Universität St.Gallen

Data Science Project Report

Santander Customer Transaction Prediction

Authored by:

Aydin Kaan, Goessi Sandro, Gubler Fabian, Schnydrig Sven

A report submitted in partial fulfillment for the
Data Science Course in the Master of Computer Science (MCS)

Data Science - 7,854,1
Institute of Computer Science (ICS-HSG)

December 2022

"One of the common fallacies is that data is opposed to intuition. Data is a tool for enhancing intuition " - Hilary Mason

1 Introduction

1.1 Project Definition

Santander Bank is a Spanish multinational financial services company headquartered in Madrid, Spain [1]. According to the "list of largest banks", it is one of the largest banks in Europe and ranks 19th globally by total assets [2]. Santander is known to be innovative and has been recognized for its digital banking capabilities. For this data science challenge from the year 2019, they invited the global data science community to help them predict which of their customers will make a transaction in the future, irrespective of the amount. To solve this problem, they have provided the Santander Customer Transaction Prediction data set, which is a collection of financial transaction data.

1.2 Task Description

Our task is to train a model that can accurately predict, based on a number of anonymized features, whether a Santander customer will make a transaction in the future or not. This is a typical binary classification problem, with the class labels being "Yes" and "No". To address such binary classification problems, there exist a variety of machine learning algorithms that can be utilized, including logistic regression, decision trees, naive Bayes, and neural networks. Our ultimate goal is to evaluate and select the model with the overall best performance and most effective predictive capabilities [3].

1.3 Business Application

This specific task is valuable for Santander as it could help them inform risk management, target marketing efforts, and improve customer retention, potentially increasing their revenue. Similarly, if a bank can predict which customers are at risk of not making a transaction, it can take steps to prevent this outcome, such as offering incentives or providing them additional support.

1.4 Motivation for Data Set Choice

We have chosen this data set because it has a very interesting problem statement with practical business relevance. Also, it is a relatively straightforward data set with limited size and only numerical features, which facilitates an easier application of machine learning models.

1.5 Model Evaluation

As stated in the terms of the challenge, the performance of each model on the test set will be evaluated using the Area Under the ROC Curve (AUC) metric. The AUC measures the performance of a binary classifier by comparing the true positive rate and false positive rate of the classifier. It ranges from 0 to 1, with a higher AUC value indicating better classifier performance [3].

1.6 Baseline

The current baseline AUC value that serves as a point of comparison is 0.92727 for the public and 0.92573 for the private leader board. This score was achieved by the "Wizardy" team, which used a blend of a neural network and LGBM model using pseudo-labels and data augmentation. For their feature engineering that led them to their solution, they were using the count of unique values, which proved to be the key to their success [3].

2 | Methodology

In order to develop a predictive model for the Santander Customer Transaction Prediction data set, we proceeded by implementing the following steps:

1. Data exploration
2. Pre-processing
3. Feature engineering
4. Data augmentation
5. Algorithm selection

2.1 Data Exploration

Data exploration is an essential step in the data science process, as it allows us to get a better understanding of the data we are working with and their characteristics. By exploring the data, we can identify potential issues or biases, understand the relationships between different features, and gain insights that can inform the development and evaluation of our models. It can also help us prepare the data for learning by detecting and addressing any missing or duplicate values, scaling or normalizing numerical data, and encoding categorical data if present.

We will first provide a brief overview of the Santander Customer Transaction Prediction data set we are using for our analysis, followed by a more detailed exploration of its features and their distributions.

2.1.1 Data Set Overview

The data set consists of three files: the training set *train.csv*, the test set *test.csv*, and a sample submission file *sample_submission.csv*. Both the train as well as test set contain 200,000 rows, each sample representing an individual customer observation. The Santander training data set consists of 202 columns, and its test data set has only 201 columns.

The additional feature in the training set is the target feature column, which is the class (label) of a row that we have to predict in the labeled data set. The target feature is therefore binary (0 or 1) and of type int64. This is a classification problem of supervised machine learning with two possible classes: non-customer-transaction (0) and customer transaction (1). In total both the train and test data sets, consist of 200 feature variables, all of which contain only numerical data of type float64. In the first column, we can find the *ID_code* which contains text (string). The *ID_code* feature is a unique id of the row. It does not have any effect on the dependent target variable. All other 200 features are anonymized and named from var_0 to var_199. So we cannot infer any contextual real-world meaning which makes it impossible to apply any industry-specific domain knowledge.

