Multiclass Cross-selling Model for Savings and Investments Using Gradient Boosting

Arwin Sohrabi

Master's thesis
2018:E76

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Numerical Analysis

CENTRUM SCIENTIARUM MATHEMATICARUM

LUND
UNIVERSITY

Master of Science
Engineering Mathematics

Multiclass Cross-selling Model for Savings and Investments Using Gradient Boosting

Arwin Sohrabi

Principal supervisor
Alexandros Sopasakis
Lund University

Assistant supervisor
Philip R. Jarnhus
Danske Bank

2018

"I am not a number, I am a free man!"

Number Six, The Prisoner (1967)

Abstract

Danske Bank has for several years modelled customer purchase behavior on category level (e.g. savings or investments). This thesis is a first attempt at predicting (first time) customer purchase behaviour on product level. Five products within two categories were chosen and modelling was done with python using gradient boosters (mainly XGBoost, but also Light GBM). Results indicated that predicting product purchase is possible, although not with the target selected for this thesis. Multiclass modelling gives additional insight into customer behaviour compared to models on category level, however, additional tuning of the models are required before the accuracy reaches the same level as the category prediction models.

Keywords: Machine learning, gradient boosting, multiclass target, XGBoost, finance, Danske Bank, customer behaviour

Acknowledgements

I am very grateful to everyone, my family in particular, that in various ways aided me during the conduction of this thesis. Still, there are a few who deserve a special thanks:

A great thanks to my supervisor Alexandros Sopasakis, who helped me from start to finish. His much appreciated support and suggestions made the process significantly easier and the thesis notably better.

A big thanks to the entire Commercial Analytics team at Danske Bank. Having the opportunity of being with these bright minds on a day-to-day basis allowed me to get inspiration as well as much appreciated insights. But perhaps more importantly, their company made the entire process much more enjoyable. A special thanks to the head of the team, Michael Aamand, who authorized the collaboration on behalf of the bank and allowed me to conduct it at location for the whole period. Also, a thanks to the members of the surrounding teams.

My biggest thanks goes to Danske Bank senior analyst Philip R. Jarnhus. Not only for supervising me at the bank, but also for being a mentor as well as a great friend. His knowledge in his field of work is matched only by his wonderful personality.

Thank you all!

Popular science

How well does your bank know you? As artificial intelligence becomes increasingly sophisticated, the demand for it increases throughout various industries. The financial sector is no exception. Being one of the largest banks in the Nordics, Danske Bank has already employed machine learning for several years. But can these techniques be even more refined?

As a customer oriented bank, Danske Bank aims at being relevant for its customers and their needs. For many years, the banks advisers have received A.I. generated leads on customer purchase behavior. These leads will let an adviser know if a specific customer is interested in a product from one of the banks many product categories (e.g. savings or investments). However, as the categories are not necessarily known the customer, the advisers are often required to make some research on their own before approaching the customer with suggestions on specific products. This brought up the question: Can customer behaviour be modelled on product level, rather than category level? If so, it would move leads from internal business definitions to customer oriented definitions (as the customers often are familiar with basic bank products). It would further give advisers a more clear entry point into the conversation with the customer, saving them time spent on research. Using a machine learning technique known as "gradient boosting", it was shown that one could indeed model customer behaviour on product level. Gradient boosters use an ensemble of "weak" predictors (predictors that are only slightly better than random guessers) to generate a prediction that is like one from a "strong" predictor (a predictor that has a high accuracy). In recent years, sophisticated free-to-use boosters have emerged which allows anyone to employ the power of machine learning. This project used two boosters known as XGBoost and Light GBM; the former being more accurate and the latter being faster. Throughout the project, the model was tuned using feature engineering and hyperparameter optimizers. In the end, the final model showed clearly that additional information could be gained by modelling on product level. However, a different target and finer tuning is required before the product level result is satisfactory enough for deployment.

Hyperparameter - Parameter whose value is decided manually (e.g. by the user) rather than by the machine learning algorithm.

Contents

Chapter 1

Introduction

Over the past few years, profound changes and adaptations have taken place regarding machine learning applications throughout various industries. Not being restricted by the field of work, cutting edge machine learning techniques allow for ever increasing gains no matter the industry. The financial sector is no different; for several years these models have been employed for activities such as e.g. trading, fraud detection, churn prediction and marketing. Danske Bank, the largest bank in Denmark, is no exception. Incorporating machine learning models in the marketing activities within the bank has been common practice for a couple of years now. These activities have proven to be an essential part in contacting customers with relevant products and information.

Up until this point the models have only predicted whether or not a customer is going to buy from a product category. While this helps marketing efforts, it still leaves a lot to be desired. Most notably, it is not possible for neither the sales advisor nor the marketer to know which specific product that should be suggested to the customer. To further this knowledge would benefit both the bank as well as the customer (who will be approached with relevant advice).

Modelling the problem at this level has previously not been attempted at the bank; partly because the need has not arisen until now, and partly because it has been assumed to be too noisy to model.

As far as methods go, the thesis will revolve around classical, supervised machine learning methods. Though there is theoretically a freedom of choice in terms of algorithms, boosting algorithms have previously been employed with great success in modelling these problems. As such, the main focus will be on XGBoost, and limited focus will be on Light GBM.

From the above mentioned information, and some added to it, the full problem formulation can be stated as

> Given the banks information about a specific customer (acquired through ordinary channels, i.e. no additional information has been gathered as part of the project), is it possible to, three months in advance, predict the purchase of one of five specific products within the investment and savings categories that said customer does not at the moment of prediction own?

The thesis title has been chosen to reflect the problem formulation – and the content of the thesis – as accurately and sparsely as possible:

Multiclass Cross-selling Model for Savings and Investments Using Gradient Boosting

Explaining non-trivial components of the title:

Multiclass Within machine learning the problem of classifying instances into one of three or more (in this case six) classes.

Cross-selling Aiming at current clients that do not own any of the target products.

Gradient boosting A machine learning technique which produces a prediction model in the form of an ensemble of weak prediction models.

Regarding the four key chapters:

2. **Background**
   Basic insights and formal definitions regarding key business and theory aspects. Most focus is on business, as the reader is assumed to be familiar with basic theory.

3. **Theory**
   Although highly relevant to the thesis, the content is unrelated to this specific project. Readers interested in specific aspects of the theory are referred to the source material.

4. **Modelling**
   The process and methods of this specific project. Some theory tied into the chapter, when deemed relevant. Much of the analysis will be performed throughout the chapter, as the results are presented.

5. **Discussion**
   Key insights and obstacles. Contains some general analysis.

Chapter 2

Background

2.1 Research background

Machine learning as an academic term emerged as early as 1959 [1], when studying the game of checkers. However, research in proximal fields had been going on since World War II as the field is closely related to (perhaps even a merge of) computer science and mathematics. There is no single formal definition of what constitutes as machine learning, but a general definition could be formulated as simply as

> *An algorithm that learns from experience rather than requiring explicit guidance.*

Whereas a conventional program needs to be explicitly told that "an e-mail fulfilling conditions C is spam", a machine learning algorithm simply requires (large) amounts of spam and non-spam email to be able to make distinctions. As such, it can adapt its pattern of detection given new data, whereas the conventional program requires to be manually rewritten. Today, machine learning is not only a popular field on its own, but is also present as an important tool in other fields, and researched by almost every major university that has a technology/nature science faculty. Machine learning is often classified into two categories:

Supervised learning: When both input as well as desired output are given to an algorithm, with the intent to find a good mapping between the two (which can then be used to properly classify new input). For instance: decision trees (e.g. this project), linear- and logistic regression, support vector machines and artificial neural networks.

Unsupervised learning: When only input is given to an algorithm, without any labeling, with the desire to find a structure. For instance: clustering and principal component analysis.

In general, a data set in machine learning is split in two different sets; **train** and **test**. The model is tuned on the train set, but its performance is ultimately tried on the test set. As such, it is very important for the modeller not to include any information from the test set when creating the model. It is not uncommon to have multiple test sets, e.g. one for "day-to-day tuning" and one for "testing the final model". Another method is to have algorithms randomly split ones data into test and train for every new tune, and thus ensuring that the test set is never twice the same.

In addition to what has been stated above, the reader is assumed to be familiar with introductory mathematical concepts as well as basic machine learning.

2.2 Business background

Danske Bank (translates to Danish Bank) is Denmark's largest bank. For almost 150 years, they have helped people and businesses in the Nordics and are currently present in 16 countries with over 20 000 employees. Today, they serve personal, business and institutional customers, and in addition to banking services, they offer life insurance and pension, mortgage credit, wealth management, real estate and leasing services [2].

Figure 2.1: Danske Bank Group corporate logo

The thesis has been conducted with Commercial analytics - a unit within Group Development responsible for data driven analysis. Employees here have backgrounds in all natural sciences, at masters or PhD level. In recent years, the demand for cutting edge data driven analysis (which includes machine learning) has increased throughout the entire financial sector, and consequently the unit has grown rapidly.

The following terms, relevant for the thesis, as defined by the bank [2]:

Analytics Any form of analysis of data, including manipulation, aggregation, segmentation and visualization. For example, Danske Bank uses analytics for product development, reporting, segmentation and modelling.

Modelling Any form of process for creating or employing a model.

Model Any scriptable transformation of data that seeks to derive information not readily available in the data through aggregation or segmentation.

Segmentation Any limits placed on attributes in a data set to divide it into subsets.

Profiling Any form of automated processing of personal data consisting of the use of personal data to evaluate certain personal aspects relating to a natural person, in particular to analyze or predict aspects concerning that natural person's performance. Profiling is a concept that enables businesses, organizations and public institutions to collect and use personal data to determine, analyze and predict an individual's personality or behaviour, interests and habits. Almost any data can potentially be used to profile. Danske Bank uses profiling for different activities and purposes. First and foremost, profiling helps us to create better experiences for our customers. For example, profiling enables the bank to target and personalize its marketing and offer products and services that are relevant to specific customers based on their personal preferences. But profiling is not only used for marketing purposes. It is also used for e.g. credit ratings and risk assessments.

The bank has been using machine learning techniques for the past few years, and is currently expanding its data driven analysis units. As far as gradient boosters go, they are frequently being employed. Employees responsible for machine learning analysis have a solid (theoretical as well as practical) understanding of various kinds of boosters as well as other machine learning algorithms.

As mentioned in the previous chapter, this project emerged due to a desire to predict customer behaviour on a deeper level than currently done. With the current category level prediction model, advisers get leads on company defined categories (who are not available to customers), and then have to spend time preparing appropriate products. Successful prediction on product level would save time and increase accuracy.

Chapter 3

Theory

Most theory here covers general topics relevant to the thesis as a whole; niched theoretical aspects relevant only to sub-sections are explained in the (next) chapter MODELLING.

3.1 Loss functions

The Loss function is simply a method for evaluating the result of a model compared to the true value. As such, any function (including constants, even though they would be useless in practice) could theoretically be passed as a valid loss function. Many loss functions measure the "distance" between right and wrong, and thus it's often an objective to minimize them (e.g. as part of an objective function).

However, even though any function can be a loss function, the choice is anything but arbitrary. There are numerous properties that can be used to define the choice of loss function. We will focus on four that provide obvious benefits, the so called **completeness** properties [10]:

1. **Completeness of Information**
   Ensures that new observations x_n contain information about a parameter function which can be used for reducing the risk of estimation of said parameter function, i.e. that one should use all available observations.

