;

Time series predictions for bank account balances

Bernd Kreynen
Chantal Olieman
Felix van Doorn
Max Spanoghe

TUDelft

Abstract

For our bachelor project we have been using machine learning to predict account balances for a large Dutch bank holding company.

The company's main interest is the integration of machine learning techniques in their systems. To enable this we have been asked to develop a product to predict account balances for the clients of associated banks.

With the clients interest in machine learning in mind we have developed a framework enabling the user to implement different machine learning and non machine learning models. The framework makes it easy to compare the implemented models using different error measures, parameters of inputs and lets the user visualize the results easily.

In this framework we have implemented our own models for the account prediction. To compare our models we started with implementing a baseline, next to this baseline we have implemented two non machine learning and one machine learning model.

The data we used to train and validate our models has been derived from the clients data warehouse. We have cut the accounts on different criteria like activity and the period they have been with the bank. After that we have normalized the data to be able to better interpreted and process it.

The machine learning techniques we want to implement require a lot of training examples, this made us decide implement a clustering model as well to create more data to train our models on. Eventually the clustering did not give us the expected results and we decided not to use it for our final model.

To give our client a suited recommendation about the machine learning libraries to use on their systems, we have implemented the same clustering method with two different libraries. After this comparison we were able to recommend our client the Scikit-learn library over the more low level Tensorflow library. From this point on we used the Scikit-learn library as well for the implementation of SVM model.

For the regression we implemented the L-1 prediction, OLS method and an SVM. Compared to the baseline, our SVM model gave the best results, however the results of the L-1 prediction closely followed the results of our SVM model.

After a better comparison we have discovered that in some cases the SVM model makes a prediction is almost exactly the same as the L-1 prediction, one the other hand, various other predictions are not based on this pattern at all. We therefore assume that after tweaking the SVM more, it will preform better and show significantly better results than the L-1 prediction.

For now we did not have time to tweak our SVM, but we have tried different inputs and parameters. As a future improvement these parameters can be tested in more detail and it would be interesting to take a closer look at different militarization methods and error measures.

In conclusion we were able to test machine learning techniques with the client's data by implementing a well working SMV model for account balance prediction. This model works on the clients systems and is validated on real client data.

Furthermore we provided our client with a framework that allows them to easily implement machine learning and non machine learning models. This framework provides the user with interfaces to build models, standard data operations and error measures. This allows the user to quickly research many different configurations. We used this framework ourselves during this project to compare our machine learning and non machine learning models.

Preface

As the last part of the undergraduate Computer Science program students work on the Bachelor Project. In this project, we had to produce a product that solves a real-world problem and carry out research to take possible solutions for this problem into account.

For our project, we have created a product for a large Dutch bank holding company. This company is constantly trying to improve the clients experience with state of the art IT technology.

The client has asked us to investigate the possibility of account balance predictions for their customers. They were especially interested in the applications of machine learning techniques into their systems. As a group, machine learning was one topic of interest in computer science and thus we saw great potential in this project.

Contents

1 Introduction

Machine learning can give us the possibility of finding patterns and trends in data. The insights gained into this data can then be used to create predictions. Using machine learning algorithms, it is possible to make predictions for time series data. For this project, we made time series predictions for bank balances. More specifically, we looked into using Support Vector Machines in order to make these predictions [19].

The purpose of our project is to develop a model that can predict the bank balances of customers of a large financial institution. This institution is also interested to see if such predictions could be made using Google TensorFlow and how such a system could be integrated into their infrastructure.

For this project our research question reads: How well can we predict bank balances? Traditionally these models have been applied to predictions of stock market data [22]. Interestingly enough, we could not find any published work concerning such predictions for bank balances. Many machine learning tools exist in Python. To be able to utilize some of these to the fullest we implemented our own framework in Python which can be used in combination with these tools. This framework is designed to conveniently combine a prediction model with different error measures and validation methods.

We will start this report with a chapter in which we clearly state the problem at the core of this project. In the following chapter we will describe the data that we have at our disposal. In the subsequent chapter we will describe the clustering techniques that we investigated in detail during the course of this project. After that we will discuss the machine learning algorithms that we used to create our predictions. In the next chapter we will elaborate on the non-machine learning and machine learning techniques we employed to create predictions. We will then move on to the design of our framework. We will follow up on that with a chapter displaying the results of our model. We have dedicated a separate chapter to the discussion of these results. In the penultimate chapter we will give an overview of the entire process. Lastly we will end this report with our conclusion.

2 Problem

2.1. Problem Definition

The goal of this assignment is to develop a prototype of an application, that will make bank balance predictions by using machine learning techniques. A secondary objective, is to evaluate whether such a system can be built using Google TensorFlow and how this could be integrated in the existing sytems of our client. The whole project description of our client can be found in Appendix E.
We framed our research question as:
How well can we predict bank balances?

2.2. Problem Analysis

Our client is looking for a way to help their customers in the best possible way, therefore they want to know if they can predict the account balance of their clients to help them foresee future difficulties. Keeping in mind that no system works perfectly, the client prefers to know how well we can predict the account balances. This means; How far into the future can we predict? For what clients can we predict? How dynamic is the model for other clients?
Another important aspect is Google Tensorflow. Our client wants to know how well Google Tensorflow can be integrated into their systems and how well it works for their desired purposes.
During the problem analysis we also made a project plan, which can be found in Appendix A.

2.3. Solution

We have designed a framework in which different machine learning and non-machine learning models can be implemented. This framework is not only useful for this project, but can be useful to implement other models and other projects as well. More about this framework can be read in section 9
In this framework we have implemented clustering in two different ways, these two implementations are described in chapter 5. For the regression we have compared three different models, described in chapter 7 and 8. Results for the clustering and regression can be found in chapter 10.
This solution does not only give the client a working model with well explained results, but the framework provides the client a way to further extend this model or easily implement their own model for future use.

2.4. Success measurements

At the beginning of the project, we were not able to make an estimate of how well a good solution should preform. Therefore we have decided to work with a baseline. Every improvement from this baseline can be seen as a small success in the project. This baseline thus allows us to measure our success during and at the end of the project.

3 Data

3.1. Main statistics

For our project, the client provided access to a data warehouse. We had to decide what data was useful for our specific problem. During the research phase of the project we made various queries and plots of data statistics. For us, this was mainly an introduction into the database and this gave a clearer insight into how much data was available and how the data was distributed. It also gave us an idea on what to use as input for our algorithmic solutions. However, a lot of these were not relevant to put into the main section of the report. The full chapter with extra plots can be found in Appendix G. In our solution and statistical analysis only data after 2004 is used, because the data warehouse has inconsistent entries before this year. The data warehouse was set up around this time and this is most likely the reason why entires before this date are inconsistent.

3.1.1. Amount of bank accounts

The total amount of bank accounts in our dataset is 1559534. These are payment accounts created before 2004.

3.1.2. Average balance per month

The goal of this project is to create predictions for a customer's account balance. Therefore, we needed to look at the account balances of customers. We looked at the average balance of each customer for each month since 01-01-2004. These are then normalized by converting them to z-scores. Finally the z-scores of each customer are averaged for each month. This gave us a representation of how the average customer's balance develops over the year. See figure 3.1

3.1.3. Minimum number of transactions

This graph shows the distribution of the number of accounts with a minimum amount of transactions per month, starting from different years shown in different colors. The x-axis was cut off at 60, because the amount of accounts starts becoming too low to be of interest to us. The y-axis was cut off at 600000. This only cuts off the 0-5 bin which is not interesting, because this is the group with a minimum of zero transactions per month which is just every account (1559534).

3.2. Promising data

With a data warehouse containing over one and a half million payment accounts and detailed information on each of these accounts, it is important to know which information is of use and which is not. This is impossible to know for certain without testing, especially since our machine learning techniques can be seen as black box methods. Therefore, it is hard to predict which statistics are useful and which are not. During

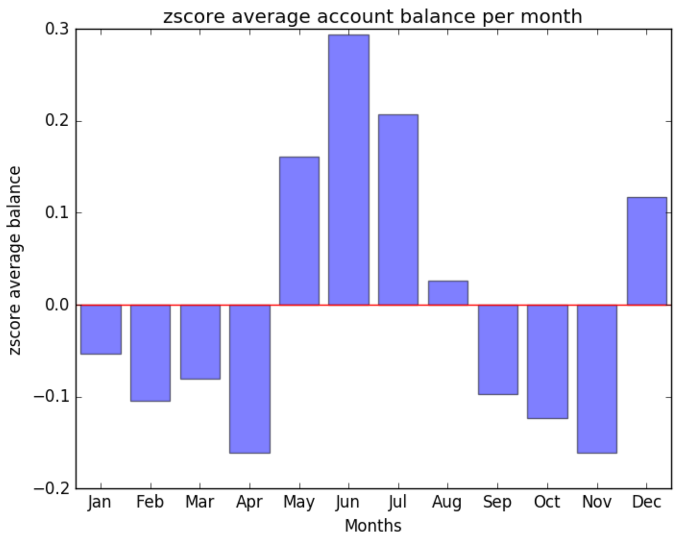

Figure 3.1: This graph shows for each month the z-scores of all accounts averaged. Here you can see the main trend for the balance on bank accounts of the year.

the benchmarking, we tested which data improved the results and which did not. More about this in Section 8.1.5.

3.3. Datasets

In order to create an accurate model based on the data available to us in the data warehouse, we need to construct our own datasets. For the sake of confidentiality, we cannot publish or share these datasets. The Python scripts and SQL queries used to create them can be found in our project repository, this is a private repository accessible only to the client. In the next sections, we will first describe how we filtered out the data that we want to use and which cut off decisions were made. Then we we describe in more detail the datasets that we created and how they are divided into validation and training sets. Normalization of the data will also be discussed, together with an example of what can go wrong and how we fixed this.

3.3.1. Cut off decisions

Before actually using the data as input for our algorithms, we made some decisions on what subset of the data we wanted to use. A good example is the fact that some accounts are inactive and not very interesting to predict for. Secondly, some accounts are new and have less information about their behavior. The cut offs are the following:

1. All accounts need to have at least 5 transactions for every month of during the last 6 years. (From 2010-01-01 to 2016-04-30)
2. Every account needs to have at least a subscription of 76 months at the bank. (roughly 6 years)

The second cut off ensures that we have accounts with a lot of information for the testing phase. The implemented methods might not need data of such a long period for all accounts.

3.3.2. Trainingset

For the machine learning techniques the data needs to be divided into training and validation sets. Normally this is split randomly into two sets. However, we are dealing with time series and therefore the data should not

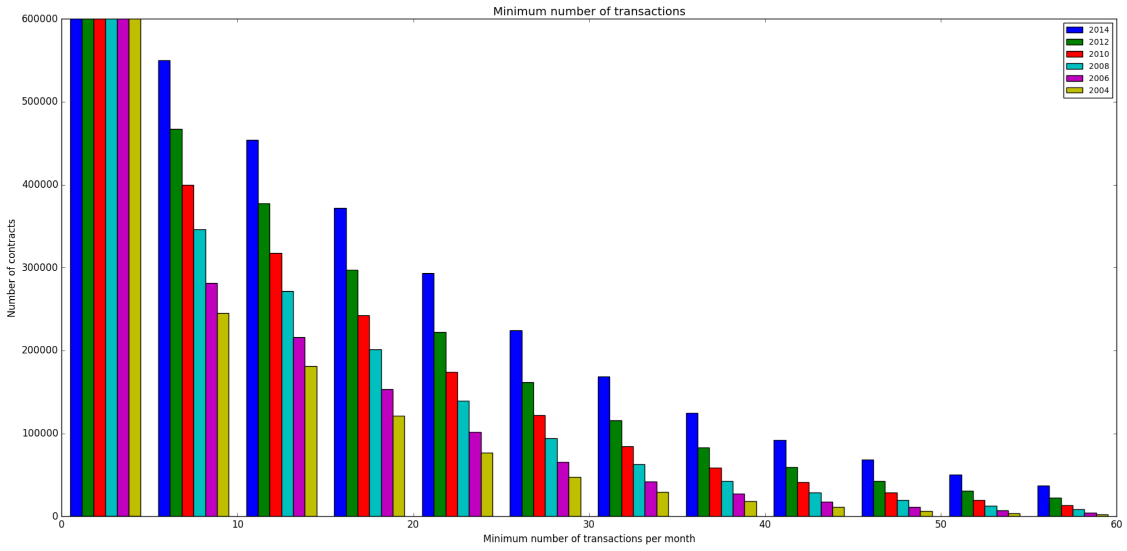

Figure 3.2: This graph shows the amount of account that have a certain minimal amount of transactions for each month since the year you can find in the legend. On the x-axis, you can find the minimal transactions. On the y-axis, you can find the amount of account for each year with a different color. The leftmost bin is cut off, because this bin contains all payment accounts (1559534)

be split on a unique identifier, but rather on time. By that we mean, that we divided the dataset as following. All the clients in the filtered set have to be at the bank from 2010-01-01 to 2016-04-30. So, the training set consists of all the feature vectors from 2010-01-01 until 2014-01-01 which results in a set of 4 years of training data.

3.3.3. Validationset

The validation set is also split on time. To make sure no data is leaked from training to validation and because our feature vectors have a time window of 12 months, we opted for selecting the validation set to be only after 2015-01-01 and thus having a gap of one year of data which is not used for either training or validation. The validation set are all feature vectors from 2015-01-01 until 2016-04-30.

3.4. Normalization

It is better to normalize data instead of using the raw value, because this will lead to scaling issues. For the feature vector as input for the algorithms, we thus need to normalize the data.

3.4.1. Z-scores

A common way to normalize data is the z-score or standard score. A z-score can be obtained from a raw score through the following formula:

$$Z = \frac{X - \mu}{\sigma} \tag{3.1}$$

Where X is the raw value, μ is the average value and σ is the standard deviation.

This conversion is used for statistics that can encounter scaling issues. In our case, some payment accounts have a much higher balance than others. The z-score describes the deviation from the mean in terms of the standard deviation. This allows for more accurate and meaningful comparison of individuals across our sample. [24]

3.4.2. Issues

The average and standard deviation (std) used for the z-score is calculated for each account independently. After further analysis of the prediction results, we found out that sometimes this normalization can be deceiving. An example is given below:

User x
average balance for all his months: 511.88 euro
standard deviation for all his months: 1914.39 euro

By using the 10 months average prediction we got an MSE of 0.000004. This is considered a small MSE, so we interpreted this as a very good prediction. We further investigated the feature vector and came to the following conclusion; Our error is presented in z-scores, which means the error gives the amount of standard deviations the prediction is off compared to the real value. Because we used MSE here, we actually have an error of 0.002 in terms of the standard deviation. If you then calculate the actual error as an absolute value, you get

$$0.002 * 1914,88 = 3.83$$

Which means we predicted this month for this account with an error of roughly 4 euros. If you check his feature vector, you can see that 4 euros relative to the 30-50 that you see in this previous months, is actually not exceptionally good. It is a decent prediction, but the calculated error deceives us into thinking this prediction is near perfect.

The cause of this is bad normalization. The average balance of this account is 511.88, whilst in most months the balance is fluctuating around 50. The high average is caused by a small number of months where the account balance is raised to 14000 euros. These few months skew the mean and standard deviation, so that the normalization is not representative anymore.

We came to the conclusion, that this problem applied to a lot of our accounts an we had to find a way to solve it. Our solution was taking the averages and standard deviations of the middle .875 percentile of the data. This means we first filtered out for each account the top and bottom 12.5% of balances as outliers. After this, filtering the mean and standard deviation are calculated. This gives a more representative normalization and enables us to interpret the errors better and make them comparable with errors of other accounts.

4 Clustering

4.1. Approach

To train our model we need training examples, however a single account might have limited data to train on, even if the client has been with the bank for a while. When we use one year for validation and a client has been with the bank for two years, we would still only have 12 data points to train on.

To alleviate this problem our models could be trained on the data of multiple payment accounts. To train on accounts similar to the one we are predicting, we decided to cluster the payment accounts. When these payment accounts are put into clusters, it is possible to train a model on a cluster and to use this trained model for the prediction of all accounts belonging to that cluster. By doing this, there will be more data available to train on. Furthermore with this approach it is also not necessary to train the model again for every single account that we want to make a prediction on. One model can be trained for the whole cluster. When we want to make a prediction on a new account we can find the cluster it belongs to and use the trained model to make a prediction. In the following sections of this chapter a description will be given about how we clustered the payments accounts.

4.2. Clustering method

In this section the method, used to cluster the payment accounts will be described. There are two common clustering techniques, hierarchical clustering and K-means clustering [21]. As described in the data chapter, we are working with a large amount of accounts. The time complexity of k-means is better than the time complexity of hierarchical clustering [21], therefore we only considered K-means.

4.2.1. Clustering Technique: K-means

KM is an unsupervised learning technique since the class labels are not known and no classes will be given as output [20]. This is needed since we have unlabeled account that need to be clustered together so that the model can be trained

The standard version of K-means is a relatively simple algorithm consisting of one initialization step and two iterative steps. The idea is that for each of the K-clusters we have a mean which characterizes this cluster. This mean is updated until the clusters do not change any more [15].

1. **Initialize:** Initialize the mean for all of the K clusters (randomly).
2. **Assign Points:** Each point in the dataset is assigned to the cluster with the nearest mean according to a predetermined distance metric.
3. **Update means:** For each of the K clusters the mean is updated. The mean of a cluster is now actually calculated as the mean from the data points that belong to the cluster.
4. **Repeat** step 2 and 3 until no data points change clusters between iterations.

4.2.2. Implementation

Python has a broadly used library called scikit-learn which has an implementation of different clustering algorithms [20]. An algorithm to perform K-means is included in these. This implementation was used to cluster the payment accounts, however an implementation of K-means was also made in TensorFlow to compare the two libraries. More on this comparison can be found in Chapter 5.

4.2.3. Feature vector

The following features where used as input for the clustering algorithm. Since these features are expressed in different metrics (e.g. balance vs age), the distance metriic can be deceived. By normalizing the features we can solve this problem. For normalization we used the z-score which is calculated with the method mentioned in Section 3.4.1.

Input	Description
age	Age of the client, considered because we thought similar age groups might show similar behaviour
average_balance	Average balance of the account. Clients with different average balances could show different behavior
avg_transactions_per_month	We think that a difference in activity on accounts can have implications on their spending behavior.
avg_transactions_amount	We think that a difference in transaction size on accounts can have implications on their spending behavior. Big spenders versus careful spenders for example.

Considered Inputs

The database contained other values that seemed interesting and were considered. The main three of these were geolocations of clients, the marital state of clients and the nationality. However both of these properties were not included in the feature vector in the end because they appeared to be sparse.

The database contains a house pricing index, this has potential for grouping people based on their locations and the house prices in this location. However since we do not have enough background knowledge about how these type of indexes work, we decided to not take this into account yet. But it is a possibility for future work.

Clients also have a value for their marital state which has potential for clustering but it was a missing value for the majority of clients.

