HAW HAMBURG

Grundprojekt

Tasmin Herrmann

Development Environment for Buy Predictions with Machine Learning

Supervision by: Prof. Dr. Kai von Luck
Submitted on: November 28, 2018

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department Information and Electrical Engineering

Contents

Abstract

I developed an environment for modeling buy predictions. Therefore, I used the yoochoose data sets and ran through the phases of CRISP-DM. For the implementation, I have relied on parallel execution on processors and graphics cards to make optimum use of given hardware resources. I used data balancing methods since the data set contains almost 95% of non-buyers. An under- and an oversampling method were used to better predict the buyers with multilayer perceptron models. The methods worked almost equally well. The implementations can be adjusted, this is especially necessary for further feature engineering and model development of buy predictions.

Keywords: Data mining; predictive analytics; recommender systems; RecSys Challenge 2015

1 Introduction

1.1 Motivation

Data is constantly generated and collected on the Internet. Companies are trying to take advantage of this data. Especially in e-commerce a lot of useful data is collected and used for analysis. One wants to get to know and understand the customer. This knowledge is then used to bind customers and recommend suitable products. Amazon already knows before the customer has bought what he will buy. They wants to use this knowledge to store the articles near the customer before he purchases them. So, a very fast delivery time is possible. But for this scenario you need a lot of data about the customer, the products and above all the behavior of the customer. In this paper I am generally dealing with the buy prediction and not with anticipatory shipping as Amazon calls their process. But the buy prediction is part of this process. In this project, the goal is to build a development environment for experiments in modeling buy predictions.

1.2 Outline

In Chapter 2, I look at buy prediction. I describe the context and questions of buy prediction. Two actual papers about buy prediction are presented. I then look in detail

at two papers on the data set of the Recommender System Challenge from 2015. In chapter 3 I describe the creation and testing of the development environment. For this I use CRISP-DM and describe the results of the necessary phases of the process model. Chapter 4 summarizes the work and gives an outlook.

2 Buy Prediction

The digital library of the Association for Computer Machinery has 1518 search results for buy prediction and 1534 for purchase prediction in 2010. There are 2474 search results for buy prediction and 2120 for purchase prediction in 2017. Over the years, the number of publications on the subject has increased almost annually.

There are different issues in the context of buy prediction. In the context of anticipatory shipping you want to know if a user buys something. Then you want to know what the customer will buy and when the customer will buy it. Also, you want to know where the order should be shipped to.

Buy prediction belongs to predictive analytics. Predictive analytics [7] includes a variety of statistical and analytical methods used to develop models that predict future events or behaviors. The shape of these predictive models depends on the behavior or event they are predicting, but most predictive models calculate a score with the likelihood the target behavior or event occurring. To create a predictive model data mining can be used. Data mining is used to identify patterns, trends or relationships among the data. These techniques are based on statistical methods like regression or time-series models. These techniques allow to gain non-obvious knowledge. Application examples are fraud detection in credit institutes [2] and predictive maintenance [6]. Predictive maintenance connects information from different devices and machines in real time to improve maintenance processes by facility managers.

Buy prediction is assigned to the context of recommender systems. Recommender systems attempt to predict the preference a user would give to an item. This information can be used to make suggestions to the user. Therefore, collaborative filtering and content-based filtering are mostly used. Collaborative filtering searches users with the same pattern of behavior like the target user and uses this information to predict the behavior of the target user. Content-based filtering searches similar items to items a user likes. The buy prediction does not only aims to predict an interest in an item, but also,

the action of purchasing an item. There are several questions that are currently being addressed in buy prediction research.

Xia et al. [9] tried to bridge the gap and improved recommendation systems by explicitly modeling consumer buying decision process and corresponding stages. They used a multi-task learning model with long short-term memory networks (LSTM) to learn consumer buying decision process. In this way, their model can capture user intentions and preferences, predicts the conversion rate of each candidate product and makes recommendations accordingly.

Bhagat et al. [1] presented their approach for repeat purchase recommendations at Amazon. They wanted to recommend products from the purchase history of a customer which he would likely buy again. They used the specific set of purchased products of a customer for the recommendation.

This is a small insight into current research, which looks at other questions in buy prediction than ours. That is why I now look at research that also deals with the data set I use and that solves similar problems.

