Investigating machine learning methods in recommender systems

Improving prediction for the top K items

Marios Michailidis

Supervisor: Professor Philip Treleaven
Supervisor: Giles Pavey

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
of the
University College London

May 2017

Declaration

I, Marios Michailidis, 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.

……

Marios Michailidis

Abstract

This thesis investigates the use of machine learning in improving predictions of the top K[1] product purchases at a particular a retailer. The data used for this research is a freely-available (for research) sample of the retailer's transactional data spanning a period of 102 weeks and consisting of several million observations. The thesis consists of four key experiments:

1. **Univariate Analysis of the Dataset:** The first experiment, which is the univariate analysis of the dataset, sets the background to the following chapters. It provides explanatory insight into the customers' shopping behaviour and identifies the drivers that connect customers and products. Using various behavioural, descriptive and aggregated features, the training dataset for a group of customers is created to map their future purchasing actions for one specific week. The test dataset is then constructed to predict the purchasing actions for the forthcoming week. This constitutes a univariate analysis and the chapter is an introduction to the features included in the subsequent algorithmic processes.

2. **Meta-modelling to predict top K products:** The second experiment investigates the improvement in predicting the top K products in terms of precision at K (or precision@K) and Area Under Curve (AUC) through meta-modelling. It compares combining a range of common machine learning algorithms of a supervised nature within a meta-modelling framework (where each generated model will be an input to a secondary model) with any single model involved, field benchmark or simple model combination method.

3. **Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment:** The third experiment demonstrates a hybrid methodology of cross validation, modelling and optimization for improving the accuracy of predicting the products the customers of a retailer will buy after having

[1] Top K or Top-K Recommendations is an industry term that recommenders use to describe the K most likely products a customer will buy [Yang et al. 2012] in the future. K is commonly an integer smaller than 20.

bought them at least once with a promotional coupon. This methodology is applied in the context of a train and test environment with limited overlap - the test data includes different coupons, different customers and different time periods. Additionally this chapter uses a real life application and a stress-test of the findings in the feature engineering space from experiment 1. It also borrows ideas from ensemble (or meta) modelling as detailed in experiment 2.

4. **The StackNet model:** The fourth experiment proposes a framework in the form of a scalable version of [Wolpert 1992] stacked generalization being extended through cross validation methods to many levels resembling in structure a fully connected feedforward neural network where the hidden nodes represent complex functions in the form of machine learning models of any nature. The implementation of the model is made available in the Java programming language.

The research contribution of this thesis is to improve the recommendation science used in the grocery and Fast Moving Consumer Goods (FMCG) markets. It seeks to identify methods of increasing the accuracy of predicting what customers are going to buy in the future by leveraging up-to-date innovations in machine learning as well as improving current processes in the areas of feature engineering, data pre-processing and ensemble modelling. For the general scientific community this thesis can be exploited to better understand the type of data available in the grocery market and to gain insights into how to structure similar machine learning and analytical projects. The extensive, computational and algorithmic framework that accompanies this thesis is also available for general use as a prototype to solve similar data challenges.

Acknowledgements

I would like to express my gratitude to my supervisors at UCL, Prof. Philip Treleaven and Giles Pavey for their guidance and for making this research possible with their friendly support, patience and domain expertise.

I would like to express my gratitude to UCL, colleagues and academic staff for providing me with the necessary knowledge, resources and support in order to be able to conduct this research.

I would like to acknowledge dunnhumby ltd for sponsoring this research project as well as for providing the rich data required to conduct the experiments.

I would also like to thank Gert Jacobusse for his great mentoring and collaboration through various machine learning challenges including the Acquire Valued Shoppers' challenge.

Last but not least, I would like to seize the opportunity to dedicate this thesis to my family and specifically my father Savvas Michailidis and my mother Androniki Kazani, who showed love and support and have been the rocks I have been leaning on throughout my life.

Contents

List of Figures

List of Tables

1. Introduction

This chapter presents an overview of the current thesis. It introduces the thesis topic and outlines my motivations for conducting the research. It then sets out the specific objectives of the research and introduces the experimental methodology.

1.1 Motivation

Background: Driven by large organisations keen to use data to develop a deep understanding of what products their customers like to buy, recommendation science has received increased attention over the last decade. The extensive number of publications [Sarwar Badrul et al. 2001] in collaborative filtering as well as the emergence of big-prize data challenges such as the Netflix competition offering $1m to the winning entry [netflix prize 2009], indicate just how valuable businesses regard their capability to predict customer tastes and behaviours with increasing levels of accuracy.

Univariate Analysis of the Dataset: Whist machine learning algorithms have gone a long way in identifying deep relationships within and extracting great predictive power from structured and unstructured data [Arel et al. 2010], exploring the underlying data through univariate analysis has commonly provided useful sources for better feature engineering and improvements in the prediction process [Domingos 2012]. Therefore, to better improve recommendations it seems a credible step to analyse and comprehend what drives the shopping process with the aim of identifying rudimentary links that connect customers to specific items. Such knowledge can later be utilised in algorithmic frameworks and modelling experiments. It can also serve as a preliminary step to understanding the underlying data utilised by this thesis.

Meta-modelling to predict top K products: Recommendation science in a generic form has had great success in providing a means to efficiently link customers to items [Herlocker et al. 2004]. The various works of [Yehuda Koren 2009] on factorization machines and collaborative

filtering stand as great innovations in the field. Another inspiring example is the work of [Salakhutdinov et al. 2007] in implementing Boltzmann's neutral networks to more accurately predict the products customers will buy.

Significant advances have been made from the era when limited data could be used for data analysis [Gandomi et al. 2015]. Increased storage capabilities have allowed companies to store extra details about transactions, as well as contextual data such as weather, time of day and sales channel. [Karatzoglou 2010] has successfully integrated such contextual information while performing collaborative filtering experiments. More recent work, specifically in the use of learning-to-rank algorithms by [Weston et al. 2013] show the potential of experimenting with new approaches in the field.

The relationship of customer to product in the form of feature interaction has been widely investigated, with approaches using collaborative filtering being amongst the most notable examples [Koren et al. 2009]. However, the (online) grocery environment is special when it comes to efficient recommendations in the sense that customers tend to choose their favourite or usual items (instead of actively looking for new challenges). Indeed, past purchase frequency is the overwhelming element in defining what will be bought. Therefore knowing how many times customer A bought product B in a fixed period of time can yield a very good prediction as to whether the customer will buy the item again within a certain time-frame (such as next week) [Boyet et al. 2005]. Additionally, there is only a certain number of recommendations a retailer can make and since the frequency with which a customer purchases an item is such an important predicto, most of these recommendations are determined by simply knowing this feature. Simple and efficient it may be, there is opportunity to significantly improve the recommendation list. This can be achieved via making use of the extensive frameworks of different machine learning families as well as leveraging the best practices in machine learning. Additionally there is room for investigating the impact of any extra features (aside from frequency of purchase) that have influence in determining the customers' next purchases.

Many of the aforementioned techniques such as recommendation science or collaborative filtering utilize machine learning. As a separate field, machine learning is also receiving increased attention due to the increases in potential that advances in computational speed brings [Mjolsness 2001]. Many techniques and approaches have been developed and are available as open-source solutions. Many of these are referred to in [Witten et al. 2005] and [Michalski 1998]. They list a large number of supervised and unsupervised machine learning techniques

including, decision trees, random forests, gradient boosted trees, Naïve Bayes classifier, neural networks, logistic regression, k nearest neighbours, support vector machines and derivatives of these methods. According to [Mohri et al. 2012] supervised algorithms are trained on labelled data, which means the ground truth is known for a given case. These techniques are discussed in detail in chapter 2.

Machine learning has evolved to include many such tools, models or general processes for creating predictive algorithms. Some machine learning approaches combine multiple algorithms to improve predictions. These approaches are known as ensemble methods and have been applied in many industry and research fields [Tan et al. 2003], [Dietterich 2000] often producing better results than single algorithm approaches. It is therefore speculated that a possible way to improve predictions in respect to what the customer will buy in the future is to utilize ensemble methods and find the right mixture of different machine learning models that by nature tend to capture different forms of interactions, be it linear or non-linear. An ensemble methodology would promote intuitiveness and possibly yield solid results in predicting what the customer will buy in the future, since instead of focusing on one approach (that has certain advantages and disadvantages), the focus could be shifted to leveraging the advantages of all methods included.

Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment: A popular application of recommendation science in the grocery (or FMCG) field is the allocation of promotional coupons as it is considered a very efficient way to increase customer loyalty. Consumers in the United States of America saved $3.8 billion in 2002 by shopping with coupons [Michelle Rubrecht 2014] which means that improvements in allocation efficiency (in terms of identifying the best coupon to customer matches) could have a huge impact in loyalty generation and in turn the retailer's bottom line.

In recommender systems, predictive models often need to be built on small subsets of customers and products with incomplete, sparse or limited data. [Hu et al. 2006] addressed this problem by proposing a hybrid user and item-based collaborative system. Additionally, predictive algorithms, irrespective of the limitations on which they have been built, need to be able to extrapolate and generalize in unforeseen environments. [Lika et al. 2014] addresses the cold start problem in recommender systems needing to make predictions for new customers and products. [Garcin et al. 2014] highlights the difference between offline and online accuracy

evaluations, demonstrating that the best recommendation strategy may be different for the instore and online environments.

Ultimately, the main objective of this experiment is to improve coupon allocation (as a means to boosting customer loyalty) within the context of an irregular testing environment. The irregularity of this environment is that predictive algorithms have to be built with limited data or with subsets of customers and products meaning that recommendations are tested on different customers and different promotional products to those which are used to train the algorithm.

The StackNet model: It has been almost 25 years since [Wolpert 1992] introduced stacked generalization (or stacking) as a way to combine the predictions of multiple machine learning models using another (Meta) model. Until today there has not been a prominent software implementation of this algorithm although the advent in computing power allows the running of multiple machine models in parallel. At the same time deep learning has (re)surfaced [Schmidhuber 2015] as a strong predictive algorithm and through its multiple hidden layers and neuron synapses, it can exploit deep relationships inherent within the data. Combining the two methodologies of stacking and deep learning could therefore potentially yield uplift in the performance of machine learning tasks. Such an approach would require several algorithms to be available, however it could hardly be stated that there is a package or library that has everything (even the common ones). For the average data scientist it requires an extensive skillset of multiple programming tools, software and other resources to be able to leverage the benefits of different techniques. Therefore availability of the algorithms is still a fundamental factor in building better prediction modes and stimulating science.

The focus in recommendation and data science in general has not only been in making smarter (as in more accurate and/or inspiring) recommendations. There is an ever increasing appetite to make recommendation generation faster, more memory efficient and more automated. In order to leverage today's big data, scalability in both recommendation and machine learning science is vital [Zhao et al. 2002]. Even the most comprehensive packages (such as R) that have a great variety of different techniques, are still lacking when it comes to scalability (for some algorithms) thus making the use of big data (with hundreds of millions of records) problematic. There is an irreversible move towards bigger and bigger data ([Mims 2010] and [Sutter 2005]) and CPU's are not getting fast enough at a fast enough rate to keep up with

scaling requirements. The only way to keep up with increasingly fast-paced environments is to make applications scalable by dividing each task into different threads.

The main motivation for using the StackNet model is to create a methodology that uses stacked generalization and applies it within a neural network architecture where the inputs of any nodes could be any machine learning model (and not just perceptions as is commonly the case). By using this approach, one would expect to yield better generalization results in various domains, including the recommendation space.

In conclusion my motivations for the current thesis can be summarised (per chapter) as:

- **Univariate Analysis of the Dataset:** To understand the factors or features (other than frequency of purchase) that drive customers to buy certain products.
- M**eta-modelling to predict top K products:** To improve recommender systems, especially for the top K items (that the customer has probably bought many times), by using cutting edge machine learning, leveraging a variety of different algorithms and approaches within a meta modelling framework and to prove that such a methodology can overcome any single model approach involved or simple ensemble method.
- **Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment:** To predict, using a hybrid method within the context of an irregular testing environment of different customers and different offers, whether customers will buy a product again after receiving a coupon for it.
- **The StackNet model:** To demonstrate that the stacked generalization method applied within a neural network framework can achieve higher levels of accuracy and to introduce new scalable applications in the scientific community as a means to further extend academia's capabilities in predictive modelling classification tasks (including the ones related to recommendation science).

1.2 Objectives of this Research

The overall goal of this thesis is to apply ensemble machine learning approaches to improve recommendations. The data used for this research is a freely-available sample of the retailer's transactional data (also enriched with many descriptive fields) and consisting of several million observations. Each experiment contributes to the overall goal in unique ways which are further analysed below:

Univariate analysis of the dataset: The main objective from this experiment is to better understand the drivers that define next purchases for customers and improve predictions for the top K (commonly 10) items they will buy in the following week. The research will use engineered features that have to do with customers' past purchasing history as well as general descriptive fields and will associate them with the propensity to buy a product in the following week. In this way a simple but insightful indication will be derived for each feature in the dataset. To facilitate capturing more information from the input data, an optimized binning technique will be deployed.

Meta-modelling to predict top K products: This chapter investigates the uplift from using a stacked generalization approach to predicting what the customers of a retailer will buy in the following week. This uplift is measured against any single algorithm model used in the stacking model, all field benchmarks and other simple ensemble approaches such as model averaging. The underlying premise is that more accurate recommendations will yield value to the customers since the recommended items are more likely to be relevant to them. A more relevant customer experience will in turn produce loyalty to the retailer and product brand.

Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment: The third experiment borrows elements from the previous two experiments in regards to feature engineering and model ensembling within a retail environment. It aims to improve recommendations in such environments by predicting with a hybrid modelling methodology which products the customers will buy again after having redeemed a coupon for them at least once. Furthermore it aims to tackle the problem of an irregular testing environment of different customers and offers by proposing a novel cross-validation methodology to measure and improve the accuracy of the predictive algorithms, the usefulness of derived features, the tuning of the algorithms' hyper parameters and the overall modelling process in general.

The StackNet model: The objective of the final experiment, namely the StackNet model is to provide an algorithmic implementation of Wolpert's stacked generalization within a feedforward neural network architecture to efficiently combine multiple machine learning models with the scope of improving accuracy in classification (and recommendation) problems. Apart from the methodology, along with the technical considerations, another aim of the thesis is to provide the algorithmic software infrastructure (in the Java programming language) to run all the algorithms in the form of a new library that could be accessed by anyone. The software will support multiple algorithms used for research, along with data pre-processing steps, feature engineering capabilities, data transformations and cross-validations methods. Therefore as an additional objective this tool aims to offer more options in multi-algorithmic approaches for large-scale problems.

The multiple objectives of this research can be summarised as follows:

- **Univariate Analysis of the Dataset:** Understand the retailer's available data set, especially in respect to its predictive power in determining what the customers are going to buy in the future.
- **Meta-modelling to predict top K products:** Leverage the benefits arising from multiple machine learning techniques and ensemble methodologies such as stacking to make more accurate recommendations as measured via multiple metrics against numerous single models, field benchmarks such as product popularity or simple ensemble methods like model averaging.
- **Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment:** Improve accuracy in predicting which products the customers will buy after having redeemed an offer for them, using a hybrid modelling methodology, assuming an irregular testing environment of different products and customers.
- **The StackNet model:** Provide an implementation of stacked generalization [Wolpert 1992] within a neural network framework as a means to combine multiple diverse models to improve the accuracy in classification tasks.

1.3 Research Methodology

The following sections explain the research methodology utilised in each of the chapters in the current thesis.

1.3.1 Univariate Analysis of the Dataset

To achieve its objectives, this thesis will use the freely available (for research) data of a big retailer in the grocery space. In summary this dataset contains a stream of customers' transactions for the period of 102 weeks. Multiple fields are known for each transaction such as the time of purchase, item price, quantity of products purchased, discounts applied, and whether the product was on promotion or not. The dataset also contains hierarchical information about the products as well as other descriptive information about the customers such as age group and/or household type.

The actual experiment will use a portion of this data (54 weeks), covering the period of week 47 to week 101. One year of transactional data was deemed enough to create the modelling datasets. The overall expectation is to learn how the aforementioned features contained within the dataset define future purchases. That is, if all these variables are known for a period of 52 weeks (from 47 to 99), to determine if it is possible to predict what the customers are going to buy in the weeks after week 99, in other words week 100 and week 101 respectively.

This chapter will initially use basic descriptive analysis in the form of basic statistics and explanatory graphs to examine the distribution of certain customer characteristics, product features and other variables. The supervised metric of AUC (Area Under the roc Curve) will then be utilized to gauge the strength of the binary target variable (as in "will buy" or "will not buy") with each one of the possible predictors. The volume of the data available will allow a thorough, statistically significant and comprehensive investigation that could easily be generalized across other similar grocery retail environments. It is therefore a rare and valuable research opportunity.

Finally, this experiment treats some of the variables as nonlinear and will use an optimized binning method (based on the AUC metric as mentioned above) to capture these non-linearities

and replace them with the log of the odds of the target variable in such a way that the relationship with the target variable can be linearized and its predictive power better captured. Furthermore the uplift in AUC is estimated for all variables considered both in absolute and proportional terms, before and after the optimized binning method is applied. The end result of this chapter will be the ranking (based on AUC) of how predictive the individual feature, or its respective family is, in determining whether the customers are going to buy a specific product the following week.

1.3.2 Meta-modelling to predict top K products

The same data sources will be used for this experiment. The question to be answered is whether predictions for the top K products the customers are going to buy the following week can be improved by using a meta-modelling approach versus all single-models involved, simple ensemble methodologies or field benchmarks. The main hypothesis is that by combing multiple machine learning methods that are different in nature (and therefore possess different advantages), a significantly better solution will be achieved than if a single-method approach were used.

The combination of models will be made via a secondary model that will use the previous methods as inputs. Undertaking such a method aims to leverage the advantages of all the different applied methodologies to reach a more generalizable solution to this classification problem. The supervised techniques to be applied will include linear regression, logistic regression, decision trees, random forests, gradient boosting machines, multilayer perception (neural networks), kernel-based models and factorization machines. Some of these methods include a stage of feature selection and may further include some data transformation processes such as scaling and outlier removal.

The training set formed includes the creation of a number of aggregated features based on the transactional data for the period of week 47 to week 99 and the target to predict is a binary indicator that shows whether the item under consideration is bought the following week (100) by a given customer. The test set uses the period of weeks 48 to 100 and the target week of 101. The training/validation split will be 80-20. The same split of data is being used to both tune the models' hyper parameters and then to make predictions. The criterion to optimize is precision K [5 10 20] and Area Under the ROC Curve (AUC). Apart from the aforementioned

algorithms, a series of different benchmarks such as product popularity and customer's frequency of purchase per item, are also derived to facilitate comparisons among the different models' results.

After all models are fitted and all predictions are made and saved for both the validation data and the (future) test data, a random forest model will be used to combine all the predictions of the validation data as inputs to maximize AUC and precision K for the test data. The final tuning of this meta model is attained using a 5-fold cross validation. The performance of this model is compared against all single models involved, the created benchmark and simple combinations of the single models such as normal average and rank-transformed average.

1.3.3 Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment

Given a set of customers of a retailer along with a subset of their past transactions where each customer has received and redeemed a coupon, a predictive modelling methodology is applied to improve predictions of whether the customers will buy the redeemed product again in the future. The dataset is divided into 2 parts. 160,000 are used for training the model and the remainder are used for testing. The datasets have minimal overlap between them as they include mostly different offers and different customers (and refer to different time periods). The objective is to maximize the AUC (Area Under the roc Curve) of whether a customer will repurchase a product previously bought via a coupon recommendation.

This experiment investigates different cross validation methodologies to tackle the small overlap between the training and test data in order to maximize AUC, ensuring that a model will be able to generalize well in unobserved data. It further demonstrates the internal cross validation results of each methodology on a subset of features generated from the transactional history of customers along with the actual results they yield in the test data. The first validation methodology includes a random stratification of the training data based on offer so that each offer is equally (proportion-wise) represented in any train and validation splits. The other methodology ensures that splits are based on the number of offers where N-1 (out of N) offers is used to build models and maximize the AUC in the n_{th} offer. The last methodology adds

another step to the previous methodology via merging all predictions from all N offers together before estimating a global AUC out of all offers' predictions.

Furthermore it creates different recommendation methodologies, one content-based and another based on collaborative filtering to generate a hybrid methodology. The first (content-based) approach assumes that the prediction of whether the customer will buy the item or not is dependent on the direct relationship he or she has with the item (i.e if it or items from the same brand were bought in the past). The model of choice Ridge Regression [Tikhonov 1977] trains on the actual number of times the customer bought the item after the offer date.

The second (collaborative filtering) approach assumes that the propensity of an item to be purchased by a customer is strongly related to the likelihood that the customer belongs to the group of customers that like the item and would have bought it even if they had never received an offer for it. In this approach the target variable is created from the transactional data by taking the natural logarithm of how many times the item was bought 90 days prior to the offer date. Separate models are then trained on each different. The model of choice was gradient boosting trees [Breiman 1997] and many features were generated from the transactional history as well as by using deep learning and Restrictive Boltzmann Machines (RBMs) [Smolensky 1986] from the raw data.

Given both models have been trained on different target variables, the predictions are transformed to ranks before combining them. The final hybrid model uses an average of the two approaches (after applying the rank transformation).

1.3.4 The StackNet Model

The StackNet Model attempts to leverage the benefits of various machine learning algorithms and approaches in order to maximize performance against various accuracy metrics. The underlying architecture of the models and how they are connected with each other is very similar to what is found in a feedforward neural network. Each trained model is a node in a modelling architecture of various layers starting from models trained directly from the input data (which constitutes the input layer). Each new layer then uses as inputs the predictions (or outputs) of the previous layer until the final output is reached (which may be zero or one in a binary problem).

[Wolpert 1992] proposed the stacked generalization methodology as a means to combine the predictions from many different neural network models by using a holdout set. This new set of features forms a new dataset that is trained with another neural network in order to improve the performance in the test data. The ability of a model to generalize in unseen data may be sensitive to the number of available observations, hence a modelling architecture of multiple levels would have to be constantly splitting the training data in order to generate unbiased samples. A critical suggestion is to be able to re-use the initial training dataset multiple times without compromising the integrity in the way information of the target variable is being carried to multiple levels. This research demonstrates a k-fold cross-validation paradigm to reconstruct the initial training data with predictions of a given algorithm.

Traditional neural networks have various ways of reaching convergence (such as back propagation). However, in the StackNet architecture, each model is validated on holdout data which is later used for further modelling (as features), which means that traditional modelling through various epocs (or iterations) would not make much difference in the final outcome versus optimizing the hyper parameters of the selected model-features. To accelerate convergence this research proposes two different types of connections among the different layers, one that assumes a direct forward connection from the models of one layer to the next and another that requires each layer to include as inputs all models from previous layers.

Finally aside from the theoretical underpinnings of this methodology, the effectiveness of the algorithm can be better comprehended with an actual implementation. Therefore multiple algorithms will be re-implemented in the Java programming language, leveraging multi-threaded technologies to create a machine learning library for the implementation of the StackNet model.

1.4 Research Contribution

This section will be divided into four parts to align with the experiments as defined in the abstract. This research contributes to existing literature in a number of new ways.

Univariate Analysis of the Dataset: The *Complete Journey* dataset [dunnhumby 2014] contains datasets of customer transactions from the grocery (or FMCG) field of a large enough breadth and volume for descriptive analysis to be considered robust and statistically significant.

This allows for better understanding of and thorough mapping of the average retail customer and enables credible insight into the factors that link customers to future purchases to be derived. The non-linearity of certain features in respect to future purchases is addressed using an optimized binning methodology. This will also facilitate future modelling of these features.

Meta-modelling to predict *top K* products: This chapter focuses on improving prediction offuture purchases using a meta-modelling approach taking advantage of a portion of the main algorithmic families that have been developed (or resurfaced) over the last decade in machine learning. Its novelty is derived from demonstrating that such an approach can outperform any single model involved in the mix, any simple model combination method or field benchmark in the grocery recommendation space. This is highlighted against the metrics of AUC and precision at K (or precision@K).

Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment: The distribution of coupons is a common challenge for recommenders to optimize in the grocery field as customer satisfaction and loyalty are influenced by it. Often such recommender systems need to be built with limited or a subset of data and be able to extrapolate well to unseen environments of different customers and offered products. This research proposes an N-offer cross validation methodology to improve predictions in such environments by maximizing AUC of the products the customers will buy again in the future. Furthermore, using the same validation methodology it proposes a novel combination of a feature-driven, content-based approach and a collaborative filtering approach to improve results on top of these single methods involved.

The StackNet model: Ultimately this chapter re-implements Wolpert's stacked generalization and combines it with a feedforward neural network architecture in order to provide a scalable framework to combine multiple algorithms in order to achieve higher accuracy in classification tasks (including but not limited to the recommendation science). The methodology is also made available to the general scientific community in the form of a machine learning library implemented in the Java programming language, aiming to address the issues arising from the unavailability of certain algorithms for large scale problems. This is the first software application fully dedicated to meta modelling.

1.5 Structure of the thesis

Structure of this thesis

- **Chapter 2 Literature Review:** describes the literature pertinent to this research and reviews background information on a number of key concepts in the areas that this research spans.
- **Chapter 3 Univariate Analysis of the Dataset:** Chapter 3 scrutinizes the retailer's available dataset and provides explanatory insight in regards to the features that are going to be used later for the prediction algorithms. It addresses the non-linearity of certain features derived from customers' transactional history and proposes a binning methodology to aid capturing it.
- **Chapter 4 Meta-modelling to predict top K products:** Chapter 4 part investigates the improvement in prediction for forecasting the customers' top K products in their next visit to the retailer's store via combining an arsenal of different supervised machine learning algorithms. Furthermore it examines the improvement in prediction of the top K products for these customers using a meta-modelling approach versus all single models involved, simple ensemble models and field-related benchmarks.
- **Chapter 5 Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment:** This chapter utilizes the findings from the previous two chapters regarding feature engineering and model combination to improve predictions regarding which products the customers of a retailer will buy again in the future after having received an offer for them. This prediction is further enhanced by a cross validation methodology tested to yield better AUC results in an irregular future environment where the scoring population includes different customers and largely different offers than those used to create the models. It then proposes a hybrid model of a content-based approach along with a collaborative filtering approach to further improve results on top of any of these two single methods involved.
- **Chapter 6 The StackNet model:** Chapter 6 describes the StackNet model which constitutes a scalable implementation of Wolpert's stacked generalization within a feedforward neural network architecture with the aim of improving predictions in

classification (as well as recommendation) tasks. Subsequently many considerations regarding efficient fitting of this algorithm are explained and various modes and modelling characteristics are analyzed. The overall usability of the model is presented through its Java implementation which accompanies this research work. Different instances of StackNet models with multiple levels and architectures are then tested to rank the likelihood of a given song being created before or after 2002 using a set of 90 numerical attributes out of 515,345 songs that come from a subset of the Million Song Dataset [Bertin-Mahieux et al. 2011].

- **Chapter 7 Conclusion:** Chapter 7 provides an overall conclusion of this research with a summary of the key findings and their implications. The thesis ends with a number of recommendations per chapter in the form of future work that could be done in this area to further improve results.

2. Background

This chapter provides the literature review that underpins the topics analysed and evolved in this thesis. The chapter starts with a historical overview of the evolution of recommender systems and the machine learning applications within this field. It then goes on to offer a comparative overview of the different machine learning methods as well as an explanation of the common statistical measures utilized in the field.

2.1 Univariate Analysis of the Dataset

2.1.1 Brief Overview of recent recommender systems

The main aim of the thesis is to improve recommendations for customers in a retail environment - specifically grocery retail. Recommendation science in this context can be defined as the *principles, techniques and applications that facilitate the process of suggesting an item (product) to a customer* [Ricci 2001]. Recommendation science has received increased attention in recent years [Sarwar Badrul et al. 2001] and has been widely used by internet companies of all sizes (notably Facebook, Amazon and Google).

Since the onset of online retailing, the ability to recommend relevant products to customers has been a hugely important marketing tool for driving sales [Weng et al. 2004]. In today's era of Big Data [Chen et al. 2012], where there is an increased capacity to store and process large quantities of data, making recommendations has become a data-driven process [Linden et a. 2003]. The advances in algorithmic data processing and in machine learning have allowed frameworks to be developed [Gandomi et al. 2015] which improve predictions and consequently recommendations.

Corporations have invested heavily in unlocking the power of their data to successfully connect customers and products. [netflix 2009] paid $1,000,000 for the algorithm that could best predict the rating a customer would attribute to a given film. [Expedia 2013] did the same for optimizing hotel rankings to maximize customer click-through and purchase rates. [StumbleUpon 2013] tried to understand the elements that make a website relevant at a given

time. Indeed, data challenges have been used by many different companies to optimize recommendations for a diverse range of products/services including retail, music, art and geography and have taken many different forms such as image-based recommendations. There are now many organizations (such as dunnhumby) whose business offering is to provide such recommendation services. There are also numerous open source applications including [LibFM 2012], [LibFFM 2015], [GraphChi 2012] and [RankLib 2013] which are specifically dedicated to this field.

The data driven version of recommendation science has three main expressions. Content-based recommendation science can be defined as the process of selecting products because they adhere to a specific set of characteristics. For example a company that recommends art to customers would use such a method to classify a piece of art as modern so as to be able to recommend it to a customer who has previously purchased modern art. However, although this method is easy and quick to implement, its exploration of the relationships which connect customers and products is very superficial. Collaborative recommendations are the next step up - they look for deeper connections. Applied to the earlier art world example, a collaborative recommendation process would seek to understand similarities between customers as it assumes that customers who share similar characteristics are likely to have similar preferences. However, depending on the size of the database, it can be a very time-consuming process to calculate similarities across all customers in order to make the best recommendations. In practice hybrid recommendations which combine the benefits of the content-based and collaborative approaches are often used and can be effective [Adomavicius et al. 2005].

In the current thesis, recommendation science will be perceived as a more abstract machine learning field. Admittedly the relationship between customer and product is vividly complex and many successful unsupervised algorithms (such as Singular Value Decomposition [Golub 1970]) are commonly used to find and map these underlying complexities and generate features that explain them. [Pedregosa 2011] defined unsupervised learning methods as those in which the training data consists of a set of input vectors without any corresponding target values. The goal in such problems is to discover groups of similar vectors within the data. This particular approach is referred to as clustering. Another example is to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization.

2.1.2 Feature Types

There are four key elements that define purchase behavior. The first element is the number of times a customer has previously purchased the product. This is a critical feature in the majority of commercial recommendation engines. A customer's purchase history and their loyalty to a specific brand or product can greatly influence their propensity to buy an item. [Meyer-Waarden 2008] demonstrated that loyal customers (who visit more often) respond more positively (than non-loyal) to product recommendations given a certain number of factors. Similarly, based on data from various businesses, [Marcus in 1998] created a Customer Value Matrix which provides an approach to defining customer value using information such as frequency of purchase and purchase cycle.

The second key element is useful in introducing customers to new products. It is based on the idea that people sharing similar characteristics will like similar products. As [Ahn 2008] describes, the cold-start problem can be addressed by using features generated from customer-to-customer and product-to-product associations through approaches such as collaborative filtering. Customer segmentation itself has gained ground with the use of unsupervised machine learning techniques such as Principal Components [Pearson 1901] analysis, Singular Value Decomposition [Golub 1970] and other forms of information decomposition techniques as well as clustering techniques such as in K-means [MacQueen 1967] and Hierarchical [Ward 1963]. The resulting customer segments can then be leveraged later on in a modelling and prediction processes. The idea of using this kind of latent space to generate features has led to the development of customer-to-item generation methods using matrix factorization techniques [Koren et al. 2009]. These techniques have also been developed further to be utilized in supervised form, for regression or classification problems. A landmark in this kind of combination of supervised learning, using unsupervised features is LibFM [Rendle 2012]. More recent advancements in this space include the inclusion of deep learning and neural networks to create a similar latent space feature library that can be used to summarize customers and products based on input data. Restrictive Boltzaman Machines [Smolensky 1986] have also been used in dimensionality reduction and have also been applied in collaborative filtering with success [Salakhutdinov et al. 2007].

The third element is based on product attributes. Linden et al. [2001] highlights the efficacy of item-based elements such as item quality or item popularity in contributing to the accuracy

of product recommendation models. Product attributes such as price, recent sales, discounts as well as attributes relating to product categorization hierchy (food→ Pizza→ Pappa John's) can be useful additions in predicting a product's likelihood of being purchased.

The fourth element that drives purchase behavior is contextual information. [Setten et al. 2004] describe this as "any information that can be used to characterize the situation of an entity". Their demonstration of context-aware recommendations constitutes state-of-the-art recommendation science today. [Adomavicius et al. 2005] also emphasize the importance of contextual information such as time, temperature and location in models from many different disciplines including e-commerce personalization, information retrieval, ubiquitous and mobile computing, data mining, marketing, and management.

In summary, the key drivers of purchase behavior can be categorized into four key groups:

- Features that describe the customer
- Features that describe the item
- Features that describe the relationship of customer and item
- Contextual Features

All the aforementioned features will be examined in accordance with the propensity of the customer to buy an item in following week.

2.1.3 Binning of features

[Dougherty et al. 1995] defines the binning of features as the discretization of continuous variables, that is to say the method through which continuous variables are transformed into discrete counterparts. Such methods may be based on unsupervised of supervised algorithms.

Supervised binning methods take into account the information contained in a target variable to define the most optimal bins against various metrics. Examples include methods used in decision tree algorithms such as CHAID [Kass 1980], CART [Breiman 1984], ID3 [Quinlan 1986], C4.5 [Quinlan 1993] and J48 [Bouckaert 2010]. While decision trees can find the optimal cut-offs iteratively both in a univariate and multivariate context as part of their learning procedure, there are also methods dedicated solely to transforming variables optimally via binning (i.e. to be used for further modelling). Many of these methods fall into MDLP (Minimal

Description Length Principle) whereby a variable is split into a certain number of bins (hybrid method) or all distinct values are considered (standard method). Then all these bins are gradually being merged based on the impact they yield against a supervised metric such as Entropy (defined at 2.2.2). This merging of bins is repeated until an optimum number of bins is formed against some criteria that have to do with complexity and optimization [Xi 2006].

Unsupervised methods bin the variables based on underlying distributions. [Han et al. 2006] enumerate various unsupervised methods including histogram-based approachs where they detail two commonly applied variations: binning based on equal width and binning based on equal population (also referred to as equal frequency). The first method consists of binning a variable based on equal intervals (e.g. every 10 points) while the latter ensures that each bin contains an equal number of samples. Clustering is another frequently used method described by [Han et al. 2006] whereby bins (or clusters) are created based on the closeness of data points. Binning based on observation and intuition also fall into the category of unsupervised binning methods.

The binning of continuous variables has been largely used in credit scoring applications where variables need to be expressed as categories in order to create credit scorecards [Lucas 2001], [Hsieh et al. 2010], [Siami et al. 2013], [Zeng 2014]. However there have also been examples where binning of continuous variables has been employed in the collaborative filtering and recommendation space. [Hao et al.2016] use discretization techniques to transform input features for collaborative filtering models predicting the occurrence of certain pathological states such as sudden cardiac death and recurrent myocardial infraction. [Poirier et. al 2010] apply an optimal discretization method of numerical features to improve the predictive algorithm for recommending movies via exploiting blogs of textual data from the web.

2.2 Meta-modelling to predict top K products

The following sections give an overview of the common ensemble methods in predictive modelling, some of the typical metrics commonly selected to optimize these methods and a selected but representative sample of the supervised algorithms that are often used in machine learning.

2.2.1 Overview of ensemble methods

The idea of combining different machine learning or statistical methods (also known as *ensembling*) to reach a better solution is not new in data science. [Breiman 1996] demonstrated that *bagging* i.e. model averaging, commonly performs better than any single model. [Granger 1969] used a forecasting average mechanism to improve forecasts and achieve lower root mean squared errors in a model predicting how many customers would use an airline service. There are various methodologies for combining models. The most common methodologies are listed below:

2.2.1.1 Simple averaging

Simple averaging is the simplest form of ensembling . It assumes each model has an equal weight in the final model. In scientific notation it could be represented by equation 2.1:

$$\hat{Y} = G(X) = \frac{1}{L}\sum_{l=1}^{L} G_l(X) = \frac{1}{L}\sum_{l=1}^{L} \hat{y}_l \qquad (2.1)$$

,where $X \in \Re$ is a tabular dataset, G(X) is the function that maps X to a target variable Y ∈R, L is the number of estimators in the ensemble, and $\hat{y}_l$ is the prediction of each estimator and $\hat{Y}$ the final prediction of the ensemble [Ashtawy et al. 2015] .

2.2.1.2 Bagging

This method is very similar to simple averaging. The difference is that each model is built on a bootstrapped set that consists of samples extracted via replacement from the main dataset. According to [Kuncheva et al. 2003], if the single models can yield diversity, that is to say bring in new information, it can benefit the overall ensemble model. Bootstrapping allows the models to become slightly different (as they are trained with different subsets of the data)

thereby increasing the diversity of information they bring to the whole. In scientific notation it may take the form of equation 2.2:

$$\hat{Y} = G(X^P) = \frac{1}{L}\sum_{l=1}^{L} G_l(X^{p_l}) = \frac{1}{L}\sum_{l=1}^{L} \hat{y}_l, \quad (2.2)$$

where $X^P \in \Re^P$ is a tabular dataset with sample size P, $G(X^P)$ is the function that maps X to a target variable $\mathbf{Y} \in R$. X^{Pl} is a tabular dataset with the sample size of P, but generated using bootstrapping via randomly selecting samples from the original X^P. L is the number of estimators in the ensemble, and $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ the prediction of each estimator and $\hat{\boldsymbol{Y}}$ the final prediction of the ensemble [Ashtawy et al. 2015].

In the context of this thesis, the term bagging will include other forms of randomized averaging, specifically *Pasting* for when random subsets of the dataset are drawn as random subsets of the samples [Breiman 1999], *Random Subspaces* [Ho 1998] for when random subsets of the dataset are drawn as random subsets of the features and *Random Patches* [Louppe et al. 2012] when base estimators are built on subsets of both samples and features.

2.2.1.3 Boosting

In *boosting* each model is added sequentially to the ensemble in order to improve overall performance. [Kearns 1988] was the first to propose a sequential approach and it can be very effective in combining weak learners into a powerful ensemble. A notable advantage of this method is that the weight of each model is adjusted and focused on the errors of the previous model(s), therefore making it easier to focus on the less explored or more difficult areas of the data space. On the other hand these methods tend to lead too overfitting. In employing boosting techniques, it is therefore important to penalize predictions via a *shrinkage* parameter or *learning rate* (or *eta*) to prevent overfitting from occuring.

There are various methods for boosting models, the most well-known of which are *Adaboost*, *Logitboost* and *Gradient Boosting* (or MART). Since the latter is most commonly used, it will also be considered by the current thesis.

The boosting may take the equation:

$$\hat{Y} = G(X) = \sum_{l=1}^{L} \gamma_l G_l(X) = \sum_{l=1}^{L} \gamma_l \hat{y}_l \quad (2.3)$$

$X \in \Re$ is the tabular dataset, G(X) is the function (which commonly takes the form of a decision tree) that maps X to a target variable Y ∈ R, L is the number of estimators in the ensemble, $\hat{y}_l$ is the prediction of each estimator in the ensemble and $\hat{Y}$ is the final prediction of the ensemble. γ_l is the shrinkage applied to each estimator. It is common that the shrinkage is constant irrespective of the estimator [Ashtawy et al. 2015].

The gradient boosting model uses the negative gradient of a differentiable loss function to update each model. This update can take the form of equation 2.4:

$$G_l(X) = G_{l-1}(X) + \gamma_l \hat{y}_l \quad (2.4)$$

In other words each estimator can be summarized as the weighted (by learning rate) sum of predictions of the preceding l-1 estimators plus the prediction $\hat{y}_l$ of the estimator l_{th} that is trained on the residuals of the $G_{l-1}(X)$ estimator with the target variable Y. In this thesis, Gradient Boosting will be used with decision trees as base learners.

2.2.1.4 Meta-model weight computation with cross-validation

Another way to combine models is by creating another model (commonly referred to as a *meta-model*) that takes as inputs the outputs of other models. For example, [Jin et al. 2009] used a generalized linear model for binary outcomes to combine different predictors for estimating the probability of a subject having a certain disease. The improvement in performance of the predictions of these models was measured using the Area Under the ROC Curve (AUC).

[Smyth and Wolpert 1999] demonstrated the sensitivity of a stacking approach to over-fitting given the involved models' complexity. A stacking model (as almost any other machine learning model) naturally over performs in the data it has been created with, causing it to lose

some of its ability to generalize in previously unseen data. The high complexity of a stacking model that naturally involves multiple algorithms at its base, each one likely to have different considerations and modelling assumptions, massively increases the possibility of over fitting. For that reason when utilizing a stacking model it is vital to do so by creating unbiased prediction for previously unseen or validation data. More commonly this is achieved via a K-fold cross validation where each model is trained on a subset of the initial data and predictions are formed for the other subset. The predictions as well as the real target values for that subset are saved for further modelling. As [Kohavi 1995] stated, this procedure reduces the variance of the final estimation although it increases the bias. According to the same paper bootstrapping each fold can reduce the variance even further.

Assuming that all different models applied to the same set have been cross-validated in exactly the same way and all predictions are saved for the same folds on the data, then these predictions can form a new set where the new target Y will be the concatenation of all validation targets of the k-folds validation sets and the new covariate matrix X will be consisted by J models where each j is a different model applied to the same cross-validation procedure. This process can be better explained via pseudocode.

The Meta modelling with K-Folds Paradigm takes the following parameters:

- SplitPercent: The percentage of the initial set to be used for validation at each k fold of cross-validation, for example 30% (and 70% for the training set)
- K: The number of cross validations to run (for example 10).
- x_0: The initial set of features to use to train each different model (classifier in this case, but it could be a regressor for a regression problem).
- y: The target or label variable , that takes values of 1 or 0.
- n: The number of training points (e.g. the rows of the dataset)
- C: The number of different classifiers within the ensemble.
- $\hat{y}$: Is 2dimensional vector of predictions with sample size equal to the rows of the k_{th} validation dataset and dimensionality equal to the number of classifiers C.
- x_1: The new set of features where each column denotes the concatenated predictions of a chosen classifier to each of the K validation sets. Its columns will be C, as many as the chosen classifiers.
- y_1: Is the concatenated subsets of the target variable y of all K validation datasets.
- G: The final Meta model to be trained with x_1 as feature set and y_1 the label.

The algorithm can be further portrayed via pseudo code by figure 2.1 as:

1. For *k*=1 to *k*=*K*, the initial dataset $\{x, y\}$ with rows n get split by *SplitPercent* to form a new train set :

   $$\text{train}_k = \{x_k, y_k\} \text{ with sample size } (1 - \text{SplitPercent x n})$$

   and validation set

   $$\text{validation}_k = \{x_m, y_m\} \text{ with sample size } (\text{SplitPercent x n})$$

   a. For *c*=1 to c=*C*, a classifier is trained on the train_k set and predictions $\hat{y}_m$ are made for the $\{x_m, y_m\}$ validation_k set

      i. Predictions get concatenated horizontally: $\hat{y} \rightarrow [\hat{y} \sim \hat{y}_m]$

   b. $\hat{y}$ is concatenated vertically: $x_1 \rightarrow [x_1 | \hat{y}]$

   c. y_m is also concatenated vertically: $y_1 \rightarrow [y_1 | y_m]$

2. The Meta model G is now fitted on the $\{x_1, y_1\}$

Figure 2.1 : Meta modelling paradigm with K-Fold cross validation

The general process of using models' predictions on some validation data as inputs to other meta-models was first introduced by [Wolpert et al. 1992] and they gave it the name of *stacked generalization* or *stacking*, where various neural networks with different structures were combined to achieve a better generalization error in a prediction task.

In the current thesis, *Stacking* will be used along with bagging and with random decision trees to achieve a better generalization error.

2.2.2 The Metrics

A common objective in recommendation science is to improve the classification accuracy of future purchase predictions (e.g. what the customer will buy in their next visit to the retailer). There are various metrics that can be used to judge the efficacy of the different modelling techniques or methods that set out to achieve this objective. It is only through using a

combination of these metrics that we can compare the merits of the different options. The current thesis will focus on the most widely used and representative families of these metrics.

2.2.2.1 Classification Accuracy

Classification accuracy, represented by formula 2.5, is probably the most common measure in classification tasks and is computed for a given cut-off probability (normally 0.5 or 50%) using the elements of the *confusion matrix* [Pearson 1904] as:

Table 2-1 : Confusion matrix and its elements

	It is the category	**It is NOT the category**
Predicted the category	True Positives (TP)	False Positives (FP)
Predicted NOT the category	False Negatives (FN)	True Negatives (TN)

Based on the elements of the confusion matrix, the classification accuracy can be denoted as:

$$\text{Classification accuracy} = \frac{\text{TP+ TN}}{\text{TP+TN+FP+FN}} \qquad (2.5)$$

A pitfall of this metric is that it does not question the ranking of the predicted score in respect to the target variable and is only focused on whether the classification is correct at a given cut-off point.

2.2.2.2 Precision@k

This metric is perhaps more suitable for recommendation science as it describes how predictive a method is at any given point in the recommendation list (where products are ranked by relevance score) [Powers 2011] – although optimization efforts tend to be focused on the results at the top of the list. The following equation displays the measure of precision for a specific k, where k refers to the product's position in the ranked list.

$$\text{Precision@}_k = \frac{TP_k}{TP_k + FP_k} \quad (2.6)$$

2.2.2.3 AUC (Area Under Curve)

The ROC (Receiver Operator Characteristics) curve was first introduced by [Green & Swets 1966]. It describes the confusion matrix of sensitivity (represented by formula 2.7) and 1-specificity (represented by formula 2.8) for each possible cut-off of the prediction's array.

$$\text{Sensitivity} = \frac{TP}{TP+FN} \quad (2.7)$$

$$\text{Specificity} = \frac{TN}{TN+FP} \quad (2.8)$$

In recommendation science, sensitivity is the percentage of customers who buy the offered (or recommended) product who were predicted to buy it and specificity is the percentage of customers who do not buy the offered product who were predicted not to buy it. An explanatory graph that breaks down the AUC to its basic elements is illustrated in figure 2.2:

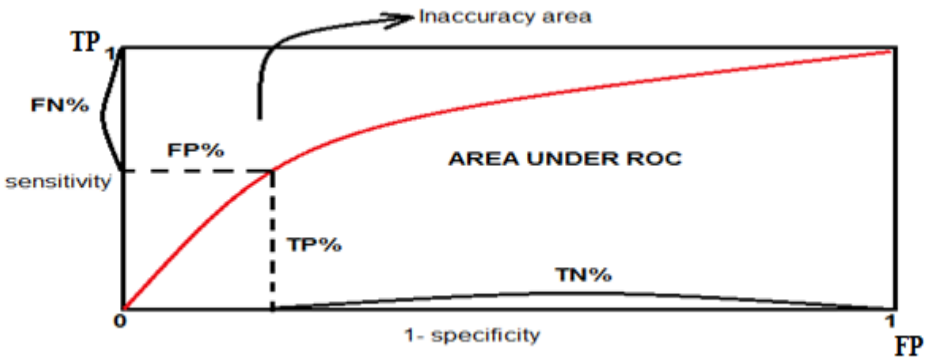

Figure 2.2: Roc Curve and AUC (Area Under the Curve)

In other words it reflects how the prediction's accuracy changes for all possible cut-offs.

More specifically the AUC formula 2.9 can be written as:

$$\mathrm{AUC}(X,Y) = \sum_{i=1}^{n+}\sum_{j=1}^{n-}\frac{L\left[f(x_i^+)>f\left(x_j^-\right)\right]+\frac{1}{2}L\left[f(x_i^+)=f\left(x_j^-\right)\right]}{2n_+n_-} \quad (2.9)$$

, where X is a feature with real values and Y another feature with 2 possible labels, one deemed as positive and one negative (commonly -1 for the negative and +1 for the positive or 0 for the negative and 1 for the positive). A sample x_n extracted from X is often a value in the range (0, 1) and expresses the probability of that n_{th} sample to belong to the positive class of Y. The Y is not included in the formula 2.9, but it has been used to determine which samples (out of n^+) belong to the positive class and which samples (out of n^-) belong to the negative class. A sample retrieved from X which belongs to the positive class can be referred to as x_i and from the negative class as x_j. L is the function which returns 1 if the argument contained in the brackets is true and 0 otherwise. For a perfect AUC score all positive observations need to have a higher score then the negative observations ($\left[\mathbf{f}(\mathbf{x_i^+}) > \mathbf{f}\left(\mathbf{x_j^-}\right)\right]$).

2.2.2.4 Pearson Correlation

Pearson's correlation coefficient, often denoted as r (or R) is a form of a bivariate linear correlation. It was introduced by Karl Pearson in the 1880s [Mukaka 2012]. The formula to obtain the coefficient for two vectors X, Y, where x_i,y_i are single samples from X,Y, is:

$$r(X,Y) = \frac{\sum_{i=1}^{n}(x_i-\overline{X})(y_i-\overline{Y})}{\sqrt{[\sum_{i=1}^{n}(x_i-\overline{X})^2][\sum_{i=1}^{n}(y_i-\overline{Y})^2]}} \quad (2.10)$$

where $\overline{X}$ is the mean for vector $X\in\Re$ and $\overline{Y}$ the mean of vector $Y\in\Re$.

2.2.3 The Algorithms

One of the main objectives of this thesis is to exploit state-of-the-art machine learning algorithms to better optimize the given metrics via ensemble methods. These algorithms can be either of a supervised or unsupervised nature. This chapter will focus mostly on the implementation details of the algorithms. Many of the algorithms have a number of hyper parameters associated with them and quite often finding the right values for these parameters is important in obtaining good estimates. The hyper parameters for each algorithm are listed in 6.3.5, under the experiment using the StackNet model.

2.2.3.1 Linear Regression

Linear or Ordinary Least Squares regression (OLS) is one of the most widely used statistical methods and consequently machine learning algorithms that attempts to linearly combine various inputs by means of finding the optimum coefficients which minimize the squared error in respect to a dependent variable $Y \in \Re$ [Craven et al. 2011]. In simple terms it minimizes equation 2.11:

$$E(W) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{N}(y_i - W^T x_i)^2 \qquad (2.11)$$

where W is the vector of coefficients $\in \Re$ and has the same size as the number of features in the dataset $X \in \Re$ with sample size N plus one more feature if a constant value is included. $\hat{y}_i$ is the predicted value for a given sample i derived from the multiplication of the transposed vector of coefficients $W^T$with the feature vector x_i of a single sample i. Very commonly OLS regression is used with a modification to account for multicollinearity [Wold 1984] in the data that can heavily bias predictions, namely regularization (often denoted as c, C or λ). The latter can be seen as a form of penalty that is applied to the coefficients in order to halt their values from growing uncontrollably. The most prominent form of regularization is the L_2 applied to the coefficients. The previous equation can now be written as 2.12:

$$\mathrm{E(W)} = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{N}}(\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2 + \frac{1}{2}\lambda W^T \mathrm{W} = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{N}}(\mathrm{y_i} - \mathrm{W^T x_i})^2 + \frac{1}{2}\lambda W^T \mathrm{W} \quad (2.12)$$

This is also called *Ridge Regression* [Tikhonov, 1946]. In matrix form the solution can be obtained via equation 2.13:

$$\widehat{\mathrm{W}} = \underset{\mathrm{W}}{\mathrm{argmin}}\, \mathrm{E(w)} = (\mathrm{X^T X} + \lambda \mathrm{I})^{-1}\mathrm{X^T Y} \quad (2.13)$$

where $\mathrm{X^T X}$ is the covariance matrix of the given features, a scalar λ for the regularization term and I an identity matrix where all the values of the diagonal have the value of 1. The $\widehat{\mathrm{W}}$ indicates the OLS estimate of W.

Equation 2.13 demonstrates that in order to find the optimal coefficients W, it is required to compute the inverse of the $\mathrm{X^T X} + \lambda \mathrm{I}$ matrix, which can be can be expensive if data dimensionality is large. In order to avoid this operation, there are methods that use iterative minimization of the loss function E. One family of these methods is the *gradient descent (GD)* [Bottou 2010].