The database also contains the nationality of clients but we decided to not use this because on the one hand we were not sure if nationality would really influence spending behavior and on the other hand because only about 12% of clients had a nationality different from 'NL' and out of those about 90% had an unknown nationality.

4.2.4. Final Decision Based on Results

For interpretation of these results, we used a method called Principal Component Analysis to transform higher dimensional data into a two-dimensional plot [2]. When interpreting these results (see Section 10.1), no distinct clusters could be found in the dataset. This can occur, because the features as input are not selected correctly, or because no explicit clusters exist inside the dataset. We did not have the time to test other feature selections and the current clustering did not add any behavioral distinction, therefore we decided to leave this out of the main regression model. In the section concerning future work, a further investigation on the addition of clustering is mentioned. To alleviate the problem of not having enough data available to train on for a single payment account, one model will be trained on the data of randomly selected payment accounts. Thus a general model will be trained for the prediction of all payment accounts.

5 Machine learning libraries

There are various libraries available to implement machine learning methods. At the start of this project the client showed interest in implementing our machine learning algorithms using TensorFlow. After research, the Scikit-learn library for Python seemed to be a more suitable option for this project. In order to highlight the differences between these two libraries, K-means clustering (Section 4.2.1) was implemented with Scikit-learn and compared with an implementation that was made in Tensorflow. In this way, we will give a more comprehensive analysis of the differences between these two libraries. The analysis includes measurable qualities, such as runtime and memory requirements, but also less quantifiable properties such as ease of use. Near the end of this chapter we will discuss these results in the context of the desired use-cases of both libraries. In the last section we will give our recommendations regarding the two libraries.

5.1. Clustering Implementation

Clustering was performed on approximately 440.000 payment accounts. The feature vectors used for these payments account are as described in Section 4.2.1.

5.1.1. Tensorflow

TensorFlow takes a more low-level approach to machine learning and requires that the user implements a K-means algorithm. Our implementation took 24 lines of code. The foundation of TensorFlow lies in the DataFlow graph. A data flow graph is a directed graph in which the the nodes represent mathematical operations or endpoints for data. The edges in this graph represent dynamically sized arrays of multiple dimensions. These arrays are often referred to as tensors. This is written in Python making it rather memory inefficient.

5.1.2. Scikit-learn

Scikit-learn already has an implementation for K-means. This allows a user to implement their own K-means script in just two lines of code if the data for the algorithm is already prepared. This library was written in Cython, which is a superset of Python that translates Python into C code.[20] This is what causes the Scikit-learn implementation to be more memory efficient than our TensorFlow implementation.

5.2. Test environment

Both solutions use an approximation of the K-Means clustering problem known as Lloyd's Algorithm. Therefore the results are expected to be similar. In order to compare the runtime and memory requirements of both implementations each implementation was executed 10 times per value of K. The value of K was varied between 3 and 10, and finally a value of 20 was tested as well to see how the implementations reacted to a higher K value. We ran them on the client's server through a Docker installation. After each run the kernel was shut down in order to prevent measurement errors.

Table 5.1: Overview of Runtime & Space Complexity for each K for Scikit-learn(SK) and TensorFlow(TF), averaged over 10 runs

K	Runtime SK	Runtime TF	Memory Required SK	Memory Required TF
3	6.97s	10.99s	223.6MB	403.6MB
4	14.79s	13.83s	221.5MB	440.2MB
5	8.75s	12.81s	214.0 MB	462.3MB
6	13.37s	18.61s	214.8MB	447.1MB
7	20.76s	23.81s	214.8MB	468.5MB
8	20.16s	25.64s	214.3MB	489.3MB
9	26.33s	32.99s	215.1MB	527MB
10	31.05s	30.66s	215.1MB	526.1MB
20	88.34s	96.0s	214.4MB	727.4MB

Both implementations of the K-Means algorithm were executed in a Python 3 Notebook in a Jupyter environment. All time measurements concern only the runtime of our K-Means algorithm itself and no pre- or postprocessing.

The measurements containing memory usage were done using functions from the resource package in Python. The input of these algorithms were approximately 440,000 four-dimensional vectors.

5.3. Runtime

Although both solutions implement the same algorithm, the Scikit-learn implementation seems to be slightly faster than our TensorFlow solution. There is a clear-cut explanation for this. Computations in TensorFlow rely on the data flow graph model. For these computations TensorFlow uses Python, whereas Scikit-learn uses Cython (not to be confused CPython, the standard interpreter) [20]. Cython is a superset of Python that translates Python code to C for increased performance.

A great advantage here for the Scikit-learn implementation is that speed-up can easily be obtained by altering the parameters you pass when initializing the object. This includes pre-computing distances of the points and possibilities to parallelize your code. When using TensorFlow, these options are not available by default and should be implemented by the programmer himself. For our analysis we have disabled these options, as this would lead to a biased comparison. The results can be found in table 8.1.

In general, a larger value for K leads to an increase in runtime. The time complexity of Lloyd's algorithm is often given as $O(nkdi)$, where n represents the number of data points, k is the number of clusters, d is the number of dimensions our points and i is the number of iterations required for convergence [21]. Therefore the occurrence of this trend was expected. There seems to be a strange jump for our Scikit-learn solution whenever the number of clusters is equal to the number of dimensions of our datapoints. We were not able to determine the exact cause of this.

5.4. Memory Requirements

As both solutions implement the same algorithm, the differences in memory usage can be attributed to differences between standard Python and Cython.

TensorFlow is again at a disadvantage here, because Python is dynamically typed and does not know primitives. Therefore it requires more memory than a similar solution in Scikit-learn, as this is again optimized by using Cython. Dynamic typing in Python requires a lot of memory generally [5]. This can be mitigated by specifying a desired datatype beforehand. Although this improved the performance of our TensorFlow solution, the Scikit implementation still performed better.

The memory requirements of both solutions can also be found in table 8.1. We do have to take into account that the functions we used for these measurements measure peak-memory usage of our implementation. The memory requirements of the Scikit-learn implementation seem to be unaffected by increasing the value of K. Our TensorFlow solution uses more memory in all cases and memory requirements increase slightly when K is increased.

5.5. Ease of Use

Ease of use is a less tangible property for software than matters like runtime or memory requirements. Ease of use depends on a number of different properties such as lines of code required, difficulty of tweaking this code and the likelihood that a programmer will create buggy code.

TensorFlow has a more low-level approach than scikit-learn. The clustering itself requires two lines of code with scikit-learn, the first is a call to the K-Means object and the second assigns labels to the clusters. In our TensorFlow implementation we required 24 lines to obtain the same functionality. Optimizing the performance of our TensorFlow solution would require more code to be written, whereas this is already done in the Scikit-learn implementation. Having pre-made algorithms ready to use leaves less room for a programmer to introduce bugs into the program.

5.6. Suitability

From the results described above, we conclude that Scikit-learn is better suited to our needs than TensorFlow. TensorFlow is a good option for designing your own machine learning methods or tweaking known algorithms. It requires more knowledge on how the algorithm works since every aspect needs to be implemented. Scikit-learn on the other hand only requires knowledge about how the algorithm behaves with varying inputs and parameters.

When it is necessary to quickly research known algorithms on new problems, Scikit-learn is more a suited library. Since it offers pre-made implementations of different machine learning techniques it allows for quickly researching of different techniques on the problem at hand.

For this project, the speed, reliability and ease of use of Scikit-learn are what made us choose this library over Tensorflow.

5.7. Recommendations

The datasets required for creating accurate predictions are too large for our desktops to handle. When we were confronted with the fact that our own computers could no longer handle the calculations we required, we were advised to use Jupyter which runs ons one of the clients servers. Unfortunately, this did not provide the increase in computing power that we required. When we wanted to use a sample from our dataset, approximately 5.0GB in size this environment ran out of memory when reading the file and was thus unable to process it.

As seen above, TensorFlow already requires a substantial amount of memory. In the case where the client would like to start building models using TensorFlow, they should make sure to allocate sufficient resources for these models. We were not able to gain access to systems that had sufficient computing power ourselves.

The most important factor that should be considered is the low-level approach of TensorFlow. Current methods used by the client all present pre-made algorithms. TensorFlow offers no such thing and therefore requires the programmer to implement all algorithms from scratch. A possible solution for this is Scikit-Flow or Skflow. This library is a hybrid form between Sckikit-learn and TensorFlow. This offers a set of pre-made algorithms to the user, while preserving TensorFlow's functionality.

6 Balance prediction

6.1. Time Series

A time series is defined as a sequence of data sampled over continuous intervals of time with equal spacing between the data points. Time series analysis is the collection of methods that can be used for retrieving valuable information from the data. One extremely important application of time series analysis is time series forecasting. Using the characteristics extracted from observed values, a model can be created that predicts future values.
The point in time from which the prediction is made is often referred to as the forecast origin [22]. The length of the interval between the forecast origin and the point in time we make a prediction for is called the forecast horizon [22]. In order to make accurate predictions it is neccessary to evaluate your forecasts. This is done by calculating the error of the forecast.

6.1.1. Bank Balance as Time Series

The main goal of our application is to create predictions of a customers bank balance. Based on a customer's transaction history, we should be able to construct a model that could predict future values of the customer's bank balance.

Given the definition of time series in the previous section, we can model the bank balance of a single customer as a time series. Using the available transaction data, we can construct a sequence of data points that has the four critical properties for time-series data.

1. We can create a sequence of data points over a continous period of time.
2. These points are constructed from successive measurements of the customer's bank balance.
3. The time period separating measurements is equal for each consecutive measurement.
4. Each period of time has at most one data point.

A lot of the applications of time series analysis involve financial markets, which have some important differences. A stock price for instance is much more volatile than a payment account. The prior changes every second, whereas the latter only has three to four mutations in a day. This means that volatility is much less of an issue for predicting bank balances than it is for stock prices.

In our case, we are also specifically interested in what time intervals give us the most accurate predictions. This depends on the size of the forecast horizon that we want to use. If we want to do a 1-step forecast, we would have to look at one month intervals. If we are interested in the accuracy of l-step forecasts (for $l = \{28, 30, 31\}$) we should take our intervals to be one day. The difference in accuracy between 1-step and l-step forecasts depends on the model we will use for forecasting. During the project we had to try different models with different forecasting horizons.

6.2. Prediction intervals

We have decided to predict one month into the future. This decision is based on a few factors. Firstly, most people have a financial cycle of one month, e.g. rent, salary and groceries. During distinct days in a cycle there can be huge fluctuations, whilst the difference between two cycles can still be very small. Therefore we see more use in a prediction made for the next financial cycle than a prediction for a moment inside the same cycle.
By predicting one day ahead, and repeating this for the amount of days in a month we can still achieve to predict one month ahead. However, this prediction will be a lot less accurate than the prediction made based on month values.
To get a good view of the balance over one month, for every day in the month we measured the balance and eventually took the mean of this balance over all days of that month as the final average. This way we can get a good idea of the overall balance this month, so we do not just "pick" a moment at random which can be deceiving.

6.3. Errors

To compare our regression results we have picked three methods for the error calculation.

- **Mean Squared Error**. This measure uses the difference between the prediction $\hat{y}_i$, and the actual data y_i. Here $(\hat{y}_i - y_i)$ is the *error*. We have no knowledge about the value of the error, this can either be positive or negative, therefore we square every error and receive the *squared error* $(\hat{y}_i - y_i)^2$. Since we can have multiple data points to calculate the error over (we use time series so we can calculate te error over a longer period of time), we want to know something about the overall error. This over all error can be calculated by taking the mean of all errors in the made predictions. The final measure we get is thus the *Mean Squared Error*:
$$MSE = \frac{1}{n} \sum_{n-1}^{i=0} (\hat{y}_i - y_i)^2 \tag{6.1}$$

- **Root Mean Squared Error** The RMSE is an extension of the MSE. Squaring the errors can be deceiving, big errors become exponentially large whilst errors below one become exponentially small. This draws errors further apart. To overcome this the RMSE can be used. This measure takes the root of the MSE which is not exactly the same as the absolute error, but brings the errors closer together. This gives us:
$$RMSE = \sqrt{\frac{1}{n} \sum_{n-1}^{i=0} (\hat{y}_i - y_i)^2} \tag{6.2}$$

- **Mean Absolute Percentage Error** The MAPE describes the error as a percentage. Firstly the difference between the predicted and real value is calculated, secondly this error is divided by the real value to scale the error. When multiple predictions are made, the average is calculated.
$$MAPE = \frac{1}{n} \sum_{n-1}^{i=0} |\frac{\hat{y}_i - y_i}{y_i}| * 100 \tag{6.3}$$

6.3.1. Error normalization

The input of our models is normalized using the z-scores described in Section 3.4.1. For this reason, the predicted and real values are also expressed in z-scores. The difference between the real and predicted value can be written as:
$$E = \frac{\hat{y} - \mu}{\sigma} - \frac{y - \mu}{\sigma} \tag{6.4}$$
This can be rewritten as:
$$E = \frac{\hat{y} - y}{\sigma} \tag{6.5}$$

The difference between the real and predicted value is expressed in terms of standard deviations. By normalizing our prediction like this we use the fluctuation of the specific account as as scaler. The real value of the predictions is thus not used, but instead we scale this with the fluctuation measured on the account. By doing this, absolute differences between real and predicted values on accounts with low fluctuations lead to higher errors than the same absolute differences on accounts with bigger fluctuations.

7
Non machine learning models

In this chapter, we will describe the non machine learning models used for the regression analysis.

7.1. Average baseline and L-1

To get an idea of how well our models work, we have picked a baseline model to compare our more complex models with. For this baseline, we picked the simple average model.

7.1.1. Simple Average Model

As a simple statistical model, we took an average over the previous data to predict the next data points. This algorithm is trivial and is thus used as a baseline algorithm.(results can be found in Chapter 10). The algorithm can be formulated as follows:
When we have a certain amount of data points:

$$X = (x(0), x(1), ..., x(n)) \tag{7.1}$$

We predict the next number $x(n+1)$ in the sequence by:

$$x(n+1) = \frac{1}{n}\sum_{i=1}^{n} x(i) \tag{7.2}$$

7.1.2. (l-1) Prediction

As a second simple model, we predicted the next data point by simply taking the previous value as a prediction for the next one.

7.2. Non machine learning

We do not want to have a narrow vision towards the techniques we considered to be suitable for this project. Because we already have a few machine learning techniques in, we looked into other techniques suitable for time series predictions. We found that statistical models can be used for predicting time series in the financial context as well. Furthermore, we want to compare these models with the machine learning models mentioned above and see which performd better on the data we have at our disposal.

7.2.1. Grey Model

Grey system theory is an interdisciplinary scientific field and was first introduced in 1980s by Deng [12]. Since then, it has evolved and has been applied to many real world problems. The application fields of the Grey System involve agriculture, economy, meteorology, medicine, history, geography and more [11]. The Grey System model uses a certain frame of the last data points. The idea is thus that you predict a future data point by only using a few older data points instead of all the historical data that is available. As input the algorithm needs a sequence of positive numbers that represent the data points in time. $GM(n,m)$ denotes a Grey model, where n is the order of the differential equation and m is the number of variables. Here a description is given for a $GM(1,1)$ model. Because of its efficiency in calculation time, this is also the model that is often used in real world situations [12]. For example:

$$X^{(0)} = (820, 840, 835, 850, 890) \tag{7.3}$$

Here you can see that a certain sequence is given. The window size in this example is 5. Which means that only the last 5 data points are used to predict the next data point in the sequence. In general, the prediction model works as follows:

First we denote a sequence of non-negative data as:

$$X^{(0)} = (x^{(0)}(1), x^{(0)}(2), ..., x^{(0)}(n)) \tag{7.4}$$

Then an accumulator function (AGO) is used to smooth the primitive sequence. This function sets a certain x on index k on the sum (accumulation) of all previous numbers. (In the example sequence this results in $X^{(1)} = (820, 1660, 2495, 3345, 4235)$)

$$X^{(1)} = (x^{(1)}(1), x^{(1)}(2), ..., x^{(1)}(n)) \tag{7.5}$$

Where

$$x^{(0)}(1) = x^{(1)}(1) \tag{7.6}$$

and

$$x^{(1)}(k) = \sum_{i=1}^{k} x^{(0)}(i) \qquad \text{for } k = 1, 2, 3, ..., n \tag{7.7}$$

The next step is to generate the mean sequence of $X^{(1)}$:

$$Z^{(1)} = (z^{(1)}(1), z^{(1)}(2), ..., z^{(1)}(n)) \tag{7.8}$$

where $z^{(1)}(k)$ is the mean of the data point on position k-1 and the one on position k:

$$z^{(1)}(k) = (0.5x^{(1)}(k) + 0.5x^{(1)}(k-1)) \qquad \text{for } k = 2, 3, ..., n \tag{7.9}$$

Now we constructed the input for the model. The model itself, $GM(1,1)$, is constructed by a first order differential equation for $x^{(1)}(k)$. It is stated in literature, that financial time series can be modeled accurately with a differential equation [11].

$$\frac{dx^{(1)}(k)}{dk} + ax^{(1)}(k) = b \tag{7.10}$$

The solution of this equation can be written as:

$$\hat{x}^{(1)}(k+1) = [x^{(1)} - \frac{b}{a}]e^{-ak} + \frac{b}{a} \tag{7.11}$$

where $\hat{x}^{(1)}(k+1)$ is the prediction x at time k+1, thus the prediction of the next data point. The terms a (called developing coefficient) and b (called grey input) can be found by the OLS (ordinary least squares) method:

$$[a, b]^T = (B^T B)^{-1} B^T Y \tag{7.12}$$

in which:

$$Y = [x^{(0)}(2), x^{(0)}(2), ..., x^{(0)}(n)]^T \tag{7.13}$$

and

$$B = \begin{bmatrix} -z^{(1)}(2) & 1 \\ -z^{(1)}(3) & 1 \\ \vdots & \vdots \\ -z^{(1)}(n) & 1 \end{bmatrix} \tag{7.14}$$

If you predict the value $\hat{x}^{(1)}(k+1)$, you want to calculate the primitive value that is linked to this accumulated value. Therefore the inverse AGO (IAGO) is performed and the predicted primitive value can be calculated by the following formula:

$$\hat{x}^{(0)}(k+1) = [x^{(1)} - \frac{b}{a}]e^{-ak}(1-e^{2}) \tag{7.15}$$

Suitability

The first advantage of this model is that it can predict future data point with decent to good performance depending on the dataset, while only a few data points are used as inputs. In this [17] study, researchers could predict future suicide rates in India based on 5 previous years (5 data points) with a$GM(1,1)$ with 2-1% errors. The error measure that has been used is the mean relative percentage error (MRPE) [17]. Secondly, the Grey Model has no parameters to be set, except for the window size. In literature it can be seen that different sizes of windows have only minor changes in the prediction accuracy [12]. Furthermore the $GM(1,1)$ has high computational efficiency when compared to machine learning methods, because no iterative training needs to be done in order to achieve a prediction. A drawback of this system is that it is not easy to change its input-output behavior, since it only takes numbers as inputs. SVMs and NNs are more flexible regarding input-output. Finally the Grey models work only on positive data, bank balances are not only positive values. At first we had hoped to solve this by for example shifting the bank balances so that all balances fall within positive values. However since this proved troublesome, we decided to use the ordinary least squares model instead, this is described in the next section.

