice see table 5.4. The issues of a pooled dataset then get magnified in preference modelling when researchers are using reductions, because longer tables mean more memory requirements and more unnecessary repetition of the same values

- potentially encourages users to store pooled tables for modeling and predicting rather than keep the relational set up that they are likely to already be working with. This at worst will shift practitioners from not saving the data in a relational set up, but in a pooled set up which suffers from the problems discussed in section 2.4.1, and at best uses up unnecessary memory in storage
- doesn't allow for faster computation by using the normalised optimisation methods discussed in section 3.4.1

A better approach that would allow to user to occupy less memory, would merge all the tables together inside the .fit() procedure (only if necessary) and then delete this big table at the end of the fitting process, thereby freeing up memory for the user for future processes.

As discussed in section 2.4.1, and at several other points in this thesis, there can be several potential benefits to incorporating the relational format into the preference modelling pipeline. In the next section we discuss the tools that exist in Python to work with a relational database format.

6.1.2 Relational databases in Python

There are ways of setting up relational data structures in Python. Pandas is a package in Python that has been developed with the R counterpart of *data.frame* in mind.

> "The pandas data structures internally link the axes of a ndarray with arrays of unique labels. These labels are stored in instances of the Index class, which is a 1D ndarray subclass implementing an ordered set ... An Index stores the labels in two ways: as a ndarray and as a dict mapping the values" (McKinney, 2010).

It also allows to merge together datasets either by the index or other specified columns. Pandas offers a natural way in which to store pairwise comparison data since the user can use these indices as a way to store the entities being compared by allowing for a 2darray to be stored as the index with its *set_index* method, which can be the two alternatives compared. Pandas can read in both csv and SQL tables, so it can work directly with a SQL relational database.

Featuretools is a Python package that uses "the *Deep Feature Synthesis* algorithm for automatically generating features for relational datasets" (Kanter and Veeramachaneni, 2015). To link several relational datasets in the feature generating mechanism, it has a data container called EntitySets. In this data container the user specifies the relationship and the direction of the relationship between several sources of data, for example how a product price table

might link into an in-store transactions table, which then might link to a customer table, this then can be queried like a comprehensive data dictionary (Labs, 2019).

The main purpose of featuretools is not to provide the EntitySet framework, but to do *Deep Feature Synthesis*, which is a process that eventually joins together all tables and computes several transformations of features in preparation for feeding it into a scikit-learn style table. For the purposes of a package that wants to enable the user to use preference models the main benefit is in that it can become implicit which table contains the list of alternatives and what other fields they are in the tables, by querying the stored *parent_variable* variable in the EntitySet object.

6.1.3 Ideal architecture of preference models

The reason preference models don't exactly fit the framework of scikit-learn can be seen in example code 6.1 an X_train array-like object will not provide enough information about what the alternatives are, it is also incapable of recognising what type of relationship is being expressed in the ground truth. The framework has been designed around problems where a solution needs to be found to map k covariates across n observations $x \in \mathbb{R}^{k \times n}$ to the ground truth values y which can be categorical, continuous or binary. However, for preference models it is crucial to know what the set of alternatives are for each decision. One way such a set up would work is if there is pre-defined formatting of the X_train array, by saying that the first two elements must be the alternatives and the ground truth. However, that set up is highly error-prone, because this can be an easy property to forget about especially for those used to working with scikit-learn. The other way is to make the user include a second array-like object which specifies the alternatives. However, this requires researchers to separate columns from the observation level table and create another object from these which is an unnecessary thing to do. Then, it needs to be made clear what type of relations the dependent variable is expressing, which is also information that is not contained in an array. Furthermore as discussed in lemma 3.3.2, for some generalised linear models alternative level covariates need to be treated differently than decision level covariates, and in those cases models would need to know which elements in the array are decision level and which ones are alternative level covariates.

EntitySet by featuretools would offer a more elegant solution to this by needing to feed in only one EntitySet object which captures all the information between entities linking to interactions and keeps the entity level features in the entity level table thereby avoiding data redundancy. On the other hand coding up the relational structure as currently is done in

featuretools would mean that users have to write a lot of extra code compared to the scikit-learn set up. Furthermore, the more other packages a Python package depends on (the more dependencies it has), the higher the risk of the code breaking and the higher the risk of incompatibilities rising between dependencies.

In this section we will present the ideal properties that we believe software that analyses preference models would have. They are:

1. **Using object oriented programming that is interoperable with scikit-learn**. scikit-learn has an arsenal of useful functions for machine learning pipelines, for example GridSearch. It has a rich community of contributors that ensure that these functions work as efficiently as it is possible. Therefore, a package that is not well integrated with scikit-learn is likely to miss opportunities for efficiently taking steps in the machine learning pipeline. Furthermore, researchers accustomed to scikit-learn, which is likely to be most Python based machine learning researchers, will not have to learn a lot of new skills to be able to use it.

2. **Having a set up that facilitates usage with relational data structures**. We have discussed in chapter 2 that it would be common to have a relational database set up of up to three different datasets in preference models: decision level data, alternative level data, decision maker level data. Having to pool these together manually causes problems addressed in 6.1.1.

3. **Working with different types of relations expressed.** We have identified five different relations that would be common to express in preference models (full rank, partial rank, subset choice, discrete choice and pairwise comparisons). A package that deals with preference models should be able to represent these differently from each other and recognise them.

4. **Ability to do reduction and aggregation in an object oriented way**. Since aggregations and reductions are a key technique in preference models, any package should be able to seamlessly (at least to the front end user) translate data into reductions and for appropriately trained models aggregations of each other.

5. **Creating pipelines that can accommodate anatomising compositions** where it is understood that each model's decisions are conditional on the previous models decisions. This is similar to how pipelines are defined by Löning and Király (2020).

6.1.4 Supervised preference modelling packages in R and Python

In this subsection we will discuss some of the available options in R and Python for preference models. In this MPhil, we will focus on implementation in Python, so for Python packages we will give more critical breakdown regarding the criteria outlined above. For R we will only mention existing packages, but we will not analyse them in the same detail. Table 6.2 shows a summary of what packages are available in R and Python.

Table 6.2: Models and available packages

Model	packages in R	packages in Python
Thurstone	*BradleyTerry2*	**Not found**
General Logit Type	**Not found**	*pylogit*, *biogeme*, *cs-rank*
Luce	*Plackett-Luce*	*choix*
Zermelo-Bradley-Terry-Elo	*BradleyTerry2*, *prefmod*,	*choix*, *cs-rank*
Elimination by aspects model	*eba*	**Not found**
Coherency Driven Model	**Not found**	**Not found**
Plackett-Luce and variants	*PlackettLuce*, *StatRank*, *pmr*	*choix*
FATE / FETA	**Not found**	*cs-rank*
Bradley-Terry trees	*psychotree*	**Not found**
Nested Logit Models	*mlogit*	*pylogit, biogeme, cs-rank*,

Packages in R

Packages in R include the *prefmod* (Hatzinger et al., 2012) which allows training of Bradley-Terry models, *BradleyTerry2* (Turner et al., 2012) which enables modelling Bradley-Terry models with covariates, allowing for the probit link function to be used instead of the logit link function, which converts it into the Thurstone type III model. It also allows using the cauchit link function which instead of using the normal distribution will link to the Cauchy distribution. Other packages in R include the *eba* (Wickelmaier and Schmid, 2004) which allows researchers to train the elimination by aspects model. *Psychotree* (Zeileis et al., 2011) has been developed to fit Bradley-Terry trees. A summary of how these packages have been used can be found in Cattelan (2012). The Plackett-Luce model can be found in *StatRank* (Soufiani and Chen, 2013), *pmr* (Lee and Philip, 2013) and *Plackett-Luce*(Turner et al., 2020). *mlogit* (Croissant, 2020) provides the ability to run nested-logit models.

Packages in Python

We will now mention some of the Python packages available for preference models and will discuss them in light of the qualities we have identified as ideal for the architecture of preference models.

choix (Maystre, 2020) has been developed to host Plackett-Luce, Bradley-Terry Luce and some ranking models. It uses the *scipy* (Virtanen et al., 2020) infrastructure to optimise loss functions, and it has an implementation for MM algorithms in Bradley-Terry. It currently doesn't support fitting these models with covariates, so only the latent strength parameters (λ) are learned. Since covariates are not used in the learning process, it also has no infrastructure to host relational databases. The data needs to be first transformed in a special dictionary format, so it does not have an internal representation of the data types, and therefore also cannot navigate between aggregation and reduction. Finally, it also is not developed in a way that aligns with scikit-learn it does not have *fit* and *predict* methods.

scikit-learn compatible	**Can deal with a relational set-up**	**Has different representations of the preference types**	**Can use aggregation and reduction**	**Clear pipeline for anatomising compositions**
×	×	×	×	×

pylogit "is a Python package for performing maximum likelihood estimation of conditional logit models and similar discrete choice models" Brathwaite (2019). Brathwaite and Walker have researched logit-type models with asymmetries and the Bradley-Terry model is a special case of the general formulation of their proposed model outlined in section 3.3.1 and in more detail in their paper Brathwaite and Walker (2018). Therefore the pylogit package can be used for estimating Bradley-Terry models augmented with covariates. The research behind pylogit is mainly focused on predicting transport choices made by individuals and as such the package is mainly focused on solving the issue of high class imbalances faced in US transport analytics, such as 5% of respondents saying they opt for cycling and 43% opt for driving (Brathwaite and Walker, 2018). Due to this research, the package contains several restrictions which could be useful given the domain for which it was developed but it doesn't generalise well enough for some of the cases of Bradley-Terry we can think of:

Issue	Potential justification in public transport domain	Why this might be a problem for general Bradley-Terry
There is an error thrown when the user tries to train on a dataset where an alternative has never been chosen, coming from the "ensure all wide alt ids are chosen" function in choice_tools.py line 1400	In the case of transport prediction where there are more limited options it might be a bug if one is never chosen by any passengers.	If we were to apply a Bradley-Terry model to sports data there can be teams that have not won a single game in the season.
Stemming from the same function, there is an error thrown when the user tries to predict the results of a set which doesn't contain all entities that have been observed in the training set.	If the user is trying to model public transport choices and one of the choices was not an available alternative to a passenger then it might be wrong to apply a model that was trained on passengers where there were more alternatives available.	In a generic Bradley-Terry case it could happen that the user wants to predict match results where only a subset of the teams are playing instead of all of the teams that have played in the training set. Though this would lead to less accurate predictions.

Further issues at the time of writing which are less due to the original research domain are:

- The package creates a dependency on statsmodels, which is a package that is difficult to install on Windows machines. At the time of writing this could only be installed if the user has a 2015 version of Microsoft Visual C++ installed, which is hard to find in the archives since the most up to date version is 2019 and that doesn't seem to work either. The solution can be found in issue number #4160 (requeijaum, 2019) in the GitHub of statsmodels where a link is posted with the correct download. Nevertheless, it appears as if though pylogit only uses statsmodels for printing a nicer table format of the learned parameters of the models.

Regarding the criteria above, the structure is closer to the scikit-learn interface than *choix*, however, it does not have a *fit* method it has a *fit_mle* method instead which renders it incompatible. In its interface it allows for specifying which covariates are decision maker level, alternative level and choice specific, however it does not allow for these to come from different sources, tables need to be merged together before they get passed onto the models. Pylogit specialises in discrete choice, so it does not host different object types for different types of relations expressed on the alternatives, and for the same reason it also doesn't have an interface that allows for reduction and aggregation. Nor does it contain pipelines.

scikit-learn compatible	Can deal with a relational set-up	Has different representations of the preference types	Can use aggregation and reduction	Clear pipeline for anatomising compositions
×	×	×	×	×

biogeme (Bierlaire, 2003) is a package also designed for modelling discrete choice. It hosts the largest variety of logit-type models as well as implementations using Monte-Carlo simulation. The package is not compatible with scikit-learn, it's programming style is unique, the covariates all need to be initialised before model fitting and all the utility equations for all alternatives in the discrete choice need to be defined manually by writing out explicitly all the covariates involved. The latter can become a very time consuming task when there are lots of alternative and/or lots of covariates to model. There is a version called *PandasBiogeme*(Bierlaire, 2018) which has an internal database representation, however, at the moment it doesn't show any support a relational database set up, although given its current set-up this is something that we can see as a natural extension to the current form of the package. Since it is also specialised on discrete choice, it doesn't host other models nor other data types, therefore, it also does not support reduction and aggregation and there is no evidence of it having pipelines.

scikit-learn compatible	Can deal with a relational set-up	Has different representations of the preference types	Can use aggregation and reduction	Clear pipeline for anatomising compositions
×	×	×	×	×

cs-rank (Pfannschmidt et al., 2019b) is the package that hosts the FATE and FETA models. It is the only package that we are currently aware of that is scikit-learn compatible, and it supports ranking, pairwise comparison and discrete choice models and it can convert these data types internally for reduced models, as we discussed with the FATE and FETA models there is an internal representation of pairwise comparisons for each object which is an internal reduction. cs-rank currently uses the scikit-learn set up, however it cannot be used for relational databases and doesn't have a solution for pipelines that include anatomising compositions.

scikit-learn compatible	Can deal with a relational set-up	Has different representations of the preference types	Can use aggregation and reduction	Clear pipeline for anatomising compositions
✓	×	✓	✓	×

Table 6.3: Summary of Python packages covering aspects for our definition of an ideal interface for preference models

package	scikit-learn compatible	Can deal with a relational set-up	Has different representations of the preference types	Can use aggregation and reduction	Clear pipeline for anatomising compositions
choix	×	×	×	×	×
pylogit	×	×	×	×	×
biogeme	×	×	×	×	×
cs-rank	✓	×	✓	✓	×

6.2 Our contribution: Designing a new Python package that adheres to the ideal properties for supervised preference model packages

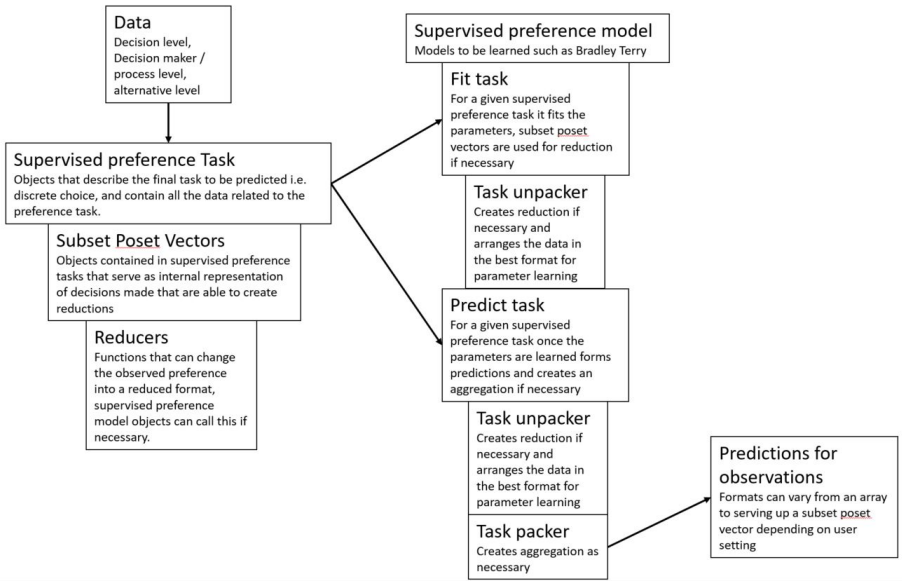
As we can see from table 6.2 there is no package currently in Python that enables practitioners to use all the variety of preference models discussed in this thesis. From table 6.3 we can see that there is no Python package that currently provides an interface that covers all of our criteria for an ideal interface outlined in section 6.1.1. Therefore we have partnered with the creators of *cs-rank* to develop an interface that does adhere to the criteria we have outlined. The working name for this package is *skpref* in this section we describe the proposed architecture for the package. We propose an interface for preference modelling that is scikit-learn compatible and allows the usage of reduction and aggregation similar to *sktime* (Löning et al., 2019) with the added difference that the package will be designed for usage with a relational data structure. We will also build a pipeline that can handle anatomising compositions. We will first present new data objects that we are introducing in Python for representing relations expressed over a set of alternatives, then we will show how these objects can handle reduction and proceed with the architecture for the model tasks to show how aggregations and reductions can be manoeuvred with them. Note that whilst in

this document we capture the rationale behind some important concepts in the package a significant amount of further documentation can be found in GitHub pages and a large part of the contributions of our research is the code that is found in GitHub.

6.2.1 An overview of the key concepts in the package infrastructure

The main new concept introduced in skpref that deviates from scikit-learn is the Supervised Preference Task object. It is designed for the researcher to be able to load their data specifying the level of the data, e.g. decision, decision maker or alternative level. The task object internally creates a representation of the preferences expressed called Subset Poset Vector, which in contains reduction methods. Models themselves will be loaded in the same way as in scikit-learn, however they have additional methods called *fit_task* and *predict_task*, which take as inputs tasks and figure out whether a reduction or aggregation is necessary and take action accordingly in the *task_unpacker* and *task_packer* functions. Figure 6.1 shows a sketch of how these object interact. We will proceed by talking about some of these new concepts in more detail in addition to the supporting documentation referenced above.

Figure 6.1: Showing how some key concepts in skpref interact.



6.2.2 Representing preference relations in Python

Recall that in figure 5.1 we have shown the hierarchy of the relations that can be expressed and we have shown that lower levels in the architecture can be directly mapped from the relation itself. Therefore reductions should be a property of the object that contains the data about the relationship expressed. We are proposing the introduction of two new data objects: the *SubsetPosetVec* and *OrderPosetVec*. For a refresher in Python data types, a short summary can be found in appendix 9.5.1.