To solve the W using this method, the vector W is initialized with some values w^0. Then (and until convergence) the E gets optimized iteratively. W gets updated by moving along the direction of the negative gradient $-\frac{\partial \mathrm{E}}{\partial \mathrm{W}}$ as shown in equations 2.14 and 2.15:

$$\mathrm{W} = \mathrm{W} - \mathrm{a}\,\frac{\partial \mathrm{E}}{\partial \mathrm{W}} \quad (2.14)$$

or in matrix notation:

$$\mathrm{W} = \mathrm{W} - \alpha\,\mathrm{X^T(XW} - \mathrm{Y}) \quad (2.15)$$

where a is the *learning rate* or the step by which the weights W are updated iteratively. The value a is typically found experimentally during a cross validation procedure.

Stochastic Gradient Descent (SGD) is another form of a gradient optimization method of a differentiable function that uses the gradient of that function to reach its minimum point. The term *stochastic* refers to the fact that the path to the minimum point can be achieved incrementally without requiring to parse the whole dataset at once, but instead sample by sample. For this specific reason SGD is commonly associated with online learning [Ma et al. 2009] for its ability to update the current parameter values as soon as the respective labels are known, using the gradient of the function. In this case the update of W can occur using one sample point x_i as illustrated in equation 2.16:

$$W = W - \alpha\, x_i\left(W^T x_i - y_i\right) \tag{2.16}$$

2.2.3.2 Logistic Regression

Logistic Regression (LR) is a modification of the initial OLS problem where the output score can be expressed as a probability for a given label Y to belong to a class of 1 or -1. In other words the interest is in predicting the label probabilities P(Y | X, W), given a feature vector X and some coefficients W. The probability that the label is 1, using the Logistic Regression model is derived using equation 2.17:

$$P(Y = 1 \mid X, W) = \sigma(W^T X) = \frac{1}{1 + e^{-W^T X}} \tag{2.17}$$

where σ is the logistic (or sigmoid) function which maps all real number into (0, 1) [Li et al. 2016]. The function to minimize is the log likelihood, denoted as logL:

$$\mathrm{LogL}(W) = \log P(Y \mid X, W) = \sum_{i=1}^{N} \log P(y_i \mid x_i, W) = \sum_{i=1}^{N} -\log\left(1 + e^{-y_i W^T x_i}\right) \tag{2.18}$$

Where log is the natural logarithm (or the logarithm to the base of the mathematical constant e), e is the Euler's number and N is the sample size of X. Adding the λ regularization term, the coefficients W can be derived via minimizing the LogL [Minka 2003]:

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\ \mathrm{LogL(W)} = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{N} -\log\left(1 + e^{-y_i W^T x_i}\right) + \frac{1}{2}\lambda W^T W \qquad (2.19)$$

There is no closed-form solution to solve 2.19 but it can be solved iteratively using Gradient Decent on W. In that case, the gradient (∇) of W (excluding regularization) in respect to the LogL can be computed with 2.20:

$$\nabla_W \mathrm{LogL(W)} = \sum_{i=1}^{N} \frac{x_i y_i}{1 + e^{y_i W^T x_i}} \qquad (2.20)$$

2.2.3.3 Linear Support Vector Machine

Linear SVM is a scalable and easy to implement model that linearly combines various features to achieve the best linear separation of two classes, normally 1 and -1. While logistic regression focuses on giving an estimate of probability to an event, SVM is more focused on getting the classification correct [Rosasco et al. 2004]. The most common loss associated with this kind of linear separation is the Hinge loss, denoted as HingeL. Given a feature set $X \in \Re$ and corresponding label $Y \in \Re$, where each $y_i \in \{-1,1\}$, the loss function can be computed as:

$$\mathrm{HingeL(W)} = \mathrm{HingeL(Y, X, W)} = \sum_{i=1}^{N} \max\{0, 1 - y_i W^t x_i\} \qquad (2.21)$$

where W is the vector of coefficients. Their optimum values (that minimize the Hinge loss) can be obtained via 2.22 assuming that there is a λ penalty:

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\ \text{HingeL(W)} = \underset{W}{\operatorname{argmin}}\ \sum_{i=1}^{N} \max\{0,1 - y_i W^t x_i\} + \frac{1}{2}\lambda W^T W \tag{2.22}$$

The optimal W can be found using gradient and sub gradient methods on W. The gradient (∇) of W (excluding regularization) in respect to the HingeL (excluding regularization) can be computed with 2.23 [Collobert et al. 2001]:

$$\nabla_W \text{HingeL(W)} = \sum_{i=1}^{N} \begin{cases} -y_i x_i\ , \text{if}\ y_i W^t x_i < 1 \\ 0,\ \text{if}\ y_i W^t x_i \geq 1 \end{cases} \tag{2.23}$$

2.2.3.4 Multilayer perceptron and neural networks

Moving away from the linear models, neural networks have been used extensively in machine learning applications and in various fields, including recommendation science for many years [Christakou et al. 2007]. Neural Networks or just NNs may take various shapes and create complicated structures using various functions for input or output. For the purposes of this thesis only the multilayer perceptron neural network type will be examined and specifically one of its most simple forms to ensure scalability. A typical multilayer perceptron with 1 hidden layer and 5 hidden neurons can be viewed in figure 2.3.

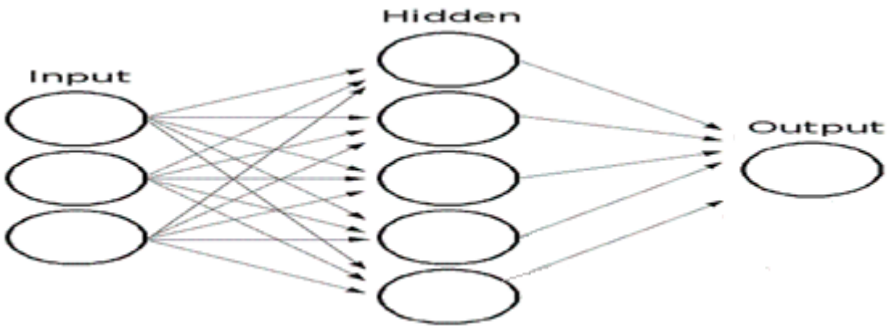

Figure 2.3: Single layer neural network[2]

As illustrated in 2.3, given some input features $X \in \Re$ with dimensionality J (X^J), a transformation takes place within the hidden layer by summing up all the dot products of each sample x_i with some W_j vectors with size equal to the dimensionality (J) of the input data. This is repeated for the output layer using the hidden layer as input and a new set of W_h with dimensionality H, equal to the number of neurons of the hidden layer. The output of this network can be used to optimize both the squared loss function presented in 2.11 for regression tasks and the LogLikelihood function 2.18 for classification tasks problems. In the context of the squared loss function, the latter can be expressed in 2.24 generically given some estimates $\widehat{Y} \in \Re$, which are the result of the output of the network and a target variable $Y \in \Re$.

$$E(\widehat{Y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.24}$$

The output of the network can be expressed as a function f which takes as input the feature set X^J and given a 2-dimensional vector of W with size J,H, it outputs estimates $\widehat{Y}$ given equation 2.25 [Ashtawy et al. 2015] :

$$\widehat{Y} = f(X^J, W^{J,H}) = G\left(\sum_{h=0}^{H}\left(w_{h,g}\sigma\left(\sum_{J=0}^{J}(w_{j,h}x_j)\right)\right)\right) \tag{2.25}$$

[2] obtained (and edited) from http://docs.opencv.org/modules/ml/doc/neural_networks.html

Where G is a linear activation function of the output neuron as G (u) = u, σ is the activation function for the hidden-layer neurons which was previously defined in 2.17 in the context of Logistic Regression. $W_{h,g}$ refers to the weights associated with the links connecting the hidden units to the output layer, $W_{j,h}$ represents the weights of input-to-hidden layer links, and x_j is the j_{th} feature of X. The weight variables $W_{0,h}$ serve as bias parameters.

Other common activation functions (apart from the sigmoid σ) are the hyperbolic tangent, denoted as tanh displayed in 2.26 and the rectifier denoted as relu [LeCun et al.2015], displayed in 2.27:

$$\tanh(u) = \frac{e^{u} + e^{-u}}{e^{u} - e^{-u}} \tag{2.26}$$

$$\mathrm{relu}(u) = \max(0, u) \tag{2.27}$$

where u is the input to a neuron. The minimization problem for the squared loss function can be expressed with 2.28:

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\, E(W) = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - G\left(\sum_{h=0}^{H} \left(w_{h,g}\sigma\left(\sum_{J=0}^{J} (w_{j,h}x_{i,j}) \right) \right) \right) \right)^2 \tag{2.28}$$

The most common way to minimize function 2.28 is by using Back Propagation (BP) [Rojas 1996] along with Gradient Descent. The main concept of BP is that starting from the output and moving backwards (towards input), the emissions of the neurons' derivatives (gradients) carry the details of the residual error with the target variable Y and formulate the updates for all weights accordingly.

The weights in a generic network with number of layers L (including input and output) are the only parameters that can be modified to make the quadratic error E as low as possible. Because E is calculated by the extended network exclusively through composition of the node functions, it is a continuous and differentiable function of the weights W_1, W_2, . . . ,W_m in the network

[Rojas 1996], where m is the size of the layers L - 1. E can be minimized by using an iterative process of gradient descent, for which the gradient is calculated as:

$$\nabla_{\mathrm{E}} = \left(\frac{\partial \mathrm{E}}{\partial W_1}, \frac{\partial \mathrm{E}}{\partial W_2}, \ldots., \frac{\partial \mathrm{E}}{\partial W_m}\right) \tag{2.29}$$

Each W_m is updated using the increment:

$$\mathrm{W}_m = \mathrm{W}_m - a\frac{\partial \mathrm{E}}{\partial W_m} \tag{2.30}$$

, where a represents a learning rate, which defines the step length of each iteration in the negative gradient direction [Rojas 1996].

2.2.3.5 Naïve Bayes Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable Y and a dependent feature vector X with size J, Bayes' theorem states the following relationship [Pedregosa et al. 2011]:

$$\mathrm{P}(\mathrm{Y}\,|x_1, x_2, \ldots, x_J) = \frac{\mathrm{P}(Y)\mathrm{P}(x_1, x_2, \ldots, x_J|\mathrm{Y})}{\mathrm{P}(x_1, x_2, \ldots, x_J)} \tag{2.31}$$

Using the naive rule that all features j are independent given the value of the class variable [Zhang 2004]:

$$\mathrm{P}(\mathrm{Y}\,|x_1, x_2, \ldots, x_J) = \frac{\mathrm{P}(Y)\ \prod_{j=1}^{J}\mathrm{P}(x_j|Y)}{\mathrm{P}(x_1, x_2, \ldots, x_J)} \tag{2.32}$$

Since $\mathrm{P}(x_1, x_2, \ldots, x_J)$ is constant given the input, equation 2.32 can be further simplified as 2.33:

$$\mathrm{P}(\mathrm{Y} \mid x_1, x_2, \ldots, x_J) \approx \mathrm{P}(Y) \prod_{j=1}^{J} \mathrm{P}(x_j|Y) \tag{2.33}$$

The estimate of Y, denoted as $\hat{\mathrm{Y}}$ can be calculated using the Maximum A Posteriori (MAP) method (from Bayesian statistics) to estimate $\mathrm{P}(Y)$ and $\mathrm{P}(x_j|Y)$ [Gauvain 1994].

$$\hat{Y} = \underset{Y}{\operatorname{argmax}}\, \mathrm{P}(Y) \prod_{j=1}^{J} \mathrm{P}(x_j|Y) \tag{2.34}$$

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $\mathrm{P}(x_j|Y)$. A widely used version of Naïve Bayes assumes a Gaussian distribution of continuous features X to belong in a class c of the label variable Y:

$$\mathrm{P}(x_j|Y) = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\boldsymbol{\pi}\boldsymbol{\sigma}_{\boldsymbol{Y}}^{\mathbf{2}}}} \exp\left(-\frac{(x_j-\mu_Y)^2}{2\sigma_Y^2}\right) \tag{2.35}$$

where parameters σ_Y (variance) and μ_Y (mean) are estimated using maximum likelihood [Hand et al. K 2001].

2.2.3.6 K Nearest Neighbours (KNN)

KNN is another supervised machine learning algorithm (for regression and classification) that is also very commonly used, particularly because it is easy to implement. It has been used in the recommendation space but has been criticized for being slow compared to other techniques.

It also requires a significant amount of memory to store the main dataset from which neighbours are discovered [Ravi et al. 2009].

The main principle behind KNN is that for each observation i to be classified from a feature set P with dimensionality J, the K closest observations are discovered based on a distance metric D from another dataset X (with the same dimensionality) where the label Y is known. The final predictions are formulated based on a predefined number of training samples (K) with closest distance D to the x_i point, and the estimate $\hat{Y}$ is made based on the majority of the class label contained in these K closest observations.

There are various ways to calculate the closeness of a data observation with another. One of the most commonly used measures is Euclidian distance. Due to its popularity Euclidian distance will be used as the main distance measure for this thesis. For two 2 observations x and p retrieved from the main feature sets X,P respectively, the Euclidian distance [Weinberger et al. 2005] with dimensionality J can be estimated as:

$$\text{Euclidian Distance}(x, p) = \sqrt{\sum_{j=1}^{J}(x_j - p_j)^2} \quad (2.36)$$

Where x_j is the value of feature j in the data point x, retrieved from feature set X and p_j the value of feature j in the data point p, retrieved from feature set P.

2.2.3.7 Decision Trees, Random Forests and Gradient Boosting Trees

Decision Trees are non-parametric algorithms used for regression or classification. Through sequential partitioning of a given feature (where the partitioning process continues until a specific goal or stopping criteria are met), they attempt to achieve more discriminating results in regards to the target variable.

The common training process for a Decision Tree algorithm can be described as follows: for a feature set X with dimensionality J, all x_1, $x_{2,\ldots,XJ}$ features are selected one by one and frequencies of the unique classes c of the target variable Y are estimated for all possible distinct values D_j of the x_j vector. Out of all possible distinct values D from all feature vectors in X, the best distinct value (D_j) for a vector X_j is determined using a metric that quantifies the amount

of information explained in respect to the target variable Y, if this value is used to partition X. Assuming the best point occurs for distinct value s in feature j, two different feature sets $\{X_1,Y_1\},\{X_2,Y_2\}$ will be created out of the initial X as displayed in equation 2.37:

$$\begin{cases} X_j \leq D_{j,s} \ \rightarrow \{\mathrm{X}_1, \mathrm{Y}_1\} \\ X_j > D_{j,s} \ \rightarrow \{\mathrm{X}_2, \mathrm{Y}_2\} \end{cases} \tag{2.37}$$

where $\mathrm{X}_1, \mathrm{X}_2$are subsets of X and $\mathrm{Y}_1, \mathrm{Y}_2$ subsets of Y. Two separate decision trees are then drawn based on the new sets and this process is repeated until certain criteria are met like the maximum depth of the tree or the total size of nodes in the tree. There are many metrics that can define the optimal split and many ways to split a tree into subsets, but in this thesis binary splits will be considered as already displayed.

A common splitting criterion for a classification task is using *Entropy* to find the best partition. Entropy measures the amount of information contained within a certain dichotomization of the data in respect to the proportion of the classes of label Y [Thomas et al. 2002]. Given C the number of distinct classes in Y, the Entropy (Denoted as En) for all distinct classes of Y can be estimated with 2.38:

$$\mathrm{En}(Y) = \sum_{\mathrm{c}=1}^{\mathrm{C}} -\mathrm{P}(\mathrm{Y} = \mathrm{c}) \log_2 \mathrm{P}(\mathrm{Y} = \mathrm{c}) \tag{2.38}$$

To better illustrate this, consider the following example of two classes. Table 2-2 presents the frequencies of the binary classes of Y:

Table 2-2: Frequency of the distinct classes of Y

Y	
c=0	**c=1**
15	15

Based on table 2-2, $\mathrm{P}(\mathrm{Y} = 0) = \frac{15}{30} = 0.5$ and $\mathrm{P}(\mathrm{Y} = 1) = 0.5$. Therefore:

$$\mathrm{En}(Y) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1.0$$

Considering a potential split point $D_{j,s} = 50$, table 2-2 can be expanded to 2-3 in order to include the frequency matrix of the class labels of Y and the two directions ($\leq$ and $>$) of $D_{j,s}$.

Table 2-3: Frequency matrix for the class labels of Y and a 2-way directions of split point $D_{j,s}$

		Y		
		c=0	**c=1**	**Sum**
$D_{j,s}$	**<= 50**	10	7	17
	> 50	5	8	13
	Sum	15	15	30

Now Entropy can be computed based on the actual frequencies of the classes c and the two directions of the split point $\mathbf{D_{j,s}}$ based on formula 2.39:

$$\mathrm{En}\left(\mathrm{Y}, \mathrm{D_{j,s}}\right) = \mathrm{P}\left(\mathrm{D_{j,s}} = '\leq 50'\right) \mathrm{En}\left(Y_{\mathrm{D_{j,s}}='\leq 50'}\right) + \mathrm{P}\left(\mathrm{D_{j,s}} = '> 50'\right) \mathrm{En}\left(Y_{\mathrm{D_{j,s}}='>50'}\right) \quad (2.39)$$

Replacing all elements based on the frequency table 2-3:

$\mathrm{P}\left(\mathrm{D_{j,s}} = '\leq 50'\right) = \frac{17}{30} \approx 0.567$,

$\mathrm{P}\left(\mathrm{D_{j,s}} = '> 50'\right) = \frac{13}{30} \approx 0.433$,

$\mathrm{En}\left(Y_{\mathrm{D_{j,s}}='\leq 50'}\right) = -\frac{10}{17}\log_2\frac{10}{17} - \frac{7}{17}\log_2\frac{7}{17} \approx 0.977,$

$\mathrm{En}\left(Y_{\mathrm{D_{j,s}}='>50'}\right) = -\frac{5}{13}\log_2\frac{5}{13} - \frac{8}{13}\log_2\frac{8}{13} \approx 0.9612,$

$\mathrm{En}\left(\mathrm{Y}, \mathrm{D_{j,s}}\right) = (0.567 \times 0.977) + (0.433 \times 0.9612) \approx 0.9704$

The final step to determining the value of the split is to measure how much information was gained before considering the $D_{j,s}$ split point and after. This *information gain* is often denoted as *IGain* (formula 2.40) and measures the difference between initial entropy (prior to splitting) and after the split:

$$\mathrm{IGain}\left(\mathrm{Y}, \mathrm{D}_{\mathrm{j,s}}\right) = \mathrm{En}(\mathrm{Y}) - \mathrm{En}\left(\mathrm{Y}, \mathrm{D}_{\mathrm{j,s}}\right) \tag{2.40}$$

In the aforementioned example $\mathrm{IGain}\left(\mathrm{Y}, \mathrm{D}_{\mathrm{j,s}}\right) = 1.0 - 0.9704 = 0.0296.$

Random Forest will be implemented through bagging (as explained in (2.2) by averaging many different randomized trees where the random factor will also be imputed by tuning various tree-specific parameters such as maximum number features to be considered for a split, maximum number of possible cut-offs for a given feature as well as other hyper parameters such as minimum number of samples in a single node, maximum tree size (in levels) and number of trees [Breiman, 2001].

Gradient Boosting Trees, or MARTs (multivariate additive regression trees) will have the form of (2.3) where Decision Trees are the base learners.

2.2.3.8 Matrix Factorization and LibFM

Non-Negative Matrix Factorization (or for simplicity NNMF) in the recommendation world is a way to summarize a (normally sparse) matrix of item-to-customer interactions. It usually consists of a U vector of size f for the customers and a V vector of size f for the items where the prediction $\hat{Y}_{ij}$ of a customer i (out of n) to buy product j (out of m) is the dot product of the two vectors as illustrated in 2.41:

$$\hat{Y}_{ij} = U_i V_j \tag{2.41}$$

The size f is a hyper parameter and is often referred to as the *latent feature* for these vectors. Assuming the loss to be minimized is the squared error E, the U,V can be obtained by equation 2.42:

$$\hat{U},\hat{V}=\underset{U,V}{\operatorname{argmin}}\, E(U,V) = \underset{U,V}{\operatorname{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{m}\left(Y_{ij} - U_i V_j\right)^2 \tag{2.42}$$

Whilst NNMF has been commonly used to capture pairwise interactions between customers and items, *LibFM* [Rendle 2012] combines linear models (such as linear regression) with factorized pairwise interactions to provide more holistic models. Assuming a squared loss function E, a target variable Y feature set X with dimensionality m and a matrix with latent features $\boldsymbol{U}$ of size m x f, the optimization function can be summarised as equation 2.43:

$$\hat{W},\hat{U}=\underset{\hat{W},\hat{U}}{\operatorname{argmin}}\, E(W,U) = \left(Y - \left(X_0 + X_1 w_1 \ldots + X_m w_m + \sum_{j=1}^{m}\sum_{d=j+1}^{m} X_j X_d U_j U_d\right)\right)^2 \tag{2.43}$$

The scoring function consists of 2 parts, the linear model and the dot product of all possible pairwise interactions within a given sample. The prediction function f from 2.43, is fully elaborated in equation 2.44:

$$f(W,U) = \left(X_0 + X_1 w_1 \ldots + X_m w_m + \sum_{j=1}^{m}\sum_{d=j+1}^{m} X_j X_d U_j U_d\right) \tag{2.44}$$

LibFM can also be solved with gradient methods where the linear and pairwise model are split into two different updates. The linear update is described in equation 2.45:

$$\nabla_W E(W) = (f(W,U) - Y)X \tag{2.45}$$

The latent features' vectors update occurs when creating the 2-way interactions and takes the form of equation 2.46 [Rendle et al. 2011]:

$$\nabla_U E(U) = (f(W,U) - Y)X \sum_{l=1}^{m} U_l X_l \tag{2.46}$$

2.3 Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment

Retailers have been using discount or promotional coupons for years as a way of driving customer loyalty. The introduction of loyalty card data has enabled them to make the process of coupon allocation more efficient. Information derived from purchase history has allowed retailers to allocate coupons based on individual customer preferences. Making coupons more relevant to customers leads to increased customer satisfaction and increased sales [Cherney et al. 1998].

One way of measuring the effectiveness of a coupon, is to calculate the likelihood of the associated product being purchased - not just in the coupon redemption period but afterwards as well. In other words a coupon can be considered effective if it has the potential to create a purchase habit. Habits tend to lead to increased sales which can have a long term impact on both the retailer and supplier. Using discrete choice modelling, [Lewis 2004] found that promotional coupons can increase annual sales for a substantial portion of exposed customers.

Restrictive Boltzmann Machines (RBMs) are which are a family of deep learning methods suitable for binary problems such as this one (will a promoted product be purchased and purchased again). RBMs may take a specific form when used to calculate the probability of a user i giving a specific rating to a certain product – or in this case a flag of 1/0 for whether the user will buy the item or not in the future. Assuming there are m users and n products, $u_i=1$ will demonstrate that user i will buy a certain product. A set of different units h are used to connect the weights of user to the respective values (of 1 or 0) for the specific products he/she will buy. The hidden units h can be seen as closely related with the latent features in matrix factorization and its size F is a hyper parameter to be tuned. The function S that maps the RBM's formulation (or in general the *Energy term*) for a set of k hidden units can simply be written as equation 2.47:

$$S(u) = -\sum_{i=1}^{m}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{i_j\, h_j u_i}^{k} - \sum_{i=1}^{m}\sum_{k=1}^{K} u_i b_i - \sum_{j=1}^{F} h_j b_j \qquad (2.47)$$

where the weights W connect the u,h visible and hidden units respectively and b their respective biases. The respective gradients of an approximation of the gradient function called *Contrastive Divergence* [Hinton 2002] for a $W_{i,j}$ can be computed as in equation 2.48:

$$\Delta w_{ij} = e\,(< u_i h_j >_{data} - < u_i h_j >_T) \qquad (2.48)$$

where e is the learning rate (normally a small value such as 0.001) and the expectation <.>T refers to the extraction of different samples and formulation of the conditional probabilities p (u_i=1/h) and p(1/V).

2.4 The StackNet model

The StackNet model is named after stacking and neural networks. The following section reviews briefly the notions of stacking and neural networks. Later it examines applications in current literature that have combined the two.

2.4.1 Stacking

[Wolpert 1992] introduced the concept of a meta model being trained on the outputs of various generalisers with the scope of minimizing the generalization error of a target variable. This methodology was successfully used to improve performances in various tasks, including translating text to phonemes and bettering the performance of a single surface-fitter.

According to [Wolpert 1992] stacked generalization includes 2 stages. In the first stage the data is split into 2 parts. A number of different generalizers (or estimators) are fitted on the first part

of the data and then predictions are made for the other part. This process is described in the paper as the *creating phase*. The *guessing phase* on the other hand gathers and concatenates all these predictions forming a new dataset and a new (meta) generalizer is used to treat these as a new data set and make predictions to some other (test) data. Wolpert also states that this approach could even be used with just 1 generalizer in the creating phase. In this particular scenario the meta learner corrects the mistakes of the single model based on the results from the second dataset.

This 2-phase process is illustrated by Wolpert in his original paper (figure 2.4):

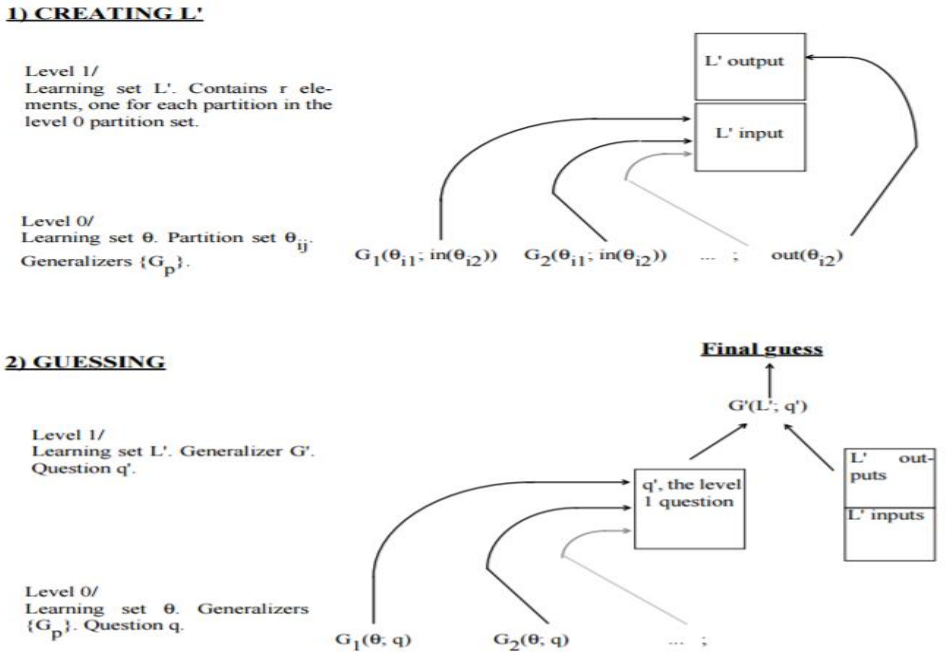

Figure 2.4: The 2 phases of the Stacked Generalization procedure[3]

2.4.2 Stacking diversity and complexity

[Wolpert 1992] addresses the importance of strong generalizers as part of the ensemble in order to achieve better (smaller) errors in the unobserved data, highlighting that "dumb" models (or models that lack sophistication) could be replaced by more sophisticated ones in order to achieve better performance. He also demonstrates that the performance of the ensemble as well

[3] Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241-259.

as its ability to generalize in an unobserved framework is affected by the diversity contained within all the generalizers. [Rogova 1994] further emphasizes that the main component determining the effectiveness of an ensemble is the level of *error-independence* within the contained generalizers. In other words, the generalizers need to be making different errors.

The need for diversity within the ensemble is explored in many different studies including [Sharkey 1996], [Sharkey et al. 1997], [Zhou et al 2002], [Melville et al. 2003], [Melville et al. 2005]. [Sharkey 1996], [Sharkey et al. 1997] define four different types of diversity in an ensemble for classification problems. The first type refers to cases where only one generalizer makes an error within each sample. This type can lead to very high accuracies. Another type of diversity that leads to high accuracy is when the majority of generalizers predict the correct answer. The third type refers to cases where at least one generalizer outputs the right answer (even this scenario can lead to good predictions). The fourth type includes cases where all generalizers make a mistake. In this scenario, getting the right predictions is particularly challenging.

Another way to measure ensemble diversity is presented in the work of [Tsoumakas et al. 2009]. Tsoumakas introduces the idea of correlation–based model pruning whereby models which are highly correlated with another are removed. The study demonstrates that pruning in this way results in a substantial reduction of the computational cost of stacking and can on occasion also improve predictions.

The concept of correlations as a way of detecting diversity within an ensemble is also presented in the work of [Kuncheva et al. 2003]. This study investigates 10 different metrics for measuring diversity within an ensemble. These metrics include four averaged pairwise measures (the Q statistic [Yule 1900], the correlation [Sneath 1973], the disagreement [Ho 1998], the double fault [Giacinto et al. 2001]) and six non-pairwise measures (the entropy of the votes [Cunningham et al. 2000], the difficulty index [Hansen et al. 1990], the variance [Kohavi-Wolpert 1996], the interrater agreement [Dietterich 2003], the generalized diversity [Partridge et al. 1997] and the coincident failure diversity [Partridge et al. 1997]). Although the study highlights the link between diversity and performance, it is unable to identify a definite link between diversity and improvements in accuracy. They further conclude that the problem of measuring diversity and so using it effectively for building better classifiers is still to be solved.

2.4.3 Neural Network

A specific type of neural network has already been defined in the current thesis in the previous chapter. Conceptually such artificial networks were first created in an attempt to mimic the biological neural networks in the human brain. [Rosenblatt 1958] was the first to create a very simple version of a neural network – the perceptron.

The usage of ANNs (Artificial Neural Networks) flourished when back propagation was developed and it was found that it could be used to combine multiple perceptrons in the form of various hidden layers between some input data and an outcome.

The advances in computing power and specifically the usage of GPUs have allowed the previously slow NN machine learning models to be run at greater speeds [Schmidhuber 2015] taking the form of today's *deep learning*.

Furthermore the inclusion of a dropout term, advances in gradient-based methods as well as regularization methods as a means to prevent neural networks from both over and under fitting have further boosted the use of these algorithms in various fields including image, sound, and text classification as well as recommendation tasks [Hinton et al. 2014].

2.4.4 Applications for combining Algorithms on multiple levels

[Breiman 1996] borrows the idea of [Wolpert 1992] for stacked generalization and extends it to regression trees as well as ridge regressions using cross validation as a means to give improved prediction accuracy.

[Van der Laan et al. 2007] propose a new prediction method for creating a weighted average of many candidate algorithms to build a super learner. They propose a fast algorithm for constructing this super learner in a prediction which uses V-fold cross-validation to select weights to combine an initial set of candidate learners.

[LeDell 2015] proposes a scalable learning methodology using a super learner (also known as stacking) to combine multiple, typically diverse, base learning algorithms into a single, powerful prediction function through a secondary learning process called meta learning . This

methodology reduces the computational burden of ensemble learning while retaining superior model performance.

[Zhou et al. 2017] propose the use of multi-level random forests to improve predictions in the image classification space and achieve similar performance to other state of the art methods including convolutional neural networks.

3. Univariate Analysis of the Dataset

This chapter presents an overview of the dataset that will be used in this thesis and maps the features that will be used in the analysis that facilitate predicting what the customer will buy next week given a visit. Given the nature of the grocery sales data, understanding the drivers that connect customer and products is critical for applying specific machine learning algorithms and to improve performance.

3.1 Overview of Available Data Sources

This chapter uses the freely-available data for research from dunnhumby ltd[4] . Specifically, the set of available data is named "*The Complete Journey"* and holds the complete transactions of 2,500 frequent buyers for a supermarket chain for a period of 102 weeks amounting to 2,595,732 entries including fields such as time stamp, discount, price, store and place. The term used to describe the buyer is "household_key" and represents the buyer-entity of the purchases that are made. It is should be noted that it is likely that the members of the same family belong to the same household_key.

The retailer has also provided information about the products (expressed via a product_id) such as the brand-manufacturer or the department (such as *Dairy*) where the product belongs to. There are 92,339 unique products along with 6,476 different manufacturers and 44 distinct departments. Additionally, in the supplied datasets there is a unique list of all the household_keys with demographic information about the household's status such *as age band, income band, marital status* and more. Coupon and promotional data are also presented in separate files, but will not be exploited in this chapter (see Chapter X).

3.2 Defining the experiment

[4] www.dunnhumby.com/sourcefiles

The following sections share generic information about the population of the experiment and the target variable. In addition, the presence of seasonality is highlighted alongside the distinctive ability of Customer's historical frequency of purchase explaining their future purchases. Later sections display a hierarchy of predictive features considered for the experiment as well as an algorithm that finds the optimal bins for each feature in respect to the target variable. This is to best capture non-linear relationships and gauge their predictive power more efficiently. The impact of this type of continuous feature discretization is gauged via estimating the difference in predictability of features (as measured by AUC) before and after binning has been applied both in actual and proportional terms.

3.2.1 Modelling population and target Variable

The main focus of this chapter is to explain some key features (based on the provided sets of data) that may affect the probability of a customer buying a product next week given a visit. The latter part emphasizes that the focus is on predicting the correct products assuming the customer has visited the store in a target time period. This is also boosted from the understanding that, generally, all 2,500 households are frequent buyers. Therefore, a model of whether the customer will visit or not is not critical in this case. It can be assumed that these are customers who do the majority of their shopper at this particular supermarket chain.

The volume of 92,000 different products is deemed unnecessarily high since some of these products were very rarely bought during the year. To simplify the process, only products that were bought more than 30 times in the last year (weeks from 50 to 102) were considered in the process - resulting in only 9,788 unique products ids, which is around *10 %* of the initial pool of items. This suggests that most of the transactions of this retailer are concentrated in a small group of products. For the scope of this analysis, each customer (out of 2,500) is related with any of the 9,788 products yielding 24,470,000 possible customer-to-product pairs to be explored for a particular week.

The fundamental question to answer then becomes: *if some of the purchasing history for these customers in respect to these products, as well as general information about the customer and the item, is known at a point in time, can sensible predictions be made for as to whether the customer will buy the product in a specific (future) point (week)?* In light of this, two different datasets are formed in different time periods (with the same features) to serve as training and

test datasets. The first set uses week 100 as the target week (for which predictions are due) while the test week is number 101. For each different target week, 52 previous weeks are used to compute features that summarize the relationship of the customer and the products. The demographics and product detail data are assumed to be stable for the purposes of this analysis. The feature engineering process that utilizes the past year of transactions to create aggregated features for up to the target week can be visualized in graph 3.1:

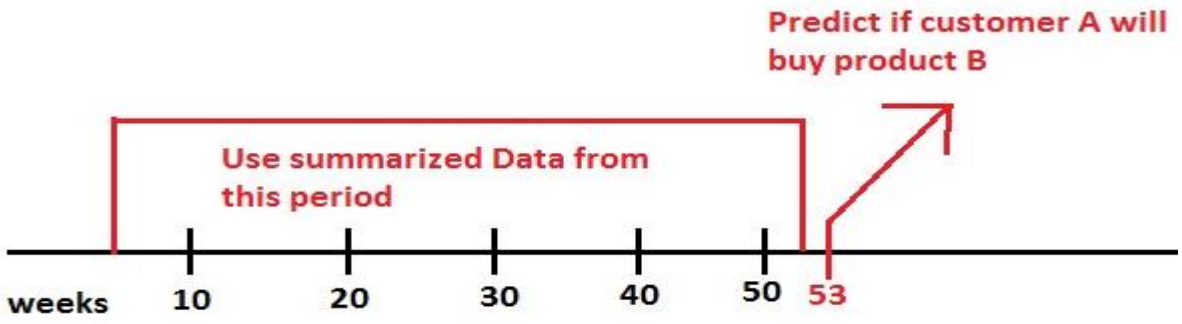

Figure 3.1 : Feature engineering process and definition of target variable

The initial set of 24,470,000 unique customer-to-product pairs is further halved by including only those customers that did visit the retailer's stores during week 53 (or 54 for the test set). Only 1,288 households visited in the target week 53 and 1,311 in week 54. The distinct number of different products bought by the selected group of customers in the target week is 27,109, and 21,540 for the test set. This can also be viewed as the total pairs triggered out of the total customer-to-item combinations, a fairly small but still viable number. For the purposes of the univariate analysis, the datasets are merged to aid the significance of the results.

3.2.2 The notion of seasonality and time lag

It is generally agreed that customer preferences may change (or discontinue as stated in [Tripsas 2008]) over time, so any feature engineering that attempts to map the customer relationship with the item or the item's global preference in a certain point in time needs to account for that possible fluctuation. Furthermore, there are products that are bought either more or less during certain periods owing to seasonality or because of unobserved factors (such as general popularity decline because of a competitor's new product). To better visualize this in figure 3.2, the product with id 826249 demonstrates fluctuation in its weekly sales over the

102-weeks timespan. Quite clearly a prediction for week 62 or 24 where product sales seem to be rising would be more optimistic versus weeks 1 or 2.

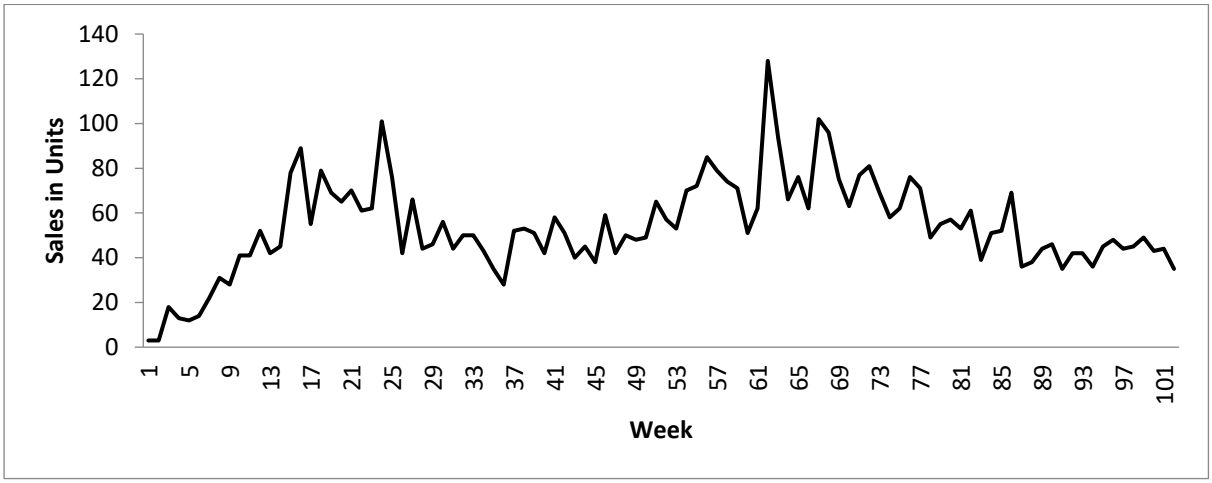

Figure 3.2 : Seasonality in weekly sales in units for product 826249

To address this, all features created through the current feature engineering process based on the transactional history will be expressed through different time periods and lags. For example, instead of simply creating the number of times the customer bought the item in the last 52 weeks, last 39, last 26, last 13, the ratios of these will be created in respect to the target week too. Note that the weeks are NOT mutually exclusive (i.e. week 13 is included in week 52). The process can be illustrated with figure 3.3:

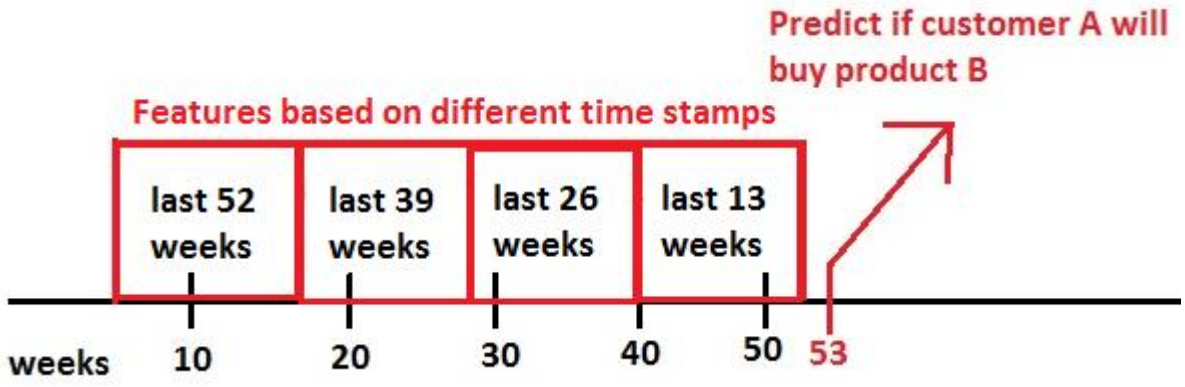

Figure 3.3 : Feature engineering process for different time stamps

3.2.3 Dominance of the frequency of purchase and exploiting the product hierarchy

The most obvious and strong finding regarding whether the customer will buy a product in a particular week or not is the number of times he/she has bought this item in the past (i.e. last 52 weeks). This can be referred to as *the frequency of purchase in the last 52 weeks*. Thus out of the 9,800 items considered for each customer, as illustrated in figure 3.4, they have almost 3% probability to buy an item he/she has bought before in the last 52 weeks (which accounts for 55% of the total pairs that actually happen in the target week) versus 0.07% for an item he/she has not:

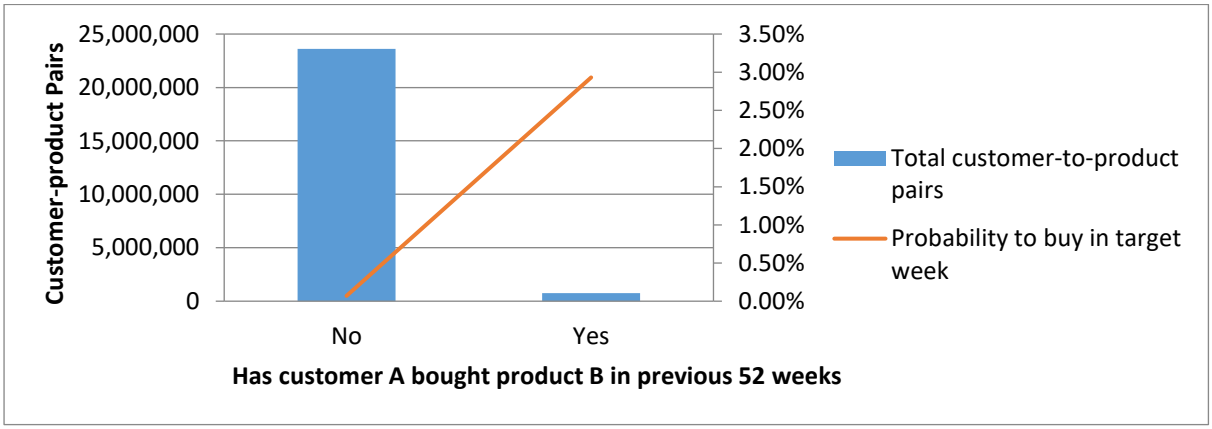

Figure 3.4 : Probability to buy an item in the target week given previous purchase status

In other words, by knowing whether the customer has bought the item in the past, it helps isolate 55% of the total customer-item pairs that occur in the target week in a much smaller part of the population. In contrast, the remaining 45% of the total occurring pairs that belong to a group of products that the customer has never bought, necessitate further information to aid the capture the underlying relationships between the two.

This gap is aimed to be filled by exploiting the product hierarchy, which in this scenario will be the *manufacturer* and the *department* in which the item belongs to, since a customer has higher chance to buy a product that belongs to a category or brand he/she has purchased before. As can be illustrated in table 3.1, a customer is twice as likely to buy an item from a manufacturer he/she bought before 12 or more times than one he/she has bought fewer times in the previous 52 weeks.

Table 3.1 : Probability to buy an item from a manufacturer given previous purchases

Times bought in previous 52 weeks	Probability to buy in target week
0 to 11	0.09%
more than 11	0.17%

3.2.4 Predictive grouping of the features

The feature space in this chapter is divided to a finite number of distinct groups to better exploit its potential and comprehend its predictive power. These groups can be summarized as:

1. Household features : These are features about the household itself and can be further divided to :
   1.1. Demographics: This is provided by the retailer, features like age or income band.
   1.2. Transaction-based: These are features that are created by aggregating the transactional data and they aim to capture loyalty (such as number of visits or average spending) and cardinality (potential of the customer to try many different products or/and tendency to buy new items.)
2. Product features: These refer to certain attributes of the product such as popularity and accessibility.
3. Manufacturer features: Same as product but for manufacturer.
4. Department features: Same as product but for department.
5. Household and product features: This refers to a set of features that map the historical relationship of a customer with the product such as times bought, average cycle and last bought.
6. Household and manufacturer features: Similar as above but for manufacturer.
7. Household and department features: Similar as above but for department.
8. Contextual features: This includes time and day of the week.

3.2.5 Optimized binning to capture non-linearity

To better assess the potential of numerical continuous variables to explain the target variable, optimized binning was used to split the feature into segments and replace each part with the log of the odds of the target variable for this segment in order to capture the inherent nonlinearities while maximizing a specific metric. As explained above and since this is essentially a binary classification problem, the metric to optimize was AUC. The logic of the algorithm is similar to the MDLP (Minimal Description Length Principle) explained in 2.1.3, with the main difference that the metric used to define the best bins at its iteration is AUC. The methodology is also expressed to work against any potential optimization metric for regression or classification.

Initially a continuous variable is divided into 100 equal (in size) bins where the odds of purchases versus non-purchases are known for the target week. Note that the number of the initial bins is a hyper parameter and can be selected differently. Then in order to reduce the number of bins and therefore increase the number of observations per bin, the best pair of neighboring bins to merge are found by considering all possible combinations and comparing the uplift of AUC until the number of bins is trimmed down to 10 (which is also a hyper parameter).

Optimized binning Algorithm is displayed using pseudo code in figure 3.5:

Input: Feature X, Initial number of bins N, final number of Bins n, metric as m

Output: Model parameters = (X_{new})

$\boldsymbol{X_{new} \leftarrow X(N)}$

$\boldsymbol{While\ N > n\ do}$:

$\quad \boldsymbol{for\ i} \in (\mathbf{2}, \ldots\ldots, \mathbf{N})\ \boldsymbol{do}$:

$\quad\quad \boldsymbol{X_{new_{temp}}(N-1) \leftarrow merge\left(X_{new}((i-1),(i))\right)}$

$\quad\quad \boldsymbol{metric_i \leftarrow m\left(X_{new_{temp}}(N-1)\right)}$

$\quad \boldsymbol{X_{new}(N-1) \leftarrow merge\left(X_{new}((\mathbf{maxindex}(metric_i)-1), maxindex(metric_i))\right)}$

$\quad \boldsymbol{N \leftarrow N-1}$

$\boldsymbol{Until\ Stopping\ criteria\ is\ met}$

Figure 3.5 : Optimised Binning Algorithm

3.3 Features' ranking and dictionary

The previous data exploration phase generated over 100 features (with lags and different time stamps) some of which were discarded from the potential pool of variables to consider for being too weak or for having significant number of missing values. The final set of features sums to 75 – all categorical, via transforming the continuous ones with the optimized binning algorithm to better capture the non-linearity that connects them with the target variable.

The following table of features provides a description of the generated features (accounting for the appropriate time lag) along with a marker that points into which one or many of the previously-mentioned features' groups the features belong to. Additionally the table is sorted in a descending manner using the AUC statistic that has been explained before. The interpretation should be that the higher this statistic, the higher the predictive power of the feature to efficiently point to the '0' or '1' class of the target variable. It should be noted that this is just the univariate descriptive power of these features. It may be assumed that many of the features share common information (or in other words there is multi-colinearity in the data) and that a feature's unique descriptive power to predict the target variable may not be accurately found via this method. Nevertheless that can be used to understand the main predictive power of each feature group.

The different feature groups as described in section 3.2.4 are denoted as C for customer, P for product, D for department and M for manufacturer. All the generated features used in this experiment are portrayed more analytically in table 3.2. The table displays the feature name, a brief description for what it represents, the different feature groups it belongs too as well as the AUC of the feature in respect to the target variable after applying the optimized binning algorithm in figure 3.5:

Table 3.2 : Features' dictionary with predictability statistics and features' groups mapping

Feature name	Feature Description	C	P	D	M	AUC
frequency26	Number of baskets the customer included the product in last 26 weeks	✓	✓			0.775

frequency39	Number of baskets the customer included the product in last 39 weeks	✓	✓		0.775
frequency52	Number of baskets the customer included the product in last 52 weeks	✓	✓		0.775
frequency13	Number of baskets the customer included the product in last 13 weeks	✓	✓		0.775
cycle_vs_lastbought	Average cycle (52 weeks) minus days ago since last bought the product	✓	✓		0.775
average_cycle52	Every how many days the customer bought the product in last 52 weeks	✓	✓		0.774
last_day_bought	Days from the target week since the customer last bought the product	✓	✓		0.774
average_cycle39	Every how many days the customer bought the product in last 39 weeks	✓	✓		0.766
average_cycle26	Every how many days the customer bought the product in last 26 weeks	✓	✓		0.747
popularity13	Number of baskets the product appeared in last 13 weeks		✓		0.747
popularity26	Number of baskets the product appeared in last 26 weeks		✓		0.742
popularity39	Number of baskets the product appeared in last 39 weeks		✓		0.739
popularity52	Number of baskets the product appeared in last 52 weeks		✓		0.735
average_cycle13	Every how many days the customer bought the product in last 13 weeks	✓	✓		0.709
frequencies_decay	frequency52 divided by frequency13	✓	✓		0.709
frequency13man	Same as frequency13 but for "manufacturer"	✓		✓	0.708
frequency26man	Same as frequency26 but for "manufacturer"	✓		✓	0.707
frequency39man	Same as frequency39 but for "manufacturer"	✓		✓	0.704
frequency52man	Same as frequency52 but for "manufacturer"	✓		✓	0.702
average_cycle52man	Same as average_cycle52 but for "manufacturer"	✓		✓	0.698
average_cycle39man	Same as average_cycle39 but for "manufacturer"	✓		✓	0.695
average_cycle26man	Same as average_cycle26 but for "manufacturer"	✓		✓	0.695
most_trialled	Number of customer who bought the item 1st time the previous week		✓		0.687
average_cycle13man	Same as average_cycle13 but for "manufacturer"	✓		✓	0.686
frequenciesman_decay	frequency52man divided by frequency13man	✓		✓	0.683
productsbought13	Total number of products the customer bought in last 13 weeks	✓			0.632
productsbought26	Total number of products the customer bought in last 26 weeks	✓			0.632
productsbought39	Total number of products the customer bought in last 39 weeks	✓			0.630
distinct_item	Distinct number of products the customer bought in last 52 weeks	✓			0.625
productsbought52	Total number of products the customer bought in last 52 weeks	✓			0.625
distinct_MANUFACTURER	same as distinct item but for "manufacturer"	✓			0.620
distinct_DEPARTMENT	same as distinct item but for "department"	✓			0.591
manpopularity52	same as popularity52 but for "manufacturer"			✓	0.590
popularity_decay	popularity52 divided by popularity13		✓		0.588
manpopularity39	same as popularity39 but for "manufacturer"			✓	0.586
manpopularity13	same as popularity13 but for "manufacturer"			✓	0.586
manpopularity26	same as popularity26 but for "manufacturer"			✓	0.585
frequency26dep	Same as frequency26 but for "department"	✓		✓	0.584
frequency39dep	Same as frequency39 but for "department"	✓		✓	0.583
frequency13dep	Same as frequency13 but for "department"	✓		✓	0.583
frequency52dep	Same as frequency52 but for "department"	✓		✓	0.579
visits26	Number of distinct days the customer visited in last 26 weeks	✓			0.577

visits13	Number of distinct days the customer visited in last 13 weeks	✓			0.577
transactions_withdiscount	Total number of transactions with discount in last 52 weeks	✓			0.577
visits39	Number of distinct days the customer visited in last 39 weeks	✓			0.574
deppopularity13	same as popularity13 but for "department"			✓	0.573
deppopularity26	same as popularity26 but for "department"			✓	0.573
deppopularity39	same as popularity39 but for "department"			✓	0.573
deppopularity_decay	deppopularity52 divided by deppopularity13			✓	0.573
visits52	Number of distinct days the customer visited in last 52 weeks	✓			0.569
transactions_withdiscountman	Number of times the manufacturer was sold with discount in 52 weeks			✓	0.567
transactions_withdiscountdep	Number of times the department was sold with discount in 52 weeks			✓	0.565
manpopularity_decay	manpopularity52 divided by manpopularity13			✓	0.563
count_newitems	Number of products the customer bought last week for the 1st time	✓			0.562
frequenciesdep_decay	frequency52dep divided by frequency13dep	✓		✓	0.559
average_cycle52dep	Same as average_cycle52 but for "department"	✓		✓	0.559
HH_COMP_DESC	Household status	✓			0.557
INCOME_DESC	Household income band	✓			0.557
average_cycle39dep	Same as average_cycle39 but for "department"	✓		✓	0.556
AGE_DESC	Household Age Band	✓			0.556
KID_CATEGORY_DESC	Household's kid category description	✓			0.555
average_cycle26dep	Same as average_cycle26 but for "department"	✓		✓	0.555
MARITAL_STATUS_CODE	Household's Marital Status	✓			0.553
average_cycle13dep	Same as average_cycle13 but for "department"	✓		✓	0.552
HOMEOWNER_DESC	Household's homeowner status	✓			0.551
average_spendingitem	Average spent on a product in last 52 weeks		✓		0.542
deppopularity52	same as popularity52 but for department			✓	0.541
HOUSEHOLD_SIZE_DESC	Household Size band	✓			0.540
average_discount	Average discount per product in basket in last 52 weeks	✓			0.540
average_discountitem	Number of times the product was sold with discount in last 52 weeks		✓		0.537
transactions_withdiscountitem	Number products the customer bought with discount in last 52 weeks	✓			0.535
visits_decay	visits52 divided by visits13	✓			0.535
average_spending	Average spending per product in basket in last 52 weeks	✓			0.533
average_quantity	Average quantity per product in basket in last 52 weeks	✓			0.531
TRANS_TIME	Time in hours where 12 am is '00' and 11pm is '23' (24 distinct values)				0.529

The strangest single feature of table 3.2 boasts an AUC of 0.775 (frequency26) which can be considered quite high as it comes from a single feature and shows once again the importance of an existing relationship between a customer and a product.