7.2.2. Ordinary Least Squares

The Ordinary Least Squares method (OLS [9]) is the foundation of the grey model. However, the grey model is extended with some features that make it only able to handle positive datapoints. Even after shifting the datapoints or normalizing them, the grey model would not be able to give us the expected results. Therefore, we have decided to implement the simple OLS method instead. For the OLS method, the inputs are averages of previous months and the output will be the average for the month we want to predict. The amount of input values can be denoted as the window size, in our case this was set to 5. This is stated in literature [12] as the best size since higher sizes would not give enough increase in accuracy compared to the extra runtime needed. We wanted to test this assumption for ourselves and plotted the result in figure 7.1. Because we tested it for ourselves and the results match to what is seen in literature, we took 5 as the standard window size to run the model. For other kinds of regression the grey model might give better results since it is a more extensive kind of regression. For our project it was less suitable.

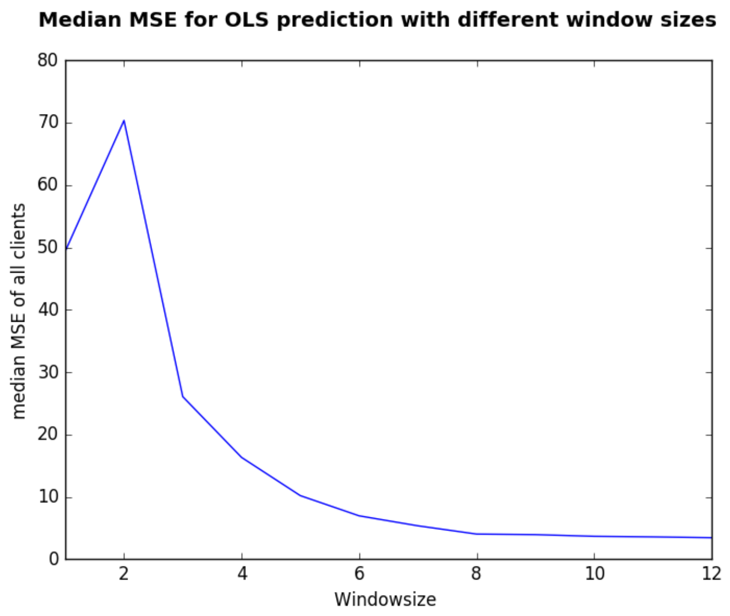

Figure 7.1: This graph shows the median MSE for all client when using different window sizes for the OLS input vector. On the x-axis you can see the varying window size and on the y-axis you can see the median of the MSE for all accounts predicted with and OLS of this window size. For example, having a window size of 2 means that only the previous 2 months are used as input. On the graph you can see that the error drops drastically when using higher window size, but from 6 onwards the error does not change that much.

8 Machine learning model

The problem at hand can be seen as a regression problem. Given the data of an account in the previous months, give an estimate of the change in balance for the next month. So we apply machine learning techniques that are capable of regression. We opted to try a recurrent neural network and a support vector machine, however we only managed to implement the support vector machine. More about this can be found in Section 11.3.

8.1. Support Vector Machines

Support vector machines (SVM) in their most basic form are used for classification [7], but they can also be used for regression and time series predictions [19]. The big difference between SVM and traditional neural networks (NN) is that NNs utilize empirical risk minimization principles and SVMs use structural risk minimization [10]. This means that NNs try to minimize the deviation between the output and the training data while SVMs try to balance fitting to the data with model complexity. If this balance is chosen correctly, this counters over-fitting.

8.1.1. Support Vector Machine Regression

When performing support vector machine regression, we try to find a linear function that fits to the data. However not all data fits to a linear function. To solve this, a mapping ϕ, is made from the input space to a higher dimensional space, called the feature space. In this higher dimensional feature space a linear regression then corresponds to a nonlinear regression in the input space [19]. If b is a bias, and w the weight vector then the output of this system is given by:

$$f(x) = (w \cdot \phi(x)) + b \tag{8.1}$$

To find the optimal linear function, f, that fits to the data we want a function, R, that determines how good we deem a certain function f to be. As mentioned SVMs try to both fit the data and keep the model complexity low, to achieve this we can minimize a formula that contains two aspects. On the one hand the formula has a term to penalize deviation from the training data and on the other hand there is a term which penalizes model complexity.

$$R(f) = \sum_{i=1}^{l} C(f(x_i), y_i) + \lambda ||w||^2 \tag{8.2}$$

Where C is a function which determines the deviation from result $f(x_i)$ with its expected value y_i (more on these cost functions later), $||w||^2$ is the square of the norm of the weight vector and penalizes the model complexity. Finally λ is a parameter which determines the weight that should be given to model complexity and in that way determines the trade-off between model complexity and deviation [19]. It can be shown that the function $f(x)$ from equation 8.1 that minimizes this expression can be given in the following form [18]:

$$f(x) = \sum_{i=1}^{N} ((\alpha_i^* - \alpha_i)(\phi(x_i) \cdot \phi(x))) + b \tag{8.3}$$

Where N is the size of the training data set and $\alpha_i^*, \alpha_i \geq 0$. After minimization only some of the coefficients $(a_i^* - a_i)$ will be nonzero due to the nature of this problem. The data points corresponding to these coefficients are called the support vectors [18]. Computing the dot product $\phi(x_i) \cdot \phi(x)$ for every data point x_i with the new input x would take a long time since due to the mapping ϕ to a higher dimensional space this can be computationally expensive. Luckily this is not necessary we can rewrite $f(x)$ one last time:

$$f(x) = \sum_{i=1}^{N} ((\alpha_i^* - \alpha_i) K(x_i, x)) + b \tag{8.4}$$

Where $K(x_i, x)$ is called the kernel function. The kernel function satisfies $K(x_i, x) = \phi(x_i) \cdot \phi(x)$ so it calculates the dot product of x_i and x in the higher dimensional feature space. For a lot of choice of ϕ this kernel function is known and simple [18].

8.1.2. Kernel Functions

Many different kernel functions are possible. To be able to use a function K as a kernel function it needs to be a positive definite function and statisfy Mercer's Conditions [8],

$$K(x, x_i) = \sum_{m}^{\infty} a_m \phi_m(x) \phi_m(x_i), \quad a_m \geq 0, \tag{8.5}$$

$$\int\int K(x, x_i) g(x) g(x_i) dx dx_i \geq 0, \quad g \in L_2 \tag{8.6}$$

[8] gives a nice overview of commonly used kernels. When we have two or more possible kernel functions, we can also combine these into a more complex kernel function simply by summing them [8]. We can also create a multidimensional kernel by calculating the tensor product of multiple kernels [8].

8.1.3. Loss Functions

In equation 8.2 we introduced a loss function C which was needed to determine the deviation from result $f(x_i)$ with its expected value y_i. This function needs to measure the distance between the obtained solution and the desired one and can also be defined in many ways. [8] gives an overview of a few commonly used loss functions for SVM regression.

8.1.4. Suitability

We can use support vector machine regression for time series prediction. According to Takens embedding theorem predicting the next value should be possible by using m previous values. The value m is called the embedding dimension [18]. So using these m previous values as input for a regression SVM to predict future values should be possible. Support vector machines seem suitable to solve our problem, because they are resilient against over-fitting and noise [10]. Both of these aspects are important in our problem since over-fitting can happen easily, due to often fairly limited data from accounts. Support vector machines have also been used in research before to predict financial time series. In [10] SVMs were for example used to predict the direction of the stock price index, they compared them to a back-propagation network (BPN) and a case-based reasoning (CBR) and found that SVMs work better than both BPN and CBR. In [18] SVMs were applied to various chaotic time series and compared to polynomial approximation, rational approximation, local polynomial approximation, Radial Basis Functions with multiquadrics as basis function and Neural Networks. They found that their SVM was only outperformed by one technique (rational approximation) in one out of the six tested instances.

8.1.5. Implementation

Pre-made implementations of SVMs for us to experiment with exist. LIBSVM [6] is a well established library for SVMs. LIBSVM provides source code in both Java and C and it is also included in Scikit-learn. We used the Scikit-learn implementation to test regression with SVMs.

features	median MSE per coid	median SE
normalized average	2.550612131112	0.944186640142
difference	2.294911	0.562655

Table 8.1: Results for a SVM with a rbf kernel function and parameters C=0.2, epsilon=0.1, gamma='auto' (1/n with n = number of features). Comparing the results for predicting the average bank balance or the difference between average bank balances.

features / nMonths	month_name_value, avg_balance_difference		avg_balance_difference	
	median MSE per coid	median SE	median MSE per coid	median SE
1	2.256563	0.5490103	2.294911	0.562655
2	2.358577	0.590181	2.365729	0.590635
3	2.449823	0.586173	2.448636	0.590052
4	2.423830	0.588169	2.441525	0.597540
5	2.411315	0.586324	2.450461	0.594367
6	2.417386	0.599997	2.454182	0.601695
7	2.423333	0.582792	2.455026	0.587534
8	2.438030	0.596455	2.439002	0.597910
9	2.430606	0.590873	2.448806	0.591172
10	2.417942	0.596481	2.441453	0.599460
11	2.426432	0.592175	2.431141	0.592546
12	2.344148	0.580216	2.350842	0.583081

Table 8.2: Results for a SVM with a rbf kernel function and parameters C=0.2, epsilon=0.1, gamma='auto' (1/n with n = number of features). The results compare the influence of the month_name_value parameter and the number of months used as input.

For the SVM various decisions had to be made, how they were made is described in the following subsections. Presented results are from SVMs that were trained on the data of 10000 months from payment accounts adhering to the restrictions made in Chapter 3. The validation is done for a sample of 1000 payment accounts on each month after 01/01/2015 until 01/04/2016. All data is normalized according to the method described in Chapter 3. The selections are compared based on two statistics. The median squared error of all predictions and the median of the mean squared error per payment account. All errors are calculated on the normalized average balance. The first gives insight into how good a prediction in general is and the second gives insight into how well the SVM predicts certain payment accounts. For both the median is calculated, because the mean is more sensitive to outliers. These outliers are months in which the average balance of the account is unusually high. These outliers will be discussed together with the results in Chapter 10.

Loss Function

The Support Vector Machine for regression implemented in Scikit-learn uses an epsilon loss function [1] [8].

Feature Selection

The first decision that had to be made for the feature vector is whether to try and predict the average balance of the upcoming month with the average balance of the previous months as input or whether to predict the difference in the average balance between the previous month and the upcoming month, with the differences of the previous months as input. For this two SVMs were trained. One with as input the past twelve average balances and one with as input the past twelve differences. The SVMs used a radial basis kernel function (rbf). The results of this can be found in Table 8.1.

From this we inferred that the differences seemed more promising, so we went ahead with those. Further decisions for the feature vector remained. More features needed to be selected and the number of months for input needed to be decided. For this a few tests were executed. First this was tested with just the differences as input, while varying the number of months this input was given for. After that another parameter, the month_name_value, was added. This is only added for the month that needs to be predicted. These are calculated in the same way the average balance per month is calculated in Section 3.1.2.

features	median MSE per coid	median SE
month_name_value, avg_balance_diff (1st and 12th month)	2.22692382723	0.543940351876
month_name_value, avg_balance_diff (1st and 12th month), inflow (1st and 12th month), outflow (1st and 12th months), inflow_nr (1st and 12th month), outflow_nr (1st and 12th month)	2.03058373635	0.536919592243

Table 8.3: Results for a SVM with a rbf kernel function and parameters C=0.2, epsilon=0.1, gamma='auto' (1/n with n = number of features). The results compare the influence of adding information about incoming and outgoing transactions.

Feature Name	Description
month_name_value	This is a value dependent on the month to predict. How it is calcualted exactly, is explained in Chapter 3
month_diff_min_1	The difference in the average balance of the previous month
month_diff_min_12	The difference in the average balance of one year ago
inflow_min_1	The average amount of an incoming transaction in the previous month
inflow_min_12	The average amount of an incoming transaction of one year ago
inflow_nr_min_1	The number of incoming transactions in the previous month
inflow_nr_min_12	The number of incoming transactions of one year ago
outflow_min_1	The average amount of an outgoing transaction of the previous month
outflow_min_12	The average amount of an outgoing transaction of one year ago
outflow_nr_min_1	The number of outgoing transactions in the previous month
outflow_nr_min_12	The number of outgoing transactions of one year ago

Table 8.4: Table describing the final feature vector used for SVM regression.

From looking at the first row of Table 8.2, it can be seen that the SVM performs best with the input of only 1 month and that the month name value parameter does help a bit. However, it can also be seen from the last two rows that the error goes down again once the 12th difference is added. So we decided to cut out the intermediate values and take the first and 12th month as input. Finally, we tried adding information about incoming and outgoing transactions for these months. The results are presented in Table 8.3.

Adding the data about incoming and outgoing transactions for different months does improve both measures slightly, so these were taken into account in the final feature vector. This final feature vector is described in Table 8.4.

Parameter Tuning

With this feature vector, we tried tuning the parameters of the support vector machine. This means tuning C, epsilon and gamma (gamma is from the radial basis function). First epsilon was set at 0.1 and different values for C and gamma were tested. For C, we went from 0.01 to 100 with each step increasing with a factor of 10. For gamma we did the same, but for 10^{-7} to 100. This gave an optimal median squared error per record of 0.5187 and an optimal mse per account of 2.0759 for C = 1 and gamma = 0.01. To verify the default choice for epsilon the same values that were tested for gamma were also testes for epsilon (while keeping C = 0.1 and gamma = 0.01). It turned out that the optimal value for epsilon with these parameters was the 0.1 that was used as a default.

Kernel

First to select features and do some parameter training an rbf kernel was selected, because this is the default kernel in Scikit-learn. Scikit-learn also offers a linear, polynomial and sigmoid kernel. After the feature selection and parameter tuning, we wanted to verify if the choice of kernel was suitable for the chosen features. The results from testing with different is presented in Table 8.5.

The mean SE column of this table shows that the rbf kernel did indeed perform best, however since we optimized the parameters with this kernel this is not surprising. The error of the linear kernel comes quite close

Kernel	median MSE per coid	mean SE
linear	2.21522466721	0.577426952614
polynomial	2.51036697357	0.612505440123
sigmoid	24.0424367063	7.96724012504
radial basis function	2.07597867799	0.518765938346

Table 8.5: Comparison of SVM results using different kernels. The final feature vector as described in Table 8.4 was used.

and the polynomial kernel is not too far off either. One interesting area of further research, would be to test optimization of features and parameters with a linear or polynomial kernel. More on interesting areas of future work can be found in Section 11.3. The sigmoid kernel function gave a very high error, however we did not have enough time to investigate why this is the case.

To summarize, the final SVM used the feature vector described in Section 8.1.5. It uses a radial basis kernel function and the parameters of the SVM are C = 0.1, epsilon = 0.1 and gamma = 0.01. More detailed results of this final SVM can be found in Chapter 10.

9 Design

Initially, we wanted to implement multiple different models and compare them. Later on we decided to stick with a single model, because of planning issues. We still see great value in implementing different solutions and comparing them, therefore we decided to build an overall framework that would enable this.

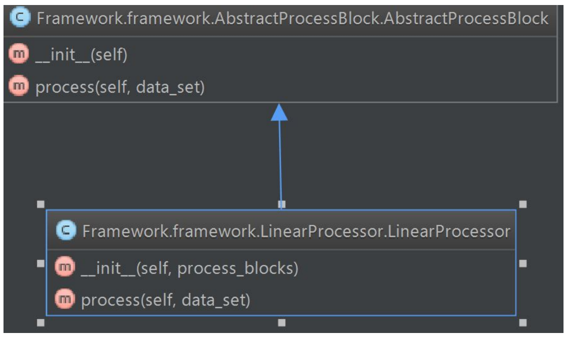
Framework.framework.AbstractProcessBlock.AbstractProcessBlock
__init__(self)
process(self, data_set)
Framework.framework.LinearProcessor.LinearProcessor
__init__(self, process_blocks)
process(self, data_set)

Figure 9.1: UML of AbstractProcessBlock.

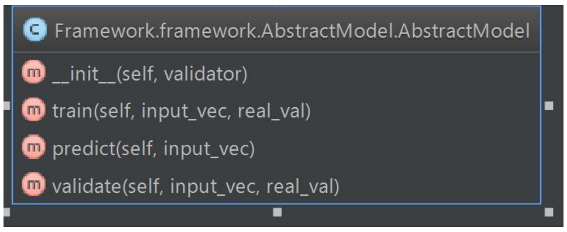
Framework.framework.AbstractModel.AbstractModel
__init__(self, validator)
train(self, input_vec, real_val)
predict(self, input_vec)
validate(self, input_vec, real_val)

Figure 9.2: UML of AbstractModel

Two main classes of the framework are the AbstractModel and the AbstractProcessBlock. The AbstractModel can be extended to implement a specific model. The AbstractProcessBlock are modules that can be used after each other to process data. This makes it easy to pre-process data (like scaling and filtering), before putting it into the model. A UML diagram of this part of the framework is given in Figure 9.1 and 9.2.

The AbstractModel relies on two more classes, the AbstractValidator and the AbstractError. An instance of the AbstractValidator represents a validation method. An instance of the AbstractError represents an error measure, for example the Mean Squared Error. This structure allows us to easily combine different validation methods with various error measures. This would be very convenient when the programmer wants to run different combinations on the same model in succession.

9.1. AbstractModel

The AbstractModel class defines an interface for implementing specific models. By doing this, we can easily reuse code with different models. A model needs to be able to do two things, it needs to be able to train itself on training data and it needs to be able to make predictions. It is also important, that we can validate these predictions. When instantiating an instance of AbstractModel, the programmer will have to pass an instance of the AbstractValidator as a parameter. Since models can use the same validation techniques, this is not model specific and therefore there is a separate class for this which will be discussed later.

9.2. AbstractProcessBlock

The abstract process block defines an interface for implementing processing blocks. These blocks are small modules which define a data operation. We can think for example of standardizing a column in a table, filtering out certain rows or creating new columns through operations on the existing columns.

These blocks can then be executed one after each other to create the input required by a certain model. This structure allows for easy reuse of these data operations and gives a lot of flexibility. If we want to test our model with a different normalization for example, we just exchange the normalization block for another one. Furthermore this structure is flexible, since it does not put any constraints on the data processing. In this structure a user can define both a very complicated pre-processor with many ProcessBlocks or a very simple one that for example only does a simple normalization.