We do not expect that users will be explicitly interacting with these data types as they do with the other Python data types. Rather, these are for the benefit of future package contributors and developers to offer the same format of capturing relations, which creates a framework on top of which it will be easier to contribute new algorithms. Classes that contain the models should be able to recognise the type of relation that has been fed to them and should be able to call a reduction technique if necessary to align the relations expressed to those that the model is equipped to solve. For example, every pairwise comparison model will first check if the data provided is a pairwise comparison and if not, it will call the *pairwise_reducer*. Therefore, it is important that reduction methods across *SubsetPosetVec* and *OrderPosetVec* are called the same name.

A *SubsetPosetVec* covers the three data types of pairwise comparisons, discrete choice and subset choice. The common property in these three data types is that everything is expressed in the form of something either being chosen or not and this data type is making use of that. It contains the following attributes:

- *top_input_data*: This is a numpy array that contains sets of all the chosen alternatives
- *boot_input_data*: This is a numpy array that contains sets of all the alternatives that have been not chosen
- *top_size_const*: Is a boolean that indicates whether the length of chosen alternatives is fixed. For example, for discrete choice and pairwise comparisons this will always be *True*, but also such a structural fix can exist in subset selection where a fixed number of items are selected.
- *top_size*: If *top_size_const* is *True* then this takes the value of an integer. In the case of pairwise comparisons and discrete choice this will be one. However, knowing this number for other fixed quantity subset selection can be helpful for aggregation methods.
- *boot_size_const:* Is a boolean, similar to *top_size_const* depicting if the alternatives not chosen are of a fixed size, as they would be in pairwise comparisons.

- *boot_size*: is an integer than when *boot_size_const* is *True* describes how many are left in the boot.

Thus a pairwise comparison without ties is a *SubsetPosetVec* that has *top_size* 1 and *boot_size* 1, a discrete choice is a *SubsetPosetVec* that has *top_size* 1 and *boot_size* greater than 1 or undefined, a subset choice and pairwise comparisons with ties have no fixed top or boot sizes. And once a *SubsetPosetVec* is defined it should be possible to create a reduction from itself based on one of the algorithms described in section 5.1. So *SubsetPosetVec* will have three methods, *discrete_choice_reducer*, *pairwise_reducer* and *classifier_reducer* which take as parameters a string that indicates which type of algorithm to use for reduction and return a *SubsetPosetVec* of the right dimensions. Figure 9.1 shows an example of how the *SubsetPosetVec* object can be currently used to reduce a subset choice into pairwise comparisons.

An *OrderPosetVec* covers partial orders and full orders. The common properties for these is that each alternative will contain a rank. It contains the following attributes:

- *ranked_alternatives*: A Python dictionary where the keys are alternatives and the values are a list of how they were ranked in each of the observations.
- *is_full_order*: Boolean which is *True* when the observations are full order and *False* otherwise.

OrderPosetVec will have the reducers that *SubsetPosetVec* has, although these will work differently, but its important that they have the same name, because models will call these reducers for all types of relations fed into them that aren't on the model's native level. It will also have a method called *partial_order_reducer* which would implement algorithm 1. As noted this would need the researcher to specify the cut off points indicating the ranks that need to be pooled together, which will be in the form of a list of integers, for example: [3,5] would mean to pool together the top 3 as one rank and then the next two observations as one rank.

6.2.3 Supervised Preference Task objects

Tasks are objects for defining preference problems in a way that doesn't require researchers to manually merge all their tables together before loading the data into an algorithm. There is a task for each type of relation that can be expressed and is ultimately driven by what the researcher is trying to predict. Every task will have the following user defined attributes:

- *primary_table*: this is a *pandas DataFrame* that contains the decision level observations, that is, it must contain at least one column which describes the set of alternatives available for the decision and a column which indicates the relationship expressed over the set of alternatives.
- *primary_table_alternatives_names*: is a string indicating the column name of the *DataFrame* that contains the alternatives presented in each decision.
- *primary_table_target_name*: is a string indicating the column where the relation is expressed over the set of alternatives.
- *secondary_tables*: This is a list of *pandas DataFrames* or just one single *pandas DataFrame* that are other tables in a relational set up. The most likely ones to have here would be alternative level information and/or decision maker / decision process level information, however, we do not want to restrict the set up to just three distinct relational tables, therefore this field allows for a list of tables to be given.
- *secondary_table_to_decision_table_link*: is a dictionary or a list of dictionaries that describe what column in the secondary table serves as a lookup in the decision level table.
- *features_to_use*: is a list of strings indicating what are the columns that should be used as covariates in the analysis in case the researcher doesn't intend for all columns in all the tables (except for columns that are used for merging tables together) to be used for prediction.

There will be a task for each kind of prediction (full order, partial order, subset choice, discrete choice and pairwise comparison). Some of the tasks might have additional user defined attributes, based on the likely structure of the data. For example, the *PairwiseComparisonTask* will have an additional attribute called *target_column_correspondence* which will be used in case the data is stored in a way where the table has the two alternatives as two columns and then a third column as a 1/0 flag indicating that the alternative in one of the two columns gets chosen or not. *target_column_correspondence* will be the name of the column in which the alternative is chosen when the *decision_table_target_name* is 1 and *decision_table_alternatives_names* will be a list of strings in that case. Tasks that consider ordered data will account for the format that sometimes the alternatives are presented as lists of columns and the value populating that column is the rank of the alternative.

Once the data is fed into the tasks, they will create the internal data types described in the previous section. Models will be then working with these internal datatypes for relations expressed which will be homogeneous for each model and they will create temporary pooled tables if necessary.

6.2.4 How model objects will interact with task objects for fitting and predicting

Models in skpref will be objects just like in *scikit-learn* and will be *scikit-learn* compatible having the same functions with some enhancements.

Normally in *scikit-learn* the *fit* method takes the parameters X and y for the matrix of covariates and the vector of ground truths respectively. In *skpref* the *fit* and *predict* functions are intended mostly for internal use to align the back-end with *scikit-learn* enabling the usage of some existing infrastructure in *scikit-learn* like *GridSearch*. The function for fitting and predicting that users will be expected to use in *skpref* is *fit_task*, *predict_task* and similarly to *predict_proba* in *scikit-learn*, *skpref* will have a *predict_proba_task* to predict probabilities rather than point estimates. Instead of taking the parameters of X and and y these will take a task.

This enables for every model to query the task presented, so that when a model like Bradley-Terry is presented with a subset choice task it knows that it needs to run a reduction by invoking the reducer of the data type contained in the task. We will call this process *task unpacking*. Models will be categorised by *types* according to their level (full order, partial order, subset choice, discrete choice, pairwise comparison and binary reduction) and each model type will have an internal *task_unpacker* and a *task_packer* method for aggregation. The former will run before the internal version of *fit* and the latter will run before *predict_task* or *predict_proba_task*. Once these are defined for a model type, all models of the same type can inherit it, so the Bradley-Terry and Thurstone models will have the same *task_packer* and *task_unpacker*. This means that once these are defined future contributors can contribute another model of the same type that will be able to handle reduction and aggregation without any additional effort and any improvement to the aggregation and reduction contributed by someone is immediately applied to all models. This way *skpref* can become a hub of the best practice for reduction and aggregation, where future contributors can iterate on theoretically improved and/or more computationally efficient methods for aggregation and reduction. The generic aggregator described in section 5.3.2 is currently implemented and is inherited by all probabilistic pairwise comparison models, whilst the Bradley-Terry to Luce aggregator, also described in section 5.3.2 is a special additional option only for Bradley-Terry models.

We can now see that whilst learning how to load up tasks, is perhaps something that users of *skpref* might need to learn and get used to, once the model task object is defined, for the front-end-user reduction and aggregation will be just as simple as fitting a model, saving time on having to combine the normalised data manually and also from having to manually

create the reductions and aggregations. We will allow researchers to call the *fit* and *predict* functions themselves, if for some reason the task architecture is not the way they would like to work, but in this case they will have create the reduced tables themselves.

6.2.5 Visualising learned model parameters

Additionally we're planning to improve the querying of coefficients. Once a *scikit-learn* model is trained it is possible to query its coefficients via using a method called *coef_* and *intercept_* which returns a *numpy* array of floats and one float respectively. In the *skpref* set-up the name of the covariates is given through the *pandas DataFrame* so there is an opportunity to return the coefficients associated to the names of the covariates in a way that it is clearer which covariate the coefficient belongs to, similar to how it is done in *statsmodels* (Seabold and Perktold, 2010).

Statsmodels calls out the name of the covariates in the results summary which makes it less user error prone compared to what scikit-learn does which returns a *numpy* array of floats and relying on the user to correctly remember the order in which columns have been passed into the model. Furthermore, with the relational set up that we propose, returning just an array of floats would not even be possible, because the joining of the tables happen in the back end of the functions and user would not know which float refers to which covariate. Finally, almost all the models we have described in chapter 3 have latent strength variables for the alternatives and these should also be queried in a way that is is clear which latent strength parameter belongs to which alternative, so they should be presented together with the alternative names. Therefore, we needed to work out a significantly better way of querying model parameters than scikit-learn. An example of how skpref displays the latent strength parameters learned for each entity can be seen in figure 6.2. In figure 6.3 we can see how the statsmodels representation of covariates gives a nice framework to show the coefficients of covariates and alternative latent strength parameters.

Figure 6.2: skpref showing the latent strength parameters in a Bradley-Terry model fitted for NBA teams in the 2016 season.

	entity	learned_strength
0	Atlanta Hawks	0.047522
1	Boston Celtics	0.580896
2	Brooklyn Nets	-1.178393
3	Charlotte Hornets	-0.278154

Figure 6.3: skpref showing the coefficients for a Luce model, on the Swissmetro data where the covariates are Cost and Travel Time, and the alternatives are Car, Swissmetro and Train. Note this output is currently being generated by the *pylogit* code which uses *statmodels* and within *skpref* this is not yet available for Bradley-Terry models that use no covariates, as that is built upon the *choix* package.

Multinomial Logit Model Regression Results

Dep. Variable:	CHOICE	**No. Observations:**	8,878
Model:	Multinomial Logit Model	**Df Residuals:**	8,873
Method:	MLE	**Df Model:**	5
Date:	Wed, 01 Mar 2023	**Pseudo R-squ.:**	0.226
Time:	16:11:21	**Pseudo R-bar-squ.:**	0.225
AIC:	9,534.703	**Log-Likelihood:**	-4,762.351
BIC:	9,570.159	**LL-Null:**	-6,153.761

	coef	std err	z	P>|z|	[0.025	0.975]
Cost	0.0003	3.11e-05	8.763	0.000	0.000	0.000
Travel Time	-0.0116	0.001	-22.819	0.000	-0.013	-0.011
Car	0.3031	0.046	6.635	0.000	0.214	0.393
Swiss Metro	0.1378	0.048	2.870	0.004	0.044	0.232
Train	-0.4409	0.048	-9.159	0.000	-0.535	-0.347

6.2.6 Grid Search

In order to allow for grid-searching and pipelines to also have the *fit_task*, *predict_task*, *predict_proba_task* methods *skpref* will create a wrapper around the existing *scikit-learn* objects and enhance them with these functions. This does mean that these functions will need to be imported from *skpref* instead of being imported from *scikit-learn*. These functions will rely heavily on the state of the art implementations in *scikit-learn* and every time *scikit-learn* gets updated with better more efficient practice *skpref* will benefit from that too.

Figure 6.4: An example of how *GridSearchCV()* works at the time of writing in skpref, note that this cell can be found amongst the example notebooks in skpref's documentation, however since that is a live document it might be subject to change. For this cell to work, the *GridSearchCV()*, *BradleyTerry()* need to have been imported and the choice task defined which is called *NBA_results_task_train* in this example.

```
to_tune = {'alpha': [1, 2, 4], 'method': ['BFGS']}
gs_bt = GridSearchCV(BradleyTerry(), to_tune,  cv=3, scoring='neg_log_loss')
gs_bt.fit_task(NBA_results_task_train)
gs_bt.inspect_results()
```

```
The model with the best parameters was:
BradleyTerry(alpha=2, method='BFGS')
With a score of -0.6265008194657992
All the trials results summarised in descending score
   alpha method  mean_test_score
1      2   BFGS        -0.626501
0      1   BFGS        -0.626742
2      4   BFGS        -0.628853
```

6.2.7 Pipelines

We would like to develop anatomising compositional pipelines in *skpref*, which is the idea of the *scikit-learn* pipeline evolved to handle more complex compositions similar to how it's done in Löning and Király (2020). In preference modelling compositions would be in the form of a series of decisions that need to be made by the decision makers. The task based set up is ideal for this information to be passed onto the next model, since for each task the set of alternatives needs to be defined by the *decision_table_alternatives_names*. In one of the examples we have provided in section 4.5 one model would predict the *department* in the dunnhumby dataset (see 9.4.1) that is being purchased and then the next model would predict the *sub_commodity_desc* that will be purchased. The second model in the composition pipeline will need the *department* column to exist to apply the necessary filters in training and predicting. In an anatomising composition pipeline the task of the second model will need to query the task of the first model to determine what is the level at which the first model is being applied.

6.2.8 Example usage of some existing code

Please refer on section 9.6 to two jupyter notebooks that illustrate how the package works.

6.2.9 State of the package and potential future work

In this section we will first highlight what has been implemented so far in the skpref package and what has not been implemented yet.

Table 6.4: Models implemented in skpref at time of thesis submission

Model	Implemented	Packages used
Ability to create classification reduction with sklearn models	✓	any skpref compatible classifier
Bradley Terry	✓	*choix* for no covariates and *pylogit* when covariates are used
Thurstone		
Plackett-Luce		
Luce		
FATE		
FETA		
Coherency Driven Model		
Nested Logit Models		
Elimination by Aspects		
Support Vector Machines		
ListNet		
Bradley Terry trees		
Plackett-Luce trees		

Table 6.5: The functions discussed in this thesis implemented in skpref at the time of submission.

Functionality	**Implemented**
OrderPosetVec	
SubsetPosetVec	✓
ClassificationReducer	✓
PairwiseComparisonTask	✓
ChoiceTask	✓
SubsetChoiceTask	
PartialOrderTask	
FullOrderTask	
Pipelines	
GridSearchCV	✓
Bradley Terry to Luce aggregator (see section 5.3.2)	✓
Generic PC to DC Probabilistic aggregator (see section 5.3.2)	✓

We currently have developed the capability to create reductions from subset choice to pairwise comparisons and we have interfaced with some models in *choix* and *pylogit* in the back-end to run the predictions, but created wrappers that make them scikit-learn compatible and fit into the task architecture we believe is optimal for preference modelling.

Chapter 7

Further research questions

In this chapter we will present research questions that we have identified during the course of this MPhil but have not had the time to explore beyond a superficial idea. In contrast with the contributions outlined in section 6.2 (designing a package in Python for applying supervised preference tasks) and section 5.3 (probabilistic aggregation from pairwise comparison models to discrete choice) which were topics that we gave more attention to.

Some further ideas we had that with more research time me might have explored are:

- Section 7.1: creating new types of tree based preference models.
- Section 7.2: using anatomising composition to test hypothesis about purchasing patterns in the retail industry.
- Section 7.3: using numerical optimisation techniques on normalised data in preference models.
- Section 7.4: points at interesting data sets that have not been mentioned in this thesis so far but could be useful for exploring preference models.

We believe that the skpref architecture and platform is ideal to develop these ideas further in an experimental setting.

7.1 Tree based algorithms for supervised preference models

There is a lot of potential for contributing new methods in the tree based space of preference models. In section 3.3.7 we have shown an existing implementation of a Bradley-Terry tree. In other fields of machine learning, creating ensembling models from trees has proven a very powerful predictive technique, we believe that these extensions have not yet been developed for Bradley Terry trees. Currently we have identified two potential contributions for extending the tree based techniques in preference models. Here present them in increasing complexity.

1. Where there's a tree there should be a forest, but as far as our search has revealed to date there is no implementation nor is there any research of results of Bradley Terry forests, though Turner et al. (2020) have hinted in their paper that random forests could be developed with the Bradley-Terry trees method, we are unaware of a published implementation. Decision trees are not considered to be very accurate and useful for prediction. Something that works much better than decision trees for prediction is a method called random forests (Breiman, 2001). In simple terms, random forests select a subset of the data and a subset of the features and fit a decision tree on this. Then it repeats the process on a different subset of the data fitting another tree and so on until hundreds and sometimes even thousands of trees are fit. Then the predictions from these trees are averaged to give the final prediction. Random Forests are a powerful technique for predictive modelling and this technique could be added as an easily available tool to researchers in the field through our machine learning software interface discussed in section 6.2. It could then be used on real data for predictive modelling tasks to see if random forests can bring as much increase in predictive accuracy in preference models as they have done in classification and regression techniques.

2. "Boosting is one of the most powerful learning ideas introduced in the last twenty years" (Hastie et al., 2009). The core idea behind boosting when it comes to weak learners such as decision trees in a random forest, is to weight some of the more accurate decision trees more than the less accurate ones. According to Hastie et al. (2009) the most popular boosting algorithm is AdaBoost which is accredited to Freund and Schapire (1997). The most recent boosting technique is called XGBoost which has proven a very powerful predictive machine learning algorithm (Chen et al., 2015). There is perhaps potential in creating a Bradley-Terry algorithm that uses AdaBoost. Since both random forests and gradient boosting have proven to be powerful predictive modelling techniques, we believe working these out will enable researchers to use more accurate models. Catching

Bradley-Terry trees up with boosting might prove very useful for these types of models as there are other more modern boosted algorithms such as XGBoost (Chen and Guestrin, 2016) or LightGBM (Ke et al., 2017).

