TU Delft

Bachelorproject

TI3806

Predicting customer loyalty

Final Report

Authors:
Jason Raats
Lars van der Zwan

Bachelor Coordinator:
Martha Larson

Supervisor:
Hayley Hung

Client:
Chris Broeren
Jornt de Nekker
Richard Verburg

December 2, 2015

Preface

This report is created by Jason Raats and Lars van der Zwan as part of the Bachelor Computer Science program at Delft University of Technology. In this thesis we will discuss the process of creating useful machine learning models to predict NPS scores at ING. The results of those models are then visualized in a dashboard.

Spending three months at ING in their main office in Amsterdam was an enjoyable experience. We would like to thank ING for giving us the opportunity to successfully complete our project and in particular we want to thank Jornt de Nekker for setting up the project and Chris Broeren for being our full-time mentor. Without Chris, we could not have achieved the same result.

Finally, we want to thank Hayley Hung, assistant professor at TU Delft and part of the Pattern Recognition & Bioinformatics Group for her feedback and the time she has spent on our project.

Jason Raats
Lars van der Zwan

Contents

Summary

The Investment Department of ING asked us to build a model that predicts customer loyalty using the Net Promoter Score (NPS) system and a dashboard that would show the results of the model. With this model and dashboard ING may be able to get an insight into which groups of customers are at risk to complain.

In the first weeks we researched various techniques in machine learning and got a grasp on the NPS system and we decided to make our problem a classification problem. Instead of predicting the score of a customer, we decided to predict whether a customer is a promoter, a passive or a detractor. In this project we have built multiple kinds of models using different techniques of machine learning, compared them and chose the best performing ones. For choosing the best models we used Receiver Operating Characteristic (ROC) curves and precision/recall curves.

We made two dashboards: one by using the Microsoft Power BI tool and one with the JavaScript D3 library. The Power BI dashboard will be implemented in the business, the D3 dashboard, while locally stored and more accessible, unfortunately is too hard to maintain.

1 Introduction

ING wants to give its customers the best service possible. But to know whether you are improving, you need to measure customer satisfaction and loyalty in some way. One way would be to use the Net Promoter Score (NPS).

This score is generated by asking a customer how likely he/she would be to recommend the company to others. The scale from which customers can choose is zero to ten. When a customer responds with a nine or ten, the customer is called a promoter; when he/she responds with seven or eight a passive; and otherwise a detractor. The NPS is the percentage of promoters minus the percentage of detractors. ING has data about Net Promoter Scores of customers. However, this data only covers a small percentage of all the customers at ING, so it is not a good representation of the overall NPS.

In this project we would like to predict the NPS of all the other customers, so ING can get a better overall view of customer loyalty. To be able to do this, we need information about the customers that gave feedback. In the end we want to see the predicted NPS per customer segment or product so that ING can investigate on which points to improve customer loyalty.

In the next chapter we define the problem and we describe what the objective of this project is, what the input and output will be, and the requirements of this project. Chapter 3 includes some background information about NPS; in chapter 4 we will describe how to obtain the data to train and test the models; and we will discuss different types of machine learning models in chapter 5. The next chapter describes how to measure the performance of these models. Chapter 7 elaborates on building the dashboard and how we implemented it in the business. Finally in the last chapters we will provide our conclusions and recommendations.

DISCLAIMER: For security and privacy reasons, some parts have been redacted. This might influence the readability of this document.

2 Problem definition

As stated in the introduction, only a small percentage of customers participate in satisfaction questionnaires. It is possible to calculate NPS from these responses, but you would not know whether the group is representative of the whole user base. Perhaps only a specific kind of people are willing to fill in these forms and this can make the results biased.

Another problem is that is is unclear what the exact NPS is for each customer segment or product. If the ING would have this kind of information, it would be able to monitor customer loyalty and experiment to improve it. The ultimate goal is to have the best NPS score possible, but reaching this goal requires better (statistical) models to gain new insights.

2.1 Project

In this section we will describe the organisation of the project. We will describe the objectives of the project, what we will deliver as end product, the input and output of our model and the requirements for our project.

2.1.1 Objective

The scope of this project is only to predict the NPS; analyzing where improvements can be made is a task for other teams. So our objective is to deliver a model that predicts the average NPS score of groups of customers. When the NPS is predicted, a (management) report should demonstrate the development of the NPS; besides predictions, this will also show past changes on which the predictions are based. Users of this report can slice and dice through the data and understand how the business is doing, based on the numbers in the report. They will also be able to see the predicted NPS score filtered by investment products, segments or filtered by client characteristics (in that order).

2.1.2 Description

In this project, we aim to produce a model that can predict how loyal customers are to ING's services. This model will apply machine learning algorithms on the extracted data in order to produce a report about the NPS of different dimensions in the business. We also aim to discover which groups of customers score higher than others, so the business is able to increase their customer satisfaction. Our focus will be strictly on the Investment department. When the model has predicted the NPS for groups of customers, the results will be shown in a (management) report.

2.1.3 Input and Output

The input of our model will be two sets of customers as documented in the database of ING. The first set contains customers with a known NPS, and the second set contains groups of similar customers without a known NPS. The output of our model will be a list of expected NPS values per group of similar customers.

2.1.4 Requirements

- Hardware
  - 64 Bit computers
  - Minimum of 8 GB RAM
  - Minimum of 1.90 GHz CPU
- Development tools
  - RStudio (R development tool for data modelling)

– IPython (Python development tool for text mining)
– Aginity (Netezza database access tool)
– Cygwin (Linux environment for Windows to run Vowpal Wabbit)

3 NPS

3.1 Theory

In 2003 the American business strategist Frederick Reichheld introduced NPS in his paper *"The one number you need to grow"* [Reichheld, 2003]. He stated that customer satisfaction is hard to measure, but that satisfaction leads to loyalty. But what does 'loyalty' mean? Loyalty is not simply the rate of visiting a company. A customer that visits the only supermarket in his town every week does not have to be loyal to that company. He just visits it because it is the only supermarket. A customer that always buys his car at the same company may be very loyal, but does not visit the company often, because he does not need a new car very often. Reichheld therefore stated that customer loyalty is the willingness of a customer to recommend the company to his friends, family and colleagues. With this in mind loyalty could be measured by one simple question, instead of long and complex customer surveys:

> On a scale from zero to ten, how likely is it that you would recommend [company X] to a friend or colleague?

The results of this question leads to three major groups that Reichheld calls *"detractors"* for scores zero to six, *"passively satisfied"* for scores seven and eight and *"promoters"* for scores nine and ten. Detractors have a negative influence on the company, because they will spread negative feedback about the company. Promoters have a positive effect on the company, because they will recommend the company to others. The Net Promoter Score is therefore determined by:

$$NPS = \text{percentage of promoters} - \text{percentage of detractors}$$

A positive NPS thus means that there are more promoters than detractors and a negative NPS means that there are more detractors than promoters. Reichheld stated that the NPS correlates with the growth of the company, so when the NPS increases, it means that the company will grow.

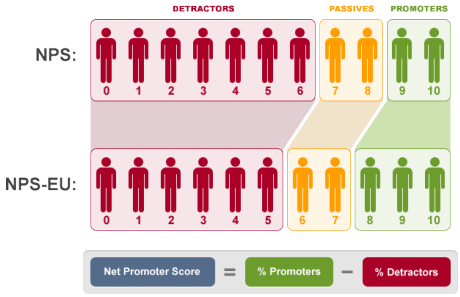
3.2 Opponents

Although NPS is used by companies all around the world, not everyone is convinced of its usefulness. For instance, Grisaffe admitted that NPS could be an indication of the state of a company, but stated that NPS has its weaknesses. Grisaffe questions Reichheld's definition of loyalty and criticises the fact that NPS is said to be 'the one' number that companies should increase. According to Grisaffe it is just "a number among others" [Grisaffe, 2007].

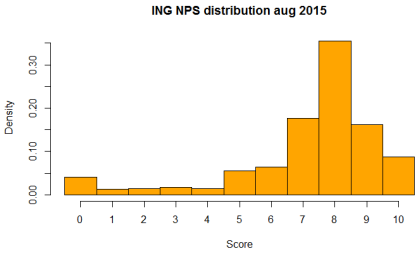
One of the main problems of NPS is placing the customers in the three groups. This is especially the case when NPS is used for comparing two companies in different countries or continents. Americans for example are known for giving more 'extreme' scores, whereas Europeans (and especially the Dutch) are more reserved in their scores. In Europe, an eight is seen as a very good score, so Europeans do not tend to give nines and tens. Europeans may give a six or a seven, whereas an American would give an eight or a nine and otherwise an American may give a zero where a European would give a three or a four. This difference in scoring causes American companies to have fewer passives and more promoters or detractors and therefore score more extreme NPS. ING Direct USA for example scored a positive NPS of 60% in July 2008 [CustomerGauge News, 2008], while the average NPS for the nine biggest banks in Australia in 2006 was around -30% [Ritson, 2007].

Due to these differences some propose a European NPS system, where one is marked as a detractor when giving a score of zero to five, passive for scores six and seven and a promoter for scores eight to ten [Dobronte, 2012]. This system is visualized in figure 2a.

In August 2015, ING had a NPS of +3% (calculated in the normal NPS system). The distribution of the customer scores was normally distributed around the average of eight, as to see in the histogram in figure 2b.



(a) Difference between normal- and European NPS categorization



(b) NPS scores at ING in August 2015

Figure 2

3.3 Alternatives in the market

NPS is not the only metric that is used to measure customer satisfaction and loyalty. There are others like the Customer Satisfaction Score (CSAT) and the Customer Effort Score (CES). These differ from NPS, but not by much; we will discuss these other metrics below.

3.3.1 Customer Satisfaction Score

For the Customer Satisfaction Score, the respondent has answer on a scale of 1-5 to express his/her satisfaction to a certain topic [van Dessel, 2014]. With this information, one is able to show whether a product or service has met or surpassed the expectations of the customer. One of the possible questions to ask customers is:

How would you rate your experience with [X]?

Respondents can choose between Very unsatisfied, Unsatisfied, Neutral, Satisfied or Very satisfied. To calculate the final CSAT, we sum up only the score of customers who have indicated they were satisfied or very satisfied. The higher the score, the better. The formula for the score is:

$$CSAT = \frac{\#\text{ of satisfied customers}}{\#\text{ of satisfaction survey responses}} \times 100$$

CSAT can be very useful to ask customers about specific products or services. The outcome of those questions will be very specific to these products or services, so it is easy to see which products or services to improve. This is however also its weakness: you cannot really tell to what degree the customer likes the company as a whole. If e.g. the customer is happy with the company, but does not like a specific product, the score of that product will be low but the company is doing a good job overall and that will not be measured through CSAT. This is where NPS can make a difference.

Another difference between CSAT and NPS is that CSAT captures the short term- and NPS the long term satisfaction of customers. Since their usefulness is dependent on what you want to measure, some companies use a mixture of NPS and CSAT to measure customer satisfaction.

3.3.2 Customer Effort Score

The Customer Effort Score is a rather new metric [van Dessel, 2014]. It was invented in 2010 and the first version of this measure was very confusing. CES was designed to measure how much effort a customer had to put into a certain interaction with the company. The question of the first version was:

How much effort did you personally have to put forth to handle your request?