9.2.1. LinearProcessor

One special instance of the AbstractProcessBlock class is the linear processor. This processor takes a list of ProcessBlocks as input in its constructor and executes them one after each other in the process() method. Since it is an instance of AbstractProcessBlocks, one of the input ProcessBlocks could also be a LinearProcessor. This allows us to bundle ProcessBlocks together in a LinearProcessor and reuse them in another LinearProcessor.

9.3. AbstractValidator

The AbstractValidator is used by the AbstractModel class to determine how to validate the model. In our opinion this should be defined in a separate class, because validation methods should not be model-dependent. When instantiating an instance of the AbstractValidator, the programmer needs to pass an AbstractError as a parameter. In its simplest form, a validator simply calls the error measure that has been assigned to it. In more complex implementations, one can choose to create an instance of the AbstractValidator that represents k-fold validation. The current desgin allows for combining different error measures with various validation methods.

9.4. AbstractError

The AbstractError is the simplest part at the core of our framework. The AbstractError defines an interface for defining error measures. In essence all these instances take an array of the predicted values and an array containing the actual values. Our design allows for different error measures to be used by validators conveniently.

9.5. Testing

We tested our framework using unittest, a popular Python testing framework. We chose to use this, because it closely resembles the popular Java testing framework JUnit. Each member of our team has experience writing test in JUnit and therefore should be able to use unittest.

We created unit tests for most parts of our framework. Some of the abstract classes are not tested directly, as they cannot be directly instantiated.

We achieved a satisfactory amount of line coverage at 100%. More importantly, we managed to achieve some minor bugs that would have proven catastrophic. This includes returning incorrect data types, retrieving rows instead of columns and vice versa.

We wrote three integration tests in order to check that these classes worked together as expected. We did not uncover any new bugs through these tests.

10
Results

10.1. Clustering

In this section, the results of clustering will be presented. As stated earlier in the report, our group put two K-Means clustering implementations to the test. The results of both implementations algorithms are expected to be roughly the same since they both implement Lloyd's algorithm, which is an approximation algorithm for K-Means clustering[14]. To visualize and interpret results of a clustering algorithm operating on high dimensional data we used the Principal Component Analysis technique [2].

As expected, the clustering assignments and centroids are roughly the same for k=3 (figure 10.1). This comparison is done for k = 4, k = 5 and k = 6 as well. These plots can be seen in the appendix C in Figure C.1, C.2 and C.3. Since we use the same algorithm for both implementations and our four plots give the same results, we have decided not to further compare the implementations with other values for k.

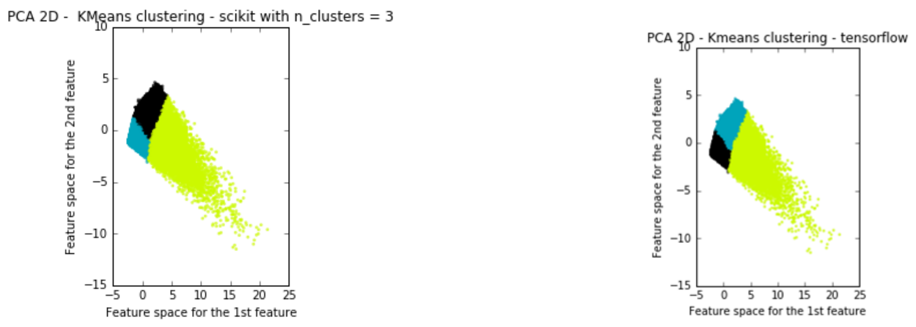

Figure 10.1: The left graph shows the PCA analysis of the clustering results on our dataset using Scikit-learn. The right graph shows the PCA analysis of the clustering results on our dataset using Tensorflow. For both graphs, the K-means algorithm is used with k = 3. Both scikit-learn and Tensorflow give the same result for k = 3.

10.2. Regression

For visualizing the results of the regression models predictions, our group used more than one error measure. For all the results in this chapter, each model is tested on a sample of 1000 accounts from the validation set we created in Chapter 3. The results are given in a histogram. This means that the results are plotted as a distribution over the error measures. We tried different representations of the results and finally one was chosen to compare the models. The next sections will describe these different representations. For the SVM model, a plot of each of the different representations is given. Later when comparing the different models, only one representation is used.

10.2.1. Feature vector and Account errors

To calculate how accurate our predictions are, we can visualize errors of single predictions For validating how well we can predict on distinct clients this measure has some limitations. To solve this, we have decided to visualize the errors in two different ways.

Per feature vector

For every distinct feature vector, one single prediction is made. If we want to see the distribution of errors per prediction, it is is useful to visualize the distribution of the feature vectors. The three different error measures for the SVM visualized per feature vector is given in Figure 10.2, 10.4 and 10.6

Per Account

For every account, multiple predictions are made, depending on the time window. By visualizing the distribution of average errors calculated per account we can, get an idea of how well a prediction for this client can be made. The three different error measures for the SVM visualized per account can be seen in figure 10.3, 10.5 and 10.7

We have decided to compare the rest of our models based on the error per feature vector. The results in figure 10.2, 10.4 and 10.6 are smoother and give a clear distribution. This is caused by the fact that the error per account is an average. A single bad prediction has an influence on the whole average and can mislead us into thinking that all predictions are bad. To make recommendations for clients for which we can make accurate predictions, the error per account can be a better estimate. To compare the results of a model, it is better to compare distinct predictions than comparing an average per account.

10.2.2. Error Measure

As stated earlier, three error measures were considered: the mean squared error (MSE), root mean squared error (RMSE) and mean absolute percentage error (MAPE).

The RMSE uses the root of the MSE, because the predictions per feature vector are no means, but just squared errors, the RMSE of those predictions can be written as:

$$RMSE = \sqrt{\frac{1}{1}\sum_{1}^{i=0}(\hat{y}_i - y_i)^2} = \sqrt{(\hat{y}_i - y_i)^2} = |\hat{y}_i - y_i| \tag{10.1}$$

For this reason we have described the RMSE for distinct feature vectors as the absolute error.

The six distributions for the SVM can be divided in:

- In Figure 10.2 and 10.3 the distribution of the absolute error per feature vector and **RMSE** per account can be seen next to each other.
- In Figure 10.4 and 10.5 the distribution of the absolute error per feature vector and **MSE** per account can be seen next to each other.
- In Figure 10.6 and 10.7 the distribution of the absolute error per feature vector and **MAPE** per account can be seen next to each other.

To compare the models, we have decided to use the RMSE for distinct feature vectors which is, as mentioned earlier, the absolute error. The error is expressed in terms of the standard deviation (more about this in Section 6.3.1). By taking the absolute error per feature (Figure 10.2), we thus visualize distribution of the the difference between the real and predicted value in standard deviations.

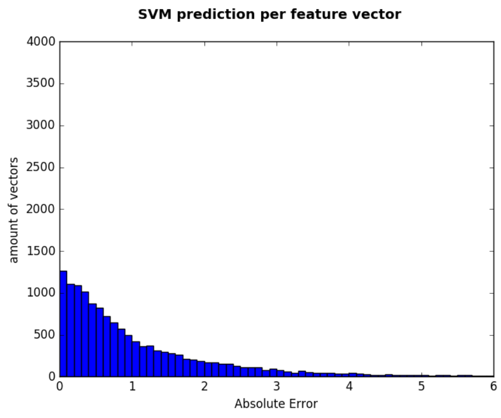

Figure 10.2: The distribution feature vectors over the absolute error for the SVM model.

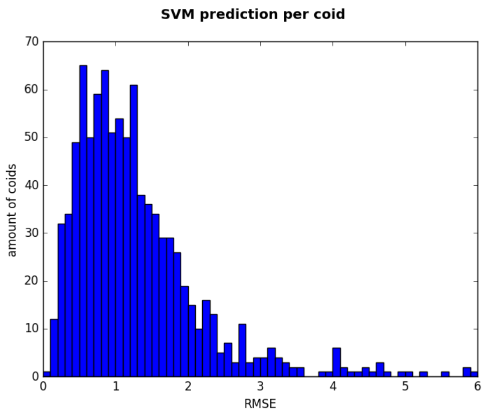

Figure 10.3: The distribution of accounts over the RMSE per account for the SVM model.

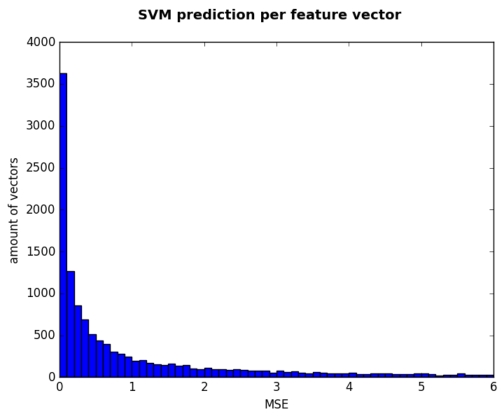

Figure 10.4: The distribution feature vectors over the MSE for the SVM model.

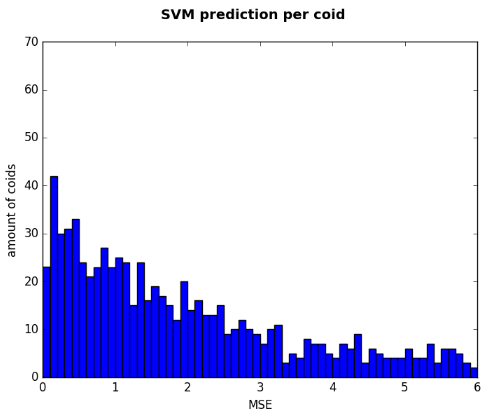

Figure 10.5: The distribution of accounts over the MSE per account on the SVM model.

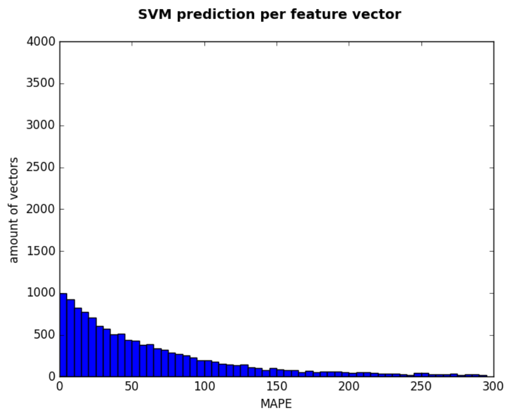

Figure 10.6: The distribution feature vectors over the MAPE for the SVM model.

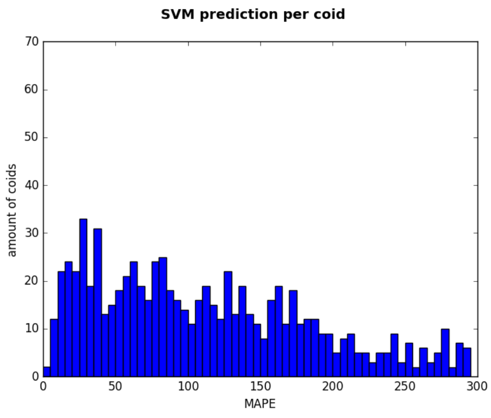

Figure 10.7: The distribution of account over the MAPE per account for the svm model.

10.3. Baseline comparison

Each model was compared to a baseline. This baseline is described in Section 7.1.

10.3.1. L-1 Prediction

The L-1 prediction model, in which the balance of the previous month is used to predict the next month, performs significantly better compared to the baseline. The results are plotted next to the baseline for comparison. In Figure 10.8 you can see that in the L-1 model more feature vectors can be predicted accurately. The first bin contains almost 1400 predictions, while the baseline only has 900 predictions in the first bin. More to the right you can see that both models have predictions that are very bad and give RMSE values higher than 10.

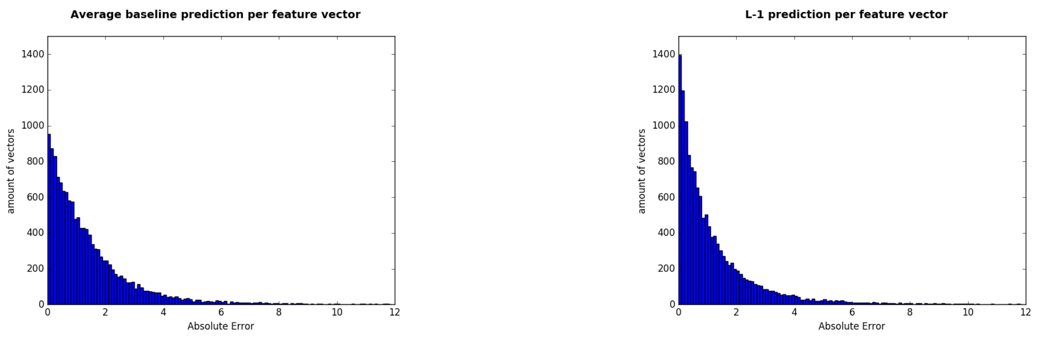

Figure 10.8: The left graphs shows the baseline prediction model, taking the average of the last 10 months to predict the next one. The right graph shows the L-1 prediction model, which takes the balance of the previous month as the prediction for the next one. On the x-axis you can see the Absolute Error. This is expressed in terms of z-scores, because the data is standardized. The y-axis shows the amount of feature vectors in the bins. For the baseline the median AE=1.02 and for the L-1 AE=0.77. Here you can see that in general, the L-1 prediction model gives better results than the baseline.

10.3.2. Ordinary Least Squares Regression

The Ordinary Least Squares model uses linear regression to find a line through the data to predict next data points. Unexpectedly, in Figure 10.9 the OLS method performs worse than the baseline. The median error of OLS is significantly higher (1.53 against 1.02). Due to this accuracy OLS seems to be a bad predictor for bank account balances.

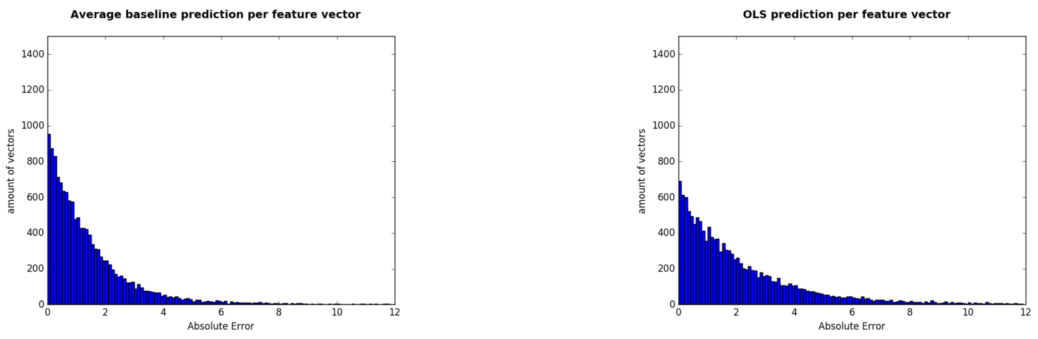

Figure 10.9: The left graphs shows the baseline prediction model of taking the average of the last 10 months to predict the next one. The right graph shows the OLS prediction model that uses the Ordinary Least Squares method to do a linear regression to prediction the next one. On the x-axis you can see the Absolute Error which is in z-scores because of the normalization of the data. The y-axis shows the amount of feature vectors in the bins. For the baseline, the median AE=1.02 and for the OLS AE=1.53. Here you can see that in general, the OLS model performs worse in term of errors than the baseline.

10.3.3. SVM Model

The results of the prediction made by the SVM model can be found here. To give a clear comparison, the results are plotted next to the baseline prediction model. The SVM is implemented as described in Chapter 8. As can be seen in Figure 10.10 the results of the SVM model are better than the baseline and also better than the L-1 prediction method. The difference in the median error is about 0.05. From these results, the SVM model as it is implemented now gives the best results compared to the other models we tested.

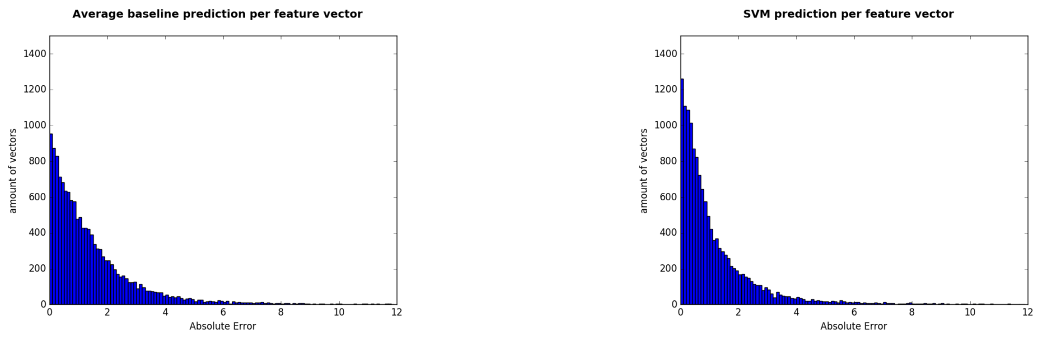

Figure 10.10: The left graphs shows the baseline prediction model of taking the average of the last 10 months to predict the next one. The right graph shows the SVM model to make predictions. On the x-axis you can see the Absolute Error which is in z-scores because of the normalization of the data. The y-axis shows the amount of feature vectors in the bins. For the baseline the median AE=1.02 and for the SVM AE=0.72 . Here you can see that in general the SVM model performs better on this data than the baseline.

10.4. Error intervals

To get an idea of the error intervals of our models, we firstly compared the median absolute error of all models on the same dataset. The SVM and L-1 models showed the lowest errors and therefore we have done some more tests to get the 25th 75th and 95th percentile as well.

Position	SVM	L-1
25th percentile	0.29	0.31
50th percentile	0.72	0.77
75th percentile	1.57	1.69
95th percentile	4.50	4.46

Figure 10.11: The error intervals of our SVM and L-1 Model, measured per feature vector using the absolute error.

10.5. Results for accounts individually

For visualization purposes, we want to plot the course of the predictions for a few accounts. We are aware that this does not provide us any generalized information about the results, but it can help both our group and the project's client to understand how the predictions perform on the validation set. We plotted these results for 2 account with a high RMSE and 2 account with a very low RMSE. We can see in Figure 10.12 and 10.13 that for those two account the predictions are not good and both these accounts have a high RMSE. The accounts with good predictions and thus a lower RMSE can be seen in Figure 10.15 and 10.16. Interestingly enough, the results for the two account with bad RMSE seem to predict using a L-1 prediction, because almost every prediction is very close to the real value of the previous month. Another account, plotted in Figure 10.14 also has this seemingly L-1 behavior, but only has two bad predictions due to a sudden increase in the bank balance. For the 2 accounts with lower RMSE this does not seem to be the case and the prediction curve follows the curve of the real values more closely.