Recommender System Challenge

The Recommender System Challenge, also called RecSys Challenge, is a contest in the context of the ACM Conference Series on Recommender Systems. In 2015 the assignment was in the context of buy prediction. The given data was a sequence of click events performed by some user during a typical session in an e-commerce website. The goal was to predict whether the user is going to buy something or not, and if he is buying, what would be the items he is going to buy. The task could therefore be divided into two sub goals. Task one was to find out if the user is going to buy items in this session. The second task was to find out if he buys what are the items he is going to buy.

The winner were Romov and Sokolov [8]. They described in their paper the winning approach for the RecSys Challenge 2015. First, they described the training and test data set. Each session is a sequence of click events with timestamp, item ID and category. The objective function assigned the purchased items to the session. They noted that sessions in training and test sets are not separated in time, although it is considered good practice for recommendation problems to predict future events based on a training set from the past. In their estimation, this leads to the fact that characteristics of time

are very useful for the question. A buying rate was defined as the fraction of buyer sessions in some subset of session and examined the buying rate in time. They found out that users bought more often during midday than in the night hours, and more likely on the weekend than during the week. Furthermore, the buying rates vary significantly among sessions with specific most clicked item. They noted that it is very important to predict whether the user will buy at least one item, because the data set is unbalanced and has only 5.5% sessions with buy events.

Therefore, Romov and Sokolov build a two-stage classifier. One classifier for the purchase detection and one for the purchased item detection. They used two groups of features to train the classifiers. The first group of features described a session and the second group described an item in a specific session. In the first group are features like the month of the session start, the day of the session start and the session duration in seconds. In the second group are features like item ID and the total number of clicks in the session for the given item. The purchase detection classifier uses only session features and the item detection classifier uses both groups.

When choosing the classification method, Romov and Sokolov did not want to resort to popular methods like XGBoost or ensemble in sklearn, because they must transform their dozens of categorical features, e.g. by one-hot encoding. They used a MatrixNet which is an implementation of gradient boosting over oblivious decision trees with careful leaf values weighting based on their variance. The MatrixNet uses hash tables as base learners for categorical features.

90% of the data was used for model training and 10% for model validation. Romov and Sokolov took the validation data to find the best thresholds for the binary classifications. For the winning approach, they optimize the item threshold by mean Jaccard similarity and the purchase threshold by the competition score on the validation set. The purchase detection classifier achieves a AUC value of 0.85 on the validation set. It has 16% precision and 77% recall for optimal thresholds. The item detection classifier achieves mean Jaccard measure of 0.765.

Cohen et al. [4] also took part at the contest. They achieved the 14th place in the challenge's final leaderboard. First, they described the data and the patterns they found. The most important findings were:

- buying events occur mainly between 6:00 am and 8:00 pm

- buying events are most likely to occur on weekends rather than on weekdays, with Tuesdays being the weakest day of the week for purchases
- if the item has been clicked on a few times in a session, the probability that the item will be bought is high
- when users spent more time on an item in a session, the probability of buying this item increases
- longer sessions lead to a purchase

This is only a selection of findings. Cohen et al. found a lot more with their data analysis. This information was used to engineer new features. They also divided the features into two groups. The first group called Features by Session (FS) provided various aspects of a session. They started with 16 features and ended with a list of 13 features. They used several feature selection algorithms (CfsSubset, OneR, InfoGain, and GainRatio) and applied a simple voting mechanism to make their decisions. The second group of features is called Features by Item in Session (FI). It provided various aspects on an item in a specific session to describe if the item is going to be bought. They started with 56 features and selected 22 features with the same feature selection process as for FS. Before model training, they under sampled the majority class to create a balanced training set. Given that the training data had 509,696 positive sessions, they randomly sampled 509,696 negative sessions from the entire training data and used all of these sessions as their training data.

Two differnet approaches were implemented. The first and easier approach was to classify each item in a session as a buy or not. Cohen et al. denoted this approach as Classification of Item per Session Approach (CIPS). The strengths of this approach were that it requires only one classifier and that it takes advantage of the fact that the competition evaluation improves more if you correctly predict the items than if you correctly predict the sessions. FS and FI were used for model training with a balanced data set. They decided to use a random subspace implementation from WEKA and used REPTree for model training. REPTree is a fast decision tree learner that builds the tree using information gain and pruning it using reduced-error. Random subspace models consist of multiple trees which were constructed systematically by pseudo randomly selecting subsets of components of the feature vector. They ran the random subspace for 50 iterations, each with a subspace consisting of half of the 35 features.