The customer has to answer on a scale from 1 to 5, but this scale is inverted from the other metrics, so a 1 meant good and 5 bad. Additionally, the word 'effort' is not easy to translate to different languages. This lead to the second (and latest) version.

The question for the second version changed to the following one:

The organization made it easy for me to handle my issue.

The scale also changed to Strongly disagree, Disagree, Somewhat disagree, Neutral, Somewhat agree, Agree and Strongly agree. This made the metric easier as well as more popular to use.

However, even with these changes, the metric is not very versatile. The measurements only apply to services and they do not pinpoint what the problem was in the first place. These problems cannot be solved by using NPS as a second metric, but studies show that there is a correlation between the two [Qu, 2013]. So when using CES, another metric than NPS must be used to compensate for these shortcomings.

4 Prepare data

Now that we now what NPS is and why it is often used in the industry, we are going to take a look at predicting the NPS for ING. We need to make predictions, because a customer has to fill in a form before we know their NPS and not everyone did this. So while we know for some customers what they have filled in, for the majority of customers we do not. We would like to know the NPS for every customer, or in our case a group of customers, to get a more accurate score. This is where machine learning comes into play.

Machine learning is a subfield of computer science and combines techniques from pattern recognition and artificial intelligence. It is used to create algorithms that learn from and make predictions with data. The algorithms that are created are called models and the data that is used is called a featureset. In this chapter we will describe how we obtained the data; in the next chapter we will describe different algorithms to create these predictions.

4.1 Which data to use?

Figure 3 shows how ING stores the NPS results in a database. A customer fills in a questionnaire and the answers are stored in a central database, to which only a few employees have access. There is a special table in the database that stores all these forms in one place. This is where we started our search for relevant data. The same database also houses other available information about transactions, balances and other financial information. The majority of data that we used was abstracted from this database. We also used other sources like AEX stock information and specific investment information, which is stored in another database at ING.

To prepare the data for the models, we structured the data as a big table in the following way. Each row represents a customer that has filled in the form at least once. Each column represents a feature. A feature is an attribute or aspect of something, in our case of the customer. Features can be binary, categorical or continuous. An example of a feature is whether the customer has a savings account or not (binary feature). All the features combined are called the featureset. A larger featureset is not necessarily better, because results depend on the features that are used. There might be one feature that is very useful to predict results, or perhaps a combination of features is needed. It is however very important to keep the featureset as small as possible to make the models run fast. The bigger the featureset, the longer it will take for the model to complete.

Our approach was to use as many features as possible and after collecting them selecting only the ones that contribute to our predictions. This approach is shown in figure 4. First we obtain data from the database, store this into multiple csv files (because we use so much data that it is not possible to obtain everything at once), combine this data in R, transform some of the features and finally store the data in one large csv file. R is a software environment for statistical computing. It is important to note that the csv at the end of this process has the structure of the table described earlier. The other csv files in earlier steps in the process did not necessarily have this structure but were transformed in R to suit our needs.

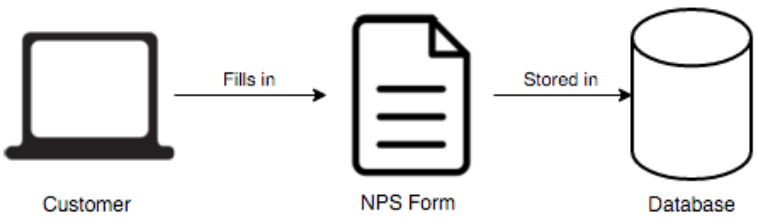


Figure 3: Step 1 of the process

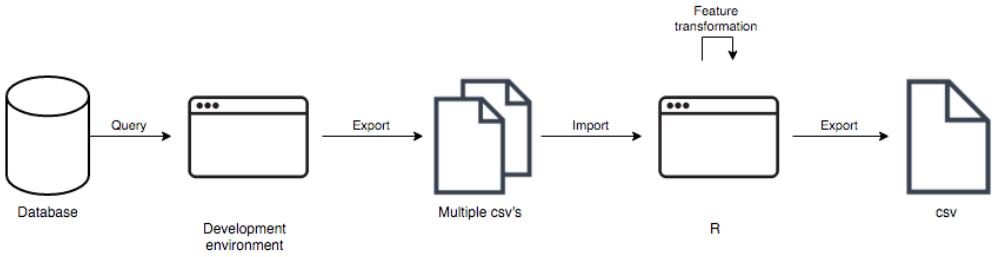


Figure 4: Step 2 of the process

4.2 Feature extraction

The data that is imported into R is called our original featureset. We can make changes to some of the features to make them more useful or to just ignore them so that we are left with a featureset that is most efficient as possible. This process is defined as feature extraction. Feature extraction is the process of transforming or selecting features to improve the quality of the original featureset [Guyon and Elisseeff, 2006]. Improving the quality of the original featureset can be done in different ways. For example, it is possible to make the set more efficient or decrease the required data storage. The ultimate goal is to create the optimal featureset that can build accurate and efficient models. To reach this goal, there are two aspects of feature extraction that can help: feature construction and feature selection. We will describe these aspects in the following sections.

4.2.1 Feature construction

Feature construction is about preprocessing the data for feature selection or for direct modelling. This preprocessing is almost always necessary, because it is usually not possible to use the original featureset as input for the models. The structure of the data that is gathered (from a database or sensor) needs to be altered for the models to reach their full potential. Transforming the data is also useful for getting a better understanding of the data that you are dealing with.

We used feature construction in a number of ways. To extract data from the database, we used some tricks to alter the data already in the query. This saves time in future steps in the process and sometimes it also makes sure that the csv files do not take up a lot of data storage, which makes the importing and other feature transformations quicker. The rest of the feature transformations were done in R. Examples of feature transformations that we used are converting categorical or continuous features into binary features, creating extra features that describe the historical changes of another feature and combining multiple features into a single feature.

Principal Component Analysis. A very common feature transformation method is the Principal Component Analysis (PCA). This transformation falls under the last example given in the previous section. PCA takes n original features, transforms them into one or more linear combinations and creates m new features from these linear combinations. This process is shown in figure 5. This figure shows a scatterplot of two variables and two arrows that represent two linear combinations. Important to note is that these linear combinations are perpendicular to each other, so the sample variance is maximized. This way the linear combinations are uncorrelated and this helps in the feature selection. We will cover this in the next section.

We have decided not to use PCA in our project, because we think it would not improve our featureset significantly (because the majority of our features are binary [R-Bloggers, 2013]) and we preferred to spend our time constructing more new features from different sources. However, this technique could be helpful to discover interesting relationships between variables and maybe increase the amount of useful features in future work.

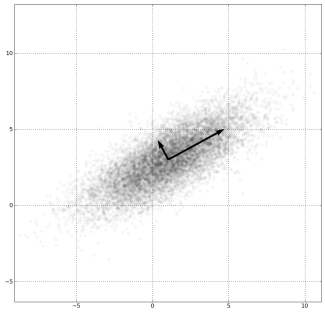


Figure 5: PCA example

Text mining. Some of the information contained in the database is represented in long strings (e.g. answers to questions.) To use this information we have to make it possible for the model to interpret these strings in some way. This is called text mining.

The TM package of the R programming language can easily mine the database. First, all the text that needs to be analyzed is transformed to lower case and punctuation and numbers are removed. After this, a so called Corpus is built: a collection of text documents or pointers to these documents. In our case a text document is one text entry of a client. We decided to use the VCorpus which physically contains the text documents instead of the PCorpus which contains just pointers to the text documents. All our text documents are in the same file, so it would not make sense to access this file outside R for every text document. Instead, we simply load the file in R once and are able to convert everything to a VCorpus. With this Corpus, all the words in the text can be brought back to their stem so verb conjugations are considered as the same words and plurals are considered as their singulars.

We want to make binary features of words that are used in the answers. When a customer uses a word, he gets a 1 as value for the feature, otherwise a 0. However, because the data contains thousands of answers, it would be senseless to make a feature of every used word. The featureset would simply become too big and filled with features that only occur for a very small group of customers. Therefore, when the document has been stemmed, a list of frequent terms is composed. These terms will be used as features.

4.2.2 Feature selection

Feature selection is the process of selecting features of the original featureset that contribute most to the end result, the predictions. This is also an important step in the process, because in our case it would not be possible to use thousands of features due to our computational limitations. We need to select only the features that make the predictions more accurate and ignore the noisy features. There are two ways to select features: using filters or wrappers.

A filter is an independent criterion that selects a feature subset without using machine learning algorithms [Motoda and Liu, 2002]. Filters are computationally less expensive than wrappers, because the performance evaluation metrics used come directly from the data. There are dozens of filters that can be used, but it is hard to know beforehand which will give the best result. A couple of popular filters are Correlation based features, Distance between distributions and Decision Trees [Guyon and Elisseeff, 2006]. We will not discuss them further here, because we decided to use wrappers only.

A wrapper is a dependent criterion that selects features based on machine learning algorithms. The reason why we have decided to use wrappers only, is because in our opinion wrappers contribute more to the end result than filters. It should be noted that while it was possible in our case to only use wrappers, a larger featureset would have required the use of filters. Again, there are a couple of wrappers that can be used.

Exhaustive. The exhaustive approach will apply the chosen machine learning algorithm on every possible feature subset. After each subset is evaluated, the one that scores best is selected. How these machine learning algorithms evaluate which featureset is best is discussed in section 5. It is easy to see this method is very computation heavy and that there are probably more efficient approaches to reach the same goal.

Heuristic. There are two heuristic approaches that are popular: Sequential forward search (SFS) and sequential backward search (SBS). SFS will start with an empty set of features, searching for the next best feature and then adding it to the set of features. This greedy approach will also take a long time if the featureset is large, but is an improvement from the exhaustive approach. Greedy is an algorithm that iteratively combines the locally optimal choice at each stage of the algorithm [Roughgarden et al., 2013]. The downside to this approach is that we may end up with a suboptimal featureset, because it could be possible that the best featureset is a combination of features that do not excel on their own. This way the features are not added to the featureset, because there are other features that score better individually. SBS is almost the same approach, but reversed.

Nondeterministic. A nondeterministic approach chooses random features and checks if they perform well. It then picks random features again and checks if this set is better than the previous one. If it does, then it replaces the newest set with the old, otherwise the new one is ignored. With this approach it is not known when the best featureset is checked; you only know whether the new set is better than the saved one.

We have used our own approach in searching for the best featureset. We decided to use an online learner package called Vowpal Wabbit and a booster package called xgboost. These modelling packages both have the ability to determine the relevance of different features. Vowpal wabbit produces a list with the percentages of relevance per feature, whereas xgboost just produces a list of the most relevant ones. With these lists, useless features can be filtered out, so modelling will be easier and thus faster. This is a big advantage when using the slower glmnet package for building models. We will discuss these different packages in section 5.

4.3 Missing data

After we completed our featureset, we noticed there were some problems with missing data. There were a couple of columns (variables) with so-called NAs (not available) in them. This was a problem, because some models cannot work with missing data. There are a few solutions to this problem.