2.1.2 Missing Data and Correlations

The analysis of the train and test data sets reveals the absence of missing data points and null values. Features are also not correlated, which impedes the use of feature engineering methods based on principle component analysis (PCA) to remove redundant features because of multicollinearity. There are no independent features that are correlated with each other higher than 0.01, indicating a very low correlation. The highest correlation that can be found in the test set is 0.00986 between the variables var_139 and var_75. These findings are likely a result of pre-processing conducted by Santander. After considering all these relevant factors, we decided as a starting point to utilize all available features in our first models.

2.1.3 Distribution of the Target Variable

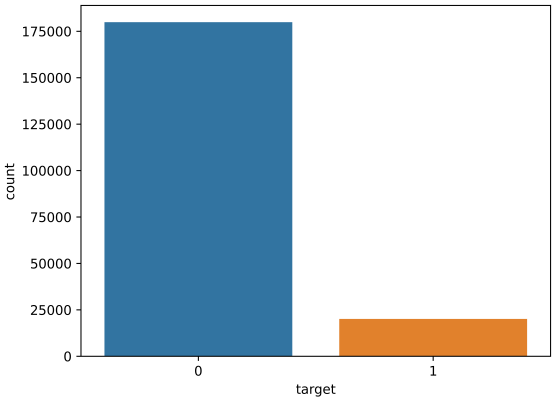

FIGURE 2.1: Target Column

The training data set is very imbalanced, with approximately 90% of the observations belonging to class 0 (non-transaction) and only 10% belonging to class 1 (transaction). Out of 200,000 rows, 179,902 are class 0 and 20,098 are class 1. This imbalanced distribution may impact the performance and generalizability of our model. To address the imbalanced nature of the data, we employed an oversampling technique. This was necessary to avoid bias towards the majority class.

2.1.4 Distribution of Feature Variables

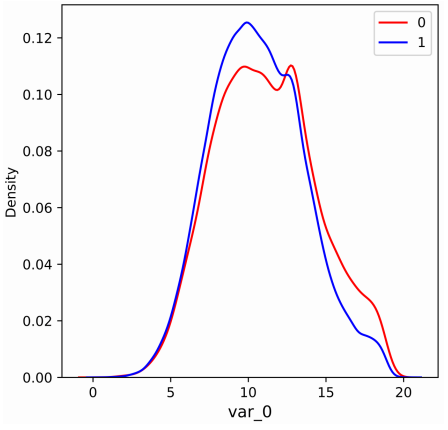

FIGURE 2.2: Density Distribution (ex: var_0)

The distribution of variables between train and test is quite similar. Therefore, for a given feature the minimum, maximum, mean, and standard deviation values for the train and test data look quite identical. In comparison, the mean values are often distributed very differently across different feature columns. A comparison in the value distribution of the feature columns of the buyers target $= 1$ and non-buyers target $= 0$ reveals some differences. The differences do not appear in all of the feature variables, but in many of them. [4].

2.1.5 Probability Function Graphs

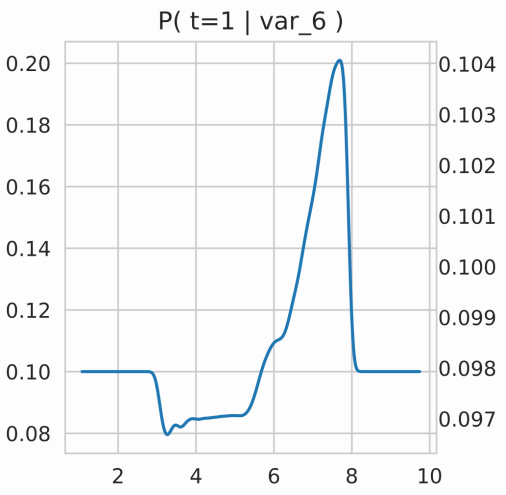

FIGURE 2.3: Probability Function

Based on the density distribution graphs, we plotted probability graphs for all features to identify the values at which the probability of a customer transaction is the highest for a particular feature column. This information can also be utilized in feature engineering to optimize the performance of our model because its predictive nature shows for which values of a certain feature the transaction chance is the highest [5].