7.2 Anatomising composition techniques and their use for predicting subset choices

Another interesting project would to be test several hypotheses for shopping behaviours using anatomising composition on the dunnhumby dataset (see 2.5.2). Anatomising compositions would be suitable to specifically answering the following questions:

- At which point of the shoppers' journey in a store does the shopping resemble more an independent decision from the available alternatives vs. where alternatives influence the decisions of people? This is an important question for planning the ranging in a store, because at some point we are suggesting that alternatives begin to compete with each other, so the impact of adding new alternatives is different when these don't reduce the probability of buying existing alternatives vs. when they do. Assuming shoppers make three decisions: Which aisles should I purchase products from (e.g. Bakery, Fishmonger, Fresh Fruits and Vegetables)? $\rightarrow$ Which subcategory should I purchase in the aisle (e.g. Bread, Baguette or Doughnut)? $\rightarrow$ Which product should I purchase within this subcategory (e.g. Hovis White 300G, Hovis Brown 300G)?
  - Do shoppers make a binary decision about purchasing products of each aisle independently of other aisles available, or do other aisles available in the store impact the decisions shoppers take? If the former is true then a binary classifier would be just as accurate on predicting shopper behaviour as a preference model. Here our working hypothesis is that there would be no significant difference in performance between a binary classifier and a preference model.
  - Do shoppers make a decision about purchasing products from a sub-category (e.g. Bread or Baguette in the Bakery aisle) independently of what sub-categories are available, or does the existence of one sub-category change the likelihood of purchasing another one? This is the same as the above question, only on a different grouping of products. Here preference models might begin to perform a little better than binary models.
  - Finally same question as the above just on a product level inside a sub category. Here we would expect preference models to perform better than binary models.

- See how anatomising composition can be used to create and suggest new methods of predicting subset selection, which is a field that currently doesn't have many solutions. We would start by building a model that predicts the number of distinct products purchased by a shopper in a subcategory of products. The number of products predicted here will determine the number of discrete choice models that will be run subsequently predicting the alternatives purchased, with the chosen product being removed from the available alternatives for the next model. In the case where logit-type models are being used this would be the equivalent of a Plackett-Luce model. For example, if a model predicts 2 unique items from the set of products $\{a, b, c, d\}$ then we first run a discrete choice model on this set, say it predicts that b is chosen, then the next discrete choice model would be considering alternatives $\{a, c, d\}$.

  This can have different flavours also, for example, if a discrete choice model is being learned using the power-set of the original list of alternatives as the actual alternatives then the prediction can be done in one step by narrowing down the alternatives to only sets that have the cardinality of the predicted number of alternatives from the previous model. FATE and FETA also have the ability to predict subsets, so we would like to compare how such a anatomising composition might work compared to the FATE and FETA algorithms.

- What anatomising composition is best for modelling the items purchased in a basket? Our working hypothesis is that to the approach above we add a model that predicts the number of units that will be purchased for each item. However since there are five steps in the composition, it would be interesting to see if skipping some improves the accuracy of prediction, or rather what combination of these compositions seems to work best.

7.3 Speeding up gradient descent on Bradley-Terry models using the normalised data set up

Application of Numerical Optimisation in Normalised data has been recently developed for Support Vector Machines (Abo-Khamis et al., 2020). A potential research question would be how to apply this for preference models and MM algorithms.

Factorised learning can lead to significant benefits when it comes to the computational speed of model fitting, and as we have seen in section 2.3.1 a normalised dataset is one of the most sensible and widespread forms of storing preference data, meaning that it is very likely

that advancements of this field would allow researchers to train models in a less costly way allowing more data to be used in training.

This technique is complementary to the question of reduction, which in general creates longer observation level tables by transforming the data. Having a faster way to fit models would be beneficial for practitioners looking to use reduction techniques that tend to create longer tables.

7.4 Further datasets

For full ranking research the Formula 1 Kaggle (2017) dataset could be considered, which includes the full rank of all drivers that participated in races between 1950 and 2017.

For a partially ordered data the London Assembly (2016) dataset can be considered, which shows the results of the London Mayor election results of 2016, which used a voting system where people have selected their first and second choice candidates.

For discrete choice data the Kaggle (2016) dataset for Expedia hotel bookings could be an interesting source. This data shows people searching for holidays and flags which hotel type has been booked for a search. Strictly speaking, this is still not a proper discrete choice domain, since it is possible to book several hotels for one search, however, we would expect that this data is still closest to a discrete choice for the vast majority of people, we will have to investigate this dataset more closely, to confirm this and decide if any filters are needed.

Chapter 8

Conclusion

Supervised preference models are an essential tool for creating predictive and conceptual models of situations where a decision maker or decision process expresses a relation over a list of alternatives. One of the advantages they offer is that through their mathematical set-up they allow to make assumptions of varying complexity regarding how the availability of certain alternatives impact the likelihood of the outcome of the relation expressed.

In this thesis, we have clearly distinguished between five different types of relations that can be expressed in preference models: pairwise comparisons, discrete choice, subset choice, partial order and full order. We conceptualised the components required for preference models: a list of alternatives, a type of relation to be expressed over the lists of alternatives, a question with which regards the relations are expressed and the decision maker / decision process that expresses such a relationship. We have expressed mathematical notations that allows us and future researchers to discuss these models in the same context.

After an exposition of a series of supervised learning preference models, their assumptions and how they are fitted to the data, we presented how they fit into today's modern machine learning framework including model validation, machine learning pipelines and model reduction and aggregation. We explored open source packages researchers have available to work with supervised preference models, and evaluated them based on four criteria:

- Are they compatible with scikit-learn code?
- Are they designed to work well on a relational data set up?
- Can different types of preferences be easily represented in them?

- Does it allow an easy way for researchers to use reduction and aggregation?
- Does it allow for users to create anatomising compositions?

We concluded that from the packages surveyed there were none that accounted for all of these properties that are specific to preference models and proposed an architecture that would solve for these objectives. We began working on these models, created a prototype package called skpref, which currently contains a Bradley-Terry model that can augmented with covariates as well as reductions to scikit-learn classifiers in the interface. Although for this work to be complete more models need to be added into them, such as the ones that have been highlighted in this thesis.

We have presented further topics where we identified a potential gap in knowledge where further research can be carried out: creating random forests from Bradley-Terry trees, creating probabilistic aggregations from pairwise comparisons to discrete choice, experimentation using anatomising compositions, using numerical optimisation on normalised data in preference models.

For the case of probabilistic aggregations, we have identified a similarity to the aggregation problem pairwise coupling solves and have suggested ways in which the pairwise coupling strategies can be adjusted for the aggregation of pairwise comparison preference models to discrete choice models. Testing these to see how they perform on real and simulated data would be an interesting next step for which the skpref package offers an ideal infrastructure. The longer term view would aim at extending any new understanding to the problem of aggregating from discrete choices to subset choices.

Hopefully this thesis serves as an exposition of some interesting concepts and offers some new ideas for working with supervised preference models and contributed towards the reader's further understanding of the field.

Chapter 9

Appendix

9.1 Proofs

9.1.1 Proof of equation 3.1

Lemma 9.1.1. *Definition 6 states:*

$$\frac{P(Y_i = c(c, \{a, c\}))}{P(Y_i = c(a, \{a, c\}))} = \frac{P(Y_i = c(c, B))}{P(Y_i = c(a, B))} \tag{9.1}$$

let $B = \{a, b, c\}$. *Given*

$$P(Y_i = c(c, \{a, c\})) + P(Y_i = c(a, \{a, c\})) = 1 \tag{9.2}$$

show that

$$P(Y_i = c(c, \{a, c\})) = \frac{P(Y_i = c(c, B))}{P(Y_i = c(c, B)) + P(Y_i = c(a, B))}$$

Proof. In the interest further brevity in notation let $P(Y_i = c(a, X))$ be denoted by $P(a \leftharpoondown X)$

$$\text{from 9.1 we have: } \frac{P(c \leftharpoondown \{a,c\})}{P(a \leftharpoondown \{a,c\})} = \frac{P(c \leftharpoondown B)}{P(a \leftharpoondown B)}$$

$$P(c \leftharpoondown \{a,c\}) = \frac{P(c \leftharpoondown B)P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)}$$

$$\text{from 9.2 we have: } 1 - P(a \leftharpoondown \{a,c\}) = \frac{P(c \leftharpoondown B)P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)}$$

$$1 = \frac{P(c \leftharpoondown B)P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)} + P(a \leftharpoondown \{a,c\})$$

$$P(c \leftharpoondown B) = \frac{P(c \leftharpoondown B)^2 P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)} + P(a \leftharpoondown \{a,c\})P(c \leftharpoondown B)$$

$$= \frac{P(c \leftharpoondown B)^2 P(a \leftharpoondown \{a,c\}) + P(a \leftharpoondown \{a,c\})P(c \leftharpoondown B)P(a \leftharpoondown B)}{P(a \leftharpoondown B)}$$

$$= (P(c \leftharpoondown B) + P(a \leftharpoondown B))\frac{P(c \leftharpoondown B)P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)}$$

$$\frac{P(c \leftharpoondown B)}{P(c \leftharpoondown B) + P(a \leftharpoondown B)} = \frac{P(c \leftharpoondown B)P(a \leftharpoondown \{a,c\})}{P(a \leftharpoondown B)} = P(c \leftharpoondown \{a,c\})$$

■

9.1.2 Proof of vanilla Bradley-Terry update under method of steepest descent equation 3.5

Lemma 9.1.2. *Show that the derivative of the loss function in Bradley-Terry as shown in equation 3.4 is the update term as shown in equation 3.5.*

Proof. Let:

$$f(x) = ln(x)$$

$$k(x) = ln(1 - x)$$

$$g(x) = \frac{1}{1 + e^{-x}}$$

$$m(\lambda) = \lambda_i - \lambda_j$$

$$Y_k = \begin{cases} 1 \text{ if } i \succ j \\ 0 \text{ otherwise} \end{cases}$$

The loss function then reads as:

$$Y_k f(g(m(\lambda))) + (1 - Y_k)k(g(m(\lambda)))$$

The derivatives yield:

$$\frac{df(x)}{dg(x)} = \frac{1}{g(x)}$$

$$\frac{dk(x)}{dg(x)} = \frac{-1}{1 - g(x)}$$

$$\frac{dg(x)}{dm(\lambda)} = \frac{dg(m(\lambda))}{dm(\lambda)}(1 + e^{-m(\lambda)})^{-1} = (-1)(1 + e^{-m(\lambda)})^{-2}(e^{-m(\lambda)})(-1) = \frac{e^{-m(\lambda)}}{(1 + e^{-m(\lambda)})^2}$$

$$\frac{dm(\lambda)}{d\lambda_i} = 1$$

$$\frac{dm(\lambda)}{d\lambda_j} = -1$$

Putting these together yield:

$$\begin{aligned}
\frac{df(x)}{d\lambda_i} &= (Y_k)\frac{1}{\frac{1}{1+e^{-(\lambda_i-\lambda_j)}}}\frac{e^{-(\lambda_i-\lambda_j)}}{(1+e^{-(\lambda_i-\lambda_j)})^2}(1) + (1-Y_k)\left(\frac{-1}{1-\frac{1}{1+e^{-(\lambda_i-\lambda_j)}}}\right)\frac{e^{-(\lambda_i-\lambda_j)}}{(1+e^{-(\lambda_i-\lambda_j)})^2}(1) \\
&= (Y_k)(1+e^{-(\lambda_i-\lambda_j)})\frac{e^{-(\lambda_i-\lambda_j)}}{(1+e^{-(\lambda_i-\lambda_j)})^2} + (1-Y_k)\left(\frac{-\left(1+e^{-(\lambda_i-\lambda_j)}\right)}{1+e^{-(\lambda_i-\lambda_j)}-1}\right)\frac{e^{-(\lambda_i-\lambda_j)}}{(1+e^{-(\lambda_i-\lambda_j)})^2} \\
&= (Y_k)\frac{e^{-(\lambda_i-\lambda_j)}}{(1+e^{-(\lambda_i-\lambda_j)})} + (1-Y_k)\frac{-1}{(1+e^{-(\lambda_i-\lambda_j)})} \\
&= (Y_k)\frac{e^{-\lambda_i+\lambda_j}}{(1+e^{-\lambda_i+\lambda_j})} + (1-Y_k)\frac{-1}{(1+e^{-\lambda_i+\lambda_j})} \\
&= (Y_k)\frac{\frac{e^{\lambda_j}}{e^{\lambda_i}}}{1+\frac{e^{\lambda_j}}{e^{\lambda_i}}} + (1-Y_k)\frac{-1}{1+\frac{e^{\lambda_j}}{e^{\lambda_i}}} \\
&= (Y_k)\frac{e^{\lambda_j}}{e^{\lambda_i}+e^{\lambda_j}} + (1-Y_k)\frac{-e^{\lambda_i}}{e^{\lambda_i}+e^{\lambda_j}}
\end{aligned}$$

■

9.2 Tables

Table 9.1: Commonly used loss and accuracy function for classifiers

loss/accuracy function name	**equation where $y \in \{0,1\}^n, \hat{y} \in \mathbb{R}^n_{\in[0,1]}$ and $y_i = y[i]$ and $\hat{y}_i = \hat{y}[i]$**
Cross entropy loss	$f_{CE}(\hat{y}, y) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$
True positives	$f_{TP}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i \geq t \cap y_i = 1]]$
False negatives	$f_{FN}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i < t \cap y_i = 1]]$
False positives	$f_{FP}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i \geq t \cap y_i = 0]]$
True negatives	$f_{TN}(\hat{y}, y, t) = \frac{1}{n}\sum_{i=1}^{n}[[\hat{y}_i < t \cap y_i = 0]]$
Accuracy	$f_{Acc}(\hat{y}, y, t) = \frac{f_{TP}(\hat{y},y,t)+f_{TN}(\hat{y},y,t)}{f_{TP}(\hat{y},y,t)+f_{TN}(\hat{y},y,t)+f_{FN}(\hat{y},y,t)+f_{FP}(\hat{y},y,t)}$
False positive rate	$f_{FPR}(\hat{y}, y, t) = \frac{f_{FP}(\hat{y},y,t)}{f_{FP}(\hat{y},y,t)+f_{TN}(\hat{y},y,t)}$
Specificity	$f_{Spec}(\hat{y}, y, t) = \frac{f_{TN}(\hat{y},y,t)}{f_{TN}(\hat{y},y,t)+f_{FP}(\hat{y},y,t)}$
Recall	$f_{RE}(\hat{y}, y, t) = \frac{f_{TP}(\hat{y},y,t)}{f_{TP}(\hat{y},y,t)+f_{FN}(\hat{y},y,t)}$
Precision	$f_{PR}(\hat{y}, y, t) = \frac{f_{TP}(\hat{y},y,t)}{f_{TP}(\hat{y},y,t)+f_{FP}(\hat{y},y,t)}$
F_1 score	$f_{F_1}(\hat{y}, y, t) = \frac{2 f_{PR}(\hat{y},y,t) f_{RE}(\hat{y},y,t)}{f_{PR}(\hat{y},y,t)+f_{RE}(\hat{y},y,t)}$
F-beta-score	$f_{F_\beta}(\hat{y}, y, \beta) = \frac{(1+\beta^2)\times f_{\text{precision}}(\hat{y},y)\times f_{\text{recall}}(\hat{y},y)}{\beta^2 f_{\text{precision}}(\hat{y},y)+f_{\text{recall}}(\hat{y},y)}$
Mean Absolute Error	$f_{MAE} = \frac{1}{n}\sum_{i=1}^{n}[\lvert\hat{y}_i - y_i\rvert]$
Root Mean Squared Error	$f_{RMSE} = \sqrt{\left(\frac{1}{n}\sum_{i=1}^{n}[(\hat{y}_i - y_i)^2]\right)}$
G-means	$f_{G\mu}(\hat{y}, y, t) = \sqrt{f_{RE}(\hat{y}, y, t) \times f_{Spec}(\hat{y}, y, t)}$

9.3 Figures

Figure 9.1: Example of how reduction works currently with SubsetPosetVec

[1]:
```
# import skpref package
import sys
sys.path.insert(0, "../..")
from skpref.data_processing import SubsetPosetVec
# import packages neede to create example data
import numpy as np
import pandas as pd
```

[2]:
```
# Create the example data
subset_choice_example_table = pd.DataFrame(
    {'choice': np.array([np.array(['a']), np.array(['a', 'd'])]),
     'rejection':np.array([np.array(['b', 'c']), np.array(['f'])])})
subset_choice_example_table
```

[2]:

	choice	rejection
0	[a]	[b, c]
1	[a, d]	[f]

[3]:
```
# Create a SubsetPosetVec
subset_vec = SubsetPosetVec(
    top_input_data=subset_choice_example_table.choice,
    boot_input_data=subset_choice_example_table.rejection)
```

[4]:
```
# Example of reducing to pairwise comparisons
subset_vec.pairwise_reducer()
```

[4]:

	observation	alt1	alt2	alt1_top
0	0	a	b	1
1	0	a	c	1
2	1	a	f	1
3	1	d	f	1

[5]:
```
# Example to reducing to binary classifier
subset_vec.classifier_reducer()
```

[5]:

	observation	alternative	chosen
0	0	a	1
1	0	b	0
2	0	c	0
3	1	a	1
4	1	d	1
5	1	f	0

9.4 Details of datasets

9.4.1 The dunnhumby dataset

The transactional data contains the following columns:

- **household_key**: used to look up household information in the customer level table (which is a decision maker/process level table in our definitions).
- **basket_id**: unique key for each basket helps separate out decisions; enabling us to can treat each basket as a subset choice relation.
- **day**: time variable that starts with 1 and ends with 711, indicating that there is just under two years of data available.
- **product_id**: unique identifier for each product.
- **quantity**: number of units purchased in that basket for that product.
- **sales_value**: total sales for that product in the basket, in the form of quantity multiplied by the price (currency is not mentioned in the data, but based on the magnitudes ranging from 0 to 840 with an average of 3.1, we can imagine that this could realistically be pounds, dollars or euros).
- **store_id**: store in which purchases have happened, this is important for determining the alternatives available to shoppers, it can be assumed that the alternatives were the list of products normally sold in the store or sold in the store on that day.
- **retail_disc**: discount applied by retailer.
- **trans_time**: time stamp of transaction.
- **week_no**: week grouping goes from 1 to 102.
- **coupon_disc**: discount applied by coupons held by the shopper.