2. **Lack of Randomization Condition**
   Restricts attention to non-randomized estimators since it shows that for every randomized estimator there exists a non-randomized one with less or equal risk.
   Note: An estimator is randomized if it doesn't only depend on deterministic observations, but also on random variables.

3. **Symmetrization Condition**
   The estimation of the value of the parametric function should be independent of the order in which the observations were received and analyzed.

4. **Rao-Blackwell Condition**
   Allows for rejection of non-informative components in the selection. A strengthening of the previous condition.

The formal proof for all properties has been omitted but can be found in [10]. The proofs might not be trivial, but it's trivial why the conditions are desired for a loss function - they greatly reduce randomness when estimating. The natural follow-up question thus becomes; which loss functions are complete, i.e. satisfy these conditions?

Theorem 3.1.1. *A family is weakly reducible iff for each integer $n \geq 2$ and each choice of real numbers $s_1, ..., s_n$ there exists a point $s_n^* = s_n^*(s_1, ..., s_n)$, which is a measurable function of $s_1, ..., s_n$ and such that*

$$\frac{1}{n}\sum_{i=1}^{n} W_t(s_i) \geq W_t(s_n^*) \quad , \quad \forall t \in T$$

Further, if the family is continuous and there for every $t \; \forall \; T$ exists limits such that $\lim_{s\to\infty} W_t(s) = \lim_{s\to-\infty} W_t(s) > W_t(x)$ for all $x \in \mathbb{R}$, the family is reducible.

Given a general loss function $w(s,t)$ it has to belong to a so called **reducible** family (for some conditions strongly- or weakly reducible is specified) to satisfy all four conditions [10].

There are a lot of functions that satisfy [3.1.1]. For instance, it can be proven that an arbitrary family of convex and real functions on $\mathbb{R}$ is reducible [10]. Even simpler, e.g. $w(x) = |x|$ is reducible (albeit not strictly convex). But to ensure satisfaction of all four completeness conditions, a strongly reducible function is required, e.g. the so called **quadratic loss function** $(\hat{\theta} - \theta)^2$ where $\hat{\theta}$ is an estimation of θ. Note that even this loss function has its limitations; being unbounded it limits the class of estimators to those with finite second moment.

3.2 Boosters

To understand Boosters, it is crucial to understand the so called **Decision tree**. Trees are a flowchart-like method for arriving at a result by repeated categorization of data. At every **branch**, data is **split** into one of several groups until it finally is classified into one of several **leafs**.

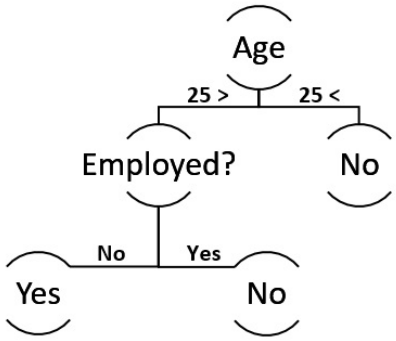

Figure 3.1: Novel tree attempting to answer "Is person a student?" with three leafs ("Yes", "No" and "No") and two branches ("Age" and "Employed?"). It crudely predicts that an unemployed person below age 25 is a student. Of course, any model is only as good as its data; providing the novel tree with data on young adults (e.g. age 18-30) will probably yield a higher accuracy than providing it with general population data (where it would e.g. classify even small children as students).

Boosters grow trees sequentially, with every new tree reducing the error of the previous one. In essence, they are the answer to a question first asked (but not answered!) three decades ago by Michael Kearns; paraphrased as "*Can a set of weak learners create a single strong learner?*" [3]. As it turns out, the answer is yes.

A **weak** classifier is one whose error is, roughly speaking, only slightly better than random guessing (i.e. 50% accuracy). Thus, a single weak learner won't produce any useful result. However, the prediction from several of them can be weighted through a "majority vote", employing the so called **wisdom of the crowd**, to produce a final prediction G(x), which is considered a classification by a single **strong** learner:

$$G(x) = sign\left(\sum_{m=1}^{M} w_m G_m(x)\right) \tag{3.1}$$

where w_m is the weight of the m:th weak learner for M weak learners. This is essentially Boosting [4]. Note that the method is similar to so called **B**ootstrap **agg**regat**ing** (Bagging), except that the underlying models are not built in parallel nor are random, but instead constructed sequentially based on the performance of the previous ones. Of course, selecting classifiers G_m is a field in itself. Nonetheless, for a more formal definition we need to introduce a few key concepts [3] (to improve and ease understanding, the original formulations have only been slightly modified):

Let c be a parameterized class of representation of a Boolean function; that

is, $c = \cup_{k \geq 1} c_k$ where c_k is a representation of a Boolean functions on $\{0,1\}^k$. Let further c be **polynomially evaluable**, i.e. that there is a polynomial-time algorithm that on input of a representation c and a vector x computes the value of $c(x)$. We assume that the representations of c is written under some standard encoding, and will denote the length in bits by $|c|$. For a given target representation c we further assume that there are fixed but arbitrary target distributions D^+ and D^- over the positive and negative examples of c respectively. Finally, assume access to two **oracles** (i.e. an abstract black-box machine used to study problems) POS and NEG that sample these distributions.

Using two classes of type c, the **target class** C and the **hypothesis class** H, we can now introduce the formal definitions of strong and weak learnability as put forward by Kearns [3]:

Definition 3.2.1. C is **strongly learnable** by H if there is an algorithm A with access to POS and NEG, taking inputs $0 < \{\epsilon, \delta\} < 1$, with the property that it – for any target representation $c \in C_k$, for any target distributions D^+ and D^-, and for any ϵ and δ – runs in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, $|c|$ and k and outputs a representation $h \in H$ that with probability at least $1 - \delta$ satisfies $D^+(c-h) \geq \epsilon$ and $D^-(h) \geq \epsilon$. We have identified c and h with the set of points on which they evaluate to 1.

Definition 3.2.2. C is **weakly learnable** by H if there is a polynomial p and an algorithm A with access to POS and NEG, taking inputs $0 < \delta < 1$, with the property that it – for any target representation $c \in C_k$, for any target distributions D^+ and D^-, and for any δ – runs in time polynomial in $\frac{1}{\delta}$, $|c|$ and k and outputs a representation $h \in H$ that with probability at least $1 - \delta$ satisfies $D^+(c-h) \geq \frac{1}{2} - \frac{1}{p(|c|,k)}$ and $D^-(h) \geq \frac{1}{2} - \frac{1}{p(|c|,k)}$.

As stated above, these defitions are the formal ways of stating that weak learners are classifiers that have an accuracy only slightly better than 50%, whereas strong learners can be arbitrary accurate. Going back to the original question, it can be formulated the way Kearns himself posed it: "*Is it the case that any C that is weakly learnable is in fact strongly learnable?*" This question was answered two years later by Robert Schapire [5], who effectively proved that the notion of strong and weak learning are equivalent; one can convert – or "boost" (hence the name) – a weak learner into one that achieves arbitrarily high accuracy (i.e. into a strong learner).

Theorem 3.2.1. *C is weakly learnable iff it is strongly learnable*

Trivially, strong implies weak. But Schapire [6] managed to prove the converse too, and subsequently invented AdaBoost (see 3.2.1) as a result. Essentially, he described a technique by which the accuracy of a weak learner could be boosted by a small but significant amount. Then he shows how this mechanism can be applied recursively to make the error arbitrarily small - a strong learner.

As apparent from [3.1], Boosters rely on their choice of classifier G_m. Being

themselves sums of other, even simpler, functions b (characterized by a set of parameters γ_n) these classifiers are also denoted as **basis functions** [4]

$$G_m(x) = \sum_{n=1}^{N} \beta_n b(x; \gamma_n) \in \{-1, 1\} \tag{3.2}$$

where β_n is the expansion coefficient. For instance, $b(x; \gamma) = 1/(1 + e^{-\gamma x})$; the so called **sigmoid function**. Typically, basis functions are fit by minimizing loss functions averaged over the training data. As will be seen below; there are various kinds of boosting. Even though the newer boosters are of type **Gradient booster** (e.g. XGBoost and Light GBM are both Gradient Boosting Decision Trees), there was initial success with so called **Adaptive boosters** (e.g. AdaBoost).

3.2.1 AdaBoost

AdaBoost hasn't been used for this thesis. However, due to its historical importance for boosters, a summarized description has been included.

The first widely used boosting algorithm, who remained the most popular one until the inception of **XGBoost** (2016), came a decade after the initial Kearns paper [3]; **AdaBoost** (a so called **Ada**ptive **Boost**er) (1995). It successfully trained multiple weak classifiers on weighted versions of the input data before combining them into a final strong prediction:

Algorithm 1 AdaBoost [4]

1: Initialize the observation weights $w_i = \frac{1}{N}, \quad i = 1, 2, ..., N$
2: For $m = 1$ to M:
3: Fit a classifier $G_m(x)$ to the training data using weights w_i
4: Compute errors ϵ_m
5: Compute $\alpha_m = log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
6: Update weights $w_i, \quad i = 1, 2, ..., N$
7: Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

Despite it's relatively "simple" layout, AdaBoost can dramatically increase the performance of even a very weak classifier. For instance, consider a two leaf classification tree. On its own it will (based on empirical testing, as the result is stochastic [4]) yield a test set error that is only slightly better than random guessing. However, with 400 boosting iterations the error is down significantly, reduced by a factor of four.

Note that the booster is basically just a straightforward implementation of [3.1]; for every iteration it updates its weights to increase accuracy of classification.

Although not initially designed for this purpose, it was later discovered [4] that AdaBoost also happens to minimize the so called **exponential loss function**

$$L(y, G_m(x)) = \exp\left[-yG_m(x)\right] \tag{3.3}$$

3.2.2 XGBoost

The e**X**treme **G**radient **Boost**er, known as **XGBoost**, is currently one of the most powerful and popular boosters, according to the creators due to its "scalability in all scenarios" [7]. It's a gradient booster, which means that gradient descent optimizers are used to sequentially add new "weak" models to improve the final "strong" model. Each new model is fitted based on the residuals of the prior models.

The following regularized objective is minimized [7]:

$$\sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \quad , \quad \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2 \tag{3.4}$$

for n data points and K different tree structures where T is the number of leaves in the tree and w are the leaf weights. l is simply any differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i (e.g. RMSE or a Log-loss function). Ω is the so called **regularization function** and controls the complexity (to avoid overfitting), with γ and λ being tuning parameters. Note that complex models with several leaf notes (i.e. large T) are penalized, but the penalty can be adjusted using γ. Depending on the loss function, there's no guarantee that every objective function can be analytically derived, so a second order Taylor approximation of [3.4] is used instead. Note that [3.4] includes functions as parameters, and thus additive methods are used to optimize it. For a fixed tree structure the optimal weight is given by

$$w^* = -\frac{\sum_i g_i}{\sum_i h_i + \lambda} \tag{3.5}$$

where h and g are first and second order derivative of the chosen loss function. [3.5] can be inserted into [3.4] to calculate the optimal value, which can be used to measure the quality of a tree (since trees in practice seldom are optimal).