3.4 Univariate Analysis

The findings of the previous section will be summarized in tandem with the hierarchy of the feature groups presented in the previous section of this chapter:

3.4.1 Household and Product features

This group of features is by far the most important one in this analysis as it dominates the top of the features board. It is sensible that the more times a customer has bought a product the higher the chance to buy it at any given week (see figure 3.6):

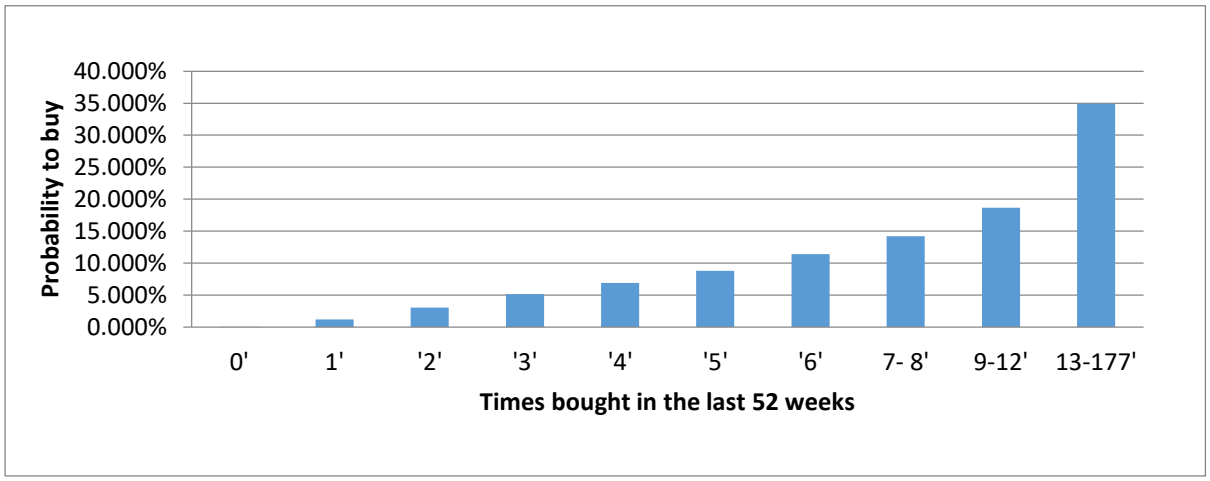

Figure 3.6 : Frequency of purchase of last 52 weeks vs. target

For items bought over 13 times, the probability to buy becomes immensely (and non-linearly) higher than the rest of the bands and can reach 35 %.

Additionally for a customer, knowing the average number of days between purchases of a product and subtracting that number by the number of days since he/she last bought it, can facilitate understanding when he or she is going to buy the item again in the future. For example if a customer buys a product every 12 days and it was last bought 5 days ago, he/she is expected to buy the item 7 days from now. Negative values for this feature represent customers who stopped buying the product after some point and positive values customers who may have just

bought the item and it is not the time yet to buy it again. This feature is represented in figure 3.7:

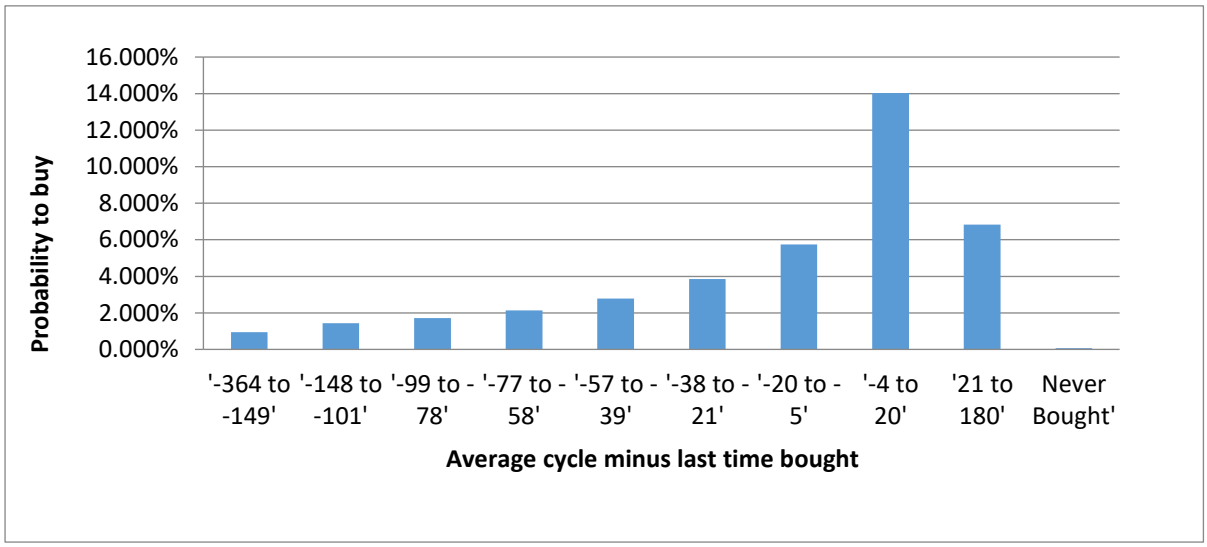

Figure 3.7 : Average Cycle minus last time bought vs target

Another interesting point arises by visualizing the ratio of purchases in the previous 52 and 13 weeks. Assuming that the customer buys the product evenly across the year then a ratio of 4 would be expected, since the first 13 weeks are included in 52. Figure 3.8 clearly shows that when the times the customer bought the item in last 52 weeks is more than 4 times bigger than the times bought in last 13 weeks, then the probability to buy the product increases.

Figure 3.8 : Frequency's decay vs target

3.4.2 Household and manufacturer or department

The importance of this group lies in when a direct relationship between a customer and product is not known and therefore these higher levels of the product hierarchy are used to infer it. As can be speculated the relationship is not as strong as with the products themselves, however the discrimination is clear (i.e. higher values denote higher probability to buy the item) as illustrated in the customer's department frequency of purchase of the last 52 weeks in figure 3.9:

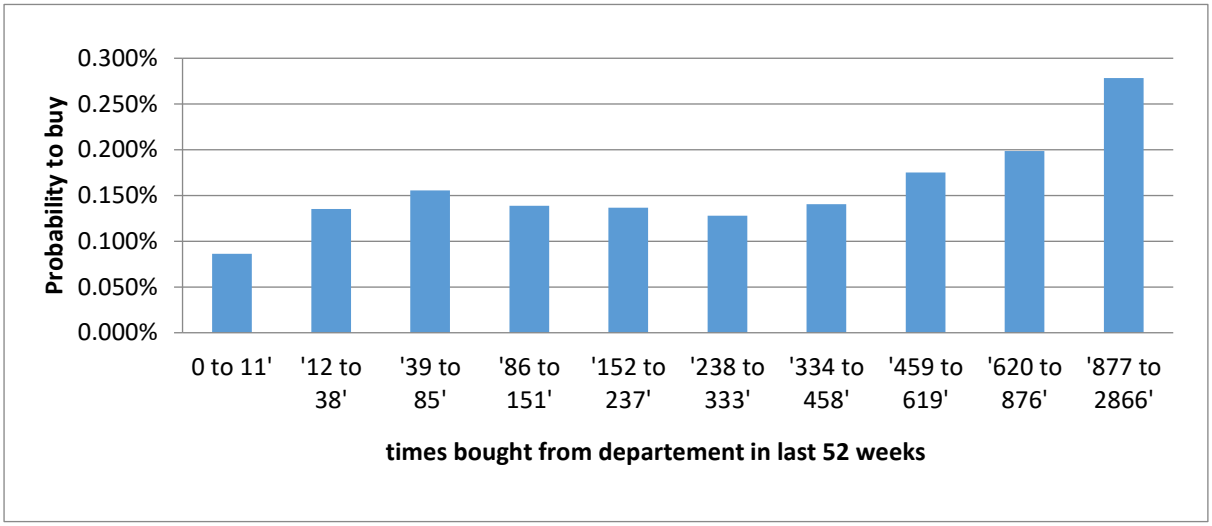

Figure 3.9 : Frequency' of purchase of department in last 52 weeks vs target

3.4.3 Product Features

The product related features rank second in the list of the most predictive features and are quite important because a household-to-product link is not always assumed. The mapping of different periods and lags allows the capturing of additional seasonality elements in the products. Product popularity over the last 52 weeks is defined as the number of different baskets, in which the product was included. There is a strong positive relationship between the times included and the probability to buy in the target week as illustrated in figure 3.10:

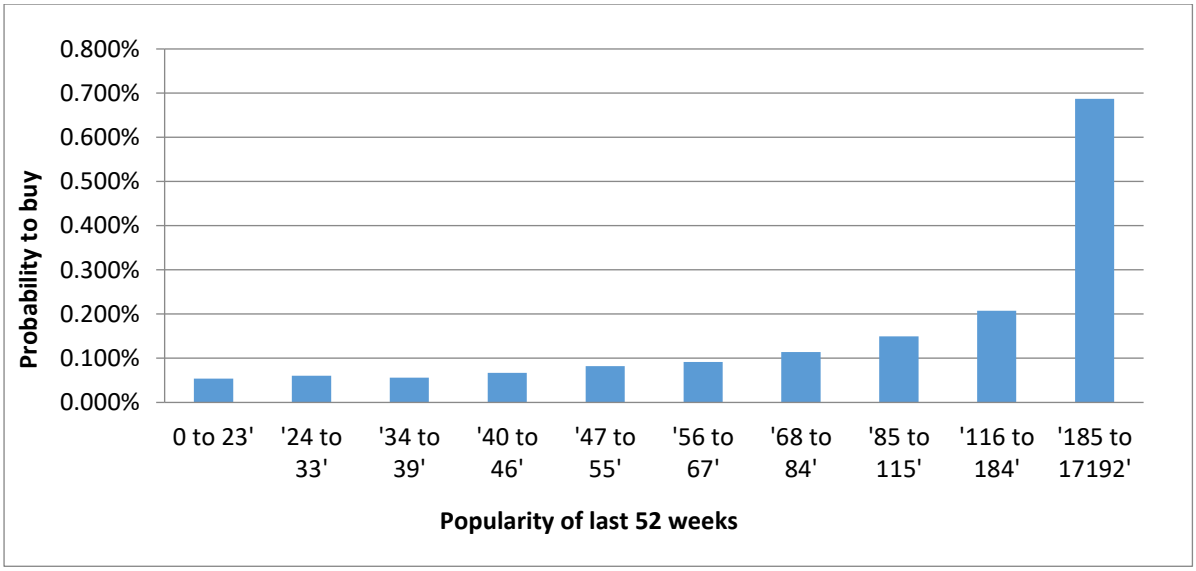

Figure 3.10 : Product popularity of last 52 weeks versus target

In order to better capture the propensity of customers to buy new products (that they have never bought before) and to map seasonality, a *trialed* feature was introduced, which can be defined as *the number of different customer that bought the items in the previous-to-the-target week that have never bought it before in previous 51 weeks*. In other words it expresses the tendency of the product to be bought for the first time in the very recent week. The greater the number of people who bought the product for the first time in the previous week, the higher the chance for a given customer to buy that item in the target week as illustrated in figure 3.11:

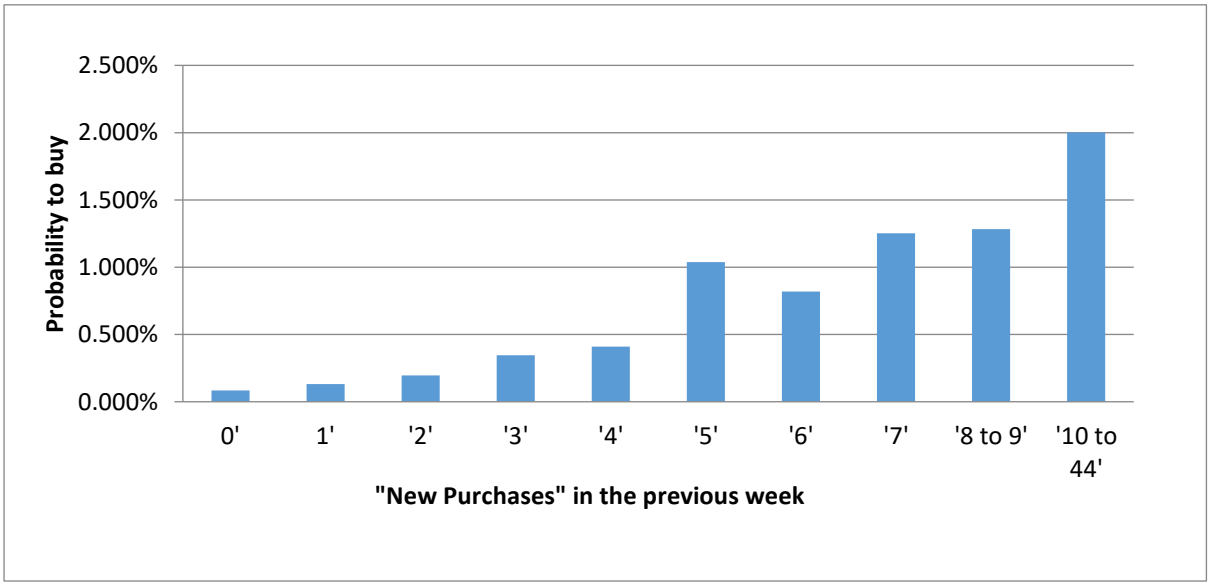

Figure 3.11 : Trialled products' popularity versus target

Compared to product popularity, the top band in this feature can yield higher probabilities, max 2% versus 0.7%.

Another interesting finding is the relationship between popularity of last 52 weeks divided by the popularity of last 13 weeks as displayed in figure 3.12. Unlike frequency of purchase, products that are slightly under-indexed (in this scenario, bought more frequently as of late) seem to have higher probability to be bought in the target week.

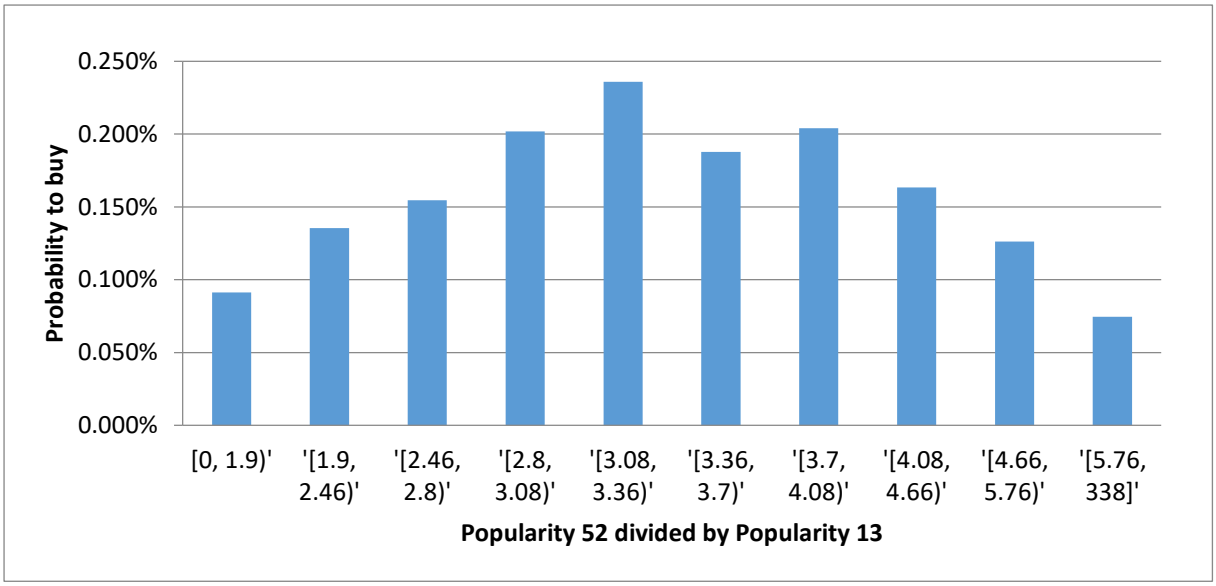

Figure 3.12 : Popularity decay versus target

3.4.4 Department and Manufacturer

The assumption with this group of features was that in cases where the product is very difficult to be determined, at least the notion of department or manufacturer can give some insight as to what products may be more popular in a specific period. Admittedly this is the weakest of all other groups and the respective popularity-based features seem to expresses a similar but much weaker positive relationship with the target variable.

3.4.5 Household features

As previously stated, household features may either be derived specifically from the transactional data or can be extracted from the demographics dataset the retailer has provided. The most dominant features in the latter group are the age in figure, income and kids bands. As

can be viewed in figure 3.13, the age groups between 25 and 54 possess higher probability to buy any product in the target week:

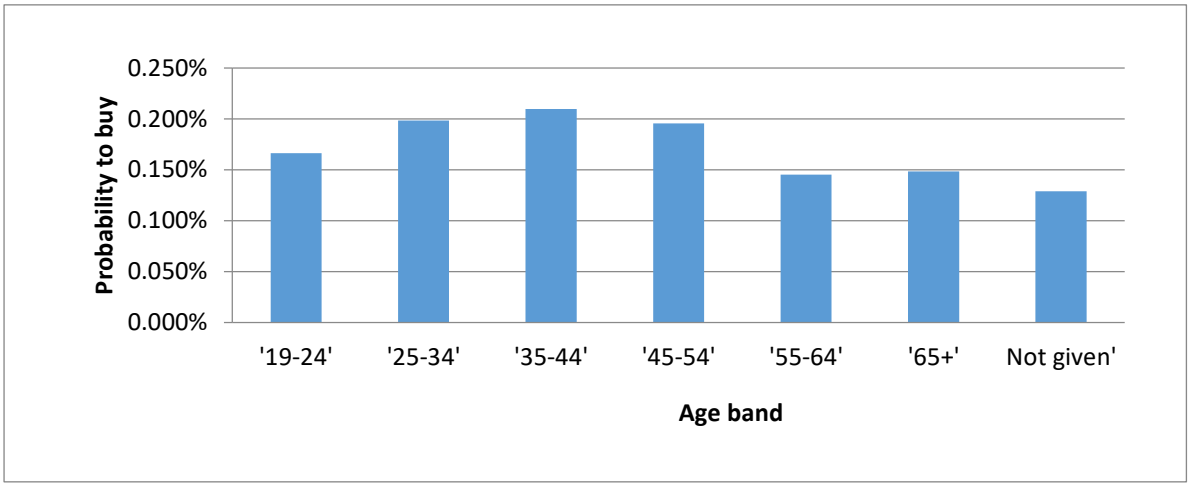

Figure 3.13 : Age band versus target

Additionally there seems to be a positive relationship between income and propensity to buy where higher income is associated with higher chance to buy a product in the target week as illustrated in 3.14:

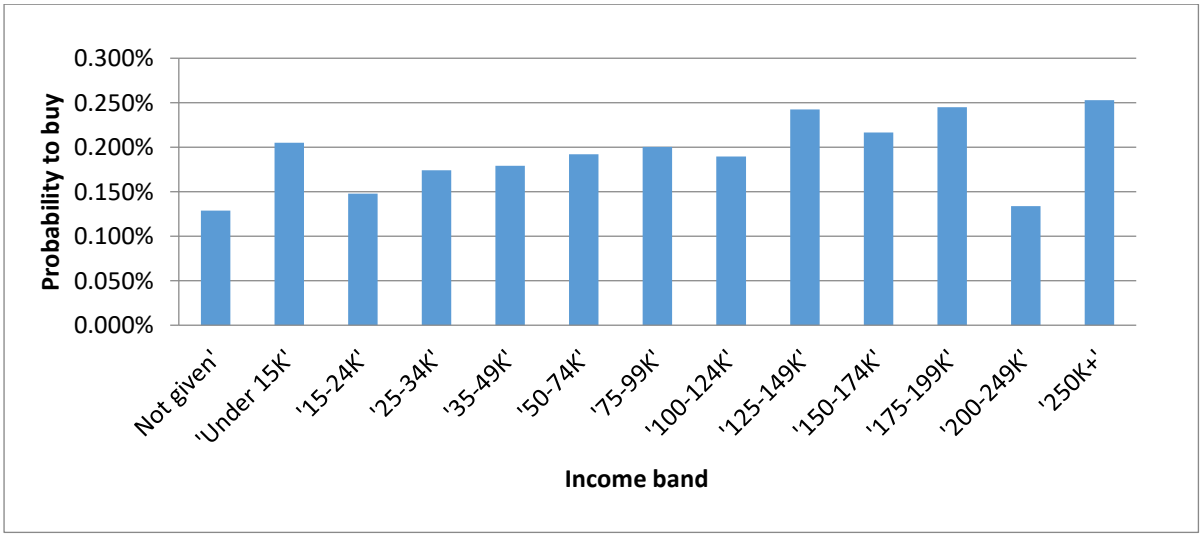

Figure 3.14 : Income band versus target

Ultimately the number of children also possesses a positive relationship with the target as more kids result in higher propensity for purchase any product as can also be viewed in figure 3.15:

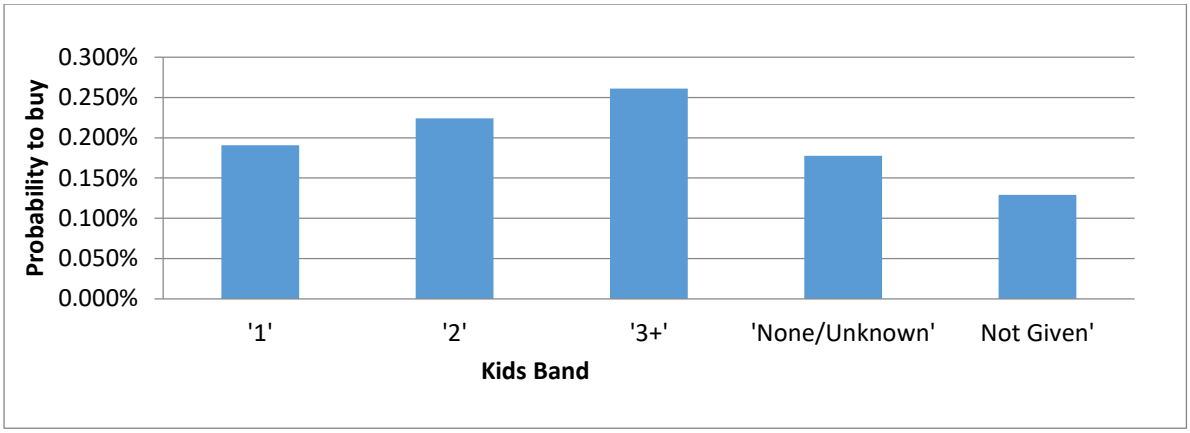

Figure 3.15 : Kids band versus target

Another household-type group of features is centered on measuring how loyal a customer is by how frequently he/she visits the retailer's stores. The assumed relationship (i.e. the more a customer visits the higher the chance to buy any item) is verified as illustrated in figure 3.16 where more visits for a customer equate to higher probability to buy any product:

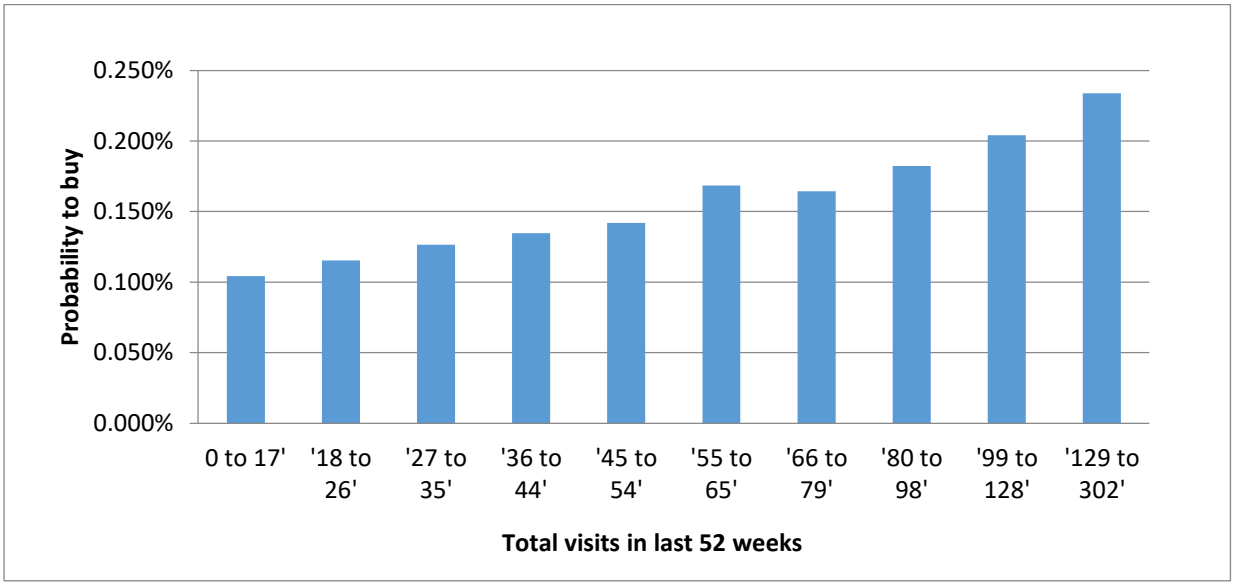

Figure 3.16 : Total visits in last 52 weeks vs target

Furthermore, customers' cardinality-based features (such as number of different products the customer has bought in 2.17) possess predictive power too:

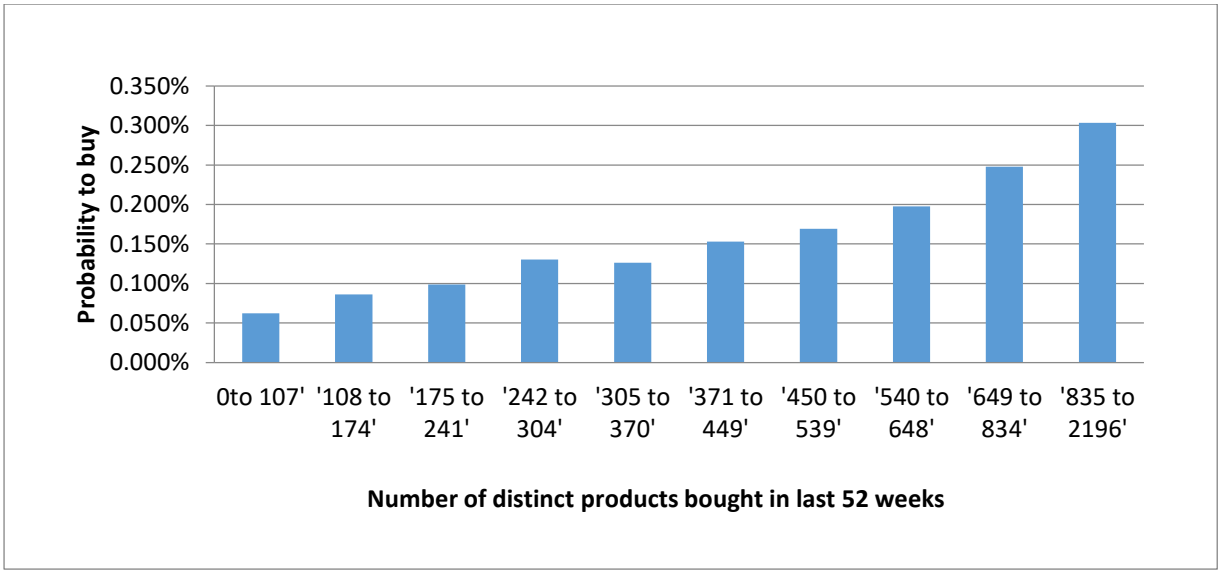

Figure 3.17: Number of distinct products vs target

Another useful household-type feature is the count of products the customers has bought for the first time in the previous (from the target) week. This feature expresses the tendency of the customer to buy items he/she has never bought before and is illustrated in figure 3.18. It displays a positive relationship with the target variable:

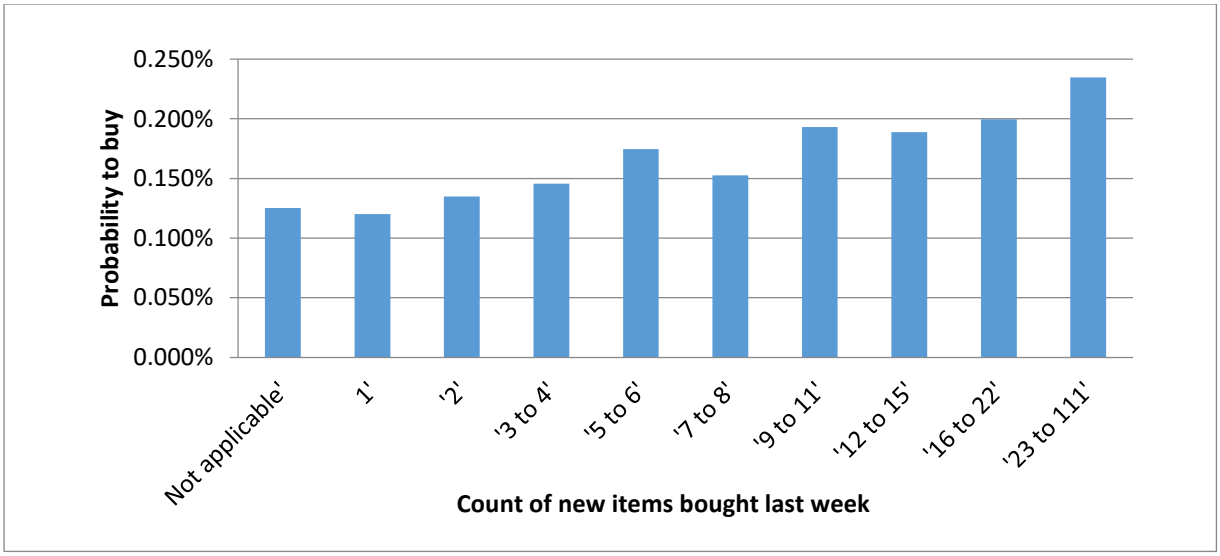

Figure 3.18 : New items bought and number of distinct products vs target

The group of customers who are more adventurous and like to try products they have never bought before have higher probability to buy any item in the target week.

3.4.6 Contextual feature – day of the week

An interesting pattern is present in the day of the week in figure 3.19. There is less tendency to buy an item in the middle of the week, compared to the other days.

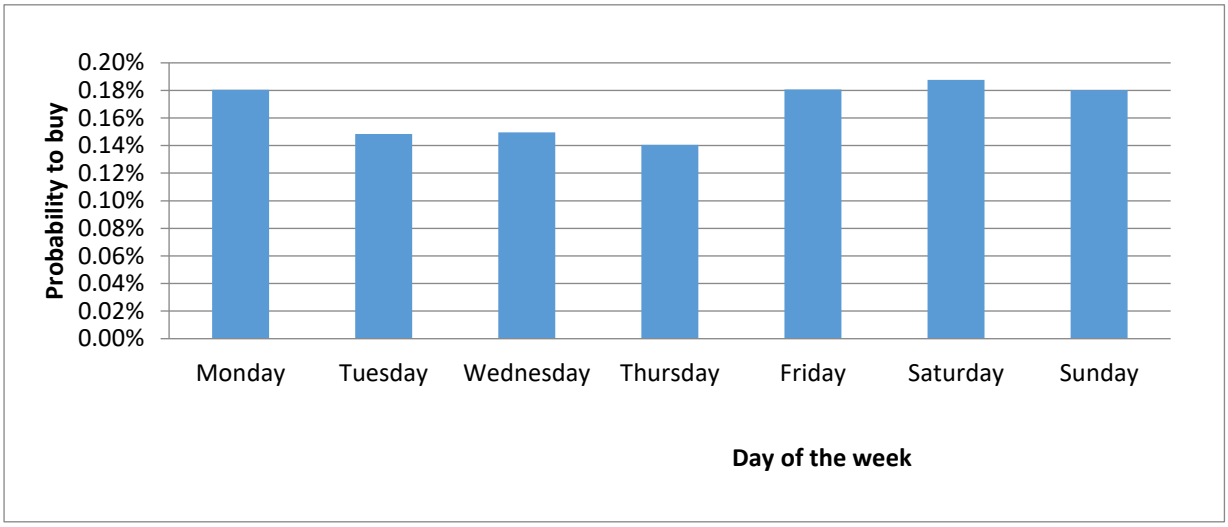

Figure 3.19 : Day pf the week vs target

3.5 Impact of binning

Under certain assumptions it is feasible to quantify the impact of binning for the numerical features in respect to AUC. For completeness the quantification will include some of the demographic features which were given as binned categorical variables even though they were not subject to the optimized binning algorithm. The proposed method suggests measuring the difference in AUC before any binning was applied and after, both in simple and proportional terms. Gain can be defined as AUC after binning minus AUC before binning. Proportional gain can be defined as gain divided by AUC before binning.

HH_COMP_DESC, KID_CATEGORY_DESC, MARITAL_STATUS_CODE and HOMEOWNER_DESC were excluded from this comparison because they could not be expressed in a numerical form. The rest of the categorical variables have been converted to numerical using the average of the band they represent. For example in AGE_DESC category '35-44' was replaced with 39.5.

Before applying binning, all missing values for a given feature were assigned a value that is lower than the minimum of all the non-missing values for that feature. This decision is derived logically as in most cases missing values in this context are generated due to absence of past purchases and are associated with a lower chance to buy any item. If missing values were replaced with the mean, this would have favored the binning method since for many of the non-binned features it would have interrupted the increasingly monotonic relationship they may have with the target variable.

After applying binning, all binned values (or categories) are replaced with the average of the target variable. For example the category of '35-44' in AGE_DESC has an average probability to buy of 0.2098%, hence all '35-44' categories are replaced with 0.002098. In this context all missing values are treated as a separate category and are replaced with the average probability to buy, same as with any other variable.

The table 3-3 presents the gain of all features in terms of AUC before and after binning, sorted in a descending manner by proportional gain.

Table 3-3: Comparison of AUC before and after binning sorted by proportional gain

Included Fields	AUC before binning	AUC after binning	AUC gain	Gain%
popularity_decay	0.4929	0.5885	0.0955	19.38%
manpopularity_decay	0.4927	0.5633	0.0706	14.32%
deppopularity_decay	0.5023	0.5728	0.0705	14.04%
visits_decay	0.4832	0.5352	0.0520	10.75%
average_spending	0.4849	0.5330	0.0481	9.92%
TRANS_TIME	0.4891	0.5293	0.0402	8.23%
average_spendingitem	0.5137	0.5423	0.0286	5.57%
average_quantity	0.5136	0.5309	0.0173	3.37%
AGE_DESC	0.5438	0.5556	0.0118	2.17%
average_cycle52man	0.6867	0.6979	0.0112	1.63%
frequency52man	0.6907	0.7019	0.0112	1.62%
average_cycle39man	0.6860	0.6955	0.0095	1.38%
average_cycle26man	0.6854	0.6947	0.0093	1.36%
average_cycle13man	0.6773	0.6864	0.0091	1.34%
frequency39man	0.6945	0.7038	0.0093	1.34%
frequency13man	0.6986	0.7077	0.0091	1.30%
frequency52dep	0.5716	0.5785	0.0069	1.20%
frequency26dep	0.5773	0.5840	0.0067	1.16%
frequency26man	0.6988	0.7068	0.0080	1.14%
frequency13dep	0.5765	0.5826	0.0061	1.06%
frequenciesman_decay	0.6768	0.6835	0.0067	0.99%
frequency39dep	0.5772	0.5828	0.0056	0.97%
average_cycle26dep	0.5497	0.5550	0.0053	0.96%
distinct_DEPARTMENT	0.5856	0.5911	0.0054	0.93%
average_cycle52dep	0.5538	0.5589	0.0051	0.92%
average_cycle39dep	0.5517	0.5565	0.0048	0.87%
average_cycle13dep	0.5473	0.5517	0.0044	0.80%
distinct_MANUFACTURER	0.6158	0.6204	0.0046	0.75%

INCOME_DESC	0.5541	0.5567	0.0026	0.47%
manpopularity52	0.5872	0.5896	0.0024	0.40%
average_discountitem	0.5344	0.5365	0.0021	0.39%
transactions_withdiscountitem	0.5335	0.5352	0.0017	0.32%
count_newitems	0.5600	0.5617	0.0016	0.29%
transactions_withdiscount	0.5757	0.5768	0.0011	0.20%
average_discount	0.5387	0.5396	0.0009	0.17%
manpopularity26	0.5845	0.5851	0.0006	0.10%
average_cycle13	0.7086	0.7092	0.0006	0.08%
average_cycle26	0.7465	0.7471	0.0006	0.08%
average_cycle39	0.7649	0.7655	0.0006	0.08%
average_cycle52	0.7739	0.7745	0.0006	0.08%
visits13	0.5764	0.5769	0.0004	0.07%
last_day_bought	0.7733	0.7739	0.0006	0.07%
deppopularity26	0.5729	0.5733	0.0004	0.07%
deppopularity39	0.5729	0.5733	0.0004	0.07%
visits26	0.5768	0.5771	0.0003	0.06%
visits39	0.5738	0.5741	0.0003	0.06%
deppopularity52	0.5404	0.5407	0.0003	0.06%
deppopularity13	0.5730	0.5733	0.0003	0.05%
visits52	0.5688	0.5691	0.0003	0.05%
productsbought39	0.6297	0.6300	0.0003	0.05%
distinct_item	0.6250	0.6253	0.0003	0.04%
popularity39	0.7382	0.7385	0.0003	0.04%
popularity26	0.7419	0.7422	0.0003	0.04%
popularity13	0.7462	0.7465	0.0003	0.04%
productsbought26	0.6317	0.6320	0.0002	0.04%
frequenciesdep_decay	0.5591	0.5593	0.0002	0.04%
productsbought52	0.6243	0.6245	0.0002	0.04%
popularity52	0.7347	0.7350	0.0002	0.03%
productsbought13	0.6318	0.6320	0.0002	0.03%
cycle_vs_lastbought	0.7744	0.7746	0.0002	0.02%
most_trialled	0.6871	0.6872	0.0001	0.01%
transactions_withdiscountdep	0.5650	0.5651	0.0000	0.01%
manpopularity39	0.5860	0.5861	0.0000	0.01%
frequencies_decay	0.7086	0.7087	0.0000	0.01%
transactions_withdiscountman	0.5672	0.5673	0.0000	0.01%
manpopularity13	0.5857	0.5858	0.0000	0.01%
HOUSEHOLD_SIZE_DESC	0.5404	0.5404	0.0000	0.00%
frequency13	0.7750	0.7748	-0.0002	-0.03%
frequency39	0.7753	0.7751	-0.0002	-0.03%
frequency52	0.7752	0.7750	-0.0002	-0.03%
frequency26	0.7754	0.7751	-0.0003	-0.04%
HH_COMP_DESC	-	0.5569	-	-
KID_CATEGORY_DESC	-	0.5554	-	-
MARITAL_STATUS_CODE	-	0.5531	-	-
HOMEOWNER_DESC	-	0.5512	-	-

Certain features have over 10% gain in terms of AUC. The features that recorded the biggest gain % were those that had an AUC near the random value of 0.5. The variable popularity_decay (that demonstrated the biggest gain %) is signifying a distinct non-linear relationship with the target variable, also illustrated in figure 3.12.

On the opposite spectrum, the features with highest absolute AUC such as frequencyXX have their AUC slightly worsened after applying binning. This is not unexpected as a customer who has bought an item more times, will have a higher chance to buy it again and any binning

applied flattens any gain that could have been derived from the increasing number of purchases. For example category '9-12' of frequency52 has an average propensity of 18.6653% to buy an item. This representation cannot leverage that fact that frequency52=9 has less propensity than frequency52=12.

Out of 72 features, 66 (or 91.17%) demonstrated some gain after applying binning, 1 out of 72 (or 1.39%) had no change and 4 out of 72 (or 5.56%) had their AUC worsened. The average gain % of AUC across all features is 1.59% or 0.0084 (in simple terms). It can be expressed that binning in most cases does facilitate the uncovering of more information in respect to the target variable and it is further expected that certain machine learning algorithm could benefit from the transformed (with binning) variables.

3.6 Conclusion

This chapter presented findings from the freely available big dataset from dunnhumby.com labelled as "the complete journey" which consists of multiple smaller sets and gave an overview of the basic elements and attributes available for this thesis as well as explained the notions of household and product as they appear in the dataset.

Many supervised machine learning methods rely solely on the features provided to produce a good result, while others can create the features themselves based on the transactional data (like matrix factorization) . The univariate analysis performed in this chapter gave a base for the features that will be inputs in various machine learning algorithms in later chapters of the thesis and also gave insight as to what may be the most predictive features and/or which areas need to be further explored.

The generated features were transformed to capture non-linear relationships through an optimized-binning method and were expressed through multiple time stamps to account for recency, seasonality, cyclicality as well as change of habits through time. The binning of variables was measured to account for an average gain of 1.59% in terms of AUC. 91.17% of the total features demonstrated a gain in the range of [0.000, 0.096] AUC points.

By segmenting the features space into multiple categories based on the customers' attributes and the product hierarchy clearly exhibited that features which capture a direct customer-to-product relationship tend to be the most predictive for explaining behavior in the target week.

Additionally product-based features as well as indirect relationships of the customer with the product (via exploiting the product's heritage through department and brand) may fill the gap of an absent direct relationship. Finally contextual features like time and day of the week do not seem very predictive, but may still add value in particular scenarios in specific algorithms.

4. Meta-modelling to predict top K products

This chapter demonstrates the use of meta-modelling to improve the performance in predicting the top K products for a group of 2,500 loyal customers of a retailer in respect to metrics such Precision at K and AUC using the same underlying data demonstrated in chapter 3. The ensemble model aims to surpass in these metrics any single model involved, any simple ensemble methods and standard benchmarks such as product popularity and customer's frequency of purchase.

4.1 Introduction

The machine learning toolkit has been expanded to many different algorithms and data transformations that in line with the recent advents in both hardware and software has permitted the ability to investigate data from many different angles. For different problems different algorithms may perform better given the underlying structure of the data and other conditions that affect the modelling process, such as the metric to optimize type of input data, volume of data, scarcity and dimensionality. For example linear regression can be best when relationships in the data are linear. The following experiment will make use of Meta modeling (and in this instance stacking) to improve performance in top K products as a means to leverage different machine algorithms and compare results over base models and simple benchmarks.

4.2 Data preparation

4.2.1 Type of features included

The data is the same as demonstrated in chapter 3 and it includes a set of 75 features as illustrated in table 3-2. They include item based features, product-hierarchy features (like department and manufacturer), customer-based features, combinations of all the previous elements, demographics and contextual features (such as time).

4.2.2 Treatment of categorical features

AGE_DESC and INCOME_DESC have been replaced with the average of the groups they represent. For example category '150-174K' of INCOME_DESC was replaced with 162 in the data. The rest of the categorical variables, namely KID_CATEGORY_DESC, HOUSEHOLD_SIZE_DESC, HH_COMP_DESC, HOMEOWNER_DESC, MARITAL STATUS_CODE have been replaced with sequential ids. The process is also known as 'label encoding' [Géron 2017]. Although this technique suffers from the assumption that the categories follow an exact ordinal relationship [Garreta et al. 2013], it has the benefit of not increasing the dimensionality of the dataset. Furthermore some models within the ensemble (such as the tree-based ones) have the capacity to deal with this, via being non-linear in how they process the features.

4.2.3 Treatment of numerical features

Numerical features have been converted using maximum absolute scaling as a means to control outliers. Applying scaling also facilitates convergence. According to [Mika 2010] scaling the data is an important factor in aiding convergence of gradient methods. Some of the algorithms deployed for the experiment are of linear nature and use Stochastic Gradient Descent (SGD) to optimize their weights.

The maximum absolute scaling method essentially rescales the data to be within the range of [-1, 1]. The process requires finding the maximum absolute value of each feature and then dividing each feature with this value [Lee et al. 2017]. This scaling method was preferred over others, because it does not shift the centre of the features, hence the algorithms can still leverage the sparsity of the data. In other words the zero values would remain as zeros after the scaling.

4.2.4 Treatment of missing values

There are 2 types of missing values in the dataset.

The first type of missing data includes values generated due to computational complications when deriving the features. For example deriving the standard deviation of purchasing cycles with only one purchase is not feasible and it is regarded as a missing value. In this context

replacing with a mean or median value (as it is commonly preferred) could create the false representation that this case represents a common purchasing pattern, where in reality these are cases that do not have many past purchases. According to [Lutz 2010] missing values could be replaced with an indicative value outside the range of the values such as -9999. Additionally assigning missing values with indicative and distinctive values (or codes) such as 999 or -9 is further cited by [Ruel et al. 2015] and [Acock 2005]. Eventually such reasoning was preferred, because it could allow certain algorithms (of non-linear nature such as tree-based ones) to isolate these values and treat them accordingly. This is feasible, because they are outside the range of the rest of the values. To avoid having missing values represented with a very large value (like -9999), instead -100 was selected to alleviate convergence difficulties with linear methods. Another reason for assigning a negative value was based on the fact that in most situations this kind of missing value was associated with absence of past purchases. Naturally, absence of past purchases is associated with lower probability to buy the item, hence a negative value could facilitate linear algorithms to capture it.

The other type of missing data refers to values explicitly given as such from the retailer's dataset, for example with the demographic features. These missing values were replaced with -1. This value satisfied the premise of having values outside the range of every other value and followed the same ordinal pattern applied to categorical features due to label encoding as described in section 4.2.2.

4.3 Training, validation and test sets

The train and test data include all customers that shopped in their respective targets weeks (53 and 54 respectively) as in this experiment the focus is on maximizing precision given visit. All the features have been computed for a period of up to 52 weeks prior to the target week with exception of demographic data that is captured during application time. The train data has 12,606,944 records (9,788 x 1,288) and the test data 12,832,068 (9,788 x 1,311. The train data is further split into training data via selecting randomly 80% of the customers who shopped in the target week (53) and all their respective (9,788) products , while the remaining formed the validation data (20%).

4.4 The meta model architecture and performance

The following subsections contain information about the meta model's overall multi-layered architecture.

4.4.1 Meta model definition

The meta model will use as inputs various other models' predictions which are fitted directly on the 80% of the customers' features of the train data (labelled as training data) and made for the 20% of the customers' features of the train data (or just the validation data). For simplicity train data is training and validation data together. Similar predictions are being made for the test data with same parameters as with the train data, but this time using the train data. All these new inputs will be stacked together and form two new data sets for the train (which will have size equal to the validation) and test sets respectively. A model will be used to fit the new train data and make the final predictions for the test data.

This process can be summarized graphically in figure 4.1:

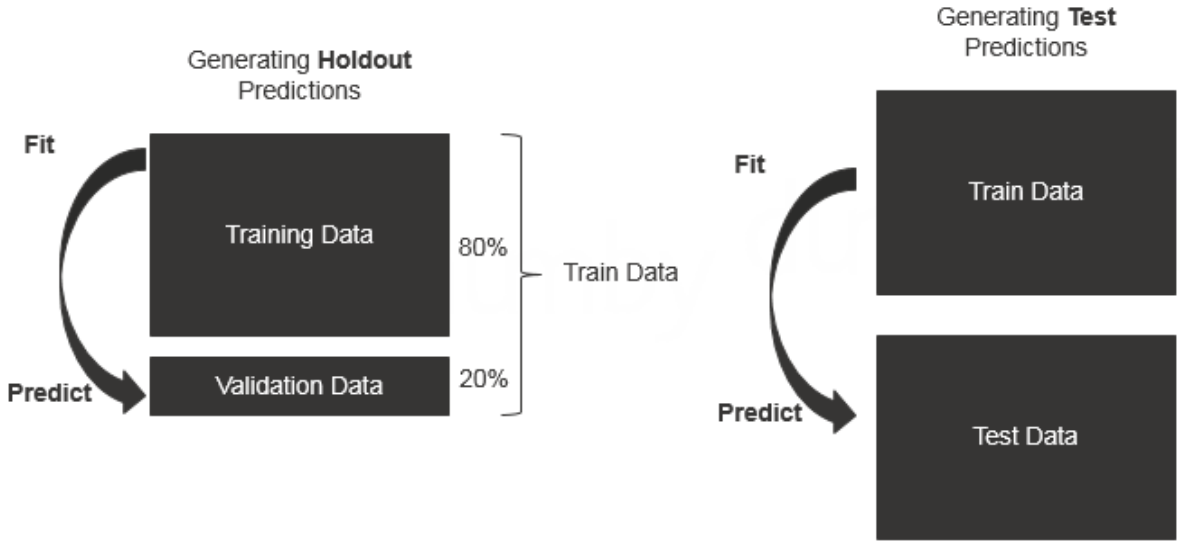

Figure 4.1 : Process for generating train and test predictions in the first layer

Note that this process does not use the k-fold paradigm (although it could) as explained in chapter 2 but a simple random split based on the unique customers that visited in the training week. The reason being that the 20% of train data is actually 2,519,432 rows, enough to build

consistent meta models on top of it. At the same time it reflects a better representation of test data that include as portion of different customers (with the same items considered) than a standard random split.

The aforementioned process is similar to a 1 hidden layer neural network where the input layer is the feature set X and the hidden layer is all the models' predictions (or activation functions) on the 20% of the validation data. Equation 4.2 describes the output of a single hidden unit, given a sample point x_i (from X) and the m (out of M) models from a vector of models S^M in equation 4.1:

$$f_1(x_i, S^M, m) = S_m^M(x_i) \tag{4.1}$$

For example in a linear regression model this S_m^M model will be the coefficients of the model multiplied by the input features x_i plus some constant value. The numbering of 1 in f (e.g f_1) demonstrates that this function takes place between the input data and the first hidden layer. The M in S demonstrates that the total size of the models' vector is M. The advantage of this method is that any model-function can be used as activation function in the meta-model: be the individual model parametric or non-parametric, regressors or classifier and may even include other ensemble type models such as boosted algorithms, bagged models or a simple arithmetic mean.