The product dataset contains just over 92 thousand observations with the following information:

- **product_id**: key we can use to match to product observed in the transactional table.
- **manufacturer**: unique code to identify whether a product is made by the same manufacturer

- **department**: a grouping of certain products containing 44 unique values example values contains: "GROCERY", "PASTRY", "MEAT-PCKGD"
- **brand**: labels whether a product is a national brand or a private label by the retailer.
- **commodity_desc**: a more granular grouping of products containing 308 unique values such as: "BREAD", "FRUIT - SHELF STABLE", "COOKIES/CONES".
- **sub_commodity_desc**: most granular grouping of products containing 2383 unique values such as "ICE - CRUSHED/CUBED ", "BREAD:ITALIAN/FRENCH", "APPLE SAUCE"
- **curr_size_of_product**: product size given in pounds and ounces

The customer level data contains the following information on 801 household_id, it should be noted that this is less than the full amount of 2500 households, showing that this data hasn't been volunteered by the majority of shoppers:

- **household_key**: way to look up values in the transactional table.
- **age_desc**: description of which age bucket the household is in available buckets are: 19-24, 25-34, 35-44, 45-54, 55-64, 65+.
- **marital_status_code**: takes values "A", "U", "B", it is unclear what marital status these refer to.
- **income_desc**: shows the income of shoppers according to some brackets: Under 15k, 15-24k 25-34k, 35-49k, 50-74k, 75-99k, 100-124k, 125-149k, 150-174k, 175-199k, 200-249k, 250k+ .
- **homeowner_desc**: indicates the housing arrangements of the shopper categories are: "Homeowner", "Unknown", "Renter", "Probable Renter", "Probable Owner".
- **hh_comp_desc**: indicates what the household of the shopper is like, it can be: "2 Adults No Kids", "2 Adults Kids", "Single Female", "Unknown", "Single Male", "1 Adult Kids".
- **household_size_desc**: indicates how many people are in the household.
- **kid_category_desc**: indicates how many children there are in the household.

There is also promotional data available, however, we will only focus on these datasets for now, at a later point we may want to bring in more information.

9.4.2 The NCAA dataset

The format of the games data (decision level data) is as follows:

- **Season**: the year in which the season began. There are slightly new players each season in the teams so it might make sense to do analysis splitting by season. Due to this being college and students generally leave within 3-5 years of joining a university, it wouldn't make much sense to pool more than 3 seasons of data together for any year.
- **DayNum**: matchday
- **WTeamID**: the id number of the winning team
- **LTeamID**: the id number of the losing team
- **[W/L]Score**: the number of points the winning team scored when the first letter is W, the number of points the losing team scored when the first letter is L. Note that these are two different columns, one called *LTeam* the other called *WTeam* we're justing noting them in one bullet point for brevity, we will follow noting all columns that are of this format in this way. Note that basketball games cannot tie, when the result is tied at the end of the fourth (last official) quarter then the game goes into overtimes until one of the teams is ahead at the end of the overtime.
- **WLoc**: Whether the winning team was playing home "H", away "A" or on neutral grounds "N".
- **NumOT**: Number of overtimes.
- **[W/L] FGM**: Number of field-goals made
- **[W/L] FGA**: Number of shots taken (field goals attempted)
- **[W/L]FG[M/A] 3**: Number of 3 pointers made / attempted. Note that these are now in 4 columns: Winning team number of 3 pointers made, winning team number of 3 pointers attempted and the same 2 for the losing team.
- **[W/L]FT[M/A]**: Free throws made / attempted
- **[W/L]OR]**: Number of offensive rebounds
- **[W/L]DR**: Number of defensive rebounds
- **[W/L]Ast**: Number of assists
- **[W/L]TO**: Number of turnovers

- **[W/L]Stl**: Number of steals
- **[W/L]Blk**: Number of blocks
- **[W/L]PF**: Number of personal fouls

9.5 Additional information

9.5.1 Data types in Python

Now that we have introduced objects in Python we would like to dedicate a short section for explaining data objects in Python, since throughout the rest of this document we will refer to some of these.

In Python data types are also objects. There are a couple of data objects that are called built-in, this means that they are available as soon as the user starts the Python program and doesn't have to be loaded separately. Loading an object separately in Python is also called importing. To import an object in Python it needs to be already saved in the computer. The built-in data types relevant to this report are: floats, integers, strings, booleans, lists, sets and dictionaries.

- Floats are numbers, they have certain properties that allow them to be mathematically manipulated (such as using in addition, subtraction, multiplication and division). Examples are: 3.14, 2.72, 0.14.
- Integers are whole numbers, for example: 1, 2, 3.
- Strings have different properties than floats and integers; they cannot be used in a mathematical operation, however, they can be used for other operations such as concatenation (joining of different strings) and duplication (repeating the string). Examples are: "Apple", "Orange", "supercalifragilisticexpialidocious"
- Booleans are objects which can only contain one of two values *True* or *False*.
- Lists are a group of Python objects a list can contain objects of any type. For example: [1, 'a', True, 5.25]
- Sets are a distinct group of Python objects, sets can contain a group of any combination of the above mentioned types except for lists and they have other operations, such as

intersections and unions, which return sets of certain properties. However, sets can contain only unique alternatives, for example, these two sets are identical {'a', 'b', 'c'} = {'a', 'a', 'b', 'c'} whereas if these were lists instead of sets, they would not be identical ['a', 'b', 'c'] $\neq$ ['a', 'a', 'b', 'c'].

- Python dictionaries have two components keys and values. Each key has a value, which can also be an object for example a list of integers. For example: {'a': [1,2,3], 'b': [4, 5, 6]}

Finally we would like to introduce numpy arrays.

> "A NumPy array is a multidimensional, uniform collection of elements. An array is characterized by the type of elements it contains and by its shape. For example, a matrix may be represented as an array of shape ($M \times N$) that contains numbers, e.g., floating point or complex numbers. Unlike matrices, NumPy arrays can have any dimensionality. Furthermore, they may contain other kinds of elements (or even combinations of elements), such as booleans or dates. Underneath the hood, a NumPy array is really just a convenient way of describing one or more blocks of computer memory, so that the numbers represented may be easily manipulated" (van der Walt et al., 2011).

9.6 Example skpref usage

9.6.1 Bradley Terry example notebook

What you will find in this notebook examples of using skpref:

- for setting up the modelling task based framework
- to fit a classifier that's being read in from scikit-learn on the same problem which in the background uses reduction and aggregation methods.
- to fit a Bradley-Terry model with and without covariates on the pairwise comparison data of basketball matches.
- for applying the GridSearch technique for model selection

[1]:
```
# Optionally change the theme of the notebook to dark
# from jupyterthemes.stylefx import set_nb_theme
# set_nb_theme('chesterish')
```

[2]:
```
# Import skpref modules
import sys
sys.path.insert(0, "../..")
from skpref.random_utility import BradleyTerry
from skpref.task import PairwiseComparisonTask
from skpref.base import ClassificationReducer
from skpref.model_selection import GridSearchCV
from skpref.utils import nice_print_results

# Import scikit-learn packages to be used in tandem with skpref architecture
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

# Import other useful packages
import pandas as pd
import numpy as np
```

Reading in the data

The example dataset will be matches played by NBA teams, we will use the 2016 season's matches to predict the results of the 2017 matches. The dataset contains:

- a column for `team1` and `team2` indicating the two teams that have played each other
- `season_start`, which indicates which season the match belongs to
- `team1_wins` takes the value of 1 if the team in column `team1` win the match, 0 if they lost (there are no ties in basketball)
- `team_1_home` takes the value of 1 if `team1` was playing in their home court 0 if they were paying away (no neutral courts in the NBA)

[3]:
```
NBA_results = pd.read_csv('data/NBA_matches.csv')
NBA_results.head()
```

[3]:
```
           team1              team2  season_start  team1_wins  team_1_home
0  Atlanta Hawks    Toronto Raptors          2014           0            0
1  Atlanta Hawks     Indiana Pacers          2014           1            1
2  Atlanta Hawks  San Antonio Spurs          2014           0            0
3  Atlanta Hawks  Charlotte Hornets          2014           0            0
4  Atlanta Hawks    New York Knicks          2014           1            1
```

[4]:
```
NBA_results.tail()
```

[4]:
```
                   team1               team2  season_start  team1_wins  \
9835  Washington Wizards      Houston Rockets          2017           0
9836  Washington Wizards  Cleveland Cavaliers          2017           0
9837  Washington Wizards        Atlanta Hawks          2017           0
9838  Washington Wizards       Boston Celtics          2017           1
9839  Washington Wizards        Orlando Magic          2017           0

      team_1_home
9835            0
9836            0
9837            1
9838            1
9839            0
```

[5]:
```
season_split = 2016
train_data = NBA_results[NBA_results.season_start == season_split].copy()
test_data = NBA_results[NBA_results.season_start == season_split+1].copy()
```

We will also use team salary data as covariates in the model later, with the idea being that a team that has more money to pay to their athletes has an advantage over other teams, by having a better chance to attract the top talent in the league.

[6]:
```
NBA_team_salary_budget = pd.read_csv('data/team_salary_budgets.csv')
NBA_team_salary_budget.head()
```

[6]:
```
             team  season_start    salary
0   Atlanta Hawks          2014  58337671
1   Atlanta Hawks          2015  71378126
2   Atlanta Hawks          2016  95957250
3   Atlanta Hawks          2017  99375302
4  Boston Celtics          2014  59418142
```

Setting up the tasks

We set up the preference learning task by using the `PairwiseComparisonTask` object in `skpref`. This is the only extra step which might be a completely new concept to seasoned scikit-learn users. Once the task is specified, say in this case a pairwise comparison task, for any models applied in skpref, whether that is a reduction via scikit-learn or even a model that is not a pairwise compar-

ison model, the package will know that the problem itself is a pairwise comparison problem and can perform reduction and aggregation adequately in the background when needed.

In this example the `PairwiseComparisonTask` has the following components:

- `primary_table`: the table that contains the observed preferences
- `primary_table_alternatives_names`: the column or columns that contain the alternatives, in this case both columns team1 and team2 contain alternatives
- `primary_table_target_name`: the column that indicates the result of the pairwise comparison
- `target_column_correspondence`: in the case of pairwise comparisons, when the alternatives are split across two columns, the column indicating the result usually takes the form 1/0 to show whether one of the columns, in our case team1 or team2 has been preferred. So in this column the user indicates that when the team1_wins column takes the value 1 that means that the alternative in the column team1 has won.
- `features_to_use`: indicates which columns to use as covariates

[7]:
```
NBA_results_task_train_LR = PairwiseComparisonTask(
    primary_table=train_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=['team_1_home']
)

# For the test task, it's possible to make a copy of the training task and
# update the primary table
NBA_results_task_predict_LR = PairwiseComparisonTask(
    primary_table=test_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=['team_1_home']
)
```

Fitting a Logistic Regression

The only covariate we will use in this for now will be the `team_1_home` column, which should return a method that only learns what the home team advantage was on average, which is the equivalent to fitting a logistic regression where whether team1 is playing home or not is the only covariate.

$P(\texttt{team1\_wins} = 1) = logit(\alpha + \beta_1 \texttt{team\_1\_home})$

[8]:
```
my_log_red = ClassificationReducer(LogisticRegression(solver='lbfgs'))
my_log_red.fit_task(NBA_results_task_train_LR)
preds = my_log_red.predict_task(NBA_results_task_predict_LR)
```

[9]:
```
# predict_task returns a SubsetPosetVector which has the attributes
# top_input_data and boot_input_data corresponding to chosen and not chosen
# alternatives.
preds.top_input_data, preds.boot_input_data
```

[9]:
```
(array(['Dallas Mavericks', 'Charlotte Hornets', 'Brooklyn Nets', ...,
        'Washington Wizards', 'Washington Wizards', 'Orlando Magic'],
       dtype=object),
 array(['Atlanta Hawks', 'Atlanta Hawks', 'Atlanta Hawks', ...,
        'Atlanta Hawks', 'Boston Celtics', 'Washington Wizards'],
       dtype=object))
```

[10]:
```
NBA_results_task_predict_LR.primary_table.head()
```

[10]:

	team1	team2	season_start	team1_wins	team_1_home
7380	Atlanta Hawks	Dallas Mavericks	2017	1	0
7381	Atlanta Hawks	Charlotte Hornets	2017	0	0
7382	Atlanta Hawks	Brooklyn Nets	2017	0	0
7383	Atlanta Hawks	Miami Heat	2017	0	0
7384	Atlanta Hawks	Chicago Bulls	2017	0	0

[11]:
```
NBA_results_task_predict_LR.primary_table.tail()
```

[11]:

	team1	team2	season_start	team1_wins	team_1_home
9835	Washington Wizards	Houston Rockets	2017	0	0
9836	Washington Wizards	Cleveland Cavaliers	2017	0	0
9837	Washington Wizards	Atlanta Hawks	2017	0	1
9838	Washington Wizards	Boston Celtics	2017	1	1
9839	Washington Wizards	Orlando Magic	2017	0	0

[12]:
```
# All this learns so far is the home team advantage, since its the only
# covariate in the test_data table
nice_print_results(
    my_log_red.predict_proba_task(NBA_results_task_predict_LR,
                                  outcome=['Dallas Mavericks', 'Atlanta Hawks']))
```

```
Dallas Mavericks  [0.58 0.   0.   ... 0.   0.   0.  ]
Atlanta Hawks     [0.42 0.42 0.42 ... 0.42 0.   0.  ]
```

[13]:
```
nice_print_results(
    my_log_red.predict_proba_task(NBA_results_task_predict_LR,
                                  column=['team1', 'team2'])
)
```

```
team1 is preferred  [0.42 0.42 0.42 ... 0.58 0.58 0.42]
team2 is preferred  [0.58 0.58 0.58 ... 0.42 0.42 0.58]
```

Fitting a Bradley Terry model

As we can see in the example above the logistic regression approach does not learn different probabilities for a team winning or losing based on which other team they are playing. The Dallas Mavericks could be playing against the strongest or weakest team in the league and their estimated probability of winning would be the same. The difference between the Bradley-Terry model and logistic regression is that Bradley-Terry learns a function that can estimate whether each team will win or lose given the other team they are playing.

The task we will use for Bradley-Terry will be defined in a slightly different way, because in the first demo we won't use any covariates, therefore we define `features_to_use=None`

In the Bradley-Terry model each team gets a latent strength parameter λ_{team}, for example $\lambda_{\text{Atlanta Hawks}}$.