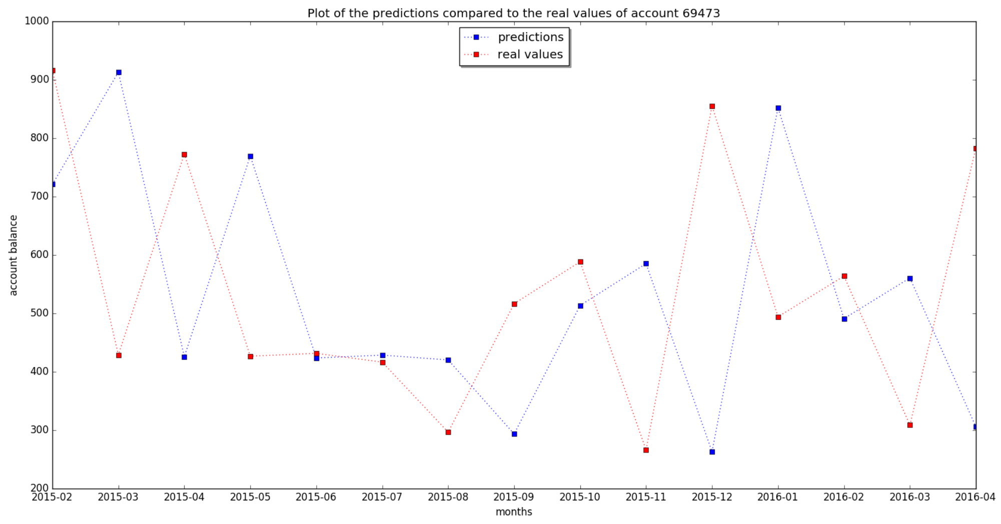

Figure 10.12: This graph shows the results of the prediction of the SVM on the validation set for one account in particular. This account is selected, because it has a high RMSE, thus having predictions with a high error. On the x-axis of the plot you can see the months for which the prediction was made. These month are the months in the validation set. On the y-axis you can see the balance of the account. On this plots you can see that the model seems to give predictions very close to the balance of the previous month. The predictions resemble the L-1 behavior.

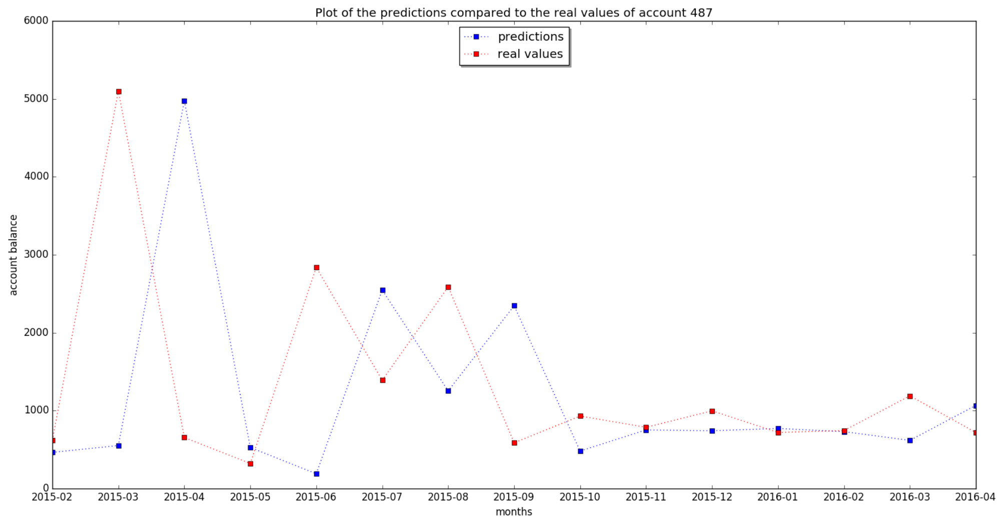

Figure 10.13: This graph shows the results of the prediction of the SVM on the validation set for one account in particular. This account is selected, because it has a high RMSE, thus having predictions with a high error. On the x-axis of the plot you can see the months for which the prediction was made. These month are the months in the validation set. On the y-axis you can see the balance of the account. On this plots you can see that the model seems to give predictions very close to the balance of the previous month. The predictions resemble the behavior of the L-1 prediction model.

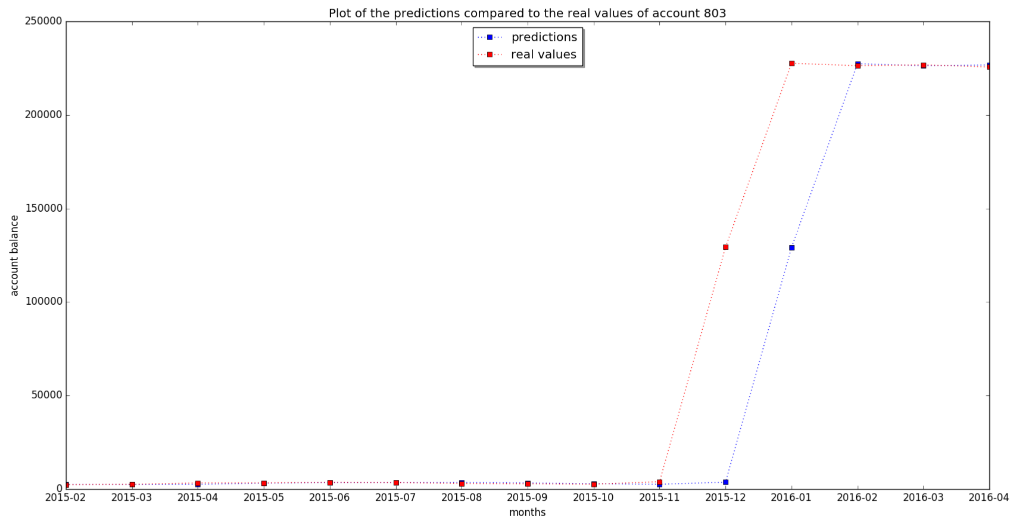

Figure 10.14: This graph shows the results of the prediction of the SVM on the validation set for one account in particular. This account is selected, because it has two bad predictions due to a sudden increase in the bank balance. But otherwise has fairly good predictions. On the x-axis of the plot you can see the months for which the prediction was made. These month are the months in the validation set. On the y-axis you can see the balance of the account. On this plots you can see that the model seems to give predictions very close to the balance of the previous month. The predictions resemble the L-1 behavior.

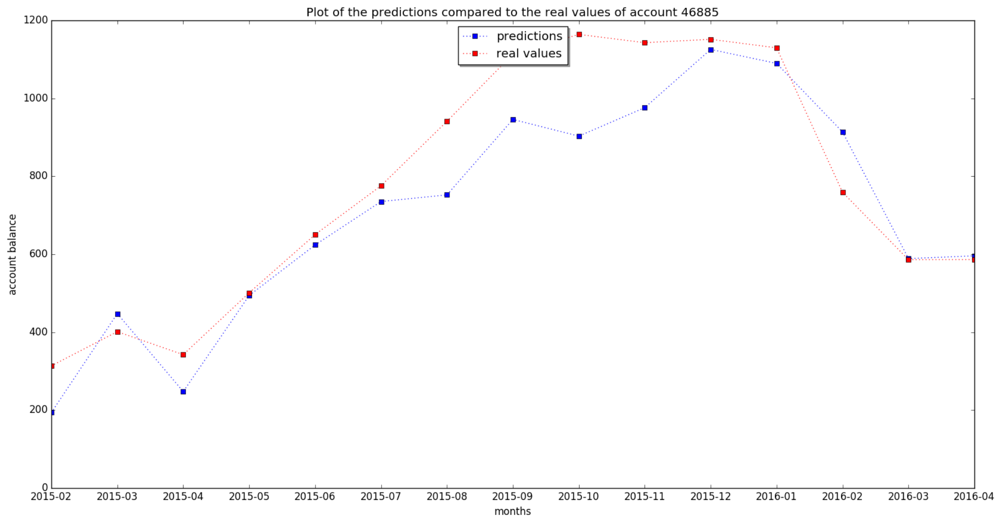

Figure 10.15: This graph shows the results of the prediction of the SVM on the validation set for one account in particular. This account is selected, because it has a low RMSE, thus having predictions with a low error. On the x-axis of the plot you can see the months for which the prediction was made. These month are the months in the validation set. On the y-axis you can see the balance of the account. On this plot you can see that the predictions are close to the real values.

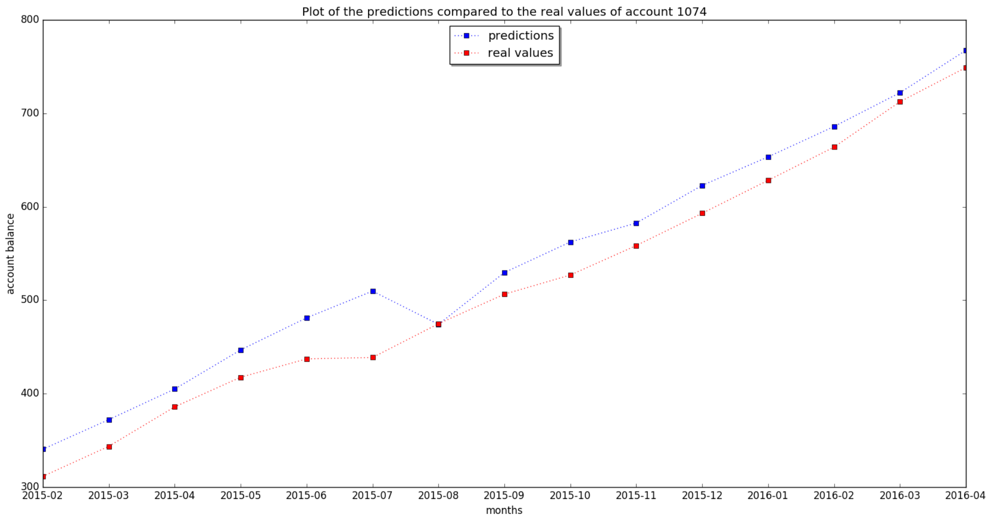

Figure 10.16: This graph shows the results of the prediction of the SVM on the validation set for one client in particular. This account is selected, because it has a low RMSE, thus having predictions with a low error. On the x-axis of the plot you can see the months for which the prediction was made. These month are the months in the validation set. On the y-axis you can see the balance of the account. On this plot you can see that the predictions are close to the real values.

11
Discussion

11.1. Interpretation of results

In Chapter 10 it can be seen that our L-1 and SVM model give similar results. A slight difference between the SVM and L-1 can be seen when comparing the median error of both models. The median error of th SVM model is 0.05 standard diviations lower than that of the L-1 model. By investigating the results of individual account for which predictions with significantly low of high RMSEs are made, we see a very interesting trend. When investigating the accounts with a high error first (figure 10.12, 10.13) it can be seen that the predictions made, come very close to the real values of the previous month. We interpret this result by concluding that the SVM model taught itself the L-1 prediction model. Another argument for this assumption is that by doing the parameter optimization for the SVM, we noticed that one of the most important inputs is the previous month with only a few other inputs, but not all previous 13 months as input works best. This can also be seen in figure 8.2.

Even though it looks like the SVM learned itself the L-1 prediction model, the results are slightly better so it has an edge over the simpler model. Secondly, our group investigated accounts with a low error (figure 10.15, 10.16) and found even more interesting results. In the graphs it can been seen that these accounts are predicted well but no L-1 trend can be observed in these predictions. Our SVM model sometimes predicts differently than the L-1 model and predicts some accounts with a higher accuracy. The SVM model is thus more dynamic and can learn more trends.

The trained SVM does not do well with predicting sudden high increases in bank balances either. This can be seen in the 2015-11 and 2015-12 prediction in Figure 10.14. Where over two months the balance suddenly increase to nearly 250000. Increases like these are almost impossible to predict because they can be caused by a lot of outside factors.

Finally we wanted to investigate the distribution of the error of both the models to be able to give better recommendation and interpretation of which model works best. Figure 10.11 shows the error distribution for the SVM and L-1 prediction based on percentiles. These percentiles can give us a better understanding of what the distribution of the errors is. For example we can now say that the top 25 percent of our predictions, made with the SVM model have a predicted value that deviates less than 0.29 standard deviations from the real value.

It can be seen that the median is very similar but the 95th percentile is lower for the SVM model so in general you can predict more accounts more precisely with the SVM. The other percentiles are also akin and no real improvement of one of over the other can be found. By looking at these results in general we can say that the SVM is this state gives only little improvement and takes longer computation time over the L-1 because of the necessity of the training-phase. Our group recommends to further investigate this behavior and try to improve the SVM because there are still a lot of improvements that can be made that expectedly improve its accuracy. Even though from the research it is clear that machine learning is more promisable than simple statistical models, for now we can only say that the results are slightly better. To be able to say this with more certainty our models have to be improved and more test should be ran.

11.2. Recommendations

Having worked with the client's systems for the past couple of weeks, we noticed a number of things the client could do to better facilitate machine learning solutions in the future.

11.2.1. Infrastructure

Machine learning requires large data sets in order to give accurate predictions. During the course of this project, we did not have access to sufficient computational resources. It was not surprising that our laptops could not handle the computations that we required for our models. However, when we could not run our models on Jupyter environment this was rather inconvenient. Reading a 5GB sample of our dataset was too much for the systems that were available to us. In the case that our client wants to take machine learning systems into production, it is of paramount importance that sufficient computational resources are made available.

11.2.2. Machine Learning Libraries

Using TensorFlow to create models similar to ours. The client would however have to consider that this would take a significant amount of time, as all algorithms would have to be implemented from scratch.

Based on our own experience, we would suggest that the client looks into Scikit-learn. Another option is Scikit-flow, which offers TensorFlow's functionality in combination with Scikit-learn style syntax and pre-made algorithms. The latest version of TensorFlow ships with Scikit-Flow by default, however the client's systems ran an older version of TensorFlow and were not compatible with the latest version.

11.3. Future work

Eventually we have some future improvements that can be made.

Clustering

Firstly, we have not included the clustering in our main model. The primary reason for this was that with the visualization created using PCA we were not able to distinguish clear clusters. Even though we have used multiple dimensions to cluster our clients on, there were no separate clusters of clients to be seen. The clustering can be improved by picking more features to cluster on and trying different clustering methods. To improve the feature selection and visualisation, a different technique for dimensionality reduction can be used such as t-SNE [23].

Cutting the data

Secondly, when deciding on the inputs for clustering and the regression we had to make a lot of cutting decisions. For example, we had to decide which accounts were active enough to make a prediction. In the future it might be interesting to test whether it is possible to make predictions for inactive or outlying accounts as well.

Predicting further ahead

For now, we only made predictions one month into the future. It would be very interesting to see how well into the future we can make predictions about our clients as well.

Normalization

Furthermore, to validate our model we used the MSE, this MSE is calculated as the difference in Z-score between the predicted and the real value. For now we used the Z-score to normalize our data. As described in chapter 3, this deviation can be deceiving. Therefore, it would be interesting to test another normalization method.

Error measures

Changing the normalization might also affect the MSE, for now the MSE is calculated on the Z-scores, but because we have seen that those can be deceiving. A low MSE does not necessarily mean that the prediction is good. It might be interesting to try different approaches for expressing the real and predicted value such that we can measure the error differently.

Parameters SVM

Eventually we did not have enough time to fully optimize our SVM. Optimizing multiple parameters of an SVM can be seen as an optimization problem itself and it can be very time-consuming as changing one parameter can affect all the others. For now we have slightly optimized our parameters, but in the future this has to be done more extensively.

Features SVM

Due to time limitations we did not have enough time to perform extensive feature selection. The DWH contains a lot of information about payment accounts and the users of these payment accounts. With more time and research we would want to investigate further which interesting data for features is stored in the DWH.

Kernel SVM

For the kernel function of the SVM most tests were done using a radial basis function. However as mentioned in Section 8.1.5 both the linear kernel and polynomial kernel had errors that were not far off from the error of the radial basis function. It would be interesting to do more extensive testing with these different kernels.

Training input size SVM

Finally the last thing for the SVM we would really want to investigate further is what happens if these SVMs are trained on bigger sets. So far these were trained on relatively small sets (the data of 10.000 months of random payment accounts, see Section 8.1.5).

More machine learning methods

At last, during our process we have decided to only implement an SVM as a machine learning technique whilst we were originally planning to implement a recurrent neural network (RNN) as well. All information gathered in the research phase about the RNN can be fount in our research report in appendix D. This information combined with our framework makes it easier to implement a RNN model as well. One of our first improvements might be comparing a RNN model with our current models.

12 Reflection

In this chapter we will reflect mainly on the process and the work we have done during the project. Some parts of the project went very well and smooth, while other difficulties should be handled differently in the future. We have tracked our progress over the weeks and the difficulties we encountered in Chapter 13.

12.1. Data processing

During all of our previous courses and projects we have always worked with clean data.

During this project we encountered a few difficulties of working with real data:

- The data warehouse was complex and overwhelming. The documentation was over 500 pages and it was sometimes unclear how to best access some date we needed.
- Some values were missing and or incorrect. The data contained a lot of outliers, for example a lot of people had no age, or a default age that had never been set.
- Most of the data had to be normalized to get good results an be able to compare results with one another.

We had not really encountered these problems in other projects before, the data we had to work with has always been clean and was mostly already provided. Because we did not have any experience with these problems we did not take them into account well enough in our initial planning.

Because of this we were sometimes lashing out in the dark with the problems we had, or only realized fairly late that they even existed. A lot of literature can be found about the best way to process the data, but we were unsure of the best approach. Furthermore because we did not know some problems even existed, it was hard to identify them and find a suitable solution.

12.2. What we learned

By being thrown into the deep during this project, we have gained new insights into how real world companies and projects work and with that learned a lot. One of the greatest lessons is that data processing is a really big and important part of the process and can not be underestimated. By doing this we have learned more about how to clean and normalize the data in a way appropriate for the data.

In a future project we would consult specialists within the company earlier in the project lifetime. In the fifth week, our colleagues from Utrecht have helped us a lot with problems we had been coping with from the beginning of the project. They have been working with the company for a long time and had a lot of experience with the data warehouse. Consulting them earlier would have saved us a lot of time.

12.3. Planning

During the first week of the project we sat together with our supervisors from the client's side and the TU. This gave us enough knowledge to write our project plan (appendix A). From the first week one we made a planning, planning ahead until the end of the project. After five weeks we realized that we had spent too much time on processing the data, and getting familiar with the systems to stick to our original planning. In week five we adapted our original planning and decided to drop some of our planned tasks (Appendix B). Furthermore we also planned a mid term meeting with a supervisor form the TU to consult him on these changes.

13 Process

13.1. Weekly progress

In this chapter we will present an overview of our progress on a week by week basis.

13.1.1. Week 1

During this week most time was spent on setting up the project. Unfortunately, not all legal arrangements had been completed. Therefore it was not possible for Max and Bernd to work on site and they had no access to the client's systems. Felix and Chantal received all required material during week, but Max and Bernd only got this in the second week.

Apart from that, the purpose of our project was not yet clearly stated. Therefore we had to specify this as well. It took some time to get this straight with the client and our supervisor.