A major problem when considering trees in general, is to find the optimal split. A primitive booster would simply enumerate over all possible splits on all features before picking the best option, and for a limited set of (preferably discrete) features this would be doable. However, when dealing with complex models that contain multiple continuous features, it becomes computationally too expensive. XGBoost solves this by using an approximate algorithm [7]; it first proposes candidate splitting points according to percentiles of feature distribution. The goal is to find candidate split points $s_{k,j}$ (for the k:th tree structure) such that

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon$$

for a rank function $r_k(s)$ and an approximation factor ϵ. This rank function represents the proportion of instances whose feature value is smaller than a certain threshold and is designed such that there will roughly be $\frac{1}{\epsilon}$ points. The algorithm then maps the continuous features into buckets split by these candidate points, aggregates the statistics and finds the best solution among proposals based on the aggregated statistics. The gain of every split is calculated through comparison with "the optimal gain", which is found using [3.5].

Another common problem for boosters is how to handle sparse data, which is common in real world applications (e.g. whenever excessive encoding is used). According to the original XGBoost paper [7], most tree learning algorithms are optimized for dense data, and handle sparsity naively. XGBoost's improvement comes mainly from visiting only non-missing entries whilst treating the missing ones as missing values. Unlike previous boosters, XGBoost has **default directions** in each tree node; whenever values classified as missing values show up, they are simply directed to the default direction, the classification algorithms thus only has to spend time on non-missing entries.

Early 2017, i.e. roughly the same time as the deployment of the first stable version of Light GBM (see 3.2.3), a histogram setting was added to XGBoost. Unlike the default XGBoost "tree-grower" it uses only a subset of possible splits and, rather than generating new sets of bins for each iteration, it reuses bins over multiple iterations. This made the algorithm significantly faster, at the cost of slightly reduced accuracy.

3.2.3 Light GBM

The **Light G**radient **B**oosting **M**achine was developed by Microsoft [8] and deployed in early 2017. Its algorithms increase both efficiency and scalability for high feature dimension and large data size.

Handling big data efficiently is non-trivial, even when using a straight-forward approach such as data- and feature reduction. However, it is this exact straightforward approach that Light GBM has chosen. The key to its success lies in two new algorithms for dealing with both aspects of the approach [8]:

Data reduction: Gradient-based One-Side Sampling (GOSS)
Since Gradient Boosters, unlike Adaptive Boosters, don't have any native sample weights, sampling becomes a non-trivial problem. Instead, gradients are used. The general idea is to simply discard low-gradient data instances. Simple as it may seem, it can't be done straight-forwardly as the data distribution might be changed by doing so. Instead, the algorithm keeps instances with gradient over a certain threshold, the top $a \times 100\%$ instances, and performs random

sub-sampling among the remanding data ($b \times 100\%$ instances). To compensate for any eventual non-stationarity, GOSS uses the constant multiplier $\frac{1-a}{b}$ to amplify the sub-sampled data instances. The multiplier will ensure focus on under-trained data whilst keeping the distribution mostly unchanged.

Feature reduction: Exclusive Feature Bundling (EFB)
Many times, sparse feature spaces contain mutually exclusive features, i.e. certain features never take non-zero values simultaneously (e.g. encoding). These kinds of features can be bundled into special single features called "exclusive feature bundles". A correct implementation of the bundling algorithm will generate the same feature histograms as when using all the individual features. Note that the EFB algorithm is a mere approximation (albeit a very efficient one), since

Theorem 3.2.2. *The problem of partitioning features into a smallest number of exclusive bundles is NP-hard*

which can be proven by reducing a NP-hard problem known as "the graph coloring problem" to the partitioning problem at hand. This is also the way in which the algorithm is designed; an adaptation of an efficient solution to the graph coloring problem.

Light GBM allows the user to choose between various objective functions [8], the default multiclass one being the so called **logloss objective function**

$$-\sum_{c=1}^{M} y_{o,c} log(p_{o,c}) \tag{3.6}$$

where M is number of possible classes, c the class label, o the observation, y a binary indicator for whether c is a correct classification for o and p the probability that o is indeed of c. The binary case $M = 2$ is simply

$$-\left[y \log(p) + (1-y) \log(1-p)\right] \tag{3.7}$$

3.2.4 Other boosting algorithms

Boosters (like the ones already mentioned) commonly only handle numerical data; any categorical data has to be encoded into numerical. Shortly after the release of Light GBM, a new open source booster was released that could handle categorical data: **Catboost** [15]. This **Cat**egorical **boost**er uses various statistical methods on combinations of categorical and numerical features. The method is dependent on properly tuning and identifying the categorical features.

3.3 Hyperparameter optimization

Hyperparameter is a commonly used term in Machine Learning applications. Although sometimes used interchangeably, the term is de facto different from

Parameter. A parameter is set during training by the model whereas a Hyperparameter is manually decided before training takes place.

Most models have a wide range of hyperparameters that can be adjusted, and it is rare to tune them all. Many of them can be set as their default values, and it is not uncommon to gain the most benefit by tuning a few key parameters. The range of parameter values being considered for optimization is often referred to as the **parameter space**. Of course, in an ideal scenario, all parameters should be considered for a parameter space that covers all possible reasonable combinations, however, this approach is neither time nor computationally viable. Rather, (more or less) sophisticated methods are employed for testing a chosen parameter space. Trivially, the optimal hyperparameters are those that minimize the specified loss function.

3.3.1 Grid search

This is the "naive" approach. Given a parameter space, grid search will train and evaluate the model for every possible combination of the provided hyperparameters [12]. However, despite being obviously computationally inefficient (even for smaller parameter spaces), it doesn't require any synchronization for the different models being trained. Thus, it can train multiple models simultaneously in parallel (a so called **embarrassingly parallel** problem). When the parameter space is small, perhaps three or fewer dimensions, grid search is a good option as one can provide a fine mesh and still be - per construction - guaranteed to find the best combination within the provided space. [12]

3.3.2 Randomized search

As the parameter space grows, it is no longer viable to explore every combination. Instead, a more feasible approach is to randomly pick parameter combinations in the parameter space. This might seem like an inferior method, but it is in fact (especially for high-dimension parameter spaces) more efficient than grid search [9].

For a **hyperparameter response function** Ψ the object is minimization over the hyperparameter space [9]. This function most definitely has a **low effective dimensionality**, i.e. is more sensitive to changes in certain parameters (since rarely, if ever, every hyperparameter affects the model equally).

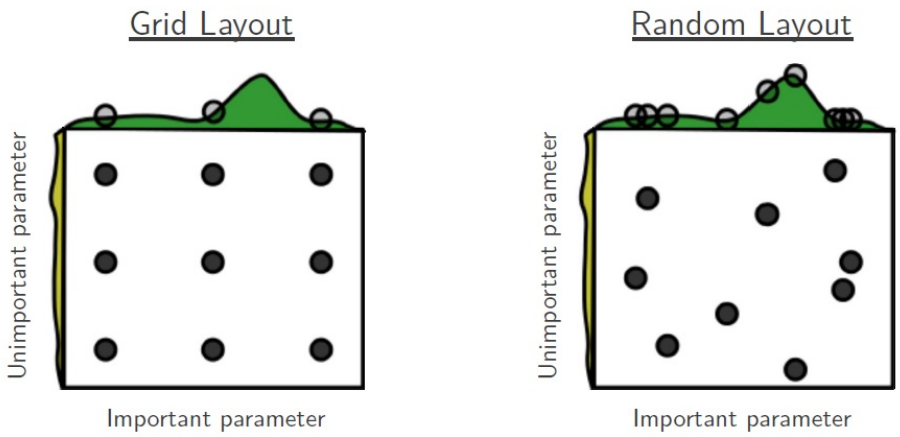

Figure 3.2:

Consider Figure 3.2; A simple low effective two dimensional parameter space for a function consisting of two parameters (whom are visualized around each square); note that one is by far more dominant (labeled **Important parameter**). Even though both of the methods evaluate nine combinations each, the grid only manages to sample three distinct values of the important parameter, whereas the random search samples nine distinct values. Of course, one has to provide reasonable distributions for the random search, but even a fine uniform mesh will evaluate more distinct values compared to a course grid layout. In the figure, as only nine combinations are evaluated, the grid is very coarse since it only has nine locations that all must be evaluated, whereas the random grid can still be arbitrarily granular since there are no restrictions on how many locations that can be provided. In higher dimensions, this difference is only further enhanced in favor of randomized search.

Unless you can have an arbitrary fine mesh for your grid search, the random search will be better [9]. In fact, random search has all the practical advantages of grid search (e.g. conceptual simplicity, easy implementation, embarrassing parallelism). Its main trade-off is a small reduction in efficiency in low-dimensional spaces for a large improvement in efficiency in high-dimensional search spaces.

3.3.3 Bayesian search

Even though randomized search might be superior to grid search, it is still "naive" in the sense that every new evaluation is independent of the previous one. There's no consideration of the sub-space it is investigating. Basyesian search will "guess" a new set of parameters to evaluate, based on the performance of the previously evaluated set. Its guesses are essentially a trade-off between exploring new areas of the parameter space, or further probe areas

with known (good) performance [11].

Brielfy explained [11], Bayesian optimization assumes that the unknown function was sampled from a (often Gaussian) process; to pick the next combination it optimizes the expected improvement over the current best result. Unlike gradient based approaches, it doesn't rely on Hessians or convexity, instead the computational powers are spent on determining the next combination to evaluate rather than actually evaluating it. As models become increasingly complex, this approach can save considerable time. However, it requires user-provided prior probabilities (so called **hyperpriors**) for the evaluated parameters (often a Gaussian prior will do, and if not, many hyperparameters can be empirically evaluated).

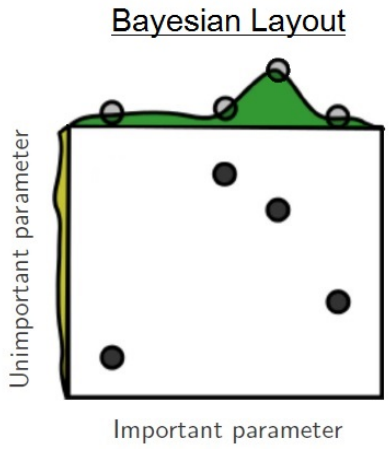

Figure 3.3: The Bayesian grid might start randomly; in this example bottom left and mid-bottom right. It might then consider something in-between, and then finally try a fourth parameter set near the most recent one (based on it being better than the first two).

Despite its apparent superiority, Bayesian optimization isn't as widely used as randomized (or even grid!) search. Perhaps due to its aforementioned dependence on choosing suitable hyperpriors, and since its more difficult to implement compared to grid- and randomized search.

3.3.4 Other optimization techniques

The human touch is not to be underestimated. Often an optimal method of finding the best setup when using "naive" optimizes, is to run several iterations, and manually tune the parameter space after every iteration. One doesn't even have to specify a "rectangular" grid as that of grid search, but could rather try custom grids. The optimal parameter setting shouldn't be found in the outer

limits of the space, as it may indicate that an even better combination exists beyond the frontier. Thus, manual tuning of algorithms should aim at capturing an optimal combination that's safely contained within the space.

As far as other automated techniques go; as it is possible to compute the gradients of hyperparameters, one could optimize using gradient-based tools. However, as mentioned above, gradient-based tools can be computationally heavy.