The first one is to simply ignore the variables where NAs occur. This method is very simple to execute, but is not preferred. These variables might be very useful for our model and we would have disregarded them using this method.

Another solution is to fill in a value where NAs occur. This value could be 0, -1000, the mean of the variable or some other arbitrary value. This solution is better than the previous one, but is also not the best way to handle missing data. Filling in one of these values will transform the distribution of the variable and will therefore influence the model in an unknown way. However, this is the way we have dealt with missing data, because we did not want to lose data and we needed another simple solution. In the case of binary variables, we just assumed that when there was no data available we could put a 0 there without losing the quality of the variable. For example: if someone has a product, he/she gets a 1 in the database. If it is unknown whether someone has the product, then we assumed that the answer is no. Logically this is true, but dealing with unknown situations calls for more in-depth research.

The last method we will discuss is seeing the missing data as a predictive problem in itself. It is exactly the same problem as we are trying to solve. We have data from some of the customers, but we want to predict the value of the customers for whom we do not know the data. Determining values for the missing data can be done via density estimation [Ghahramani and Jordan, 1994].

The variable that is missing data is compared with another variable to look at the density between the two variables, after which we can choose values that will fit in the density to keep it the same as the original. This way we will not influence the data. For continuous variables mixture of Gaussians can be used and for discrete variables mixture of Bernoullis. We will not dive deeper into these methods, because we have not used them, but future work could expand on this.

4.4 Conclusion

We have seen that finding the right features is very important in the process of predicting the NPS for groups of customers. We have applied feature extraction in several steps of the process, used text mining to see if answers of customers are useful, came up with a solution for feature selection and we made a decision what to do with missing data. We are aware that there is still a lot of room for improvement, but given the short amount of time, we are confident that we worked as efficiently as possible. In the next section we will discuss different kinds of machine learning algorithms. We will also discuss the abovementioned software packages.

5 Machine learning

After selecting the features it is possible to build the models. We will use machine learning to predict NPS. Machine learning is the field of artificial intelligence that covers the development of algorithms that computers can learn from. There are two ways to continue our process: supervised learning or unsupervised learning. Supervised learning is the process of applying machine learning algorithms on labeled data (for both the input object and output object), whereas unsupervised is just using an input object. In this project we only used supervised learning algorithms, because our input data was labeled.

5.1 Approach

We wanted to start simple and built the complexity over time. After a couple of weeks we created more complex algorithms which were more suited for our project. We ended up with the process described in figure 6. We have used three different kind of models; how we built these are describe in this section.

5.1.1 Linear regression

In machine learning there are two different kind of predictive tasks: predicting a value or predict a category. When you want to predict a value, then we call this regression. If the algorithm has to choose between a finite set of categories it is called classification. We started building a simple linear regression model.

Linear regression will fit a line (in case of one feature as input) or a plane (in case of multiple features as input) through the data points according to some error function to predict the desired value. An error function is also called a loss function; we will further discuss this in the paragraph below. When there is only one feature as input, we call the model a Simple Linear Regression (SLR) and otherwise call it Multiple Linear Regression (MLR). The general formula of linear regression is as follows:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon$$

where Y is the predicted value, β_i is the unknown regression coefficient of feature i, X_i is the value of feature i and ϵ is the error that occurs. In the case of SLR, the simple $Y = \beta_0 + \beta_1 X_1$ problem needs to be solved. To find the best values for all β, a so-called loss function needs to be solved.

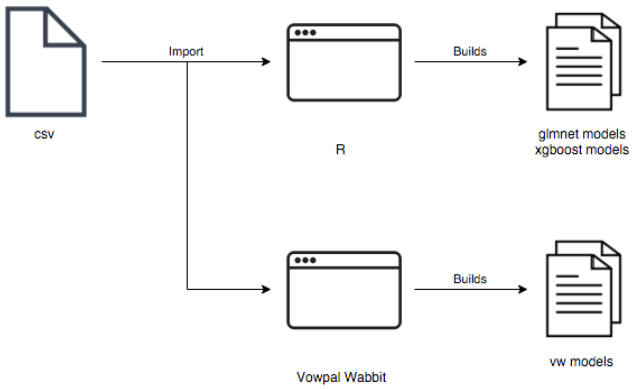
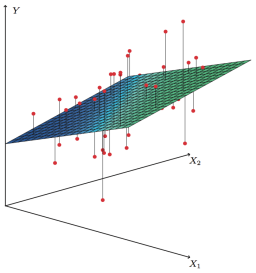
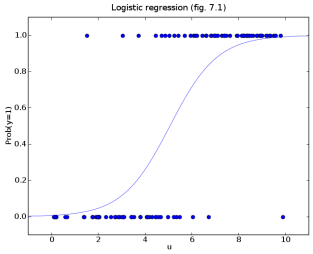


Figure 6: Step 3 of the process



(a) Linear Regression example



(b) Logistic Regression example

Figure 7

Loss function. A loss function is a function that will express the cost of values of one or more variables as a real number. We will demonstrate how a loss function works with the help of figure 7a where an example of a MLR model is shown. In this case there are two variables shown (x_1 and x_2) and y is the value we want to predict. As shown, the plane cannot go through all the data points. So which plane is going to be the best plane for predicting y? To find out, we need to be able to compare two planes and show which one is better.

To do this we are going to give a penalty if the plane is far away from most of the data points and reward it if it is close to the data points. A simple example of this method is the least squares loss function. Least squares will calculate the euclidean distance from the plane to each point (as shown in the figure) and square this distance. The formula looks like this:

$$RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i-1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2$$

where RSS is the residual sum of squares, y_i is the real value, $\hat{y}_i$ is the predicted value, β_i is the estimated regression coefficient and x_i is the value of feature i. In order to get the best fitted plane, we need to minimize RSS so that the euclidean distance from the data points to the plane is minimal [James et al., 2014].

Least squares is not the only loss function you can apply. One of the main disadvantages of least squares is that outliers dominate the result. Other loss functions have their own advantages, it is the job of the data scientist to choose the right loss function for the right problem.

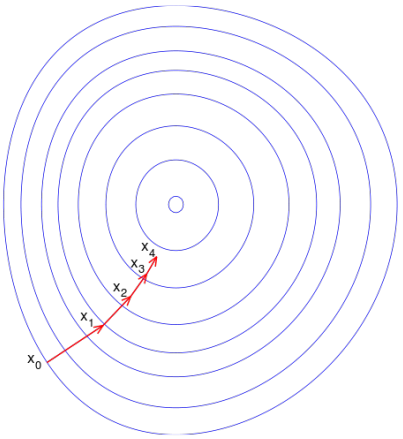
The first models we built were not very good and took a long time to complete. We came to the conclusion that fitting a straight plane through our data points is not an effective way of making predictions. In figure 7a it is shown how the algorithm tries to fit a plane between the data points. If there is no linear correlation between the points, then the errors will get very large. This was the case in our data set.

5.1.2 Logistic classification

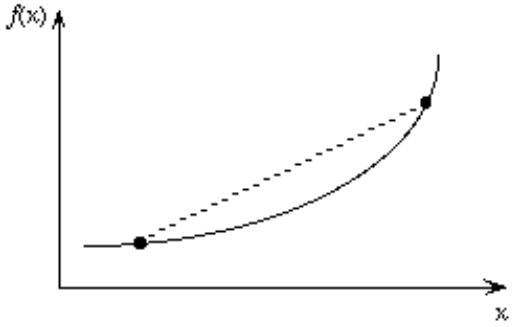
At this point we decided to convert our regression problem into two classification problems. To calculate the NPS we only need two measures: how many promoters and detractors are there? We will build two models: one predicts if someone is a promoter (1 if true, otherwise 0) and the other will predict if someone is a detractor. If someone is not a promoter, it does not automatically mean that he/she is a detractor, because they could also be a passive. Recall from section 3 that passives do not influence the NPS.

We found a special case of linear regression that is designed for classification: logistic classification. The base of this model is the same as linear regression, but we transform each term into its log. In figure 7b it is shown why logistic classification is an useful model to apply in our project.

It is easy to see that the line shown is closer to values 0 and 1 than when we would use a straight line. Converting to this new method takes only one transformation in the formula:



(a) Gradient descent



(b) Convex function

Figure 8

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

We need to apply the log function; this is where the name logistic comes from. This formula will construct the characteristic line shown in figure 7b. This is an example with just one feature, but it is possible to apply this to multiple features as stated in the formula. This method increased our prediction accuracy, so we decided to use this algorithm.

We found a package in R that could easily help us build the logistic models. This package is called *glmnet*, where *glm* stands for generalized linear model and *net* refers to elastic-net regularization[Friedman et al., 2010]. Elastic-net regularization is a combination of Lasso and Ridge regularization, which we will cover section 5.3. *glmnet* is thus in fact a linear model generator, but you can also create logistic models with it. This is because linear and logistic models are inhererently not that different.

There was however still a problem. Even though the predictions were getting better, it still took quite some time to calculate them. When we gave a featureset with 4000 features as input, the model took more than an hour to complete. Since we had limited time during this project, we wanted to decrease the run time but also wanted to keep the quality of the predictions. This is where the online learner Vowpal Wabbit comes into play.

5.1.3 Online learner

The algorithms described earlier are considered to be offline learners. This means that all the data points have to be in memory to make calculations. With an online learner it is possible to load in one data point at a time and update the model accordingly. This prevents many memory issues and it is also quicker to compute correlations between features of just one data point than of thousands at a time.

We found an online learning package called Vowpal Wabbit. It is built with speed in mind and we noticed that from the beginning. By the end of the project we could create 1700 models in just one day with the same 4000 features we putted into *glmnet*. There are a few reasons why it is this much faster. Because VW can update the model with just one "submodel" at a time, it can run on multiple threads. This means it can build multiple models at the same time in parallel. It also uses a trick that is called online gradient descent. We will first explain what offline gradient descent is.

Gradient descent. The concept of gradient descent is shown in figure 8a. The red arrows are the negative gradients of each point x_i and the blue lines are contour lines of the loss function. In this figure it is shown that gradient descent estimates a minimum (x_0) of the chosen loss function and then calculates in which direction it should go to come closer to that minimum. If the loss function is convex, then gradient descent will always find the global minimum; otherwise a local minimum may be found (this depends on the first estimated point). A convex function is a continuous function where every pair of points on the line lies below or above the function. This situation is shown in figure 8b. For more information about convexity, read [Boyd and Vandenberghe, 2004].

To determine in which direction to move next, it is necessary to know in which direction the slope is minimal. This could be achieved by minimizing the derivative of the loss function. Note that in figure 8a the red arrows are orthogonal to the blue lines, because that is the minimum distance between the contour lines. The formula to simulate this process looks as follows:

$$w = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)$$

where w is to estimated, η is the step size (learning rate) and Q_i is the i-th observation in the training set. We will discuss train and test sets in section 5.2, but the main idea is to split the data with labels in a train and test set to be able to evaluate the created model. Notice we could fill in the formula of RSS for Q to apply gradient descent for least squares. It is necessary to pick a value for η, because gradient descent needs to know when to stop. When the improvement of the next step is lower than ηQ_i, the algorithm will stop. Otherwise the algorithm might take a very long time to stop, or not stop at all, and after many iterations w will converge eventually. Choosing a good value for η will keep the algorithm fast, accurate and will ensure that the algorithm stops.