The output layer is the prediction that comes out by combining all other f_1 models' predictions (or neurons) through another model L and leads in this case to probabilistic output similar in concept to a Softmax output layer. The function that connects the predictions with the target variable through another (Meta) model is summarized by equation 4.2:

$$f_2(x_i, L, S^M) = L\left(f_1(x_i, S_1^M), f_1(x_i, S_2^M), \dots., f_1\left(x_i, S_M^M\right)\right) \tag{4.2}$$

Where L is the Meta model used to combine all other previous models' predictions given their activations-models in S.

4.4.2 Meta model base layer and performance

The Meta model's performance in the based layer is explained in tandem with the model parameters chosen to maximize it and the metrics selected to capture it.

4.4.2.1 Model parameters

In this experiment the first level consists of 10 different machine learning models, regressors or classifiers, all trained on the input features while outputting predictions with dimensionality equal to one (which connotes probability to buy in classifiers and the regression continuous output for regressors). The specific model selection was chosen in order to cover at least one representative of each one of the common algorithmic families, however some, mostly KNN-based have been excluded due to being too inefficient.

Each model has been tuned mildly around the default hyper parameters using the 20% (validation set) to determine the best set of parameters. This process was not time consuming as the basic logic is the models don't have to be heavily tuned, in similar way where a Random Forest is consisted of weaker trees to reduce variance and improve its ability to generalize to unseen data. Another reason towards that end is the fact that the output predictions will be used for the Meta model, therefore are required to be able to generalize rather than being dataset-specific. Table 4-1 demonstrates the models used in the experiment's first layer and their main parameters:

Table 4-1 : Single models involved in the ensemble and their hyper parameters

Models in first layer	Parameters
Ridge regression	C =0.001
Neural net classifier	C=0.00001, learn_rate=0.009,h1,h2=(30,20),connection=relu, out=Softmax
Neural Net regression	C=0.00001, learn_rate=0.009,h1,h2=(30,20),connection=relu, out=Linear
Naïve Bayes classifier	Shrinkage=0.1
Logit (L2 regul)	maxim Iterations=100, C=1.0
Logit (L1 regul)	maxim Iterations=100, C=1.0
SVM (Linear kernel)	maxim Iterations=100, C=1.0
LibFm classifier	maxim Iterations=100,C=0.0001, init_values=0.05,learn_rate=0.01,Lfeatures=4
Gradient boosted Random Forest regressor	estimators=300,max_depth=12,max_features=0.3,min_leaf=12.0,shrinkage=0.04, row subsample=0.95
Gradient boosted Random Forest classifier	estimators=300,max_depth=12,max_features=0.3,min_leaf=12.0,shrinkage=0.04, row subsample=0.95

Linear models have been preferred because they were faster to run and easier to converge.

Ultimately all experiments have been run on a Linux server with 32 cores and 256 GB of RAM. One of the advantages of this methodology is that all the aforementioned models can be run in parallel and when all of them are completed, then the next layer can be initiated making full leverage of all resources available. All software used was designed specifically for this experiment using the Java programming language. The software used comes from the [StackNet 2017] Meta Modelling Framework and library which is explained in chapter 6.

4.4.2.2 Metrics

Table 4.2 demonstrates the performance of the aforementioned models in AUC in both validation and test data (for consistency) and the precisions @5, @10 and @20 for the test set with the same models:

Table 4-2 : Performance of single models in UC and Precision@K

Models in first layer	AUC TRAIN	AUC TEST	Precision@5	Precision@10	Precision@20
Ridge regression	0.86036	0.85466	21.59%	17.20%	13.38%
Neural net classifier	0.86277	0.85105	27.39%	21.04%	15.57%
Neural net regression	0.84320	0.83338	**27.73%**	21.49%	15.80%
Naïve Bayes classifier	0.83668	0.81483	8.32%	8.23%	7.73%
Logit (L2 regularization)	0.85099	0.83982	26.84%	20.61%	15.31%
Logit (L1 regularization)	0.84274	0.82045	15.60%	11.23%	9.01%
SVM (Linear kernel)	0.81874	0.80474	14.21%	11.23%	9.17%
LibFm classifier	0.84593	0.83504	18.41%	14.88%	11.87%
Gradient boosted Random Forest regressor	0.86277	0.84803	26.38%	21.04%	15.16%
Gradient boosted Random Forest classifier	0.87761	**0.86515**	27.25%	**21.74%**	**15.96%**

It is not surprising that gradient boosted trees performed the best in terms of AUC in both training and test as there were significant non-linear relationships as presented in chapter 3. The performance of linear models is also commendable. In chapter 3, frequency of purchase was demonstrated to be one of the most important predictors in determining next purchases. At the same time there were other features (like popularity decay) which were exhibiting nonlinear relationships with the target variable. The Neural Network with linear output can (theoretically) exploit these two types of features and demonstrates a precision @5 higher than tree-based models. It seems that the Gradient boosted Random Forest classifier regains the lead in precision @10 and @20 making use of the apparent nonlinear relationships present in the data to maximize its predictive accuracy.

The AUC is particularly high given that it may take values between 0.5[5] and 1. The High AUC occurs because the datasets include all retention and non-retention combinations of customer-item pairs and therefore it is easy to discriminate the former from the latter. Additionally in an FMCG environment having previous experience in buying products greatly augments the chance of such pairs occurring again in the future making the overall discrimination of any model much stronger given it includes these kind of relational features.

Ultimately the performance of the models in both training and test seem to vary significantly and this may be due to the time lag between train and test which causes some of the models to lose their ability to generalize as efficiently in future data (overfitting). Other models (particularly the L1 ones) may have been underperforming because some of the hyper parameters force the models to remain more in the surface instead of searching and leveraging deeper relationships (underfitting). Irrespective of the reasons that may lead to such gaps, assuming the link between training-validation and train-test is not compromised (so that an overfitted or underfitted model in the validation data is understood as such in the test data too), the Meta model can make use of such information to improve results on the test data.

4.4.3 Meta model output layer performance

Similar to the base layer models, performance for the output layer is explained in tandem with the model parameters chosen to maximize it, the benchmarks to compare against and the overall performance based on the pre-defined metrics.

4.4.3.1 Model parameters

The two set of predictions (one for the validation data and one for the test data) became inputs to a random forest classifier Meta model that had two outputs, the probability to buy an item next week and the probability not to buy next week. The reason this algorithm was chosen was because it is known to generalize well to unseen data (due to bagging), it is nonlinear and previous experiment showed similar algorithms to dominate performance wise.

[5] That would entail random prediction in connection with the target variable

The model had its hyper parameters tuned through a simple 50-50 random validation step (again) based on unique customers. The final hyper parameters include 3,000 estimators, maximum tree depth of 6, feature subsample of 0.3, leaf size of l00, row subsample of 0.9 and uses the Entropy metric (or information Gain) as the main criterion to determine the split. The first thing to note is the bigger ensemble size than before, product of the much smaller data (4 times smaller than before) that allows to run more bags faster and at the same time provides an extra layer against over fitting. All other parameters are constrained (or more reserved) compared to the initial models in order to maximize performance. For example performance was decreasing with higher maximum depth because of the inability of the model to remain general hence not overfitting its training data. Similarly low feature subsample ensures there is not over-reliance in specific input feature (or first layer model), while high minimum leaf boosts significance of the results in each node via increasing its size.

Ultimately the final Meta model was built under similar principles of the previous models, having its hyper parameters tuned through an equivalent validation procedure with focus on avoiding overfitting, while leveraging most of the benefits arisen from the individual model-inputs of the previous layer.

4.4.3.2 Benchmarks

In order to compare the performance of the meta model a number of different baselines have been used. The first baseline is the best model's performance from the previous layer for the test data (which was Gradient Boosted Random Forest classifier for AUC, precision@10 and precision@20 and Neural Network Regression for Precision@5). Comparison with this baseline basically demonstrates whether Meta modelling actually yields better results than any one of the previous base models. Another baseline, arguably the most basic one, is the popularity of the products purchased in the last 26 weeks for the given customer population. Popularity as explained in Chapter 3 is defined as the number of baskets a product has been included in the customers baskets. This metric facilitate gauging how better the model is to discriminate irrespective of prior personalised knowledge. The personalised version of the previous metric is the frequency of purchase of the item per customer, given his/her visits in the last 26 weeks. Equivalently this metric measures how many times a customer has included an item in the basket, in the last 26 weeks (and like popularity ignores quantity).

Apart from these simple metrics, some ensemble metrics will be used as well. A simple (equally weighted) average of all input 10 models will be used to assess whether combining the results yields any uplift in the aforementioned metrics. However this method is still biased in a way because the output of some models is now always a probability since many regressors have been used in previous phases beforehand. To counter this effect a simple Ranking average will be used to combine all the models. This method is basically a simple average after transforming all scores to their rank value based on their order. This ensures that even models with higher means and variances (that may even exceed the bounds of 0 and 1) can now be blended in a fair manner along with the rest of the models. Also precision and AUC are both affected only by the ranking of the score, therefore maintaining a probabilistic output is not really necessary.

These five metrics will be used to compare performance against the performance of the Random Forest Classifier Meta model in both actual and proportional manner.

4.4.3.3 Graphical representation of the model

The following graph 4.2 shows a graphical representation of the whole in order to make apparent the similarity of this modelling procedure with the single layer neural network.

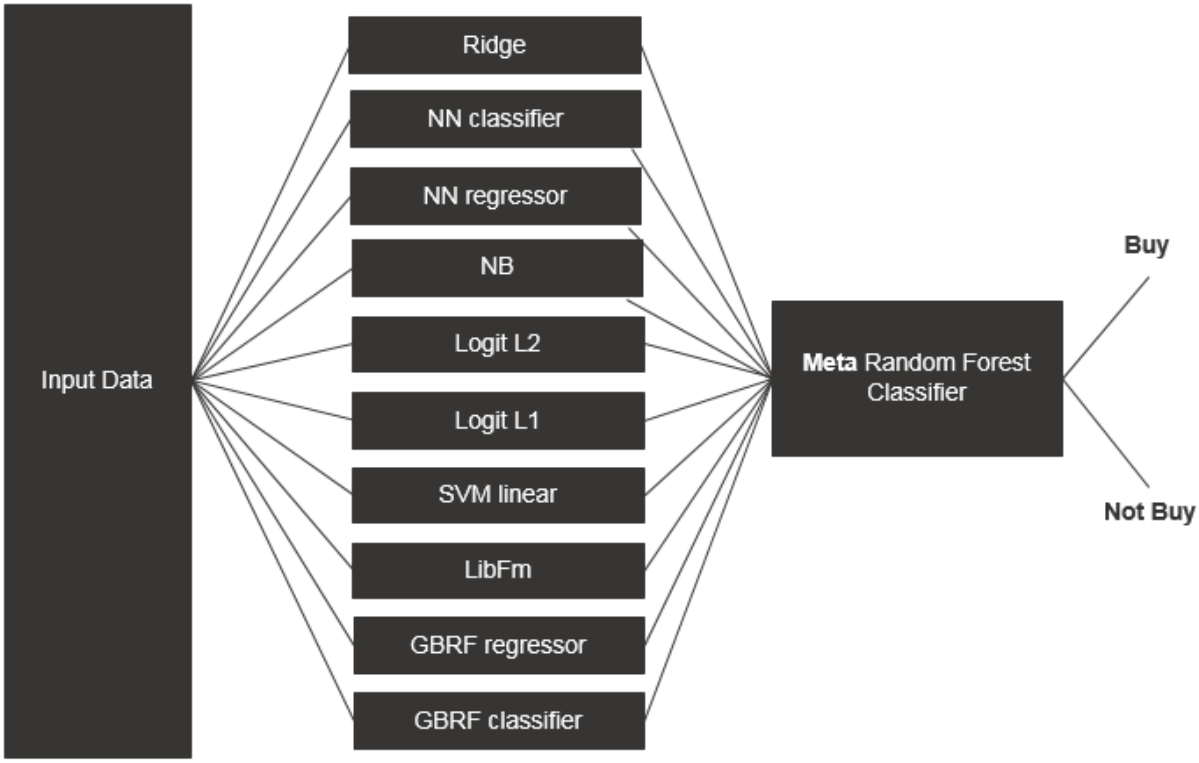

Figure 4.2 : Illustration of the stacking (Meta) model

4.4.3.4 Results

Table 4.3 shows the results of all benchmarks and the meta model in actual results:

Table 4-3 : Results of benchmarks, single best models and Meta model in AUC, Precision@K

MODELS	AUC_TEST	Precision@5	Precision@10	Precision@20
Popularity 26 weeks	0.75548	12.53%	8.62%	6.20%
Frequency 26 weeks	0.75636	27.54%	20.72%	15.01%
Average of all 10 models	0.85374	26.90%	21.24%	15.80%
Rank Average of all 10 models	0.85708	21.98%	17.41%	13.69%
GBRF (AUC, P@10,P@20) or NNreg(P@5)	0.86515	27.73%	21.74%	15.96%
Meta Random Forest Classifier	**0.86759**	**28.17%**	**22.16%**	**16.20%**

Table 4-4 portrays the proportional difference/deterioration of each model in comparison to the Meta model:

Table 4-4 : results with proportional differences to the Meta model

MODELS	AUC_TEST	Precision@5	Precision@10	Precision@20
Popularity 26 weeks	-12.92%	-55.50%	-61.11%	-61.71%
Frequency 26 weeks	-12.82%	-2.22%	-6.51%	-7.33%
Average of all 10 models	-1.60%	-4.50%	-4.13%	-2.47%
Rank Average of all 10 models	-1.21%	-21.95%	-21.43%	-15.50%
GBRF (AUC, P@10,P@20) or NNreg(P@5)	-0.28%	-1.57%	-1.89%	-1.48%
Meta Random Forest Classifier	0.00%	0.00%	0.00%	0.00%

By all metrics considered the meta model Random Forest classifier outperforms all of the benchmarks and simple ensemble methods. The popularity and frequency benchmarks have performed relatively poorly in terms of AUC in comparison to the rest of the methods and benchmarks, however the latter one has done commendably well in precision, due to the fact that it is a personalized metric. Interestingly the average of all previous models performs relatively well in all precision metrics but worse than ranking average in terms of AUC.

The Gradient Boosted Random Forest from the first layer (a base model) seems to outperform all other simple benchmarks and simple ensemble methods (although it is also a part of it). Given the time lag and the significant difference in performance between train and test among the 10 models, a simple (or even a ranking) average is suboptimal.

Compared to a simple average, the Meta model leverages the strengths and weaknesses of each model as well as the shared information between them and is able to outscore every other benchmark (including the GBRF) by at least 1.4% in all precision metrics and a small but potentially valuable 0.28% in AUC. Especially at precision@10 (which seems to be the metric the retailer most favours) the difference reaches close to 2%. The proportional difference against the popularity metric exceeds 55% for all precision metrics and is roughly 13% for AUC.

Subsequently the Meta modelling methodology, given the time lag and the commendable variance among the based models (that led simple ensemble methods to fail) has managed to outperform any other single model or simple benchmark and it should be noted that even some of the input-base models were product of ensembling themselves (like the GBRF) . This methodology could have worked with more models (possibly improving on the results over the best single model even further) or even less base models for a simpler yet faster solution. It should also be noted that such methodology does not need to be strictly associated with this particular problem of improving results for the *top k* products, but could be refactored possible with different holdout methods to be utilized in other problems of typical classification of regression.

4.4.3.5 Estimating financial impact of stacking model

The potential uplift of the stacking approach in terms of revenue (for the grocery market) could be calculated under certain assumptions. The dataset contains 276,484 unique baskets spanned over 102 weeks. The average price of the selling item based on the transactional data is approximately 3.10$ for the same period. Every basket contains 9.39 products on average. Since the average number of products in a basket is near 10, it can be assumed that uplift in precision at 10 is the most appropriate to base any calculations regarding financial impact.

It can be further assumed that the best single model for precision at 10 could be used as the baseline to measure the uplift. The uplift in precision at 10 between the stacking model and the best performing single model (GBM) is 0.42% (or 22.16% minus 21.74%). Assuming this uplift is incremental and the GBM model is already providing recommendations, on every 238 (or 100 divided by 0.42) items sold, there is 1 extra item that would come from the stacking model.

The dataset contains 2,595,732 purchased items from 2,500 households (for the same period of 102 weeks). It can be roughly estimated that 10,906 extra times (or 2,595,732 divided by 238) could be sold over the same period for the same number of households. Since the average price for any item was calculated to be approximately 3.10$, the additional revenue subject to these extra items would be 33,808.6$ (or 3.10$ multiplied by 10,906). The estimated revenue across a more mature market with 10 million shoppers (that demonstrate similar purchasing behaviour) would be approximately 13,500,000$ (if 2,500 shoppers account for 33,808.6$).

Although this approach of estimating the financial benefit assumes that all uplift in offline precision of the stacking model versus the best performing single model is incremental (which is unlikely to be the case), yet it does not account for the long term benefits of providing better recommendations to customers and the effect they may have to the overall customer satisfaction and loyalty. It should be noted that according to [Anderson et al. 2003] loyal customers may be worth up to 10 times as much as the average customer (for a given seller) over their buying lifetimes.

4.5 Conclusion

A noble goal of the recommendation science in an FMCG environment is to improve the ranking of the recommendation given, especially to loyal customers that are predisposed to buy more. This objective can naturally be optimised via creating a powerful set of features that best describe the customer to item relationship as well as selecting suitable machine learning algorithms to fit on this data and leverage the linear and nonlinear relationships inherent within these features. To further boost results a Meta modelling methodology can be considered which combines various machine learning algorithms and uses their outputs predictions as inputs to a new higher level (meta) model.

Similarly, with a single hidden layer neural network many different models have been fitted in parallel on a subset of the retailers training data for a given week and made predictions (outputs) for another subset as well as the test data that occur in a future week. These predictions are then *stacked* together forming new modelling datasets and are used to build a new Random Forest (Meta) model outputting the probability to buy or not to buy a product resembling the typical softmax function of neural network classifier.

Compared to a number of different simple benchmarks (like frequency of purchase and item popularity) and ensembling methodologies (like average and rank average), the Meta model has performed the best in all metrics considered, scoring significantly better against most of the baselines; outperforming all and significantly outperforming most

This methodology could be further improved via adding more base level models (or neurons) at the cost of more computational time or with less models to save time at the cost of some loss in terms of accuracy. Particularly where the additional base models capture a new aspect of the problem, such methodology, with some changes in the validation framework and the input futures and models could be extended to different optimization problems of classification or regression.

5. Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment

This chapter details a hybrid recommendation methodology to improve the accuracy of predictions regarding which products the customers of a retailer will buy again in the future after having received and redeemed an offer for them. This methodology is applied within an irregular testing environment where the models needs to be built with a subset of customers and offers and validated in an environment of different customers and offers (as well as different time periods)..

5.1 Introduction

The focus of this chapter is to demonstrate a hybrid methodology containing a content-based approach and a collaborative filtering approach to improve the accuracy of predictions for which products the customers of a retailer will buy again in the future, assuming they have already received and redeemed an offer for them. The accuracy of this methodology is validated based on an irregular testing environment. Irregular in this context is defined by the fact that the recommendation models need to be trained on a subset of customers and offered products and applied to a different set of customers and offered products (as well as different time periods). In such an environment, finding a suitable cross validation methodology to train the models, create and select features as well as tune the models' hyper parameters was proven to be critical for boosting the results in the test data. The cross validation methodology and the hybrid model is based on the co-winning solution with Gert Jacobusse of the "Acquire valued shoppers challenge" predictive modelling competition held on the kaggle.com platform.

5.2 Problem to Solve

There were 310,000 customers that were sent and redeemed a coupon for a product. The simplified statement of the problem is:

Will the customer buy the redeemed product again in the future?

5.3 The data

In this challenge the analyst was given 2 datasets of:

- 160,000 customers as training set and
- 150,000 customers as a test set

The training set had two additional features not contained in the test data. The first is the number of times the customer bought the offered product after redeeming it once in the past and the second is just a binary indicator that specifies whether the customer bought the offered product at least once, past the first coupon's redemption. Additional data fields included the value of each transaction, the department where the product was conceptually taken from, the quantity of the purchase as well as the product measure and size. All these fields are common to the typical retailer datasets.

In addition there was a source dataset containing a subset of the customers' transactions for the training and test sets commencing from sometime in the past until the day the customer redeemed the offered coupon. The redemption date in the test data is always in the future compared to the training data.

There is no specific field dedicated to represent the product (such as a product id as it is commonly referred to), however in the context of this experiment a product can be defined as the unique combination of three basic elements of the product hierarchy that are present in the data, namely:

1. The brand the product belongs to
2. The category
3. The company that produced it

There were 37 different offers 23 of which mostly appear in the training set (and 24 in total) and the remaining 13 mostly appear in the test set (and 29 in total). This uneven distribution of offers between training and test data is presented in figure 5.1:

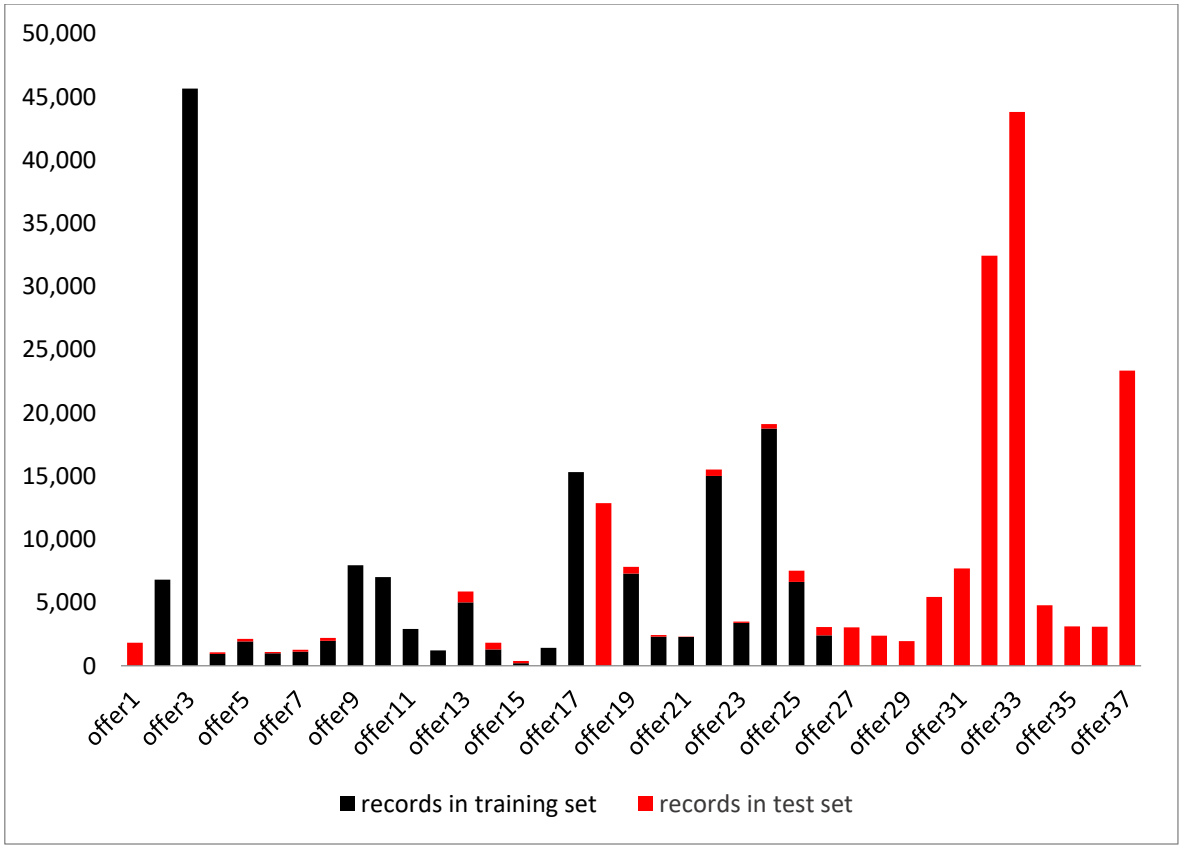

Figure 5.1: Uneven distribution of offers between training and test sets

In summary, the training of models would have to be made on different customers and different offers (as well as different time periods) than the ones available in the test data which constitutes an irregular (and difficult) environment to create accurate predictions for.

5.4 Objective to optimize

The objective function to be maximized was the AUC. The ROC (Receiver Operator Characteristics) curve was first introduced by [Reen and Swets 1966] and it portrays the confusion matrix of sensitivity and 1-specificity for each possible cut-off of the prediction's array. This metric was explained previously in 2.2.2.3.

For this particular problem, sensitivity is the percentage of those that did buy the offered product past the offered date and the model did predict they'll do so. Similarly specificity can

be defined (in this scenario) as the percentage of these customers that did not buy the offered product after they were sent the coupon and were correctly classified as such by the model.

5.5 Cross Validation Strategy

The cross validation strategy was driven from the fact that there were (almost entirely) different offers in the test data than in the training data as demonstrated in figure 5.1. Three different validation methodologies were considered as a means to internally gauge performance and optimize the modelling parameters as well as derive and select features to maximize performance in the test data.

The first method was a random K-fold cross validation (as defined in 2.2.1) stratified based on the offers. The stratification ensures that every offer is represented equally (as a proportion) to the training and validation data. In other words if offer z makes up for 10% of the total samples, K is 5 and sample size is 160,000, then in the first fold 128,000 will be used for training (or 160,000 x 80%) and offer z will account for 12,800 of the training cases (or 128,000 x 10%). At the same time the validation data for that same fold will have the remaining cases which are 32,000 and 10% (or 3,200) of them will be attributed to offer z. This method was repeated with multiple K in the range of [5, 15].

The second cross validation methodology was formulated to always build a model with N-1 offers and use the n_{th} to test it. This process is repeated N times until all offers are scored and the average AUC per offer is retrieved. The process is also illustrated bellow.

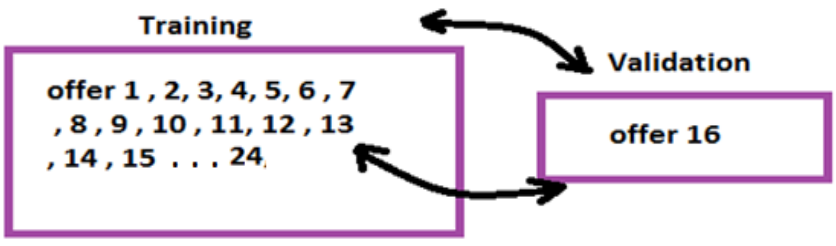

Figure 5.2: N-Offer Cross validation procedure

In other words what is optimized is the average AUC per offer (denoted as AUC_{per_offer} for simplicity) for each offer in the training set, also expressed in equation 5.1:

$$\mathrm{AUC}_{\mathrm{per\_offer}}(\widehat{\mathrm{Y}}, \mathrm{Y}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{n}=1}^{\mathrm{N}} \mathrm{AUC}(\widehat{\mathrm{Y_n}}, \mathrm{Y_n}) \quad (5.1)$$

, where $\widehat{\mathrm{Y}}$ is the prediction vector for all samples in the training dataset and Y the actual labels. $\widehat{\mathrm{Y_n}}$, $\mathrm{Y_n}$ refer to a subset of these predictions, labels, limited to the samples belonging to $\mathrm{offer_n}$. These predictions ($\widehat{\mathrm{Y_n}}$) were generated using all other offers' samples as inputs to a model. Assuming a feature set X and offer n>1, to generate the feature set $\mathrm{X_m}$ used to build a model that predicts $\widehat{\mathrm{Y_n}}$, all samples attributed to any of the N offers are concatenated vertically apart from those samples belonging to n as $\mathrm{X_m} = [\mathrm{X_1}| \dots |\mathrm{X_{n-1}}|\mathrm{X_{n+1}}| \dots |\mathrm{X_N}]$. The same applies to generate $\mathrm{Y_m}$ from Y. Then an estimator (or model) is used to train on the pair of $\{\mathrm{X_m}, \mathrm{Y_m}\}$ to produce estimates $\widehat{\mathrm{Y_n}}$, based on $\mathrm{X_n}$as input. This reasoning for making predictions using N-1 offers can be defined as *leave-one-offer-out* for future reference.

Although this cross validation approach is sensible because it utilises into its schema the fact that the offers in the test data are generally unknown, however it does not essentially optimizes for the actual (overall) AUC. Optimizing for the overall AUC connotes that the prediction needs to discriminate well against any offer (or against any sample) and not just within the offer. This type of AUC can be denoted as $AUC_{overall}$ measured after all $\widehat{Y_n}, Y_n$ from 5.1 are concatenated vertically.

$$\mathrm{AUC}_{\mathrm{overal}}(\widehat{\mathrm{Y}}, \mathrm{Y}) = \mathrm{AUC}([\widehat{\mathrm{Y}}_1|\widehat{\mathrm{Y}}_2| \dots |\widehat{\mathrm{Y}}_\mathrm{N}], [\mathrm{Y}_1|\mathrm{Y}_2| \dots |\mathrm{Y}_\mathrm{N}]) \quad (5.2)$$

Based on the known offers, the average propensity was differing significantly ranging from 7.4% for offer9 to 50.7% for offer2. Table 5-1 demonstrates the sorted average propensity to buy each offer based on the training data.

Table 5-1: Sorted average propensity to buy per offer

Offer	Propensity to buy
offer2	0.507
offer24	0.434
offer17	0.424
offer25	0.378
offer26	0.341
offer22	0.321
offer23	0.305
offer19	0.285
offer15	0.230
offer5	0.214
offer4	0.210
offer8	0.199
offer3	0.196
offer6	0.194
offer16	0.186
offer21	0.177
offer7	0.166
offer20	0.166
offer14	0.161
offer13	0.143
offer10	0.106
offer12	0.106
offer11	0.085
offer9	0.074

Optimizing only based on the current schema and given the differences in propensity levels per offer, could generate a model where a sample for a given offer is not comparable to all other samples from all other offers. To further demonstrate this problem, consider the following two tables (5-2 and 5-3). The first table (5-2) displays the sorted (per target and score) predictions and actual values for a small random sample of 10 cases for a random offer.

Table 5-2: Sorted predictions and actual values for a random sample and a given offer x

Prediction for $offer_x$	Actual for $offer_x$
0.91	1
0.56	1
0.77	1
0.44	1
0.33	1

0.46	0
0.34	0
0.23	0
0.2	0
0.05	0

The AUC given this set of predictions and target information is **0.88** .The second table (5-3) also displays the sorted (per target and score) predictions and actual values for a small random sample of 10 cases for a different random offer.

Table 5-3: Sorted predictions and actual values for a random sample and a given offer y

Prediction for $offer_y$	Actual for $offer_y$
0.65	1
0.6	1
0.59	1
0.57	1
0.55	1
0.58	0
0.56	0
0.53	0
0.52	0
0.51	0

The AUC for this sample is again 0.88, however the range of values is smaller here as all scores are between 0.51 and 0.65. Table 5-4 shows the (vertical) concatenation of tables 5-2 and 5-3, denoted as [$offers_x$ | $offer_y$].

Table 5-4: Vertical merge of tables 5-2 and 5-3

Prediction for [$offers_x$ | $offer_y$]	Actual for [$offers_x$ | $offer_y$]
0.91	1
0.56	1
0.77	1
0.44	1
0.33	1
0.46	0
0.34	0
0.23	0
0.2	0
0.05	0

0.65	1
0.6	1
0.59	1
0.57	1
0.55	1
0.58	0
0.56	0
0.53	0
0.52	0
0.51	0

In this example, the AUC after the merge of the samples is again 0.825, which connotes a 6.25% drop over the individual AUCs. This occurs because the negative samples in table 5-3 have on average higher score than negative samples of 5-2 and lean more towards the positive values of 5-2 superseding more of them.

Coalescing all this information, namely prime knowledge of the distribution of offers in the test data, the diversity of known propensities per offer in the train data plus the potential loss in overall AUC if predictions of different offers are not intersecting optimally, led to the formulation of a third cross validation approach.

The thrid cross validation strategy used the same approach with the second of leave-one-offer-out when estimating AUC, but instead of only monitoring the average AUC per offer (or AUC_{per_offer}), the predictions and labels for all offers were concatenated vertically (similarly as tables 5-2 and 5-3 were used to form table 5-4) and the overall AUC (or $AUC_{overall}$) was computed based on that concatenated frame. The final metric to maximize (denoted as AUC_{final}) was the average of the two AUCs as presented in the following equation.

$$\mathrm{AUC_{final}}(\mathrm{AUC_{overall}}, \mathrm{AUC_{per_{offer}}}) = \frac{\mathrm{AUC_{overall}}}{2} + \frac{\mathrm{AUC_{per\_offer}}}{2} \quad (5.3)$$

The features used to test which cross validation method performs better were derived from the transactional history and included past counts of purchases from the same category, brand or company and combinations of them. The features used connote a subset of those used in 5.6.1 and the focus was not to achieve the highest accuracy but compare which validation method performs better before significant amount of time is invested in finding the best features,

algorithms and overall modelling parameters. The supervised model used to validate the approaches was Logistic regression (defined in 2.2.3), modelled to predict the probability of whether a customer will repeat or not. The only parameter that was changing subject to performance throughout the different validation schemas was the L2 regularization parameter (denoted as c). The results of the different cross validation methodologies are listed in the following table.

Table 5-5: Results on AUC for all validation schemas

Validation schemas	AUC_CV	AUC_TEST
Stratified K-Fold (K=5)	0.683	0.579
Stratified K-Fold (K=10)	0.699	0.578
Stratified K-Fold (K=15)	0.712	0.576
Leave-one-offer validation	0.655	0.588
Leave-one-offer + concatenation	0.632	0.601

The stratified K-Fold methods gave very promising internal results, however they underperformed in the test data compared to the other methods. The Leave-one-offer out method experienced a drop in the local results, but substantial improvement in the test. Ultimately the Leave-one-offer plus concatenation had the weakest local results but the best performance in the test data. Due to the substantial difference between the train and the test data, the more common cross validation methods were not able to generalize well, whereas the methods that included the Leave-one-offer approach were able to achieve better results. Based on these outcomes, the latter validation schema, namely Leave-one-offer plus concatenation (or AUC_{final}) was selected to aid the modelling process.

5.6 The Strategies

A fundamental difficulty in any FMCG's predictions is the fact that they can be quite different in nature and therefore it becomes quite challenging to create a holistic model (that could potentially be applicable to any offer). On the other hand, creating a product-specific model cannot generalize very effectively (especially to cold-start problems where the customer has never bought the product before), plus it has to be built on limited data.

Based on the aforementioned challenges two main *strategies* were formed to better generalize in the test data as well as deal with the cold-start problem:

1) **Content-based**: Make predictions scrutinizing the relationship of the customer with a product and any of its sub-elements (of brand, category and company).
2) **Collaborative filtering**: Find those customers that although they do not have a direct relationship with the product, still look like other customers who have such relationship.

5.5.1 Content based strategy 1: Exploit relationship of customer with product

The following sections contain information regarding the reasoning for employing strategy 1, the features generated under it, the data pre-processing steps that were utilized to improve performance, the selected algorithms and the actual performance in respect to the AUC metric in the test data.

5.5.1.1 Assumption

The assumption of the first strategy was that a customer who has bought from the same, company, brand and category will have higher chance to buy the products once offered. This model generalizes well against any item and any product because it maps only the relationship of the customer with the item by means of how many times the relationship occurred in the past.

5.5.1.2 Feature engineering

The generated features map the timeline of the relationship a customer had with a product prior to the sending of the coupon. Consider the following figure (5.3) where a customer could have bought a product in certain past occasions:

Figure 5.3 : Timeline of customer with coupon redemption

A customer could have bought a product multiple times in the past at different intervals. At some point in time he/she received an offer (in this case a coupon) to buy this product and he/she responded positively via buying the product. The algorithm is tasked to predict whether the product will be bought again by the same customer in the future. This approach assumes that there is a pre-existing relationship of the customer buying the product or any of the hierarchy groups (such as brand, category or company) in the past. Therefore the generated features try to gauge how strong that relationship is as measured by different time intervals such as last 30, 60, 90, 120, 150, 180, 360 or more days.

For example consider feature category_30 which shows how many times a customer has bought from the same category (as the offered product) in the last 30 days prior to the offer. The assumption is that if a customer has bought from the same category multiple times in the past, there is higher chance that he/she will become a repeater after receiving and redeeming an offer. Apart from customer-to-product related features, customer-based only and product-based only features were generated. Customer-based only features refer to attributes that describe the customers' preferences to the store such as average spending and number of visits over fixed periods of time. Product-based only features include general category, brand and company popularity for the same time intervals as well as average price and spending. The final list of features for strategy one is displayed bellow in greater detail in table 5.6:

Table 5.6: List features' descriptions derived for strategy one

Features	Description of customers' features
category_30	Times bought the same category in last 30 days
category_60	Times bought the same category in last 30 to 60 days
category_120	Times bought the same category in last 90 to 120 days
category_180	Times bought the same category in last 120 to 180 days
category_360	Times bought the same category in last 180 to 360 days
category_over360	Times bought the same category in more than 360 days
brand_120	Times bought the same brand in last 90 to 120 days
brand_180	Times bought the same brand in last 120 to 180 days
brand_360	Times bought the same brand in last 180 to 360 days
brand_over360	Times bought the same brand in more than 360 days
company_30	Times bought the same company in last 30 days

company_60	Times bought the same company in last 30 to 60 days
company_90	Times bought the same company in last 60 to 90 days
company_120	Times bought the same company in last 90 to 120 days
company_180	Times bought the same company in last 120 to 180 days
company_360	Times bought the same company in last 180 to 360 days
company_over360	Times bought the same company in more than 360 days
category_brand_30	Times bought the same category&brand in last 30 days
category_brand_60	Times bought the same category&brand in last 30 to 60 days
category_brand_90	Times bought the same category&brand in last 60 to 90 days
category_company_30	Times bought the same category&company in last 30 days
category_company_60	Times bought the same category&company in last 30 to 60 days
brand_company_30	Times bought the same brand&company in last 30 days
brand_company_60	Times bought the same brand&company in last 30 to 60 days
brand_company_90	Times bought the same brand&company in last 60 to 90 days
brand_company_120	Times bought the same brand&company in last 90 to 120 days
brand_company_180	Times bought the same brand&company in last 120 to 180 days
brand_company_360	Times bought the same brand&company in last 180 to 360 days
brand_company_over360	Times bought the same brand&company in more than 360 days
category_brand_company_30	Times bought the same category&brand&company in last 30 days
category_brand_company_60	Times bought the same category&brand&company in last 30 to 60 days
category_brand_company_90	Times bought the same category&brand&company in last 60 to 90 days
category_brand_company_120	Times bought the same category&brand&company in last 90-120 days
category_brand_company_180	Times bought the same category&brand&company in last 120 -180 days
category_brand_company_360	Times bought the same category&brand&company in last 180-360 days
distinct_bought_company	Number of dIstinct companies has bought
distinct_bought_category_brand	Number of distinct categories and brands' combos has bought
distinct_bought_category_company	Number of distinct categories and companies' combos has bought
distinct_bought_brand_company	Number of distinct brand and companies' combo has bought
transaction purchase count_30	proportion of total transactions that occurred 30 days
transaction purchase count_60	proportion of total transactions that occurred 30 to 60 days
transaction purchase count_90	proportion of total transactions that occurred 60 to 90 days
transaction purchase count_120	proportion of total transactions that occurred 90 to 120 days
transaction purchase count_180	proportion of total transactions that occurred 120 to 180 days
amount_paid30	Average amount paid by (per transaction) in the last 30 days
amount_paid60	Average amount paid by (per transaction) in the last 30 to 60 days
amount_paid90	Average amount paid by (per transaction) in the last 60 to 90 days
amount_paid120	Average amount paid by (per transaction) in the last 90 to 120 days
amount_paid180	Average amount paid by (per transaction) in the last 120 to 180 days
Interaction: brand_30_transaction purchase count_120	Interaction of the times the brand was bought in the last 30 days with the proportion of the total transactions in the last 120 days
Interaction: brand_company_30 _distinct_bought_category	Interaction of the times the brand and company combo was bought in the last 30 days with the distinct number of different categories

All possible pairwise interactions of features were considered and their contributions in AUC was gauged based on the cross validation procedure as defined in 5.5 after adding each interaction one-by-one in a forward manner. Only 2 interactions were found to improve AUC_{final} and are listed at the bottom of table 5.6.

5.5.1.3 Pre-processing

To ensure unbiased predictions given the large amount of personal transactions and the potential threat of outliers, extreme observations both by means of quantity (>25) and amount spend (>£30) where excluded from the analysis. Additionally missing values where simply replaced by -1 to ensure that there is no overlap with the mostly-positive values that most of the features boasted. The logic for the treatment of missing values is also explained in 4.2.4.

5.5.1.4 Modelling

The target variable was the number of times the customer bought a recommended product past the offered date and not the binary indicator. Because of the nature of the different offers training on the actual counts provided an extra layer of confidence for these offers that are bought multiple times. Adding the number of times the customer bought a recommended product as samples' weight and treating the problem as binary did not yield better results than regarding it as a regression task.

The preferred model was Ridge Regression (which is Least Squares regression with L2 regularization) trained on the number of times the customers bought the product past the offered date with a high alpha of 49. This parameter was selected based on the same cross validation procedure explained in section 5.5.

5.5.1.5 Performance

The ridge model did quite well in its higher predicted scores (i.e. commonly items the customer has bought before) in the left part of curve as illustrated in figure 5.4 for the validation data.

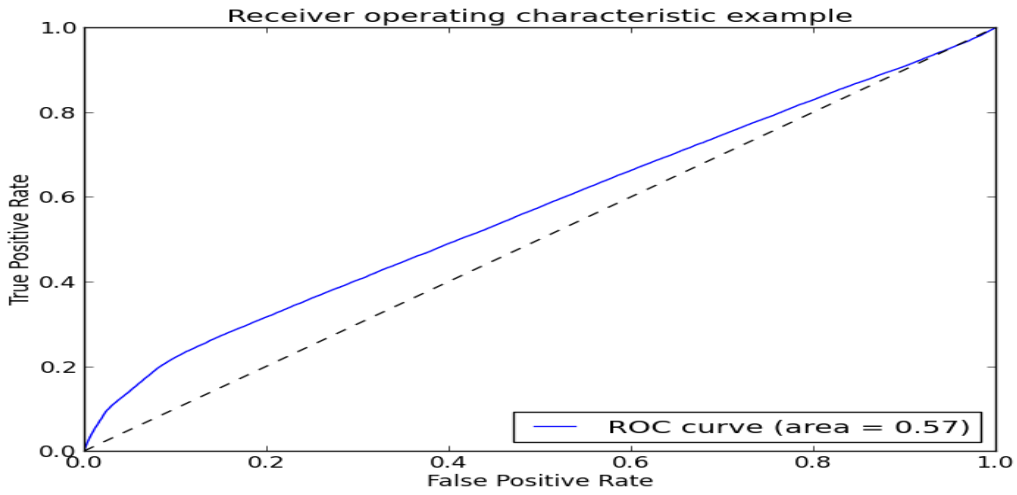

Figure 5.4 : ROC curve of strategy 1 based on the validation schema

It scored 0.610 in the test set.

5.5.2 Collaborative filtering Strategy 2: Customer "looks like" one who had bought the item

Similar with strategy 1, the following sections contain information regarding the reasoning for employing strategy 2, the features generated under it, the pre-processing steps that took place to improve performance, the selected algorithms and the actual performance in respect to the AUC metric.

5.5.2.1 Assumption

The general principle was to find what drives the customers to buy the products irrespective of the sending of the offer. In other words the main question was*:*

Would the customers have bought the product, had they not received the offer anyway?

This is achieved by observing how much a customer that there is some evidence he/she likes the product, looks like another one. This is (naturally) quite powerful for items the customer has no previous relationship direct or indirect.

5.5.2.2 Feature engineering

The generated features for this approach were more abstract in regards to the relationship of the customer with the product. They map or cluster customer behaviours and characteristics such as loyalty, attitude towards specific product departments and level of cardinality.

More specifically the exact features included:

- Counts of top 30 (most popular) departments
- Counts of top 30 (most popular) categories
- Mean quantity, amount purchased by the customer
- Number of records, visits, departments, categories, companies, brands bought by each customer as cardinality measure.
- Mean number of brands by category, categories by department, categories by date, dates by category, departments by date, dates by department
- Percent during the weekend of quantity, amount, records, visits
- Percent returned (aka transactions with negative value)

Additional features were created with *deep learning* by training two Restricted Boltzmann Machines (RBMs) with Bernoulli distribution on Boolean indicators for all remaining (53 after top 30) departments and next 100 (after top 30) categories. Those RBMs were trained with a learning rate of 0.05, 20 iterations and 10 components that are input to the modelling. Only the test set was used for training. There was no feature selection to this strategy and the level of noise and collinearity of the data was handled with high shrinkage during hyper parameter tuning phase of the modelling process.

5.5.2.3 Pre-processing

In order to avoid extreme observations, all the features where transformed by taking the natural logarithm. The most fundamental pre-processing step was how the target variable was formulated as it did not use the repeaters' count or the binary indicator. Instead the target variable was the logarithm of the number of times the customer bought the product 90 days before the coupon was sent as portrayed in figure 5.5:

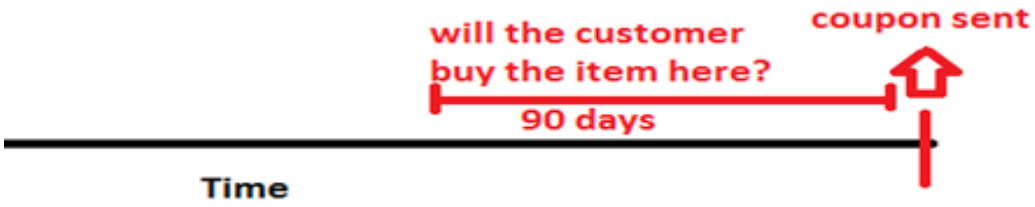

Figure 5.5 : Target variable was formed 90 days prior to sending the coupon

5.5.2.4 Modelling

The models of choice that used all the aforementioned features for strategy 2 were 2 GBMs [Pedregosa et al. 2011] (Gradient Boosting Machines – Regression) trained on the natural logarithm of counts as computed 90 days prior to sending the coupon (to avoid extreme observations). Separate models were made for each offer. The first GBM was trained on customers who did receive the offer. The second on those who did not. The reason behind this division was that customer showed different behaviour in propensity levels to buy the product before and after receiving the coupon, so the models attempted to capture this information. The GBMs were trained with a learning rate of 0.1, 400 estimators, and maximum features per level equal to 2, no usage of subsampling with a least squares loss.

5.5.2.5 Performance

Strategy 2 did very well on new items (right part of the curve) and on higher probability purchased items. Also it discriminated quite equivalently across all offers. It scored 0.616 on the test set. The graphical representation of the ROC curve and the AUC for the validation set can be visualized in figure 5.6:

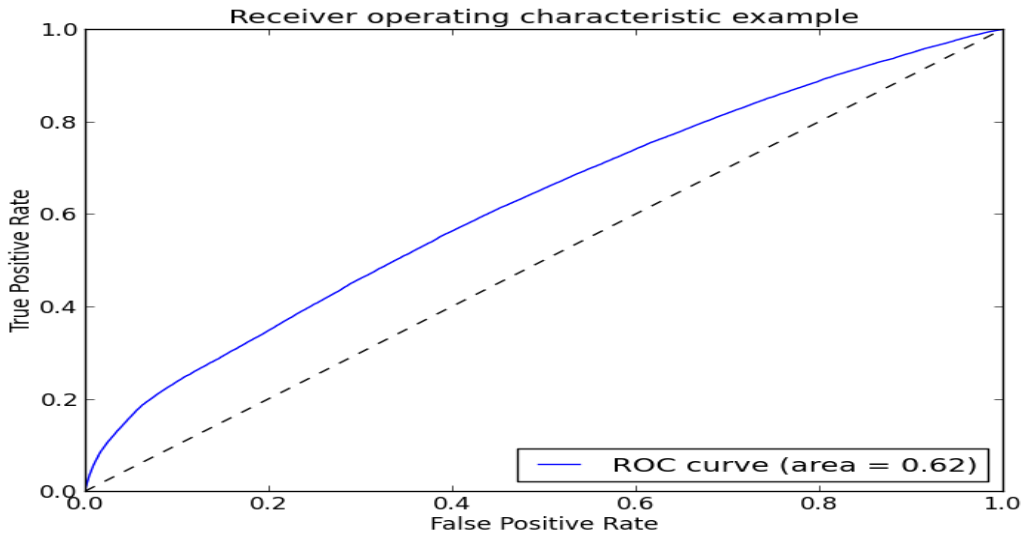

Figure 5.6 : ROC curve of strategy 2 based on the validation schema

5.5.3 Blending the strategies

To decide how to combine (or not) the two strategies, the distribution of the scores for certain offers was observed. For instance offers 1230218 and 1208329 had similar score distributions for strategy 1 and strategy 2 as illustrated by figure 5.7:

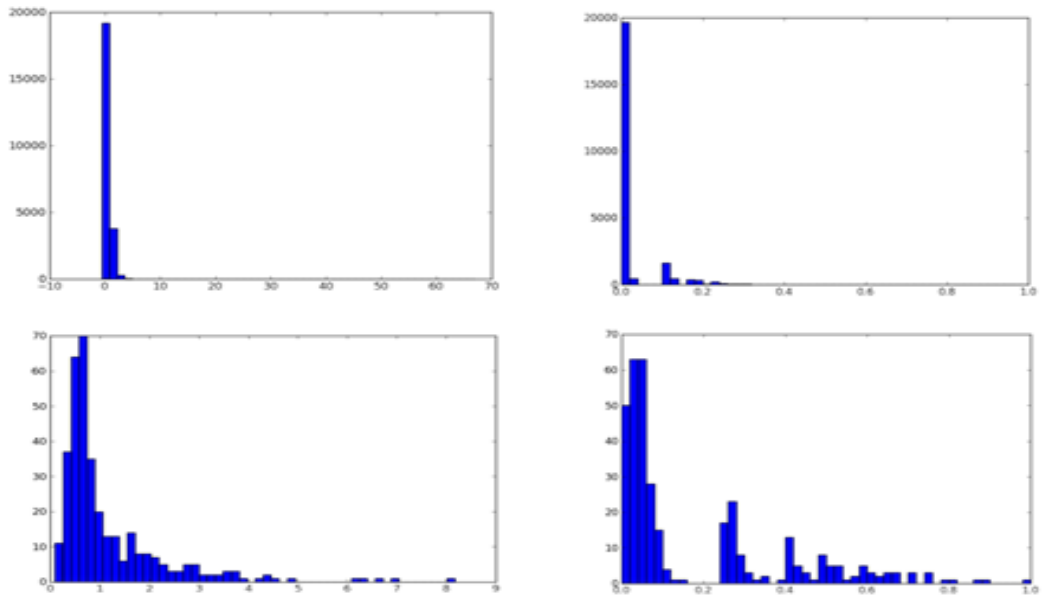

Figure 5.7 : Distribution of strategy 1(left) and 2 (right) for two offers offer_37 and offer_24

As previously cited, the first model was predicting expected quantity of items' bought and the second model on the logarithm of counts as measured 90 days before the sending of the coupon. AUC is a metric that focuses on the ranking of the score distribution and expects that observations with higher score should be correlated with higher probability to buy the item. Whether the score correctly measures that probability is irrelevant to AUC as long as higher scored cases have higher chance to buy the item and vice versa.