The Bradley-Terry model learns these strength parameters to maximise the likelihood according to the following formulation for observation i:

$$P(\texttt{team1\_wins} = 1)_i = \frac{e^{\lambda_{\texttt{team1}_i}}}{e^{\lambda_{\texttt{team1}_i}} + e^{\lambda_{\texttt{team2}_i}}}$$

[14]:
```
NBA_results_task_train_BT = PairwiseComparisonTask(
    primary_table=train_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=None
)

NBA_results_task_predict_BT = PairwiseComparisonTask(
    primary_table=test_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=None
)
```

[15]:
```
# Fitting Bradley Terry model
mybt = BradleyTerry(method='BFGS', alpha=1e-5)
mybt.fit_task(NBA_results_task_train_BT)
```

[16]:
```
mybt.params_
```

[16]:

	entity	learned_strength
0	Atlanta Hawks	0.047522
1	Boston Celtics	0.580896
2	Brooklyn Nets	-1.178393
3	Charlotte Hornets	-0.278154
4	Chicago Bulls	-0.037967
5	Cleveland Cavaliers	0.489737
6	Dallas Mavericks	-0.386261
7	Denver Nuggets	-0.040408
8	Detroit Pistons	-0.225709
9	Golden State Warriors	1.538386
10	Houston Rockets	0.765613
11	Indiana Pacers	-0.005751
12	Los Angeles Clippers	0.550265
13	Los Angeles Lakers	-0.773690
14	Memphis Grizzlies	0.153646
15	Miami Heat	-0.022175
16	Milwaukee Bucks	0.018291
17	Minnesota Timberwolves	-0.470415
18	New Orleans Pelicans	-0.328205
19	New York Knicks	-0.548175
20	Oklahoma City Thunder	0.344454
21	Orlando Magic	-0.655354
22	Philadelphia 76ers	-0.716305
23	Phoenix Suns	-0.888314
24	Portland Trail Blazers	0.019229
25	Sacramento Kings	-0.426973
26	San Antonio Spurs	1.115135
27	Toronto Raptors	0.462682
28	Utah Jazz	0.535025
29	Washington Wizards	0.361368

We can use the latent alternative strength parameters that Bradley-Terry models learn to rank the teams, either by sorting the `mybt.params_` `DataFrame` by the `learned_strength` parameter, or by running the `rank_entities` function

[17]:
```
mybt.rank_entities(ascending=False)
```

[17]:
```
['Golden State Warriors',
 'San Antonio Spurs',
 'Houston Rockets',
 'Boston Celtics',
 'Los Angeles Clippers',
 'Utah Jazz',
 'Cleveland Cavaliers',
```

```
 'Toronto Raptors',
 'Washington Wizards',
 'Oklahoma City Thunder',
 'Memphis Grizzlies',
 'Atlanta Hawks',
 'Portland Trail Blazers',
 'Milwaukee Bucks',
 'Indiana Pacers',
 'Miami Heat',
 'Chicago Bulls',
 'Denver Nuggets',
 'Detroit Pistons',
 'Charlotte Hornets',
 'New Orleans Pelicans',
 'Dallas Mavericks',
 'Sacramento Kings',
 'Minnesota Timberwolves',
 'New York Knicks',
 'Orlando Magic',
 'Philadelphia 76ers',
 'Los Angeles Lakers',
 'Phoenix Suns',
 'Brooklyn Nets']
```

[18]:
```
# we can create the probability for each team winning in a specific observaion,
nice_print_results(
    mybt.predict_proba_task(NBA_results_task_predict_BT,
                            outcome=['Atlanta Hawks', 'Washington Wizards'])
)
```

```
Atlanta Hawks       [0.61 0.58 0.77 ... 0.42 0.   0.  ]
Washington Wizards  [0.   0.   0.   ... 0.58 0.45 0.73]
```

[19]:
```
nice_print_results(
    mybt.predict_proba_task(NBA_results_task_predict_BT,
                            column=['team1', 'team2'])
)
```

```
team1 is preferred  [0.61 0.58 0.77 ... 0.58 0.45 0.73]
team2 is preferred  [0.39 0.42 0.23 ... 0.42 0.55 0.27]
```

[20]:
```
mybt.predict_choice_task(NBA_results_task_predict_BT)
```

[20]:
```
array(['Atlanta Hawks', 'Atlanta Hawks', 'Atlanta Hawks', ...,
       'Washington Wizards', 'Boston Celtics', 'Washington Wizards'],
      dtype=object)
```

[21]:
```
preds = mybt.predict_task(NBA_results_task_predict_BT)
```

[22]:
```
preds.top_input_data, preds.boot_input_data
```

[22]:
```
(array(['Atlanta Hawks', 'Atlanta Hawks', 'Atlanta Hawks', ...,
        'Washington Wizards', 'Boston Celtics', 'Washington Wizards'],
       dtype=object),
 array(['Dallas Mavericks', 'Charlotte Hornets', 'Brooklyn Nets', ...,
        'Atlanta Hawks', 'Washington Wizards', 'Orlando Magic'],
       dtype=object))
```

Augmenting the models with covariates

In this section we will start introducing more covariates in the models above, we will introduce one additional covariate which is the team salary budget. We can also see how we can define a single task which we can use to run different models in skpref.

[23]:
```
NBA_results_task_train = PairwiseComparisonTask(
    primary_table=train_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=['salary', 'team1_home'],
    secondary_table=NBA_team_salary_budget,
    secondary_to_primary_link={
        'team': ['team1', 'team2'],
        'season_start': 'season_start'
    })

NBA_results_task_predict = PairwiseComparisonTask(
    primary_table=test_data,
    primary_table_alternatives_names=['team1', 'team2'],
    primary_table_target_name ='team1_wins',
    target_column_correspondence='team1',
    features_to_use=['salary', 'team1_home'],
    secondary_table=NBA_team_salary_budget,
    secondary_to_primary_link={
        'team': ['team1', 'team2'],
        'season_start': 'season_start'
    })
```

Reduction to logistic regression with covariates

Here we fit a logistic regression on three covariates, whether `team1` is playing home or not, `team1`'s salary budget and `team2`'s salary budget. $P(\texttt{team1\_wins} = 1) = logit(\alpha + \beta_1 \texttt{team\_1\_home} + \beta_2 \texttt{team1\_salary} + \beta_3 \texttt{team2\_salary})$

[24]:
```
my_log_red = ClassificationReducer(LogisticRegression(solver='lbfgs'))
my_log_red.fit_task(NBA_results_task_train)
preds = my_log_red.predict_task(NBA_results_task_predict)
```

[25]:
```
# We can investigate the internal table that was fed into LogisticRegression.
 ↪fit()
my_log_red.model_input.head(7)
```

[25]:

	team1_wins	team_1_home	salary_team1	salary_team2
0	1	0	99375302	85753772
1	0	0	99375302	117228164
2	0	0	99375302	95964560
3	0	0	99375302	129458084
4	0	0	99375302	89524016
5	0	1	99375302	107015203
6	0	1	99375302	115375243

[26]:
```
# We can also investigate the coefficients which were learned
my_log_red.model.coef_
```

[26]:
```
array([[ 5.35210228e-15,  1.54775613e-08, -1.54775613e-08]])
```

We can see that the coefficients learned for β_2 and β_3 are very similar to each other, just opposite signs. ClassificationReducer allows users the option to take the difference in features directly rather than split them out, effectively learning the following model: $P(\texttt{team1\_wins} = 1) = logit(\alpha + \beta_1 \texttt{team\_1\_home} + \beta_2(\texttt{team1\_salary} - \texttt{team2\_salary}))$

[27]:
```
my_log_red = ClassificationReducer(
    LogisticRegression(solver='lbfgs'),
    take_feature_diff_for_pairwise_comparison=True
)
my_log_red.fit_task(NBA_results_task_train)
preds = my_log_red.predict_task(NBA_results_task_predict)
```

[28]:
```
my_log_red.model_input.head(7)
```

[28]:

	team1_wins	team_1_home	salary_diff
0	1	0	13621530
1	0	0	-17852862
2	0	0	3410742
3	0	0	-30082782
4	0	0	9851286
5	0	1	-7639901
6	0	1	-15999941

[29]:
```
my_log_red.model.coef_
```

[29]:
```
array([[2.67602286e-15, 1.54775613e-08]])
```

[30]:
```
preds.top_input_data, preds.boot_input_data
```

[30]:
```
(array(['Atlanta Hawks', 'Charlotte Hornets', 'Atlanta Hawks', ...,
        'Washington Wizards', 'Washington Wizards', 'Washington Wizards'],
       dtype=object),
 array(['Dallas Mavericks', 'Atlanta Hawks', 'Brooklyn Nets', ...,
        'Atlanta Hawks', 'Boston Celtics', 'Orlando Magic'], dtype=object))
```

[31]:
```
# All this learns so far is the home team advantage, since its the only
# covariate in the test_data table
nice_print_results(
    my_log_red.predict_proba_task(NBA_results_task_predict,
                                  column='team1')
)
```

```
team1 is preferred  [0.55 0.43 0.51 ... 0.59 0.53 0.61]
```

Bradley Terry model with salary covariate

Here we augment the initial Bradley-Terry model to learn the following relationship:

$$P(\texttt{team1\_wins} = 1)_i = \frac{e^{(\lambda_{\texttt{team1}_i} + \beta_1 \texttt{team1\_salary}_i)}}{e^{(\lambda_{\texttt{team1}_i} + \beta_1 \texttt{team1\_salary}_i)} + e^{(\lambda_{\texttt{team2}_i} + \beta_1 \texttt{team2\_salary}_i)}}$$

[32]:
```
mybt = BradleyTerry(method='BFGS', alpha=1e-5)
mybt.fit_task(NBA_results_task_train)
mybt.rank_entities(ascending=False)
```

[32]:
```
array(['Golden State Warriors', 'San Antonio Spurs', 'Houston Rockets',
       'Utah Jazz', 'Boston Celtics', 'Oklahoma City Thunder',
       'Washington Wizards', 'Toronto Raptors', 'Los Angeles Clippers',
       'Denver Nuggets', 'Atlanta Hawks', 'Indiana Pacers',
       'Chicago Bulls', 'Cleveland Cavaliers', 'Memphis Grizzlies',
       'Miami Heat', 'Milwaukee Bucks', 'Charlotte Hornets',
       'Minnesota Timberwolves', 'Portland Trail Blazers',
       'New Orleans Pelicans', 'Sacramento Kings', 'Detroit Pistons',
       'Dallas Mavericks', 'Philadelphia 76ers', 'New York Knicks',
       'Phoenix Suns', 'Los Angeles Lakers', 'Orlando Magic',
       'Brooklyn Nets'], dtype=object)
```

[33]:
```
nice_print_results(mybt.predict_proba_task(NBA_results_task_predict,␣
↪column=['team1', 'team2']))
```

```
team1 is preferred  [0.69 0.48 0.75 ... 0.65 0.43 0.82]
team2 is preferred  [0.31 0.52 0.25 ... 0.35 0.57 0.18]
```

[34]:
```
mybt.predict_choice_task(NBA_results_task_predict)
```

[34]:
```
array(['Atlanta Hawks', 'Charlotte Hornets', 'Atlanta Hawks', ...,
       'Washington Wizards', 'Boston Celtics', 'Washington Wizards'],
      dtype=object)
```

[35]:
```
mybt.predict_task(NBA_results_task_predict).top_input_data
```

[35]:
```
array(['Atlanta Hawks', 'Charlotte Hornets', 'Atlanta Hawks', ...,
       'Washington Wizards', 'Boston Celtics', 'Washington Wizards'],
      dtype=object)
```

[36]:
```
mybt.bt_with_feats.get_statsmodels_summary()
```

[36]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                      Multinomial Logit Model Regression Results
==============================================================================
===
Dep. Variable:                      CHOICE   No. Observations:
2,460
Model:             Multinomial Logit Model   Df Residuals:
2,429
Method:                               MLE   Df Model:
31
Date:                     Wed, 01 Mar 2023   Pseudo R-squ.:
0.107
Time:                             16:10:40   Pseudo R-bar-squ.:
0.089
AIC:                             3,107.966   Log-Likelihood:
-1,522.983
BIC:                             3,288.012   LL-Null:
-1,705.142
==============================================================================
==========
                             coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
----------
salary                  1.717e-08   3.65e-06      0.005      0.996   -7.13e-06
7.17e-06
Atlanta Hawks              0.0810     41.439      0.002      0.998     -81.138
81.300
Boston Celtics             0.7344     52.254      0.014      0.989    -101.682
103.151
Brooklyn Nets             -0.9380     65.383     -0.014      0.989    -129.086
127.210
```

Charlotte Hornets	-0.1415	50.101	-0.003	0.998	-98.339
98.056					
Chicago Bulls	0.0621	46.030	0.001	0.999	-90.156
90.280					
Cleveland Cavaliers	-0.0049	112.740	-4.33e-05	1.000	-220.970
220.960					
Dallas Mavericks	-0.4891	46.300	-0.011	0.992	-91.236
90.258					
Denver Nuggets	0.2237	69.393	0.003	0.997	-135.784
136.232					
Detroit Pistons	-0.4001	55.137	-0.007	0.994	-108.468
107.667					
Golden State Warriors	1.4940	41.902	0.036	0.972	-80.632
83.620					
Houston Rockets	0.9021	50.079	0.018	0.986	-97.252
99.056					
Indiana Pacers	0.0727	44.101	0.002	0.999	-86.363
86.508					
Los Angeles Clippers	0.2355	78.355	0.003	0.998	-153.337
153.808					
Los Angeles Lakers	-0.7116	42.901	-0.017	0.987	-84.796
83.373					
Memphis Grizzlies	-0.0434	58.483	-0.001	0.999	-114.668
114.581					
Miami Heat	-0.0820	42.759	-0.002	0.998	-83.888
83.724					
Milwaukee Bucks	-0.1289	51.425	-0.003	0.998	-100.920
100.663					
Minnesota Timberwolves	-0.1561	78.274	-0.002	0.998	-153.569
153.257					
New Orleans Pelicans	-0.3903	42.901	-0.009	0.993	-84.476
83.695					
New York Knicks	-0.6348	44.785	-0.014	0.989	-88.412
87.143					
Oklahoma City Thunder	0.4593	47.565	0.010	0.992	-92.766
93.685					
Orlando Magic	-0.7402	44.632	-0.017	0.987	-88.217
86.736					
Philadelphia 76ers	-0.5043	60.789	-0.008	0.993	-119.649
118.640					
Phoenix Suns	-0.6594	63.503	-0.010	0.992	-125.123
123.804					
Portland Trail Blazers	-0.2206	65.293	-0.003	0.997	-128.192
127.751					
Sacramento Kings	-0.3933	41.448	-0.009	0.992	-81.630
80.843					
San Antonio Spurs	0.9525	53.485	0.018	0.986	-103.876

```
105.781
Toronto Raptors            0.2806     56.240      0.005      0.996    -109.949
110.510
Utah Jazz                  0.8459     77.655      0.011      0.991    -151.355
153.047
Washington Wizards         0.2946     43.216      0.007      0.995     -84.407
84.997
================================================================================
==========
"""
```

Example using GridSearchCV()

The models we have fitted above also have hyperparameters, such as the method of gradient descent or regularisation. To optimise the hyperparameter selection, we can use `GridSearchCV()`. `GridSearchCV()` tries out a series of hyperparameter combinations and runs a k-fold cross-validation on an accuracy metric determined by the user to check which ones have performed best.

[37]:
```
to_tune = {'alpha': [1, 2, 4], 'method': ['BFGS']}
gs_bt = GridSearchCV(BradleyTerry(), to_tune,  cv=3, scoring='neg_log_loss')
gs_bt.fit_task(NBA_results_task_train)
gs_bt.inspect_results()
```

```
The model with the best parameters was:
BradleyTerry(alpha=2, method='BFGS')
With a score of -0.6265008194657992
All the trials results summarised in descending score
   alpha method  mean_test_score
1      2   BFGS        -0.626501
0      1   BFGS        -0.626742
2      4   BFGS        -0.628853
```

[38]:
```
# Showing that sklearn.metrics works also
to_tune = {'alpha': [1, 2, 4], 'method': ['BFGS']}
gs_bt = GridSearchCV(BradleyTerry(), to_tune,  cv=3, scoring=f1_score)
gs_bt.fit_task(NBA_results_task_train)
gs_bt.inspect_results()
```

```
The model with the best parameters was:
BradleyTerry(alpha=4, method='BFGS')
With a score of 0.6337744652191032
All the trials results summarised in descending score
   alpha method  mean_test_score
2      4   BFGS         0.633774
1      2   BFGS         0.631136
0      1   BFGS         0.630085
```

[39]:
```
to_tune = {'C': [0.5, 1, 2, 4, 8], 'solver': ['saga'], 'penalty': ['l1','l2'],
           'fit_intercept': [True, False]}
gs_lr = GridSearchCV(ClassificationReducer(LogisticRegression()), to_tune,
                     cv=3, scoring='neg_log_loss')
gs_lr.fit_task(NBA_results_task_train)
gs_lr.inspect_results()
```

```
The model with the best parameters was:
ClassificationReducer(model=LogisticRegression(C=0.5, penalty='l1',
                                               solver='saga'))
With a score of -0.6865126660183437
All the trials results summarised in descending score
    model__C  model__fit_intercept model__penalty model__solver  \
0        0.5                 True             l1          saga
13       4.0                 True             l2          saga
6        1.0                False             l1          saga
2        0.5                False             l1          saga
5        1.0                 True             l2          saga
16       8.0                 True             l1          saga
10       2.0                False             l1          saga
19       8.0                False             l2          saga
9        2.0                 True             l2          saga
14       4.0                False             l1          saga
18       8.0                False             l1          saga
4        1.0                 True             l1          saga
11       2.0                False             l2          saga
1        0.5                 True             l2          saga
17       8.0                 True             l2          saga
15       4.0                False             l2          saga
7        1.0                False             l2          saga
8        2.0                 True             l1          saga
3        0.5                False             l2          saga
12       4.0                 True             l1          saga

    mean_test_score
0         -0.686513
13        -0.686516
6         -0.686516
2         -0.686517
5         -0.686517
16        -0.686517
10        -0.686518
19        -0.686518
9         -0.686518
14        -0.686518
18        -0.686518
4         -0.686518
```

```
11        -0.686519
1         -0.686519
17        -0.686519
15        -0.686520
7         -0.686520
8         -0.686520
3         -0.686521
12        -0.686522
```

[40]:
```
gs_lr.predict_task(NBA_results_task_predict).top_input_data
```

[40]:
```
array(['Atlanta Hawks', 'Charlotte Hornets', 'Atlanta Hawks', ...,
       'Washington Wizards', 'Washington Wizards', 'Washington Wizards'],
      dtype=object)
```

[41]:
```
nice_print_results(gs_lr.predict_proba_task(NBA_results_task_predict,␣
↪column='team1'))
```

```
team1 is preferred  [0.55 0.43 0.51 ... 0.59 0.53 0.61]
```

[42]:
```
nice_print_results(gs_bt.predict_proba_task(NBA_results_task_predict,␣
↪column='team1'))
```