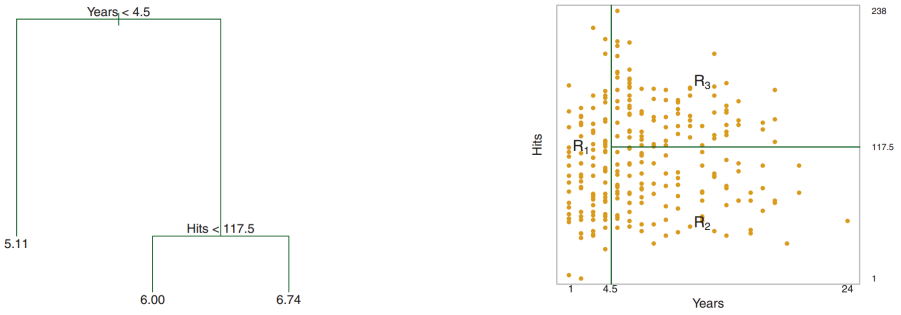
Now we now how a traditional gradient descent works, we will take a look at the online version which Vowpal uses. This is a slightly different approach, because Vowpal only loads one data point at a time. Vowpal does not have to estimate a local minimum, because it will start at the first data point in the training set. Then for every next data point it will update w the following way:

$$w = w - \eta \nabla Q_i(w)$$

It is possible to run over the same data point multiple times to improve w. In the worst case the first data point is a bad estimate of the local minimum; then online gradient descent has to travel a long distance to reach that minimum. But because it improves slowly (since it loads in one data point at a time), it is possible that you need to load data points multiple times. The best way to read data points more often is to randomize the order in which the data is loaded, so after every data point is processed the order is randomized to prevent cycles. It is also possible to use a dynamic learning rate η. Vowpal has the option to use more types of gradient descent, but these types are out of the scope of this project. To read more about these and other functions of Vowpal Wabbit, read [Langford, 2015].

We have experimented with different Vowpal settings and we noticed that after 70 iterations over all data points the improvements converged. Passing all the data 70 times sounds inefficient, but Vowpal is still very fast because of the reasons stated earlier. This was not the only reason we used Vowpal, because it also has the option to show the relevancy percentage of each feature. This percentage indicates how much the feature contributes to the model. If a feature does not improve the model, then it will display a percentage of 0. If the feature is positively correlated, a positive percentage is shown (with a maximum of 100%), otherwise a negative percentage is shown (with minimum of -100%). We will explain how to get features to 0% in section 5.3. This way we could make a featureset where only relevant features are included.

This function is helpful to make the featureset for *glmnet* smaller. We deleted all features where the relevancy percentage was between 20% and -20%, because those features do not contribute much to the model. Normally you only delete features with a percentage of 0%, but we would



(a) Decision Tree

(b) Three region partition

Figure 9

still have too many features left if this criterion were to be used. This improved the speed of *glmnet* dramatically, but the quality of the predictions also decreased. Luckily this decrease was not significant, so we did not have to ignore *glmnet*: it was still useful to run. Results can be found in section 6.

5.1.4 Boosting

After improving the speed of *glmnet* and using a new model, we wanted to add one more model. In other projects at ING they use the package *xgboost* [Chen et al., 2015]. This package helps to build a booster in R. Boosting is based on the idea of building a model multiple times, combining the results and therefore reducing bias and variance. This is one of the main reasons boosting is one of the most state-of-the-art methods for supervised learning [Hastie et al., 2009]. At ING, they use decision trees to boost the model.

Decision Trees. Decision Trees are easy to understand, because of the way they can be visualized. Figure 9a shows how to construct a tree. This model was built to predict salary of baseball players. There are two variables used in this example: years and hits. In the first internal node (also called the root), the first data split is located. If a player is active for less than 4.5 years, then we go to the left; otherwise we go to the right. In case of the first event we are done after this step, because there are no more internal nodes. In the other case we need to take one more step: if the player hit less than 117.5 times, we go to the left; otherwise we go to the right. The values displayed at the leaves represent the log of the predicted salary.

The model is easy to follow, but it clearly has its weaknesses. Trees on their own are not very accurate because of their structure. In figure 9b the model is plotted over the data points and the quality of the predictions is not very high. We can further divide the data by adding more internal nodes, but if we do this too many times the model will overfit. Overfitting means that a model will predict the values of the training set very well, but when the model wants to predict the values of test data it performs badly. This is because the model is trained too tightly on the training data and has lost its flexibility. We will discuss how to solve this problem in section 5.3.

If Decision Trees are not very accurate for our data, why are we using them as a model? As stated earlier, boosting can help to make every model less biased and reduce the variance. Boosting will stepwise improve one tree through creating new trees and combine the results. The algorithm will begin by setting $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set. $\hat{f}(x)$ is the model we will optimize and r_i is the residual for element i, which is the predictor y_i in the beginning of the process. Then the model will loop B times and at each iteration it does the following: it will take a bootstrap sample from the training data (discussed in section 5.2.2), fits a tree out of this data with D leafs and calculates $\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^b(x)$ and $r_i = r_i - \lambda \hat{f}^b(x_i)$. B, D and λ are chosen by the user, where B is the number of iterations (number of trees to build), D is the number of leaves

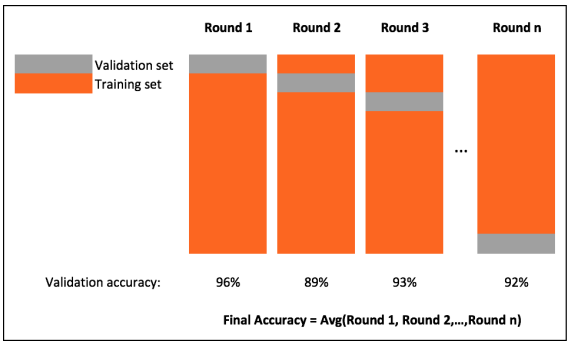


Figure 10: Cross validation

the tree should contain and λ is a penalty to prevent overfitting. To learn more about overfitting, see section 5.3. The output of this model can be summarized with the following formula:

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

There are a lot of parameters that the user has to give as input. To run lots of models with different parameters we made a script that automated this process and fills in multiple combinations of parameters. Then we choose the best model on some metric (described in section 6.1 and the features that are used in the tree are passed on as a featureset for the *glmnet* model, just as we did with Vowpal.

To get the best results with boosting, λ is usually very small. This means that there is a large penalty involved and that the model is learning very slowly. This is why boosting models need a very big B to become successful: it needs to build many trees to make significant changes in the model. It is however one of the fastest methods in machine learning, so this does not have to be a problem with today's hardware. For example, we made 1400 *xgboost* models with more than 4000 features in just two hours, whereas *glmnet* could only produce one model with the same amount of features in one hour. This is because *xgboost* can run on multiple threads on the computer and building trees is computationally less expensive than to fit a plane between millions of data points.

5.2 Resampling

To be able to evaluate a created model, it is necessary to resample the data with labels. Resampling will refit models to different samples from the data set. If we would use all the data to build our models, we cannot test how well the performance of that model is. This is because when running the created model on the same data that it was trained on, it would give a biased result, because the model has already "seen" the data.

The solution is to split the data randomly into two parts: a training and a test set. The model will be trained on the training set and evaluated on the test set. This will give the most unbiased view of how well the model performs. However, doing this just once will not give the actual performance of the model, because splitting the data randomly again will give a different result. There are a couple of ways to resample the data multiple times.

5.2.1 Cross-validation

Cross-validation is the process of splitting the data into multiple parts and using some parts as training data and the rest as test data. An example is shown in figure 10. There are two ways to do this: exhaustive and non-exhaustive.

Exhaustive. Exhaustive cross-validation will train and test on every possible combination of the original data set. This could be done by choosing p data points as the validation set and the rest as the training set. This method is called Leave-p-out cross validation (LpO CV). This is however very inefficient, because the complexity of this method is $O(2^n)$. This means that a larger input n (the amount of data points in the training set) takes exponentially more time to compute. This is why this method is almost never used.

A special case of LpO CV is when $p = 1$. This is called leave-one-out cross validation (LOOCV). This is also an exhaustive method, but the complexity is $O(n)$, which is much better than LpO CV (where $p > 1$). There are only n ways to use one data point as validation set, so the algorithm has to apply cross-validation n times. So now there is a linear relation with input n instead of exponential. Even though this method is better, for large n this still takes too long to compute, so a non-exhaustive method is preferred.

Non-exhaustive. Instead of specifying how many data points should be used as validation set, it is now necessary to specify how much equally divided partitions of the data set should be made. This is called k-fold cross validation, where k stands for the amount of equally sized subsamples. One partition will be used as test set; the other $k - 1$ partitions as training set. Note that when $k = n$ this is exactly the same as LOOCV and this becomes an exhaustive method.

We used k-fold cross validation in all our models. The default value of k for *glmnet* and *xgboost* is 10, which means 9 partitions will be used for training and 1 as testing. Another often used value for k is 5, which we used. Research has shown that these values result in the best performance [Kohavi, 1995]. This research also shows that the use of stratification is very helpful in finding the best performance as well. Stratification will make sure that the mean of the labels is approximately the same for each partition in case of regression. In the case of classification each fold will have roughly the same ratio of zero and ones.

Even when there is not much labeled data, cross-validation can help to make an accurate model. Running the models multiple times on different subsets of the data will give a more accurate result than training and testing once. This is because all the data is used to train the final model. When the dataset is really small, then there is another solution: bootstrapping.

5.2.2 Bootstrap

Bootstrapping is a form of sampling with replacement. This is useful to do when the training set is small. Bootstrapping works as follows: if n is the amount of datapoints, then we will take a sample of size n with replacement, which means that some data points may be repeated. We will take B bootstraps and for each sample we will run the model. Bootstrap can then estimate the standard error or a confidence interval for each coefficient [Hastie et al., 2009].

So even if the original dataset is small, we can create "new" datasets with this method. This way we can investigate a few attributes of the estimated sampling distribution: bias, variance and confidence interval are a couple of examples. Doing bootstrap in combination with decision trees is an effective way of building a model. In the results section 6 we describe how useful boosting was in comparison with logistic classification and an online learner.

5.3 Regularization

Regularization is used to prevent overfitting of the models. An example of overfitting is shown in figure 11. Here you can see that the error of a model has two values: a training error and a test error. The objective of the model is to find the minimum for the test error, not the training error. At first, the more complex the model, the lower both errors. But after a while, the test error begins to increase. When this happens, it is called overfitting. The model is adding too much noise (uses too many features) and this will have a negative impact on validation. To reduce the chance of overfitting, regularization is used. There are a couple of methods to apply regularization. We will only discuss Lasso and Ridge.