Given the different range of possible values of the two groups, the outputs were converted to ranks (to account for the fact that they were trained on different targets). This method is also explained as *rank averaging* by [Henk van Veen 2015]. Based on the similarity of the distributions (in respect to score's order and probability to buy), the strategies were given equal weight. This can be represented with equation 5.3:

$$\text{hybrid}(\hat{\text{Y}}_{cb}, \hat{\text{Y}}_{cf}) = \frac{\hat{\text{Y}}_{cb}^{rank}}{2} + \frac{\hat{\text{Y}}_{cf}^{rank}}{2} \tag{5.4}$$

, where the hybrid mode is defined using two predictions as inputs, the content based (denoted as $\hat{\text{Y}}_{cb}$) and the collaborative filtering based (denoted $\hat{\text{Y}}_{cf}$) .The final result is the average of the two predictions after transforming them into ranks to account for the fact that they have been generated using models trained with different target variables. To create the rank of $\hat{\text{Y}}_{cb}$ (which is denoted as $\hat{\text{Y}}_{cb}^{rank}$) , the vector $\hat{\text{Y}}_{cb}$ gets sorted in an ascending manner and becomes $\hat{\text{Y}}_{cb}^{sort}$. Then for every i out of K, $\hat{\text{Y}}_{cb}^{sort} = [\hat{\text{y}}_{cb,0}^{sort}, \cdots, \hat{\text{y}}_{cb,K}^{sort}]$, $\hat{\text{Y}}_{cb}^{rank} = [\hat{\text{y}}_{cb,0}^{rank}, \cdots, \hat{\text{y}}_{cb,K}^{rank}]$ where:

$$\hat{\text{y}}_{cb,i}^{rank} = \begin{cases} 0 & i = 0 \\ \text{i} & \hat{\text{y}}_{cb,i}^{sort} > \hat{\text{y}}_{cb,i-1}^{sort} \\ \hat{\text{y}}_{cb,-1}^{rank} & \hat{\text{y}}_{cb,i}^{sort} = \hat{\text{y}}_{cb,i-1}^{sort} \end{cases} \tag{5.5}$$

Creating these 2 diverse strategies was critical in achieving the top score, because each method tried to leverage the weaknesses of the other, hence their combination yielded a significant boost in AUC for the test data. The final AUC on the test data is 0.626. The consolidated results of the 2 individuals strategies and their combination is listed below:

Table 5.7: AUC results on individual strategies and combined for the test data.

Strategies	AUC_TEST
Strategy 1: Content-based	0.610
Strategy 2: Collaborative filtering	0.616
$strategy1_{rank}$ x 0.5 + $strategy2_{rank}$ x 0.5	**0.626**

5.7 Conclusion

This chapter described a hybrid method for improving predictions on what the customers of a retailer will buy again in the future given historical purchases of the retailer's products due to coupons' offers. In other words the challenge was focusing on predicting the recommendations that would be more suitable in creating a habit of purchasing these offered products. The displayed methodology was adjusted to make predictions in an irregular environment where the train data different significantly from the test data by means of having different customers, different offered products and different time periods.

Building a reliable cross validation strategy was integral for selecting and tuning a model that could generalize efficiently in the test data. Three different cross validation strategies were considered. The first strategy was a random k-fold cross validation stratified based on the offer. The second method connoted a leave-one-offer-out schema, accounting for the fact that the test data was consisting primarily from offers not present substantially in the train data. The last strategy gave an equal weight to the AUC as computed from the second strategy plus the overall AUC after concatenating all predictions for all n-1 offers again based on the second method. The third method that boasted the best results on the test data (and the smaller gap between train and test AUC performance) was conceptualized from the fact the average propensity of the offers differ significantly. Predicting well inside the histogram of predictions for one offer did not ensure that these predictions will be comparable (propensity wise) with predictions of all other offers resulting in a possible AUC loss.

The hybrid method deployed to maximize AUC in the test data had the form of two separate strategies, each one with its own pipeline of data pre-processing, handling of missing values, features' derivations as well as models' selection and tuning of their hyper parameters. The first strategy was based on content-based filtering and the second strategy on collaborative filtering.

The first approach assumed a direct (via purchasing a product) or indirect (via purchasing from the same department, brand or manufacturer) existing relationship that a customer may have with a product. This approach used simple ridge regression to estimate the actual number of times the customer is going to buy the product (again) in the future.

The second strategy attempted to estimate a score of a customer buying a product even when such relationship as explained before does not exist. This was exploited via finding the characteristics of customers that bought the items (included in the offers) prior to sending the coupon and cross-reference them with these that did receive the coupon to detect similarities. This approach utilized one model per different offer to estimate the tendency of a customer buying the corresponding items using gradient boosted machines and unsupervised models as part of its input data. The target variable for this approach was the natural logarithm of the number of times the customer bought the item 90 days before the coupon was sent.

The ensemble of the two approaches was challenging because the first model was fit on an untransformed regression response variable (e.g. the quantity) and the other model with was the natural logarithm of the number of times the customer bought the product 90 days prior the sending of the coupon. To maximize AUC, these predictions were transformed to ranks and were equally weighted to achieve the best performance in the test data.

6. The StackNet Model

This chapter covers the properties of the StackNet Model. The StackNet Model is a scalable Meta modelling methodology based on a feedforward neural network architecture implemented in the Java programming language where each single activation function is approximated via the usage of different machine learning algorithms with the overall aim to improve accuracy in any machine learning supervised problems.

6.1 Introduction

The rigorous interest in developing better and faster machine learning models in tandem with the rapid growth of the hardware power has made it possible to scale increasingly complex prediction algorithms in order to improve prediction accuracy in many different data science fields. Many algorithms that were developed in the past and were abandoned due to their expensive computational requirements – such as deep learning – are now being re-examined as a means to improve predictions even further. In 1992 stacked Generalization was introduced as a way to combine many different neural network models of similar architecture with another neural network model in order to improve prediction accuracy.

The StackNet model is a methodology primarily based on neural networks in order to combine many different algorithms so that every single link function between layers is replaced with a different machine learning algorithm. The intuition behind this is that the underlying data rarely follow perfectly a specific distribution and an ensemble of different models with different parametric-or-not assumptions can achieve better performance (at the cost of additional computational power).

Note that the function that connects the input layer with one hidden unit h (out of H) from the first hidden layer, takes the form of a linear regression where a single sample x (out of X) with dimensionality J is multiplied with a set of weights $W^{J,H}$ to output an estimate. The function f_1 that describes the above link is displayed in equation 6.1:

$$f_{1,h}(x) = \sum_{j=1}^{J}(G(x_j)W_{j,h}) \quad (6.1)$$

, given a linear activation G on the input sample x from a dataset X and the $W_{j,h}$, the weights that link feature j with the single neuron h . The hidden layer outputs (f_1), assuming there is no other activation taking place. Assuming that this function can be described as a single estimator in the form of a linear regression, this could be re-written more generally as in equation 6.2:

$$f_{1,h}(x, s) = s\big(G(x_j)\big) = s\big(x_j\big) \quad (6.2)$$

, where s is linear regression function. The G function can be removed as the connection is assumed to be linear in respect to the input data point x. The proposed methodology can be extended so that s can be any other machine learning algorithm that given some input data x, produces (and outputs) a score. In the case that many different s algorithms are used, this methodology has the potential to achieve better results than the individual algorithms that comprise it. The initial version of the model is built in the Java programming language.

In contrast to feedforward neural networks, rather than being trained through back propagation, the network is built iteratively one layer at a time using Wolpert's stacked generalization, each of which uses the final target Y as its target. StackNet's ability to improve accuracy is demonstrated via creating different instances of StackNet models with multiple levels and architectures which are then used to rank best the likelihood of a certain song being created before or after 2002 using a set of 90 numerical attributes out of 515,345 songs that come from a subset of the Million Song Dataset [2011]. The latter is a freely-available collection of audio features and metadata for a million contemporary popular music tracks with focus on using this metadata to predict the year a song was created.

Two additional experiments are made, one that measures the trade-off between model complexity and performance and another that investigates the trade-off between models' diversity within the StackNet model and performance. Both these experiments link back to the considerations in literature (section 2.4.2) for building ensembles that are not computational expensive and perform as best as possible.

6.2 Software Review

The following sections briefly reviews other software work in the predictive analytics space with an ensemble functionality. It also reviews the Java programming language in the context of this thesis.

6.2.1 Machine learning Packages

The most popular machine learning software with the ability to combine many algorithms with various ensemble methods is [sklearn 2013], implemented in python with multithreaded capabilities , bringing together many prominent packages focused in different machine learning methods , including [Liblinear 2008] (for large scale linear modelling) and the award winning [Libsvm 2011] (for support vector machines) . The [Weka 2009] data mining Software has made extensive use of ensemble methods in the Java programming language. [Ranklib, 2013] also written in Java, provides modules to combine many learn-to-rank algorithms with various ensemble methods such as bagging and boosting. Many packages exist in the R programming language such as [Carret 2012] or [Rattle 2011] with the aim of bringing many algorithms under the same framework in order to facilitate modelling via ensembling. [Keras 2013], which is based on [Theano 2010] made it possible to combine easily and efficiently many different deep learning architectures.

The [H2O 2016] predictive modelling open source software package contains a module called *Stacked Ensemble* that uses Super Learning or Stacked Regression defined as a class of algorithms that involves training a second-level "metalearner" to find the optimal combination of the base learners. [LeDell 2015] proposed a scalable learning methodology with a software application to combine multiple, typically diverse, base learning algorithmswith a Super Learner.

Although the stacked generalization concept was introduced in 1992 by Wolpert et al. and a few software applications leveraged this to boost accuracy, even after the advent of deep learning (as of the recent era), there has not been until today any prominent software package that makes use of this methodology with the score of expanding it onto more than one level. Nevertheless the concept of deep ensembling has been very frequently used especially in the form of Gradient Boosting Trees and the award winner [Xgboost 2016].

6.2.2 Java programming Language

Java was first developed in 1995 and has now one of the most popular programming languages in the world, however is the first choice when it comes to data science and machine learning [Puget 2016]. It is also deemed more verbose than Python [Prechelt 2000] and R [Murtagh 2005]. However Java, is still a good choice when it comes to building large distributed machine learning systems. One of Java's main advantages over many other languages is that it can be used without many (if any) changes with any operational system and it is also relatively quick compared to most object oriented languages [Brose et al. 2001] .

Regarding the current thesis, although it is acknowledged that Java is not as fast as C or C++, nevertheless was deemed to be the language of choice to develop the application given its overall characteristics and specifically its popularity, safety and simplicity [Tiobe 2017].

6.3 StackNet Model

The StackNet model refers to an extension of [Wolpert's 1992] stacked generalization to multiple algorithms using a neural network architecture with multiple layers where each neuron's function is replaced with a different machine learning algorithm each time. The name of the model originates from"Stack" that directly refers to "stacked generalization" and "net" because of the aforementioned neural network architecture. The model developed by the author (as methodology) was first used (and the term was introduced) in the winning solution of "Truly Native"[6] [2015] data modelling competition hosted by the popular platform Kaggle.com. The final StackNet structure which won that challenge can be viewed in figure 6.1 and included 4 layers with various algorithms for each one. StackNet has also been used (and won) other predictive modelling challenges such as the Homesite Quote Conversion[7] [2016] also hosted by the kaggle.com platform.

[6] http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/
[7] http://blog.kaggle.com/2016/04/08/homesite-quote-conversion-winners-write-up-1st-place-kazanova-faron-clobber/

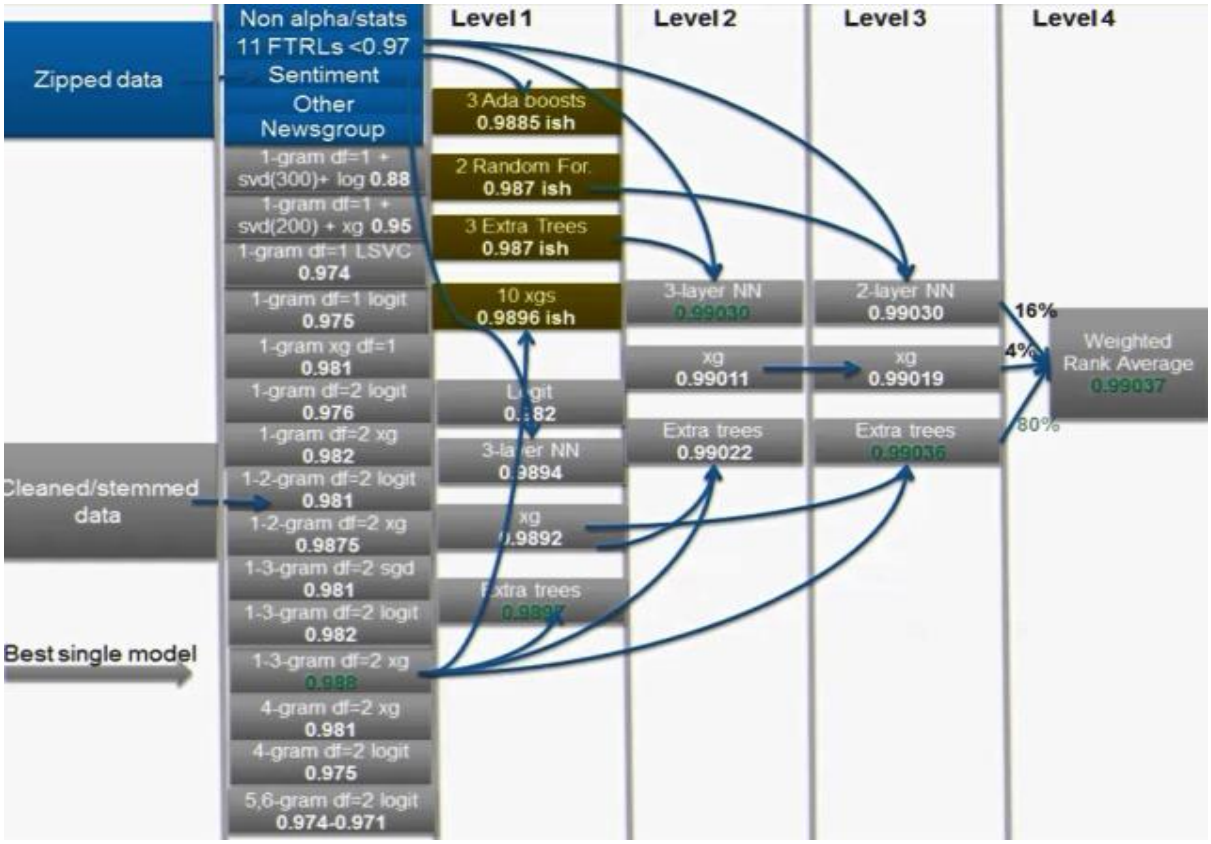

Figure 6.1 : StackNet model with 4 layers used to maximize AUC and win the Truly Native Kaggle challenge.

6.3.1 Mathematical formulation

The function that connects the input layer with the neuron h in the first hidden layer was defined previously (assuming a linear activation function on the input data) equation 6.3:

$$f_{1,h}(x, s) = s(x_j) \tag{6.3}$$

where s could be any (machine learning) algorithm that takes some input data x and outputs a score. Assuming there is a vector S with size H that contains the functions of different algorithms in respect to the input data, $f_{1,h}$ can be re-written as 6.4 for a given neuron h:

$$f_1(x, S) = S_h(x_j) \tag{6.4}$$

To add another layer, the outputs of all $f_1(x, S, h)$ will be used as inputs to a f_2 function that given a new (Meta) model l, attributed to a neuron m (in the second layer), the original vector of S models (with size H) of the first hidden layer and the input data x, will have the form of 6.5:

$$f_{2,m}(x, l, S) = l\big(f_1(x, S_1), f_1(x, S_2), \dots., f_1(x, S_H)\big) \tag{6.5}$$

Following the same reasoning as 6.4, if there is a vector L of neurons of size M, then f_2 could be re-written as 6.6 for a given neuron m :

$$f_2(x, L, S) = L_m\big(f_1(x, S_1), f_1(x, S_2), \dots., f_1(x, S_H)\big) \tag{6.6}$$

Instead of having different vectors of models for different layers (in this case S for layer one and L for layer two), there could be a 2-dimenional vector V that holds all these algorithms with size N,D^N, where N is the number of the hidden layers and D_n the number of hidden neurons (or models) within the hidden layer n. Therefore, replacing L, S from 6.6 with V would result in 6.7:

$$f_2(x, V) = V_{2,m}\left(f_1\big(x, V_{1,1}\big), f_1\big(x, V_{1,2}\big), \dots., f_1\big(x, V_{1,D_1}\big)\right) \tag{6.7}$$

This logic could be extrapolated to any number of layers N, where the n_{th} layers uses the outputs of the predictions of the leyer n-1 to as inputs in order to output a score. Assuming there is a neuron k in the n_{th} layer, its output would be generated using 6.8:

$$f_n(x, V) = V_{n,k}\left(f_{n-1}\big(x, V_{n-1,1}\big), f_{n-1}\big(x, V_{n-1,2}\big), \dots., f_{n-1}\big(x, V_{n-1,D_{n-1}}\big)\right) \tag{6.8}$$

6.3.2 Modes

The "stacking" element of the StackNet model could be run with 2 different modes. The first mode (also set as the default) is the one already mentioned in 6.8 and assumes that each layer uses the predictions (or output scores) of the direct previous one, which is similar to a typical feedforward neural network. The second mode (also called *restacking*) assumes that each layer uses previous neurons activations as well as all previous layers' neurons. Therefore the previous formula can be re-written as equation 6.9 (assuming the layers N>3):

$$\mathrm{f_n(x,V)} = \mathrm{V}_{n,k}\begin{pmatrix} \mathrm{f_{n-1}(x,V_{n-1,1})}, \mathrm{f_{n-1}(x,V_{n-1,2})}, \dots, \mathrm{f_{n-1}(x,V_{n-1,D_{n-1}})}, \\ \mathrm{f_{n-2}(x,V_{n-2,1})}, \mathrm{f_{n-2}(x,V_{n-2,2})}, \dots, \mathrm{f_{n-2}(x,V_{n-2,D_{n-2}})}, \\ \dots, \\ \mathrm{f}_{n-N+1}\mathrm{(x,V}_{n-N+1,1}\mathrm{)}, \mathrm{f}_{n-N+1}\mathrm{(x,V}_{n-N+1,2}\mathrm{)}, \dots, \mathrm{f}_{n-N+1}\mathrm{(x,V}_{n-N+1,D_{n-N+1}}\mathrm{)} \end{pmatrix} \quad (6.9)$$

The intuition behind this mode is driven from the fact that the higher level algorithm have extracted information from the input data, but rescanning the input space may yield new information not obvious from the first passes. This is also driven from the forward training methodology discussed below and assumes that convergence needs to happen within one model iteration. The following graph (6.2) illustrates the difference between the two modes.

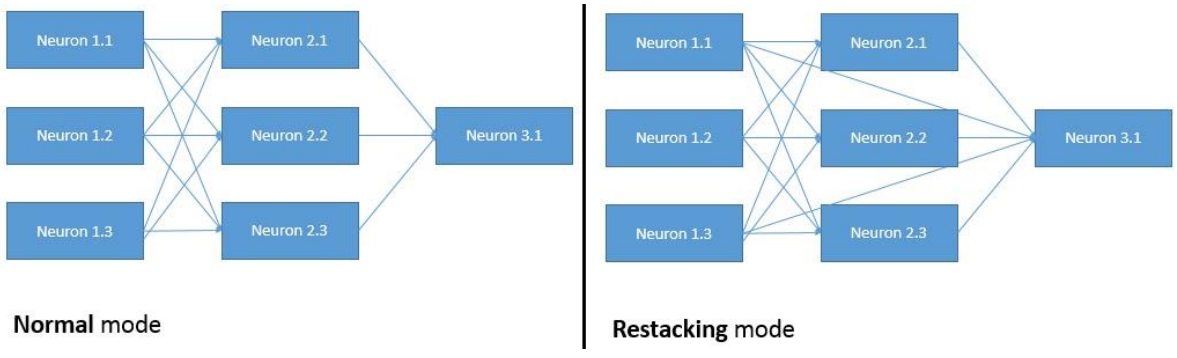

Figure 6.2 : StackNet's link modes

6.3.3 Training with K-fold cross validation

The typical neural networks are most commonly trained with a form of *back propagation*. Back propagation requires a differentiable loss function. In the premise that any machine learning algorithm could be included in a StackNet model, it is not currently easy to formalize the back propagation training approach since not all losses from all models are differentiable. Therefore stacked generalization is used to train this network instead.

Stacked generalization requires a forward training methodology that splits the data into two parts – one of which is used for training and the other for predictions. The reason this split is necessary is to avoid over fitting. However splitting the data in just 2 parts would mean that in each new layer the second part needs to be further dichotomized. This has the effect of increasing the bias as each algorithm will have to be trained on increasingly less data.

To overcome this drawback the algorithm utilizes a k-fold cross validation (where k is a hyper parameter) so that all the original training data is stored in different k batches thereby outputting as many predictions as there are samples in the training data. Therefore the training process is consists of 2 parts:

1. Split the data k times and run k models to output predictions for each k part and then concatenate the k parts back together to the original order so that the output predictions can be used in later stages of the model. This process is illustrated below in figure 6.3:

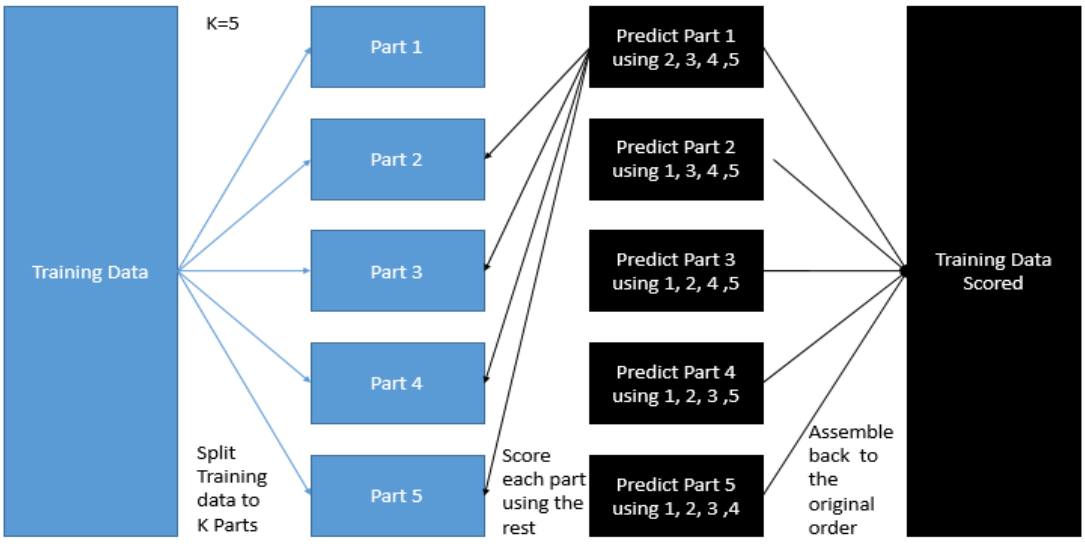

Figure 6.3 : Example of K-fold scoring-output for StackNet given an algorithm in a neuron where K=5

2. Rerun the algorithm on the whole training data to be used later on for scoring the external test data. There is no reason to limit the ability of the model to learn using 100% of the training data since the output scoring is already fairly unbiased (given that it is always scored as multiple holdout sets).

It should be noted that (1) is only applied during training to create unbiased predictions for the second layers' model to fit on the predictions of the previous layer during scoring time (and after model training is complete) only (2) is in effect.

The k-fold may also be viewed as a form of regularization where smaller number of folds (but higher than 1) ensure that the validation data is big enough to demonstrate how well a single model could generalize. On the other hand higher k means that the models come closer to running with 100% of the training and may yield more unexplained information. The best values could be found through cross validation.

Another possible way to implement this could be to save all the k models and use the average of their predicting to score the unobserved test data, but this will result in all the models not being trained with 100% of the training data and may therefore be suboptimal. It should be noted that the loss function the StackNet model optimizes is defined by the last model in the last layer and therefore it is algorithm-specific. For example if a logistic regression model is chosen in the last layer, all other models' outputs from previous layers are used within the logistic regression model to optimize a log likelihood function.

The optimal parameters O of the $V_{n,k}$ algorithm (which may be weights, nodes, latent vectors, support vectors or else depending of the algorithm's type) , denoted as $O_{V_{n,k}}$ can be specified given a loss function LL to minimize (suitable for the algorithm's type), a normal connection mode and a target variable Y as:

$$\widehat{O_{V_{n,k}}} = \underset{O_{V_{n,k}}}{\operatorname{argmin}} \, LL(O_{V_{n,k}}, \left(f_{n-1}(x, V_{n-1,1}), f_{n-1}(x, V_{n-1,2}), \ldots., f_{n-1}(x, V_{n-1,D_{n-1}})\right), Y) \quad (6.10)$$

, where the minimization of the loss function is subject to some parameters $O_{V_{n,k}}$ of $V_{n,k}$, the input data of this algorithm as produced from (1) and the target variable Y. For example if the

squared loss function E was used (in the place of L) as the main function to minimize, given a dataset X and a single sample from it as x_i ,6.10 could be re-written as:

$$\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}} = \underset{\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}}{\operatorname{argmin}} E\left(\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}\right) = \underset{\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \mathrm{V}_{n,k}(x_i)\right)^2 \tag{6.11}$$

It should be noted, that function LL does not need to be only subject to minimization, but maximization too. Also, theoretically, the function LL may not need the target variable Y at all and unsupervised models could be used too, for any layer that is not the N_{th} (last)

6.3.4 The input data type (software specific)

The StackNet model as implemented in Java supports three different input data formats:

6.3.4.1 Java's double 2-dimensional array

This is one of the most common Java objects and can be perceived as arrays of double arrays, coded as double [][]. This Object was chosen to be used so that the algorithms are accessible to anyone without requiring a special input format, however it is not the friendliest method to do so from a memory point of view as it is a complex object that consists of many smaller in-memory objects, all with different addresses making column-wise loops relatively slow. However it has been found to be quite efficient when there are many features in the input data (so that every in-row loop is more efficient).

6.3.4.2 Fixed-size matrix or fsmatrix

This is a complex object that is also interpreted as a 2-dimensional array, but similar to C arrays it is actually a 1-dimensional Java double array with a fixed size equal to the product of rows and column of the desired data matrix. This object requires much less memory to store its data and can be quite fast for most operations. However it still connotes a dense representation and therefore cannot scale in very sparse problems. The *fsmatrix* require a row and column dimension to be initialized like:

```
fsmatrix sample_matrix= new fsmatrix(int rows, int columns);
```

6.3.4.3 Sparse matrix or smatrix

Sparse matrix refers to a way to store the data so that all zero elements are not captured. In other words it represents a mapping that points only to the non-zero elements and consist of 3 java 1-dimensional arrays – one double array to store the non-zero elements, a integer array to hold the dimension's position (e.g. the column) for each of the non-zero elements and the last integer array connotes the start and end index for each row based on the previous 2 arrays. Depending on the algorithms the 2 integer arrays may switch positions (so that the first points to rows and the second one to columns) in order to speed up the training and scoring process. Additionally (and after activation) the matrix contains a hash-table for quick column or row value accessing, however this comes with additional memory overhead and it is optional. Depending on the sparsity of the input data this matrix may increase the training speed multiple times, requiring much less data. The smatrix can be created via providing any of the previous two input data objects (fsmatrix or 2-dimensional Java double array) or can be constructed manually via providing the three 1-dimensional Java arrays as stated previously.

6.3.5 The objects

While there many existing algorithms in the java programming language, in order to address the needs to scalability and optimal performance under the StackNet framework, most of the algorithms had to be written from scratch. Although the initial arsenal of available algorithms in the StackNet software may not be deemed very rich, nevertheless crucial effort was made to include representatives from most algorithmic families which fall into the following categories.

6.3.5.1 Tree-based algorithms

Tree-based algorithms are likely to be commonly used and form one of the most predictive class of algorithms. These algorithms include standard decision tree regressors and classifiers. Depending on the way multiple trees can be combined this class also includes Random Forests (e.g. bagging of trees) or Gradient boosted Trees for both regression and classification. It should be noted that for binary classification the trees also support an AUC split criterion apart from the common ones (e.g. information gain).Gradient Boosting has been implemented so

that the base estimators are Random Forests and no single decision trees, however the default number of trees for such forests is one which equates them to decision trees. A Gradient Boosting model is implemented via initializing it:

```
GradientBoostingForestClassifier model = new GradientBoostingForestClassifier();
```

Then any number of hyper parameters may be defined as:

```
model.estimators=100;
model.threads=3;
model.verbose=true;
model.copy=false;
model.trees=1;
model.shrinkage=0.1;
model.cut_off_subsample=1.0;
model.feature_subselection=0.1;
model.max_depth=8;
model.max_features=1.0;
model.max_tree_size=-1;
model.min_leaf=2.0;
model.min_split=5.0;
model.Objective="RMSE";
model.row_subsample=0.9;
model.seed=1;
```

All models follow the exact same structure and the details about the tuneable hyper parameters can be found in the Javadoc accompanying the release of the software.

To train the algorithm the fit() method is invoked that takes as input a data object (any of the three kinds) as defined above. The response variable defined in the hyper parameter section and not in the fit method for optimization reasons:

```
smatrix X = null;
double response []= new double [X.GetRowDimension()];
model.target=response;
model.fit(X);
```

Once the algorithm is fitted, predictions can be extracted in the form of probabilities or class results:

```
double probabilities[]=model.predict_proba(X);
double classes[]=model.predict(X);
```

The tree-related parameters are presented in table 6-1 as they appear within StackNet.

Table 6-1: Tree specific hyper parameters in StackNet

Parameter	Explanation
max_depth	Maximum depth of the tree
objective	The objective based on which the split is determined. It may be "RMSE "for regression and "Entropy" for classification and "AUC" for binary classification
row_subsample	Proportion of observations to consider
max_features	Proportion of columns (features) to consider in each level
cut_off_subsample	Proportion of best cut offs to consider. This controls how Extremely Randomized the tree will be
feature_subselection	Proportion of columns (features) to consider for the whole tree
min_leaf	Minimum weighted sum of cases to keep after splitting node
min_split	Minimum weighted sum of cases to split a node
max_tree_size	Maximum number of nodes allowed in the tree

The Random forest related parameters are presented in table 6-2:

Table 6-2: Random Forest specific hyper parameters in StackNet

P**arameter**	**Explanation**
estimators	Number of trees to build. In most situations after 100 it does not improve dramatically more

The Gradient Boosted Forests of trees' related parameters are presented in table 6-3:

Table 6-3: Gradient Boost Random Forest of trees' specific hyper parameters in StackNet

Parameter	Explanation
estimators	Number of Random Forests to build. In most situations after 100 it does not improve dramatically mor.
trees	Number of trees in each Forest. The default is 1 which basically connotes a tree estimator
shrinkage	Penalty applied to each estimator. Smaller values prevent overfitting. Needs to be between 0 and 1. There is also a negative correlation between estimators and shrinkage.

6.3.5.2 Linear regression

Linear regression is implemented with L2 (Ridge), or L1 (Lasso) regularizations and can be trained with various optimizations algorithms such as the ordinary method (with matrix multiplications) and stochastic gradient decent methods.

The Linear Regression related parameters are presented in table 6-4:

Table 6-4: Linear Regression hyper parameters in StackNet

Parameter	Explanation
C	Regularization value, the more, the stronger the regularization. A value here basically triggers a Ridge regression
Type	Can be one of "Routine", "SGD" Routine is the Ordinary Least Squares method which is solved with matrix multiplications
Objective	"RMSE"
learn_rate	For SGD
UseConstant	If true it uses an intercept
maxim_Iteration	Maximum number of iterations

6.3.5.3 Logistic regression

Logistic regression is also implemented with L2 (Ridge), or L1 (Lasso) regularizations and can be trained with a Newton-Raphson (with matrix multiplication) method, stochastic gradient decent method and the Liblinear's implementation. Multinomial logistic regression is implemented via running 1 model for each one of the distinct classes of the target variable.

The Logistic Regression related parameters are presented in table 6-5:

Table 6-5: Logistic Regression hyper parameters in StackNet

Parameter	Explanation
C	Regularization value, the more, the stronger the regularization
Type	Can be one of "Liblinear", SGD". Default is Liblinear.
learn_rate	For SGD
UseConstant	If true it uses an intercept.
maxim_Iteration	Maximum number of iterations

6.3.5.4 Linear support vector machines

StackNet includes Liblinear's fast implementations for both regression and classification as well as SGD for when a linear kernel is selected. These models are denoted as LSVC for classification and LSVR for regression.

The LSV related parameters are presented in table 6-6:

Table 6-6: LSVC and LSVR hyper parameters in StackNet

Parameter	Explanation
C	Regularization value, the more, the stronger the regularization.
Type	"SGD"
learn_rate	For SGD
UseConstant	If true it uses an intercept
maxim_Iteration	Maximum number of iterations

6.3.5.5 LibFM

The factorization machines' representative in the StackNet software is LibFM (which is commonly used in the recommender systems' area). This is implemented with L2 regularization and convergence is reached via using stochastic gradient decent and supports both regression and classification.

The libFM related parameters are presented in table 6-7:

Table 6-7: libFM hyper parameters in StackNet

Parameter	Explanation
C	Regularization value, the more, the stronger the regularization
C2	Regularization value for the latent features
Lfeatures	Number of latent features to use.
init_values	Initialise values of the latent features with random values between [0,init_values)
learn_rate	For SGD
maxim_Iteration	Maximum number of iterations
Type	Only "SGD"
UseConstant	If true it uses an intercept

6.3.5.6 Neural networks

A very specific architecture of neural networks has been implemented as this software did not aim to become a comprehensive deep learning library but rather a tool that achieves better accuracy via combining different machine learning algorithms leveraging the pros and cons of each algorithmic family (or at least the most prominent representatives) using CPU. Therefore a-two-layer neural network has been implemented that supports regularization. The number of hidden units in each layer is a hyper parameter. Both regression and classification can be run so that they optimize a multi-label objective directly or via breaking down to many single-response problems.

The neural networks are denoted as *Softmaxnnclassifier* for classification and *Multinnregressor* for regression. The main difference is that the *Softmaxnnclassifier* has a softmax output layer suitable for classification problems and *Multinnregressor* has a simple linear output activation. The parameters of these models are presented in table 6-8:

Table 6-8: hyper parameters of Softmaxnnclassifier and Multinnregressor in StackNet

Parameter	Explanation
C	Regularization value, the more, the stronger the regularization
h1	Number of the 1st level hidden units
h2	Number of the 2nd level hidden units
init_values	Initialise values of hidden units with random values between [0,init_values)
smooth	Value to divide gradients and aid convergence
connection_nonlinearity	Can be one of "Relu","Linear","Sigmoid","Tanh". Commonly Relu performs best.
learn_rate	For SGD
maxim_Iteration	Maximum number of iterations
Type	Only "SGD".
UseConstant	If true it uses a bias/intercept in each node.

6.3.5.7 Naïve Bayes

The simple Naïve Bayes implementation was included to provide quick solutions at the cost of – in most cases – some loss in accuracy. A scaling or regularization parameter has been added to control the size of the product in the probability estimation.

The only parameter associated with Naïve Bayes is presented in table 6-9:

Table 6-9: hyper parameters of Naïve Bayes in StackNet

Parameter	Explanation
Shrinkage	Can be seen as a form of a penalty to avoid really big products' failures.

6.3.5.8 All algorithms

All included algorithms follow a similar structure where, after initialization, any number of hyper parameters may be added. Additionally most of the algorithms support *scaling* to aid convergence, particularly useful for linear algorithms optimized via gradient based methods. The default scaling object is a *maxscaler()* which connotes that every feature is divided by its absolute maximum value. This ensures that all values will be within the range of [-1, 1]. The

scaling method is a hyper parameter of the model and gets invoked in *fit()* and *predict()* methods.

Additionally, all estimators include a seed (integer) value in order to be able to replicate any randomized procedures included in the algorithms. They also include a *threads* term that controls parallelism within the model training and predicting. There are some other miscellaneous options too, like whether to copy the data or to print updates about the algorithms' progress via setting a *verbose* parameter.

All algorithms can accept hyper parameters within the one command using a string of space separated parameters as *parameter_name:value*. Most algorithms support some form of verbosity so that they print information about their progress and can be copied. By invoking the PrintInformation() , the details of the given object are printed.

6.4 Using StackNet for "Song year of release" classification

The functionality of StackNet, and specifically, its ability to combine different machine learning models in order to achieve a better classification outcome can be better demonstrated through an experimentation with real data. In the following experiment different versions of StackNet with different modes, levels and structures will be used to rank best the likelihood of a certain song being created before or after 2002 using a set of 90 numerical attributes out of 515,345 songs that come from a subset of the Million Song Dataset [Bertin-Mahieux et al. 2011].

6.4.1 Training and test data

The current experiment will use the *YearPredictionMSD* Data Set available in University of California (UCI's) Machine Learning Repository [Lichman 2013] and connotes a part of the Million Song Dataset. The data set contains 515,345 rows, each one representing a song that is described using 90 numerical features as well as an indicator in the beginning ranging from 1922 to 2011 stating the year the song was created. The nature of the features is not within the scope of this experiment, however it is stated in the online repository that the features are

extracted from the 'timbre' features from [The Echo Nest API nd] which is an online resource that provides metadata and audio analysis for millions of tracks and powers many music applications on the web and smart phones. The first 12 features are related to timbre averages and the remaining 78 to timbre covariance. These are calculated based on all 'segments', each segment being described by a 12-dimensional timbre vector.

The experiments uses the first 463,715 examples for training and the last 51,630 examples for testing purposes. This split is suggested from the online resource, because it was designed in such a way so that it avoids the 'producer effect' via making certain that no song from a given artist ends up in both the training and test data set. Furthermore the target variable (namely the year the song was created) is converted into a binary indicator for whether a given song was created before or during 2002 (0) or after 2002 (1). The cut-off year of 2002 was selected to proportionally balance the number of 0s and 1s in the data. For consistency with other experiments the metric to optimize is again AUC and Loglikelihood.

6.4.2 First layer single model

The StackNet model utilizes different models as nodes to its first layer which is in direct connection with input data. The first layer in this experiments consists of 12 models shuffled from different machine learning methods, with the idea that diversity in early levels can yield better predictions in later levels than the single model-nodes involved in this ensemble framework. All models have been manually trained before entered into the StackNet model in order to optimize their hyper parameters, using a random 5-fold cross validation process on the training dataset.

The selected models and their most important hyper parameters can be viewed below in table 6-10:

Table 6-10: First Layer models in StackNet

Models in first layer	Parameters
Logistic Regression (Logistic_M1)	C=0.5
Random Forest Classifier (Random_Forest_M1)	estimators=100, max_depth=15, max_features=0.3, min_leaf=5, row subsample=0.95
SVM (Linear Kernel) (Linear_Support_Vector_M1)	C=3.0
LibFm Classifier (LibFm_M1)	maxim Iterations=16, C= 0.000001, init_values =0.9, learn rate =0.9, Lfeatures=3
Naïve Bayes (Naïve_Bayes_M1)	shrinkage=0.01
Neural Net Classifier NN_2layersRelu_M1	C= 0.000001, learning rate=0.009, maxim_iterations=20, h1,h2=(30,20), act=relu, out=softmax
Gradient Boosted RandomForest Classifier (GBM_M1)	estimators=100, max_depth=8, max_features=0.5,min_leaf=2.0, eta=0.1, row subsample=0.9
Linear Regression (Linear_Regression_M1)	C=0.00001
Random Forest Classifier (RandomForest_M1)	estimators=100, max_depth=8, max_features=0.5, min_leaf=2.0, row subsample=0.9
Gradient Boosted RandomForest Regressor (GBM_Regressor_M1)	estimators=100, max_depth=9, max_features=0.5,min_leaf=2.0, eta=0.1, row subsample=0.9
RandomForest Regressor (Random_Forest_Reg_M1)	estimators=100, max_depth=14, max_features=0.25 ,min_leaf=5, row subsample=1.0
Linear Support Vector Regression (Linear Support_Vector_Reg_M1)	C=3.0

Each model is run directly on the 90 features available in the dataset with no other data pre-processing apart from maximum scaling. The performance of each one of the models in terms of AUC and logloss is displayed in the following table 6-11:

Table 6-11: Performance of 1st layer models in StackNet

StackNet: First Layer Models	AUC	Loglikelihood
Logistic_M1	0.7759	0.5796
Random_Forest_M1	0.7913	0.5578
Linear_Support_Vector_M1	0.7753	4.3356
LibFm_M1	0.7757	0.5797
Naïve_Bayes_M1	0.6432	1.7416
NN_2layersRelu_M1	0.8002	0.5457
GBM_M1	0.8034	**0.5409**
Linear_Regression_M1	0.7744	0.7313
RandomForest _M1	0.7779	0.6148
GBM_Regressor_M1	**0.8045**	0.5990
Random_Forest_Reg_M1	0.7883	0.5634
Linear_Support_Vector_Reg_M1	0.7753	0.6295

The GBM models have performed best both in terms of AUC and logloss, with neural networks being close behind. The best reported AUC is 0.804 which implies a strong discriminative

capability of classifying songs for whether they were created before or after 2002. The best reported logloss is 0.541.

6.4.3 2nd layer single models

The models selected for the 2nd layer are only trained on 12 features – those being the output predictions of the M1 models. These models can also be viewed as single-layer StackNets since each one corresponds to a Meta algorithm that takes as inputs previous models' predictions. This time the selected number of models is only 4, since the initial 90-dimensional feature set is already compressed down to 12 outputs and adding many more models is likely to recycle the same information leading to overfitting. The final architecture has been found through various trials of slightly different structures based on the 5-fold cross validation schema mentioned before.

The parameters of each model in the second layer is illustrated in table 6-12:

Table 6-12: Second layer models in StackNet

Models in second layer	Parameters
Logistic Regression (Logistic_l2_M2)	C=0.5
Random Forest Classifier (Random_Forest_M2)	estimators=1000, max_depth=7, max_features=0.4, min_leaf=1, row subsample=1.0
Gradient Boosted RandomForest Classifier (GBM_M2)	estimators=1000, max_depth=5, max_features=0.5, min leaf=1.0, shrinkage=0.01, row subsample=0.9
Neural Net Classifier (NN_2layersRelu_M2)	maxim Iterations=16, C= 0.000001, init_values=0.9, learn rate =0.9, Lfeatures=3

Note that in the second modelling phase, the optimum parameters of the models have become more *modest*. For example the tree-based models have significantly *smaller depths* than their predecessors. This occurs naturally since the underlying features-predictions from the previous models are more correlated with target variable as there are meant to be predictions for it. Therefore the new models do not need to be so exhaustive with the underlying feature set when optimizing the error in respect to the target variable. Table 6-13 shows the absolute results of the new M2 models as well as the proportion of improvement versus best results of the M1 models:

Table 6-13: Performance of 2nd Layer models in StackNet

StackNet: Second Layer Models	AUC	Loglikelihood	AUC%	Loglikelihood%
Logistic_l2_M2	**0.8102**	0.5328	0.71%	1.49%
Random_Forest_M2	0.8088	0.5343	0.53%	1.21%
GBM_M2	0.8100	**0.5327**	0.69%	1.52%
NN_2layersRelu_M2	0.8101	0.5332	0.69%	1.41%

The best AUC has now increased to the 0.81+ area which is a +0.71% proportional improvement from the best M1 model in terms of AUC. The proportional impact in loglikelihood is superior as the biggest increase is around 1.5% better than the best predecessor. Interestingly, different types of 2nd layer models are better for each metric which may connote that extra benefit could be derived via adding one more layer to the StackNet as it can be assumed that each underlying model is utilizing the input information slightly differently.

6.4.4 3rd layer models

In the final layer, apart from running a new meta classifier on the output of the previous 4-dimensional layer, a second meta classifier will be used that activates the "*restacking*" StackNet mode which brings up to the same level all previous models (from all previous layers) . In other words the First M3 StackNet will be run on a 4-dimensional feature set and the second M3 Stacknet on 16-dimensional (12 + 4) feature set in order to compare the ability of the restacking model to re-recycle information and perform better than the simple one. The parameters of each model in the third and final layer is illustrated in table 6.14:

Table 6-14: Third layer models in StackNet

2 different StackNets	Parameters
Random Forest Classifier (Random_Forest_M3)	estimators=1000, max_depth=6, max_features=0.7, Restacking OFF
Random Forest Classifier (Random_Forest_Restack_M3)	estimators=1000, max_depth=6, max_features=0.7, Restacking On

Table 6.15 shows the results of all models included in the 2 StackNets as well as the absolute and proportional (compared to the 1st layer model) performance of the new models:

Table 6-15: Performance of 3-Layer StackNets and their predecessors

StackNet: First Layer Models	AUC	Loglikelihood	AUC_dif	Loglikelihood%
Logistic_M1	0.7759	0.5796	-	-
Random_Forest_M1	0.7913	0.5578	-	-
Linear_Support_Vector_M1	0.7753	4.3356	-	-
LibFm_M1	0.7757	0.5797	-	-
Naïve_Bayes_M1	0.6432	1.7416	-	-
NN_2layersRelu_M1	0.8002	0.5457	-	-
GBM_M1	0.8034	0.5409	-	0.0000
Linear_Regression_M1	0.7744	0.7313	-	-
RandomForest _M1	0.7779	0.6148	-	-
GBM_Regressor_M1	0.8045	0.5990	0.0000	-
Random_Forest_Reg_M1	0.7883	0.5634	-	-
Linear_Support_Vector_Reg_M1	0.7753	0.6295	-	-
StackNet: Second Layer Models				
Logistic_l2_M2	0.8102	0.5328	0.71%	1.49%
Random_Forest_M2	0.8088	0.5343	0.53%	1.21%
GBM_M2	0.8100	0.5327	0.69%	1.52%
NN_2layersRelu_M2	0.8101	0.5332	0.69%	1.41%
StackNets of Level 3				
Random_Forest_M3(Restack:OFF)	0.8105	0.5323	0.74%	1.58%
Random_Forest _M3 (Restack:ON)	**0.8115**	**0.5309**	0.87%	1.84%

Both StackNet models yielded a small uplift in both AUC and log likelihood compared to their direct predecessors. The structure of the first StackNet model (without Restacking) assumes direct relationships from one layer to another. It has performed marginally better than the best M2 model. Figure 6.4 displays the modelling architecture which assumes that there is a 90-dimenional input dataset where all M1 models were trained on:

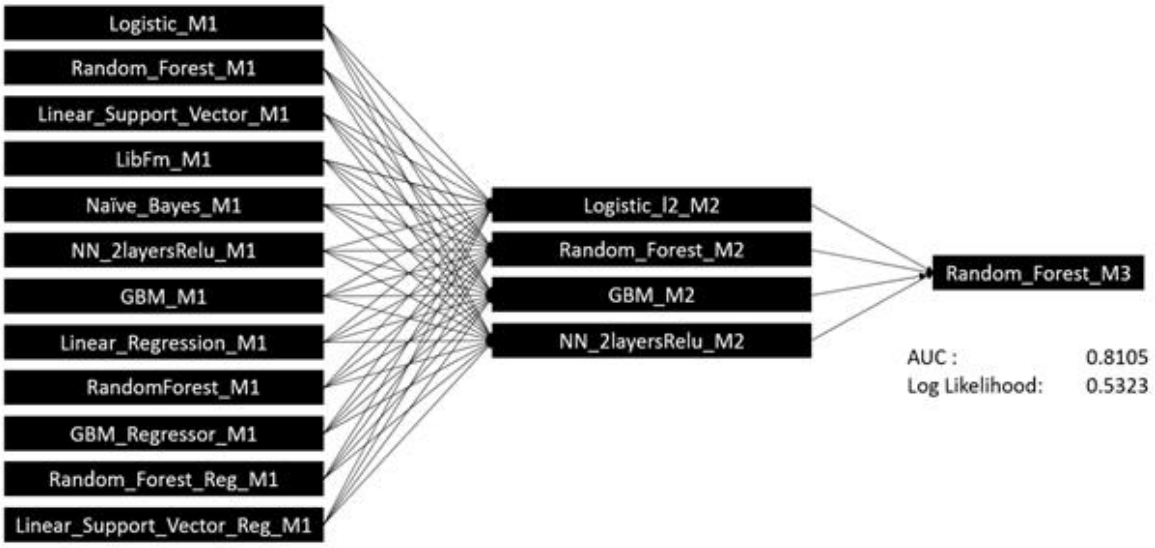

Figure 6.4 : 3-layer StackNet with Restacking OFF

The activation of the Restacking option has improved the results in both metrics even further. One may find it difficult to comprehend how it is possible that some models which have been

trained with other models, to still benefit from the presence of the latter in the modelling process. One need to note that the StackNet methodology assumes a normal forward pass of the data where each algorithm is fitted directly to predict the target variable. In other words there is not a concept of epochs as it is common in neural network models, since the algorithm is trained using cross validation and making predictions based on the prediction errors of this process for each model. In other words a specific model cannot be re-fitted to improve on the errors it might have produced and it is left to the next-level to account for the errors. However information missed in the early stages of the process may not be fully retrievable later on. Restacking allows higher level models to re-use information contained in early models. It is further possible that having more information about the data (as superior high-levelled Meta models might do) can allow the algorithms to seize the initial data from different angles and explore information not visible the first time.

Figure 6.5 demonstrates how the outlook of the models differs from figure 6.4 when restacking mode is used:

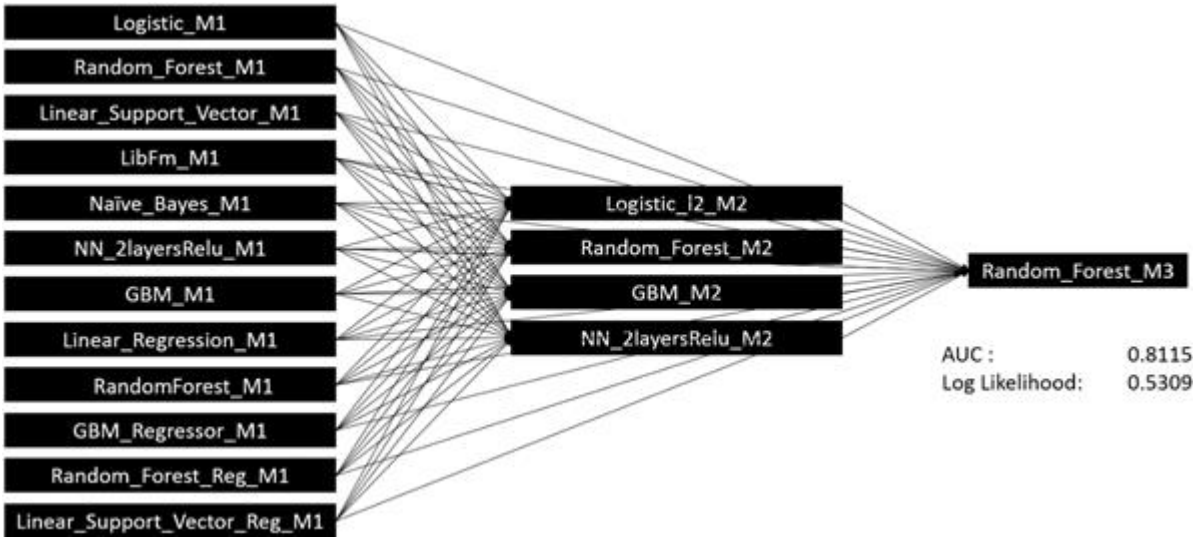

Figure 6.5 : 3-layer StackNet with Restacking ON

The actual Java code to execute the 3-Layer StackNet starts with initializing a *StackNetClassifier* Object:

```
StackNetClassifier StackNet = new StackNetClassifier (); // Initialise a StackNet
```

Which is then followed by a 2 dimensional String array with the list of models in each layer along with their hyper parameters in the form of as in "*estimator [space delimited hyper parameters]*":

```
String models_per_level[][]=new String[][];// holds the parameters for each model

//First Level
{"LogisticRegression C:0.5 maxim_Iteration:100 verbose:true",
"RandomForestClassifier bootsrap:false estimators:100 threads:25
cut_off_subsample:1.0 feature_subselection:1.0 max_depth:15 max_features:0.3
max_tree_size:-1 min_leaf:2.0 min_split:5.0 row_subsample:0.95",
"LSVC C:3 maxim_Iteration:50",
"LibFmClassifier maxim_Iteration:16 C:0.000001 lfeatures:3 init_values:0.9
learn_rate:0.9 smooth:0.1",
"NaiveBayesClassifier Shrinkage:0.01",
"softmaxnnclassifier maxim_Iteration:20 C:0.000001 tolerance:0.01 learn_rate:0.009
smooth:0.02 h1:30 h2:20 connection_nonlinearity:Relu init_values:0.02",
"GradientBoostingForestClassifier estimators:100 threads:25 verbose:false trees:1
rounding:2 shrinkage:0.1 feature_subselection:0.5 max_depth:8 max_features:1.0
min_leaf:2.0 min_split:5.0 row_subsample:0.9",
"LinearRegression C:0.00001",
"GradientBoostingForestClassifier estimators:100 threads:3 verbose:true trees:1
rounding:2 weight_thresold:0.4 feature_subselection:0.5 max_depth:8
max_features:1.0 min_leaf:2.0 min_split:5.0 row_subsample:0.9",
"GradientBoostingForestRegressor estimators:100 threads:3 trees:1 rounding:2
shrinkage:0.1 feature_subselection:0.5 max_depth:9 max_features:1.0 min_leaf:2.0
min_split:5.0 row_subsample:0.9",
"RandomForestRegressor estimators:100 internal_threads:1 threads:25 verbose:true
cut_off_subsample:1.0 feature_subselection:1.0 max_depth:14 max_features:0.25
max_tree_size:-1 min_leaf:2.0 min_split:5.0 Objective:RMSE row_subsample:1.0",
"LSVR C:3 maxim_Iteration:50" ,
//Second Level
"RandomForestClassifier estimators:1000  threads:25 verbose=false
cut_off_subsample:0.1 feature_subselection:1.0 max_depth:7 max_features:0.4
max_tree_size:-1 min_leaf:1.0  min_split:2.0
row_subsample:1.0",
"GradientBoostingForestClassifier estimators:1000 threads:25 verbose:false trees:1
rounding:4 shrinkage:0.01 feature_subselection:0.5 max_depth:5 max_features:1.0
min_leaf:1.0 min_split:2.0 row_subsample:0.9",
"softmaxnnclassifier maxim_Iteration:20 C:0.000001 tolerance:0.01 learn_rate:0.009
smooth:0.02 h1:30 h2:20 connection_nonlinearity:Relu init_values:0.02",
"LogisticRegression C:0.5 maxim_Iteration:100 verbose:false" ,
//Third Level
"RandomForestClassifier estimators:1000  threads:25 verbose=false
cut_off_subsample:0.1 feature_subselection:1.0 max_depth:6 max_features:0.7
max_tree_size:-1 min_leaf:1.0  min_split:2.0 row_subsample:1.0"}
;

StackNet.parameters=models_per_level; // adding the models' specifications
```

The remaining parameters to be specified include the cross validation training schema, the Restacking mode option, setting a random state as well as some other miscellaneous options:

```
StackNet.threads=4; // models to be run in parallel
StackNet.folds=5; // size of K-Fold
StackNet.stackdata=true; // use Restacking
StackNet.print=true; // this helps to avoid rerunning should the model fail
StackNet.output_name="restack";// prefix for each layer's output.
StackNet.verbose=true; // it outputs
StackNet.seed=1; // random state
```

Ultimately given a data object X and a 1 dimensional vector y, the model can be trained using:

```
StackNet.target=y; // the target variable
StackNet.fit(X); // fitting the model on the training data
```

6.4.5 Summary of the experiment

Using a StackNet model on the YearPredictionMSD Data Set to predict if a given song was created before or after 2002, has resulted in improved performance over AUC and log likelihood compared to the single models involved in the process.