2.1.6 Unique and Duplicated Values

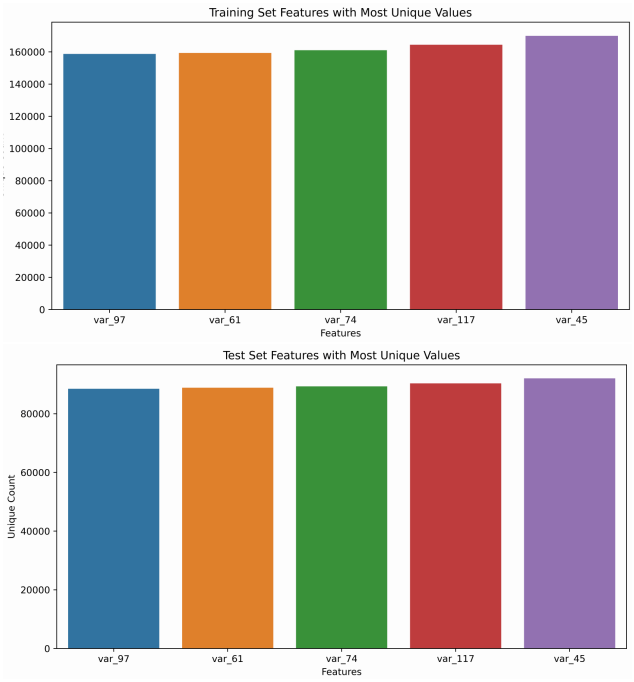

FIGURE 2.4: Features with most unique values

Another interesting pattern in our data is that each feature in the training set has more unique values than features in the test set. Our research showed that this observed phenomenon is likely due to the presence of synthetic records in the test set [6].

2.2 Pre-processing:

Real vs. Synthetic Data

During the data exploration process, it was seen that the statistics of the training set and the test set are very similar. However, the distribution of the number of unique values (across features) is significantly different between the training set and the test set (s. graph 2.4). It appears that the test set consists of both authentic and synthetic samples generated by sampling techniques from the distribution of the real authentic samples.

Based on this observation by YAG320 [6], we tried to identify the synthetic samples. We did this by looking at an observation and going through all of the features and checking whether each feature value is unique or not. If at least one of the feature values is observed to be unique, it can be inferred that the sample is very likely to be authentic. If a given observation does not have a single unique value, we assume it is a synthetically generated sample. While we cannot conclude this with absolute certainty, based on the distribution of this data set, the probability of this not being the case is quite low, and therefore we proceed with this assumption. Applying the described method to our test data set, we can create a split of our test data into "real" and "synthetic" observations, which cover 100,000 samples each (i.e., each set contains half of the entire test data set).

2.3 Feature Engineering

We used a multi-data set approach in which we engineered additional features based on our starting training and test sets. Almost all feature engineering approaches we consider depend on how we concatenate the data before applying the respective methods. For that reason, we combine our training and test sets before we extract new features. In this case, the test set portion does not include the synthetic samples, as that would have a skewing impact on the results, especially when calculating the unique value counts.

As already discussed in section 2.1, the features in the training and test data set are anonymized and not correlated. This makes it harder for us to use some methods for feature engineering, such as principal component analysis (PCA) and generating new features based on domain knowledge. Thus, we focused on following methods for feature engineering:

- Unique counts
- Reverse features (incl. partial removal)

We sequentially applied these feature engineering techniques to generate a total of three data set versions. Version 1 and 2 have two sub-versions,

depending on the method to calculate the unique counts. Thus, we have 5 data sets in total:

- **Version 0:** No feature engineering
- **Version 1:** Unique counts
- **Version 2:** Unique counts & reverse features

2.3.1 Unique counts

Through experimentation with the data set, we noticed the information on unique count is quite impactful when training the model. However, there are different ways one can integrate the information of unique counts:

- Binary classification whether value is unique
- Numerical value depending on "uniqueness"

Deep-Dive: Binary Classification
After splitting the test set into real and synthetic samples, we concatenated the real samples with the training set. After doing so, we checked, for each variable, whether the sample value is unique or not. We added the results as a new column in the data set. Thus, we increased our number of features from 200 to 400 (i.e., adding one additional column for each feature which indicates whether this respective feature is unique (1) or not (0)).

Deep-Dive: Multi-layer feature generation
Our second approach is based on a more sophisticated approach to include the uniqueness of a variable in our model training. It is based on the kernel developed by R. Nandepu [7] and works as follows:

- Similar to before, firstly, we created a data frame consisting of the training samples and the authentic test samples
- We then created an additional column for each feature by taking the minimum of the value 10 and the respective count value
- Then for each variable, we developed three additional variables based on the following:
  - sum: Set to 0 if count value is less than 2
  - sum2: Set to 0 if count value is less than 3
  - sum3: Set to 0 if count value is less than 5

At the end, we receive a new test and training set, both of which include 1,000 features.