Once the project description was clear and we had set up most of our tools, we started with the research phase and our project plan.

What we planned to get done:

- Setting up organizational structure between members.
- Meetings with client and supervisors.
- Orient toward relevant scientific topics
- Arranging the SCRUM Plans and project plan.
- Produce: requirements + organizational document (project plan)

What actually got done:

- Set up organizational structure between members.
- Meetings with client and supervisors.
- Orientation through literature study.
- Arranging the SCRUM Plans and project plan.
- Stating requirements for the end product.
- Produce: requirements + organizational document (project plan)

Difficulties

- No complete access to facilities for our entire group and thus not able to start as planned.
  Solved by: Max and Bernd worked from the university on their own laptops. In week 2 they had to set up their accounts at the client's location.
- Unclear project description.
  Solved by: Felix and Chantal discussed this the first day with the supervisors, after consultation with the rest of our group a clearer description was written.

13.1.2. Week 2

The second week we mainly focused on our literature study. We discovered that there was a lot of useful information to be found. In consultation with our supervisor, we decided to finish the report in the third week so that we could deliver a more in-depth research report.

What we planned to get done:

- Literature research of state of the art algorithms.
- Produce: Literature research, which will conclude on the algorithms that are interesting to look at further during the next weeks.

What actually got done:

- Literature research of state of the art algorithms.
- Literature research, which will conclude on the algorithms that are interesting to look at further during the next weeks.

Difficulties

- A lot of decisions had to be made and various questions still remained that where hard to answer with the research we had performed at that time.
  Solved by: Delaying the deadline for the report to the third week.

13.1.3. Week 3

This week we finished the research report and started with discovering the data by visualizing statistics and running some queries.

The biggest setback in this week was the overwhelming size and complexity of the data warehouse (DWH). The documentation of the warehouse was a file of over 550 pages and it was hard to filter out good accounts. Our basic queries were already quite long and took 10 minutes to run on average. Therefore it took us more time than expected to explore the data.

This week we made some decisions about the benchmarking as well and made scripts in Python to visualize data from the data warehouse.

What we planned to get done:

- Constructing test cases based on real data from our data warehouse.
- First implementation of 2 different algorithms.
- Create visual representation of results

What actually got done:

- More literature research about the benchmarking and data analysis.
- Deciding on how to benchmark.
- Explore structure of data warehouse, starting with the queries.
- Documenting and visualizing useful data.
- Creating scripts for visualizing data in Python

Difficulties

- It took us some extra time to finish the benchmarking and data analysis part in the report.
- Getting to know the basics of our data warehouse alone took us a few days, after that it was also complicated to visualize useful data and understand what we were dealing with.
  Solved by: We made sure to first work with the data and sort out what we needed from it. Therefore we were forced to move most of the tasks planned for week 3 to a later moment in the project.

13.1.4. Week 4

During the fourth week we made a decision to cluster the clients before making a prediction. This was not something we planned to do beforehand, hence our planning was not correct anymore.

This week we spent a lot of time querying the data warehouse to obtain all information needed for the clustering and prediction itself. Apart form the querying, we also had to start normalizing the results properly to make sure they have similar magnitudes relative to each other.

What we planned to get done:

- First implementation of a 3rd algorithm.
- Improving previous algorithms

What actually got done:

- Improving our research report with obtained results from the DWH.
- Normalizing old queries in a suitable way.
- Deciding to start clustering our customers for better results.
- Literature research for a suitable clustering method.
- Deciding the feature vector for the clustering.
- Writing queries for the feature vector for clustering.

Difficulties

- Our results from the data warehouse were too messy and hard to compare
  Solved by: Normalizing most of the data in a way suitable for each individual entry.
- We realized that we did not have enough information about one account to make a single prediction
  Solved by: Clustering the accounts to obtain more similar accounts for training.
- Because we only now started with the clustering we had to rework our planning, therefore again it was hard to finish the tasks planned for week 4

13.1.5. Week 5

During the fifth week, we did a midterm review to see what we had achieved so far and what still had to be done.

We had spent a lot more time with the data analysis than expected and also inserted the clustering in our solution. Therefore we fell behind on our initial planning.

In response we adapted our planning to make it useful again. The biggest difference lies in the regression approach. In our initial plan we wanted to implement two machine learning algorithms for the regression. Keeping in mind that it is better to deliver one working algorithm than two concepts, we decided to research the clustering properly and only implement one machine learning algorithm for the regression.

Furthermore we had a meeting with a TU supervisor to talk about these difficulties. Something that came up here was the software engineering goal of this project. Up until now our project tended more to a research project than a software engineering project.

This caused us to focus a bit more on the software engineering. In consultation with our TU supervisor, we decided to make some changes to take software architecture more into account.

Keeping in mind that the original plan was to implement more than one machine learning algorithm to compare different approaches, we decided to build a framework where various models can be implemented. Inside this framework, it is easy to compare different models on one data set or try different data sets on one model.

During this week, we got to finish the feature vector for the clustering and got quite far with the feature vector for the regression. Apart form this we focused on removing outliers from our data set.

At last, we met with some colleagues from the innovation lab in Utrecht. They had been working on a balance prediction model as well and were able to give us a lot of tips. Furthermore, someone who had been working with the data warehouse for over 30 years helped us a lot with writing queries.

What we planned to get done:

- Testing/Bench-marking/comparing the algorithms.

What actually got done:

- Implementing a suitable clustering method.
- Filtering out useful clients.
- Creating the feature vector for the clustering.
- Simple clustering of the filtered clients.
- Start with building of the feature vector for the regression.
- Making a revised planning for the upcoming period.
- Meeting with a TU supervisor to talk about our progress.

Difficulties

- We had fallen behind on our initial planning.
  Solved by: Making a revised planning for the weeks to come.
- There was not enough time left to finish all our initial plans, because we had adopted clustering into solution as well and because we needed to focus more on the software engineering aspects.
  Solved by: In the adapted planning, we decided to only implement one off-the-shelf machine learning algorithm, namely the SVM.

- We noticed a lot of outliers in our data, which made the clustering underperform.
  Solved by: Our colleagues at the innovation lab had more knowledge about the DWH than we did, they helped us by forming a query that filtered out almost all outliers. They also explained us how we could store tables inside the data warehouse since we previously did not have access to that.
- One of the main aspects is still software engineering and we have not been doing that so far, thus we do not have that much to hand in for the SIG review
  Solved by: Our TU supervisor advised us to postpone the SIG deadline one week to make sure we can get enough feedback. After this meeting we decided to start implementing a framework for our model to keep working on software engineering too.

13.1.6. Week 6

During the fifth week there was not much time to implement the framework, so we started with this in week 6. To guarantee a good quality of the framework we had a lot of discussions about the structure during this week.

When the framework was finished we successfully implemented two baseline models (L-1 and 12 month avg) and started on implementing the SVM.

This week we also finished the feature vector for the SVM. Also we set aside a final validation set, which we will not touch anymore until the final validation.

In this week the clustering showed us some good results, because the adapted normalization and removing of the outliers.

What we planned to get done:

- Fine tuning the clustering method.
- Building regression feature vector.
- Implementing baseline.
- Implementing Grey model.

What actually got done:

- Fine tuning the clustering method, mainly by normalizing input.
- Implementing k-means in TensorFlow
- Finishing regression feature vector with all useful clients.
- Finishing average baseline.
- Starting with Grey model.
- Finishing a simple framework.
- Starting with applying the basic SVM to our dataset
- Setting aside a validation set inside the DWH.

Difficulties

- Our client did not agree with the decision to drop the second machine learning algorithm and thus TensorFlow.
  Solved by: After discussing a solution with our group, we met again with the client. Our proposal

was to implement K-means in TensorFlow, the same clustering method we had already used with the Scikit-learn library. This way we could compare TensorFlow with a more high level off-the-shelf implementation. After this comparison, it is easier to support our decision to not build a Recurrent Neural Network in TensorFlow

- Technical problems with the facilities provided by our client:
  - On friday our VDI did not work due to "maintenance-issues". Because of this, we were not able to access our work since our normal work environment is not allowed to be connected to Git or have Python installed.
  - We cannot upload files larger than 25mb to the Docker via the access page in a web browser.
  Solved by: We focused more on the report and things we were able to access, unfortunately the whole weekend our VDI remained useless. After contact with the help desk a ticket was submitted to solve the problem.

13.1.7. Week 7

In the beginning of this week our VDI was still down. This gave us some time to focus on proper documentation and updating the report.

Because we had to hand in our SIG code for the code review this week, we also spent some time rearranging and completing the framework and writing a lot of tests.

After our "week 7" meeting with our TU supervisor, we realized that we had a strangely high MSE. Our TU supervisor pointed this out to us, but we could not pinpoint directly what the cause was. After some consultation with the whole group, we found out what the problem was.

What we planned to get done:

- Finishing the input vector for the SVM.
- Implementing SVM.

What actually got done:

- Finishing the framework.
- Testing the framework.
- Comparing the TensorFlow implementation against the Scikit-learn implementation.
- Adding the portfolio feature to the clustering
- Creating the complete feature vector for regression and put it on the Docker

Difficulties

- Strangely high MSE on most of our models
  Solved by: Consulting with our whole group, walking through our process and understanding a bit more of what we had to normalize and what the interpretation is of the results we acquired. By doing this we found that the normalization was not performed correctly. From this we learned that we have to take a few steps back to reflect on what the results mean and whether they are feasible.
- The docker is only able to run a 1% sample
  Solved by: Consulting with the client. No solution has been found, because the docker is limited in memory. Running the code on our VDIs is the only solution so far. Having more computational power and memory is a recommendation towards the client when they want to scale these kinds of solutions in production.

13.1.8. Week 8

This week we have been working to get results for the model by performing predictions on the validation sets. We could not use the feedback from SIG since we delivered the code one week after the deadline, because of a deadline extension. However we also cleaned the code and made some small changes to the framework. Unfortunately, one group member got ill during this week and had to rest all week on doctor's orders. Therefore we were lighter on resources this week. We contacted the Student Advisors about this problem and we also mailed the Project managers from both TU Delft and the client.

What we planned to get done:

- Visualizing results.
- Benchmarking the model.
- SVM improvements.
- Restructure code with new framework implementation.

What actually got done:

- Visualizing results.
- Benchmarking the model.
- SVM improvements.
- Restructure code with new framework implementation.
- Restructured report to fit the rubrics and feedback of our TU supervisor.

Difficulties

- Chantal got ill and could not come to the university for four days.
  Solved by: When she got better, she immediately consulted the study advisors. Apart from that, the rest of the team kept working.

- The input data for our algorithms on still had some NULL values in them. This prevented us from performing normalizations and this needed to be fixed as soon as possible. **Solved by:** In the data warehouse, we adjusted the queries to make sure no NULL values were present. There were some calculations that needed division and some divisors were 0. The result of this division needed to be a 0 and not a NULL.

13.1.9. Week 9

In the beginning of the ninth week, we received our SIG feedback, it was a bit later than expected, but we started immediately after we received it. We received a four star rating which means better than average maintainability. The comments were that a higher rating was not obtained due to unit size and duplication. Mostly our main methods still had a lot of duplication and was not divided into methods a lot. Finally we also received the comment that we did not have a lot of test code compared to the other code. We started to calculate the MSE based on distinct predictions instead of averages per account and came to the conclusion that this reduced our MSE significantly. After further investigating this, we came to the conclusion that our normalization had some problems and this affected our predictions and thus the errors. This could be fixed quickly after detecting where it went wrong. During this week, we further investigated predictions with high MSEs. We discovered that something was wrong regarding the monthly averages used in the feature vector. This forced us to query all the training and visualization data again and rerun all our models to obtain the correct results.

What we planned to get done:

- Running untouched validation set.
- Finishing everything that is left.

What actually got done:

- Improving test coverage.
- Processing the SIG feedback.
- Removing outliers from average and standard deviation calculation and thus fixing the normalization.
- Re-querying our feature vectors
- Running the final validation sets to get final results on all our models.
- Finishing the final report.

Difficulties

- We got very small errors and felt something was wrong.
  Solved by: Looking at individual cases of small errors, we came to the conclusion that our normalization was imperfect and changed it.
- By investigating single predictions we came across errors in the monthly averages.
  Solved by: Recreating all training and validation data and run all our models again on the good datasets, to create correct plots.
- SIG feedback, duplication, unit size and testing.
  Solved by: We immediately processed the SIG feedback. We made extra methods in the mains to reduce code duplication there. Extra tests were also written to reduce the imbalance between testing and regular code.
- Max his laptops were stolen. Therefore he could no longer proofread our report.
  Solved by: We sent excerpts of the report for Max to spellcheck. His other tasks were already completed and therefore required no solution.

13.2. Group

At the beginning of the project, our client provided us with some useful tools to keep track of our project.

13.2.1. Jira

During the project our client provided us with an issue tracker. Through this tool, we could easily keep track of the tasks that had to be done and who was going to do them. Over the weeks we got used to the tool and used it more during each sprint.

The tool is able to provide us with a lot of figures about the created tasks and thus the work delivered, two of those can be seen in Figure 13.1 and 13.2 .

13.2.2. Stash

For git repository management our client provided us with Atlassian Stash, with this program we could easily manage our git repository. It allowed us to keep a good overview over the network flow and easily review the created pull requests.

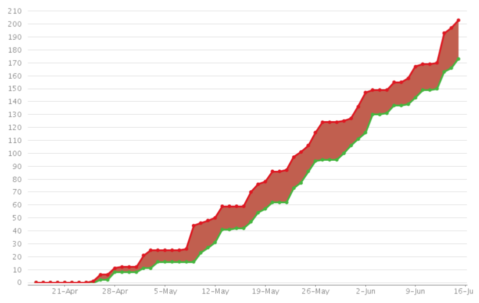

Figure 13.1: Created and resolved issues against the days of our project.

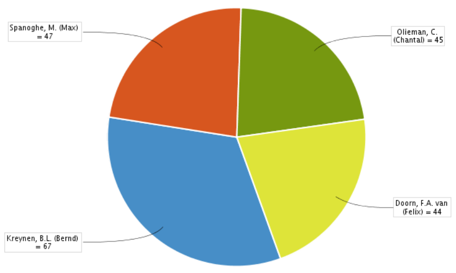

Figure 13.2: The amount of issues assigned to individual group members.

13.2.3. Communication within the group

During the project we had a thought through communication structure within the group. Every day we would start with discussing how far we have gotten and what had to be finished before the end of the day and at the end of the week.

Every Monday, we would have a big team meeting to evaluate our progress until that week and look at the project broad planning. In this meeting, we would discuss the tasks that needed to be finished in the upcoming week. These tasks were mostly divided amongst group members on Monday. Usually a few remained unassigned, those were picked up later in the week by team member who had already completed their assigned tasks.

Because we always worked at the same location, it was easy to consult each other when decisions had to be made. Occasionally when problems occurred or difficult decisions had to be made, we sat down with our whole group to discuss what was going on and find a suitable solution together. This way we kept everyone up to date with the decisions we made and thought all steps through carefully.

14 Conclusion

We have shown that it is possible to create accurate predictions of bank balances through the use of support vector machines. To make these predictions, we have built a framework in Python for our client. Despite the client's initial interest to implement our machine learning algorithms in TensorFlow, we decided to base this part of our framework on Scikit-learn. This framework is better suited for rapid research into already existing techniques. The modular design of this framework allows the client to add desired functionality to this system.

The main question for this project was: How well can we predict bank balances? With the model we have created, we have managed to accurately predict the balances of a large group of clients. However, for the remainder of the clients we feel that our predictions are not accurate enough to be used in practice. To make our predictions good enough for all clients, many things could be done to improve our model.

Our machine learning model performed better than our baseline and non-machine learning models. For our client, this means that machine learning algorithms are a viable tool for creating bank balance predictions. We can therefore state that we were successful in achieving the project's main goal.

Initially the client had wished to have these models implemented using TensorFlow. We identified that this was not in their best interest, as it was not the most suitable tool for the system that they wished for. The framework that we have created during this project is better suited for the client's purpose of integrating existing machine learning techniques into their environment.

A Project Plan

A.1. Project Assignment

In this section we will give a short description of our client and why they are interested in our project. We will also describe the purpose of this project. After that we will discuss the specification our assignment

A.1.1. Project Environment

The client of this project is a large Dutch bank holding company, one of the leading financial institutions in the Netherlands. Their main focus is providing customers with mortgages, savings accounts and payment accounts.
The client also develops IT applications to give customers insight into their financial situation.

A.1.2. Project Goal

The goal of this assignment is to develop a prototype of an application that will make bank balance predictions by using Machine Learning techniques. A secondary objective, is to evaluate whether such a system can be built using Google TensorFlow and how this could be integrated in the existing sytems of the client .

A.1.3. Assignment Specification

The project team will have to create a prototype of an application that satisfies the goal specified in the previous section. The assignment is split into two main timeframes. In the first of these, the team will research feasible machine learning techniques for time series prediction and Google TensorFlow implementations of these methods. The second phase concerns implementation of the actual prototype and tuning of the machine learning methods employed.

A.2. Requirements

Must Haves

- Take user data from the clients SQL database to use as input.
- Output personalized prediction of bank balance for an upcomming time slot.
- The algorithm makes use of a Machine Learning method to learn the behavior of the client.
- Report on research in literature for state of the art techniques for predicting time series and comparing them. By this comparison a decision will be made on the best option for our program.

Should Haves

- Simple GUI in which the client can assign a user for prediction and get a prediction of the bank balance in the form of a number.

Could Haves

- Possibilities of parameter tuning in the GUI.
- Extra visualization options.

Won't Haves

- No financial advice will be given except for a prediction of the bank balance of the specific user.

A.3. Time Planning

Week 1

- Set up organizational structure between members.
- Meetings with the client and supervisors.
- Orientation on scientific topics
- Arranging the SCRUM Plans and Product plan.
- Produce: requirements + organizational document (product plan)

Week 2

- Literature research of state of the art algorithms.
- Produce: Literature research which will conclude on the algorithms that are interesting to look at further during the next weeks.

Week 3

- Constructing test cases from real data from our databases.
- First implementation of 2 different algorithms.
- Implement visual representation of results

Week 4

- First implementation of a 3rd different algorithm.
- Improving previous 2nd algorithms.
- Improving previous 3rd algorithm.

Week 5

- Testing/Bench-marking/comparing the algorithms.

Week 6

- Making a decision on what algorithm to implement in more detail.

- Decide on what improvements still can be done and implement them.

Week 7

- Further implement the final system with the chosen algorithm and improve this.

Week 8

- Starting with the finishing of the report.

Week 9

- Further implement extra features (Could have)

- Finishing the report.

Week 10

Final week with presentations. All preparation for the final presentation will be made in this week. Maybe very small changes to documents and our software solution can be made.

A.4. Organization of the process

A.4.1. Agile Process

Our group will be using an agile approach to create this software solution. This means that we will use a variant of the SCRUM method (link).