Building the various layers sequentially, it is clear that every new layer improves the performance of their inputs (or their predecessor). The models need to become *shallower* or simpler (parameter-wise) as StackNet becomes deeper to account for the already-compressed information contained in the Meta-models.

Activating the Restacking mode and in the absence of the ability to recycle information through multiple epochs (as it is normally the case with neural network frameworks), has allowed to increase performance over a single feedforward direct approach. Computational time of the models is primarily a factor of the available cores since every model can be run in parallel.

6.5 Investigating diversity-performance trade-off

In chapter 2.4.3 it was highlighted that a crucial component for improving the performance of an ensemble is the diversity of the models contained. The following experiment investigates a case of two ensembles built for a binary classification task, the first including models of linear nature and the second models from various algorithmic families.

6.5.1 The data

The data used in this experiment can be found in kaggle.com[8]. The train dataset contains 33K rows of anonymized historical information (summarized by 9 features) of the employees of a company regarding their role within that company and the resources to which they have access. The test data contain 59K rows and have similar structure. The dataset also contains a binary target variable which connotes whether the employee should have access privileges or not. The aim of such a classification model is to minimize the human involvement required to grant or revoke employee access via predicting whether an employee should have special accesses or not. The objective to optimize is AUC against models that either predict the probability for the target to be 1 or just a score (for regression models). All the features are of categorical nature and are expressed as integer codes. The features contain high cardinality. The number of unique values for each feature is demonstrated in table 6-16.

Table 6-16: Features and number of distinct values

features	unique values
feature1	7,518
feature2	4,243
feature3	128
feature4	177
feature5	449
feature6	343
feature7	2,358
feature8	67
feature9	343

[8] https://www.kaggle.com/c/amazon-employee-access-challenge/data

In order to increase the feature space and allow the algorithms to produce different results, a sample of n-way pairwise interactions of features were created where n=5. In other words all possible combinations of 2, 3, 4 and 5 features were considered. The generated features based on this process were assigned a new unique code. The resulted interactions were found using random K-fold cross validation testing against AUC while using a logistic regression model. The final table of features with interactions is presented below:

Table 6-17: Generated n-way interactions and type of interaction

Feature interactions	n-way
Feature1_Feature2	2
Feature1_Feature4	2
Feature2_Feature3	2
Feature2_Feature4	2
Feature2_Feature5	2
Feature2_Feature6	2
Feature2_Feature7	2
Feature2_Feature8	2
Feature3_Feature6	2
Feature3_Feature7	2
Feature4_Feature7	2
Feature6_Feature7	2
Feature1_Feature2_Feature3	3
Feature1_Feature2_Feature4	3
Feature1_Feature3_Feature4	3
Feature1_Feature3_Feature5	3
Feature1_Feature4_Feature5	3
Feature1_Feature5_Feature6	3
Feature1_Feature5_Feature8	3
Feature1_Feature6_Feature8	3
Feature2_Feature3_Feature4	3
Feature2_Feature3_Feature6	3
Feature2_Feature3_Feature7	3
Feature2_Feature4_Feature6	3
Feature2_Feature4_Feature7	3
Feature2_Feature5_Feature6	3
Feature2_Feature5_Feature7	3
Feature2_Feature5_Feature8	3
Feature2_Feature6_Feature7	3
Feature2_Feature7_Feature8	3
Feature3_Feature4_Feature8	3

Feature4_Feature6_Feature7	3
Feature5_Feature6_Feature7	3
Feature1_Feature2_Feature3_Feature4	4
Feature1_Feature2_Feature3_Feature5	4
Feature1_Feature2_Feature3_Feature8	4
Feature1_Feature2_Feature4_Feature5	4
Feature1_Feature3_Feature4_Feature5	4
Feature1_Feature3_Feature5_Feature6	4
Feature1_Feature4_Feature7_Feature8	4
Feature2_Feature3_Feature4_Feature7	4
Feature2_Feature3_Feature5_Feature6	4
Feature2_Feature3_Feature5_Feature7	4
Feature2_Feature3_Feature7_Feature8	4
Feature2_Feature4_Feature7_Feature8	4
Feature1_Feature3_Feature4_Feature5_Feature7	5
Feature1_Feature3_Feature4_Feature5_Feature8	5
Feature2_Feature3_Feature4_Feature7_Feature8	5

Given the categorical nature of features, they have been transformed using dummy coding. This means that each distinct value of a feature becomes its own binary variable indicating whether that value is present in a sample row (denoted as 1) or not (denoted as 0). The representation of this data within the algorithm is sparse as explained in 6.3.4.3 to allow for memory-efficient computations.

6.5.2 The diversity metric

In 2.42 pairwise correlation among models' predictions was highlighted as a possible means of measuring the overall diversity of an ensemble. Assuming all level 1 models ae positively correlated with each other, the overall diversity of the ensemble can be measured via taking the average of all entries of the level 1 predictions' corresponding Pearson correlation matrix as R. Given the definition of Pearson correlation (r) in 2.2.2.4, an input size of N level 1 models, the overall diversity for that first level based on correlation can be expressed as:

$$\text{diversity}(R) = \frac{1}{N \times N} \sum_{n=1}^{N} \sum_{k=1}^{N} r(n,k) \tag{6.12}$$

Where R is the correlation matrix of all level 1 predictions and $r_{n,k}$ the pairwise Pearson correlation of the prediction of model n and the prediction of model k. Higher values would connote a lower diversity, because the correlation (or similarity) between models is higher.

6.5.3 The ensembles' structure

Two different ensembles were considered with nine input (level 1) models and one Meta model. In both ensembles the Meta model is a Random Forest Classifier, however the input level 1 models differ between the 2 ensembles.

The first ensemble consists of 9 Logistic regression models trained with different C (L2), regularization parameters, and optimization methods. These parameters were randomized within some intervals. The interval for C was [0.001, 100]. Once the C value was set, the rest of the parameters were tuned to get better performance as measured from inside StackNet's K-Fold cross validation mechanism for K=5. Table 6-18 presents the models' name and their hyper parameters:

Table 6-18: Models and hyper parameters for the first ensemble

Level 1 models of linear ensemble	parameters
Logistic Regression_1	C=1.5 maxim_Iteration=100
Logistic Regression 2	C=0.002 maxim_Iteration=60
Logistic Regression 3	C=0.01 maxim_Iteration=200
Logistic Regression 4	C=5.5 maxim_Iteration=100
Logistic Regression 5	C=0.8 maxim_Iteration=100
Logistic Regression 6	C=10.0 maxim_Iteration=100
Logistic Regression 7	C=6.0 maxim_Iteration=100
Logistic Regression 8	C=15.0 maxim_Iteration=100
Logistic Regression 9	C=3.5 maxim_Iteration=100

The second ensemble includes models from other algorithmic families outside the linear spectrum. The parameters of these models were tuned manually to maximize AUC based on StackNet's internal cross validation schema. Table 6-19 presents the models' name and hyper parameters of this mixed ensemble:

Table 6-19: Models and hyper parameters for the mixed ensemble

Level 1 models of mixed ensemble	parameters
LogisticRegression_1	C=1. maxim_Iteration=100
LogisticRegression_2	C=0.001 maxim_Iteration=60
LSVC_3	C=0.01 maxim_Iteration=100
LinearRegression_4	C=20. maxim_Iteration=10
LibFmClassifier_5	maxim_Iteration=50 C=0.000001 C2=10. Lfeatures=2
softmaxnnclassifier_6	maxim_Iteration=30 C=0.00001 h1=30 h2=30 connection_nonlinearity=Relu
GradientBoostingClassifier_7	shrinkage=0.16 estimators=300 max_depth=7 max_features=0.6
LogisticRegression_8	C=0.5 maxim_Iteration=20
LSVC_9	C=0.5 maxim_Iteration=100

The mixed ensemble consists of three logistic regression models, two linear support vector machines (denoted as LSVC), 1 linear regression model, one libFM classifier, a gradient boosted tree model and a neural network model with Softmax output layer and 2 hidden layers connected using a rectifier activation.

6.5.4 The ensembles' first layer performance

Tables 6-20 and 6-21 illustrate the performance each one of the models in terms of AUC for both the internal K-fold cross validation and the actual results in the test data:

Table 6-20: Linear models' performance in AUC for cv and test

model	cv AUC	test
Logistic Regression_1	0.894	0.911
Logistic Regression_2	0.886	0.899
Logistic Regression_3	0.883	0.893
Logistic Regression_4	0.893	0.913
Logistic Regression_5	0.893	0.909
Logistic Regression_6	0.890	0.911
Logistic Regression_7	0.893	0.912
Logistic Regression_8	0.887	0.908
Logistic Regression_9	**0.894**	**0.913**

average	0.890	0.908

Table 6-21: mixed models' performance in AUC for cv and test

model	cv AUC	test
LogisticRegression_1	**0.893**	**0.910**
LogisticRegression_2	0.886	0.899
LSVC_3	0.891	0.906
LinearRegression_4	0.875	0.890
LibFmClassifier_5	0.890	0.909
softmaxnnclassifier_6	0.881	0.900
GradientBoostingClassifier_7	0.851	0.865
LogisticRegression_8	0.880	0.893
LSVC_9	0.873	0.882
average	0.880	0.895

Apart from the individual results, the average of all models' AUC is displayed for both the internal cross validation and test results. The linear ensemble consists on average of stronger models with better performance in the internal validation and test data. The best model in the linear ensemble (LogisticRegression_9) boasts an AUC of 0.913 in the test data, while the best model in the mixed ensemble (LogisticRegression_1) scores only 0.911 in the test data. Additionally the overall average AUC of all models for the linear ensemble in the test data (of 0.908) is higher than the equivalent one for the mixed ensemble (of 0.895).

6.5.5 The ensembles' diversity

In order to estimate diversity as defined in 6.5.2, the Pearson correlation matrix of all predictions of all models needs to be computed. Table 6.22 illustrates the correlation matrix of all linear models' predictions for the test data. The model's numbering follows the same order as in table 6-20 (i.e. model1 is Logistic Regression_1 and model9 is Logistic Regression_9):

Table 6-22: linear models' correlation matrix

	model1	model2	model3	model4	model5	model6	model7	model8	model9
model1	1.000	0.959	0.936	0.986	0.998	0.967	0.984	0.947	0.995
model2	0.959	1.000	0.985	0.952	0.954	0.934	0.951	0.915	0.959
model3	0.936	0.985	1.000	0.912	0.937	0.886	0.909	0.861	0.925
model4	0.986	0.952	0.912	1.000	0.974	0.995	1.000	0.985	0.998
model5	0.998	0.954	0.937	0.974	1.000	0.950	0.971	0.928	0.986
model6	0.967	0.934	0.886	0.995	0.950	1.000	0.996	0.997	0.987
model7	0.984	0.951	0.909	1.000	0.971	0.996	1.000	0.987	0.997
model8	0.947	0.915	0.861	0.985	0.928	0.997	0.987	1.000	0.973
model9	0.995	0.959	0.925	0.998	0.986	0.987	0.997	0.973	1.000

Using the formula of 6.5.2, the estimated diversity is 0.9648.

Table 6-23 illustrates the equivalent table for the mixed ensemble, where model1 is Logistic Regression 1 and model9 is LSVC9:

Table 6-23: mixed models' correlation matrix

	model1	model2	model3	model4	model5	model6	model7	model8	model9
model1	1.000	0.957	0.863	0.892	0.991	0.938	0.824	0.919	0.779
model2	0.957	1.000	0.888	0.923	0.951	0.950	0.784	0.871	0.771
model3	0.863	0.888	1.000	0.926	0.845	0.852	0.775	0.837	0.881
model4	0.892	0.923	0.926	1.000	0.881	0.874	0.763	0.846	0.819
model5	0.991	0.951	0.845	0.881	1.000	0.935	0.803	0.902	0.763
model6	0.938	0.950	0.852	0.874	0.935	1.000	0.773	0.849	0.743
model7	0.824	0.784	0.775	0.763	0.803	0.773	1.000	0.881	0.753
model8	0.919	0.871	0.837	0.846	0.902	0.849	0.881	1.000	0.836
model9	0.779	0.771	0.881	0.819	0.763	0.743	0.753	0.836	1.000

The estimated diversity based on 6-23 correlation matric is 0.8726**.** As it is expected, the mixed ensemble has higher diversity as on average, the pairwise correlations between models' predictions for the test data are lower than these in the linear model. The consolidated results in table 6-24:

Table 6-24: Linear and mixed models' level 1 diversity

Ensemble type	**diversity**
Linear ensemble	0.9648
Mixed ensemble	0.8726

Based on the metric defined in 6.5.2, it can be concluded that the mixed ensemble is more diverse than the linear one, which is not surprising given the bigger variety of the algorithms contained.

6.5.6 The ensembles' final performance

These nine model's predictions were input to a higher level (Meta) classifier. Both ensembles' level 1 output became input to Random Forest (level 2) classifier. The parameters of this model included 300 trees, maximum tree depth equal to 8 and the proportion of features to consider at each level of the tree was set to 50%. These parameters were obtained from within StackNet's cross validation procedure. The final results for the internal and test AUC results for both Random Forest models are presented in table 6-25:

Table 6-25: Linear and mixed models' level 1 diversity

Level 2 input	**cv AUC**	**test**
Random Forest on linear Ensemble	0.896	0.914
Random Forest on mixed ensemble	**0.901**	**0.917**
Difference (mixed – linear)	+0.005	+0.003

The Level 2 Random Forest classifier that was trained on the outputs of the mixed ensemble gave better results for AUC (cv + test), although the best individual model of the mixed ensemble was not better than the best individual model of the linear ensemble, nor the average AUC of the models contained in the mixed ensemble was better than the one of the models contained in the linear ensemble.

6.5.7 Conclusion diversity-performance trade-off

The findings of the current experiment suggest that diversity (as measured based on correlation) of inferior layers is critical for getting better results in the Meta layer. This was demonstrated via creating 2 different ensembles, one that contained models of linear nature and another that contained models from various algorithmic families. The former ensemble had on average

stronger individual models than the latter ensemble and the diversity metric (computed from the predictions' correlation matrix) showed lower diversity for the linear ensemble than the mixed ensemble. A Random Forest Meta level 2 classifier trained on the outputs of both ensembles demonstrates consistently (between internal CV and test results) better performance for the mixed model. This concludes that selecting different algorithmic families as input (level 1) models generates higher diversity and achieves better performance than solely maximizing the performance of one classifier (or one family of classifiers).

Ultimately, the diversity within the models proved to be more important in securing a better generalization in the test data than having on average stronger but more correlated models within the ensemble. While this finding may not be consistent when the models do not boast a certain level of accuracy in respect to the target variable, however in the context of models having strong predictive power (as in the example where all models had and AUC>0.85) diversity was deemed more important for obtaining a better result.

6.6 Investigating ensemble plateauing

Formulating ensemble methods comes at a computation cost that based on the level of sophistication may be quite considerable. Investigating the trade-off between diversity and performance in 6.5 exhibited an interesting finding. Under certain assumptions regarding strong predictors in the ensemble that boast positive correlations with one another, diversity was more important in obtaining a better generalization error than having strong correlated models. [Zhou et al. 2002] demonstrated that using a large number of models in an ensemble is not better (performance-wise) than a (diverse) subset of these models. Combining all the information, it is worth investigating to what extend (if any) adding more models to the ensemble does not bring performance uplift.

6.6.1 The data

The data for this experiment is exactly the same as in 6.5.

6.6.2 The setup of the experiment

To investigate the (potential) point to which performance starts downgrading, a pool of 36 level 1 models was generated. It included 27 random models and the 9 models from the mixed ensemble described in 6.5 which were significantly tuned based on cross validation performance. The 27 models come from the following algorithmic families:

- Linear models
- Random forests
- Gradient boosted trees
- Neural networks with 2 hidden layers and Relu activation
- Linear support vector machines
- Factorization machines (LibFM)

The hyper parameters of these 27 models were initially randomized and then were recalibrated (mildly) based on the K-fold cross-validation performance from within StackNet, where K=5. The final list of models, their hyper parameters and their average cross validation AUC is listed in table 6-26:

Table 6-26: Pool of 36 models (9 +27) along their parameters and AUC cv performance

Level 0 model with index	Parameters	cv AUC
LogisticRegression_1	C=1. maxim_Iteration=100	0.893
LogisticRegression_2	C=0.001 maxim_Iteration=60	0.886
LSVC_3	C=0.01 maxim_Iteration=100	0.891
LinearRegression_4	C=20. maxim_Iteration=10	0.875
LibFmClassifier_5	maxim_Iteration=50 C=0.000001 C2=10. Lfeatures=2	0.890
softmaxnnclassifier_6	maxim_Iteration=30 C=0.00001 h1=30 h2=30 connection_nonlinearity=Relu	0.881
GradientBoostingClassifier_7	shrinkage=0.16 estimators=300 max_depth=7 max_features=0.6	0.851
LogisticRegression_8	C=0.5 maxim_Iteration=20	0.880
LSVC_9	C=0.5 maxim_Iteration=100	0.873
LogisticRegression_9	C=5. maxim_Iteration=100	0.893
LogisticRegression_10	C=0.01 maxim_Iteration=120	0.883
LSVC_11	C=0.1 maxim_Iteration=200	0.884
LinearRegression_12	C=30. maxim_Iteration=20	0.877
LibFmClassifier_13	maxim_Iteration=40 C=0.00001 C2=15. Lfeatures=1	0.889
softmaxnnclassifier_14	maxim_Iteration=35 C=0.0005 h1=20 h2=20 connection_nonlinearity=Relu	0.882
GradientBoostingClassifier_15	shrinkage=0.15 stimators=400 max_depth=7 max_features=0.6	0.851

LogisticRegression_16	C=0.2 maxim_Iteration=20	0.877
LSVC_17	C=0.1 maxim_Iteration=100	0.846
LogisticRegression_18	C=10.0 maxim_Iteration=200	0.890
LogisticRegression_19	C=0.005 maxim_Iteration=90	0.884
LSVC_20	C=0.035 maxim_Iteration=200	0.889
LinearRegression_21	C=40. C=0.025 maxim_Iteration=30	0.878
LibFmClassifier_22	maxim_Iteration=50 C=0.000001 C2=20. Lfeatures=3	0.890
softmaxnnclassifier_23	maxim_Iteration=35 C=0.00005 h1=15 h2=10 connection_nonlinearity=Relu	0.881
GradientBoostingClassifier_24	shrinkage=0.15 estimators=500 max_depth=7 max_features=0.6	0.852
LogisticRegression_25	C=0.04 maxim_Iteration=20	0.869
LSVC_26	C=0.05 maxim_Iteration=100	0.809
LogisticRegression_27	C=30. maxim_Iteration=200	0.876
LogisticRegression_28	C=0.0005 maxim_Iteration=150	0.884
LSVC_29	C=0.01 maxim_Iteration=250	0.891
LinearRegression_30	C=50. maxim_Iteration=30	0.879
LibFmClassifier_31	maxim_Iteration=50 C=0.000001 C2=20 Lfeatures=3	0.890
softmaxnnclassifier_32	maxim_Iteration=35 C=0.00055 h1=25 h2=10 connection_nonlinearity=Relu	0.882
GradientBoostingClassifier_33	shrinkage=0.25 estimators=100 max_depth=6 max_features=0.6	0.836
LogisticRegression_34	C=0.004 maxim_Iteration=20	0.855
LSVC_35	C=0.005 maxim_Iteration=200	0.720

In order to estimate the plateauing, the following simulation steps are formulated:

1. Defined the number of simulations S=50.
2. In each simulation the order of the (N=36) level 1 models (which can be defined as S_1 to S_{36}) is randomly changed (i.e. shuffled).
3. Assuming a target variable Y, one-by-one the 36 models' predictions are used as inputs to a level 2 Meta classifier (denoted as F_2). This Meta classifier is a Random forest with 300 trees, maximum tree depth equal to 8 and the proportion of features to consider at each level of the tree was set to 50%. This is the same Meta Classifier used in 6.6. In other words there are 36 rounds in each simulation and equal number of level 2 models are built in each round. The first round builds an F_2 model using only the first randomly shuffled level 1 (S_{n1}) model. The second round builds the F_2 model with two inputs stacked together ($S_{n1} \sim S_{n2}$) until the dimensionality for the input of the F_2 model reaches 36 ($S_{n1}, S_{n2}, \ldots, S_{n36}$) when all level 1 models have been stacked at round 36.
4. The cross validated AUC is computed at the end of each round.
5. The average cross validated AUC is reported for each round/order after all 50 simulations are completed.

Consider the following pseudocode assuming a number of base level (f_1) predictions are already made. Comments are made with *italics*:

1. A_AUC= [0,0,…,0] *# array with size 36 initialized with zero values*
2. I= [1,2,…,36] *# array of indices*
3. S= [$f_1(x, S_1), f_1(x, S_2), \ldots, f_1(x, S_{36})$] *# base models predictions on the input data x AS 2d matrix with 33K rows and 36 columns.*
4. Y=[1,1,1,….,1] *# Array with size around 33K where each $y_i \in \{0,1\}$*
5. For s=1 → s=50 *# for 50 simulations*
    a. F1data = [] *# empty array to be populated with predictions from S*
    b. I_sfhuffled→random_shuffle(I) *# indices I are randomly shuffled*
    c. For n=1→ n=36 *# for all 36 base (f_1) models*
        i. F1n = S [I_sfhuffled [n]] *# retrieve a random f_1 prediction from S*
        ii. F1data → [F1data ~ F1n] *# stack predictions f_1 to F1data*
        iii. AUCsn=0 *# initialize AUC for s simulation and model round n*
        iv. Lm=Random Forest Classifier (params) *# the F2 model, initialized given some parameters*
        v. AUCsn=performKfold(Lm, F1data, Y, k=5) *# obtain an average AUC out of 5 estimates given 5 Lm models trained on 80% of the data {F1data, Y} and making predictions to the remaining 20% of the data.*
        vi. A_AUC [n]= A_AUC [n] + AUCsn *# add AUC estimate to n round*
6. For n=1→ n=36
    a. A_AUC[n]= A_AUC[n]/50 *# obtain average AUC for all rounds based on all simulations*

Figure 6.6: Pseudo code for generating average AUC estimates per round

6.6.3 Results of the experiment

Table 6-27 demonstrates the consolidated (average) results for each model round after 50 simulations:

Table 6-27: Model rounds and cross-validation AUC

Round	Average cv AUC
round1	0.87015
round2	0.89036
round3	0.89369
round4	0.89499
round5	0.89588
round6	0.89683
round7	0.89731
round8	0.89749
round9	0.89776
round10	0.89796
round11	0.89808
round12	0.89846
round13	0.89872
round14	0.89885
round15	0.89892
round16	0.89891
round17	0.89901
round18	0.89904
round19	0.89920
round20	0.89920
round21	0.89924
round22	0.89932
round23	0.89929
round24	0.89930
round25	0.89933
round26	0.89942
round27	0.89947
round28	0.89951
round29	0.89948
round30	0.89952
round31	0.89952
round32	0.89955
round33	**0.89960**
round34	0.89958
round35	0.89958
round36	0.89958

The results are also illustrated in graphical format in Figure 6.7:

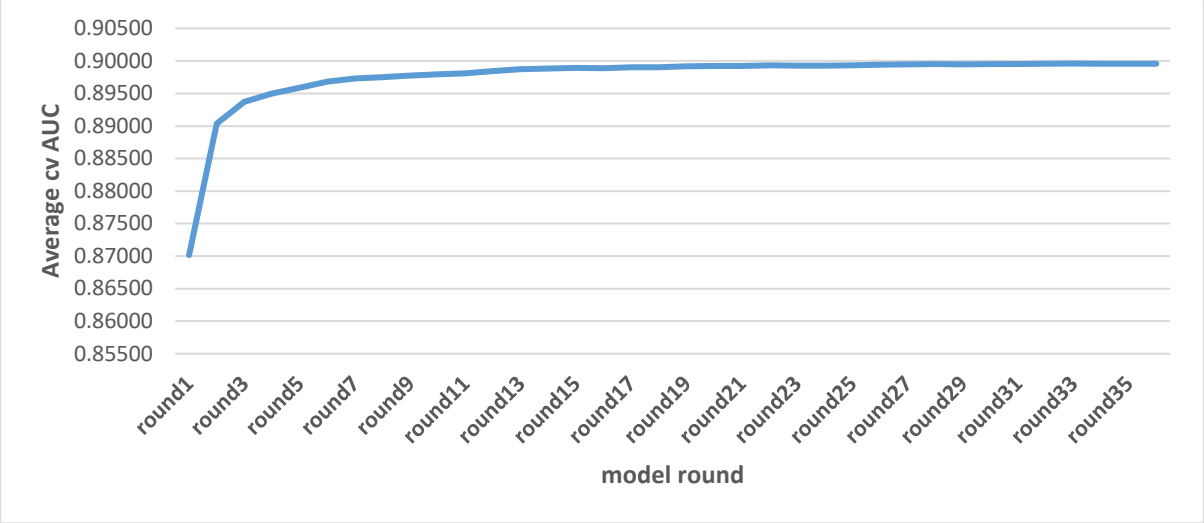

Figure 6.7: Model round versus cross-validation AUC

Both sources of information conclude that the plateauing of AUC does occur. The best average AUC performance is obtained when the 33rd model is inserted in the ensemble and (on average) models that enter past this point deteriorate the performance. It would appear that most of the AUC gain occurs within the first 10 rounds where AUC starts (on average) from 0.87 and ends at approximately 0.898. The remaining (26) rounds are only able to lift AUC up to 0.899.

What is noteworthy is that the Meta learner performance (0.901) of the first 9 models which had been manually tuned and were also part of the mixed ensemble in experiment 6.5, demonstrated higher cross validation AUC than any of the ensembles in any round-order of this experiment. It should be re-highlighted that these 9 models were also part of the current experiment, which potentially concludes that in order to get better performance out of the ensemble, it is better to include a number of diverse models, but increasing the size of the ensemble will not necessary yield better results, instead the optimum number models as well as the diversity of the algorithms need to be investigated for a given task.

Although on average the AUC of all the different rounds is not superior to the ensemble built on top of the mixed ensembles' 9 features, however there have been specific simulations where a certain number of input models (in most cases less than 15 input models) is able to surpass the performance of 0.901. The results for simulations and all rounds are in the appendix 8.3, which further supports the argument for a diverse ensemble with as many models as required to get better generalization results.

6.6.4 Conclusion of the experiment

This experiment investigated when (and if) the plateauing within the second level of a StackNet model occurs given a certain number of input models. It demonstrated that given a certain number of randomized simulations and Meta learners built on random subsets of level 1 models, the plateauing of the performance does occur and performance uplift past the 11^{th} model is incrementally minimal in comparison to the uplift occurring in the first 10 entered input models. Performance also starts to drop after a certain point.

Part of the input level 1 models included those used in 6.5. Interestingly, on average, for any randomly constructed input size dataset for the Meta learner, the performance of the ensemble is not able to surpass this which was manually tuned in 6.5. However there have been a few rounds of ensemble sizes of less than 15 that express a superior performance.

There is evidence to conclude that simply increasing the size of the ensemble will not give better generalizations results. Instead effort is required to generate diverse models and the optimal number of models needs to be specified based on cross validation results, potentially along with feature selection techniques as suggested by [Zhou et al. 2002].

6.7 Future Work

The StackNet model will be as powerful (in terms of accuracy) as the strength of the algorithms that is consisted of. Including more algorithms such as the award winning xgboost would greatly improve the overall performance. Additionally compatibility with some of the already prominent Java packages in machine learning such WEKA and RankLib would increase the reach of different algorithmic families and will add diversity to the StackNet's solutions.

The model would benefit for more data pre-processing steps (apart from scaling) to be part of the spectrum of the available hyper parameters. Feature selection or feature elimination algorithms could be invoked in a similar way inside the *fit()* and *predict()* methods . Other additions could generate unsupervised features from the raw data (such as PCA and SVD) becoming themselves hyper parameters of the model. Other parameters' additions could include subsampling methods, providing variables' importance and different regularizations methods for deciding the weight for each model.

For the meantime the StackNet model supports only classification (although it does accept in its core regressors as input-neurons). as opposed to forecasts / regressions The reason is that regression data, especially response variables that are too dependent on time (such as sales or demand) would require a cross validation framework that during the training process respects the time order of the data, making the use of StackNet not optimal as it primarily relies on the unbiased cross validation estimates generated in each part of the k-fold paradigm. On the other hand making one split based on time would mean that not many consecutive levels could be built with adequate data or the volume of the input data would have to be much bigger in order to account for the constant sub splitting of the data. To extend StackNet to include regression, the validation framework and how is implemented would be critical, however it is definitely a goal worth pursuing.

The model has been tested with FMCG data in this thesis and has performed well via improving the overall performance and yield significant uplift against any of the input algorithms or simple or weighted averaging of them. In addition the methodology has been used successfully in winning data modelling challenges in the NLP space such as the Dato classification challenge for detecting specific type of advertising from the contents of a website hosted in [Kaggle 2015] and in the insurance space via detecting claims for the Homesite insurance in [Kaggle 2016]. Furthermore StackNet (the software) was used by multiple top 10 solutions in its first unofficial public release for the Renthop Kaggle classification challenge in 2017[9] to best predict rental prices. It is suggested that the methodology is extended to more diverse problems like image and sound classification.

This ensemble framework allows many models to be combined, each one with its own specifications to order to achieve a more generalizable outcome. It is expected that this type of Meta model can be very complicated (or *blackbox*) to extract comprehensible information about the data. At the same time, even though both training and scoring can happen to some extend in parallel, it is natural that given the size of the ensemble the computational cost may be high especially when considering productionzing such approaches for large scale applications. It would be therefore advisable to extract the learnings of such process and compress it into simpler solutions – in other words go from a complicated (possibly computationally very expensive) model back to much simpler one, while maintaining a sensible level of accuracy.

[9] The main blog is here : https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries/discussion/30012

6.8 Conclusion

Chapter 6 covered the StackNet model which connotes a methodology with an architecture similar to a normal feed forward neural network that makes use of stacked generalization in multiple levels in order to combine many algorithms and improve accuracy in many typical machine learning tasks that occur within the recommendation space and beyond. The initial implementation of the model is in Java because it is deemed to have certain advantages over other languages taking into account, speed, safety, popularity, platform compatibility and accessibility.

The mathematical formulation of the model shares many similarity with that of the neural networks and it can be run with two different models, one of which assumes direct connection of each model layer only with the next geographical layer and another mode (also called *restacking*) that assumes each layer's neuron is connected with all previous layers' neurons before invoking activation. The main objective of the second mode is to counter the drawback of the training process that assumes each layer is activated only once (i.e there is only one model iteration).

The training method of the model follows the principles of stacked generalization that assumes the data need to be split so that only the predictions in the hold out data are carried over in the next modelling phase. That model, in order to address the re-usability of data and the loss of unnecessary otherwise useful information, performs a k-fold cross validation in order to make certain that all the original input data is scored and pushed forward as features (or neuron outputs) in the next layers.

The objects of the model have been described and they all follow a similar structure including an initialization step, followed by a phase where a different number of hyper parameters may be set to improve each individual model's performance. Many algorithmic families are being represented in the StackNet model including Tree based algorithms, neural networks, LibFM, linear models, K nearest neighbours, kernels based methods, Naïve Bayes and more to be added other time. All models are trained using either typical Java data objects - to address compatibility and accessibility - or as other data types (dense to sparse) to address other needs regarding performance and memory optimization. Pre-processing steps like scaling have been added into the hyper parameters space of any algorithm along with many other attributes.

Furthermore different structures of StackNet are applied to the YearPredictionMSD Data Set to predict if a given song was created before or after 2002, demonstrating the uplift in terms of AUC and loglikelood occurring via this approach through the various levels of the training process. The sample code provided covers many aspects of the StackNet model and shows the impact they have on the overall outcome.

Finally it is proposed that the model extends each arsenal of available algorithms via adding compatibility with other prominent machine learning packages or award winning implementations of different algorithms, add more data pre-processing steps in its hyper parameter spectrum and is used in many other possible diverse classification problems outside the recommendations space.

7. Conclusion and future work

This chapter reviews the finding of this thesis based on the 4 different experiments that comprises it and also provides suggestions for future work and development plans to improve the work even further.

7.1 Conclusion

Formerly the thesis has utilized four different experiments all within the scope of improving recommender systems, where each one provides unique but complementary elements towards this goal.

Univariate Analysis of the Dataset: Chapter three gave an overview of the available data (for research) that dunnhumby owns and comprises of millions of customer transactions for the course of two years for a specific retailer. The available data source include both customer transactions, demographic details and product level information data.

Sequentially the training and test data used is formulated respecting the time order ensuring that past data is used to predict future data. Additionally a number of features are created and expressed in tandem with the target variable which simply connotes whether the customers bought a specific item next week given a number of personal transactions over the past 52 weeks. Three main factors were identified as the most prominent drivers for deriving such features, namely customer based features, product features and contextual data such as time or week number. Combinations of these proved to be the most indicative features for predicting the response variable. Exploiting the product hierarchy of the item space (such as department and manufacturers) described information possibly not captured by a direct customer-to-item relationship

The relationship of the features in respect to the target variable was not always deemed as linear, therefore a brute-force optimized binning algorithm was introduced and utilized to capture such nonlinearities and uncover how much unique information each variable yields in respect to the target variable and a ranking was provided to describe the most prominent feature families.

To further promote the understanding in the FMCG recommendation field, the thesis provided an illustrative catalogue of the most representative families of such features along with descriptions regarding their derivations.

Meta-modelling to predict *top K* products: One of the main goals of the recommender systems in an FMCG environment is to improve recommendations via focusing on retention of loyal customers (that drive the most income) and also reward their loyalty via offering more relevant and personalized recommendations., therefore optimizing for *top K* precision is often used in marketing (where K is normally a small number between 5 and 20) The available data as described in the previous chapter demonstrated significant linear and non-linear relationship, making it technically difficult to identify an algorithm that could easily excel in both without significant pre-processing.

To improve precision for the *top K* products of each customer and leverage the different relationships inherent within the data, ensemble modelling was used and specifically Wolpert's stacked generalization. The training data was split into 2 parts (training and validation) where various repressors and classifiers were built using the training data and predictions were made for the validation and test data (which was the following -i.e. future - week). Gradient boosting machines and neural networks seemed to perform the best in maximizing precision and overall discrimination (as measured by highest AUC).

Furthermore, after combining (or stacking) all predictions in the validation data, forming a new training dataset, a random forest classifier was used to train on this data achieving higher precision @5, 10 and 20, as well as better overall AUC, against some field-base benchmarks (such as customer's frequency of purchase of specific products and product popularity), all previous individual models that contributed to the stacking model, and a simple average as well as a rank-transformed average.

Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment: Chapter 5 examined a hybrid recommender system to improve accuracy of predictions for whether the customer of a retailer will buy again a recommended product assuming an irregular environment. The irregular environment was defined by different customer, different offered products and different time periods between the train and the test data. More specifically the overall aim was to maximize AUC for whether a customer will buy again a product within 2 months after having redeemed a coupon for it. Therefore the evaluation

metric measured the ability of the recommender system to suggest offers that are capable of creating a habit.

Finding a cross validation methodology suitable for tis irregular environment was critical in obtaining reliable estimates for the test data, since the training data was consisted of transactions of different customers and different offers than the test data. Additionally the test data was chronologically placed in the future. Three different cross validation methodologies were examined. The first methodology measured AUC after splitting the train data randomly , but stratified based on offer ensuring all offers were proportionally equally populated in all folds of the validation procedure. The second method used N-1 offers to build a model and measure AUC for the offer left aside for validation. This process was repeat N times and the average AUC was retrieved. The reasoning for this method was derived form fact that the test data included different offers than the train data, therefore an ideal model should be able to predict offers that were not included in its training process. The last method connotes the continuation of the second method where all predictions from all offers are concatenated into a single frame before calculating the overall AUC across all predicted offers for all samples. Ultimately the third methodology used the average of this holistic AUC along with the AUC of the second methodology. The intuition for the last method was based on the fact that the offers in the training data boasted different propensities and a sample had to be comparable not only within the offer but against any other offer. Given a set of features derived from the transactional history and a logistic regression model, this last methodology performed best in the test data, demonstrating a smaller gap between the cross validation and test results.

Based on the last cross validation methodology, 2 different approaches/strategies were formulated. One content-based and another based on collaborative filtering. The first strategy assumed a direct or indirect relationship of a customer with the recommended product. This approach used ridge regression fitted using the future quantity of items bought as the response variable, creating a number of different features based on the customers' transactional history. The second strategy attempted to match the shopping habits of customers that were offered the products with another group of customer that had bought these items prior to the sending of the coupon. It utilized gradient boosted trees to predict the natural logarithm of the number of times the customers bought the product 90 days before the actual coupon was sent. The features for this approach were more generic about the customer and not in relations with specific products. Unsupervised features based on neural networks were used to create summaries of different past customer activity.

The combination of the two approaches yielded the best results in the test data. Their merging was challenging because the first model was fitted to a regression response variable (e.g. the quantity) and the other approach natural logarithm of the counts the customer bought the item in the past. AUC is focused in the ranking of the prediction array, therefore all predictions from both approaches were transformed into ranks and were weighted equally to achieve the best AUC in the test data.

The StackNet Model: Chapter 6 described the properties of the StackNet model – a new Meta modelling methodology that utilizes Wolpert's stacked generalization of combining multiple models assuming a feedforward neural network architecture. Although the model was described formally for the first time in the current thesis, online references of the term has been used in numerous predictive modelling competition where such methodology was deemed the winner.

The model shares similar properties with a simple multilayer perceptron type of neural network, where each perceptron may be replaced with any machine learning algorithm, regressor or classifier. The transformation function is no longer needed as it is now inherent to the selected algorithm. The methodology is implemented in the Java programming language because it was deemed a valid trade-off between, speed, safety, compatibility and popularity.

There two available modes referring to the type of connections each layer has with the previous ones. The normal mode assumes that each layer's neurons (or algorithms) takes as input the predictions of all algorithms in the direct previous layer. The second mode (called restacking) allows a layer to receive predictions from all previous neurons in all preceding layers, including the input data. The reasoning for the existence of the second mode is the fact that the training of the model occurs in one epoch so the model does not have the chance to revisit the initial data unless it is forced, allowing it yield extra information if any. Irrespective of which mode gets activated, all the models in a layer can be run in parallel to facilitate faster convergence.

The created software supports many different input data formats to address the need for sparsity, performance and compatibility within the native Java code. Additionally many algorithms have been written from scratch to address the needs to a scalable efficient software in Java. Most commonly-used algorithmic families have been included such as tree-based methods, neural networks, linear methods, kernels, nearest neighbours, factorization machines and naïve Bayes. Among the implementations is Rendle's LibFm and the award winning Liblinear.

Finally different StackNet architectures are being tested on the YearPredictionMSD Data Set in order to predict if a given song is created before or after 2002. It is shown that adding multiple modelling levels while tweaking the StackNet's options can facilitate better performance.

7.2 Future Work

Univariate Analysis of the Dataset: Chapter 3 utilized mostly the transactional data to give insight about the factors that connect customers with future product purchases. In addition to transactional data, it also used some demographics of the customers and some very basic contextual data such as time of visit. Therefore further insight could be gained via examining other potential factors in the same space.

The examination of the dataset demonstrated significant *linear* (such as how many times a customer has bought an item in the past) and *nonlinear* (such as when was the last time a customer bought an item versus every how often the item is being bought) relationships in respect to future purchases . To assess the predictability of the variables, an optimized binning methodology was utilized. While the latter ensures that the nonlinear relationships of the features (with respect to the response variable) are captured, still it does take away from the linear relationships. Another methodology that may well be utilized is the [MARS 1991] model that could potentially examine the variables not just in a univariate but also a multivariate context in order to give a more fair assessment for the predictability of a single variable.

Ultimately a similar features examination approach could be implemented in other retailer environments, not just the grocery market in order to compare how consistently the discovered relationships are present in different datasets.

Meta-modelling to predict *top K* products: This chapter combined various algorithms to improve performance for the *top k* products for each customer using the available feature space as developed in the previous chapter. It would be of great interest to compare whether there is performance uplift via enriching the data set with additional features such as contextual data or whether additional transactional history prior to one year would improve results even further.

The number of different models used in the ensemble required significant amount of time to tune and find the best hyper parameters. It would be vital for future performance optimization

to discover a reliable and at the same time not very time-consuming way to locate a good set of such hyper parameters that maximize the performance of any given response variable.

In addition to saving computational time, the experiment was run with an ensemble of 10 models, however bigger and more diverse ensembles could be exploited to improve performance for the given data. Additionally unsupervised methods (such as principal components analysis or singular value decomposition) could have been considered as a way to seize the data from different angles and train various classifiers with the new input data to generate new information to then feed onto for the Meta model.

Ultimately a similar Meta modelling methodology could be implemented in other retailer markets besides the grocery one in order to compare how well such approach would hold with different input data.

Hybrid method to predict repeated, promotion-driven product purchases in an irregular testing environment: The implemented methodology was formed in order to optimize AUC or in other words to maximize the overall discrimination of items (re)bought after sending a coupon or not. In most situation the retailer has a limited number of coupons to consider prior to sending the offers , therefore metric that take into account this information such as precision@k per product would be a good alternate way to approach the problem and it would have been interesting to compare whether such approach could work for this particular problem. At the same time considering each customer could receive a certain number of coupons reversing the previous problem, thereby maximizing precision per customer (instead of per product) would also be noteworthy.

A Meta-modelling as detailed in previous chapter was not deemed feasible to further improve the score in this challenge because the test data were well ahead in the future and very different distribution-wise with the training data, even when using a one-offer-out cross validation approach. Comparing with the potential uplift from other experiment present in the thesis (chapter 4 &6), different ensemble methods could be developed that create unbiased cross-validation estimates that respect the time-element and therefore can be used to improve predictions through Meta modelling.

The StackNet Model: The StackNet model will be as powerful as the algorithms available at its disposal to solve problems. In order to be more competitive and useful to the scientific community it will have to integrate more algorithms and their (award winning)

implementations, especially those made available in the Java programming language such the ones contained in the Weka suite or the H2O software package for predictive analytics.

Data pre-processing steps such as features selection, feature elimination , providing variables' importance , model selection and hyper parameter tuning could be additional improvements that would make the sued of the software more autonomous, making it easier for the data science community to work independently on one platform.

Significant improvement could be achieved via extending the current methodology (made available for classification tasks) to general purpose regression problems. In order to do so efficiently a cross-validation framework needs to be implemented that takes into account the time element if presented in the data.

Depending on how big or how deep a StackNet model may be, the resulting ensemble solution can be very computationally expensive as well as hard to derive insightful information from the data especially when considering large-scale , (possibly) real time solutions. To counter this, it is suggested that a process gets formulated that extracts the predictive (or insightful) elements from a complicated model back to a much simpler one, while maintaining a good level of the initial accuracy.

Ultimately the current methodology could be extended to other machine learning fields outside the recommendation field, in problems as diverse as image or sound classification as its already successful implementation in the fields of insurance and natural language processing make a case for ability to extend this methodology to any machine learning task.

8. Appendices

8.1 Table of full univariate results for the first experiment

Table 8.1 illustrates the full list of results for the variables' predictive power for defining future purchases that were considered when scrutinizing *the complete Journey* dataset. It displays the AUC as well as Information Gain after binning all continuous variables. It also displays which level of the product hierarchy each variable corresponds too as well as it provides a short description for each one of them.

Table 8-1 : Full Univariate results of binned variables measuring AUC and I-Gain for experiment 1

Feature name	Feature Description	C	P	D	M	AUC	Igain
frequency26	Number of baskets the customer included the product in last 26 weeks	✓	✓			0.775	0.0039
frequency39	Number of baskets the customer included the product in last 39 weeks	✓	✓			0.775	0.0039
frequency52	Number of baskets the customer included the product in last 52 weeks	✓	✓			0.775	0.0038
frequency13	Number of baskets the customer included the product in last 13 weeks	✓	✓			0.775	0.0039
cycle_vs_lastbought	Average cycle (52 weeks) minus days ago since last bought the product	✓	✓			0.775	0.0036
average_cycle52	Every how many days the customer bought the product in last 52 weeks	✓	✓			0.774	0.0035
last_day_bought	Days from the target week since the customer last bought the product	✓	✓			0.774	0.0033
average_cycle39	Every how many days the customer bought the product in last 39 weeks	✓	✓			0.766	0.0035
average_cycle26	Every how many days the customer bought the product in last 26 weeks	✓	✓			0.747	0.0035
popularity13	Number of baskets the product appeared in last 13 weeks		✓			0.747	0.0011
popularity26	Number of baskets the product appeared in last 26 weeks		✓			0.742	0.0011
popularity39	Number of baskets the product appeared in last 39 weeks		✓			0.739	0.0010
popularity52	Number of baskets the product appeared in last 52 weeks		✓			0.735	0.0010
average_cycle13	Every how many days the customer bought the product in last 13 weeks	✓	✓			0.709	0.0032
frequencies_decay	frequency52 divided by frequency13	✓	✓			0.709	0.0029
frequency13man	Same as frequency13 but for "manufacturer"	✓			✓	0.708	0.0007
frequency26man	Same as frequency26 but for "manufacturer"	✓			✓	0.707	0.0007
frequency39man	Same as frequency39 but for "manufacturer"	✓			✓	0.704	0.0006
frequency52man	Same as frequency52 but for "manufacturer"	✓			✓	0.702	0.0006

average_cycle52man	Same as average_cycle52 but for "manufacturer"	✓		✓	0.698	0.0006
average_cycle39man	Same as average_cycle39 but for "manufacturer"	✓		✓	0.695	0.0006
average_cycle26man	Same as average_cycle26 but for "manufacturer"	✓		✓	0.695	0.0006
most_trialled	Number of customer who bought the item 1st time the previous week		✓		0.687	0.0008
average_cycle13man	Same as average_cycle13 but for "manufacturer"	✓		✓	0.686	0.0006
frequenciesman_decay	frequency52man divided by frequency13man	✓		✓	0.683	0.0006
productsbought13	Total number of products the customer bought in last 13 weeks	✓			0.632	0.0003
productsbought26	Total number of products the customer bought in last 26 weeks	✓			0.632	0.0003
productsbought39	Total number of products the customer bought in last 39 weeks	✓			0.630	0.0002
distinct_item	Distinct number of products the customer bought in last 52 weeks	✓			0.625	0.0002
productsbought52	Total number of products the customer bought in last 52 weeks	✓			0.625	0.0002
distinct_MANUFACTURER	same as distinct_item but for "manufacturer"	✓			0.620	0.0002
distinct_DEPARTMENT	same as distinct_item but for "department"	✓			0.591	0.0001
manpopularity52	same as popularity52 but for "manufacturer"			✓	0.590	0.0001
popularity_decay	popularity52 divided by popularity13		✓		0.588	0.0001
manpopularity39	same as popularity39 but for "manufacturer"			✓	0.586	0.0001
manpopularity13	same as popularity13 but for "manufacturer"			✓	0.586	0.0001
manpopularity26	same as popularity26 but for "manufacturer"			✓	0.585	0.0001
frequency26dep	Same as frequency26 but for "department"	✓	✓		0.584	0.0001
frequency39dep	Same as frequency39 but for "department"	✓	✓		0.583	0.0001
frequency13dep	Same as frequency13 but for "department"	✓	✓		0.583	0.0001
frequency52dep	Same as frequency52 but for "department"	✓	✓		0.579	0.0001
visits26	Number of distinct days the customer visited in last 26 weeks	✓			0.577	0.0001
visits13	Number of distinct days the customer visited in last 13 weeks	✓			0.577	0.0001
transactions_withdiscount	Total number of transactions with discount in last 52 weeks	✓			0.577	0.0001
visits39	Number of distinct days the customer visited in last 39 weeks	✓			0.574	0.0001
deppopularity13	same as popularity13 but for "department"		✓		0.573	0.0001
deppopularity26	same as popularity26 but for "department"		✓		0.573	0.0001
deppopularity39	same as popularity39 but for "department"		✓		0.573	0.0001
deppopularity_decay	deppopularity52 divided by deppopularity13		✓		0.573	0.0001
visits52	Number of distinct days the customer visited in last 52 weeks	✓			0.569	0.0001
transactions_withdiscountman	Number of times the manufacturer was sold with discount in 52 weeks			✓	0.567	0.0001
transactions_withdiscountdep	Number of times the department was sold with discount in 52 weeks		✓		0.565	0.0001
manpopularity_decay	manpopularity52 divided by manpopularity13			✓	0.563	0.0001
count_newitems	Number of products the customer bought last week for the 1st time	✓			0.562	0.0001
frequenciesdep_decay	frequency52dep divided by frequency13dep	✓	✓		0.559	0.0001
average_cycle52dep	Same as average_cycle52 but for "department"	✓	✓		0.559	0.0001

HH_COMP_DESC	Household status	✓			0.557	0.0001
INCOME_DESC	Household income band	✓			0.557	0.0001
average_cycle39dep	Same as average_cycle39 but for "department"	✓		✓	0.556	0.0000
AGE_DESC	Household Age Band	✓			0.556	0.0001
KID_CATEGORY_DESC	Household's kid category description	✓			0.555	0.0001
average_cycle26dep	Same as average_cycle26 but for "department"	✓		✓	0.555	0.0000
MARITAL_STATUS_CODE	Household's Marital Status	✓			0.553	0.0000
average_cycle13dep	Same as average_cycle13 but for "department"	✓		✓	0.552	0.0000
HOMEOWNER_DESC	Household's homeowner status	✓			0.551	0.0000
average_spendingitem	Average spent on a product in last 52 weeks		✓		0.542	0.0000
deppopularity52	same as popularity52 but for department			✓	0.541	0.0000
HOUSEHOLD_SIZE_DESC	Household Size band	✓			0.540	0.0000
average_discount	Average discount per product in basket in last 52 weeks	✓			0.540	0.0000
average_discountitem	Number of times the product was sold with discount in last 52 weeks		✓		0.537	0.0001
transactions_withdiscountitem	Number products the customer bought with discount in last 52 weeks	✓			0.535	0.0001
visits_decay	visits52 divided by visits13	✓			0.535	0.0000
average_spending	Average spending per product in basket in last 52 weeks	✓			0.533	0.0000
average_quantity	Average quantity per product in basket in last 52 weeks	✓			0.531	0.0000
TRANS_TIME	Time in hours where 12 am is '00' and 11pm is '23' (24 distinct values)				0.529	0.0000

8.2 Additional charts of the features in experiment 1

The following charts display additional information for some of the variables not analytically covered in the first experiment.