2.3.2 Reverse features

Based on the probability figures we showed earlier, we noticed that some graphs are directed to the right, while others are directed to the left. Some are neither directed to the right nor to the left.

To exploit this observation, we identified the features with the highest probability on the right and flipped them so that for all features the highest probability will be on the left. Furthermore, we dropped the features where the probability is neither directed to the right nor to the left.

To do so, we leveraged the kernel created by SIBMIKE [8]. In total, we inverted the direction of approximately half of the features by flipping them from the right to left. In addition, we removed around 15 features based on our argumentation above.

2.4 Data Augmentation

As observed during data exploration, only ten percent of the samples show customers that completed a transaction (i.e., target variable is equal to 1). This suggests that there is an imbalance in the data in terms of the target variable. To overcome this challenge, we need to carefully implement data augmentation techniques.

We have looked at various data augmentation techniques, such as undersampling, oversampling, and class weighting. Naturally, all techniques have their advantages and disadvantages. Nevertheless, we decided to use oversampling for the following reasons:

- In undersampling, we would need to "sacrifice" a part of the information in the data, which we wanted to avoid
- Class weights do not always work very well with some algorithms (e.g., tree-based methods) and can be difficult to set up
- One potential disadvantage of oversampling is that it can lead to a data set that is no longer representative of the original population (i.e., if synthetic data do not accurately reflect the characteristics of the underrepresented class). As a result, the model trained on an oversampled data set may not perform as well on unseen data

This may be one of the reasons why oversampling did not improve our score for the CNN and we therefore didn't use it there. For the LGBM on the other hand it improved our score and we applied oversampling to the minority class (Target = 1) by three times, thereby increasing the overall number of samples and the proportion of samples with Target = 1 in the training data set. We also doubled Target = 0 samples, so the new sample distribution is similar to the original.

2.5 Algorithm selection

	V0: without feature eng.	V1: with unique count		V2: with unique count and reverse feature	
		Binary	Multi-Layer	Binary	Multi-Layer
Logistic Regression	Yes		No		No
Naive Bayes	Yes		No		No
ANN	Yes		No		No
CNN	Yes		Yes		Yes
Light GBM	Yes		Yes		Yes

TABLE 2.1: Models chosen with corresponding version for feature engineering.

Table 2.1 represents the models we have applied during our group project and the corresponding version of the feature engineering we used. As stated at the beginning, this competition aims to explore the classification problem, which customers make a specific transaction in the future - irrespective of amount. Given the type of problem, we have a multitude of options to choose. We selected the models as shown in table above.

For both the Logistic Regression and Naive Bayes, we applied the default values. Given the higher results achieved for the Light GBM and CNN, we will only apply the data sets with the engineered features (i.e., version 1 and 2) for those and not for the other three algorithms tested.

Deep-Dive: Neural Networks

We started with a feedforward neural network and explored different variations of the number of hidden layers (1-8) and neurons (2-500), optimizers (Gradient Descent, Stochastic Gradient Descent, Adagrad, Adadelta, RMSprop and Adam), learning rates (0.0001-0.1) and also explored the Wide and Deep Neural Network architecture which was presented in a research paper by Google in 2016 [9]. After generating the additional features as described in the feature engineering section, we realized that our ANN didn't benefit much from the extraction of those extra features and focused on the development of a new convolutional neural network.

The CNN allowed us to one time use the original

features as input and the additional adjacent inputs based on the uniqueness of those features as explained in the features engineering section. Another advantage of convolutional neural networks is that it finds one pattern among all the variables that appears similar, which is important for generalisation and helps avoid overfitting.

For the CNN we started out with a pretty basic configuration which starts with batch normalisation followed by two fully connected linear layers. The first linear layer takes a single value of a feature plus the adjacent value(s) about the uniqueness of this feature (according to the different methods described in the feature engineering section) and runs it through the number of hidden dimensions. So in other words, every feature plus the additional uniqueness information about this feature is its own example and gets mapped to the number of hidden dimensions. The second layer then gives us the final prediction by taking the output of the first linear layer and mapping the output to a dimension of 1.

We did not do any systematic hyperparameter tuning for the CNN and rather followed an engineering-driven approach where we manually tested out different things until we arrived at our final configuration. For example we tested the number of hidden dimensions from 5 to 500 and also tried out various optimizers (Gradient Descent, Stochastic Gradient Descent, Adagrad, Adadelta, RMSprop and Adam) and learning rates (0.0001-0.1) to check whether convergence happens in reasonable time and we don't overshoot. Overall, 200 hidden dimensions, and Adam, with a learning rate of 1E-3, produced the best results.