A.5. Organization of the process

A.5.1. Agile Process

Our group will be using an agile approach to create this software solution. This means that we will use a variant of the SCRUM method (link).

A.5.2. Versioning our software product

For the versioning of our product we will use Stash (Git). We will be using the internal repository of out client. This is the tool of choice of the client and offers us excellent version control.

A.5.3. Report

We have decided to work on our report 4 hours per person each week. This will ideally be every Thursday morning.

A.5.4. Roles

We would like to assign certain roles to different members. This because we can always address the right person for questions and or tasks that need to be fulfilled.
Chief communication: Chantal Olieman
Chief Scrum: Max Spanoghe
Chief Versioning: Felix Van Doorn
All team members bear responsibility for developing, testing and reporting throughout the course of the project.

A.5.5. Work location

The team will work at the client sit in Den Bosch for two days a week starting from Week 2. The other days the team will work in Delft, preferably at EEMCS.

A.5.6. Meetings

Monday 9:00 Sprint Meeting

Monday 10:00 Meeting with Cliff & Wouter

Thursday 14:00 Meeting with Thomas

Friday 09:00 Sprint Review

Tuesday, Wednesday, Thursday 9:00 Daily scrum meeting.

A.6. Quality Assurance

In this section we will describe what measures will be taken into account to guarantee the quality of the product.

A.6.1. Modularity

As we are creating a prototype, modular design of our product is essential. This would allow our client to easily pick what code they would like to reuse for a new version of this software. This will also allow us to smoothly exchange our classifier of choice.

A.6.2. Accuracy

As our application will employ Machine Learning techniques, accuracy of these techniques is essential to deliver a good product. An inaccurate classifier will render our prototype effectively useless.

A.7. Project Risks

In this section we will briefly describe all factors we have identified as forseeable risks at the start of this project.

A.7.1. Security

The data we will be working with during the course of this project is considered to be sensitive. It is of the utmost importance that we uphold the security measures requested by the client These measures include removing smartcards from laptops, not working with sensitive data on (semi-) insecure networks and sharing files containing sensitive information on external services.

A.7.2. Lack of data

In order to make accurate predictions for a given customer, this customer needs to have sufficient data available to work with. If the customer is a new customer, it is almost certain that our prototype will be unable to make a reasonable prediction for this customer. Although it is unlikely that the majority of the customer will fall under this category, we will also be examining ways of dealing with such cases properly.

B
Mid Term Update

Mid-term Update

The planning we made turned out to be a little bit optimistic on the retrieving of the data. We spend a lot of time getting to know the data warehouse and after that building our queries. Below you can see our main tasks in the past four weeks.

Week 1

- Set up organizational structure between members.
- Meetings with the client and supervisors.
- Orientation on scientific topics
- Arranging the SCRUM Plans and Product plan.
- Stetting our final guidelines for the end product.
- Produce: requirements + organizational document (product plan)

Week 2

- Literature research of state of the art algorithms.
- Produce: Literature research which will conclude on the algorithms that are interesting to look at further during the next weeks.

Week 3

- More literature research about the benchmarking and data analysis.
- Deciding how to benchmark.
- First feeling of the database, getting started with the queries.
- Documenting and visualizing useful data.

Week 4

- Improving our research report with obtained results from the DWH.
- Normalizing the old queries.
- Deciding to start clustering our customers for better results.

- Literature research for a suitable clustering method.
- Deciding the feature vector for the clustering.
- Writing queries for the feature vector for clustering.

Conceptual planning

Since we did not stick to the original planning for mostly the past two weeks, we had to adapt our planning for the upcoming

ve weeks as well. Unfor- tunately analysing the data and creating suitable test and training sets took us a lot more time than expected. Therefore in our conceptual planning we dropped some task we previously thought were able the mange but would not be possible any more. Here we kept in mind that it would be betting delivering a good working product than a product that does not work at all.

Week 5

- A suitable clustering method implemented and working.
- Filtering out useful clients.
- Creating a validation set.
- Creating feature vectors of coid's for the clustering.
- Simple clustering the
  ltered clients.

Week 6

- Fine tuning the clustering method.
- Build regression feature vector.
- Implementing baseline.
- Implementing Grey model.

Week 7

- Finishing the input vector for the SVM.
- Implementing SVM.

Week 8

- Visualize all result.
- Benchmarking the model.

Week 9

- Running untouched validation set.
- Finishing everything that is left.

Week 10

Final week with presentations. All preparation for the final presentation will be made in this week. Maybe very small changes to documents and our software solution can be made.

C
Clustering results

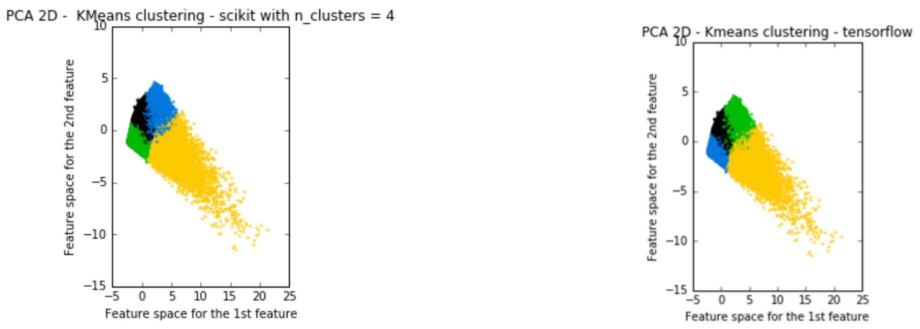

Figure C.1: The left graph shows the PCA analysis of the clustering results on our dataset using scikit-learn. The right graphs shows the PCS analysis of the clustering results on our dataset using Tensorflow. For both graphs, the k-means algorithm is used with k = 4. Both scikit-learn and Tensorflow give the same result for k = 4.

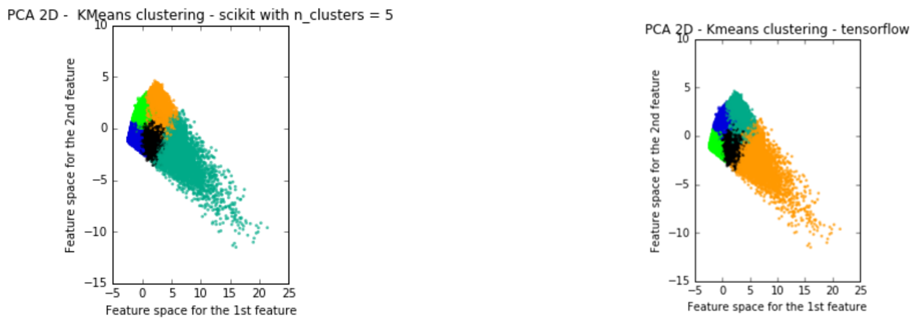

Figure C.2: The left graph shows the PCA analysis of the clustering results on our dataset using scikit-learn. The right graphs shows the PCS analysis of the clustering results on our dataset using Tensorflow. For both graphs, the k-means algorithm is used with k = 5. Both scikit-learn and Tensorflow give the same result for k = 5.

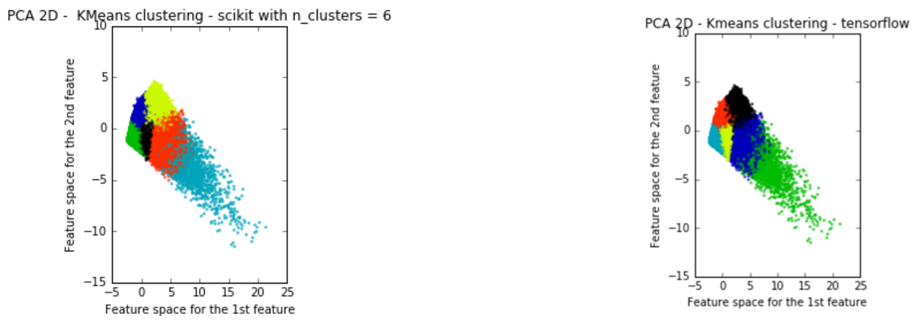

Figure C.3: The left graph shows the PCA analysis of the clustering results on our dataset using scikit-learn. The right graphs shows the PCS analysis of the clustering results on our dataset using Tensorflow. For both graphs, the k-means algorithm is used with k = 6. Both scikit-learn and Tensorflow give the same result for k = 6.

D

Recurrent Neural Networks

Traditional feed forward neural networks work very well on classification of single cases. Unfortunately, neural networks are not able to take connections between multiple instances into account. This is what makes Recurrent Neural Networks (RNN) special. Consider the simple feed forward neural network $y = W \cdot x$, an output y is created by multiplying the input vector x with the weight matrix W. If we translate this to an RNN, a sequence of inputs $x_1, ..., x_n$ is needed. The output y_i is calculated by $y_i = W \cdot x_i + W_r \cdot y_{i-1}$. The weight matrix W_r is used to include all previous outputs into the calculation of the i-th output.

D.1. Long Short Term Memory

RNN's have a good ability to recognize short term dependencies, unfortunately in practice RNN's are not capable to learn long term dependencies. [3]

Fortunately, a specific kind of RNN's is able to deal with this problem, the Long Short Term Memory networks(LSTM). The LSTM consist of a set amount of connected modules. These separated modules are able to "remember" information for an arbitrary length of time. Every separated module consists of four neural network layers working together. The key to the LSTM is the *cell state*, a "red line" flowing through the entire network from module to module. The cell state will be denoted as C_t, where i is the number of the module the cell state is in. Inside every module, three *gates* decide whether information is added to the cell state or not. These gates consist of a neural network layer and a point-wise multiplication operation. These gates will have a sigmoid layer, deciding to let everything (1) or nothing (0) through and a tanh layer pushing values between -1 and 1.

- The first gate is called the *forget gate* this sigmoid layer decides if we should keep information gathered in the previous modules (thus the information in the incoming cell state C_{i-1}). This is decided by inputs x_i and y_{t-1}. For every value in C_{i-1} a 1 or 0 is outputted. This can be written as $f_i = \sigma(W_f \cdot [y_{i-1}, x_i] + b_f)$ Where W_f represents the weight vector for function f_t. The value for C_i up until now will thus be:

$$C_t = C_{i-1} \cdot f_i. \tag{D.1}$$

- To decide whether we will store the new information in the cell state, we use the *input gate layer* This layer consists of a sigmoid layer ($g_t = \sigma(W_g \cdot [y_{i-1}, x_i] + b_g)$), deciding whether or not to store the information created in this module. A tanh layer creates candidate values $\tilde{C}_i$ for C_i. These candidate values $\tilde{C}_i$ are calculated by $\tilde{C}_i = \tanh(W_C \cdot [y_{i-1}, x_i] + b_C)$. C_i now looks like:

$$C_i = C_{i-1} \cdot f_i + \tilde{C}_i \cdot g_i. \tag{D.2}$$

- Lastly we have to compute our output h_t. This is done by another sigmoid layer o_i ($o_i = \sigma(W_o \cdot [y_{i-1}, x_i] + b_o)$). This layer decide which parts of the cell state C_i we want to output from our module. The cell state is put through another tanh layer, to creates outputs between -1 and 1. The final output will then look like:

$$y_i = \tanh(C_i) \cdot o_i \tag{D.3}$$

Instead of a normal neural network with inputs x_t and outputs y_t, we have now created a model that is able to remember information for a long time through the cell state. The recurrent neural networks are known for its loops and once we build a loop inside the cell state, we will have a way to store parts of the information indefinitely.

D.2. Suitability

As stated before, the LSTM is capable of "remembering" information from previous inputs. This together with the learning of the algorithm makes it a great candidate solution for our problem. It is thus no surprise that financial time series predictions with RNN's have often been discussed in literature [16], [4]. The effects have shown to be good for financial time series in stock prices and currency exchange rates [4], [13]. However, our data is seemingly different, a clients payment account is not as volatile as the data that can be extracted from stock markets or currency exchange rates. Neural networks have to be trained from scratch and are sensitive to underfitting, as well as overfitting. Here lies a difficulty in this approach, what amount of training data will suffice and when will we have to stop training the model?

D.3. Decisions to be made

This section will shortly outline the decisions that will have to be made when implementing the TensorFlow LSTM for our specific problem.

- Since the architecture of the network is very open this will be our biggest decision. The modules described above can be connected in different ways and the amount of modules will have a big impact on the performance.
- Furthermore the learning rate λ has to be set to decide how fast the model is trained.

Apart form that, we have to decide what input we will feed the RNN and what output we expect, more about this will be explained in the data section.

D.4. Implementations

A well working existing implementations of recurrent neural networks can be found in the RNN application of TensorFlow. This existing implementation is created for language predictions. But the principle works the same was as for time series. In this project we plan on partially adapting this implementation to make it possible to TensorFlow for RNNs.

E
Project Description

De desbetreffende bank hecht veel waarde aan de financiële weerbaarheid van haar klanten. Om dit bij klanten te kunnen verbeteren, is het belangrijk om inzicht te hebben in de financiële situatie. De financiële situatie uit het verleden is zichtbaar in de transactiehistorie. Hierin valt in detail terug te zien welke mutaties er zijn geweest op een rekening. Van elk van deze mutaties is tevens bekend in welke (uitgaven)categorie deze transacties zijn geclassificeerd (al dan niet gecorrigeerd door de klanten zelf).

Naast het inzicht in het verleden, de bank geïnteresseerd in inzicht in de toekomst. Hiermee wil het klanten beter in staat stellen om financieel in control te zijn. Daar waar het verleden een accuraat beeld geeft, betreft de toekomst echter altijd een voorspelling, met de daarbijhorende nauwkeurigheid.

Eind 2015 heeft Google zijn tweede generatie zelflerende software opensource gemaakt onder de naam TensorFlow. TensorFlow is de opvolger van de in 2011 gebouwde deep learning-infrastructuur DistBelief voor het (interne) Google Brain-project. Sinds die tijd is het een erg actief project gebleken

(https://github.com/tensorflow/tensorflow/graphs/commit-activity). De bank is voor diverse toepassingen geïnteresseerd in de toepassing en het gebruik van deep learning algoritmes en frameworks. Een daarvan is TensorFlow.

Concreet is de bank opzoek naar de toepasbaarheid van TensorFlow in de context van het voorspellen van het toekomstige saldo van een rekening, om zo klanten beter financieel weerbaar te maken.

Aanvullende vragen:

- Tot hoe ver in de toekomst is met voldoende zekerheid het saldo te voorspellen?
- Aan welke eisen moet de rekening voldoen om het toekomstige saldo te kunnen voorspellen?
  - Aantallen transacties?
  - Verhouding interne/externe transacties?
  - Type rekening: betaalrekening/spaarrekening?
- Wat komt er kijken bij het inzetten van TensorFlow in een productieomgeving
  - Integratie met bestaande systemen (gezien vanuit TensorFlow)
  - Resourceinschatting / realtime-vs-batch

F Framework Documentation

Contents:

class `AbstractModel.`**`AbstractModel`** (*validator*)

> **`predict`** (*input_vec*)
> Making a prediction by using the input and perhaps some other aspects of the model.
>
> **Parameters**
>
> - **`self`** (*Model*) – The model itself
> - **`input_vec`** (*List of inputs. The size of the list is the amount of predations we want to make. The size and type of the input is depending on the model.*) – For every validation instance the suited input values, depending on the model.
>
> **Returns** The prediction made by the model
>
> **Return type** List of values
>
> **`train`** (*input_vec, real_val*)
> This method trains the model, based on the type of model implemented.
>
> Uses the predict method
>
> **Parameters**
>
> - **`self`** (*Model*) – The model itself
> - **`input_vec`** (*List of inputs. The size of the list is the amount of training instances. The size and type of the input is depending on the model.*) – For every training instance the suited input values, depending on the model.
> - **`real_val`** (*List of values. The size of the list is the amount of training instances.*) – The actual value derived from the data.
>
> **Returns** Nothing, update the model.
>
> **`validate`** (*input_vec, real_val*)
> Making a prediction by using the input and perhaps some other aspects of the model.
>
> **Parameters**
>
> - **`self`** (*Model*) – The model itself
> - **`input_vec`** (*List of inputs. The size of the list is the amount of predations we want to make. The size and type of the input is depending on the model.*) – For every validation instance the suited input values, depending on the model.
> - **`real_val`** (*Array of floats.*) – The set of actual values.
>
> **Returns** The prediction made by the model
>
> **Return type** List of values
>
> **`validator`** = **None**
> Initialize the model with chosen parameters. And depending on the model create the needed attributes
>
> **Parameters**
>
> - **`self`** (*Model*) – The model itself
> - **`params`** (*List of values. The size is depending on the model.*) – The parameters to define in the model.
>
> **Returns** Nothing, initialize the class.

class `AbstractValidator.`**`AbstractValidator`** (*params*, *error*)

`error` = None
Initialize the validator with chosen parameters. And depending on the validator create the needed attributes

Parameters

- **`self`** (*Validator*) – The validator itself
- **`params`** (*List of values. The size is depending on the validator.*) – The parameters to define in the validator.

Returns Nothing, initialize the class.

`validate` (*input_vec*, *real_val*)
Validate the model by making a prediction, then comparing the prediction with the real value by using the error method.

Parameters

- **`self`** (*Validator*) – The validator itself
- **`input_vec`** (*List of inputs. The size of the list is the amount of training instances. The size and type of the input depends on the validator.*) – For every validation instance the suited input values, depending on the model.
- **`real_val`** (*List of values. The size of the list is the amount of validation instances.*) – The actual value derived from the data.

Returns The error of the model over the validation set.

Return type List of values

class `AbstractError.`**`AbstractError`**

`error` (*pred_val*, *real_val*)
Calculate the error by comparing each value predicted by the model with the actual value. The error measure has to be determined here too.

Parameters

- **`self`** (*Error*) – The error itself
- **`pred_val`** (*List of values.*) – The value predicated by the model.
- **`real_val`** (*List of values.*) – The actual value derived from the data.

Returns The error between the real and predicted value.

Return type List of values

class `AbstractProcessBlock.`**`AbstractProcessBlock`**

`process` (*data_set*)
Transform the data and return it :param self: The class itself :type self: PreProcessBlock :param data_set The dataset to process :type data_set DataFrame

Returns A processed dataframe

Return type DataFrame

G
Data

G.1. Main statistics

For our project we had access to the entire Datawarehouse of our client. We had to decide what data was useful for the specific problem. In this chapter the main statistics about the data are given. For us, this was mainly an orientation into the database and gave us a clearer insight how much data is at hand and how the data is distributed. Also it gave us an idea how what to use as input for our algorithmic solutions. In our solution and statistical analysis only data after 2004 is used because the Datawarehouse has inconsistent data before this year.

G.1.1. Amount of bank accounts

The total amount of bank accounts in our dataset is 1559534. These are active payment accounts, created before 2004.