8.2.1 Marital Status

Figure 8.1 displays the marital status in relation to the probability of buying any item next week. Married people have higher probability buying any item as can be seen in the graph below:

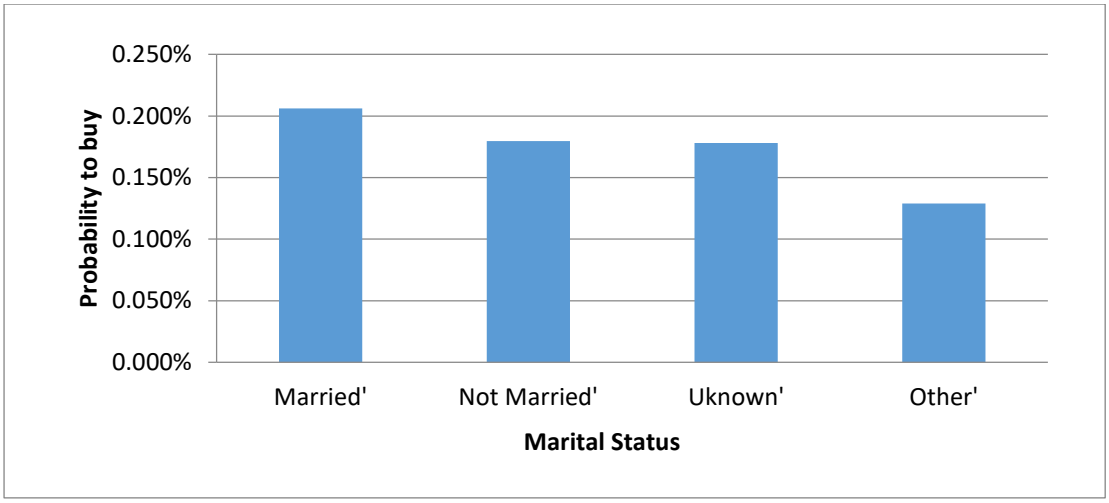

Figure 8.1 : Marital status versus target variable

8.2.2 Household composition

Figure 8.2 illustrates the different household composition types along with the probability to buy any product in the future week. Smaller families seem to have higher probability to buy any given product:

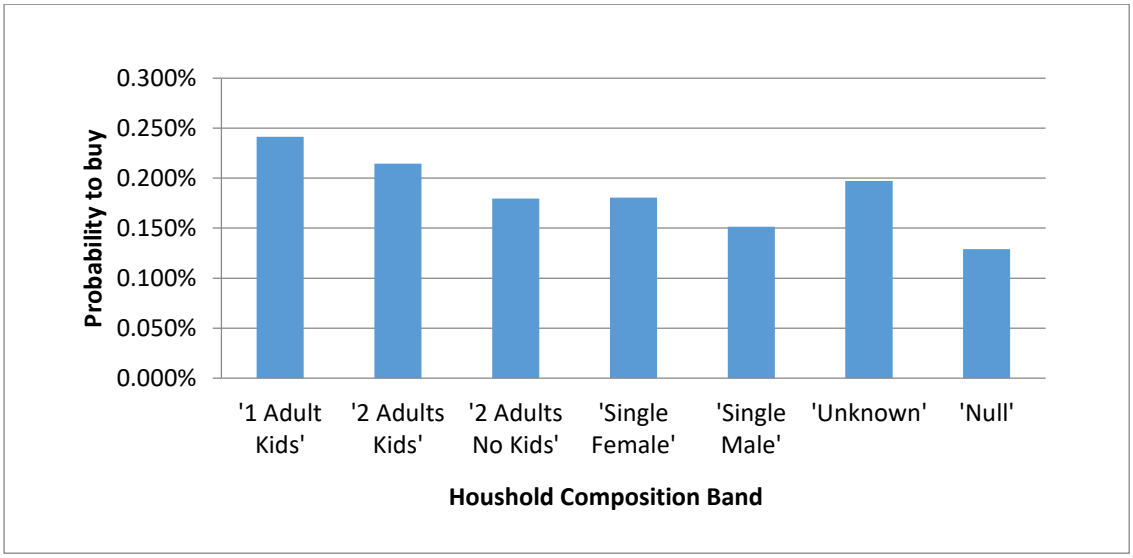

Figure 8.2 : Household composition type versus target

8.2.3 Household Size

Figure 8.3 displays the Household size versus the target variable:

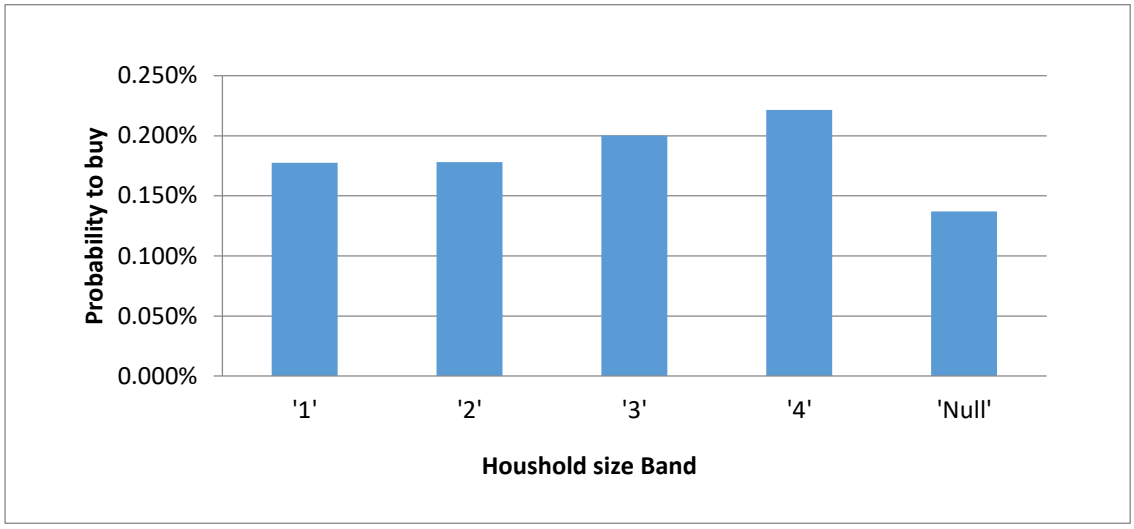

Figure 8.3 : Household size and target variable

8.2.4 Kids' Category number

The number of kids in the family seems to be positively correlated with the probability to buy a given item in the future week as can be visualized through 8.4:

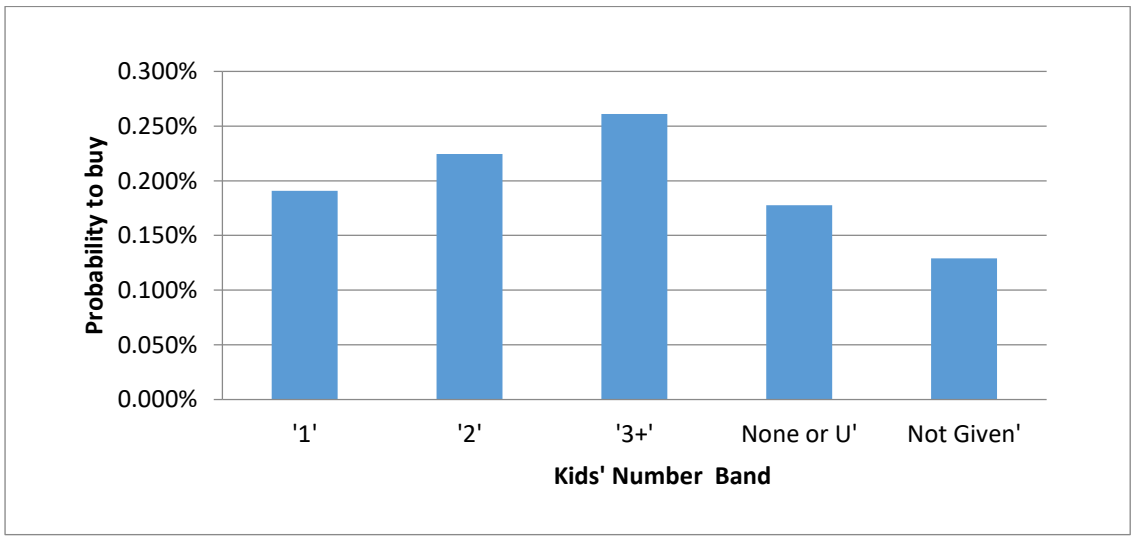

Figure 8.4 : Kids' number and target variable

8.2.5 Average Cycle of buying an item in last 52 weeks

The following chart 8.5 portrays the probability to buy an item in the future week given the average number of days it takes for that item to be bought by the customer as measured in the last 52 weeks. The graph is peculiarly nonlinear, but this is primarily because the very small numbers imply that customer bought the item many times in a small period of time (e.g. stocked up), therefore there is less need for future purchases:

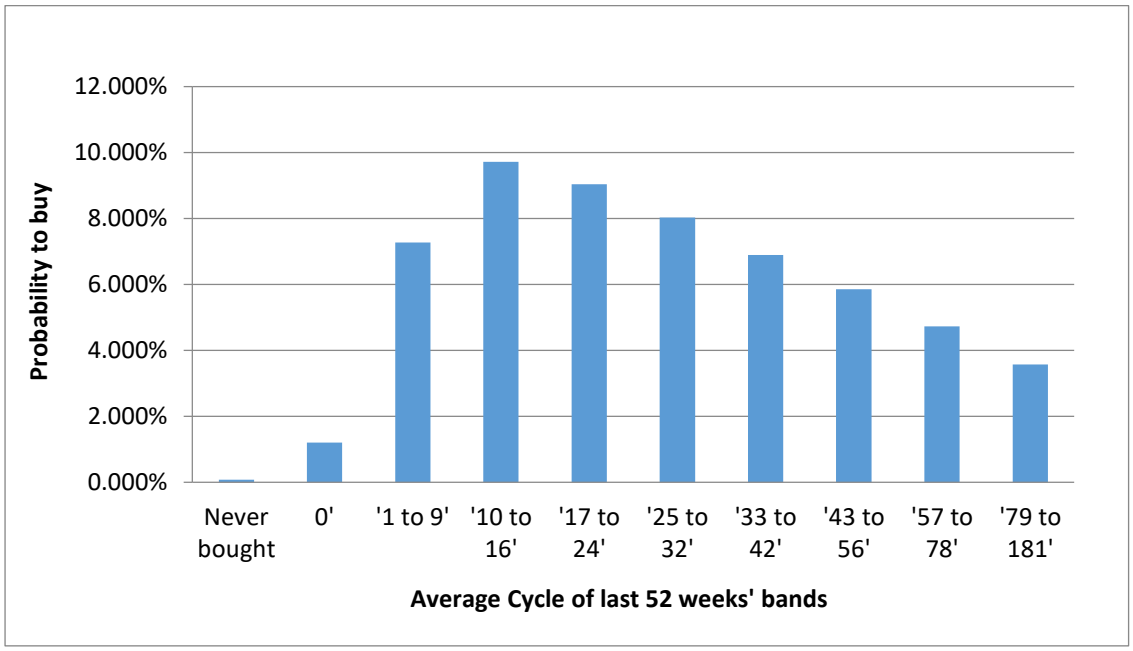

Figure 8.5 : Average number of days to buy the item in the last 52 weeks versus target

8.2.6 Average Cycle of buying an item in last 52 weeks

Figure 8.6 shows the number of times a customer has bought an item that comes from the same manufacturer as the item to be considered for a possible future purchase. The relationship may not be as linear as someone would expect.

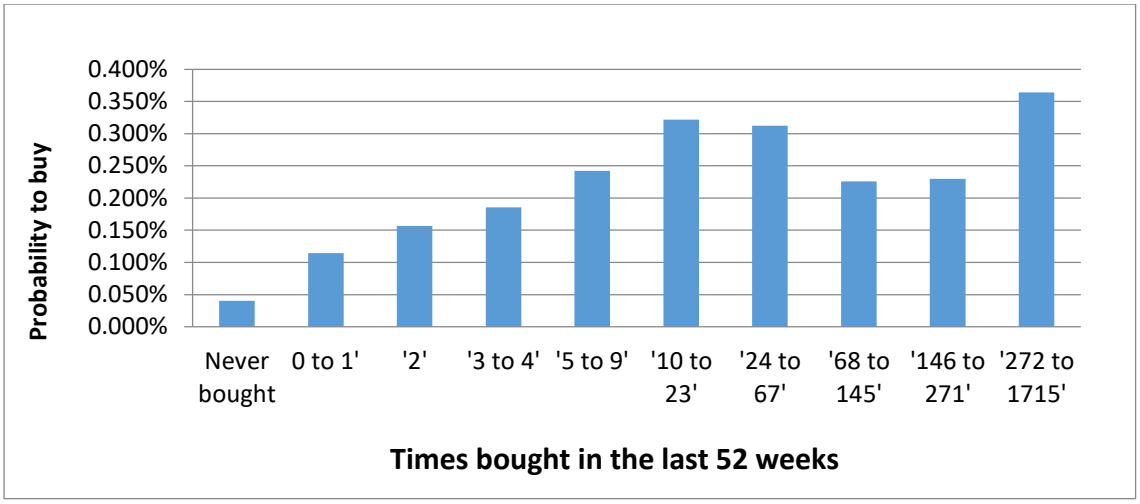

Figure 8.6 : Times bought from same Manufacturer versus target

8.2.7 Total items bought in last 52 weeks

The next feature can be seen as a measure of how loyal a customer is, given the number of total units he/she has purchased over the last 52 weeks. The relationship is fairly linear with the probability to buy any given item in the next week as evident by 8.7:

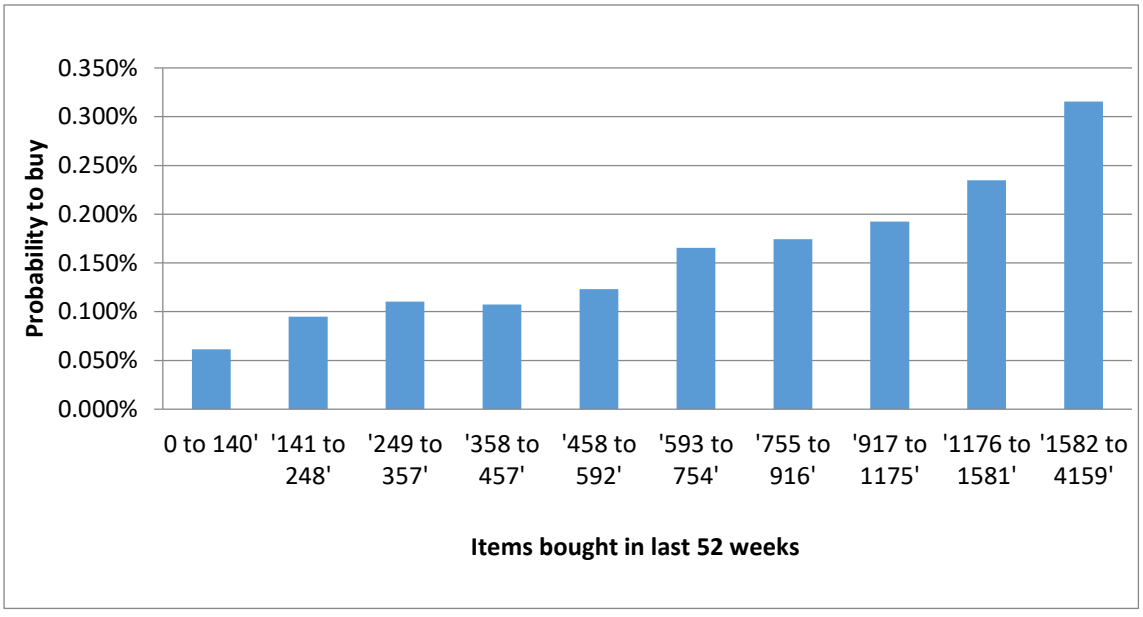

Figure 8.7 : Total items bought versus Target

8.2.8 Total transaction with any discount in last 52 weeks

This is another customer-level variable derived from the purchase history of the customers and demonstrates how many times they leverage promotional opportunities. It can also be seen as form of loyalty and the relationship is fairly linear with the target variable as portrayed in 8.8.

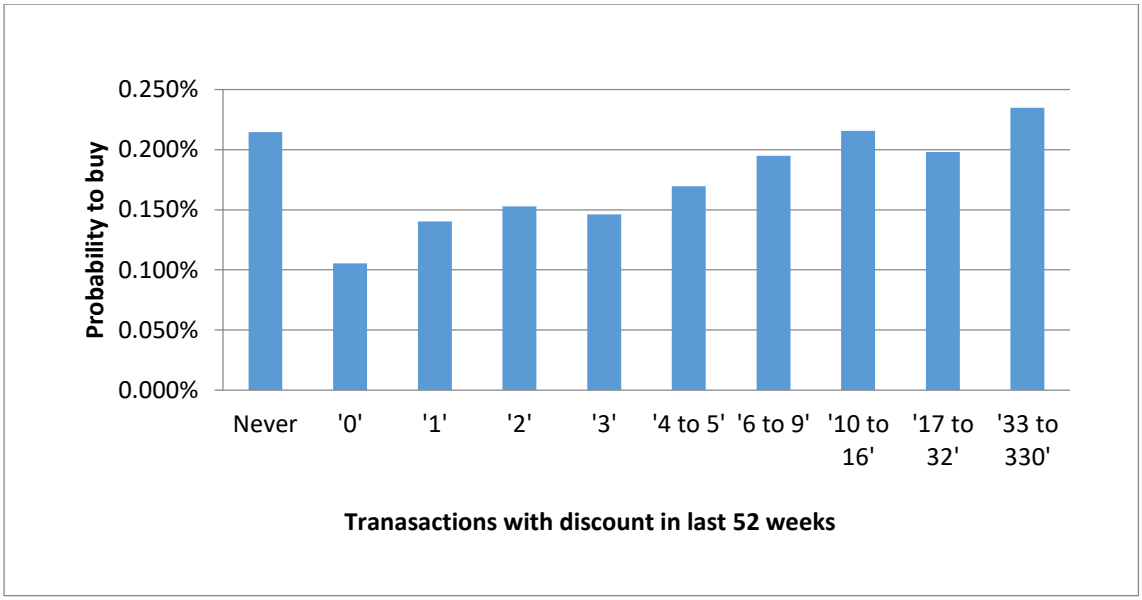

Figure 8.8 : Total transatcions with discount vs target

8.2.9 Average Spend per item in last 52 weeks

Customers who generally buy cheaper (on average) items they tend to have higher probability to buy any given item. This is easy to conceive as most items of the retailer fall within a certain price-range. This relationship is also exposed in the following chart 8.9:

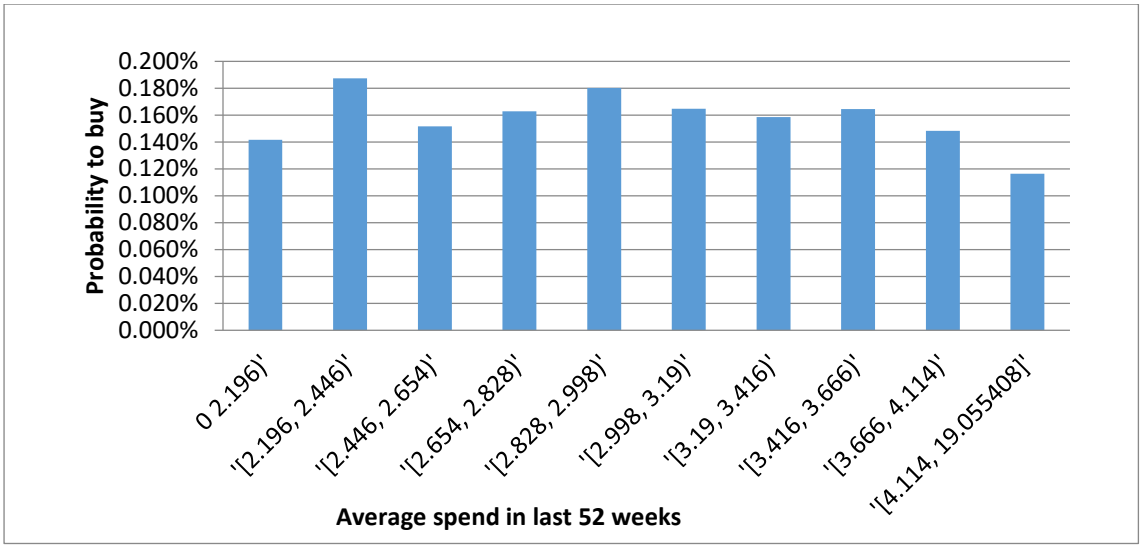

Figure 8.9 : Average Spend per item in last 52 weeks versus target

8.3 All simulations and rounds' AUC results for plateauing experiment in 6.6

Table 8-2 demonstrates the full results from all 50 simulations run as part of the experiment in 6.6.

Table 8-2: Simulations' rounds results and cross validation AUC

Round	Ensemble input size	cv AUC
1	1	0.888939
1	2	0.891189
1	3	0.893003
1	4	0.892853
1	5	0.89341
1	6	0.896835
1	7	0.898658
1	8	0.898369
1	9	0.898132
1	10	0.897842
1	11	0.897973
1	12	0.898314
1	13	0.897963
1	14	0.898653
1	15	0.898902
1	16	0.898717
1	17	0.898391
1	18	0.898497
1	19	0.898593
1	20	0.898617
1	21	0.898865
1	22	0.898704
1	23	0.899372
1	24	0.899569
1	25	0.899241
1	26	0.899624
1	27	0.899869
1	28	0.899832
1	29	0.899717
1	30	0.899326
1	31	0.89957
1	32	0.899765
1	33	0.899766
1	34	0.899554
1	35	0.899617
1	36	0.899621
2	1	0.874569
2	2	0.881201
2	3	0.889765
2	4	0.888908
2	5	0.889636
2	6	0.896584
2	7	0.895865
2	8	0.896044
2	9	0.895862
2	10	0.896291
2	11	0.896368

2	12	0.897413
2	13	0.898765
2	14	0.899406
2	15	0.89918
2	16	0.898845
2	17	0.898625
2	18	0.899346
2	19	0.899349
2	20	0.898949
2	21	0.89922
2	22	0.898865
2	23	0.898673
2	24	0.899147
2	25	0.899122
2	26	0.899431
2	27	0.899314
2	28	0.89976
2	29	0.899804
2	30	0.899595
2	31	0.900055
2	32	0.899817
2	33	0.899866
2	34	0.899484
2	35	0.899687
2	36	0.899508
3	1	0.873868
3	2	0.888107
3	3	0.891835
3	4	0.891114
3	5	0.898352
3	6	0.898642
3	7	0.897879
3	8	0.897735
3	9	0.898099
3	10	0.898342
3	11	0.89953
3	12	0.899614
3	13	0.899664
3	14	0.899544
3	15	0.899902
3	16	0.900007
3	17	0.900496
3	18	0.900331
3	19	0.900523
3	20	0.900439
3	21	0.900021
3	22	0.900281
3	23	0.900068
3	24	0.900167
3	25	0.899912
3	26	0.900033
3	27	0.899929
3	28	0.900233
3	29	0.899957
3	30	0.899878
3	31	0.899723
3	32	0.89971
3	33	0.899821
3	34	0.899653
3	35	0.899355
3	36	0.899739
4	1	0.888461
4	2	0.891069
4	3	0.892528

4	4	0.893014
4	5	0.893691
4	6	0.89385
4	7	0.896744
4	8	0.896221
4	9	0.896145
4	10	0.896123
4	11	0.896187
4	12	0.896387
4	13	0.896564
4	14	0.898482
4	15	0.899003
4	16	0.899233
4	17	0.899099
4	18	0.898868
4	19	0.898805
4	20	0.898956
4	21	0.898747
4	22	0.89922
4	23	0.898854
4	24	0.899019
4	25	0.899102
4	26	0.89888
4	27	0.899008
4	28	0.898898
4	29	0.89932
4	30	0.899476
4	31	0.899206
4	32	0.899224
4	33	0.899389
4	34	0.899342
4	35	0.899448
4	36	0.899492
5	1	0.85652
5	2	0.896044
5	3	0.896312
5	4	0.89626
5	5	0.895959
5	6	0.89618
5	7	0.89669
5	8	0.896578
5	9	0.896581
5	10	0.896764
5	11	0.896483
5	12	0.898161
5	13	0.897778
5	14	0.898044
5	15	0.898216
5	16	0.898143
5	17	0.898279
5	18	0.898322
5	19	0.898954
5	20	0.899415
5	21	0.899378
5	22	0.899144
5	23	0.899165
5	24	0.899212
5	25	0.899171
5	26	0.89917
5	27	0.899081
5	28	0.899231
5	29	0.899452
5	30	0.899586
5	31	0.899036

5	32	0.899486
5	33	0.898991
5	34	0.899114
5	35	0.899617
5	36	0.89963
6	1	0.888939
6	2	0.88915
6	3	0.890435
6	4	0.893558
6	5	0.893667
6	6	0.894147
6	7	0.894348
6	8	0.895752
6	9	0.895482
6	10	0.896001
6	11	0.896147
6	12	0.89599
6	13	0.898564
6	14	0.898534
6	15	0.898815
6	16	0.899104
6	17	0.899458
6	18	0.899077
6	19	0.899896
6	20	0.900086
6	21	0.900073
6	22	0.899927
6	23	0.900061
6	24	0.899818
6	25	0.899932
6	26	0.89995
6	27	0.899857
6	28	0.899899
6	29	0.899963
6	30	0.899831
6	31	0.900014
6	32	0.89983
6	33	0.899926
6	34	0.899588
6	35	0.899587
6	36	0.899429
7	1	0.889559
7	2	0.893755
7	3	0.895564
7	4	0.896745
7	5	0.899096
7	6	0.899252
7	7	0.899242
7	8	0.899127
7	9	0.898889
7	10	0.899079
7	11	0.899094
7	12	0.899126
7	13	0.898978
7	14	0.898664
7	15	0.898573
7	16	0.898593
7	17	0.898657
7	18	0.898623
7	19	0.898703
7	20	0.898675
7	21	0.898597
7	22	0.898682
7	23	0.898761

7	24	0.899061
7	25	0.899277
7	26	0.89891
7	27	0.898925
7	28	0.899146
7	29	0.899742
7	30	0.899631
7	31	0.899787
7	32	0.899854
7	33	0.899819
7	34	0.899592
7	35	0.89925
7	36	0.899686
8	1	0.8796
8	2	0.891198
8	3	0.897057
8	4	0.899672
8	5	0.899465
8	6	0.899643
8	7	0.898777
8	8	0.898515
8	9	0.898667
8	10	0.898855
8	11	0.898433
8	12	0.898577
8	13	0.898533
8	14	0.898536
8	15	0.898771
8	16	0.898683
8	17	0.898823
8	18	0.89877
8	19	0.899235
8	20	0.898897
8	21	0.899299
8	22	0.899215
8	23	0.89908
8	24	0.899209
8	25	0.899389
8	26	0.899413
8	27	0.899346
8	28	0.899418
8	29	0.899164
8	30	0.899497
8	31	0.899342
8	32	0.899472
8	33	0.899789
8	34	0.89951
8	35	0.89965
8	36	0.899636
9	1	0.892438
9	2	0.896743
9	3	0.897281
9	4	0.897122
9	5	0.898818
9	6	0.900127
9	7	0.899783
9	8	0.900028
9	9	0.899834
9	10	0.899746
9	11	0.899785
9	12	0.899592
9	13	0.899609
9	14	0.899668
9	15	0.899815

9	16	0.89952
9	17	0.899852
9	18	0.899799
9	19	0.89961
9	20	0.899622
9	21	0.899258
9	22	0.89921
9	23	0.899424
9	24	0.899566
9	25	0.89935
9	26	0.899667
9	27	0.899751
9	28	0.899369
9	29	0.899387
9	30	0.899453
9	31	0.899769
9	32	0.899809
9	33	0.899775
9	34	0.899789
9	35	0.899813
9	36	0.899687
10	1	0.882331
10	2	0.891557
10	3	0.899555
10	4	0.899065
10	5	0.898567
10	6	0.898805
10	7	0.898721
10	8	0.898383
10	9	0.897857
10	10	0.897445
10	11	0.897676
10	12	0.898413
10	13	0.898633
10	14	0.898775
10	15	0.898673
10	16	0.898743
10	17	0.898499
10	18	0.898445
10	19	0.898332
10	20	0.898158
10	21	0.898163
10	22	0.898902
10	23	0.898924
10	24	0.899012
10	25	0.899109
10	26	0.898934
10	27	0.899249
10	28	0.898971
10	29	0.898919
10	30	0.89919
10	31	0.899012
10	32	0.898997
10	33	0.898864
10	34	0.899502
10	35	0.899611
10	36	0.899469
11	1	0.857564
11	2	0.896341
11	3	0.897131
11	4	0.897135
11	5	0.897641
11	6	0.898047
11	7	0.898269

11	8	0.898001
11	9	0.89791
11	10	0.898069
11	11	0.897891
11	12	0.899949
11	13	0.899979
11	14	0.899485
11	15	0.899401
11	16	0.899462
11	17	0.899457
11	18	0.89882
11	19	0.899163
11	20	0.899114
11	21	0.899265
11	22	0.899248
11	23	0.899225
11	24	0.899568
11	25	0.899494
11	26	0.899821
11	27	0.900096
11	28	0.899751
11	29	0.900001
11	30	0.899828
11	31	0.899826
11	32	0.89964
11	33	0.899848
11	34	0.899976
11	35	0.899843
11	36	0.899701
12	1	0.873344
12	2	0.887281
12	3	0.89069
12	4	0.890959
12	5	0.893531
12	6	0.896479
12	7	0.896111
12	8	0.896547
12	9	0.897692
12	10	0.898069
12	11	0.897989
12	12	0.899072
12	13	0.899363
12	14	0.899273
12	15	0.899136
12	16	0.900151
12	17	0.900116
12	18	0.899923
12	19	0.899741
12	20	0.899685
12	21	0.899128
12	22	0.899345
12	23	0.899281
12	24	0.899098
12	25	0.899322
12	26	0.899716
12	27	0.899379
12	28	0.899303
12	29	0.899877
12	30	0.900011
12	31	0.899763
12	32	0.89977
12	33	0.89982
12	34	0.899726
12	35	0.899613

12	36	0.899817
13	1	0.873344
13	2	0.884197
13	3	0.89405
13	4	0.893942
13	5	0.897493
13	6	0.897867
13	7	0.898177
13	8	0.898215
13	9	0.9005
13	10	0.900339
13	11	0.900796
13	12	0.901086
13	13	0.900931
13	14	0.900985
13	15	0.899771
13	16	0.899891
13	17	0.899853
13	18	0.899516
13	19	0.899816
13	20	0.899677
13	21	0.899885
13	22	0.89998
13	23	0.900213
13	24	0.899958
13	25	0.899963
13	26	0.899895
13	27	0.900054
13	28	0.900144
13	29	0.899988
13	30	0.899827
13	31	0.899664
13	32	0.899753
13	33	0.899566
13	34	0.899759
13	35	0.899575
13	36	0.899879
14	1	0.877636
14	2	0.887403
14	3	0.888201
14	4	0.893938
14	5	0.895583
14	6	0.895808
14	7	0.895918
14	8	0.897392
14	9	0.89749
14	10	0.89761
14	11	0.897824
14	12	0.898055
14	13	0.900015
14	14	0.900132
14	15	0.900215
14	16	0.899917
14	17	0.900821
14	18	0.900429
14	19	0.900591
14	20	0.90042
14	21	0.900045
14	22	0.900009
14	23	0.899743
14	24	0.899617
14	25	0.899969
14	26	0.899612
14	27	0.899664

14	28	0.899729
14	29	0.89958
14	30	0.899423
14	31	0.899711
14	32	0.899636
14	33	0.899446
14	34	0.899705
14	35	0.899883
14	36	0.899719
15	1	0.891452
15	2	0.893092
15	3	0.894886
15	4	0.899034
15	5	0.899005
15	6	0.898628
15	7	0.898714
15	8	0.898975
15	9	0.898789
15	10	0.898995
15	11	0.898645
15	12	0.89874
15	13	0.898921
15	14	0.899194
15	15	0.899276
15	16	0.899003
15	17	0.898841
15	18	0.898556
15	19	0.898832
15	20	0.898708
15	21	0.899076
15	22	0.898981
15	23	0.89889
15	24	0.898926
15	25	0.898797
15	26	0.899164
15	27	0.899034
15	28	0.899073
15	29	0.899268
15	30	0.899575
15	31	0.899532
15	32	0.899578
15	33	0.899585
15	34	0.899285
15	35	0.89931
15	36	0.899576
16	1	0.888939
16	2	0.896025
16	3	0.895895
16	4	0.898935
16	5	0.899006
16	6	0.89939
16	7	0.899338
16	8	0.899418
16	9	0.900333
16	10	0.900004
16	11	0.899915
16	12	0.900414
16	13	0.900204
16	14	0.900215
16	15	0.899983
16	16	0.899777
16	17	0.89987
16	18	0.899797
16	19	0.899328

16	20	0.899516
16	21	0.899371
16	22	0.899465
16	23	0.899673
16	24	0.899412
16	25	0.899422
16	26	0.899712
16	27	0.899514
16	28	0.899767
16	29	0.899366
16	30	0.899817
16	31	0.899418
16	32	0.899375
16	33	0.899758
16	34	0.899703
16	35	0.899344
16	36	0.899603
17	1	0.874569
17	2	0.888803
17	3	0.893404
17	4	0.896261
17	5	0.899285
17	6	0.898893
17	7	0.898533
17	8	0.89868
17	9	0.898914
17	10	0.898971
17	11	0.898934
17	12	0.898648
17	13	0.899853
17	14	0.899632
17	15	0.899659
17	16	0.899347
17	17	0.899576
17	18	0.89934
17	19	0.899265
17	20	0.899238
17	21	0.899834
17	22	0.899616
17	23	0.899545
17	24	0.899196
17	25	0.89915
17	26	0.89897
17	27	0.898857
17	28	0.899117
17	29	0.899101
17	30	0.899287
17	31	0.898835
17	32	0.899071
17	33	0.899068
17	34	0.899042
17	35	0.899689
17	36	0.899781
18	1	0.808227
18	2	0.893495
18	3	0.895597
18	4	0.896548
18	5	0.896822
18	6	0.895641
18	7	0.896355
18	8	0.898404
18	9	0.897951
18	10	0.898161
18	11	0.898053

18	12	0.898019
18	13	0.897933
18	14	0.89803
18	15	0.899574
18	16	0.899668
18	17	0.899423
18	18	0.899176
18	19	0.899247
18	20	0.899204
18	21	0.898932
18	22	0.899417
18	23	0.899707
18	24	0.899241
18	25	0.899488
18	26	0.899943
18	27	0.899784
18	28	0.899708
18	29	0.89976
18	30	0.89964
18	31	0.899585
18	32	0.899614
18	33	0.899675
18	34	0.899482
18	35	0.899769
18	36	0.899748
19	1	0.808227
19	2	0.889594
19	3	0.892304
19	4	0.894871
19	5	0.894535
19	6	0.895792
19	7	0.896101
19	8	0.896189
19	9	0.896334
19	10	0.896602
19	11	0.896487
19	12	0.896833
19	13	0.897074
19	14	0.897777
19	15	0.898383
19	16	0.898235
19	17	0.898059
19	18	0.899179
19	19	0.899404
19	20	0.899159
19	21	0.899154
19	22	0.898951
19	23	0.898864
19	24	0.898591
19	25	0.898634
19	26	0.899106
19	27	0.899271
19	28	0.899239
19	29	0.899384
19	30	0.899304
19	31	0.899318
19	32	0.899562
19	33	0.899633
19	34	0.899134
19	35	0.899349
19	36	0.899717
20	1	0.892438
20	2	0.893977
20	3	0.894211

20	4	0.894034
20	5	0.895269
20	6	0.896827
20	7	0.896633
20	8	0.896732
20	9	0.896865
20	10	0.897026
20	11	0.896921
20	12	0.897014
20	13	0.89725
20	14	0.897077
20	15	0.897519
20	16	0.897156
20	17	0.897484
20	18	0.897352
20	19	0.897239
20	20	0.897345
20	21	0.897563
20	22	0.897525
20	23	0.897448
20	24	0.897971
20	25	0.897926
20	26	0.899304
20	27	0.899132
20	28	0.898843
20	29	0.898928
20	30	0.899355
20	31	0.899455
20	32	0.899435
20	33	0.899558
20	34	0.899857
20	35	0.899856
20	36	0.89971
21	1	0.888461
21	2	0.889627
21	3	0.890176
21	4	0.892617
21	5	0.893163
21	6	0.894198
21	7	0.893836
21	8	0.895924
21	9	0.895751
21	10	0.89841
21	11	0.898692
21	12	0.900014
21	13	0.899943
21	14	0.900248
21	15	0.90021
21	16	0.900026
21	17	0.89961
21	18	0.899638
21	19	0.899823
21	20	0.899723
21	21	0.899962
21	22	0.900285
21	23	0.900107
21	24	0.900238
21	25	0.899884
21	26	0.899855
21	27	0.899844
21	28	0.899997
21	29	0.89969
21	30	0.89972
21	31	0.899496

21	32	0.899612
21	33	0.899435
21	34	0.899528
21	35	0.899609
21	36	0.89965
22	1	0.877636
22	2	0.896801
22	3	0.896944
22	4	0.896826
22	5	0.897182
22	6	0.896704
22	7	0.897903
22	8	0.897556
22	9	0.897746
22	10	0.897621
22	11	0.897967
22	12	0.897672
22	13	0.899877
22	14	0.900038
22	15	0.900162
22	16	0.899912
22	17	0.899881
22	18	0.899748
22	19	0.899745
22	20	0.899867
22	21	0.899637
22	22	0.899636
22	23	0.899672
22	24	0.899678
22	25	0.899466
22	26	0.899484
22	27	0.899635
22	28	0.899676
22	29	0.89961
22	30	0.899447
22	31	0.899422
22	32	0.899701
22	33	0.899781
22	34	0.899842
22	35	0.899342
22	36	0.899142
23	1	0.874569
23	2	0.881201
23	3	0.88947
23	4	0.89805
23	5	0.89856
23	6	0.898919
23	7	0.899108
23	8	0.89963
23	9	0.899736
23	10	0.89967
23	11	0.899812
23	12	0.899695
23	13	0.899896
23	14	0.899578
23	15	0.899674
23	16	0.899304
23	17	0.900386
23	18	0.900323
23	19	0.900797
23	20	0.900682
23	21	0.90035
23	22	0.900599
23	23	0.90012

23	24	0.899969
23	25	0.900167
23	26	0.900252
23	27	0.900265
23	28	0.900378
23	29	0.899728
23	30	0.899738
23	31	0.899851
23	32	0.899755
23	33	0.899956
23	34	0.899622
23	35	0.899717
23	36	0.899432
24	1	0.879476
24	2	0.891048
24	3	0.892769
24	4	0.895594
24	5	0.894744
24	6	0.895492
24	7	0.896795
24	8	0.897396
24	9	0.897755
24	10	0.897453
24	11	0.897478
24	12	0.898535
24	13	0.898546
24	14	0.898545
24	15	0.899068
24	16	0.899982
24	17	0.899686
24	18	0.899328
24	19	0.900172
24	20	0.899854
24	21	0.899912
24	22	0.899895
24	23	0.899633
24	24	0.899731
24	25	0.899784
24	26	0.899932
24	27	0.899798
24	28	0.899793
24	29	0.899805
24	30	0.899732
24	31	0.899463
24	32	0.89961
24	33	0.899591
24	34	0.899875
24	35	0.899236
24	36	0.899756
25	1	0.88124
25	2	0.89096
25	3	0.891757
25	4	0.892391
25	5	0.892616
25	6	0.895349
25	7	0.899452
25	8	0.899082
25	9	0.899184
25	10	0.898622
25	11	0.898886
25	12	0.897861
25	13	0.898623
25	14	0.898318
25	15	0.898003

25	16	0.897589
25	17	0.897886
25	18	0.898253
25	19	0.898624
25	20	0.899027
25	21	0.899155
25	22	0.899424
25	23	0.899103
25	24	0.898949
25	25	0.898907
25	26	0.899269
25	27	0.899147
25	28	0.899141
25	29	0.899232
25	30	0.899832
25	31	0.899547
25	32	0.899621
25	33	0.899828
25	34	0.899842
25	35	0.899689
25	36	0.899317
26	1	0.874569
26	2	0.888741
26	3	0.898832
26	4	0.899864
26	5	0.90035
26	6	0.900529
26	7	0.900288
26	8	0.89945
26	9	0.900165
26	10	0.900033
26	11	0.900581
26	12	0.900104
26	13	0.899894
26	14	0.89976
26	15	0.899495
26	16	0.899388
26	17	0.899866
26	18	0.899652
26	19	0.899934
26	20	0.899306
26	21	0.899341
26	22	0.899434
26	23	0.899458
26	24	0.900215
26	25	0.900254
26	26	0.899897
26	27	0.899978
26	28	0.900031
26	29	0.899855
26	30	0.899701
26	31	0.899469
26	32	0.899576
26	33	0.89955
26	34	0.899493
26	35	0.899728
26	36	0.899215
27	1	0.891452
27	2	0.892866
27	3	0.894278
27	4	0.895929
27	5	0.895903
27	6	0.896172
27	7	0.896208

27	8	0.895899
27	9	0.895893
27	10	0.895985
27	11	0.897376
27	12	0.897228
27	13	0.897546
27	14	0.898081
27	15	0.898147
27	16	0.897907
27	17	0.898103
27	18	0.898126
27	19	0.897929
27	20	0.898915
27	21	0.898711
27	22	0.899024
27	23	0.89872
27	24	0.898643
27	25	0.898812
27	26	0.898605
27	27	0.898724
27	28	0.898982
27	29	0.898428
27	30	0.898519
27	31	0.898823
27	32	0.898869
27	33	0.898639
27	34	0.899091
27	35	0.899488
27	36	0.899106
28	1	0.891452
28	2	0.892433
28	3	0.895434
28	4	0.896106
28	5	0.895434
28	6	0.895733
28	7	0.895922
28	8	0.896261
28	9	0.896731
28	10	0.896506
28	11	0.896177
28	12	0.897195
28	13	0.897295
28	14	0.897246
28	15	0.897534
28	16	0.897657
28	17	0.897944
28	18	0.897938
28	19	0.897822
28	20	0.897816
28	21	0.898214
28	22	0.898281
28	23	0.898287
28	24	0.898063
28	25	0.897731
28	26	0.898134
28	27	0.89879
28	28	0.899405
28	29	0.899001
28	30	0.899143
28	31	0.899279
28	32	0.899194
28	33	0.899245
28	34	0.89955
28	35	0.899547

28	36	0.899245
29	1	0.873344
29	2	0.893371
29	3	0.895197
29	4	0.895268
29	5	0.895896
29	6	0.898011
29	7	0.898379
29	8	0.898123
29	9	0.897597
29	10	0.898509
29	11	0.8983
29	12	0.898928
29	13	0.899297
29	14	0.899022
29	15	0.898924
29	16	0.899214
29	17	0.898854
29	18	0.898883
29	19	0.899153
29	20	0.899227
29	21	0.899041
29	22	0.898806
29	23	0.898809
29	24	0.899013
29	25	0.89861
29	26	0.898387
29	27	0.898367
29	28	0.899121
29	29	0.898759
29	30	0.898938
29	31	0.899088
29	32	0.898926
29	33	0.898983
29	34	0.899047
29	35	0.899087
29	36	0.89951
30	1	0.887965
30	2	0.891271
30	3	0.893625
30	4	0.895727
30	5	0.894757
30	6	0.89613
30	7	0.895934
30	8	0.895653
30	9	0.895755
30	10	0.895385
30	11	0.895825
30	12	0.89626
30	13	0.896651
30	14	0.896763
30	15	0.897839
30	16	0.897796
30	17	0.897451
30	18	0.897577
30	19	0.898726
30	20	0.89844
30	21	0.898556
30	22	0.89872
30	23	0.898615
30	24	0.898539
30	25	0.898597
30	26	0.898609
30	27	0.898503

30	28	0.898644
30	29	0.898546
30	30	0.899391
30	31	0.899578
30	32	0.899426
30	33	0.89984
30	34	0.899801
30	35	0.899568
30	36	0.899566
31	1	0.873344
31	2	0.887225
31	3	0.890313
31	4	0.893277
31	5	0.894366
31	6	0.897768
31	7	0.897378
31	8	0.897433
31	9	0.898247
31	10	0.898075
31	11	0.898795
31	12	0.899708
31	13	0.899553
31	14	0.899512
31	15	0.899648
31	16	0.899714
31	17	0.900112
31	18	0.899899
31	19	0.900041
31	20	0.899976
31	21	0.899994
31	22	0.899548
31	23	0.899534
31	24	0.899719
31	25	0.899665
31	26	0.899596
31	27	0.90024
31	28	0.899813
31	29	0.90017
31	30	0.899676
31	31	0.899639
31	32	0.899678
31	33	0.899827
31	34	0.899757
31	35	0.899586
31	36	0.899692
32	1	0.888461
32	2	0.889431
32	3	0.890206
32	4	0.892275
32	5	0.89214
32	6	0.894376
32	7	0.896571
32	8	0.896982
32	9	0.897229
32	10	0.897647
32	11	0.898004
32	12	0.898578
32	13	0.898558
32	14	0.899565
32	15	0.899526
32	16	0.899278
32	17	0.89932
32	18	0.899678
32	19	0.899725

32	20	0.899495
32	21	0.899523
32	22	0.899349
32	23	0.899447
32	24	0.89952
32	25	0.899307
32	26	0.899335
32	27	0.899762
32	28	0.89963
32	29	0.899731
32	30	0.899969
32	31	0.900025
32	32	0.899968
32	33	0.89974
32	34	0.89993
32	35	0.900002
32	36	0.899409
33	1	0.874569
33	2	0.877314
33	3	0.888059
33	4	0.889989
33	5	0.893638
33	6	0.895053
33	7	0.895909
33	8	0.896256
33	9	0.897636
33	10	0.897358
33	11	0.897153
33	12	0.897185
33	13	0.897375
33	14	0.897387
33	15	0.897861
33	16	0.89778
33	17	0.897762
33	18	0.897877
33	19	0.897577
33	20	0.897752
33	21	0.899117
33	22	0.899045
33	23	0.898991
33	24	0.898841
33	25	0.898882
33	26	0.898999
33	27	0.899023
33	28	0.898997
33	29	0.899119
33	30	0.898774
33	31	0.898937
33	32	0.89914
33	33	0.899444
33	34	0.899683
33	35	0.899646
33	36	0.899575
34	1	0.843332
34	2	0.876716
34	3	0.89315
34	4	0.892684
34	5	0.893762
34	6	0.894797
34	7	0.895147
34	8	0.894813
34	9	0.894798
34	10	0.895372
34	11	0.895551

34	12	0.897529
34	13	0.897225
34	14	0.897893
34	15	0.89791
34	16	0.897863
34	17	0.897699
34	18	0.897772
34	19	0.897561
34	20	0.897648
34	21	0.897708
34	22	0.899399
34	23	0.89949
34	24	0.899423
34	25	0.89955
34	26	0.8997
34	27	0.900131
34	28	0.899831
34	29	0.900069
34	30	0.899886
34	31	0.899907
34	32	0.899723
34	33	0.899264
34	34	0.899652
34	35	0.899885
34	36	0.899573
35	1	0.888939
35	2	0.889441
35	3	0.891887
35	4	0.892987
35	5	0.892661
35	6	0.895463
35	7	0.898062
35	8	0.897727
35	9	0.898229
35	10	0.89818
35	11	0.897957
35	12	0.898173
35	13	0.898715
35	14	0.898535
35	15	0.898505
35	16	0.89845
35	17	0.899411
35	18	0.899615
35	19	0.899921
35	20	0.899807
35	21	0.899612
35	22	0.899716
35	23	0.89968
35	24	0.899256
35	25	0.899517
35	26	0.899669
35	27	0.899635
35	28	0.899696
35	29	0.899269
35	30	0.899379
35	31	0.89947
35	32	0.899614
35	33	0.899932
35	34	0.899459
35	35	0.899422
35	36	0.899296
36	1	0.889272
36	2	0.89353
36	3	0.895329

36	4	0.895668
36	5	0.895324
36	6	0.896116
36	7	0.897939
36	8	0.897879
36	9	0.897874
36	10	0.899616
36	11	0.899762
36	12	0.899768
36	13	0.899938
36	14	0.900025
36	15	0.899834
36	16	0.900023
36	17	0.899774
36	18	0.899829
36	19	0.899933
36	20	0.899855
36	21	0.899721
36	22	0.899594
36	23	0.899516
36	24	0.89945
36	25	0.899591
36	26	0.89963
36	27	0.89946
36	28	0.899939
36	29	0.899851
36	30	0.899997
36	31	0.899814
36	32	0.8998
36	33	0.899894
36	34	0.899839
36	35	0.899559
36	36	0.899766
37	1	0.882331
37	2	0.890289
37	3	0.890275
37	4	0.893882
37	5	0.897316
37	6	0.898218
37	7	0.898508
37	8	0.898239
37	9	0.898176
37	10	0.898322
37	11	0.897793
37	12	0.89818
37	13	0.898229
37	14	0.899793
37	15	0.899834
37	16	0.899211
37	17	0.899385
37	18	0.899446
37	19	0.899236
37	20	0.899523
37	21	0.899251
37	22	0.899798
37	23	0.899775
37	24	0.900285
37	25	0.89988
37	26	0.899978
37	27	0.900027
37	28	0.900036
37	29	0.899759
37	30	0.90016
37	31	0.89945

37	32	0.899775
37	33	0.899714
37	34	0.899534
37	35	0.899412
37	36	0.899871
38	1	0.8796
38	2	0.889377
38	3	0.892953
38	4	0.894816
38	5	0.897706
38	6	0.8977
38	7	0.899494
38	8	0.899693
38	9	0.89915
38	10	0.899014
38	11	0.898985
38	12	0.898907
38	13	0.898908
38	14	0.899004
38	15	0.898734
38	16	0.898865
38	17	0.898972
38	18	0.899486
38	19	0.899357
38	20	0.899214
38	21	0.89899
38	22	0.898828
38	23	0.898712
38	24	0.898811
38	25	0.898828
38	26	0.898887
38	27	0.899183
38	28	0.89925
38	29	0.899508
38	30	0.899552
38	31	0.899471
38	32	0.899598
38	33	0.899426
38	34	0.899522
38	35	0.899845
38	36	0.899247
39	1	0.857766
39	2	0.884345
39	3	0.895834
39	4	0.895728
39	5	0.896385
39	6	0.89635
39	7	0.896974
39	8	0.897039
39	9	0.897134
39	10	0.897147
39	11	0.897217
39	12	0.897174
39	13	0.897325
39	14	0.897279
39	15	0.897975
39	16	0.89786
39	17	0.897776
39	18	0.897953
39	19	0.897987
39	20	0.897954
39	21	0.897839
39	22	0.898285
39	23	0.898362