Deep-Dive: Light GBM

We would like to provide an overview of the most relevant hyper-parameters we used for the Light GBM. In total, we experimented with five different combinations based on our data set with the feature engineered version based on multi-layer approach. The combination with which we achieved the highest score was further investigated with the other data sets. In our case, this was the combination 5 (i.e., highlighted in blue).

	C1	C2	C3	C4	C5
learning_rate	0.1	0.1	0.01	0.01	0.01
num_leaves	31	15	31	15	15
max_depth	-1	-1	-1	-1	-1
min_data_in_leaf	20	80	20	80	50
min_sum_hessian	n/a	10	n/a	10	10
bagging_fraction	1.0	0.6	1.0	0.6	0.6
bagging_freq	0	5	0	5	5
feature_fraction	1.0	0.05	0.5	0.5	0.05
lambda_l1	0.0	1.0	0.0	1.0	1.0
Rank of Scores	5	3	4	2	1

TABLE 2.2: Light GBM Hyper-Parameters: List of different combinations

Deep-Dive: Linear Blending

Once we had the final results, we created a linear blend with various weights based on the best results from the Light GBM and Convolutional Neural Network. For example, for a given prediction, we would take the prediction of LGBM multiplied by the weight of the LGBM model plus the prediction of the CNN multiplied by the weight of the CNN model and divide this number by the sum of the weights.

3 | Results

Following our methodology, we trained our model accordingly. As the data is unbalanced and the Kaggle competition was ranked by the achieved AUC we'll also focus on this metric in the results section.

The following table represents the public AUC score we achieved when applying our models to the test data set and submitting it on Kaggle without any feature engineering.

Model	AUC
Logistic Regression	0.61623
Naive Bayes	0.67519
ANN	0.81197
CNN	0.89531
LGBM	0.86427

Table 3.1: Public score results on Kaggle from different initial models

As you can see the performance of the CNN and LGBM was far superior to the other models, as the other models (especially the logistic regression) were not complex enough to capture the patterns of this data set fully.

For the training of our models after feature engineering, we focused on the CNN and LGBM model and obtained the following public score when submitting on Kaggle. Please note that for Light GBM the results are based on the hyper-parameter combination 5.

Model	Data	AUC
CNN	Version 1 - Binary	0.91914
CNN	Version 1 - Sum	0.92008
CNN	Version 2 - Binary	0.91743
CNN	Version 2 - Sum	0.91812
LGBM	Version 1 - Binary	0.90560
LGBM	Version 1 - Sum	0.92361
LGBM	Version 2 - Binary	0.90337
LGBM	Version 2 - Sum	0.92295

Table 3.2: Public score results on Kaggle from different models

We can see that both the LGBM and CNN model performed the best in version 1 with the multi-layered features about the uniqueness of the respective feature.

When we compare our best score to our baseline (i.e., public score), we can see that we have narrowly missed the baseline (i.e., first place solution from Kaggle) and we would need some more improvements at the third digit to match or surpass the baseline. It is also important to notice that the feature engineering was essential for getting from a score of 0.895 to 0.92 and above. The linear blend with a ratio of 5:1 (LGBM:NN) improved our score slightly.

	Best Score	Delta baseline
Convolutional NN	0.92008	-0.00719
Light GBM	0.92361	-0.00366
Linear Blend (5:1)	0.92385	-0.00342

Table 3.3: Final Results comparison to baseline

4 | Discussion

4.1 Discussion of results

From the results shown in the previous section, we came to a few conclusions.

Section 2.5 demonstrated that Logistic Regression and Naive Bayes performed slightly higher than 60% (in terms of AUC), but are significantly lower than the results with the neural networks and Light GBM. For that reason, we only focused on neural networks and Light GBM models.

Furthermore, the results of both the Convolutional Neural Network and Light GBM perform above 92% (in terms of AUC) and thus are relatively close to the baseline.