G.1.2. Distribution of amount of months accounts are present

We are interested in the availability of transaction data. The older the payment account, the more likely it becomes that this account contains sufficient transaction information to use as training data. The cutoff point at the right end of x-axis is 149, because this is the amount of months between 01-01-2004 and the present. See figure G.1

G.1.3. Average balance per month

The goal of this project is to create predictions for a customer's account balance. Therefore, we will need to look at the account balances of customers. We looked at the average balance of each customer for each month since 01-01-2004. We then normalized these balances by converting them to z-scores. Then we averaged the z-scores of each customer for each month. This gave us a representation of how the average customer's balance develops over the year. See figure G.2

G.1.4. Distribution of amount of transactions for each bank account

This statistic shows the distribution of transaction over payment accounts. We did not take into account new customers with less than ten transactions since we want "active" accounts. This group of customers do not provide us with a sufficient amount of data and would make the rest of the graph difficult to read. See figure G.3

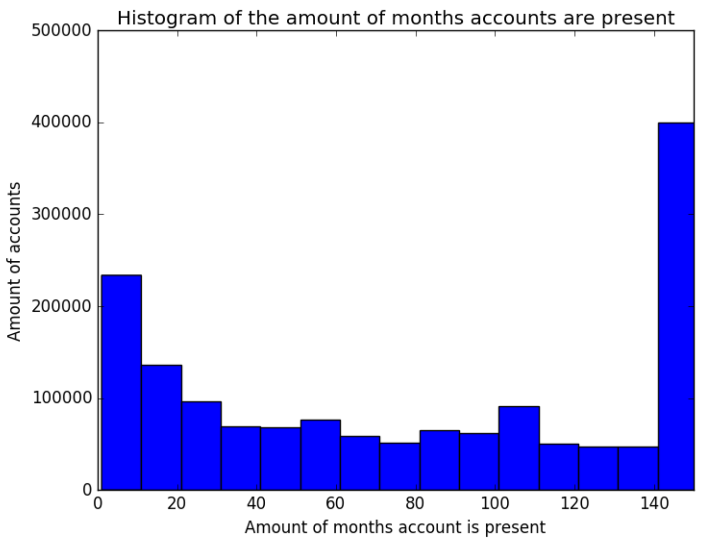

Figure G.1: Histogram showing distribution of subscription length of accounts. On the x-axis you can see the amount of months the account is at this bank, on the y-axis you can see the amount of accounts in the bins. The big bin to the right occurs because every account older than 140 months falls in this bin.

G.1.5. Distribution of average amount of transactions per month for all accounts

The amount of transactions per month is a better measure of payment activity. It was interesting to see how the largest individual group, is the group of people with on average 0-5 transactions per month between 2004 and 2016.

G.1.6. Minimum number of transactions

This graph shows the distribution of the number of accounts with a minimum amount of transactions per month starting from different years shown in different colors.The x-axis was cut off at 60 because the amount of accounts starts becoming too low to be of interest to us. The y-axis was cut off at 600000, this only cuts off the 0-5 bin which is not interesting because this is the group with a minimum of zero transactions per month which is just every account.

G.2. Promising data

With a data warehouse containing over one and a half million payment accounts and detailed information on each of these accounts, it is important to know which information is of use and which is not. This is impossible to know for certain without testing, especially since our machine learning techniques can be seen as black box methods. Therefore it is hard to predict which statistics will be useful and which will not be. During the benchmarking we will extensively test which data is useful and which data does not have a big influence on our results.

G.3. Datasets

In order to create an accurate model based on the data available to us in the data warehouse, we need to construct our own datasets. Due to confidentiality, we cannot publish or share these datasets. The Python scripts and SQL queries used to create them can however be found in our project. First we give an explanation on how we filtered out the data that we want to use and which cut off decisions we made.Then we describe in more detail the datasets that we created and how we divided them in training and validation sets. Also

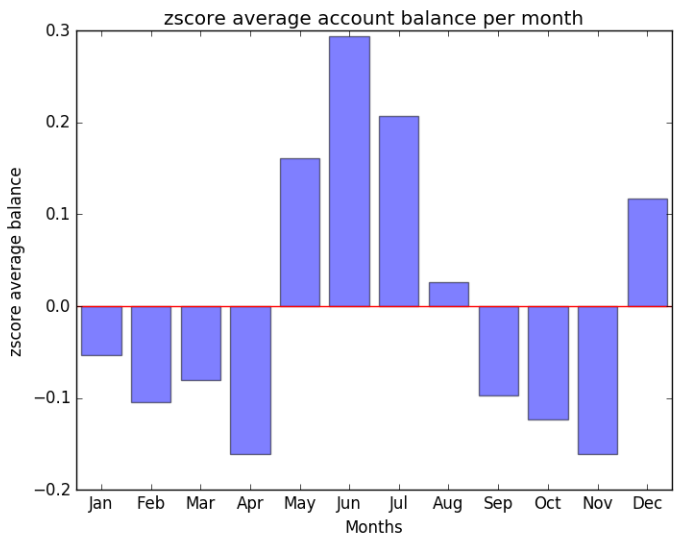

Figure G.2: This graph shows for each month the zscores of all accounts averaged. Here you can see the main trend for the balance on bank accounts of the year.

normalization will be discussed together with an example of what can go wrong and how we fixed this.

G.3.1. Cut off decisions

Before actually using the data as input for our algorithms we made some decisions on what subset of the data we wanted to use. A good example is the fact that some account are inactive and not very interesting to predict for. Secondly some account are too new and do not provide use enough individual behavioral information. The cut offs are the following:

1. All accounts need to have at least 5 transactions for every month of during the last 6 years. (From 2010-01-01 to 2016-04-30)

2. Every account needs to have at least a subscription of 76 months at the bank. (roughly 6 years)

G.3.2. Trainingset

For the machine learning technique SVM we need to devide the data into training and validation sets. Normally you split randomly and you have two sets accordingly. However, we are dealing with timeseries and therefore we do not want to split our data on a unique id, but rather on time. By that we mean that we divided the dataset as following. We know that all the client in the filtered set have to be at the bank from 2010-01-01 to 2016-04-30. So, the training set consists of all the feature vectors from 2010-01-01 until 2014-01-01 which results in a set of 4 years of trainings data.

G.3.3. Validationset

For the validationset we also splitted on time. To make sure no data is leaked from training to validation and because our feature vectors have a time window of 12 months, we opted for selecting the validation set to be only after 2015-01-01 and thus having a gap of one year of data which we will not use for either training or validation. In general the validation set are all feature vectors from 2015-01-01 until 2016-04-30.

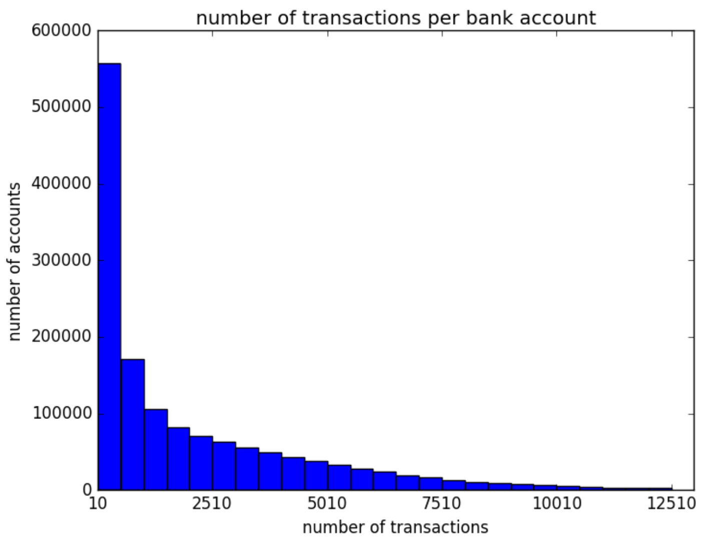

Figure G.3: This graph shows the distribution of the amount of transactions each account has.

G.4. Normalization

It is better to normalize data instead of using the raw value, because this will lead to scaling issues. For the feature vector as input for the algorithms, we thus need to normalize the data.

G.4.1. Z-scores

A common way to normalize data is the Z-score or standard score. A Z-score can be obtained from a raw score through the following formula:

$$Z = \frac{X - \mu}{\sigma} \tag{G.1}$$

Where X is the raw value, μ is the average value and σ is the standard deviation.
This conversion is used for statistics that can encounter scaling issues. In our case, some payment accounts have a much higher balance than others. The Z-score describes the deviation from the mean in terms of the standard deviation. This allows for more accurate and meaningful comparison of individuals across our sample. [24]

G.4.2. Issues

The average and standard deviation (std) used for the z-score is calculated for each account independently. After further analysis of the prediction results we found out that sometimes this normalization can be deceiving. An example is given below:

User x
average balance for all his months: 511,88 euro
standard deviation for all his months: 1914,39 euro

By using the 10 months average prediction we got an MSE of 0.000004. This is considered a small MSE, so we interpreted this as a very good prediction. We further investigated the feature vector and came to the following conclusion; Our error is presented in z-scores which means the error gives the amount of standard deviations the prediction is off compared to the real value. Because we used MSE here, we actually have an error of 0.002 in terms of standard deviation because we need to take the square root. If you then calculate

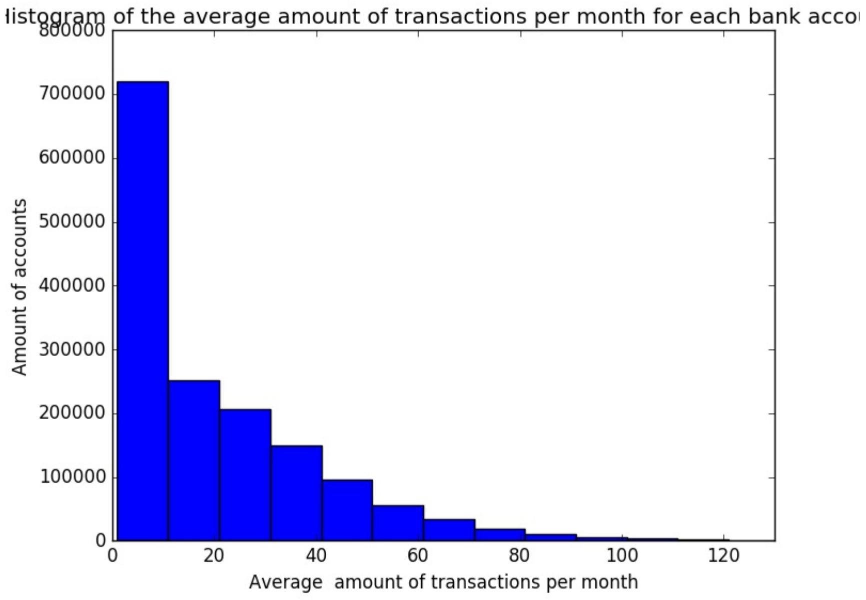

Figure G.4: This graph shows the distribution of average transactions per month per account.

the actual error in absolute value, you get

$$0.002 * 1914,88 = 3.83$$

Which means we predicted this month for this account with an error of roughly 4 euros. If you check his feature vector you can see that 4 euros against the 30-50 that you see in this previous months, is actually not exceptionally good. It is a decent prediction, but the calculated error deceives us into thinking this prediction is near perfect.

The cause of this is bad normalization. The average balance of this account is 511,88 whilst in most months the balance is fluctuating around 50. The high average is caused by a small number of months where the account balance is raised to 14000 euros. These few months skew the mean and std so that the normalization is not representative anymore.

We came to the conclusion that this problem applied to a lot of our accounts an we had to find a way to solve it. Our solution was taking the averages and stds of the middle .90 percentile of the data. This means we first filtered out for each account the top and bottom 5% of balances as outliers. After this filtering the mean and std are calculated. This gives a more representative normalization and enables us to interpret the errors better and make them comparable with errors of other accounts.

G.5. Methods

All transaction data was stored in a data warehouse. The SQL queries we used to obtain these statistics and feature vectors can be found in the queries directory of our project.
The results were subsequently exported to a .csv file by means of the built in functionality in Teradata.
We then used Python scripts to visualize the statistics we had obtained. The **matplotlib** and **pandas** packages offered all of the functionality we needed for visualizing our data. The scripts we used to create our visualizations and feature vectors can also be found in our project, in the visualization directory.

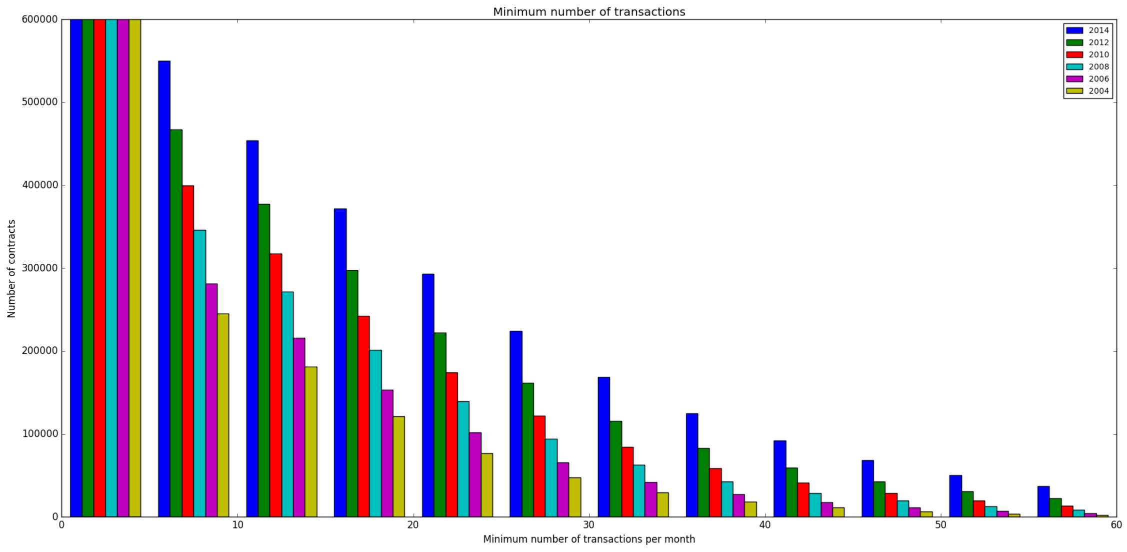

Figure G.5: This graph shows the amount of account thats have a certain minimal amount of transactions for each month since the year you can find in the legend. On the x-axis you can find the minimal transactions. On the y-axis you can find the amount of account for each year with a different color.

H
Info Sheet: Time series predictions for bank account balances

Client: A large bank holding company
Presentation date: 24th June 2016

Project description

The purpose of our project was to develop a model that could predict the bank balances of customers of our client. For this project our research question was: How well can we predict bank balances?
Our client is a large Dutch bank holding company with five distinct brands. The IT and Change department makes sure the company keeps up with new IT developments.
The core challenge of this project was the development of an entire system from scratch. This began with retrieving the data from the datawarehouse and leading all the way up to the visualization of our final output. Our main research was used to find a suitable machine learning approach for our model. We compared multiple existing approaches and finally decided which ones to eventually test. During our project we came across some unexpected difficulties, as a main consequence we had to change our project plan during the fifth week. More information about the process can be read in our final report, Chapter 13
The eventual product we have created is a framework that enables our client to compare machine learning and non-machine learning techniques. In this framework we have implemented and compared multiple models for account balance prediction. During the project we compared two different Python libraries that can be used for machine learning. This comparison supports our recommendations to the client for future machine learning implementations. In addition to that, we have given recommendations for the usage and improvements on our regression model as well.

Team members

Bernd Kreynen		b.l.l.kreynen@tudelft.nl
Interests:	Data science, software engineering, statistics	
Role:	Chief code quality.	
Chantal Olieman		c.olieman@tudelft.nl
Interests:	Algorithmics, data science, optimization	
Role:	Chief communications.	
Felix van Doorn		f.a.vandoorn@tudelft.nl
Interests:	Software testing, software engineering, data science	
Role:	Chief versioning and testing.	
Max Spanoghe		m.spanoghe@tudelft.nl
Interests:	Data science, optimization, statistics	
Role:	Chief scrum.	

Contributions The complete list of contributions can be found in our Jira, this project is private for the sake of confidentiality. Please contact our team if needed.

Client and TU Coach

Wouter Poncin	Client	System Desginer IT & Change
Thomas Abeel	TU coach	Delft Bioinformatics Lab

A censored version of the final report for this project can be found at: http://repository.tudelft.nl

Bibliography

[1] 2016. URL `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html`.

[2] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[4] Armando Bernal, Sam Fok, and Rohit Pidaparthi. Financial market time series prediction with recurrent neural networks. 2012.

[5] Carl Friedrich Bolz, Lukas Diekmann, and Laurence Tratt. Storage strategies for collections in dynamically typed languages. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages &#38; Applications*, OOPSLA '13, pages 167–182, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2374-1. doi: 10.1145/2509136.2509531. URL `http://doi.acm.org/10.1145/2509136.2509531`.

[6] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach Learn*, 20(3):273–297, sep 1995.

[8] Steve R Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14, 1998.

[9] G. D. Hutcheson. Ordinary least-squares regression. *Hutcheson, The SAGE Dictionary of Quantitative Management Research*, 2011.

[10] Kyoung jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55 (1-2):307–319, sep 2003.

[11] Deng Julong. Introduction to grey system theory.

[12] Erdal Kayacan, Baris Ulutas, and Okyay Kaynak. Grey system theory-based models in time series prediction. *Expert systems with applications*, 37(2):1784–1789, 2010.

[13] VV Kondratenko and Yu A Kuperin. Using recurrent neural networks to forecasting of forex. *arXiv preprint cond-mat/0304469*, 2003.

[14] Steven Lloyd. Least squares quantization in PCM. *IEEE Transaction on Information Theory*, 28(2):129–137, 1982.

[15] David JC MacKay. *Information theory, inference and learning algorithms.* Cambridge university press, 2003.

[16] Nijolė Maknickienė, Aleksandras Vytautas Rutkauskas, and Algirdas Maknickas. Investigation of financial market prediction by recurrent neural network.

[17] Kalyan MONDAL and Surapati PRAMANIK. The application of grey system theory in predicting the number of deaths of women by committing suicide-a case study. *Journal of Applied Quantitative Methods*, page 48, 2015.

[18] Sayan Mukherjee, Edgar Osuna, and Federico Girosi. Nonlinear prediction of chaotic time series using support vector machines. In *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pages 511–520. IEEE, 1997.

[19] Klaus Robert Muller, Alexander J Smola, Gunnar Ratsch, Bernhard Scholkopf, Jens Kohlmorgen, and Vladimir Vapnik. Using support vector machines for time series prediction. *Advances in kernel methods—support vector learning, MIT Press, Cambridge, MA*, pages 243–254, 1999.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[22] Ruey S. Tsay. *Analysis of Financial Time Series, Second Edition.* John Wiley & Sons, Inc., 2005.

[23] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[24] Robert Walrath. *Encyclopedia of Child Behavior and Development*, chapter Standard Scores, pages 1435–1436. Springer US, Boston, MA, 2011. ISBN 978-0-387-79061-9.