```
team1 is preferred  [0.67 0.47 0.7  ... 0.64 0.45 0.79]
```

[43]:
```
gs_bt.rank_entities(ascending=False)
```

[43]:
```
array(['Golden State Warriors', 'San Antonio Spurs', 'Houston Rockets',
       'Utah Jazz', 'Boston Celtics', 'Oklahoma City Thunder',
       'Washington Wizards', 'Toronto Raptors', 'Los Angeles Clippers',
       'Denver Nuggets', 'Atlanta Hawks', 'Indiana Pacers',
       'Chicago Bulls', 'Cleveland Cavaliers', 'Memphis Grizzlies',
       'Miami Heat', 'Milwaukee Bucks', 'Charlotte Hornets',
       'Minnesota Timberwolves', 'Portland Trail Blazers',
       'Detroit Pistons', 'New Orleans Pelicans', 'Sacramento Kings',
       'Philadelphia 76ers', 'Dallas Mavericks', 'New York Knicks',
       'Phoenix Suns', 'Los Angeles Lakers', 'Orlando Magic',
       'Brooklyn Nets'], dtype=object)
```

We will demonstrate skpref on discrete choice data

We wil use the swissmetro dataset available to download on https://transp-or.epfl.ch/pythonbiogeme/examples_swissmetro.html This dataset tracks 470 respondents on which transportation alternative they have taken. There are 3 options in general: train, car and swissmetro. More details on the original use of the dataset can be found here: http://strc.ch/2001/bierlaire1.pdf

Since at the moment of writing skpref still didn't have a discrete choice model interfaced, we will reduce the discrete choices to pairwise comparisons.

In this notebook we will:

- Start by tranforming the original swissmetro dataset into something that skpref can handle.
- Fit a logistic regression using the data and the `ClassificationReducer()` method.
- Fit a Bradley-Terry model using reduction to pairwise comparisons.
- Show how two different aggregation methods for going from a pairwise comparison model to a discrete choice model work.
- Show an example using `GridSearchCV()` and how to specify aggregation methods in `GridSearchCV()`
- Show some of the evaluation methods that can be applied using skpref

[1]:
```
import pandas as pd
pd.options.display.max_columns = 999
import numpy as np
import sys
sys.path.insert(0, "../..")
from skpref.base import ClassificationReducer
from skpref.random_utility import BradleyTerry
from skpref.task import ChoiceTask
from skpref.metrics import f1_score, log_loss, log_loss_compare_with_t_test
from skpref.utils import nice_print_results
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from copy import deepcopy
from sklearn.linear_model import LogisticRegression
from skpref.model_selection import GridSearchCV
```

[2]:
```
swissmetro = pd.read_csv("data/swissmetro.dat", sep='\t')
swissmetro.head()
```

[2]:

	GROUP	SURVEY	SP	ID	PURPOSE	FIRST	TICKET	WHO	LUGGAGE	AGE	MALE	\
0	2	0	1	1	1	0	1	1	0	3	0	
1	2	0	1	1	1	0	1	1	0	3	0	
2	2	0	1	1	1	0	1	1	0	3	0	
3	2	0	1	1	1	0	1	1	0	3	0	
4	2	0	1	1	1	0	1	1	0	3	0	

```
   INCOME  GA  ORIGIN  DEST  TRAIN_AV  CAR_AV  SM_AV  TRAIN_TT  TRAIN_CO  \
0       2   0       2     1         1       1      1       112        48
1       2   0       2     1         1       1      1       103        48
2       2   0       2     1         1       1      1       130        48
3       2   0       2     1         1       1      1       103        40
4       2   0       2     1         1       1      1       130        36

   TRAIN_HE  SM_TT  SM_CO  SM_HE  SM_SEATS  CAR_TT  CAR_CO  CHOICE
0       120     63     52     20         0     117      65       2
1        30     60     49     10         0     117      84       2
2        60     67     58     30         0     117      52       2
3        30     63     52     20         0      72      52       2
4        60     63     42     20         0      90      84       2
```

Looking at the data we can see that each row represents a choice. The full explanations of variables can be found here: https://transp-or.epfl.ch/pythonbiogeme/examples/swissmetro/swissmetro.pdf

Changing the format of the tables for skpref

1) In this table the availability of alternatives is marked by `TRAIN_AV`, `CAR_AV`, `SM_AV` which indicate with 1 if the alternative is available and 0 otherwise. We need to convert these to a column that contains a list of alternatives for each row.

2) The choices are indicated by the `CHOICE` column, which contains 0 for unknown (we will dropping these), 1 for Train, 2 for Swissmetro and 3 for a Car usage. We need to name these explicitly. We could just create a list of alternatives called 1,2,3 in step 1 which would bypass step 2, however, we prefer the clarity of having the alternatives named explicitly.

Using only a subset of the available features

Under normal circumstances we would use one-hot encoding on a lot of the binary features before training a serious model, however, to make the demo simple, we will only use the travel time and cost features. Users can of course do whatever feature transformations they like before fitting a model. To build a classifier based on travel time and costs, we first need to split some columns from the swissmetro dataset into a secondary table.

[3]:
```
train_vals = swissmetro[['TRAIN_TT', 'TRAIN_CO']].copy()
train_vals.columns = ['Travel Time', 'Cost']
train_vals.reset_index(inplace=True)
train_vals['alternative'] = 'Train'
swissmetro_vals = swissmetro[['SM_TT', 'SM_CO']].copy()
swissmetro_vals.columns = ['Travel Time', 'Cost']
swissmetro_vals.reset_index(inplace=True)
swissmetro_vals['alternative'] = 'Swiss Metro'
car_vals = swissmetro[['CAR_TT', 'CAR_CO']].copy()
car_vals.columns = ['Travel Time', 'Cost']
car_vals.reset_index(inplace=True)
```

```
car_vals['alternative'] = 'Car'
dummy_secondary_table = (
    train_vals.append(swissmetro_vals.append(car_vals))
).sort_values('index')
dummy_secondary_table.rename(columns={'index': 'merge_index'}, inplace=True)
dummy_secondary_table.head()
```

[3]:

	merge_index	Travel Time	Cost	alternative
0	0	112	48	Train
0	0	63	52	Swiss Metro
0	0	117	65	Car
1	1	103	48	Train
1	1	60	49	Swiss Metro

[4]:
```
binary_concats = (swissmetro.TRAIN_AV.astype(str) +
                  swissmetro.CAR_AV.astype(str)  +
                  swissmetro.SM_AV.astype(str)
                 )

alts = []
for i in binary_concats.values:
    if i == '111':
        alts.append(['Train', 'Car', 'Swiss Metro'])
    elif i == '100':
        alts.append(['Train'])
    elif i == '000':
        alts.append(['None'])
    elif i == '010':
        alts.append(['Car'])
    elif i == '001':
        alts.append(['Swiss Metro'])
    elif i == '101':
        alts.append(['Train', 'Swiss Metro'])
    elif i == '110':
        alts.append(['Train', 'Car'])
    elif i == '011':
        alts.append(['Car', 'Swiss Metro'])

swissmetro['alternatives'] = alts

swissmetro['chosen'] = np.where(swissmetro.CHOICE.values==1, 'Train',
                       np.where(swissmetro.CHOICE.values==2, 'Swiss Metro',
                       np.where(swissmetro.CHOICE.values==3, 'Car',
                                'unknown')))
swissmetro = swissmetro[swissmetro.CHOICE != 0].copy()
swissmetro = swissmetro.reset_index()[['alternatives', 'chosen', 'index']]
swissmetro.rename(columns={'index': 'merge_index'}, inplace=True)
```

```
swissmetro.head()
```

[4]:

	alternatives	chosen	merge_index
0	[Train, Car, Swiss Metro]	Swiss Metro	0
1	[Train, Car, Swiss Metro]	Swiss Metro	1
2	[Train, Car, Swiss Metro]	Swiss Metro	2
3	[Train, Car, Swiss Metro]	Swiss Metro	3
4	[Train, Car, Swiss Metro]	Swiss Metro	4

[5]:
```
swissmetro.alternatives.values
```

[5]:
```
array([list(['Train', 'Car', 'Swiss Metro']),
       list(['Train', 'Car', 'Swiss Metro']),
       list(['Train', 'Car', 'Swiss Metro']), ...,
       list(['Train', 'Car', 'Swiss Metro']),
       list(['Train', 'Car', 'Swiss Metro']),
       list(['Train', 'Car', 'Swiss Metro'])], dtype=object)
```

Fit a logistic regression

This will fit a logistic regression that uses only travel time and cost as covariates, with the following formulation for observation i and $a \in \{\text{Car, Train, Swiss Metro}\}$:

$$P(Y_i = a) = logit(\lambda_a + \beta_1(\text{Travel Time})_a + \beta_2(\text{Cost})_a)$$

Below we showcase how the ChoiceTask wrapper can deal with creating this reduction

[6]:
```
train, test = train_test_split(swissmetro, random_state=1, test_size=0.1)

swiss_metro_train = ChoiceTask(train, 'alternatives', 'chosen',
                               features_to_use=['Travel Time', 'Cost'],
                               secondary_table=dummy_secondary_table,
                               secondary_to_primary_link={
                                   'merge_index': 'merge_index',
                                   'alternative': 'alternatives'
                               }
                              )

swiss_metro_test = ChoiceTask(test, 'alternatives', 'chosen',
                              features_to_use=['Travel Time', 'Cost'],
                              secondary_table=dummy_secondary_table,
                              secondary_to_primary_link={
                                   'merge_index': 'merge_index',
                                   'alternative': 'alternatives'
                               }
                             )
```

[7]:
```
my_log_red = ClassificationReducer(LogisticRegression(solver='lbfgs'))
my_log_red.fit_task(swiss_metro_train)
log_reg_preds = my_log_red.predict_proba_task(swiss_metro_test,
                                        ['Swiss Metro','Train', 'Car'])
```

[8]:
```
nice_print_results(log_reg_preds)
```

```
Swiss Metro  [0.49 0.56 0.4  ... 0.47 0.35 0.51]
Train        [0.3  0.45 0.17 ... 0.36 0.29 0.26]
Car          [0.33 0.   0.2  ... 0.45 0.36 0.25]
```

[9]:
```
outocme_preds = my_log_red.predict_task(swiss_metro_test)
print(outocme_preds.top_input_data[:5])
print(outocme_preds.boot_input_data[:5])
```

```
['Swiss Metro' 'Swiss Metro' 'Swiss Metro' 'Swiss Metro' 'Swiss Metro']
[array(['Car', 'Train'], dtype='<U11') array(['Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')]
```

Fit a Bradley-Terry model without covariates

Here we reduce the discrete choice problem to pairwise comparisons and fit a Bradley-Terry model.

The way these observations are broken down are such that when alternative a is chosen from the set A then an observation is expressed in a way that we say the chosen alternative was preferred to all not-chosen alternatives, $a \succ j \forall j \in A \setminus a$. This transforms a table that looks like this:

Decision	Alternatives	Choice
1	{a, b, c}	a
2	{b, c}	c

into a table that looks like this:

Decision	Alternatives	Choice
1	{a, b}	a
1	{a, c}	a
2	{b, c}	c

A pariwise comparison model such as Bradley-Terry can be trained on the second table. Perhaps to make it similar to the pairwise comparison example we can also express the above table in the more familiar format:

Decision	Alternative 1	Alternative 2	Alternative 1 is chosen
1	a	b	1
1	a	c	1
2	b	c	0

and fit the Bradley-Terry model to learn the latent strength parameters for each alternative (e.g. λ_a for alternative a):

$$P(\text{Alternative 1 is chosen})_i = \frac{e^{\lambda_{\text{Alternative 1}_i}}}{e^{\lambda_{\text{Alternative 1}_i}} + e^{\lambda_{\text{Alternative 2}_i}}}$$

We can see that the Bradley-Terry probabilities begin to take into account the other alternatives that are offered to the decision makers, in contrast with logistic regression, which assumes that the probability of taking Swissmetro is the same whether a decision maker has a car as an alternative, a train or both. Once the Bradley-Terry model is trained, predictions have to be aggregated to discrete choice. In this section we showcase two aggregation methods one we call the Luce method the other the independent transitive method.

The Luce method (the default setting when aggregating a Bradley-Terry model)

For alternatives $\{a, b, c\}$ when we fit the Bradley-Terry model we learn the function $f(a), f(b), f(c)$ which include their strength parameters and potentially some covariates, in the econometrics literature this would be known as finding out the utility of each alternative. In the simplest case the utility equations only contain the strength parameters of the alternatives (e.g. λ_a for alternative a). The Luce aggregation method would predict the probability of choosing a from $\{a, b, c\}$ as:

$$\frac{e^{f(a)}}{e^{f(a)} + e^{f(b)} + e^{f(c)}}$$

.

The independent transitive aggregation method

Let's denote the probability of chosing a from $\{a, b, c\}$ as $P(a \succ \{a, b, c\})$. Suppose that we have a probabilistic pairwise comparison predictor (such as Bradley-Terry) that can provide us with the probability of preferring one over any two alternatives $P(i \succ \{i, j\}) \forall i, j \in \{a, b, c\}$.

The independent transitive method stems from the following logic:

1. For a to be chosen from $\{a, b, c\}$, a would have to be preferred to b and c, that is $a \succ \{a, b\} \cap a \succ \{a, c\}$
2. Assuming that a being preferred to c is independent from a being preferred to b, the probability of a being preferred to b and c is: $P(a \succ \{a, b\} \cap a \succ \{a, c\}) = P(a \succ \{a, b\})P(a \succ \{a, c\})$
3. In this three-alternative aggregation example, only 2 other things can happen in addition to a being chosen, b can be chosen or c can be chosen. Each of which can be expressed as we have expressed the probability of a being chosen in bullet 2.
4. By dividing the probability of a being chosen by all the three different possible outcomes (a is chosen, b is chosen or c is chosen), we arrive to the final equation of the probability that a is chosen from $\{a, b, c\}$:

$$\frac{P(a \succ \{a,b\})P(a \succ \{a,c\})}{P(a \succ \{a,b\})P(a \succ \{a,c\}) + P(b \succ \{a,b\})P(b \succ \{b,c\}) + P(c \succ \{a,c\})P(c \succ \{b,c\})}$$

[10]:
```
# Fit a Bradley Terry model with no features
swiss_metro_train_BT = ChoiceTask(train.drop('merge_index', axis=1),
                                  'alternatives',
                                  'chosen', features_to_use=None)

swiss_metro_test_BT = ChoiceTask(test.drop('merge_index', axis=1),
                                 'alternatives',
                                 'chosen', features_to_use=None)
```

[11]:
```
my_BT_red = BradleyTerry(method='BFGS', alpha=1e-5)
my_BT_red.fit_task(swiss_metro_train_BT)
preds = my_BT_red.predict_proba_task(swiss_metro_test_BT,
                                     ['Swiss Metro','Train', 'Car'])
```

[12]:
```
nice_print_results(preds)
```

```
Swiss Metro  [0.55 0.83 0.55 ... 0.55 0.55 0.55]
Train        [0.11 0.17 0.11 ... 0.11 0.11 0.11]
Car          [0.35 0.   0.35 ... 0.35 0.35 0.35]
```

[13]:
```
preds
```

[13]:
```
{'Swiss Metro': array([0.54530381, 0.8344241 , 0.54530381, ..., 0.54530381,
0.54530381,
        0.54530381]),
 'Train': array([0.10820537, 0.1655759 , 0.10820537, ..., 0.10820537,
0.10820537,
        0.10820537]),
 'Car': array([0.34649083, 0.        , 0.34649083, ..., 0.34649083, 0.34649083,
        0.34649083])}
```

[14]:
```
choice_preds = my_BT_red.predict_task(swiss_metro_test_BT)
print(choice_preds.top_input_data[:5])
print(choice_preds.boot_input_data[:5])
```

```
['Swiss Metro' 'Swiss Metro' 'Swiss Metro' 'Swiss Metro' 'Swiss Metro']
[array(['Car', 'Train'], dtype='<U11') array(['Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')
 array(['Car', 'Train'], dtype='<U11')]
```

Fitting a Bradley-Terry model with covariates

We now fit a Bradley-Terry model using the Travel Time and Cost covariates so that the equation above becomes:

$$P(\text{Alternative 1 is chosen})_i = \frac{e^{\lambda_{\text{Alternative 1}_i} + \beta_1(\text{Travel Time})_i + \beta_2 \text{Cost}_i}}{e^{\lambda_{\text{Alternative 1}_i} + \beta_1(\text{Travel Time})_i + \beta_2 \text{Cost}_i} + e^{\lambda_{\text{Alternative 2}_i} + \beta_1(\text{Travel Time})_i + \beta_2 \text{Cost}_i}}$$

[15]:
```
# Fit a Bradley Terry model with features
# Reducing training set because on my PC this gives a memory error
swiss_metro_train_BT_feats = ChoiceTask(
    train.sample(frac=0.5), 'alternatives', 'chosen',
    secondary_table=dummy_secondary_table,
    secondary_to_primary_link={
        'merge_index': 'merge_index',
        'alternative': 'alternatives'
    },
    features_to_use=['Travel Time', 'Cost'])

swiss_metro_test = ChoiceTask(test, 'alternatives', 'chosen',
                              features_to_use=['Travel Time', 'Cost'],
                              secondary_table=dummy_secondary_table,
                              secondary_to_primary_link={
                                   'merge_index': 'merge_index',
                                   'alternative': 'alternatives'
                               }
                             )

my_BT_red_feats = BradleyTerry(method='BFGS', alpha=100, max_iter=100000)
my_BT_red_feats.fit_task(swiss_metro_train_BT_feats)
```

[16]:
```
swiss_metro_test = ChoiceTask(test, 'alternatives', 'chosen',
                              features_to_use=['Travel Time', 'Cost'],
                              secondary_table=dummy_secondary_table,
                              secondary_to_primary_link={
                                   'merge_index': 'merge_index',
                                   'alternative': 'alternatives'
                               }
                             )
preds = my_BT_red_feats.predict_proba_task(swiss_metro_test,
                                    ['Swiss Metro','Train', 'Car'])
```

[17]:
```
my_BT_red_feats.predict_task(swiss_metro_test).top_input_data
```

[17]:
```
array(['Swiss Metro', 'Swiss Metro', 'Swiss Metro', ..., 'Car', 'Car',
       'Swiss Metro'], dtype=object)
```

[18]:
```
nice_print_results(preds)
```

```
Swiss Metro  [0.56 0.75 0.67 ... 0.41 0.38 0.69]
Train        [0.13 0.25 0.09 ... 0.14 0.15 0.11]
Car          [0.31 0.   0.24 ... 0.45 0.47 0.21]
```

[19]:
```
ind_trans_preds = my_BT_red_feats.predict_proba_task(swiss_metro_test,
                                    ['Swiss Metro','Train', 'Car'],
                                    aggregation_method='independent transitive')
nice_print_results(ind_trans_preds)
```

```
Swiss Metro  [0.63 0.75 0.74 ... 0.44 0.4  0.77]
Train        [0.06 0.25 0.04 ... 0.07 0.09 0.05]
Car          [0.31 0.   0.22 ... 0.49 0.51 0.18]
```

[20]:
```
preds_outcome_ind_trans = my_BT_red_feats.predict_task(swiss_metro_test,
                              aggregation_method='independent transitive')
preds_outcome_Luce = my_BT_red_feats.predict_task(swiss_metro_test)
```

[21]:
```
dummy_secondary_table.groupby('alternative').mean()
```

[21]:

alternative	merge_index	Travel Time	Cost
Car	5363.5	123.795209	78.742077
Swiss Metro	5363.5	87.466350	670.340697
Train	5363.5	166.626025	514.335477

[22]:
```
my_BT_red_feats.bt_with_feats.get_statsmodels_summary()
```

[22]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                      Multinomial Logit Model Regression Results
==============================================================================
===
Dep. Variable:                      CHOICE   No. Observations:
8,878
Model:             Multinomial Logit Model   Df Residuals:
8,873
Method:                                MLE   Df Model:
5
Date:                      Wed, 01 Mar 2023   Pseudo R-squ.:
0.226
Time:                              16:11:21   Pseudo R-bar-squ.:
0.225
AIC:                              9,534.703   Log-Likelihood:
-4,762.351
BIC:                              9,570.159   LL-Null:
-6,153.761
==============================================================================
```

```
                  coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Cost            0.0003   3.11e-05      8.763      0.000       0.000       0.000
Travel Time    -0.0116      0.001    -22.819      0.000      -0.013      -0.011
Car             0.3031      0.046      6.635      0.000       0.214       0.393
Swiss Metro     0.1378      0.048      2.870      0.004       0.044       0.232
Train          -0.4409      0.048     -9.159      0.000      -0.535      -0.347
==============================================================================
"""
```

Fit Bradley-Terry model with GridSearch

The models we have fitted above also have hyperparameters, such as the method of gradient descent or regularisation. To optimise the hyperparameter selection, we can use `GridSearchCV()`. `GridSearchCV()` tries out a series of hyperparameter combinations and runs a k-fold cross-validation on an accuracy metric determined by the user to check which ones have performed best.