Looking more in detail at the results from Light GBM and Convolutional Neural Networks, we can see that our engineered features did help us to achieve a better score. Those engineered features really were the key for getting from an AUC of 0.9 to 0.92 and above. When we compare our different data sets to each other, we observe that the version 1 with multi-layered feature engineering performed the best with both Light GBM and the CNN. These results indicate two things:

- Feature engineering with the multi-layer approach performed better compared to the approach without feature engineering and the one with binary classification. The impact was more pronounced for the LGBM model as the CNN performed only slighlty better with the multi-layer approach (vs. binary)
- Reversing the features had a slightly negative impact on the final score
- Blending our results from Light GBM and CNN linearly improved our scores slightly. To do so, we applied a numerical search approach and found that the ratio 5:1 (based on LGBM:CNN) was the combination that improved our final score the most

Looking retrospectively, we can state that our final models did not fail. For some of the initial neural networks our algorithm did not converge initially because we set the learning rate too high, but obviously this was easy to fix and we simply adjusted the learning rate until we saw a smooth convergence.

Some of the initials models such as logistic regression did not deliver satisfying results as they were not complex enough to capture the patterns of our data fully. However, with some modifications, we also see potential to increase their scores, but we doubt that ultimately we would perform as high as we did for Light GBM and neural networks.

As the variables were anonymized by Santander and the goal was to maximize performance in this Kaggle competition, it made sense to sacrifice interpretability with more complex models, such as Light GBM and Neural Networks for the sake of superior performance. The ANN also was not able to match the performance of the CNN due to the reasons already mentioned in section 2.5

4.2 Outlook

Despite our relatively high score, we had some more ideas along several dimensions that we wanted to explore to improve our solution, but could not given the time constraints. After getting a score above 0.92 it becomes extremely difficult to further improve your model and you fight for

improvements at the third decimal. Nevertheless we tried to identify some areas on which we would work if we would have an additional three months to get those improvements.

More advanced blend
We simply used a simple linear combination of the Light GBM and CNN models for our final prediction. Using a neural network to blend instead of a simple linear blending could have further increased our score.

Other feature engineering methods
As mentioned earlier, there are different way to include the information on unique counts in our model training. During our project, we focused on two ways. Given more time, we could have have tried other ways to do so, thereby potentially increasing our score.

Data Augmentation
While we performed different data augmentations, it would be possible to further explore different data augmentations techniques and potentially identify a better sampling ratio or improved technique.

Pseudo-labelling
Apparently pseudo-labelling improved the score by 0.0005 for some participants which is something we did not try and could be done in the future.

Scaling
For the CNN we simply used batch normalisation and for Light GBM no data scaling at all. It is possible that using different kinds of data scalers like MinMax or Standard Scaler could have improved our results further.

Extensive Hyperparameter Tuning
We believe it would have been possible to further increase our results by using a systematic and extensive hyperparameter search. For Light GBM we only used a small subset of potential hyperparameters and for the neural networks we followed more of an engineering-driven approach by testing out different variables manually and seeing how it affects our model. For the LGBM we could have used grid search, random search or Bayesian optimization. For the neural networks we could have tried to find a better configuration with the Keras Tuner for Tensorflow or Ray Tune for Pytorch.

Bibliography

[1] Santander bank. URL https://en.wikipedia.org/w/index.php?title=Santander_Bank&oldid=1118791810. Page Version ID: 1118791810.

[2] Ahmad Rehan Yamaguchi Yuzo, Terris Harry. The world's 100 largest banks, 2022 | s&p global market intelligence. URL https://www.spglobal.com/marketintelligence/en/news-insights/latest-news-headlines/the-world-s-100-largest-banks-2022-69651785.

[3] Santander Banco. Santander customer transaction prediction. URL https://kaggle.com/competitions/santander-customer-transaction-prediction.

[4] Raimondi Federico. Exploratory data analysis - santander customer transaction prediction. URL https://federicoraimondi.github.io/myProjects/Santander_Customer_Transaction_Prediction/Exploratory_Data_Analysis/Data%20Exploration.html.

[5] Deotte Chris. Modified naive bayes - santander - [0.899]. URL https://kaggle.com/code/cdeotte/modified-naive-bayes-santander-0-899.

[6] YAG320. List of fake samples and public/private LB split. URL https://kaggle.com/code/yag320/list-of-fake-samples-and-public-private-lb-split.

[7] Nandepu Raghun. Santander customer transaction prediction: An end to end machine learning project. URL http://bit.do/medium-vidhya.

[8] SIBMIKE. Are vars mixed up time intervals? URL https://kaggle.com/code/sibmike/are-vars-mixed-up-time-intervals.

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide amp; deep learning for recommender systems, 2016. URL https://arxiv.org/abs/1606.07792.

Declaration of Authorship

We declare that this report, and the work presented in it as our own. We confirm that:

- This work was done wholly while in candidature for a degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by ourselves jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signatures:

Aydin Kaan

Goessi Sandro

Gubler Fabian

Schnydrig Sven