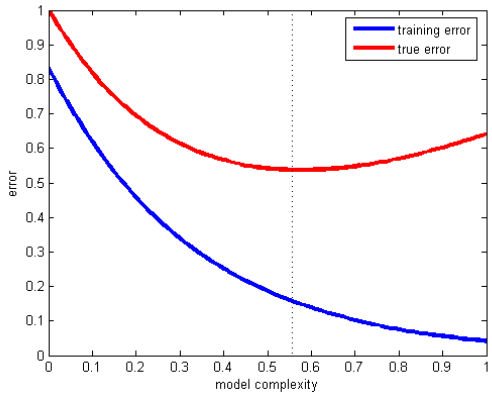


Figure 11: Train and test error

5.3.1 Lasso (L1-norm).

Recall the loss function RSS, where the distance between the predicted value and the real value was squared to penalize models that did not predict accurately. The goal was to minimize this value in order to get the best fitting model for the test data. If the model uses too many features, noise will eventually increase the RSS, as shown in figure 11. To make sure that only useful features are used to predict, we are going to use the Lasso method. The formula will change to:

$$E_x(\beta) = min(RSS + R(\beta))$$
$$E_x(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{m} |\beta_j|$$

Where E_x is the error we want to minimize, RSS still has the same formula, $R(\beta)$ is the penalty function we will apply, λ_1 is the weight of the Lasso method and $|\beta_j|$ is the L1-norm of coefficient vector β. The idea is to shrink the value of β_j so that its estimate will go to zero. If the coefficient is zero, the feature is not used in the model which will not only lead to a faster model, but also lowers the chance of overfitting. Note that when λ_1 has a value of zero, the normal RSS function has returned and no Lasso is applied. The higher λ_1, the larger the influence on coefficient β_j. It is normal to pick $\lambda_1 = 0.001$ to give a soft penalty for noisy features. We have run multiple models with different values for λ_1. The results of these models will be discussed in section 6.

5.3.2 Ridge (L2-norm).

While Lasso will force the coefficients to zero, Ridge will force coefficients to nearly zero. This means that no coefficients will drop to zero and will always contribute to the model in some way. Sometimes this is preferred over Lasso (for example if there are not so many features available). The formula will almost look the same as with Lasso:

$$E_x(\beta) = min(RSS + R(\beta))$$
$$E_x(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_2 \sum_{j=1}^{m} \beta_j^2$$

Here we need to choose another λ_2 value and the coefficient vector β_j is squared. Because β_j is squared, the value will never be zero. This is shown in figure 12. On the left we see the Lasso method, on the right Ridge is shown. The blue areas are the constraint regions $|\beta_1| + |\beta_2| < t$ and $\beta_1^2 + \beta_2^2 < t$ respectively. The red ellipses are the contour line of the RSS function. The estimated

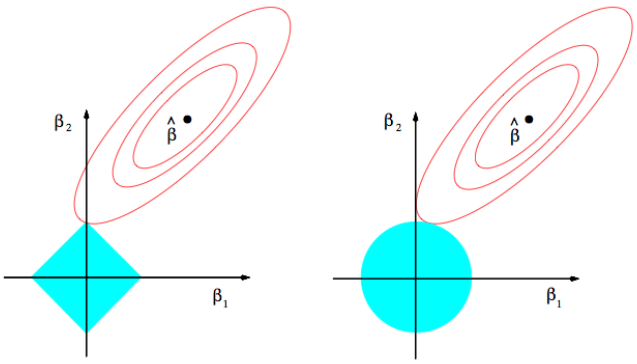


Figure 12: Lasso and Ridge regularization

$\hat{\beta}$ is the same for both plots. The goal is to follow the contour starting from $\hat{\beta}$ and return the values of both β_1 and β_2 when a contour line (red ellips) hits the constraint function (the blue area) for the first time. Because the contour lines are elliptical, the odds are higher that they will hit the blue area of Lasso in one of the corners of the square. At every corner of the square there is one coefficient that is zero. This is not the case in Ridge: when the ellips hits the circle, both coefficients have a positive value greater than zero. This is why Lasso eliminates features completely and Ridge only weakens them.

It is also possible to combine Lasso and Ridge together, and make use of both advantages. This is a simple step from the formulas above:

$$E_x(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{m}|\beta_j| + \lambda_2 \sum_{j=1}^{m}\beta_j^2$$

Notice we can derive both formulas from this one, with setting either $\lambda_1 = 0$ or $\lambda_1 = 0$. If we set both weights to zero, the normal RSS function is obtained. We have made a script that will create multiple models with different values for Lasso and Ridge. The results are described in the next section.

5.4 Conclusion

There are many models, loss functions, regularization techniques and other machine learning elements to choose from. Combining the right algorithms to obtain the best model is very hard and one needs to have lots of experience and knowledge to be able to find this model. What is described in this section does not even scratch the surface of all possible kinds of machine learning tools, but it is a good starting point for people that are new to machine learning.

In the end we used three kinds of models: a logistic classifier, a boosting model and an online learner. Each of those models has their own strengths and weaknesses, but we will see in the next section that the results do not differ much from each other.

6 Results

After building the models, we have to determine which ones perform best; see Figure 13. In this chapter we will discuss the results of our models. However, we have to determine how we measure our models before we can view any results. Therefore we start this chapter with a description of performance metrics.

6.1 Performance metrics

Several metrics are used to measure the performance of our models so we can compare them. In this section we will discuss the metrics we (do not) use and why we (do not) use them. Figure 14 shows the confusing matrix containing the different kind of conditions and the metrics that can be calculated.

6.1.1 Accuracy

The easiest metric for the performance of a model is the accuracy, which is basically just the percentage of well-predicted values. Accuracy may sound very intuitive, but it has some disadvantages. The first disadvantage is that the raw predictions we get are not binary. They give a probability of being 1. Therefore, when using accuracy, a cutoff is needed to determine which predictions should be mapped to 1 and which should be mapped to 0. The second disadvantage is that accuracy does not deal with unbalanced classes. Let us presume that we have a class containing 98 observations represented as value 0 and a class that has just 2 observations represented as value 1. It is easy to build a model that always predicts 0 and this model would have an accuracy of 98%, but in fact the model does not have any value. Therefore, when comparing models, we will not use accuracy as a meaningful metric.

6.1.2 Area under the curve

The area under the curve (AUC) is a metric for measuring binary predictions. With AUC actually the area under the ROC curve (Receiver Operating Characteristic) is meant. ROC analysis was developed during World War II for radars in order to decide whether the signal they received was

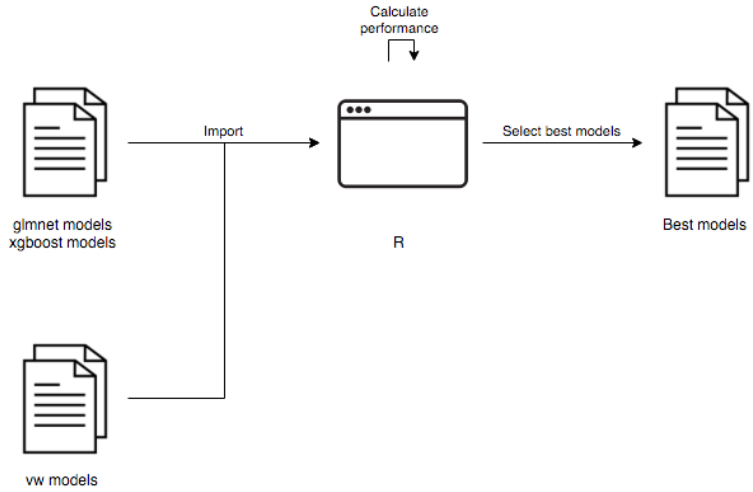


Figure 13: Step 4 of the process

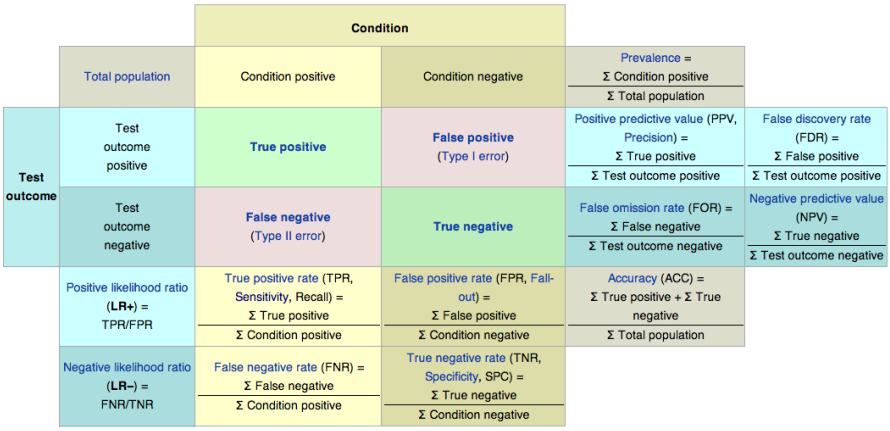


Figure 14: Confusion matrix [Hanson, 2014]

an enemy target, a friendly ship or just noise. The ability of a radar to make the right choices was called ROC. In the 1970s the metric appeared to be very useful for interpreting (medical) test results [Tape, 2005]. The ROC curve contains the relationship between the amount of positive predicted values that should be positive and the amount of positive predicted values that should be negative, in other words, the true positive rate (TPR) and the false positive rate (FPR). An AUC of 0.5 means that the predictions are random; an AUC of 1 means that all the predictions are correct. The advantage of AUC over accuracy is that it deals with unbalanced classes and the cutoff of the raw predictions.

6.1.3 Precision and recall

Whereas AUC looks at the true and false positive rate, precision and recall are metrics that look at relevance. Precision is defined as

$$\text{Precision} = \frac{\text{amount of true positives}}{\text{amount of true positives} + \text{amount of false positives}}$$

and recall as

$$\text{Recall} = \frac{\text{amount of true positives}}{\text{amount of true positives} + \text{amount of false negatives}}$$

Precision may be considered as the metric that indicates whether a model produces more relevant than irrelevant results, whereas recall indicates whether a model produces most of the relevant results. As precision and recall are two metrics instead of one, it is difficult to state when a model is better. It depends on the purpose of the model; sometimes high precision is needed, sometimes a higher recall may be more useful. Because our model will be used for multiple purposes, we look at precision/recall curves, but don't choose which of both metrics we want to optimize.

6.2 Performance results

In this project we distinguish between two different groups of customers on which we will run our models. There is a group of customers who filled in a survey on the ING website, or had a planned conversation with ING, but did not answer the question about NPS and a very large group that did not fill in any survey and did not have a planned conversation. For the first group, we can use the features obtained by text mining. For the second group we cannot, because we do not have any information of them. For this reason we have built models with the text mining features and

without these features. Of course, this separation is made for models that predict promoters, as well as for models that predict detractors. Because of this separation, we split the known data in three parts. The models were trained on 80% of the data and the other 20% was split in two test sets: one set contains the features obtained by text mining; the other one does not. The training set contained 28666 observations, the text mining test set 3583 and the test set without text mining features 3584.
For all four kinds of models we have made ROC and Precision/Recall plots of the models generated by *glmnet*, *xgboost* and Vowpal. Beside those plots, we created plots of the *glmnet*, *xgboost* and Vowpal model with the highest AUC. Finally, we plotted the scores of models with and without text mining in plot, so we can see the difference that text mining makes. In the following sections we will discuss all four sorts of models and their best working *glmnet*, *xgboost* and Vowpal models. We will discuss the plots of their performance and their five most important features. This is just a very small selection, but it would be impossible to discuss them all.