In the second approach Cohen et al. divided the challenge's task into two levels like Romov and Sokolov [8]. In the first level they predicted buying sessions and predicted the items that are going to be bought in these sessions in the next level. They called this approach Two Level Classification Approach (TLC). They trained the first model with the FS on balanced data. The second classifier was trained on the FI, but considered only the sessions with purchases. They used the same algorithm and settings as in the CIPS approach described.

By analyzing the predictions which generated by their two approaches Cohen et al. noticed that the rate of the false positive classification was very high and decided to filter the predictions. The rules they used were to increase the threshold on sessions with one item to 0.6 and for sessions with more than four items to 0.6. With this solution the TLC produced 597,453 points on the competition score, while CIPS produced 649,946 points. The CIPS approach performed better than the TLC approach. The first classifier of TLC performed less well than the second classifier for item prediction. Cohen et al. believed that the session classifier could be improved. They compared the solution files of the two approaches and found that they differ in terms of sessions and items predicted. They wrote it implies that ensemble the two into one approach may improve the score.

3 Development Environment

I use the cross-industry standard process for data mining (CRISP-DM) from [3] for this project. The subsections are phases of this process. I did not run through the deployment phase, because it was not necessary for reaching the goals of this project.

3.1 Business Understanding

Objectives In this phase I only want to predict if a user is going to buy something or not. I do not want to predict which items they will buy or when they will buy it.

The task is to build a model which predicts purchases on the yoochoose data with the CRISP-DM steps. At the end a trained model with results and the pre-processing steps should be available. In addition, it is expected that this implementation can be adapted so that different model procedures and pre-processing procedures can be tested. The implementation of additional models is part of the following project. The following

project will use the developed environment of this project and is based on the chosen techniques of this project.

Situation I have some hardware requirements. The ML AG[1] has hardware resources at its disposal that are to be used in the environment (see Appendix A.1). The code is written and tested on computers with two graphics cards. I have two of this work computers. For longer running times, such as model training, the compute clusters can be used with 10 graphics cards. On the work computers, execution of code is done locally or in a docker container. On the compute clusters, I run the code in a docker container. Using Docker [5] allows me to execute the code machine-independent. The model techniques should support distributed computing, especially on graphics cards.

3.2 Data Understanding

Initial Data Collection The data I use for developing a buy prediction are the datasets of the RecSys Challenge 2015. For the issue, I need the training data files yoochoose-clicks.dat and yoochoose-buys.dat. In the readme file are some information about the columns of the data sets. The test file was only for the competition to have a collective test set where the label is unknown. I use jupyter notebook and pandas to analyze the data. Therefore, I write an own jupyter notebook for data collection. It downloads the data files which have the dat format, but they can be treated like csv format files. I load the click and buy data into pandas dataframes and set the data types like datetime64 for the column Timestamp. Then I save the pandas dataframes as parquet files on our system. Therefore, I use the pyarrow package. Apache Parquet is a columnar storage format which saves column information like the data types of pandas dataframe columns.

Data Description The yoochoose-click data has about 33 million rows and 4 columns. The pandas dataframe of the yoochoose-clicks needs 1.32 GB RAM. The yoochoose-buy data has about 1.2 million rows and 5 columns. The pandas dataframe of the yoochoose-buys needs 55.24 MB RAM. In Table 2 are the columns and their data types described. I decide to set Session ID, Item ID and Category to the data type object, so that there is no possibility to calculate with their values. It makes no sense to calculate mathematical

[1] ML AG is a working group on machine learning at the /* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */ (CSTI) laboratory at HAW Hamburg.

	Yoochoose-clicks	Yoochoose-buys
Rows	33,003,944	1,150,753
Columns	4	5
Memory usage	1.32 GB	55.24 MB

Table 1: Data shape

yoochoose-clicks	
Name	**Data type**
Session ID	Object
Timestamp	Datetime64
Item ID	Object
Category	Object
yoochoose-buys	
Name	**Data type**
Session ID	Object
Timestamp	Datetime64
Item ID	Object
Price	Float64
Quantity	Int64

Table 2: Data types

operations such as an average on these columns, because this calculation is not meaningful for a categorical value.