39	24	0.898108
39	25	0.898414
39	26	0.898149
39	27	0.898959
39	28	0.898885
39	29	0.898796
39	30	0.898804
39	31	0.899513
39	32	0.899602
39	33	0.899443
39	34	0.899738
39	35	0.899253
39	36	0.899665
40	1	0.808227
40	2	0.893495
40	3	0.895597
40	4	0.896593
40	5	0.896461
40	6	0.895683
40	7	0.896605
40	8	0.896098
40	9	0.89893
40	10	0.899434
40	11	0.900085
40	12	0.900215
40	13	0.900251
40	14	0.900335
40	15	0.900391
40	16	0.900212
40	17	0.900346
40	18	0.900428
40	19	0.9003
40	20	0.900377
40	21	0.900369
40	22	0.90053
40	23	0.900489
40	24	0.900358
40	25	0.900302
40	26	0.900283
40	27	0.900285
40	28	0.900101
40	29	0.899954
40	30	0.899924
40	31	0.900082
40	32	0.900061
40	33	0.900078
40	34	0.899791
40	35	0.899898
40	36	0.899709
41	1	0.889272
41	2	0.890176
41	3	0.892407
41	4	0.894658
41	5	0.895065
41	6	0.895691
41	7	0.895824
41	8	0.895894
41	9	0.895751
41	10	0.895567
41	11	0.895749
41	12	0.895329
41	13	0.895461
41	14	0.895804
41	15	0.895814

41	16	0.89542
41	17	0.895711
41	18	0.895536
41	19	0.89719
41	20	0.897273
41	21	0.897553
41	22	0.897406
41	23	0.897291
41	24	0.897808
41	25	0.897894
41	26	0.898414
41	27	0.898377
41	28	0.899156
41	29	0.899184
41	30	0.899266
41	31	0.899375
41	32	0.898928
41	33	0.89967
41	34	0.899493
41	35	0.899722
41	36	0.89953
42	1	0.875029
42	2	0.891034
42	3	0.893633
42	4	0.894017
42	5	0.894879
42	6	0.89518
42	7	0.894878
42	8	0.897558
42	9	0.898964
42	10	0.898633
42	11	0.898391
42	12	0.899454
42	13	0.899197
42	14	0.899395
42	15	0.899259
42	16	0.899305
42	17	0.899044
42	18	0.898854
42	19	0.898731
42	20	0.898928
42	21	0.898841
42	22	0.898547
42	23	0.898837
42	24	0.898297
42	25	0.898852
42	26	0.898657
42	27	0.898673
42	28	0.898608
42	29	0.898584
42	30	0.89849
42	31	0.898349
42	32	0.898825
42	33	0.899572
42	34	0.899526
42	35	0.89945
42	36	0.899562
43	1	0.726286
43	2	0.888872
43	3	0.890987
43	4	0.892395
43	5	0.893807
43	6	0.894465
43	7	0.893911

43	8	0.894079
43	9	0.894204
43	10	0.894144
43	11	0.895923
43	12	0.895978
43	13	0.89645
43	14	0.896192
43	15	0.896512
43	16	0.897568
43	17	0.898439
43	18	0.898793
43	19	0.898618
43	20	0.899588
43	21	0.899531
43	22	0.899565
43	23	0.899793
43	24	0.899642
43	25	0.899414
43	26	0.899589
43	27	0.899526
43	28	0.899813
43	29	0.899816
43	30	0.899388
43	31	0.899707
43	32	0.899578
43	33	0.899477
43	34	0.8997
43	35	0.899193
43	36	0.899442
44	1	0.857766
44	2	0.895708
44	3	0.899843
44	4	0.899843
44	5	0.898914
44	6	0.898615
44	7	0.898824
44	8	0.898863
44	9	0.898737
44	10	0.898699
44	11	0.898953
44	12	0.899356
44	13	0.899095
44	14	0.89919
44	15	0.898943
44	16	0.899024
44	17	0.89909
44	18	0.899229
44	19	0.899566
44	20	0.899642
44	21	0.900291
44	22	0.900352
44	23	0.899992
44	24	0.899823
44	25	0.899945
44	26	0.899806
44	27	0.899796
44	28	0.899456
44	29	0.899326
44	30	0.899442
44	31	0.89936
44	32	0.899455
44	33	0.89957
44	34	0.899467
44	35	0.899594

44	36	0.89948
45	1	0.843332
45	2	0.892907
45	3	0.893914
45	4	0.894425
45	5	0.895924
45	6	0.896389
45	7	0.896358
45	8	0.896503
45	9	0.896543
45	10	0.897526
45	11	0.897358
45	12	0.897179
45	13	0.897826
45	14	0.897468
45	15	0.897334
45	16	0.897854
45	17	0.897821
45	18	0.898718
45	19	0.89851
45	20	0.898624
45	21	0.899152
45	22	0.899055
45	23	0.899256
45	24	0.899082
45	25	0.899279
45	26	0.899023
45	27	0.899104
45	28	0.899028
45	29	0.899294
45	30	0.899496
45	31	0.899574
45	32	0.899697
45	33	0.899562
45	34	0.89954
45	35	0.899795
45	36	0.899693
46	1	0.874506
46	2	0.893331
46	3	0.896094
46	4	0.896263
46	5	0.895901
46	6	0.896243
46	7	0.896671
46	8	0.89683
46	9	0.897796
46	10	0.898051
46	11	0.897905
46	12	0.89934
46	13	0.899613
46	14	0.899189
46	15	0.899021
46	16	0.899039
46	17	0.899066
46	18	0.899381
46	19	0.899103
46	20	0.899006
46	21	0.89917
46	22	0.899193
46	23	0.898891
46	24	0.899483
46	25	0.899693
46	26	0.89997
46	27	0.900026

46	28	0.899967
46	29	0.899869
46	30	0.899835
46	31	0.899656
46	32	0.899719
46	33	0.899914
46	34	0.899747
46	35	0.899903
46	36	0.899508
47	1	0.857766
47	2	0.893624
47	3	0.89898
47	4	0.89886
47	5	0.898074
47	6	0.897926
47	7	0.898577
47	8	0.898921
47	9	0.899284
47	10	0.900366
47	11	0.900146
47	12	0.900193
47	13	0.899988
47	14	0.90001
47	15	0.899902
47	16	0.89999
47	17	0.899888
47	18	0.899989
47	19	0.900094
47	20	0.899901
47	21	0.899847
47	22	0.900084
47	23	0.900133
47	24	0.90008
47	25	0.900059
47	26	0.89995
47	27	0.899929
47	28	0.899558
47	29	0.899678
47	30	0.899675
47	31	0.899778
47	32	0.899614
47	33	0.899611
47	34	0.899255
47	35	0.89973
47	36	0.899988
48	1	0.857564
48	2	0.893558
48	3	0.896732
48	4	0.896221
48	5	0.896422
48	6	0.898606
48	7	0.898781
48	8	0.899939
48	9	0.899481
48	10	0.899842
48	11	0.899713
48	12	0.899539
48	13	0.899492
48	14	0.899201
48	15	0.899409
48	16	0.899171
48	17	0.89896
48	18	0.898984
48	19	0.899671

48	20	0.899573
48	21	0.899694
48	22	0.899283
48	23	0.899668
48	24	0.899467
48	25	0.899686
48	26	0.899575
48	27	0.899414
48	28	0.899383
48	29	0.899089
48	30	0.899258
48	31	0.899474
48	32	0.899484
48	33	0.899775
48	34	0.899792
48	35	0.899626
48	36	0.899688
49	1	0.87152
49	2	0.887391
49	3	0.887489
49	4	0.890667
49	5	0.894639
49	6	0.896554
49	7	0.896198
49	8	0.895573
49	9	0.897042
49	10	0.898167
49	11	0.897887
49	12	0.899414
49	13	0.899409
49	14	0.899805
49	15	0.898943
49	16	0.899326
49	17	0.899559
49	18	0.899762
49	19	0.900099
49	20	0.899984
49	21	0.899961
49	22	0.900079
49	23	0.899896
49	24	0.899861
49	25	0.900137
49	26	0.900207
49	27	0.899996
49	28	0.899941
49	29	0.899662
49	30	0.899719
49	31	0.899971
49	32	0.899588
49	33	0.899569
49	34	0.899475
49	35	0.899428
49	36	0.899538
50	1	0.887472
50	2	0.891457
50	3	0.892766
50	4	0.891866
50	5	0.893035
50	6	0.89579
50	7	0.898327
50	8	0.898065
50	9	0.8982
50	10	0.898184
50	11	0.898264

50	12	0.899007
50	13	0.899138
50	14	0.899037
50	15	0.898608
50	16	0.898361
50	17	0.898865
50	18	0.899206
50	19	0.899341
50	20	0.899264
50	21	0.899195
50	22	0.899366
50	23	0.899223
50	24	0.899178
50	25	0.899869
50	26	0.899883
50	27	0.899875
50	28	0.899854
50	29	0.899887
50	30	0.899851
50	31	0.8999
50	32	0.899914
50	33	0.899585
50	34	0.899712
50	35	0.899389
50	36	0.899575

8.4 Equation Glossary

This section includes all labelled equations.

8.4.1 Model averaging

$$\widehat{Y} = G(X) = \frac{1}{L}\sum_{l=1}^{L} G_l(X) = \frac{1}{L}\sum_{l=1}^{L} \hat{y}_l \tag{2.1}$$

8.4.2 Model bagging

$$\widehat{Y} = G(X^P) = \frac{1}{L}\sum_{l=1}^{L} G_l(X^{p_l}) = \frac{1}{L}\sum_{l=1}^{L} \hat{y}_l \tag{2.2}$$

8.4.3 Boosting principle

$$\widehat{Y} = G(X) = \sum_{l=1}^{L} \gamma_l G_l(X) = \sum_{l=1}^{L} \gamma_l \hat{y}_l \tag{2.3}$$

8.4.4 Gradient Boosting update

$$G_l(X) = G_{l-1}(X) + \gamma_l \hat{y}_l \tag{2.4}$$

8.4.5 Classification accuracy

$$\text{Classification accuracy} = \frac{\text{Tp+ Tn}}{\text{Tp+Tn+Fp+Fn}} \tag{2.5}$$

8.4.6 Precision at K

$$\text{Precision}_k = \frac{\text{Tp}_k}{\text{Tp}_k+\text{Fp}_k} \tag{2.6}$$

8.4.7 Sensitivity

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP+FN}} \tag{2.7}$$

8.4.8 Specificity

$$\text{Specificity} = \frac{\text{TN}}{\text{TN+FP}} \tag{2.8}$$

8.4.9 AUC

$$\text{AUC}(X, Y) = \sum_{i=1}^{n+} \sum_{j=1}^{n-} \frac{L\left[f(x_i^+) > f\left(x_j^-\right)\right] + \frac{1}{2} L\left[f(x_i^+) = f\left(x_j^-\right)\right]}{2n_+ n_-} \tag{2.9}$$

8.4.10 Pearson Correlation

$$r(X, Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sqrt{[\sum_{i=1}^{n}(x_i - \bar{X})^2][\sum_{i=1}^{n}(y_i - \bar{Y})^2]}} \tag{2.10}$$

8.4.11 Squared error of OLS

$$\mathrm{E}(\mathrm{W}) = \frac{1}{2}\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2 = \frac{1}{2}\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i} - \mathrm{W^T x_i})^2 \tag{2.11}$$

8.4.12 Squared error of Ridge (accounting for L2 regularization)

$$\mathrm{E}(\mathrm{W}) = \frac{1}{2}\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i} - \hat{\mathrm{y}}_\mathrm{i})^2 + \frac{1}{2}\lambda W^T \mathrm{W} = \frac{1}{2}\sum_{\mathrm{i}=1}^{\mathrm{N}}(\mathrm{y_i} - \mathrm{W^T x_i})^2 + \frac{1}{2}\lambda W^T \mathrm{W} \tag{2.12}$$

8.4.13 Ridge solution in matrix form

$$\widehat{\mathrm{W}} = \underset{\mathrm{W}}{\operatorname{argmin}}\, \mathrm{E}(\mathrm{w}) = (\mathrm{X^T X} + \lambda \mathrm{I})^{-1}\mathrm{X^T Y} \tag{2.13}$$

8.4.14 Gradient Descent update of W

$$\mathrm{W} = \mathrm{W} - \mathrm{a}\,\frac{\partial \mathrm{E}}{\partial \mathrm{W}} \tag{2.14}$$

8.4.15 Gradient Descent update of W in matrix notation

$$\mathrm{W} = \mathrm{W} - \alpha\, \mathrm{X}^T(\mathrm{XW} - \mathrm{Y}) \tag{2.15}$$

8.4.16 Stochastic Gradient Descent update of W using 1 sample point

$$\mathrm{W} = \mathrm{W} - \alpha\, x_i\left(W^T x_i - y_i\right) \tag{2.16}$$

8.4.17 Probability of y=1 with the Logistic Regression Formula

$$\mathrm{P}(\mathrm{Y} = 1 \mid \mathrm{X}, \mathrm{W}) = \sigma(\mathrm{W}^{\mathrm{T}}\mathrm{X}) = \frac{1}{1+ \mathrm{e}^{-\mathrm{W}^{\mathrm{T}}\mathrm{X}}} \quad (2.17)$$

8.4.18 Log Likelihood function for Logistic Regression

$$\mathrm{LogL}(\mathrm{W}) = \log\mathrm{P}(\mathrm{Y} \mid \mathrm{X}, \mathrm{W}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} \log\mathrm{P}(\mathrm{y_i} \mid \mathrm{x_i}, \mathrm{W}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} -\log\left(1 + \mathrm{e}^{-\mathrm{y_i}\mathrm{W}^{\mathrm{T}}\mathrm{x_i}}\right) \quad (2.18)$$

8.4.19 Estimating Coefficients W for Logistic Regression

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\ \mathrm{LogL}(\mathrm{w}) = \underset{W}{\operatorname{argmin}} \sum_{\mathrm{i}=1}^{\mathrm{N}} -\log\left(1 + \mathrm{e}^{-\mathrm{y_i}\mathrm{W}^{\mathrm{T}}\mathrm{x_i}}\right) + \frac{1}{2}\lambda W^{T}\mathrm{W} \quad (2.19)$$

8.4.20 Gradient of W in respect to minimizing Log Likelihood

$$\nabla_{\mathrm{W}}\mathrm{LogL}(\mathrm{W}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{\mathrm{x_i}\mathrm{y_i}}{1+ \mathrm{e}^{\mathrm{y_i}\mathrm{W}^{\mathrm{T}}\mathrm{x_i}}} \quad (2.20)$$

8.4.21 Hinge Loss function

$$\mathrm{HingeL}(\mathrm{W}) = \mathrm{HingeL}(\mathrm{Y}, \mathrm{X}, \mathrm{W}) = \sum_{\mathrm{i}=1}^{\mathrm{N}} \max\{0, 1 - \mathrm{y_i}\mathrm{W}^{\mathrm{t}}\mathrm{x_i}\} \quad (2.21)$$

8.4.22 Estimating coefficients W for hinge Loss

$$\widehat{W} = \underset{W}{\operatorname{argmin}}\ \mathrm{HingeL}(\mathrm{W}) = \underset{W}{\operatorname{argmin}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \max\{0, 1 - \mathrm{y_i}\mathrm{W}^{\mathrm{t}}\mathrm{x_i}\} + \frac{1}{2}\lambda W^{T}\mathrm{W} \quad (2.22)$$

8.4.23 Gradient of W in respect to minimizing Hinge Loss

$$\nabla_{W}\text{HingeL}(W) = \sum_{i=1}^{N} \begin{matrix} -y_i x_i \ , \text{if } y_i W^t x_i < 1 \\ 0, \text{ if } y_i W^t x_i \geq 1 \end{matrix} \tag{2.23}$$

8.4.24 Generic squared error function

$$E(\widehat{Y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.24}$$

8.4.25 Output function of an one-hidden layer ANN

$$\widehat{Y} = f(X^J, W^{J,H}) = G\left(\sum_{h=0}^{H}\left(w_{h,g}\sigma\left(\sum_{J=0}^{J}(w_{j,h}x_j)\right)\right)\right) \tag{2.25}$$

8.4.26 Hyperbolic Tangent activation function

$$\tanh(u) = \frac{e^u + e^{-u}}{e^u - e^{-u}} \tag{2.26}$$

8.4.27 Relu activation function

$$\text{relu}(u) = \max(0, u) \tag{2.27}$$

8.4.28 estimating the Weights of an one-hidden layer ANN

$$\widehat{W} = \underset{W}{\text{argmin}}\, E(W) = \underset{W}{\text{argmin}} \sum_{i=1}^{N}\left(y_i - G\left(\sum_{h=0}^{H}\left(w_{h,g}\sigma\left(\sum_{J=0}^{J}(w_{j,h}x_{i,j})\right)\right)\right)\right)^2 \tag{2.28}$$

8.4.29 Gradient of Ws in respect to minimizing the squared loss in an one-hidden layer ANN

$$\nabla_{\mathrm{E}} = \left(\frac{\partial \mathrm{E}}{\partial W_1}, \frac{\partial \mathrm{E}}{\partial W_2}, \ldots., \frac{\partial \mathrm{E}}{\partial W_m}\right) \tag{2.29}$$

8.4.30 Gradient of any W vector in respect to minimizing the squared loss in an one-hidden layer ANN

$$\mathrm{W}_m = \mathrm{W}_m - \alpha \frac{\partial \mathrm{E}}{\partial W_m} \tag{2.30}$$

8.4.31 Bayes' Theorem

$$\mathrm{P}(\mathrm{Y} \mid x_1, x_2, \ldots, x_J) = \frac{\mathrm{P}(Y)\mathrm{P}(x_1, x_2, \ldots, x_J \mid \mathrm{Y})}{\mathrm{P}(x_1, x_2, \ldots, x_J)} \tag{2.31}$$

8.4.32 Bayes' Theorem after assumption of independence

$$\mathrm{P}(\mathrm{Y} \mid x_1, x_2, \ldots, x_J) = \frac{\mathrm{P}(Y) \prod_{j=1}^{J} \mathrm{P}(x_j \mid Y)}{\mathrm{P}(x_1, x_2, \ldots, x_J)} \tag{2.32}$$

8.4.33 Bayes' Theorem after assumption of independence, excluding constant Denominator

$$\mathrm{P}(\mathrm{Y} \mid x_1, x_2, \ldots, x_J) \approx \mathrm{P}(Y) \prod_{j=1}^{J} \mathrm{P}(x_j \mid Y) \tag{2.33}$$

8.4.34 Obtaining estimate for Y using Bayes' Theorem based on the simplified formula

$$\hat{Y} = \underset{Y}{\text{argmax}}\ \mathrm{P}(Y)\ \prod_{j=1}^{J} \mathrm{P}(x_j|Y) \tag{2.34}$$

8.4.35 Probability of a continuous feature Xj that follows a Gaussian distribution to belong to a class of Y based on the Naïve Bayes' theorem

$$\mathrm{P}(x_j|Y) = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\pi\sigma_Y^2}}\ \exp\left(-\frac{(x_j-\mu_Y)^2}{2\sigma_Y^2}\right) \tag{2.35}$$

8.4.36 Euclidian Distance between two data points x,p

$$\text{Euclidian Distance}(\mathrm{x},\mathrm{p}) = \sqrt{\sum_{\mathrm{j}=1}^{\mathrm{J}}(\mathrm{x_j}-\mathrm{p_j})^2} \tag{2.36}$$

8.4.37 Rule for partitioning the data based on split point $D_{j,s}$

$$\begin{cases} X_j \le D_{j,s} \ \rightarrow \{\mathrm{X_1}, \mathrm{Y_1}\} \\ X_j > D_{j,s} \ \rightarrow \{\mathrm{X_2}, \mathrm{Y_2}\} \end{cases} \tag{2.37}$$

8.4.38 Entropy formula given Y with distinct classes C

$$\mathrm{En}(Y) = \sum_{\mathrm{c}=1}^{\mathrm{C}} -\mathrm{P}(\mathrm{Y}=\mathrm{c})\log_2 \mathrm{P}(\mathrm{Y}=\mathrm{c}) \tag{2.38}$$

8.4.39 Entropy formula given Y with distinct classes C and split point $D_{j,s}$

$$\mathrm{En}\left(\mathrm{Y},\mathrm{D}_{\mathrm{j,s}}\right)=\mathrm{P}\left(\mathrm{D}_{\mathrm{j,s}}=\prime\leq 50\prime\right)\mathrm{En}\left(Y_{\mathrm{D}_{\mathrm{j,s}}=\prime\leq 50\prime}\right)+\mathrm{P}\left(\mathrm{D}_{\mathrm{j,s}}=\prime>50\prime\right)\mathrm{En}\left(Y_{\mathrm{D}_{\mathrm{j,s}}=\prime>50\prime}\right) \quad (2.39)$$

8.4.40 Information Gain formula

$$\mathrm{IGain}\left(\mathrm{Y},\mathrm{D}_{\mathrm{j,s}}\right)=\ \mathrm{En}(\mathrm{Y})-\mathrm{En}\left(\mathrm{Y},\mathrm{D}_{\mathrm{j,s}}\right) \quad (2.40)$$

8.4.41 Non-Negative Matrix Factorization Prediction

$$\hat{Y}_{ij}=\ U_i V_j \quad (2.41)$$

8.4.42 Estimating U, V based on squared loss

$$\widehat{\mathrm{U}},\widehat{\mathrm{V}}=\underset{\mathrm{U,V}}{\mathrm{argmin}}\ \mathrm{E}(\mathrm{U},\mathrm{V})=\underset{\mathrm{U,V}}{\mathrm{argmin}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\sum_{\mathrm{j}=1}^{\mathrm{m}}\left(\mathrm{Y}_{\mathrm{ij}}-\mathrm{U}_{\mathrm{i}}\mathrm{V}_{\mathrm{j}}\right)^2 \quad (2.42)$$

8.4.43 Estimating W, U based on squared loss in libFM

$$\widehat{\mathrm{W}},\widehat{\mathrm{U}}=\underset{\widehat{\mathrm{W}},\widehat{\mathrm{U}}}{\mathrm{argmin}}\ \mathrm{E}(\mathrm{W},\mathrm{U})=\left(Y-\left(X_0+X_1w_1\ldots+X_mw_m+\ \sum_{j=1}^{m}\sum_{d=j+1}^{m}X_jX_dU_jU_d\right)\right)^2 \quad (2.43)$$

8.4.44 Prediction function of libFM

$$\mathrm{f}(\mathrm{W},\mathrm{U})=\left(\mathrm{X}_0+\mathrm{X}_1\mathrm{w}_1\ldots+\mathrm{X}_\mathrm{m}\mathrm{w}_\mathrm{m}+\ \sum_{\mathrm{j}=1}^{\mathrm{m}}\sum_{\mathrm{d}=\mathrm{j}+1}^{\mathrm{m}}\mathrm{X}_\mathrm{j}\mathrm{X}_\mathrm{d}\mathrm{U}_\mathrm{j}\mathrm{U}_\mathrm{d}\right) \quad (2.44)$$

8.4.45 linear update of libFM

$$\nabla_{\mathrm{W}}\mathrm{E}(\mathrm{W})=(\mathrm{f}(\mathrm{W},\mathrm{U})-\mathrm{Y})\mathrm{X} \quad (2.45)$$

8.4.46 Latent features' update of libFM

$$\nabla_U E(U) = (f(W, U) - Y)X \sum_{l=1}^{m} U_l X_l \quad (2.46)$$

8.4.47 Energy function for RBMs

$$S(u) = -\sum_{i=1}^{m} \sum_{j=1}^{F} \sum_{k=1}^{K} W_{i_j h_j u_i}^{k} - \sum_{i=1}^{m} \sum_{k=1}^{K} u_i b_i - \sum_{j=1}^{F} h_j b_j \quad (2.47)$$

8.4.48 Estimate of W in RBMs

$$\Delta w_{ij} = e\ (< u_i h_j >_{data} - < u_i h_j >_T) \quad (2.48)$$

8.4.49 Level 1 estimators' function for stacking

$$f_1(x_i, S^M, m) = S_m^M(x_i) \quad (4.1)$$

8.4.50 Level 2 (Meta) estimators' function

$$f_2(x_i, L, S^M) = L\left(f_1(x_i, S_1^M), f_1(x_i, S_2^M), \ldots., f_1\left(x_i, S_M^M\right)\right) \quad (4.2)$$

8.4.51 Average AUC using the Leave-one-offer-out schema

$$\mathrm{AUC}_{\mathrm{per\_offer}}(\hat{Y}, Y) = \frac{1}{N}\sum_{n=1}^{N} AUC(\widehat{Y_n}, Y_n) \quad (5.1)$$

8.4.52 Average AUC using the Leave-one-offer-out schema plus vertical concatenation

$$\mathrm{AUC}_{\mathrm{overal}}(\widehat{Y}, Y) = \mathrm{AUC}([\widehat{Y}_1|\widehat{Y}_2|\ldots|\widehat{Y}_N], [Y_1|Y_2|\ldots|Y_N]) \quad (5.2)$$

8.4.53 Average of the N-offer-out AUC and overall AUC

$$\mathrm{AUC}_{\mathrm{final}}(\mathrm{AUC}_{\mathrm{overall}}, \mathrm{AUC}_{\mathrm{per}_{\mathrm{offer}}}) = \frac{\mathrm{AUC}_{\mathrm{overall}}}{2} + \frac{\mathrm{AUC}_{\mathrm{per\_offer}}}{2} \quad (5.3)$$

8.4.54 Average of two predictions converted to ranks

$$\mathrm{hybrid}(\widehat{Y}_{cb}, \widehat{Y}_{cf}) = \frac{\widehat{Y}_{cb}^{rank}}{2} + \frac{\widehat{Y}_{cf}^{rank}}{2} \quad (5.4)$$

8.4.55 Ranking a sorted vector.

$$\hat{y}_{cb,i}^{rank} = \begin{cases} 0 & i = 0 \\ \mathrm{i} & \hat{y}_{cb,i}^{sort} > \hat{y}_{cb,i-1}^{sort} \\ \hat{y}_{cb,-1}^{rank} & \hat{y}_{cb,i}^{sort} = \hat{y}_{cb,i-1}^{sort} \end{cases} \tag{5.5}$$

8.4.56 Connection function between input data and one hidden unit in the form of as linear perceptron

$$\mathrm{f}_{1,\mathrm{h}}(\mathrm{x}, s) = \mathrm{s}\big(\mathrm{G}(\mathrm{x}_\mathrm{j})\big) = \mathrm{s}\big(\mathrm{x}_\mathrm{j}\big) \tag{6.2}$$

8.4.57 Connection function between input data and one hidden unit in the form of any algorithm

$$\mathrm{f}_1(\mathrm{x}, s, h) = \mathrm{s}\big(\mathrm{G}(\mathrm{x}_\mathrm{j})\big) = \mathrm{s}\big(\mathrm{x}_\mathrm{j}\big) \tag{6.2}$$

8.4.58 Connection function between input data and one hidden unit in the form of any algorithm assuming linear activation on input

$$\mathrm{f}_{1,\mathrm{h}}(\mathrm{x}, s) = \mathrm{s}\big(\mathrm{x}_\mathrm{j}\big) \tag{6.3}$$

8.4.59 Connection function between input data and one hidden unit in the form of any algorithm simplified

$$\mathrm{f}_1(\mathrm{x}, S) = \mathrm{S}_h\big(\mathrm{x}_\mathrm{j}\big) \tag{6.4}$$

8.4.60 Connection function between first and second hidden layer

$$f_{2,m}(x, l, S) = l\left(f_1(x, S_1), f_1(x, S_2), \ldots., f_1(x, S_H)\right) \quad (6.5)$$

8.4.61 Connection function between first and second hidden layer simplified

$$f_2(x, L, S) = L_m\left(f_1(x, S_1), f_1(x, S_2), \ldots., f_1(x, S_H)\right) \quad (6.6)$$

8.4.62 Connection function between first and second hidden with generic vector of estimators

$$f_2(x, V) = V_{2,m}\left(f_1\left(x, V_{1,1}\right), f_1\left(x, V_{1,2}\right), \ldots., f_1\left(x, V_{1,D_1}\right)\right) \quad (6.7)$$

8.4.63 Connection function for any given layer n

$$f_n(x, V) = V_{n,k}\left(f_{n-1}\left(x, V_{n-1,1}\right), f_{n-1}\left(x, V_{n-1,2}\right), \ldots., f_{n-1}\left(x, V_{n-1,D_{n-1}}\right)\right) \quad (6.8)$$

8.4.64 Connection function for any given layer through restacking mode

$$f_n(x, V) = V_{n,k}\begin{pmatrix} f_{n-1}\left(x, V_{n-1,1}\right), f_{n-1}\left(x, V_{n-1,2}\right), \ldots, f_{n-1}\left(x, V_{n-1,D_{n-1}}\right), \\ f_{n-2}\left(x, V_{n-2,1}\right), f_{n-2}\left(x, V_{n-2,2}\right), \ldots, f_{n-2}\left(x, V_{n-2,D_{n-2}}\right), \\ \ldots, \\ f_{n-N+1}\left(x, V_{n-N+1,1}\right), f_{n-N+1}\left(x, V_{n-N+1,2}\right), \ldots, f_{n-N+1}\left(x, V_{n-N+1,D_{n-N+1}}\right) \end{pmatrix} \quad (6.9)$$

8.4.65 Optimizing parameters for any estimator in StackNet

$$\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}} = \underset{\mathrm{O}_{\mathrm{V}_{n,k}}}{\operatorname{argmin}}\ \mathrm{LL}(\mathrm{O}_{\mathrm{V}_{n,k}}, \left(\mathrm{f}_{\mathrm{n}-1}(\mathrm{x}, \mathrm{V}_{n-1,1}), \mathrm{f}_{\mathrm{n}-1}(\mathrm{x}, \mathrm{V}_{n-1,2}), \ldots., \mathrm{f}_{\mathrm{n}-1}(\mathrm{x}, \mathrm{V}_{n-1,D_{n-1}})\right), \mathrm{Y}) \quad (6.10)$$

8.4.66 Example optimizing parameters for any estimator assuming a squared loss function

$$\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}} = \underset{\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}}{\operatorname{argmin}}\ E\left(\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}\right) = \underset{\widehat{\mathrm{O}_{\mathrm{V}_{n,k}}}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \mathrm{V}_{n,k}(x_i)\right)^2 \quad (6.11)$$

8.4.67 Diversity of an ensemble based on the correlation matrix of predictions

$$\mathrm{diversity}(\mathrm{R}) = \frac{1}{\mathrm{N}\times\mathrm{N}} \sum_{\mathrm{n}=1}^{\mathrm{N}} \sum_{\mathrm{k}=1}^{\mathrm{N}} \mathrm{r}(\mathrm{n}, \mathrm{k}) \quad (6.12)$$

Bibliography

Acland, D., & Levy, M. (2011). Habit formation, naiveté, and projection bias in gym attendance.

Acock, A. C. (2005). Working with missing values. Journal of Marriage and family, 67(4), 1012-1028.

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering, 17(6), 734-749.

Adomavicius, G., & Tuzhilin, A. (2011). Context-aware recommender systems. In Recommender systems handbook (pp. 217-253). Springer US.

Aher, S. B., & Lobo, L. M. R. J. (2011). Data mining in educational system using weka. In International Conference on Emerging Technology Trends (ICETT) (Vol. 4, No. 10, pp. 100-110).

Ahn, H. J. (2008). A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. Information Sciences, 178(1), 37-51.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician, 46(3), 175-185.

Anand, S. S., & Mobasher, B. (2007). Contextual Recommendation, From Web to Social Web: Discovering and Deploying User and Content Profiles: Workshop on Web Mining, WebMine 2006, Berlin, Germany, September 18, 2006. Revised Selected and Invited Papers.

Anderson, R. E., & Srinivasan, S. S. (2003). E-satisfaction and e-loyalty: A contingency framework. Psychology & marketing, 20(2), 123-138.

Arel, I., Rose, D. C., & Karnowski, T. P. (2010). Deep machine learning-a new frontier in artificial intelligence research [research frontier]. IEEE Computational Intelligence Magazine, 5(4), 13-18.

Ashtawy, H. M., & Mahapatra, N. R. (2015). BgN-Score and BsN-Score: Bagging and boosting based ensemble neural networks scoring functions for accurate binding affinity prediction of protein-ligand complexes. BMC bioinformatics, 16(4), S8.

Bates, J. M., & Granger, C. W. (1969). The combination of forecasts. Journal of the Operational Research Society, 20(4), 451-468.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G. & Bengio, Y. (2010, June). Theano: A CPU and GPU math compiler in Python. In Proc. 9th Python in Science Conf (pp. 1-7).

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., & Lamere, P. (2011, October). The Million Song Dataset. In ISMIR (Vol. 2, No. 9, p. 10).

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

Bouckaert, R. R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., & Scuse, D. (2010). WEKA Manual for Version 3-7-8. The university of WAIKATO.

Boyer, K. K., & Hult, G. T. M. (2005). Customer behavior in an online ordering application: A decision scoring model. Decision Sciences, 36(4), 569-598.

Bratko, I., Michalski, R. S., & Kubat, M. (1999). Machine learning and data mining: methods and applications.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). Classification and regression trees. CRC press.

Breiman, L. (1996). Stacked regressions. Machine learning, 24(1), 49-64.

Breiman, L. (1997). Arcing the edge. Technical Report 486, Statistics Department, University of California at Berkeley.

Breiman, L. (1999). Pasting small votes for classification in large databases and on-line. Machine Learning, 36(1), 85-103.

Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.

Brose, G., Vogel, A., & Duddy, K. (2001). Java programming with CORBA: advanced techniques for building distributed applications (Vol. 6). John Wiley & Sons.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O. & Layton, R. (2013). API design for machine learning software: experiences from the scikit-learn project. arXiv preprint arXiv:1309.0238.

Chang, C. C., & Lin, C. J. (2011). LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27.

Chen, H., Chiang, R. H., & Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. MIS quarterly, 36(4).

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. arXiv preprint arXiv:1603.02754.

Christakou, C., Vrettos, S., & Stafylopatis, A. (2007). A hybrid movie recommender system based on neural networks. International Journal on Artificial Intelligence Tools, 16(05), 771-792.

Collobert, R., & Bengio, S. (2001). SVMTorch: Support vector machines for large-scale regression problems. Journal of machine learning research, 1(Feb), 143-160.

Craven, B. D., & Islam, S. M. (2011). Ordinary least squares regression (pp. 224-228). Sage Publications.

Cunningham, P., & Carney, J. (2000). Diversity versus quality in classification ensembles based on feature selection.Technical Report TCD-CS-2000-02, Department of Computer Science, Trinity College Dublin.

Dang, V. (2013). Ranklib.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In International workshop on multiple classifier systems (pp. 1-15). Springer Berlin Heidelberg.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. Machine Learning, 40:2, 139–157

Domingos, P. (2012). A few useful things to know about machine learning. Communications of the ACM, 55(10), 78-87.

Dougherty, J., Kohavi, R., & Sahami, M. (1995, July). Supervised and unsupervised discretization of continuous features. In Machine learning: proceedings of the twelfth international conference (Vol. 12, pp. 194-202).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

Dunnhumby (2014). The Complete Journey. Dunnhumb USA, retrieved from

https://www.dunnhumby.com/sourcefiles

Ensemble methods (2010), retrieved from http://scikit-learn.org/stable/modules/ensemble.html

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). LIBLINEAR: A library for large linear classification. Journal of machine learning research, 9(Aug), 1871-1874.

Freund, Y., & Schapire, R. E. (1995, March). A desicion-theoretic generalization of on-line learning and an application to boosting. In European conference on computational learning theory (pp. 23-37). Springer Berlin Heidelberg.

François Chollet (2015). keras, GitHub, GitHub repository, https://github.com/fchollet/keras, 5bcac37.

Friedman, J. H. (1991). Multivariate adaptive regression splines. The annals of statistics, 1-67.

Gandomi, A., & Haider, M. (2015). Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35(2), 137-144.

Gao, W., Jin, R., Zhu, S., & Zhou, Z. H. (2013, May). One-Pass AUC Optimization. In ICML (3) (pp. 906-914).

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., & Huber, A. (2014, October). Offline and online evaluation of news recommender systems at swissinfo. ch. In Proceedings of the 8th ACM Conference on Recommender systems (pp. 169-176). ACM.

Garreta, R., & Moncecchi, G. (2013). Learning scikit-learn: machine learning in python. Packt Publishing Ltd.

Gauvain, J. L., & Lee, C. H. (1994). Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains. IEEE transactions on speech and audio processing, 2(2), 291-298.

Golub, G. H., & Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische mathematik, 14(5), 403-420.

Granger, C. W. (1989). Invited review combining forecasts—twenty years later. Journal of Forecasting, 8(3), 167-173.

Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.

Giacinto, G., & Roli, F. (2001). Design of effective neural network ensembles for image classification processes.Image Vision and Computing Journal, 19:9/10, 699–707.

Green, D. M., & Swets, J. A. (1966). Signal detection theory and psychophysics. Society, 1, 521.

H2O (2016) Stacked Ensembles, retrieved from http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 10-18.

Han J., Kamber M. (2006) Data Mining: Concepts and Techniques, Second Edition, University of Illinois at Urbana-Champaign

Hand, D. J., & Yu, K. (2001). Idiot's Bayes—not so stupid after all? International statistical review, 69(3), 385-398.

Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology, 143(1), 29-36.

Hansen, L., & Salamon, P. (1990). Neural network ensembles.IEEE Transactions on Pattern Analysis and Machine Intelligence, 12:10, 993–1001.

Hao, F., & Blair, R. H. (2016). A comparative study: classification vs. user-based collaborative filtering for clinical prediction. BMC medical research methodology, 16(1), 172.

Henk van Veen (2015) KAGGLE ENSEMBLING GUIDE, Retrieved from: https://mlwave.com/kaggle-ensembling-guide/

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS), 22(1), 5-53.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. Neural computation, 14(8), 1771-1800.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

Ho, T. (1998). The random space method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20:8, 832–844

Hsieh, N. C., & Hung, L. P. (2010). A data driven ensemble classifier for credit scoring analysis. Expert systems with Applications, 37(1), 534-545.

Hu, R., & Lu, Y. (2006, November). A hybrid user and item-based collaborative filtering with smoothing on sparse data. In Artificial Reality and Telexistence--Workshops, 2006. ICAT'06. 16th International Conference on (pp. 184-189). IEEE.

Iskold, A. (2007). The art, science and business of recommendation engines. Retrieved April, 5, 2012.

Jean Francois Puget (2016), The Most Popular Language For Machine Learning Is… retrieved from https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Language_Is_Best_For_Machine_Learning_And_Data_Science?lang=en

Jia Li (nd), Logistic Regression, Department of Statistics, The Pennsylvania State University, retrieved from sites.stat.psu.edu/~jiali/course/stat597e/notes2/logit.pdf

Jin, H., & Lu, Y. (2009). The optimal linear combination of multiple predictors under the generalized linear models. Statistics & probability letters, 79(22), 2321-2327.

Juan, Y., Chin, W. S., & Zhuang, Y. (2015). Libffm: A library for field-aware factorization machines.

Karatzoglou, A., Amatriain, X., Baltrunas, L., & Oliver, N. (2010, September). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In Proceedings of the fourth ACM conference on Recommender systems (pp. 79-86). ACM.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. Applied statistics, 119-127.

Kearns, M. (1988). Thoughts on hypothesis boosting. Unpublished manuscript, 45, 105.

Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai (Vol. 14, No. 2, pp. 1137-1145).

Kohavi, R., & Wolpert, D. (1996). Bias plus variance decomposition for zero-one loss functions. In L. Saitta (Ed.), Machine Learning: Proc. 13th International Conference (pp. 275–283). Morgan Kaufmann.

Koren, Y. (2009). The bellkor solution to the netflix grand prize. Netflix prize documentation, 81, 1-10.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.

Küchler, M. (1983). Die Schätzung von Wählerwanderungen: Neue Lösungsversuche. In Wahlen und politisches System (pp. 632-651). VS Verlag für Sozialwissenschaften.

Kuhn, M., Wing, J., Weston, S., Williams, A., Keefer, C., Engelhardt, A. & Lescarbeau, R. (2014). Caret: classification and regression training. R package version 6.0-24.

Kuncheva, L. I., & Whitaker, C. J. (2003). Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine learning, 51(2), 181-207.

Kyrola, A., Blelloch, G. E., & Guestrin, C. (2012, October). GraphChi: Large-Scale Graph Computation on Just a PC. In OSDI (Vol. 12, pp. 31-46).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

LeDell, E. E. (2015). Scalable Ensemble Learning and Computationally Efficient Variance Estimation (Doctoral dissertation, University of California, Berkeley).

Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In Advances in neural information processing systems (pp. 556-562).

Lee D., & Drabas T. (2017). Learning PySpark: Build data-intensive applications locally and deploy at scale using the combined powers of Python and spark 2.0. Packt Publishing Ltd.

Lewis, M. (2004). The influence of loyalty programs and short-term promotions on customer retention. Journal of marketing research, 41(3), 281-292.

Lichman, M. (2013). UCI Machine Learning Repository, CA: University of California, Irvine,School of Information and Computer Science, retrieved from http://archive.ics.uci.edu/ml

Lika, B., Kolomvatsos, K., & Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. Expert Systems with Applications, 41(4), 2065-2073.

Linden, G. D., Jacobi, J. A., & Benson, E. A. (2001). U.S. Patent No. 6,266,649. Washington, DC: U.S. Patent and Trademark Office.

Li, W., Liu, H., Yang, P., & Xie, W. (2016). Supporting regularized logistic regression privately and efficiently. PloS one, 11(6), e0156479.

Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing, 7(1), 76-80.

Louppe, G., & Geurts, P. (2012, September). Ensembles on random patches. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 346-361). Springer, Berlin, Heidelberg.

Lucas, A. (2001). Statistical challenges in credit card issuing. Applied Stochastic Models in Business and Industry, 17(1), 83-92.

Luchman, J. N. (2015). Chaid: Stata module to conduct chi-square automated interaction detection. Statistical Software Components.

Lutz, Mark. Programming python. “O’Reilly Media, Inc.", 2010.

Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009, June). Identifying suspicious URLs: an application of large-scale online learning. In Proceedings of the 26th annual international conference on machine learning (pp. 681-688). ACM.

Marcus, C. (1998). A practical yet meaningful approach to customer segmentation. Journal of consumer marketing, 15(5), 494-504.

McMahan, B. (2011). Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (pp. 525-533).

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

Melville, P., & Mooney, R. J. (2003, August). Constructing diverse classifier ensembles using artificial training examples. In IJCAI (Vol. 3, pp. 505-510).

Melville, P., & Mooney, R. J. (2005). Creating diversity in ensembles using artificial data. Information Fusion, 6(1), 99-111.

Meyer-Waarden, L. (2008). The influence of loyalty programme membership on customer purchase behaviour. European Journal of Marketing, 42(1/2), 87-114.

Mims C. (2012). Why CPUs Aren't Getting Any Faster. MIT Technology Review, retrieved from https://www.technologyreview.com/s/421186/why-cpus-arent-getting-any-faster/

Minka, T. P. (2003). A comparison of numerical optimizers for logistic regression. Unpublished draft.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). Foundations of machine learning. MIT press.

Mjolsness, E., & DeCoste, D. (2001). Machine learning for science: state of the art and future prospects. Science, 293(5537), 2051-2055.

Montgomery, J. M., & Nyhan, B. (2010). Bayesian model averaging: Theoretical developments and practical applications. Political Analysis, 18(2), 245-270.

Mukaka, M. M. (2012). A guide to appropriate use of correlation coefficient in medical research. Malawi Medical Journal, 24(3), 69-71.

Murtagh, F. (2005). Correspondence analysis and data coding with Java and R. CRC Press.

Netflix Prize Competition (2009), retrieved from http://www.netflixprize.com/index

Neural Networks (nd), retrieved from http://docs.opencv.org/modules/ml/doc/neural_networks.html

O'brien, M. R., Off, G. W., & Cherney, T. L. (2001). U.S. Patent No. 6,321,210. Washington, DC: U.S. Patent and Trademark Office.

Opitz, D., & Maclin, R. (1999). Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research, 11, 169-198.

Partridge, D., & Krzanowski, W. J. (1997). Software diversity: Practical statistics for its measurement and exploitation.Information & Software Technology, 39, 707–717.

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2(11), 559-572.

Karl Pearson, F. R. S. (1904). Mathematical contributions to the theory of evolution. Dulau and Co., London.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830.

Poirier, D., Fessant, F., & Tellier, I. (2010, August). Reducing the cold-start problem in content recommendation through opinion classification. In Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on (Vol. 1, pp. 204-207). IEEE.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. IEEE Computer, 33(10), 23-29.

Quinlan, J. R. (1986). Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81–106

Quinlan, J. R. (1993). C4. 5: Programming for machine learning. Morgan Kauffmann, 38.

Rao, C. R., Rao, C. R., Statistiker, M., Rao, C. R., & Rao, C. R. (1973). Linear statistical inference and its applications (Vol. 2, pp. 263-270). New York: Wiley.

Ravi, V. T., & Agrawal, G. (2009, May). Performance issues in parallelizing data-intensive applications on a multi-core cluster. In Proceedings of the 2009 9th IEEE/ACM

International Symposium on Cluster Computing and the Grid (pp. 308-315). IEEE Computer Society.

Rendle, S. (2012). Factorization machines with libfm. ACM Transactions on Intelligent Systems and Technology (TIST), 3(3), 57.

Rendle, S., Gantner, Z., Freudenthaler, C., & Schmidt-Thieme, L. (2011, July). Fast context-aware recommendations with factorization machines. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval (pp. 635-644). ACM.

Reyes, A., Sucar, L. E., & Morales, E. F. (2009). AsistO: A qualitative MDP-based recommender system for power plant operation. Computación y Sistemas, 13(1).

Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook (pp. 1-35). Springer US.

Rosasco, L., De Vito, E., Caponnetto, A., Piana, M., & Verri, A. (2004). Are loss functions all the same? Neural Computation, 16(5), 1063-1076.

Rogova, G. (1994). Combining the results of several neural network classifiers. Neural networks, 7(5), 777-781.

Rojas, R. (1996). The backpropagation algorithm. In Neural networks (pp. 149-182). Springer Berlin Heidelberg.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Ruel, E. E., Wagner III, W. E., & Gillespie, B. J. (2015). The Practice of Survey Research. Sage.

Rubrecht M (2014). Coupon history helps you understand changes, retrieved from m.savannahnow.com/savvy-shopper/2014-02-02/coupon-history-helps-you-understand-changes#gsc.tab=0

Ruslan Salakhutdinov (2007) ,Restricted Boltzmann Machines for Collaborative Filtering University of Toronto, 6 King's College Rd., Toronto, Ontario M5S 3G4, Canada

Said, A. (2010, September). Identifying and utilizing contextual data in hybrid recommender systems. In Proceedings of the fourth ACM conference on Recommender systems (pp. 365-368). ACM.

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007, June). Restricted Boltzmann machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning (pp. 791-798). ACM.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001, April). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web (pp. 285-295). ACM.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117.

Sharkey, A. J. C. (1996). On combining artificial neural nets. Connection Science, 8(3-4), 299-314.

Sharkey, A., & Sharkey, N. (1997). Diversity, selection, and ensembles of artificial neural nets. Neural Networks and their Applications (NEURAP'97), 205-212.

Siami, M., & Hajimohammadi, Z. (2013). Credit scoring in banks and financial institutions via data mining techniques: A literature review. Journal of AI and Data Mining, 1(2), 119-129.

Smolensky, Paul (1986). Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory. In Rumelhart, David E.; McLelland, James L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations. MIT Press. pp. 194–281.

Smyth, P., & Wolpert, D. (1999). Linearly combining density estimators via stacking. Machine Learning, 36(1-2), 59-83.

Sneath, P., & Sokal, R. (1973). Numerical Taxonomy. W.H. Freeman & Co.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

Stanford University (2013), Introduction to Information Retrieval – Evaluation, retrieved from http://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf

Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb's journal, 30(3), 202-210.

Tan, A. C., & Gilbert, D. (2003). Ensemble machine learning on gene expression data for cancer classification.

Team, T. T. D., Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., ... & Belopolsky, A. (2016). Theano: A Python framework for fast computation of mathematical expressions. arXiv preprint arXiv:1605.02688.

The Echo Nest Analyze (nd), API, retrieved from http://developer.echonest.com

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit scoring and its applications. Siam.

Tikhonov, A. N., & Arsenin, V. Y. (1977). Solutions of ill-posed problems.

Tikhonov, A. N. (1963). Regularization of incorrectly posed problems. SOVIET MATHEMATICS DOKLADY.

TIOBE (2017), TIOBE Index for May 2017, retrieved from https://www.tiobe.com/tiobe-index/

Tripsas, M. (2008). Customer preference discontinuities: A trigger for radical technological change. Managerial and decision economics, 29(2-3), 79-97.

Tsoumakas, G., Dimou, A., Spyromitros, E., Mezaris, V., Kompatsiaris, I., & Vlahavas, I. (2009). Correlation-based pruning of stacked binary relevance models for multi-label learning. In Proceedings of the 1st international workshop on learning from multi-label data (pp. 101-116).

University of Auckland notes on additive trees and gradient boosting (nd), retrieved from https://www.stat.auckland.ac.nz/~yee/784/files/ch10BoostingAdditiveTrees.pdf

Van der Laan, M. J., Polley, E. C., & Hubbard, A. E. (2007). Super learner. Statistical applications in genetics and molecular biology, 6(1).

Van Setten, M., Pokraev, S., & Koolwaaij, J. (2004, August). Context-aware recommendations in the mobile tourist application COMPASS. In International Conference

on Adaptive Hypermedia and Adaptive Web-Based Systems (pp. 235-244). Springer Berlin Heidelberg.

Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. Journal of the American statistical association, 58(301), 236-244.

Weinberger, K. Q., Blitzer, J., & Saul, L. K. (2005). Distance metric learning for large margin nearest neighbor classification. In Advances in neural information processing systems (pp. 1473-1480).

Weng, S. S., & Liu, M. J. (2004). Feature-based recommendations for one-to-one marketing. Expert Systems with Applications, 26(4), 493-508.

Weston, J., Yee, H., & Weiss, R. J. (2013, October). Learning to rank recommendations with the k-order statistic loss. In Proceedings of the 7th ACM conference on Recommender systems (pp. 245-248). ACM.

Weston, J., Yee, H., & Weiss, R. J. (2013, October). Learning to rank recommendations with the k-order statistic loss. In Proceedings of the 7th ACM conference on Recommender systems (pp. 245-248). ACM.

Williams, G. (2011). Data mining with rattle and R: the art of excavating data for knowledge discovery. Springer Science & Business Media.

Witten, I. H., & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.

Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., & Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with Java implementations.

Wold, S., Ruhe, A., Wold, H., & Dunn, III, W. J. (1984). The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses. SIAM Journal on Scientific and Statistical Computing, 5(3), 735-743.

Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241-259.

Xi' an (2006). Advanced Data Mining and Applications, Second International Conference, ADMA 2006, China, August 14-16, Proceedings

Zhang, H. (2004). The optimality of naive Bayes. AA, 1(2), 3.

Yang, X., Steck, H., Guo, Y., & Liu, Y. (2012). On top-k recommendation using social networks. In Proceedings of the sixth ACM conference on Recommender systems (pp. 67-74). ACM.

Yule, G. (1900). On the association of attributes in statistics. Phil. Trans., A, 194, 257–319.

Zeng, G. (2014). A necessary condition for a good binning algorithm in credit scoring. Applied Mathematical Sciences, 8(65), 3229-3242.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning (p. 116). ACM.

Zhao, L., Hu, N. J., & Zhang, S. Z. (2002). Algorithm design for personalization recommendation systems. Journal of computer research and development, 39(8), 986-991.

Zhou, Z. H., Wu, J., & Tang, W. (2002). Ensembling neural networks: many could be better than all. Artificial intelligence, 137(1-2), 239-263.

Zhou, Z. H., & Feng, J. (2017). Deep forest: Towards an alternative to deep neural networks. arXiv preprint arXiv:1702.08835.