Table 1: AUC of the best *glmnet* models

Model	AUC
promoters with text mining	0.8152856
promoters without text mining	0.6053267
detractors with text mining	0.7980818
detractors without text mining	0.6109227

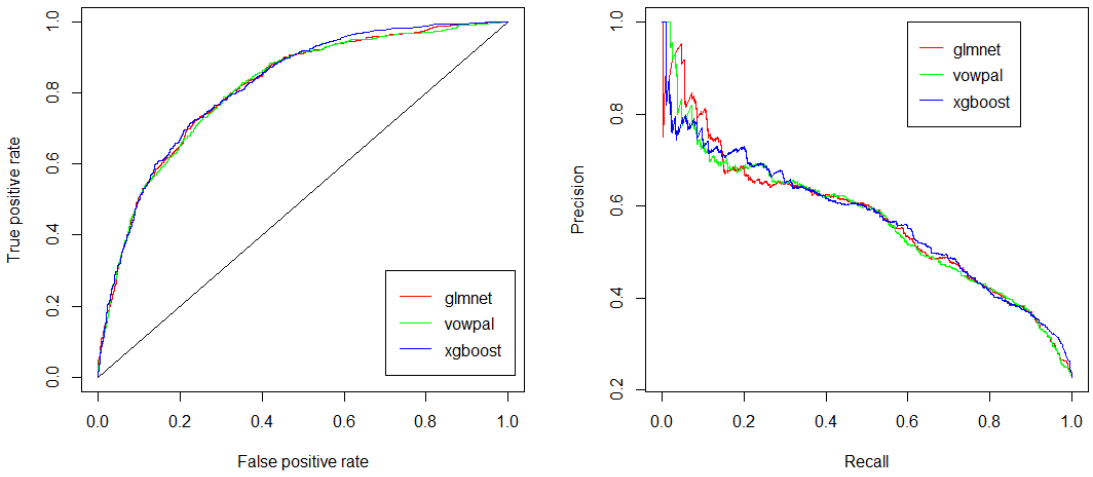
Table 2: Parameters and AUC of the best *xgboost* models

Model	nthread	lambda	maxdepth	AUC
promoters with text mining	3	0.012	12	0.8213348
promoters without text mining	2	0.015	9	0.6096526
detractors with text mining	3	0.012	12	0.7976572
detractors without text mining	2	0.003	9	0.6175699

Table 3: Parameters and AUC of the best Vowpal models

Model	nn layers	L_1	L_2	lr	AUC
promoters with text mining		0.00001	0	0.9	0.8135206
promoters without text mining	8			0.1	0.5705313
detractors with text mining		0	0.0007	0.4	0.79474
detractors without text mining		0	0.0009	0.2	0.5992647

6.2.1 Promoters with text mining



(a) ROC curves

(b) Precision/recall curves

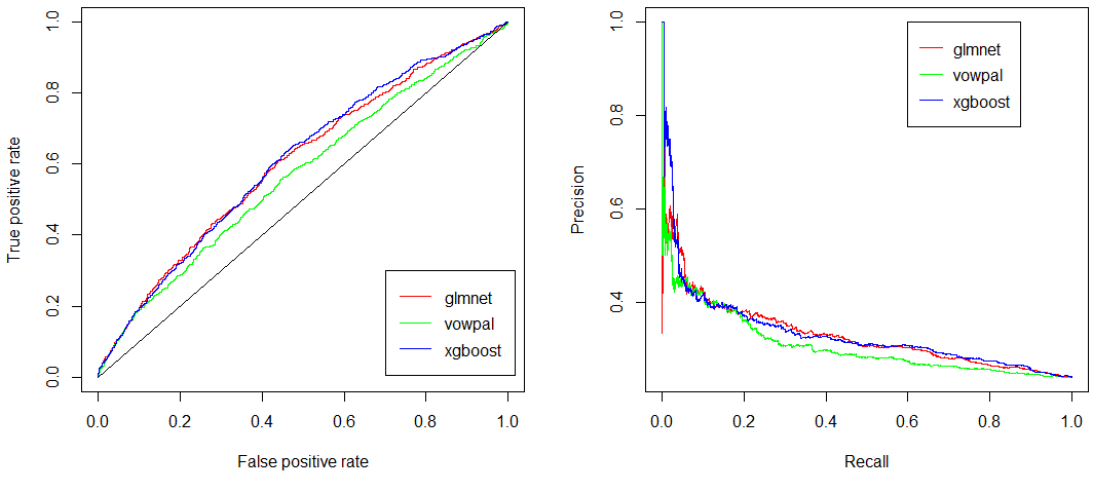
Figure 15: Best promoter with text mining

We built 27 *glmnet* models, 27 *xgboost* models and 1010 Vowpal models using text mining features. In figures **??**, **??** and **??** the results of these models are shown.

With text mining, we obtained some features that really contributed to the model. In the case of the promoters model, the top 5 of features is a text mining feature. The features <removed for security>, <removed for security> and <removed for security> are a strong indication of being a promoter. The features <removed for security> and <removed for security> also score very well, but they indicate that a customer is not a promoter. The full list of features that are used in the *glmnet* and *xgboost* models can be found in table **??**.

In figure 15a we see that the differences between the best *glmnet*, *xgboost* and Vowpal models are quite small. The AUC of the *xgboost* is the highest, 0.8213348. The *glmnet* scores second, 0.8152856 and the Vowpal model scores 0.8135206. So, although the *xgboost* model scores best, all three methods deliver a well scoring model. Looking at the precision/recall curve in figure 15b we also see small differences. When the recall is between 0.0 and 0.2 the *glmnet* model has a higher precision; between a recall of 0.2 and 1.0 the *xgboost* model scores slightly better than the other two. The parameters and scores of the models can be found in tables 1, 2 and 3.

6.2.2 Promoters without text mining



(a) ROC curves

(b) Precision/recall curves

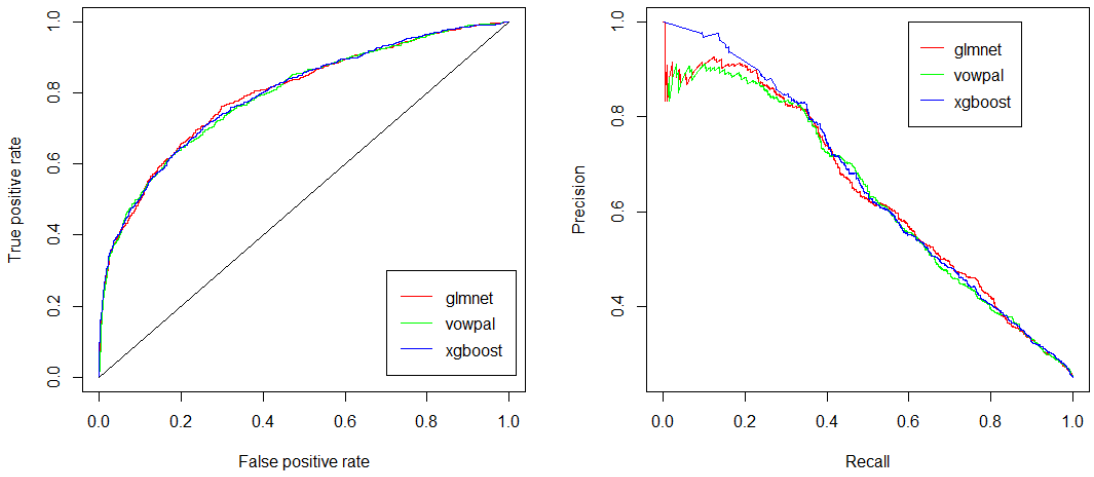
Figure 16: Best promoter without text mining

We built 125 *glmnet* models, 125 *xgboost* models and 1010 Vowpal models that do not use text mining features. In figures **??**, **??** and **??** the results of these models are shown.

When the text mining is not used, we obviously see that other features become important. In this case the top 5 of features is <removed for security>, <removed for security>, <removed for security>, <removed for security> and <removed for security>. The <removed for security> feature may be interesting when we want to know the NPS of different years. The full list of features that are used in the *glmnet* and *xgboost* models can be found in table **??**.

The results of these models are less accurate than those of the models with text mining. In figure 16 we see that the differences between the best *glmnet*, *xgboost* and Vowpal models are again quite small, but are far below the results of the models with text mining. The differences between those two kinds of models are plotted in figure **??**. Looking at the ROC curve of the best models, it immediately becomes clear that the best Vowpal model is less accurate than best model of the other two methods. In this case the *xgboost* model scores best with an AUC of 0.6096526, closely followed by the *glmnet* model that has an AUC of 0.6053267. The best Vowpal model has an AUC of only 0.5705313. The precision/recall curve shows the same results; the *xgboost* and *glmnet* models perform almost equally. In this plot, the curve of the Vowpal model is below the other two models, especially when we look at a recall higher than 0.2. The parameters and scores of the models can be found in tables 1, 2 and 3.

6.2.3 Detractors with text mining



(a) ROC curves

(b) Precision/recall curves

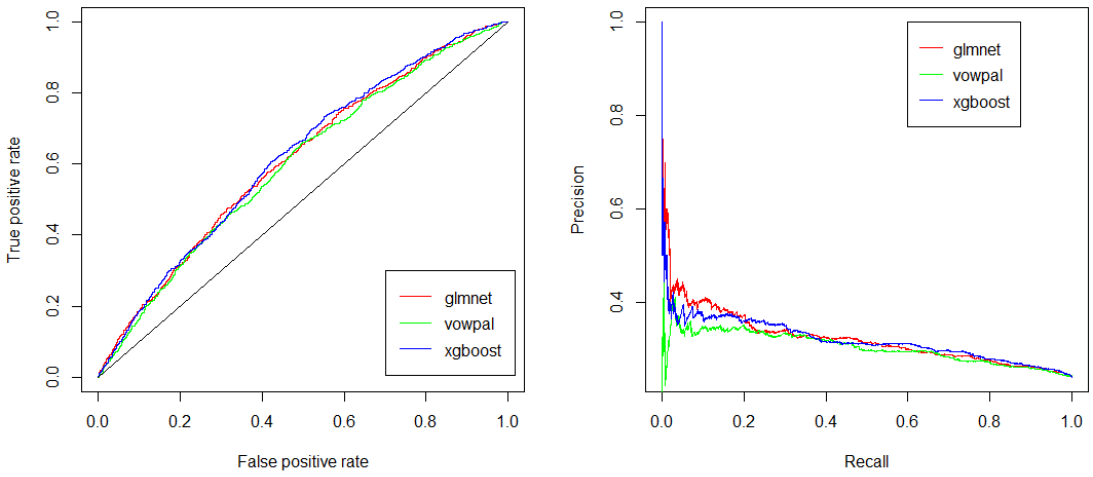
Figure 17: Best detractor with text mining

We built 27 *glmnet* models, 27 *xgboost* models and 1010 Vowpal models using text mining features. In figures **??**, **??** and **??** the results of these models are shown.