3.4 Feature engineering

A **feature** is best described as an attribute shared by all individual elements of data (e.g. a column). Although essential in machine learning, feature engineering is a de facto informal topic. It is about manipulating data features (e.g. removing unwanted ones, adjusting existing ones or adding self-made) in order to simplify data interpretation for the machine learning algorithm of choice. Preferably, one would engineer their features first, before trying to optimize the hyperparameters. Even though sophisticated automated feature engineering tool-kits have started to emerge [13], optimal features are best identified through expert knowledge of ones data. Feature engineering might be one of the main reason why machine learning isn't yet fully automated, as it depends heavily on intuition, context and practical experience. As put by world-renowned computer scientist professor Andrew Ng [14]:

> *"Applied machine learning" is basically feature engineering.*

There are multiple ways of performing feature engineering; a common method is to rely on various forms of grading (often provided by the machine learning algorithm of choice) to separate features that contain relevant information from those that do not. For instance when working with boosters, a metric may be how often a feature is split at the nodes; more splits imply more usage of said feature whereas e.g. zero splits imply no usage of the feature. Creating new features is also heavily dependant on the machine learning algorithm of choice; a common method is to transform existing features so that they better "fit" ones algorithm.

3.5 Hierarchical clustering

Clustering means to group data into subsets (or clusters) based on statistical features and is perhaps the most common form of unsupervised learning [20]. Every cluster should be internally coherent and distinct from the others. Note that this is not **classification**, which is a supervised form of learning!

There are numerous forms of clustering, and many require the number of clusters to be prespecified. However, so called **hierarchical clustering** does not.

This form will output a hierarchy - a structure that is typically visualized by a **dendrogram** (see figure 3.4).

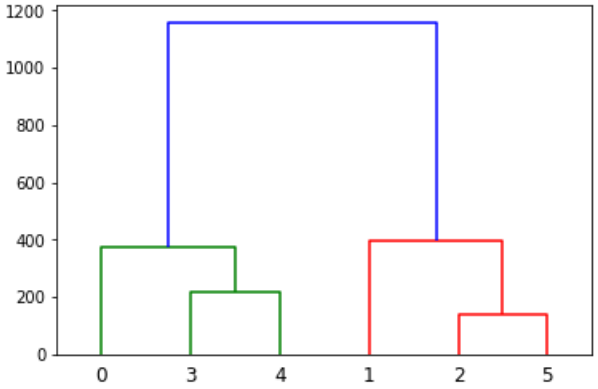

Figure 3.4: Dendrogram created with randomly generated data. The y-axis (length of dendrogram) is the similarity of the two clusters that were merged. For instance, elements 2 and 5 are more similar to each other than 3 and 4 are. This value (the height) can be used for manual partitioning, by cutting the dendrogram at some pre-determined height.

Although ultimately different based on data, there are a few ways to determine the cutting-point [20]:

1. Cut at a prespecified level of similarity.
2. Cut where the gap between two successive similarities is largest, i.e. creating "natural" clusters based on the shape of the dendrogram. For figure 3.4 this could be at 300; creating four clusters.
3. Apply a formula, e.g. the median similarity value.
4. Prespecify the numbers of clusters, and select a cutting point accordingly.

The standard algorithm for hierarchical clustering is called **Hierarchical Agglomerative Clustering** (HAC). It basically treats each data point as a single cluster at the outset and then successively merges (or agglomerates) pairs of clusters until all clusters have been merged into a single cluster that contains all data points [20]. The "merges" and similarities can be depicted using a dendrogram: Each node is a cluster and has two "children" representing the clusters that formed it. Each leaf is a data point. The root will always contain the entire analyzed data set.

The key to clustering lies in the choice of two cluster dissimilarity measures, **Metric** and **Linkage criteria**, which decide how "close" two clusters are. Metrics are a mathematical field of their own; they concern how one calculates the

distance between elements. For numerical data, the measure can be trivial (e.g. Euclidean), but there are measures even for non-numeric data. Linkage measures the dissimilarity between groups. Common measures includes various metrics between individual data points in the two clusters being analyzed (e.g. "single linkage"; smallest distance between two clusters).

There is no single right method for picking cluster dissimilarity measures; it depends entirely on the data. If lucky, experimentation and proper visualization might reveal "natural" clusters within the data. Clusters can be used to e.g. exclude features/data points that are too dissimilar to the rest of the data and/or to find hidden patterns and similarities.

Chapter 4

Modelling

As detailed numbers regarding e.g. product purchase is considered confidential information, graphs will either - if possible - be normalized or - if not possible to normalize - be plotted without their descriptive axis.

4.1 Target

Note! The initial target consisted of an additional product. However, when initial (basic) modelling began, it was discovered that this product behaved strange compared to the others; being modelled very poorly. After some business research, it was discovered that this product was in fact a bundle of several other (to each other unrelated) products. It was therefore removed from the target.

Crucial for any model is the Target; the feature which you want to understand and create a model around. Depending on the target, a machine learning algorithm is picked. For this thesis the target is of **multiclass** type, i.e. non-binary containing more than two classes. More specifically, the target consists of five different products and one non-purchase class - six classes in total. The products (exclusively of type **Investment** or **Savings**) are listed below according to their encoding number (and will henceforth be interchangeably referred to as either their name or their encoding number):

1. **Flexinvest Fri**
   Professional care and advisory for investments of 100 000 DKK or more.
2. **June**
   An algorithm-driven portfolio management tool based on **Modern Portfolio Theory**. Customers choose one of five funds based on personal risk aversion. Minimum invested sum is 100 DKK.
3. **Mutual funds**
   A professionally managed investment fund that pools money from multiple investors in order to purchase securities.

4. **Savings one off**
   Any single deposit into a savings account (note: an account used for saving money is not necessarily the same as a savings account).
5. **Savings recurring**
   When numerous consecutive savings fulfill a set of requirements the account is considered as this product.

Of these products, Mutual funds had to be further adjusted, using provided "corrections data" to remove certain sales that would otherwise distort the prediction. Savings one off is the odd one out, as it can be argued whether or not it's actually a product in the same sense as the rest; however, it may indicate a possible first contact with the bank (e.g. a customer acquiring a savings account). But it may also be someone allocating capital (perhaps for purchasing a different product). Savings one off is, as can perhaps be guessed from its description, the most common product acquired:

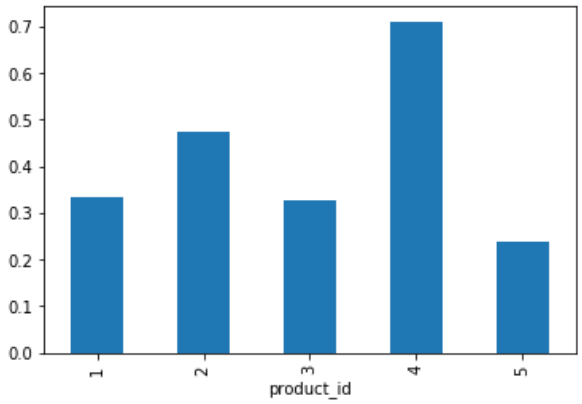

Figure 4.1: Normalized distribution of products. Expected result given product descriptions (i.e. Savings one off being the most popular).

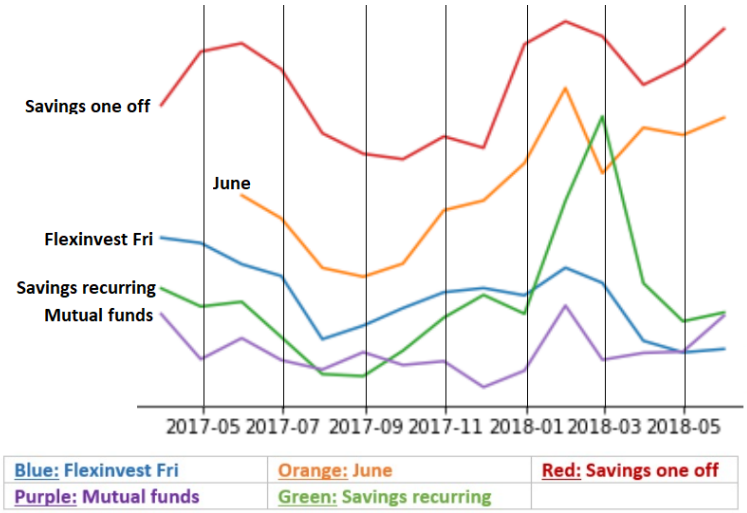

Figure 4.2: Time series depicting product purchases over time. X-axis denotes time and Y-axis denotes number of products (restricted from business). Note that June was launched after inception of time series, and that it has been (almost) steadily increasing ever since.

As can be seen, there's been a decline for Flexinvest Fri that contrasts against the inception of June (and increase of Managed account). Both June and Flexinvest Fri are essentially, from consumer point of view, automated investment products. The main difference is in the amount handled; June requires a minimum deposit of 100 DKK whereas Flexinvest Fri is aimed at investors with a minimum deposit of 100 000 DKK. Thus, it's not a stretch to imagine that new clients choose to allocate their assets to June (perhaps slowly over time) rather than putting (perhaps most of it) in a Flexinvest Fri account at once. June is also cheaper than Flexinvest Fri.

When modelling, an approach of one-date-one-sale was used, meaning that dates with multiple sales had to be handled. From a business point-of-view, it's argued that multiple sales on the same date imply a possibility that one (or more) of the products were sold on the advisers suggestion rather than independently chosen previously by the customer.

The non-trivial procedure of acquiring the final target can roughly be summed up as following:

1. Import two sets of data; one containing information about the sales (**Sales**) and another containing information about every customer (denoted by the bank as **ABT**).
2. Add to ABT an additional column denoting whether or not a customer

already owns a product of type "Savings" or "Investments". As the model is targeting first-time customers, those that already have purchased are not of interest for the target.

3. Label encode products (encoding presented above).
4. Date sales three months before occurrence; as this is when the sale is trying to be predicted based on existing observations.
5. Merge ABT with Sales; this will however include the dates where multiple sales have occurred (roughly 3% of events); since the target aim is to have one-date-one-sale, this has to be handled. Four methods were considered (in all cases some sales data will be removed) before **d)** was selected (reason explained below).
   (a) Remove all dates with multiple sales, and only keep those that have one sale.
   (b) Remove additional sales from every date, only keeping the first occurring sale on each date. As the sales are logged in an arbitrary order this won't necessary preserve the customer initiated sale.
   (c) Group all multiple sale events into a new category: class 7.
   (d) Reallocate duplicate pairs into their respective category and remove the 4's (Savings one off) from all pairs (e.g. [2, 2] is logged as a 2 and [4, 1] is logged as 1). Remove remaining multi-sales.
6. Match the final target to the ABT.
7. Remove all rows that indicate a previous customer purchase, or an initiated purchase, i.e. purchase within next three months; this will only leave the so called **negative targets**.
8. Re-add first product purchase for customers; these are the **positive targets**.

The complete target is now acquired. It might not have been clear why option d) was chosen at step 5. One has to look at some underlying data:

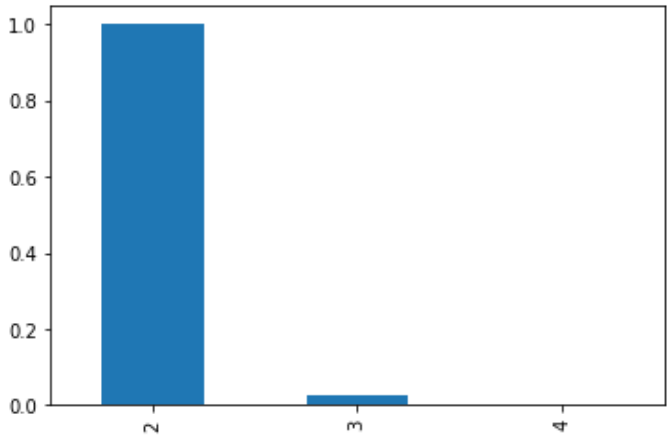

Figure 4.3: Normalized distribution of multisale events reveals absence of non-dual sales.