In this section we will show how aggregation works with GridSearch, it is possible to just add `aggregation_method` in the `predict_proba_task` and the ouptuts work as expected. Note in this example we have chosen a very different alpha to the models above so that there is some slight difference in the ouptuts to two decimal places, so that we can see that different parameters were learned.

[23]:
```
to_tune = {'alpha': [100,1000], 'method': ['BFGS']}
gs_bt = GridSearchCV(BradleyTerry(), to_tune,  cv=3, scoring='neg_log_loss')
gs_bt.fit_task(swiss_metro_train_BT_feats)
gs_bt.inspect_results()
```

```
The model with the best parameters was:
BradleyTerry(alpha=100, method='BFGS')
With a score of -0.5345453019698206
All the trials results summarised in descending score
   alpha method  mean_test_score
0    100   BFGS        -0.534545
1   1000   BFGS        -0.546477
```

[24]:
```
nice_print_results(gs_bt.predict_proba_task(
    swiss_metro_test,['Swiss Metro','Train', 'Car'],
    aggregation_method='independent transitive'))
```

```
Swiss Metro  [0.63 0.75 0.74 ... 0.44 0.4  0.77]
Train        [0.06 0.25 0.04 ... 0.07 0.09 0.05]
Car          [0.31 0.   0.22 ... 0.49 0.51 0.18]
```

[25]:
```
gs_bt.best_estimator_.bt_with_feats.get_statsmodels_summary()
```

[25]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                      Multinomial Logit Model Regression Results
==============================================================================
===
Dep. Variable:                      CHOICE   No. Observations:
8,878
Model:             Multinomial Logit Model   Df Residuals:
8,873
Method:                                MLE   Df Model:
5
Date:                      Wed, 01 Mar 2023   Pseudo R-squ.:
0.226
Time:                              16:11:38   Pseudo R-bar-squ.:
0.225
AIC:                               9,534.703   Log-Likelihood:
-4,762.351
BIC:                               9,570.159   LL-Null:
-6,153.761
==============================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Cost            0.0003   3.11e-05      8.763      0.000       0.000       0.000
Travel Time    -0.0116      0.001    -22.819      0.000      -0.013      -0.011
Car             0.3031      0.046      6.635      0.000       0.214       0.393
Swiss Metro     0.1378      0.048      2.870      0.004       0.044       0.232
Train          -0.4409      0.048     -9.159      0.000      -0.535      -0.347
==============================================================================
"""
```

Evaluation methods

In this section we show some of the evaluation methods available in skpref, specifically how to use `log_loss` and `log_loss_compare_with_t_test`. Please see the documentation for more details on how these work.

[26]:
```
print(f"The F1 score of the Luce aggregation was \
{f1_score(swiss_metro_test.subset_vec, preds_outcome_Luce): .3}")
print(f"The F1 score of the generic aggregation was \
{f1_score(swiss_metro_test.subset_vec, preds_outcome_ind_trans):.3}")
```

```
The F1 score of the Luce aggregation was  0.598
The F1 score of the generic aggregation was 0.598
```

[27]:
```
random_probs = {
    'Swiss Metro': np.ones(len(test)) * (1/3),
    'Train': np.ones(len(test)) * (1/3),
```

```
    'Car': np.ones(len(test)) * (1/3)
}
log_reg_preds
print(f"The log loss for each alternative in the Logistic Regression reduction␣
 ↪was \n \
{log_loss(swiss_metro_test.subset_vec, log_reg_preds)}")
print(f"The log loss for each alternative in the Luce aggregation was \n \
{log_loss(swiss_metro_test.subset_vec, preds)}")
print(f"The log loss for each alternative in the generic aggregation was \n \
{log_loss(swiss_metro_test.subset_vec, ind_trans_preds)}")
print(f"The log loss for each alternative assigning random probability was \n \
{log_loss(swiss_metro_test.subset_vec, random_probs)}")
```

```
The log loss for each alternative in the Logistic Regression reduction was
 {'Swiss Metro_log_loss': 0.73, 'Train_log_loss': 0.44, 'Car_log_loss': 0.57}
The log loss for each alternative in the Luce aggregation was
 {'Swiss Metro_log_loss': 0.7, 'Train_log_loss': 0.38, 'Car_log_loss': 0.52}
The log loss for each alternative in the generic aggregation was
 {'Swiss Metro_log_loss': 0.71, 'Train_log_loss': 0.38, 'Car_log_loss': 0.52}
The log loss for each alternative assigning random probability was
 {'Swiss Metro_log_loss': 0.8, 'Train_log_loss': 0.5, 'Car_log_loss': 0.61}
```

[28]:
```
print(f"The t-test for H0: Luce aggregation = Generic aggregation \
{log_loss_compare_with_t_test(swiss_metro_test.subset_vec, preds,␣
 ↪ind_trans_preds)}")
print(f"The t-test for H0: Generic aggregation = random probability \
{log_loss_compare_with_t_test(swiss_metro_test.subset_vec, ind_trans_preds,␣
 ↪random_probs)}")
print(f"The t-test for H0: Generic aggregation = Logistic Regression \
{log_loss_compare_with_t_test(swiss_metro_test.subset_vec, ind_trans_preds,␣
 ↪log_reg_preds)}")
print(f"The t-test for H0: Generic aggregation = random probability \
{log_loss_compare_with_t_test(swiss_metro_test.subset_vec, ind_trans_preds,␣
 ↪random_probs)}")
```

```
The t-test for H0: Luce aggregation = Generic aggregation {'Swiss Metro': 0.02,
'Train': 0.74, 'Car': 0.27}
The t-test for H0: Generic aggregation = random probability {'Swiss Metro': 0.0,
'Train': 0.0, 'Car': 0.0}
The t-test for H0: Generic aggregation = Logistic Regression {'Swiss Metro':
0.18, 'Train': 0.0, 'Car': 0.0}
The t-test for H0: Generic aggregation = random probability {'Swiss Metro': 0.0,
'Train': 0.0, 'Car': 0.0}
```

```
../..\skpref\metrics\_classification.py:254: RuntimeWarning: divide by zero
encountered in log
  logged = np.log(predicted1[_alternative])
../..\skpref\metrics\_classification.py:258: RuntimeWarning: invalid value
```