For this type of models, again, we obtained strongly contributing features through text mining. Not surprisingly, these were the same features as in the promoters model, but in this case, they indicate the opposite. So <removed for security> and <removed for security> indicate a detractor, <removed for security>, <removed for security> and <removed for security> indicate "not a detractor". The full list of features that are used in the *glmnet* and *xgboost* models can be found in table **??**.

Whereas the *xgboost* had the highest AUC for the models concerning promoters (using text mining), here the AUC of the *glmnet* model is the highest; 0.7980818. The *xgboost* scores second, 0.7976572; and the Vowpal model scores 0.79474. So, again all 3 methods deliver a well-scoring model and the differences between them are small. We can see these small differences in figure 17. The ROC curve of the *glmnet* model is a bit higher than the other two models. However, looking at the precision/recall the *xgboost* model scores better for a recall smaller than 0.2. With a recall between 0.2 and 1.0 the models almost perform similarly. The parameters and scores of the models can be found in tables 1, 2 and 3.

6.2.4 Detractors without text mining



(a) ROC curves

(b) Precision/recall curves

Figure 18: Best detractor without text mining

We built 125 *glmnet* models, 125 *xgboost* models and 1010 vowpal models that do not use text mining features. In figures **??**, **??** and **??** the results of these models are shown.

For the models predicting detractors without text mining the <removed for security> and <removed for security> are important, just as for the promoters model. The other three features of the top 5 are <removed for security>, <removed for security> and <removed for security>. The full list of features that are used in the *glmnet* and *xgboost* models can be found in table **??**.

The results of the models are slightly better than those of the promoter models without text mining. The *xgboost* model is the best model once more, with an AUC of 0.6175699. The *glmnet* scores similarly with an AUC of 0.6109227 and the Vowpal is again the worst with an AUC of just 0.5992647. However, the precision/recall curves are all lower than those of the promoter models. Compared to the ROC curves we see a similar picture. The *xgboost* and *glmnet* models perform almost the same, the Vowpal models perform worse. The parameters and scores of the models can be found in tables 1, 2 and 3.

6.3 Conclusion

We decided to use ROC curves and precision/recall curves to compare our models. The AUC of the ROC curve was chosen as metric to choose the best models. For all four sorts of models we built models using *xgboost*, *glmnet* and Vowpal. When predicting promoters, the *xgboost* models have the highest AUC. When predicting detractors with text mining features, the *glmnet* model is the best one and when predicting detractors without text mining features, again a *xgboost* model has the highest AUC. The features obtained through text mining make a huge difference. When using these features, we achieve AUCs around 0.8. When they are not used, we obtain AUCs around 0.6.

7 Dashboard

Part of the project was to deliver a dashboard so we can show ING employees how the NPS is distributed. Not everyone in the business understands these types of complicated models and a dashboard would thus be easier for employees to work and communicate with. In the end we delivered two versions of the dashboard. The first version will be implemented in the business after we are gone, the second is made to meet the software engineering requirement of this project. In this section we will describe how we made both versions and what their strengths and weaknesses are.

Building the dashboard is the last step of the process. This is shown in figure 19. We selected the best models and stored the data of all investment customers in a csv file. We ran the selected models on the rest of the customers (of whom we do not know their NPS) and saved the predictions in one or more csv files. The last step is to make a dashboard out of these csv files. As mentioned in the previous paragraph, we execute this last step for two different dashboards.

DISCLAIMER: The Power BI dashboard screenshot is removed for security and privacy reasons and the D3 dashboard contains fake data.

7.1 Power BI

The dashboard that we implemented in the business was built with Power BI. Power BI is a rather new Business Intelligence software package from Microsoft. It is still in beta, but the functions that were available were more than enough to suit our needs. Power BI is also used by another team at ING, to which we will hand over our product, so they are familiar with the software and know how to maintain it. A screenshot of the dashboard is shown in figure **??**.

Power BI is one of the most innovative BI packages to date. It is really simple to use and even non-technical people can drag and drop variables to build charts. It has support for a lot of Microsoft software built in, so if the business uses their software suite it is very easy to combine multiple programs with each other. A couple of well known programs are Excel, Access and Azure.

We organized multiple presentations for other teams within ING to show the dashboard and they loved the clear visualizations. For example, it is possible to select a specific bar from a bar chart, after which the other charts on the same page will filter their data for that selected group. The other BI tools that are used at ING do not offer this type of interaction, so we received positive feedback about this functionality. One team even wanted to implement our dashboard as

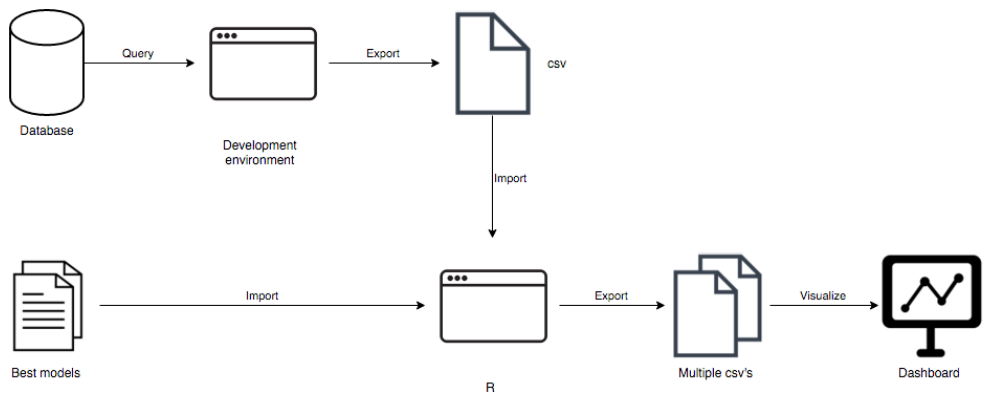


Figure 19: Step 5 in the process

a weekly report. Together we built the first beta version. Unfortunately there was no time left to finish the dashboard completely, because that team shifted their goals after a couple of weeks.

There are however also downsides to Power BI. To really make use of all its features, you have to upload the data to Microsoft's cloud storage. Big companies like ING do not want to do this, because they need to know where their customers' data is stored and who has access to it. Since storing the data in the cloud is not an option, we used a local version of Power BI. This makes it more difficult to distribute the dashboard to other parties, but ING is working on a solution.

We think our product is in good hands and the business can do great things with it in the future. We tried to match the expectations of our client as much as possible and we think we have succeeded in that goal. It is unfortunate that a reorganization at ING occurred during this project, but the latest version of our dashboard could be changed quickly to better suit their new needs. Next we are going to discuss the dashboard that had to be made to meet the software engineering requirement of this project.

7.2 D3

We made the second dashboard from scratch. We wanted to make a dashboard that would meet the software engineering requirement of this project, but also wanted to address the shortcomings of Power BI. We decided that a website would be the best choice to build our own dashboard. We used an open source javascript library called D3 in combination with HTML and CSS. We used Jasmine for testing purposes.

D3 is a powerful library to visualize anything you want. You can for example make spinning globes, make little games and most importantly show charts. There is however a slight learning curve. We spent over three weeks finishing the dashboard, but we think it was the right call to use D3. Although we had less time to spend on solely improving our models, it allowed us to deliver a product that we built ourselves, showing what we learned during our bachelor programme as well as during this project. A screenshot of the dashboard is shown in figure **??**.

It looks almost the same as Power BI and that is because Power BI uses D3 too. We tried to replicate the first version of our Power BI dashboard and it is almost identical. Our D3 dashboard is in some aspects a better choice to implement than the Power BI version. Its data does not have to be in the cloud: with a simple internal server it is possible to host the website for ING employees only. You can also alter the dashboard to make it exactly as the business requires, whereas Microsoft will decide which features are going to be implemented in Power BI.

<removed for security>

7.2.1 Data preparing

The D3 dashboard will need multiple csv files with specific information. These csv files are built in the csv builder which we have written in R. The csv builder needs a dataset in order to make subsets of it. First, there is a function that selects the data of the last month in the database. Then, there are functions to calculate the amount of promoters in a matrix or list, a function that calculates the amount of detractors and a function that calculates the NPS. Then, there are functions that use these calculations and export various csv files. These are the print functions. A description of these functions can be found in Appendix **??**.

7.2.2 Implementation

We will now get a little more technical. In figure 20 the structure of the D3 dashboard is shown. You can see that it is possible to make a dashboard with just a couple of files. We have divided the different aspects of websites into different folders. There is a dedicated folder for each css, javascript and data file. We will now explain the most important files of the dashboard.

index.html This is the HTML file where the structure of the website is declared. We used six different *div* elements that hold the six different charts. See figure **??** for more information. Each

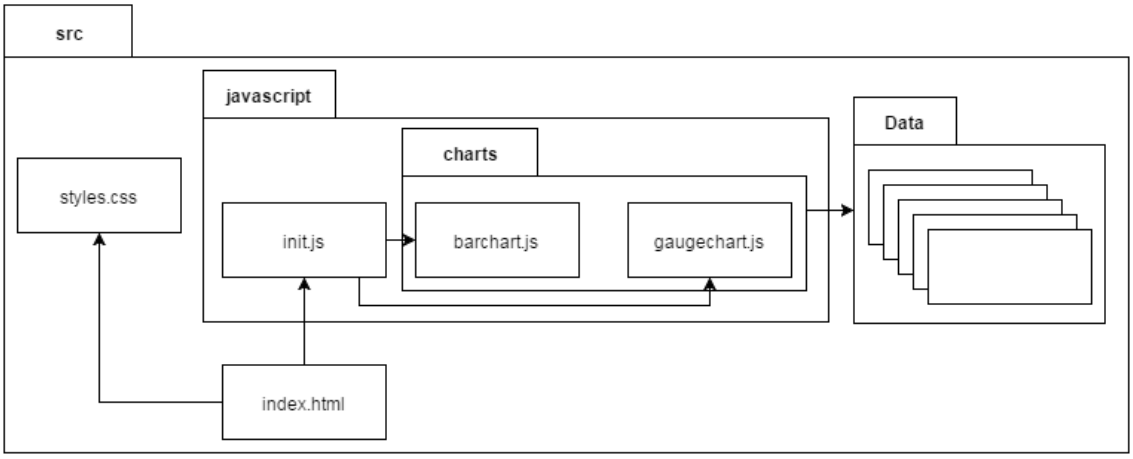


Figure 20: D3 src folder structure

of these *div* elements has its own id, so the charts can be allocated to their position. There is also a button placed in the lower right corner for the user to refresh the charts.