The first obvious insight is the almost complete absence of non-dual multisale events, which begs the question; what are the most common dual-sale combinations?

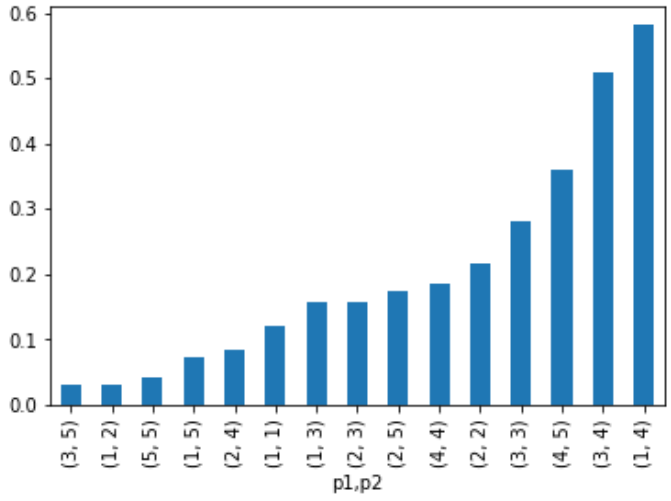

Figure 4.4: Normalized combinations of multisale events shows dominance of product 4; Savings one off.

There appears to be a clear dominance of pairs containing product 4. As suggested above; the pairs containing a 4 are most likely just about money being moved from savings to purchase a product, and should thus most likely be considered a sale event for the non-4 product in the pair. As the aim is to predict customer purchase, the duplicate sales can also be moved to their respective products (e.g. a [2, 2] sale to product 2). Doing these two adjustments would annihilate most multi-sales, as only 25% of its original volume would remain (equivalent to less than 1% of the full data).

Even still, there's the question of whether the pairs reflect the distribution of product occurrences. To test this, one million random product pairs were generated where the drawing was weighted according to the real data product distribution. If the real pairs are truly random, the generated pairs plot should match the real one.

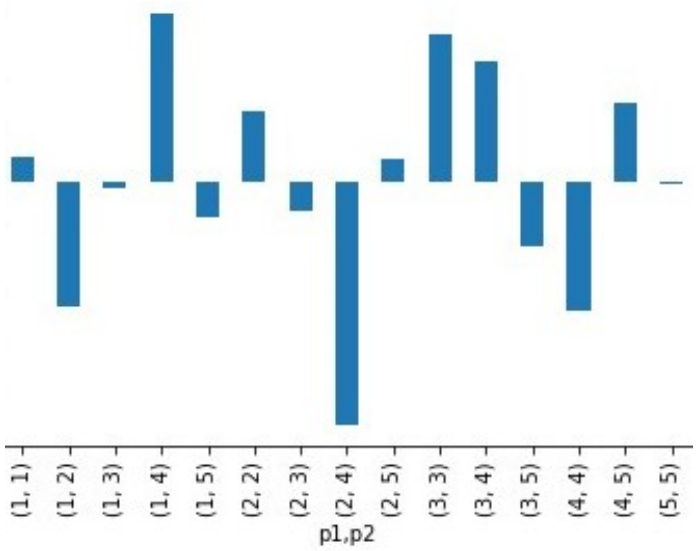

Figure 4.5: Real pairs minus generated pairs. Y-axis is percentage; difference is of one-digit integer size (i.e. not decimals) apart from pair [2,4]. Real pairs are apparently not randomly distributed.

The distribution of the real pairs is far from random. In fact, some of the lesser frequent random pairs are over 10 times as frequent as their real-world counterparts.

Looking more closely at the pairs more frequent in the real data, we see numerous pairs that occur more often than they should according to the random pairing, e.g. the most popular pair [1, 4] occurs twice as often as its random counterpart. This suggest that product 4 is popular for pairing. The expla-

nation can be as simple as customer convenience; first a sum is allocated into a savings accounts of sorts, being "Savings one off", and then transferred to purchase the other product of the pair, i.e. an allocation where the Savings one off simply is a "middle man" (as speculated above).

After reallocating the sales that contain a 4, and the dual sales, the multisale category becomes to insignificant to keep, and the rest of the pairs are discarded.

4.2 Basic model

Note! The target used in this section is a rudimentary one which proved insufficient for deployment due to business reasons - i.e. certain anomalous features were detected when doing advanced analysis (see next section ADVANCED MODEL). All results presented in this section are thus, although theoretically valid, not comparable to those in the next section (including the final model). This target should be seen as a simpler version of the final target.

In this thesis, the term **basic model** implies applying a package without much (or any) tuning. Any adjustments should be near-trivial and understandable for an amateur machine learner and not requiring any deep knowledge of the algorithms to employ. The subsections are in order of development; a basic model was first trained, then default multiclass models were trained before engineering their features. Finally, a hyperparameter optimization was performed.

4.2.1 Binary model

The very first model wasn't a multiclass. Instead, all products were bundled into one positive target (1) and the rest of the data got denoted as the negative target (0). Also, the first models were using only two months of data for training, rather than employing the full set. The reason was strictly computational; Python (Pandas) store loaded files directly onto memory; which is limited even for the banks Heavy Load Memory machines.

A third month (two months after the training set) is used for testing. The training months where deliberately chosen such that they will have the same test month as the full data set. This way, the simple model can be compared against the final one. The hypothesis is that the final model should predict more accurately, having far more data to train on, and if this is the case, the learning curve of the simple model shouldn't stagnate.

After suitable cleaning and preparation of both test and train set (e.g. merging the aforementioned target), standard XGBoost and Light GBM are used for the first models. All information has been converted/encoded to numerical types. As can be suspected, the classes are heavily imbalanced.

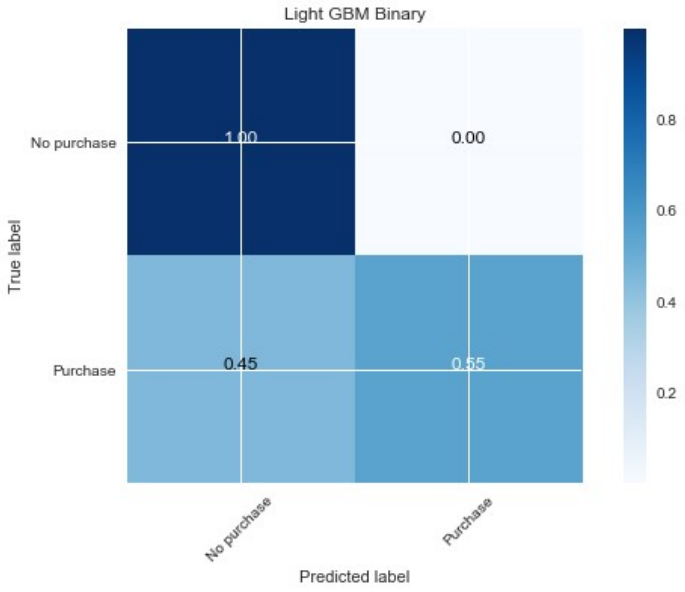

Figure 4.6: Normalized confusion matrix. Upper right value not actually 0; classes too imbalanced to reveal the significant amount of false positives. As this is the simplest model to generate, this will be considered the "worst case" performance.

Due to the heavy class imbalance the clients predicted as **False positive** (e.g. as incorrectly wanting to purchase, upper right corner) are seen as being 0. But there's a significant amount hiding behind the 0.00; especially considering that every client predicted as customer will result in a marketing attempt. Roughly one eighth of the purchase predictions would aim marketing at these clients that do not intend to purchase anything. Purchase was correctly predicted 55% of times, i.e. almost half of the potential customers would be lost with this model. On the plus side, Light GBM is very fast; training took less than four minutes.

The XGBoost model took two hours to run, which is 40x longer than the Light GBM. However, the results were also notably better; here the 0.00 is indeed 0 since there are virtually no false positives. The number of "lost" customers (i.e. potential purchases predicted as no purchase) remains the same. Thus, the benefit of using XGBoost seems to be that you'd avoid marketing towards customers with no intention to purchase: It is better at avoiding False positives.

Strictly counting marketing approaches, the two models both manage to capture roughly the same amount of potential customers actually willing to purchase, and loose roughly the same amount of them too. The difference lies in marketing towards unwilling customers; for the extra hours the XGBoost model would save roughly 12% of total effort – not an insignificant number.

Fortunately, as the models are both "out-of-the-package" and use only a small subset of the available data, there is room for improvement. Further, the models aren't even simple versions of the desired model; the one sought after is the multiclass one.

4.2.2 Multiclass model

The first basic multiclass models are, like the binary ones, trained on merely two months of full data.

The very first multiclass model employs Light GBM. As before, the non-purchases are virtually all correctly predicted; however, a significant percentage of the calls will be made to people with no intent of purchase. Regarding the **accuracy** (i.e. percentage of product purchases correctly predicted), the result can be considered mediocre at best; one third were correctly predicted. The percentage of correctly predicted customers (55%) is the same as when doing binary prediction: **No information is lost when predicting multiclass.**

The next model was the standard XGBoost. In the multiclass case this model took roughly half a day to train, compared to Light GBM taking 15 minutes. It did however manage to almost completely eliminate the false positives, which makes it well worth the extra hours: Every marketing outreach will be to a customer, albeit maybe for the wrong product. The accuracy remains at one third.

4.2.3 Feature engineering

Gradient Boosters, including XGBoost, use a measure known as **feature importance** [4] to indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. Essentially, at each node, the region associated with the node is partitioned into two sub-regions; within each a separate constant is fit to the response values. The idea is to choose variables such that one gets maximal estimated improvement $\hat{i}_t^2$ for the entire region associated with the node. To generalize over additive trees, one simply averages over said trees, which also gives a stabilizing effect. In practice, the feature measure values are relative, and thus the largest is often set to 100, and the others are then scaled accordingly [4].

Using feature importance, calculated automatically by XGBoost, columns are ranked and removed to see how significantly performance is affected. The measure ranks features based on how often they are split: more split means the feature is used more often. A feature with no splits isn't being used at all. To measure efficiency, the percentage of correct product purchase predictions is used as a first indicator (i.e. the same accuracy measure as before). With a full

set of columns, the accuracy is one third.

Both XGBoost and Light GBM have near-identical lists for their 15 most important features; only the internal order for a few of them is different. Some testing indicates that one can remove over 85% of the columns, and still retain the same accuracy (roughly one third) for both models. In fact, keeping only the 15 most important features (a fraction of the total) is sufficient enough for an accuracy of almost 30%.