```
encountered in multiply
  np.nan_to_num(logged * binarized_outcome) +
../..\skpref\metrics\_classification.py:261: RuntimeWarning: divide by zero
encountered in log
  logged2 = np.log(predicted2[_alternative])
../..\skpref\metrics\_classification.py:265: RuntimeWarning: invalid value
encountered in multiply
  np.nan_to_num(logged2 * binarized_outcome) +
```

Bibliography

Abo-Khamis, M., Im, S., Moseley, B., Pruhs, K., and Samadian, A. (2020). A relational gradient descent algorithm for support vector machine training. *arXiv preprint arXiv:2005.05325*.

Adam, L., Van Camp, A., Destercke, S., and Quost, B. (2020). Inferring from an imprecise Plackett–Luce model: application to label ranking. In *International Conference on Scalable Uncertainty Management*, pages 98–112. Springer.

Adams, E. S. (2005). Bayesian analysis of linear dominance hierarchies. *Animal Behaviour*, 69(5):1191–1201.

Adler, L. M. (1957). A modification of Kendall's tau for the case of arbitrary ties in both rankings. *Journal of the American Statistical Association*, 52(277):33–35.

Albanese, D. (2019). Mlpy github repository. https://github.com/manhtuhtk/mlpy. Accessed: 13/05/2019.

Aly, M. (2005). Survey on multiclass classification methods. *Neural Netw*, 19:1–9.

Antonini, G., Gioia, C., Frejinger, E., and Themans, M. (2007). Swissmetro: description of the data.

Armijo, L. (1966). Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3.

Arrow, K. J. (1951). *Social choice and individual values*, volume 12. Yale university press.

Assembly, L. (2016). London election results 2016, wards, boroughs, constituency. https://data.london.gov.uk/dataset/london-elections-results-2016-wards-boroughs-constituency. Accessed: 17/09/2020.

Bali, T. G. (2003). The generalized extreme value distribution. *Economics letters*, 79(3):423–427.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. (2022). Gradients without backpropagation. arXiv preprint arXiv:2202.08587.

Bekkar, M., Djemaa, H. K., and Alitouche, T. A. (2013). Evaluation measures for models assessment over imbalanced data sets. *J Inf Eng Appl*, 3(10).

Benson, A. R., Kumar, R., and Tomkins, A. (2018). A discrete choice model for subset selection. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 37–45.

Benzie, M. (2019). The longest an NCAA bracket has ever stayed perfect. https://www.ncaa.com/news/basketball-men/bracketiq/2018-03-28/longest-ncaa-bracket-has-ever-stayed-perfect. Accessed: 07/09/2020.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305.

Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR.

Beygelzimer, A., Dani, V., Hayes, T., Langford, J., and Zadrozny, B. (2005). Error limiting reductions between classification tasks. In *Proceedings of the 22nd international conference on Machine learning*, pages 49–56.

Bierlaire, M. (2003). Biogeme: A free package for the estimation of discrete choice models. In *Swiss Transport Research Conference*, number CONF.

Bierlaire, M. (2018). Pandasbiogeme: a short introduction. *Report TRANSP-OR: Lausanne, Switzerland*, page 181219.

Bierlaire, M., Axhausen, K., and Abay, G. (2001). The acceptance of modal innovation: The case of swissmetro. In *Swiss Transport Research Conference*, number CONF.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Borch, C. (2021). Machine learning and social theory: Collective machine behaviour in algorithmic trading. *European Journal of Social Theory*, pages 503–520.

Box, G. E. and Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243.

Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Brathwaite, T. (2019). pylogit github. https://github.com/timothyb0912/pylogit. Accessed: 06/08/2019.

Brathwaite, T. and Walker, J. L. (2018). Asymmetric, closed-form, finite-parameter models of multinomial choice. *Journal of Choice Modelling*, 29:78 – 112.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brock, W. A. and Durlauf, S. N. (2001). Discrete choice with social interactions. *The Review of Economic Studies*, 68(2):235–260.

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

Burton, M. and Rigby, D. (2009). Hurdle and latent class approaches to serial non-participation in choice models. *Environmental and Resource Economics*, 42(2):211.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136.

Caron, F. and Doucet, A. (2012). Efficient bayesian inference for generalized bradley–terry models. *Journal of Computational and Graphical Statistics*, 21(1):174–196.

Cattelan, M. (2012). Models for paired comparison data: A review with emphasis on dependent data. *Statistical Science*, pages 412–433.

Cattelan, M., Varin, C., and Firth, D. (2013). Dynamic bradley–terry modelling of sports tournaments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 62(1):135–150.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Chen, T., He, T., Benesty, M., Khotilovich, V., and Tang, Y. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4.

Chen, Y., Fan, J., Ma, C., and Wang, K. (2019). Spectral method and regularized MLE are both optimal for top-k ranking. *Annals of statistics*, 47(4):2204.

Cheng, W., Dembczynski, K., and Hüllermeier, E. (2010a). Label ranking methods based on the Plackett-Luce model. In *ICML*.

Cheng, W., Rademaker, M., De Baets, B., and Hüllermeier, E. (2010b). Predicting partial orders: ranking with abstention. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 215–230. Springer.

chess.com (2020). Yesterday our 50,000,000th member signed up. https://twitter.com/chesscom/status/1342982640366989318. Accessed: 25/05/2021.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

Colwell, D. and Gillett, J. (1982). Spearman versus Kendall. *The Mathematical Gazette*, 66(438):307–309.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Coulom, R. (2007). Computing "Elo ratings" of move patterns in the game of go. *ICGA Journal*, 30(4):198–208.

Coulom, R. (2008). Whole-history rating: A bayesian rating system for players of time-varying strength. In *International Conference on Computers and Games*, pages 113–124. Springer.

Croissant, Y. (2020). Estimation of random utility models in R: The mlogit package. *Journal of Statistical Software*, 95(11):1–41.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Debreu, G. (1960). *The American Economic Review*, 50(1):186–188.

Doignon, J.-P., Pekeč, A., and Regenwetter, M. (2004). The repeated insertion model for rankings: Missing link between two subset choice models. *Psychometrika*, 69(1):33–54.

Dominos (2022). Dominos pizza prices vs. sizes. https://www.dominos.co.uk/menu?start=true. Accessed: 20/08/2022.

Donoho, D. (2017). 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745–766.

Drori, I., Krishnamurthy, Y., Rampin, R., Lourenço, R., One, J., Cho, K., Silva, C., and Freire, J. (2018). Alphad3m: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*.

dunnhumby; Venkatesan Raj (2020). The complete journey. https://www.dunnhumby.com/source-files/. Accessed: 06/09/2020.

Élő, A. E. (1978). *The rating of chessplayers, past and present*. Arco Pub.

Engström, P. and Forsell, E. (2018). Demand effects of consumers' stated and revealed preferences. *Journal of Economic Behavior & Organization*, 150:43–61.

Escalante, H. J., Montes, M., and Sucar, L. E. (2009). Particle swarm model selection. *Journal of Machine Learning Research*, 10(2).

Evgeniou, T., Boussios, C., and Zacharia, G. (2005). Generalized robust conjoint estimation. *Marketing Science*, 24(3):415–429.

Fahrmeir, L. and Tutz, G. (1994). Dynamic stochastic models for time-dependent ordered paired comparison systems. *Journal of the American Statistical Association*, 89(428):1438–1449.

Falmagne, J.-C. and Regenwetter, M. (1996). A random utility model for approval voting. *Journal of Mathematical Psychology*, 40(2):152–159.

Faraggi, D. and Reiser, B. (2002). Estimation of the area under the ROC curve. *Statistics in medicine*, 21(20):3093–3106.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874.

Ferri, C., Hernández-Orallo, J., and Salido, M. A. (2003). Volume under the ROC surface for multi-class problems. In *European conference on machine learning*, pages 108–120. Springer.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322.

Ford, L. R. (1957). Solution of a ranking problem from binary comparisons. *The American Mathematical Monthly*, 64(8):28–33.

Forinash, C. V. and Koppelman, F. S. (1993). Application and interpretation of nested logit models of intercity mode choice. *Transportation research record*, (1413).

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Friedman, J. H. (1996). Another approach to polychotomous classification. *Technical Report, Statistics Department, Stanford University*.

Fry, T. R. and Harris, M. N. (1998). Testing for independence of irrelevant alternatives: some empirical results. *Sociological Methods & Research*, 26(3):401–423.

Fürnkranz, J. (2002). Round robin classification. *The Journal of Machine Learning Research*, 2:721–747.

Gillen, B., Montero, S., Moon, H. R., and Shum, M. (2015). BLP-Lasso for aggregate discrete choice models of elections with rich demographic covariates. *USC-INET Research Paper*, (15-27).

github (2008). Github version control website. https://github.com/.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26.

Goldstein, A. A. (1967). *Constructive real analysis*. Courier Corporation.

Google, D. (2020). Classification: ROC Curve and AUC. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Accessed: 20/12/2020.

Gormley, I. C. and Murphy, T. B. (2008). Exploring voting blocs within the Irish electorate: A mixture modeling approach. *Journal of the American Statistical Association*, 103(483):1014–1027.

Gruca, T. S. (1990). How regular is regularity? an empirical test of the regularity assumption. *ACR North American Advances*.

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., and Pedreschi, D. (2018). A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42.

Guiver, J. and Snelson, E. (2009). Bayesian inference for Plackett-Luce ranking models. In *proceedings of the 26th annual international conference on machine learning*, pages 377–384.

Guo, Y., Tian, P., Kalpathy-Cramer, J., Ostmo, S., Campbell, J. P., Chiang, M. F., Erdogmus, D., Dy, J. G., and Ioannidis, S. (2018). Experimental design under the Bradley-Terry model. In *IJCAI*, pages 2198–2204.

Gupta, P. (2019). *Learning Context-Dependent Choice Functions*. Paderborn University, Paderborn.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422.

Haerder, T. and Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM computing surveys (CSUR)*, 15(4):287–317.

Hanke, M., Halchemko, Y. O., and Oosterhof, N. N. (2019). pymvpa github repository. https://github.com/PyMVPA/PyMVPA. Accessed: 13/05/2019.

Harrington, J. L. (2016). *Relational Database Design and Implementation*, chapter 5,6. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition.

Harris, D. and Harris, S. L. (2010). *Digital design and computer architecture*. Morgan Kaufmann.

Hastie, T. and Tibshirani, R. (1997). Classification by pairwise coupling. *Advances in neural information processing systems*, 10:507–513.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hatzinger, R., Dittrich, R., et al. (2012). Prefmod: An R package for modeling preferences based on paired comparisons, rankings, or ratings. *Journal of Statistical Software*, 48(10):1–31.

Hoens, T. R., Polikar, R., and Chawla, N. V. (2012). Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1):89–101.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Hoo, Z. H., Candlish, J., and Teare, D. (2017). What is an ROC curve?

Hornik, K., Stinchcombe, M., White, H., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hornsby, A. N. and Love, B. C. (2020). How decisions and the desire for coherency shape subjective preferences over time. *Cognition*, 200:104244.

Huber, J., Payne, J. W., and Puto, C. (1982). Adding asymmetrically dominated alternatives: Violations of regularity and the similarity hypothesis. *Journal of consumer research*, 9(1):90–98.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17):1897–1916.

Hunter, D. R. et al. (2004). MM algorithms for generalized Bradley-Terry models. *The annals of statistics*, 32(1):384–406.

Hunter, D. R. and Lange, K. (2004). A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37.

IDSIA/CogBotLab (2019). Pybrain github repository. https://github.com/pybrain/pybrain. Accessed: 13/05/2019.

Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Jatana, N., Puri, S., Ahuja, M., Kathuria, I., and Gosain, D. (2012). A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology*, 1(6):1–5.

Jenkinson, A. F. (1955). The frequency distribution of the annual maximum (or minimum) values of meteorological elements. *Quarterly Journal of the Royal Meteorological Society*, 81(348):158–171.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Machine Learning Proceedings 1994*, pages 121–129. Elsevier.

Kaggle (2016). Expedia hotel recommendations. https://www.kaggle.com/c/expedia-hotel-recommendations/data. Accessed: 17/09/2020.

Kaggle (2019). Google Cloud and NCAA ML Competition 2019- Men's. https://www.kaggle.com/c/mens-machine-learning-competition-2019. Accessed: 06/09/2020.

Kaggle, C. G. (2017). Formula 1 race data. https://www.kaggle.com/cjgdev/formula-1-race-data-19502017. Accessed: 17/09/2020.

Kang, D.-K. and Kim, M.-J. (2015). Poisson model and bradley terry model for predicting multiplayer online battle games. In *2015 Seventh International Conference on Ubiquitous and Future Networks*, pages 882–887. IEEE.

Kanter, J. M. and Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE.

Kato, K., Li, Y., and Gupta, A. (2018). Compositional learning for human object interaction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 234–251.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154.

Kempthorne, O. (1952). The design and analysis of experiments.

Király, F. J. and Qian, Z. (2017). Modelling Competitive Sports: Bradley-Terry-Élő Models for Supervised and On-Line Learning of Paired Competition Outcomes. arXiv preprint arXiv:1701.08055.

Király, F. J., Löning, M., Blaom, A., Guecioueur, A., and Sonabend, R. (2021). Designing machine learning toolboxes: Concepts, principles and patterns. arXiv preprint arXiv:2101.04938.

Knerr, S., Personnaz, L., and Dreyfus, G. (1990). Single-layer learning revisited: a stepwise procedure for building and training a neural network. In *Neurocomputing*, pages 41–50. Springer.

Kohavi, R., John, G. H., et al. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324.

Kumar, A. (2016). santoku package. https://github.com/arunkk09/santoku. Accessed: 24/11/2020.

Kumar, A., Naughton, J., and Patel, J. M. (2015). Learning generalized linear models over normalized data. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1969–1984.

Labs, F. (2019). Representing data with entitysets. https://docs.featuretools.com/loading_data/using_entitysets.html. Accessed: 13/05/2019.

Lambooij, M. S., Harmsen, I. A., Veldwijk, J., de Melker, H., Mollema, L., van Weert, Y. W., and de Wit, G. A. (2015). Consistency between stated and revealed preferences: a discrete choice experiment and a behavioural experiment on vaccination behaviour compared. *BMC medical research methodology*, 15(1):19.

Lance Ashdown, T. K. (1993). *Oracle Database Concepts, 11g Release 2*. https://docs.oracle.com/cd/E11882_01/server.112/e40540.pdf.

Lee, P. H. and Philip, L. (2013). An R package for analyzing and modeling ranking data. *BMC medical research methodology*, 13(1):1–11.

Liu, N. N., Zhao, M., and Yang, Q. (2009). Probabilistic latent preference analysis for collaborative filtering. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 759–766.

Liu, T.-Y. (2011). *Learning to rank for information retrieval*. Springer Science & Business Media.

Löning, M., Bagnall, A., Ganesh, S., Kazakov, V., Lines, J., and Király, F. J. (2019). sktime: A Unified Interface for Machine Learning with Time Series. In *Workshop on Systems for ML at NeurIPS 2019*.

Löning, M., Bagnall, A., Ganesh, S., Kazakov, V., Lines, J., and Király, F. J. (2019). sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*.

Löning, M. and Király, F. (2020). Forecasting with sktime: Designing sktime's New Forecasting API and Applying It to Replicate and Extend the M4 Study. *arXiv preprint arXiv:2005.08067*.

Luce, D. (1959). *Individual Choice Behaviour*. John Wiley and Sons, New York.

Lutz, M. (2010). *Programming Python: powerful object-oriented programming*. " O'Reilly Media, Inc.".

Marschak, J. (1960). Binary-choice constraints and random utility indicators. In *Economic Information, Decision, and Prediction*, pages 218–239. Springer.

Maxion, R. A. and Roberts, R. R. (2004). Proper use of ROC curves in Intrusion/Anomaly Detection. *School of Computing Science Technical Report Series*.

Maystre, L. (2018). *Efficient Learning from Comparisons*. EPFL, Lausanne.

Maystre, L. (2019). choix package github repository. https://github.com/lucasmaystre/choix. Accessed: 22/12/2020.

Maystre, L. (2020). choix documentation.

McCullagh, P. and Nelder, J. (1989). *Generalized Linear Models, Second Edition*. Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. Chapman & Hall.

McFadden, D. (1978). Modeling the choice of residential location. *Transportation Research Record*, (673).

McFadden, D. (2001). Economic choices. *American economic review*, 91(3):351–378.

McHale, I. and Morton, A. (2011). A Bradley-Terry type model for forecasting tennis match results. *International Journal of Forecasting*, 27(2):619–630.

McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

Meyer, B. (1997). *Object-oriented software construction*, volume 2. Prentice hall Englewood Cliffs.

Michailidis, M. (2017). *Investigating machine learning methods in recommender systems*. PhD thesis, UCL (University College London).

Nagler, J. (1994). Scobit: An alternative estimator to logit and probit. *American Journal of Political Science*, pages 230–255.

Narkhede, S. (2020). Understanding confusion matrix. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62. Accessed: 20/12/2020.

NCAA (2020). What is march madness: The NCAA tournament explained. https://www.ncaa.com/news/basketball-men/bracketiq/2020-04-20/what-march-madness-ncaa-tournament-explained. Accessed: 06/09/2020.

Negahban, S., Oh, S., and Shah, D. (2017). Rank centrality: Ranking from pairwise comparisons. *Operations Research*, 65(1):266–287.

Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384.

Netfix (2006). Netflix prize. https://www.netflixprize.com/. Accessed: 01/12/2020.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

Öztürké, M., Tsoukiàs, A., and Vincke, P. (2005). Preference modelling. In *Multiple criteria decision analysis: State of the art surveys*, pages 27–59. Springer.

Papp, I., Király, F., Manolopoulou, I., Pfannschmidt, K., and Gupta, P. (2021). skpref github. https://github.com/skpref/skpref.

Pathak, B., Garfinkel, R., Gopal, R. D., Venkatesan, R., and Yin, F. (2010). Empirical analysis of the impact of recommender systems on sales. *Journal of Management Information Systems*, 27(2):159–188.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perens, B. et al. (1999). The open source definition. *Open sources: voices from the open source revolution*, 1:171–188.

Petty, M. D. and Weisel, E. W. (2019). Model composition and reuse. In *Model Engineering for Simulation*, pages 57–85. Elsevier.

Pfannschmidt, K., Gupta, P., and Hüllermeier, E. (2018). Deep architectures for learning context-dependent ranking functions. arXiv preprint arXiv:1803.05796.

Pfannschmidt, K., Gupta, P., and Hüllermeier, E. (2019a). Learning choice functions: Concepts and architectures. arXiv preprint arXiv:1901.10860.

Pfannschmidt, K., Gupta, P., and Hüllermeier, E. (2019b). Learning choice functions: Concepts and architectures. *CoRR*, abs/1901.10860.

Plackett, R. L. (1975). The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202.

Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Price, D., Knerr, S., Personnaz, L., and Dreyfus, G. (1995). Pairwise neural network classifiers with probabilistic outputs. In *Advances in neural information processing systems*, pages 1109–1116.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rao, P. and Kupper, L. L. (1967). Ties in paired-comparison experiments: A generalization of the Bradley-Terry model. *Journal of the American Statistical Association*, 62(317):194–204.

Ray, P. (1973). Independence of irrelevant alternatives. *Econometrica: Journal of the Econometric Society*, pages 987–991.

Refregier, P. and Vallet, F. (1991). Probabilistic approach for multiclass classification with neural networks. In *Artificial Neural Networks*, pages 1003–1006. Elsevier.

requeijaum (2019). statsmodels github issues page. https://github.com/statsmodels/statsmodels/issues/4160?fbclid=IwAR0gXIWOVXbbuty3i-I4dWHWwToNy6YsThTZhkLqaF6Qd9zMOrqCiSKvg20#issuecomment-398098959. Accessed: 06/08/2019.

Riefer, P. S., Prior, R., Blair, N., Pavey, G., and Love, B. C. (2017). Coherency-maximizing exploration in the supermarket. *Nature human behaviour*, 1(1):1–4.

Rodrigues, F. (2019). Reviewing every barack obama march madness bracket. https://camd.northeastern.edu/gameplan/2019/03/20/reviewing-every-barack-obama-march-madness-bracket/. Accessed: 07/09/2020.

Sagiroglu, S. and Sinanc, D. (2013). Big data: A review. In *2013 international conference on collaboration technologies and systems (CTS)*, pages 42–47. IEEE.

Samuelson, P. A. (1938). A note on the pure theory of consumer's behaviour. *Economica*, 5(17):61–71.

Schäfer, D. and Hüllermeier, E. (2015). Dyad ranking using a bilinear Plackett-Luce model. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 227–242. Springer.

Schattschneider, E. (1960). The semi-sovereign people (new york: Holt, rinehart andwinston, 1960). *SchattschneiderThe Semi-Sovereign People1960*.

Schauberger, G. and Tutz, G. (2019). Btllasso: a common framework and software package for the inclusion and selection of covariates in Bradley-Terry models. *Journal of Statistical Software*, 88(9).

Scikit learn online documentation (2019). Contributing. https://scikit-learn.org/stable/developers/contributing.html. Accessed: 13/05/2019.

Seabold, S. and Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.

Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656.

Simonson, I. (1989). Choice based on reasons: The case of attraction and compromise effects. *Journal of consumer research*, 16(2):158–174.

Simonson, I. and Tversky, A. (1992). Choice in context: Tradeoff contrast and extremeness aversion. *Journal of marketing research*, 29(3):281–295.

Sonnenburg, S., Strathmann, H., Lisitsyn, S., Gal, V., García, F. J. I., Lin, W., De, S., Zhang, C., frx, tklein23, Andreev, E., JonasBehr, sploving, Mazumdar, P., Widmer, C., Zora, P. D. ., Toni, G. D., Mahindre, S., Kislay, A., Hughes, K., Votyakov, R., khalednasr, Sharma, S., Novik, A., Panda, A., Anagnostopoulos, E., Pang, L., Binder, A., serialhex, and Esser, B. (2019). Shogun github repository. https://github.com/shogun-toolbox/shogun. Accessed: 13/05/2019.

Soufiani, H. A. and Chen, W. (2013). Statrank: Statistical rank aggregation: Inference. *Evaluation, and Visualization*.

Spearman, C. (1904). The proof and measurement of association between two things.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133.

Strobl, C., Wickelmaier, F., and Zeileis, A. (2011). Accounting for individual differences in Bradley-Terry models by means of recursive partitioning. *Journal of Educational and Behavioral Statistics*, 36(2):135–153.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Takane, Y. (1981). Maximum likelihood estimation in the generalized case of thurstone's model of comparative judgment. *Japanese Psychological Research*, 22(4):188–196.

Thurstone, L. L. (1927a). A law of comparative judgment. *Psychological Review*, 34(4):273–286.

Thurstone, L. L. (1927b). The method of paired comparisons for social values. *The Journal of Abnormal and Social Psychology*, 21(4):384–400.

Thurstone, L. L. (1927c). Psychophysical analysis. *The American Journal of Psychology*, 38(3):368–389.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tibshirani, R. (2011). Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282.

Train, K. E. (2009). *Discrete choice methods with simulation*. Cambridge university press.

Turner, H., Firth, D., et al. (2012). Bradley-Terry models in R: the BradleyTerry2 package. *Journal of Statistical Software*, 48(9).

Turner, H. L., van Etten, J., Firth, D., and Kosmidis, I. (2020). Modelling rankings in R: The PlackettLuce package. *Computational Statistics*, pages 1–31.

Tutz, G. (1986). Bradley-Terry-Luce models with an ordered response. *Journal of mathematical psychology*, 30(3):306–316.

Tversky, A. (1972). Elimination by aspects: A theory of choice. *Psychological review*, 79(4):281.

Urama, K. C. and Hodge, I. D. (2006). Are stated preferences convergent with revealed preferences? empirical evidence from Nigeria. *Ecological Economics*, 59(1):24–37.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. 13:22–2830.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Varian, H. R. (2006). Revealed preference. *Samuelsonian economics and the twenty-first century*, pages 99–115.

Varian, H. R. (2010). *Intermediate Microeconomics: A Modern Approach: Ninth International Student Edition*. WW Norton & Company.

Vasanen, A. (2012). Beyond stated and revealed preferences: the relationship between residential preferences and housing choices in the urban region of Turku, Finland. *Journal of Housing and the Built Environment*, 27(3):301–315.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

von Haefen, R. H., Massey, D. M., and Adamowicz, W. L. (2005). Serial nonparticipation in repeated discrete choice models. *American Journal of Agricultural Economics*, 87(4):1061–1076.

Wang, T. and Lin, Q. (2019). Hybrid predictive model: When an interpretable model collaborates with a black-box model. *arXiv preprint arXiv:1905.04241*.

Wang, Z., Albarghouthi, A., and Jha, S. (2020). Abstract universal approximation for neural networks. *arXiv preprint arXiv:2007.06093*.

Wasserman, L. (2013). *All of statistics: a concise course in statistical inference*. Springer Science & Business Media.

Wauthier, F., Jordan, M., and Jojic, N. (2013). Efficient ranking from pairwise comparisons. In *International Conference on Machine Learning*, pages 109–117. PMLR.

Wickelmaier, F. and Schmid, C. (2004). A matlab function to estimate choice model parameters from paired-comparison data. *Behavior Research Methods, Instruments, & Computers*, 36(1):29–40.

Wilco, D. (2020). The absurd odds of a perfect NCAA bracket. https://www.ncaa.com/news/basketball-men/bracketiq/2020-01-15/perfect-ncaa-bracket-absurd-odds-march-madness-dream#:~:text=Virtually%20all%20bracket%20pools%20disregard,That's%209.2%20quintillion. Accessed: 07/09/2020.

Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235.

Wu, T.-F., Lin, C.-J., and Weng, R. C. (2004). Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5(Aug):975–1005.

XGBoost developers (2019). Xgboost github. https://github.com/dmlc/xgboost. Accessed: 13/05/2019.

Zeileis, A., Strobl, C., Wickelmaier, F., and Kopf, J. (2011). psychotree-recursive partitioning based on psychometric models: Version 0.12-1.

Zermelo, E. (1929). Die berechnung der turnier-ergebnisse als ein maximumproblem der wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, 29(1):436–460.

Zhou, K., Xue, G.-R., Zha, H., and Yu, Y. (2008). Learning to rank with ties. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–282.

Zhu, X. and Klabjan, D. (2020). Listwise learning to rank by exploring unique ratings. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 798–806.

Zito, T., Wilbert, N., Wiskott, L., and Berkes, P. (2019). MDP github repository. https://github.com/mdp-toolkit/mdp-toolkit. Accessed: 13/05/2019.

Zöller, M.-A. and Huber, M. F. (2019). Benchmark and survey of automated machine learning frameworks. *arXiv preprint arXiv:1904.12054*.