This file is the heart of the website: it will call all the other files that are necessary. It will call *styles.css* to style the website and *init.js* to generate the charts. When the button is pressed, a function inside of *init.js* will update the charts with the data that is stored in the specific data folder.

styles.css This file is used to style the graphs and the *div* elements. Here we declared for example how the axes of the bar chart should look; that negative NPS scores should have a red color; and which font size the chart titles should be. Doing this in one file make it easy to change multiple aspects of the website with minimal code. This makes the site maintainable and flexible.

init.js There are three functions in this file: $init()$, $updateCharts()$ and $isNumeric(n)$. The function $init()$ will be called when the website finishes loading the html elements. This is to make sure every id is declared so that the charts created can be placed inside that *div* element. When it is called, it will create one *GaugeChart* and five *BarChart* objects. It will render them all and in the end all the charts are updated via the $updateCharts()$ function. This function is also called when the button is pressed.

barchart.js This is where D3 is used to create a bar chart. We will only cover *barchart.js*, because *gaugechart.js* has the same structure. A *BarChart* object consists of two parts: the axes and a body part. Two different functions will take care of rendering the axes. When the chart has just been created, $renderAxes()$ will be called to initialize the components of the axes. If the chart is already in use and the chart has to update the axes, then $rescaleAxes()$ is called. This is where the chart determines whether the data of the x-axis is linear or ordinal and rescale the axes if necessary. The x-axis will be linear if the data contains only numerical values and will be ordinal if the values are strings.

The body will always be re-rendered. The body contains the bars that will represent the data. If the value of a bar is negative, then the class attribute of that bar is *neg* and otherwise it is *pos*. This way, we can declare in the *style.css* what the color should be for both classes. This functionality is also present in the *GaugeChart*, but instead of using bars there is an arc that changes color.

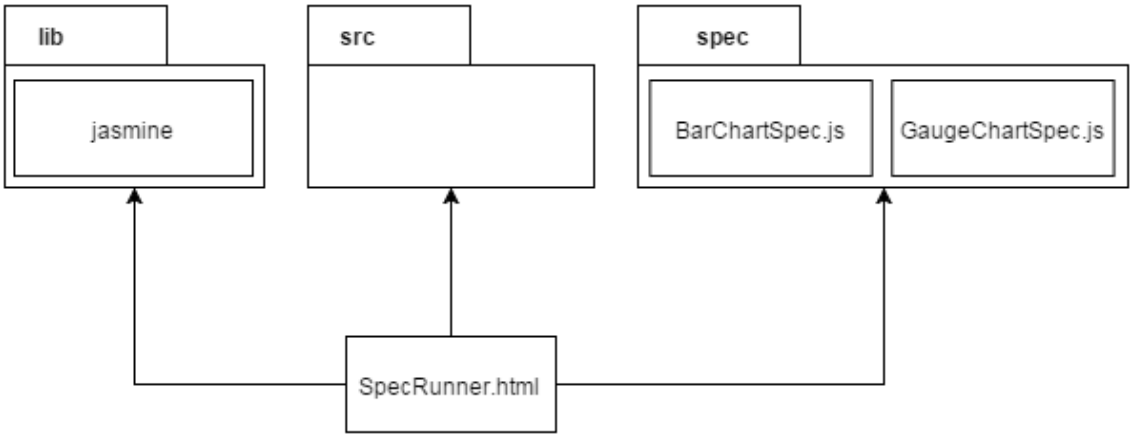


Figure 21: Jasmine structure

7.2.3 Testing

We are using the Jasmine library to help us test the code. The structure of the website had to be extended to use this library. This structure is shown in figure 21. To run the tests, the *SpecRunner.html* file has to be opened in a browser. This file contains the references to the source code and the tests. There are three folders: in the *lib* folder all the code to use the Jasmine package can be found, in the *src* folder the source code described in section 7.2.2 is saved, and the *spec* folder is where all the tests are stored.

We wrote two test files: $BarChartSpec.js$ and $GaugeChartSpec.js$. Each file only contains tests that are relevant to their subject. There are tests for checking the dimensions of the charts, whether they created the right class attributes and whether the chart is placed in the right position. There are around 200 lines of code versus 300 lines of source code. This shows that we took testing seriously and we wanted to deliver high-quality code. We are not the only one who think our code is of above average quality, as is described in the next section.

7.3 SIG evaluation

We sent the code of our own dashboard to the Software Improvement Group (SIG) in week 8 and 12 of the project. The evaluations can be found in appendix **??**. The first time we sent our code, our dashboard was far from done. We had spend only two weeks on building it and at that time we did not really understand how D3 worked. It came as a surprise that we still earned four out of five stars for this unfinished code.

The reason we did not earn five stars was because we had lots of duplicated code and the size of the methods were too large. At that time, we had five different files to create five different bar charts. These files looked almost identical, which is an inefficient programming concept. We knew that this was going to be a problem for our evaluation, but there was no time left to change it.

We fixed this issue in the final version of the dashboard. As showed in figure 20 we now only use one bar chart file to create multiple instances. We are pleased with the solution; it was difficult to understand D3 in a short time and implement the features in the correct way. We fixed the issue by making $BarChart(id, path)$ a class which takes an id and the path to the data. The id refers to the id of the *div* the chart should go in. This way you can add multiple charts to different *div* elements and each chart will only take care of its own environment.

We also improved the amount of tests. In the first edition of the dashboard there were no tests. After fixing the issue described above, we started writing tests for the new situation. We built little unit tests that test only one thing at a time. SIG is also positive about these added tests. They stated that the ratio between source code and test code is very healthy. We have worked to reach this result and we are very pleased with SIG's response.

Unfortunately, we could not improve our four star rating to five. We were close to achieving this, but the unit size of the methods were still too large. This is partly because of how D3 works and partly because we did not want to implement extra methods just to keep unit size small. Looking back, we could have put in more effort in decreasing unit size, but we think we used our limited time to fix other more urgent issues.

7.4 Conclusion

We have created two dashboards: one in the powerful program Power BI and one we developed ourselves. The Power BI version received positive feedback from the business and the D3 version received positive feedback from SIG. Both dashboards have their own strengths and weaknesses, but overall we are very pleased with the results. Both versions could be used by ING, but they are only interested in the Power BI version. This is the version that will be implemented and hopefully used for a long time.

8 Conclusion

Although we experienced a very busy three months, they were absolutely worth it. We have learned a lot and we might have found our future profession. Working together with the team at ING was very satisfying and it was interesting to see how the business operates. This made the project a success for us.

We have mixed feelings about the results. On one hand we score above our expectations with the text mining features, but if we leave them out the performance drops significantly. Although this is unfortunate, we should keep in mind that this is the first time we have applied machine learning algorithms and we have not identified other research instance of predicting NPS using this method.

We can see that the models we used gave approximately similar accuracy levels. *xgboost* and *glmnet* performed almost identically, as is to be expected because we used important features of *xgboost* as imput for *glmnet*. It was interesting to see how Vowpal Wabbit works and the results show it works significantly differently from the other models. Unfortunately this did not mean it outperformed the other models, but maybe in a different environment it can excel.

Building the dashboards took quite some time. Looking back, we have put in more time than we anticipated at the start of the project. We do not regret this, since we intended to deliver quality work. <Removed for security>

It was a delight to work with the people at ING and with our supervisor Hayley. They all wanted us to succeed and put a lot of effort into our project. At the start of the project, the expectations were unknown and this made it exciting to begin the project.

<Removed for security>

9 Recommendations

There are many things in our project we could improve on. Since the project only lasted about three months, we had limited time to explore all possible solutions. We have also found a lot of literature and information during our research that we were unable to experiment with. We will sum up our recommendations in this chapter.

9.1 Feature extraction

Although we used many features, the results were subpar without the text mining. We would like to find new features that we can apply on all customers and not only on those that have filled in a questionnaire. If we had more time, we would have tried web scraping a news site or combining more information from other databases at ING.

For *xgboost* and Vowpal we used the complete featureset to build the models, but for *glmnet* we used the relevant features of *xgboost*. In our case *glmnet* still performed as well as the other models, but we do not know whether another approach would have made an improvement. More research should be done to know the impact on the performance of *glmnet*.

9.2 Machine learning

An obvious recommendation is to try other machine learning methods. We read that neural networks perform really well, so this is one model that we want to try to build.

We have made a script to automate the process of building different models with different parameters, but if we would have access to a more powerful machine we could have produced more models with other parameters. Parameters we would have liked to experiment (more) on are e.g. which cross-validation to use; the amount of regularization; and how deep the decision trees are allowed to be.

9.3 Dashboard

The D3 dashboard we made does not have many functions. It would be nice to be able to filter or to choose the type of data you want to see. This is what is possible in Power BI and one of the reasons ING does not want to use our dashboard. If more time were spent on the functionality of the D3 dashboard, this would make it more valuable.

References

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[Chen et al., 2015] Chen, T., He, T., and Benesty, M. (2015). *xgboost: Extreme Gradient Boosting*. R package version 0.4-2.

[CustomerGauge News, 2008] CustomerGauge News (2008). Ing direct usa bank hits nps hi-score of 60. `http://customergauge.com/news/ing-direct-usa-bank-hits-nps-hi-score-of-60/`.

[Dobronte, 2012] Dobronte, A. (2012). Why there needs to be a european variant of the net promoter score. `https://nl.checkmarket.com/2012/01/we-need-an-nps-eu/`.

[Friedman et al., 2010] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22.

[Ghahramani and Jordan, 1994] Ghahramani, Z. and Jordan, M. (1994). Supervised learning from incomplete data via an em approach. pages 120–127.

[Grisaffe, 2007] Grisaffe, D. B. (2007). Questions about the ultimate question: conceptual considerations in evaluating reichheld's net promoter score (nps). *Journal of Consumer Satisfaction Dissatisfaction and Complaining Behavior*, pages 20–36.

[Guyon and Elisseeff, 2006] Guyon, I. and Elisseeff, A. (2006). *An Introduction to Feature Extraction*, volume 207 of *Studies in Fuzziness and Soft Computing*. Springer Berlin Heidelberg.

[Hanson, 2014] Hanson, N. (2014). De-confusion tables, a shiny application for understanding binary classifiers.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer-Verlag New York.

[James et al., 2014] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143.

[Langford, 2015] Langford, J. (2015). Vowpal wabbit. `https://github.com/JohnLangford/vowpal_wabbit/wiki`.

[Motoda and Liu, 2002] Motoda, H. and Liu, H. (2002). Feature selection, extraction and construction. *Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol*, 5:67–72.

[Qu, 2013] Qu, Y. (2013). Nps vs. ces – are they mutually exclusive? `https://www.cebglobal.com/blogs/nps-vs-ces-are-they-mutually-exclusive-3/`.

[R-Bloggers, 2013] R-Bloggers (2013). Finding patterns amongst binary variables with the homals package. `http://www.r-bloggers.com/finding-patterns-amongst-binary-variables-with-the-homals-package/`.

[Reichheld, 2003] Reichheld, F. F. (2003). The one number you need to grow. *Hardvard Business Review*, 81(12):46–54.

[Ritson, 2007] Ritson, M. (2007). Net promoter scores australia 2006. `http://www.tmiaust.com.au/downloads/NPS/Mark_Ritson_NPS_Survey.pdf`.

[Roughgarden et al., 2013] Roughgarden, T., Sharp, A., and Wexler, T. (2013). Guide to greedy algorithms. http://web.stanford.edu/class/archive/cs/cs161/cs161.1138/handouts/120%20Guide%20to%20Greedy%20Algorithms.pdf.

[Tape, 2005] Tape, T. G. (2005). The area under an roc curve.

[van Dessel, 2014] van Dessel, G. (2014). Measuring customer satisfaction: Csat, ces and nps compared. https://www.checkmarket.com/2014/11/csat-ces-nps-compared/.

Appendices

Removed for security.