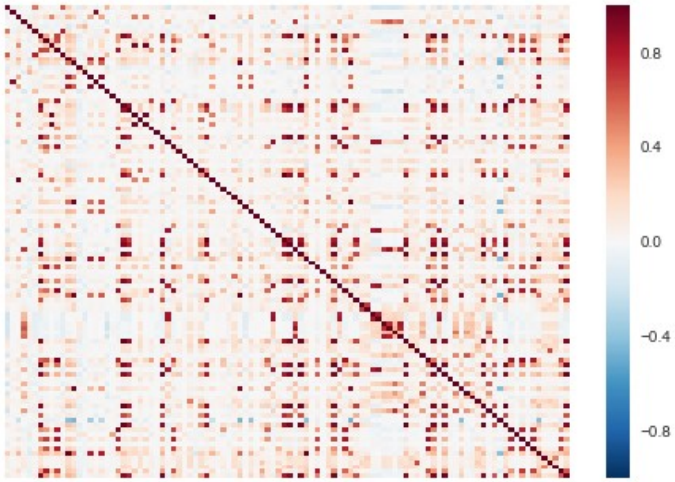

Figure 4.7: Correlation matrix for the top 100 features (ranked by importance). Clusters of correlation appear, but features mostly uncorrelated. A quick check indicates that many of the heavily correlated clusters are filled with NaN and have no business correlation.

There seems to be local clusters of correlation, which is to be expected as many columns simply provide the same information, albeit for a few (e.g. three) consecutive months. If then customers have regular savings, a correlation will appear. A quick check reveals that most of the highly correlated columns are indeed the same category but for different months (e.g. two investment columns that have nearly 1 in correlation; as they both almost entirely consist of NaN). Many of these columns should be only NaN, as the aim is to target clients with neither savings nor investment products, so the absence of NaN values in those columns indicates a missclassification (although, often the faulty values are no more than a handful of percent). The fact that some of these features also popped up as "important" implies that the boosters exploit this missclassification in some way. This will be dealt with in the next section Advanced model.

As far as the models and prediction goes, the feature engineered data with fewer columns generated little to no change, which in itself is a minor achievement as

both the data size as well as the runtime are reduced. In fact, the trimming of features reduced the data size to such an extent that the entire data now can (without much difficulty) be loaded directly into the RAM of the banks Heavy Load Virtual Machine.

When employing the full data, the result (for XGBoost) barely differs. In fact, the full data provides a slightly worse fit compared to when only using two months data. The accuracy is at just below one third and the overall prediction of customers is stable at around 55%. Notably, the amount of false positives have increased. Looking at a self-provided learning curve (generated using Light GBM), this result becomes apparent:

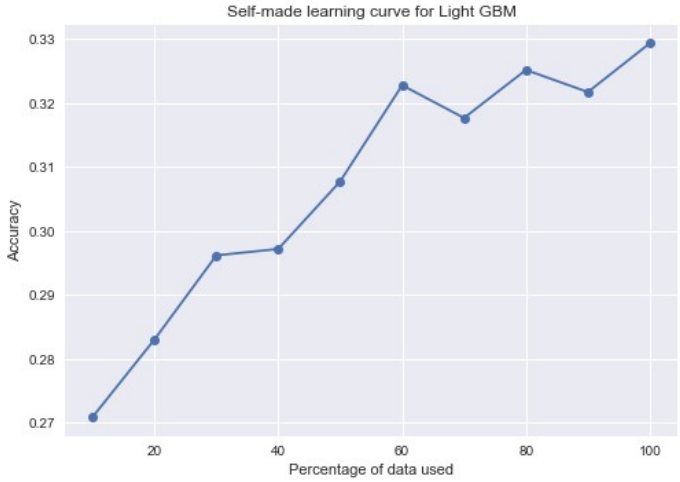

Figure 4.8: Graph implies that more data does not imply higher accuracy; there seems to be severe stagnation after using 60% of the data. The picture is deceiving, as the y-axis is broken. This knowledge will be used when determining what section of the data to use as "test".

4.2.4 Hyperparameter optimization

Boosters have numerous hyperparameters that can be tuned and with a dataset this major, it's highly non-trivial to find an optimal setting. Focus will thus be to tune a subset consisting of the most important parameters based on suggestion from senior data scientists at the bank who have had previous hands-on experience with applying boosters to similar data. As far as tuning methods go, there are two main approaches; grid- and randomized search (**see 2.3**). Both are crude and non-intelligent in the sense that they do not take previous results into consideration.

As the data is quite big, and the parameter space very large, randomized search

is the only viable option as it will cover a far larger section of the parameter space compared to grid search during the same time. Hyperparameter tuning was done for both Light GBM as well as for XGBoost.

So far, whilst the XGBoost has provided better results, it's been far slower. However, XGBoost has a histogram setting that allows it to bucket continuous features into discrete bins (much like Light GBM), thus saving time. A full run with reduced features took a mere hour, which is faster than the default XGBoost, but still slower than Light GBM. However, given the superior results, it is well worth the wait. The Histogram XGBoost didn't fail in performance, and provided a near-identical result to the default XGBoost with an accuracy of around one third and almost no false negatives (this was before optimizing any hyperparameter). Taking this into consideration; from this point and on-wards **only XGBoost was employed**. Light GBM wasn't accurate enough for this particular project.

An initial tuning of the non-histogram XGBoost using randomized search for five parameters with a total of 42 000 combination was finished in roughly eight hours. Even though the accuracy remains at a solid one third, there are certain improvements; only products 1 and 4 see a drop in performance. On the downside, there are a few more false positives, but still well within the range of being considered as insignificant. Looking at the cross validation, with RMSE as measure, there are no indications of overfitting with this setup. As far as the elementary methods go, which most people with basic knowledge of python can learn (given the prepared data), this was the best possible model.

4.3 Advanced model

An advanced model is one that requires either deep insights of the data, provided through e.g. professional knowledge or non-trivial statistical tools (for advanced feature engineering), or advanced non-trivial tools (such as implementation of a sophisticated hyperparameter optimizer).

There was an attempt to model the two categories Savings and Investments separately, and then merge the results into a single prediction. The hypothesis was that the two categories should have distinct behaviors, which the booster could exploit. However, when running two models and merging them, the result (albeit satisfactory) wasn't as accurate as when running a single model. There might be a business reasons behind this, e.g. that for customers savings and investments just are two sides of the same coin: asset allocation. Someone might purchase a Flexinvest Fri in order to invest a large sum, whilst someone else might purchase the same exact product for saving (because they have higher risk adversity).

The first attempt at sophisticated feature engineering was to exploit any even-

tual differences of distributions. Essentially, one product at a time was selected for analysis (the method doesn't differ depending on product); in this section we'll highlight **product four** who had most false negatives (but the method is similar for all products). The predictions were divided into **True negatives** (0 predicted as 0), **False negatives** (4 predicted as 0) and **True positives** (4 predicted as 4). **False positives** (0 predicted as 4) are de facto negligible. The idea was to find differences in distribution for the features in these three groups that could help manually categorize some of the mismatches.

Two statistical methods were used: **Bhattacharyya's distance** [16] (for categorical features) and **Kolmogorov-Smirnov 2-sample-test** [17] (for continuous features). Both of them compare the distribution of two identical features from two different datasets, and return a number depending on their similarity. Identity means 0 for Bhattacharyya, as it is a distance, and 1 for Kolmogorov-Smirnov, as it is a p-value.

Bhattacharyya's distance is closely related to the **Bhattacharyya coefficient** which can be used to determine how similar two (discrete) samples are. It has benefits over other similar measures, e.g. it also considers differences in standard deviations (and not only for mean). For two probability distributions p and q the distance is defined as

$$BD(p,q) = -\ln\left[BC(p,q)\right] \quad , \quad BC(p,q) = \sum \sqrt{p(x)q(x)} \tag{4.1}$$

If the distributions are identical, the sum will be one and the distance zero. However, if there's no similarity, the coefficient will be zero and thus the distance will go to infinity.

Kolmogorov-Smirnov 2-sample-test is used to determine how similar two (continuous) samples are. It is, like Bhattacharyya, superior to similar tests since it's sensitive to differences in both location and shape of the empirical cumulative distribution functions of the two samples being compared. The null hypothesis, that the samples (of size n and m) are drawn from the same distribution, is rejected at level α if

$$\sup_x |F_{1,n}(x) - F_{2,m}(x)| > \sqrt{-\ln\left(\frac{\alpha}{2}\right)\frac{n+m}{2nm}} \tag{4.2}$$

The left term is known as **the Kolmogorov-Smirnov statistic on 2 samples** where $F_{1,n}(x)$ and $F_{2,m}(x)$ are the empirical distribution functions of the first and the second sample respectively and x thus being an element of the sample. As the statistic can be seen as a p-value, it can then be treated accordingly (i.e. compared to tables with common levels for α).

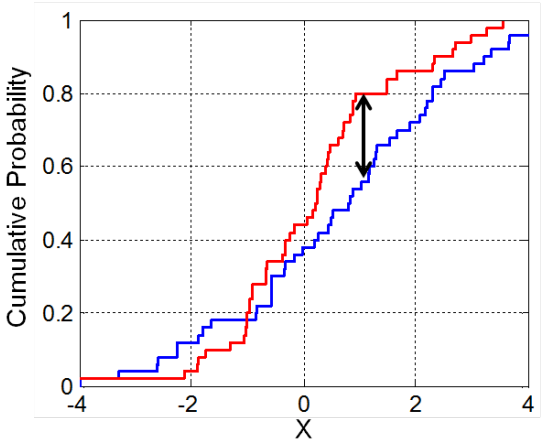

Figure 4.9: Illustration of the two-sample Kolmogorov–Smirnov statistic [18]. Red (above arrow) and blue (below arrow) lines each correspond to an empirical distribution function, and the black arrow (maximum difference) is the statistic. Data used for plot is generated from a normal distribution with X being generated values (4 being the ultimate value implies that all values are less than or equal to 4). If e.g. customer age was depicted instead, the x-axis would range from 18 to 123.

No categorical feature of value was detected, but several continuous one; many of them related to savings and investment volume. This was an interesting find, as per definition of the target, these features are supposed to be zero. The investment volume is indeed zero for most clients, but a handful have non-zero values, and a majority of that handful have a positive target. The reason is that the bank has a threshold for when a client is considered as owning "investments", and all non-zero values here are below that threshold (and thus, as far as the bank is concerned, they don't have any investment volume). However, there's an additional binary feature that keeps track of whether or not someone owns any investment products. This feature was also deemed as very significant, which is even more noteworthy as, per construction of the target, it is supposed to be entirely zero.

A similar situation occurred for clients having non-zero savings, as savings can be defined in two main ways; either someone who owns a savings product (for instance a product 5; savings recurring) or someone who has money in their account (without it being specifically listed as e.g. a savings account). The thesis (as it aims at marketing products) uses the former definition and thus there will still be many clients who have money in their accounts. Removing anyone who has any sum of money in their account makes no sense as it will delete almost all the data (a main reason for approaching a bank in the first place is to store money). As with investments, there's a feature that keeps track

of whether or not someone own any savings products, and again, there where non-zero elements here.

As may be recalled from **section 2.3**, the booster has a metric known as feature importance, which ranks features based on number of splits. The single most important feature was **customer age**. This category ranged from age 0 to 123 – as Danske Bank is the largest bank in Denmark, it is natural that all ages are covered. Some age-ranges have explanations; although young children can't be customers themselves, their parents can purchase products, e.g. funds, in their name. Similarly, as the oldest living Danish person at the time of writing this thesis is reportedly almost 110 years old, those with ages above that most likely belong to deceased customer whose accounts haven't been terminated yet for various reasons. However, customers above age 100 are an insignificant portion of the clientele, whereas customers under age 18 (who can't be marketed to, and thus are irrelevant for this thesis) make up a significant portion of the data.

Removing all customer below age 18 significantly improved accuracy. In retrospect, it can be argued that this group should have been removed already when constructing the target, for business and legal reasons. The improvement isn't a surprise; customers below age 18 haven't made purchases themselves, and thus their personal data won't aid in predicting their purchase behavior (if anything, it could provide faulty patterns). Plotting product purchases against age reveals why the model regards customer age as one of the most important features, as there appears to be distinct preferences related to customer age:

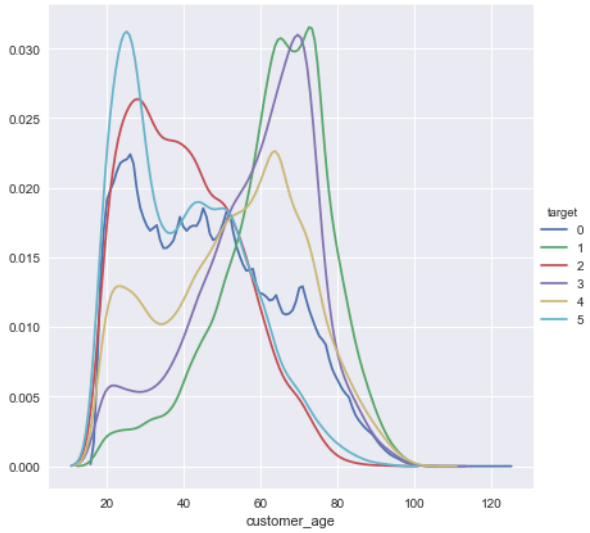

Figure 4.10: Normalized plot of product purchase related to age. Distinct purchase preferences are picked up by the model - hence why "customer age" is the most important feature.

The final attempt at exploiting features was to use hierarchical clustering to reveal useful patterns (and perhaps be able to exclude some more features). The dendrogram seen in **image 4.11** was generated using a linkage known as **Ward's criterion** [20]. Also known as "Ward's minimum variance method", it defines the distance between two clusters A and B as

$$\frac{n_A n_B}{n_A + n_B} ||m_A - m_B||^2 \tag{4.3}$$

where n_i is the number of elements in cluster i and m_i is the center of cluster i. As can be seen, if clusters are equally far apart, Ward's method will merge the smaller ones.

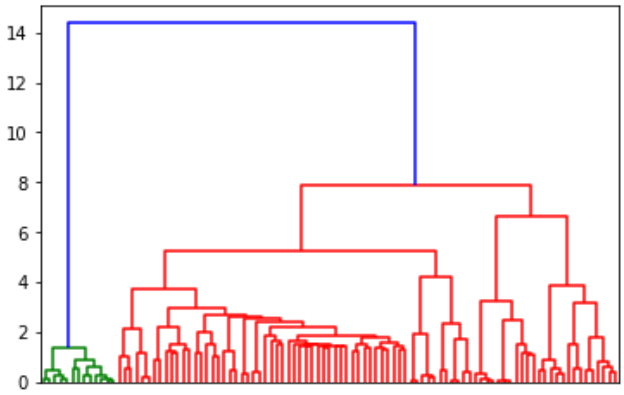

Figure 4.11: Data presented as a dendrogram of hierarchical clusters. Features that contain much NaN are in one cluster (green). It appears that there are two distinct clusters. Looking at the data, it's clear that the green cluster (as seen in the figure) is nothing more than NaN-heavy columns. Within the larger red cluster there are, per definition, two distinct subclusters. Even though one is smaller than the other, it's still too large to be removed without significantly affecting the accuracy. The median distance is 1.1, but cutting the branches at that value didn't improve the model but instead reduced its accuracy.

When predicting on new clients, none of them will of course have neither Savings nor Investment products, and hence these two features were removed altogether from both test and train data. The real question is whether to keep or remove clients that previously own a product in any of the categories. It might seem counter-intuitive to keep them, as none of the customer we'll be predicting on will own a product in any of the categories. However, the booster might pick up valuable patterns (without actually knowing who owns a product, as the features are removed!). Trivially, these clients will be removed from the test data, as clients who previously owned products aren't of interest to predict (per definition of the project). The testing method was **five fold cross validation with**

stratified shuffle split. "Cross validation" means testing the model on data previously not seen by it, and "five fold" implies that five different train/test splits were tried (from the same data). "Stratified shuffling" is a method that preserves the distribution of the target in every split.

A **ROC-AUC** calculation was also done [19]. The **Receiver Operating Characteristics** (ROC) curve is a commonly used tool for visualizing the performance of classifiers. Let **tp rate** be true positives over total positives and **fp rate** be false positives over total negatives. The ROC curve is simply a 2D graph plotting tp rate on the y-axis and fp rate on the x-axis. Thus, points on the northwestern part of the graph are desired as they indicate a low amount of false positives and a high amount of true positives. The diagonal line, x=y, indicates random guessing. The **Area Under the Curve** (AUC), reduces the curve to a single value between 0 and 1 (as it is a portion of the unit square). However, note that the random guess will generate 0.5 and as such, no actual classifier will have an AUC close to that value (even a value far below it, e.g. 0.2, is satisfactory as this implies that predictions can simply be "turned" to achieve a 0.8 in AUC). Likewise, it is very rare to see values over 0.9 with big real-life data. A machine learner can use the AUC-value to detect anomalies in the model. In fact, one of the first advanced models for this project showcased a stunning 0.92 in AUC-value, which was deemed too high for this type of data. After meticulously checking the code, a few errors were detected, and removing them provided a more "reasonable" AUC (presented below).

Since the data is multiclass and the curve can only be plotted for binary, some manipulation was required: A probability prediction was run, and the probability of all positive targets was summed and stored as a single positive target, and compared against the negative target. The highest probability of the two was selected as the prediction.

As known from before, the classes are highly imbalanced, and after removing customers with savings and investment products from the data there is even more imbalance. During initial tries, the model had severe problems in being able to model any product at all; generating low one digit values as accuracy. However, there was patterns in the probabilities of prediction that indicated non-homogeneity, i.e. the booster might predict one product as more likely than the other (but all products are clearly less probable than no-purchase, and thus customer is predicted as 0). One solution to this is to change the threshold for when the algorithm considers a customer as being a specific target. Of course, this will increase the number of false positives, so it's a trade-off.

Experimentation with the data had shown that keeping savings and investment customers in the training set improves the accuracy, despite leaving them out in the test set. This implies that the behaviour of customers already owning the product is similar enough to first time customers for the booster to be able to exploit the information.

Four different stratified shuffle split cross-validated models were created with varying thresholds (based on a heuristics; the mean percentage of every target). Results clearly indicated that different thresholds generated different accuracy, with lower thresholds resulting in higher accuracy as well as a higher rate of false positives (since the model would classify a lot more client as customers, and thus capturing more "real" ones whilst also letting in a lot of "false" ones). The threshold is a trade-off decided based on business knowledge; a few key aspects to look at would be the profit of a sale versus the cost of a marketing approach aimed at an unwilling customer. One must also consider the channel; phone is more labour expensive (but also has a higher potential pay-off) than e.g. an encouraging email or banner at the customers personal banking website. Further, Danske Bank is a customer oriented bank, aiming at maximizing customer satisfaction as well as profits; an undesired phone call from ones bank may not lead to a canceled account, but could very well reduce the overall satisfaction!

The final model is now ready to be trained. So called **model stacking** will be employed; the idea is to combine multiple models (in this case two models, but it could be multiple ones). Here an XGBoost model is used as primary model, and a standard Decision tree as secondary model.

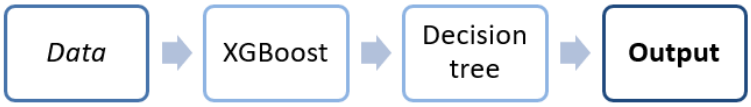

Figure 4.12: Training data is first given to an XGBoost model. The fitted model will predict the probability for every one of the six targets for the test set. The output will be a six-column matrix consisting of continuous probabilities, and is given to the Decision tree model, together with a binary target (assembled from the original multiclass target by letting class 0 remain 0 and classes 1-5 become a new class 1). This tree only has one split to find; at what probability will a customer is purchase (or not). The model will output a vector that contains predictions of whether or not clients are customers. From this tree it can also be extracted a cut-off probability for purchase/no purchase.

Note that the stacked model is essentially a custom Decision tree with two branches - each one a model. Additional models could be added based on how much data that is available after every stack. One could also add models in parallel rather than sequentially, e.g. a categorical booster that trains on only the categorical data. The output of the two boosters could then be averaged. Note that whereas parallel models are trained on the same data (albeit perhaps subsets of it, e.g. one on categorical features and one on the rest), sequential models must be trained on the test set of the previous model.

The ultimate model, this time testing on a new test set (same month as when testing the basic model), had poor accuracy for predicting products. However, the **ROC generated 0.83 for AUC**. Compared to other models in the bank, this is considered a good AUC value. Trying different test data generated a similar value and no apparent faults in the code were detected by neither the author nor by senior data scientist at the bank. Applying the "classic" ROC to the multiclass model requires certain data manipulation as described above.

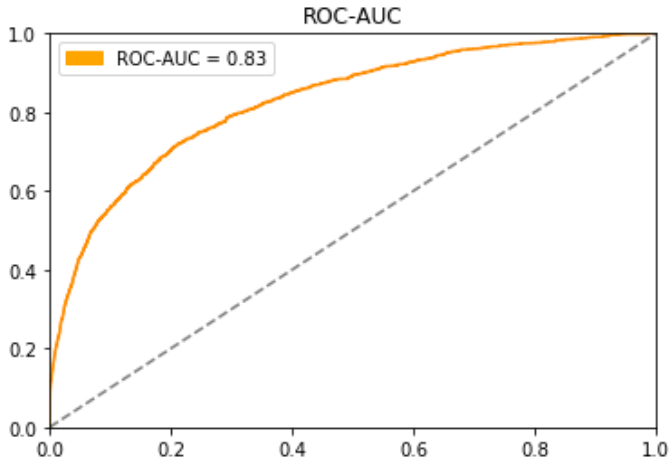

Figure 4.13: ROC for the final model with multiclass merged to binary by classifying all products as "purchase". Dotted gray line denotes ROC AUC of 0.5, i.e. random guess.

There are however other methods of generating a ROC curve and an AUC value [19]. One way of handling n classes is to plot n different ROC graphs. A bigger issue is how to calculate an AUC. The two-class problem allows for a simple area measure, but an n-class introduces problems. One approach is to calculate n separate AUC, and then weight them according to the class prevalence in the data (in our case, it would most likely reduce to the above-mentioned binary case). A different, more sophisticated, approach would be to exploit the fact that the AUC is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. One plots the zero class against the others, and then every positive target against the others. In our case it would be 15 plots. The individual AUC could then be visualized in a matrix, and an averaged ROC could be plotted.

A common way in the bank to measure the efficiency of models is to looks at segments of the prediction; often many models are very confident about a subset of their predictions. The probability predictions for all customers are sorted (descending according to probability to purchase a product) and then binned, and the percentage of actual purchases in every bin is then plotted. This allows

marketeers see how many successful leads they can expect simply by selecting the top n predictions (without considering if the model actually predicted them as a purchase or not). The benchmark is the average probability of purchase (which simply is true events divided by all events), and the gain is termed **lift** for the top n purchases, i.e. percentage of true events for top n predictions divided by percentage of true events for entire data.

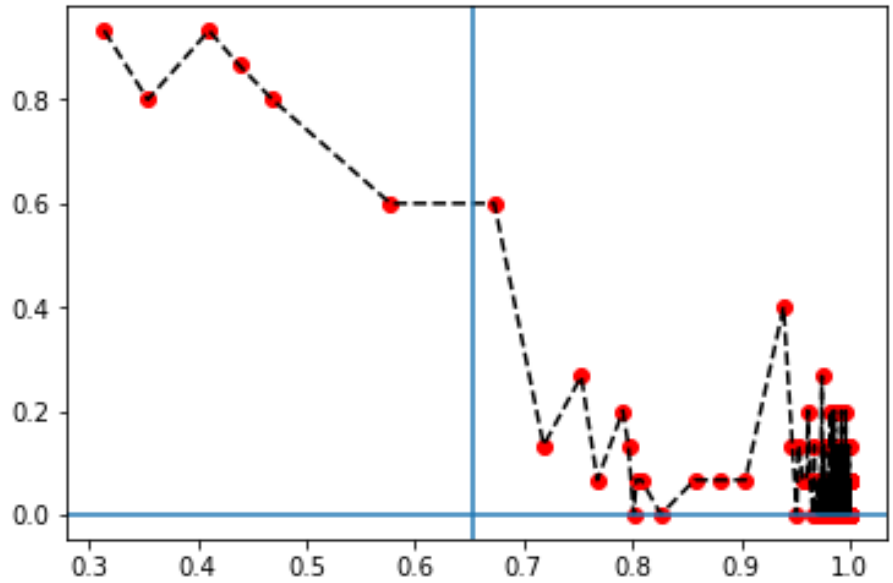

Figure 4.14: Every red dot is a bin with 15 predictions, y-axis depicts percentage of purchase in bins and x-axis is the probability of non-purchase. The vertical blue line is the cut-off for when the model is confident enough to classify an event as a purchase (65%) and the horizontal blue line is the average event rate (0.2 %). As can be seen, the model managed to find the cutoff (containing roughly 100 customers) before a sharp decline in purchase per bin. The **lift** for this model is 8.6 for the top 5% of predictions (which is considered good). Advisers can select an acceptable lift (or accuracy) and then call customers accordingly.

As noted, the lift is calculated as a binary measure; one simply looks at "any purchase". In a practical scenario an adviser would call the customers that have strong leads, and then use the suggested product as a starting point for the conversation.

Looking at the distribution for the predictions above the cut-off:

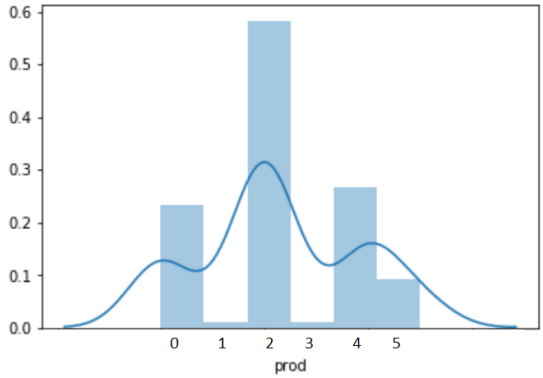

Figure 4.15: A clear majority of customers among the top leads have purchase as intent. When including customers beyond the cut-off (i.e. more than top 100), the amount of non-purchases increase a lot faster than purchases (due to far lower accuracy). Product 2, June, is the most commonly predicted.

Compared to the overall product distribution of the test set (no purchase is obviously most dominant among the entire test set, but has been removed):

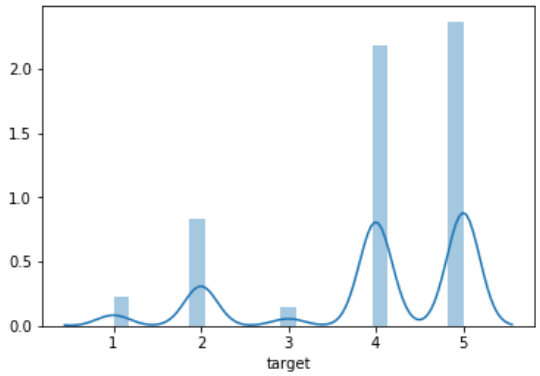

Figure 4.16: Similar to the top leads, product (denoted as *target*) 1 and 3 are badly represented due to poor predictions. For the other products, there's a slight skew in distribution.

Chapter 5

Discussion

The good performance of the ROC indicates that the model can sort products and non-products well, but the low accuracy (without threshold manipulation) is far from satisfactory. In fact, in order to get a satisfactory accuracy, one would have to lower the threshold until the false positives became dominant (and essentially "killed" the model). To keep the model intact, and increase accuracy, the only other option is to include clients who own either investment or savings products; and thus only be able to predict one of the categories. In the end, the question of whether or not it is possible to model on product level turned out to be similar to the question of whether or not it is possible to model on category level - in some cases 'yes', and in other cases 'no'; it depends on the target. More importantly is thus the insights about ones data, to be able to in advance predict whether or not product modelling is possible.

This particular target, with its most narrow definition, could not be modelled. As such, the specific answer to the question posed in the INTRODUCTION is **no** (*as predicted by Betteridge's law of headlines*).

Still, there are clear indications that modelling on product level rather than category level provides additional information about customer behaviour (e.g. with a slightly redefined target for this project). The complexity and approach differ slightly compared to a binary model, but not so much that a professional machine learner can't adapt to it. The additional time spent will pay off in increased understanding of customer behaviour. But even though there are clear advantages with modelling on product level rather than on category level, there's still room for significant improvements of the model. Adjusting marketing strategy requires solid results from a business point of view: the ratio of accurately predicting specific products needs to increase; as a first step above random guess, and on the long run above 50%.

Perhaps the leads aren't optimally aimed at advisers, but rather better used for online marketing (e.g. banners on the customers personal banking site). And

perhaps the most gain is to be made by looking at a few (perhaps only one) specific products, rather than all five (depending on what type of gain the bank is trying to maximize).

Not every product can be modelled alongside others as a "product level" object. Some products are, due to business reasons, bundles of other products. These are, as far as modelling concerned, not on equal term as "true" individual products, and will model worse as they contain noise. With that being said, they can still be modelled, but should not be bundled with actual product level products. As they aren't actually categories, they shouldnt be modelled with categories either, and are most likely best modelled alone.

There are no practical benefits of optimizing with grid search. Practical settings (and not only when big data is being employed) often have too large parameter spaces to make an efficient grid search worthwhile. Given the time often available, only very small sub-spaces can be explored. Rather, using randomized search will circumvent this issue and let the user decide how much time they want to explore a sub-space of user-decided size. Of course, the best optimizer is the Bayesian one, but far more difficult to employ on a multiclass target compared to "out-of-the-package" randomized search ones.

Feature engineering is in many cases "hit or miss", but nontheless a crucial step in any modelling process. Often, as with this project, sheer "feature adjustment" (e.g. capping age) will show significant improvements that perhaps even many optimizes can't achieve. There is a multitude of methods to choose from, and the optimal approach depends on the project.

In the end, XGBoost proved to be the only booster fit for this project. Microsoft's Light GBM was faster, but less accurate, and when employing XGBoost's histogram option the time difference became narrow enough for Light GBM to be discarded from the thesis. With that being said, Light GBM's speed makes it a worthwhile option when it comes to experimenting, as the difference in performance and usage isn't too extreme. Yet, it is significant enough for it to be worth the extra time when deploying an actual business project. Given the multitude of boosters (and other machine learning algorithms) that are freely available, stacking models are recommended as they often increase accuracy.

Chapter 6

Future work

This thesis scraped the surface of a new approach to modelling: product level rather than category level. As such, there are multiple different ways of furthering research in the field.

As far as this specific project is concerned, it would benefit from a pilot (i.e. giving leads to advisers for a test on real customers). The pilot could help determine how many of the accurately predicted product purchases that are "trivial", i.e. that an adviser would confidently pitch without having to do any research. Before a pilot, one could also try to calculate the optimal thresholds. Further, for a true comparison, there should also be an advanced binary model.

This thesis only looked at five products. There are of course no such limits as far as the algorithms are concerned; additional products could be added. One could perhaps look at if there's a relationship between number of products being investigated and prediction accuracy.

Regarding unrelated follow-up projects in the same field (product level predictions) there are, as stated above, several different approaches. One idea could be to not look at first time buyers of a category, but instead look at first time buyers of a specific product, that already have purchased something different from the same category (e.g. predict first purchase of fund Y given previous purchase of fund X).

Of course the model, as it is, could be further polished and improved upon. By tweaking the threshold of the decision tree appropriately, one could get a new dataset where the event rate is increased four times. This data set is large enough to be modelled, and could in turn be stacked an additional time, for further event rate increase. The trade-off would be that the extra time spent has to result in enough additional "certain" leads apart from the 100 generated by the model presented in the thesis.

Bibliography

[1] Arthur L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development, vol 3, p. 210-229, 1959.

[2] Danske Bank. Retrieved from www.danskebank.com

[3] Michael Kearns. *Thoughts on Hypothesis Boosting*. Machine Learning class project, 1988.

[4] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd edition, Springer, 2009.

[5] Robert E. Schapire. *The Strength of Weak Learnability*. Machine Learning, 5, 197-227, Kluwer Academic Publishers, 1990.

[6] Robert E. Schapire. *Explaining AdaBoost*. Empirical Inference, p. 37–52, Springer, 2013.

[7] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 785–794, ACM, 2016.

[8] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. Advances in Neural Information Processing Systems, p. 3149–3157, 2017.

[9] James Bergstra and Yoshua Bengio. *Random Search for Hyper-Parameter Optimization*. Journal of Machine Learning Research, 13, p. 281-305, Microtome Publishing, 2012.

[10] Lev B. Klebanov, Svetlozar T. Rachev and Frank J. Fabozzi. *Robust and Non-Robust Models in Statistics*. Nova Science Publishers, 2009.

[11] Jasper Snoek, Hugo Larochelle and Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. Advances in Neural Information Processing Systems, 25, p. 2960–2968, 2012.

[12] Raul Garreta, Guillermo Moncecchi, Trent Hauck and Gavin Hackeling. *Scikit-learn: Machine Learning Simplified*. Packt Publishing, 2017.

[13] Larry Hardesty. *Automating big-data analysis*. MIT News. October 16, 2015. Retrieved from news.mit.edu/2015/automating-big-data-analysis-1016.

[14] Andrew Ng. *Machine Learning and AI via Brain simulations*. Presentation given at Stanford University, 2013. Retrieved from ai.stanford.edu/∼ang/slides/DeepLearning-Mar2013.pptx.

[15] Anna V. Dorogush, Vasily Ershov and Andrey Gulin. *CatBoost: gradient boosting with categorical features support*. NIPS 2017 ML Systems Workshop.

[16] Anil K. Bhattacharyya. *On a measure of divergence between two statistical populations defined by their probability distributions*. Bulletin of the Calcutta Mathematical Society, 35, p. 99–109, 1943.

[17] John W. Pratt and Jean D. Gibbons. *Concepts of Nonparametric Theory*. Springer, 1981.

[18] Bscan (username), CC0. Retrieved from https://commons.wikimedia.org/w/index.php?curid=25223119

[19] Tom Fawcett. *An introduction to ROC analysis*. Pattern Recognition Letters, 27, p. 861-874, 2006.

[20] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. *Introduction to Information Retrieval*. Online edition, Cambridge University Press, 2008.

Master's Theses in Mathematical Sciences 2018:E76
ISSN 1404-6342

LUTFNA-3047-2018

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/