Data Exploration The yoochoose-clicks has about 9.2 million unique Session IDs, therefore each Session ID has 3.6 clicks on average. The max value of clicks is 200. This seems to be a hard limit, because 26 IDs reach this number of clicks. The yoochoose-clicks has data from 2014-04-01 to 2014-09-30. Whereby, the most clicks were in august. The users viewed 52,739 different items in this period of time. They looked at the item with the Item ID 643078800 147,419 times. It is the most considered item. There are 5,657 items which were only viewed once. The category "S" is the most common category. It occurs 10,769,610 times, which is 33%. The category "S" indicates a special offer. A number between 1 to 12 indicates a real category identifier. They are most common after the category "S" and they each account for up to 5%. Then the brands follow. The value of the category is the context of the click. If an item has been clicked in the context of a promotion or special offer, then the value will be "S". If the context was a brand, then the value will be an 8-10 digits number. If the item has been clicked under regular

category, i.e. sport, then the value will be a number between 1 to 12.

The yoochoose-buys has 509,696 unique Session IDs, therefore each buyer bought 2.3 distinct items on average. The session with most purchases has bought 144 distinct items. 20.1% of the buyers bought 1 item and 11.5% bought 2 distinct items in a session. The yoochoose-buys is also available for the same period of time as the yoochoose-clicks. The most distinct items where bought in august. The item with the ID 643078800 is bought the most in a session. It was bought in 15,203 sessions, which is 1.3%. The price describes the sales price of the item, when the item was bought. The min value is 0, but it is the representation of a missing value. The max value is 334,998, but there is no information about the currency. The average price is 1,423. The quantity describes how many items of one ID were bought in a session. The min value is 0, but it is the representation of a missing value. The max value is 30 and the average quantity is 0.6. The average is less than 1, because there are many missing values.

Data Quality Verification In the yoochoose-clicks and yoochoose-buys, the missing values were cleansed and replaced by 0. I did not find any missing values in Session ID, Item ID and timestamp. In category are 49.5% missing values and in price and quantity are 53.0% missing values. I did not detect any errors or data anomalies.

3.3 Data Preparation

Data Selection I decide to consider the entire data of yoochoose-clicks and yoochoose-buys, because the available hardware is suitable for the size and the quality of the data is sufficient.

Feature Engineering In this phase I use the features described in [4] instead of taking time to develop a good feature set by our own. I implement 13 features and the label to predict (see Table 3). The features are implemented in a python file with tests of the pytest framework. The first implementation of the feature extraction has a runtime of 2.5 hours on the entire data set. Every feature has its own python function for calculation. Average time clicks, maximal time clicks, and maximal clicks item are the features which need the most runtime. Therefore, I decide to improve the implementation of these three items to reduce the runtime. They have in common to compute an aggregation value like a maximum or an average with the pandas agg() function.

Name	Data type	Description
Session time	Float64	Session time in seconds
Average time clicks	Float64	Average time between two clicks in seconds
Maximal time clicks	Float64	Maximal time between two clicks in seconds
Day week	Int64	Day of the week
Month year	Int64	Month of the year
Time day	Int64	Time during the day – morning | afternoon | evening | night
Number clicks	Int64	Number of clicks
Maximal clicks item	Int64	Maximal number of clicks on one item
Percentage popular items	Float64	Percentage of items in the session that are popular – I compute an overall popularity score for each item as #buys/#clicks.
Distinct items	Int64	Number of distinct items
Average price	Float64	Average price of clicked items
Percentage S	Float64	Percentage of "S" category from all categories of the items
Percentage items with category	Float64	Percentage of items that have category
Label	Int64	0 if the user bought nothing and 1 if the user bought something

Table 3: Feature set

	Percentage factor	Change 1	Change 2	Change 3
Average time clicks	To first version	0.56	0.72	0.15
	To predecessor version	0.56	1.25	0.21
Maximal time clicks	To first version	0.55	0.72	0.15
	To predecessor version	0.55	1.30	0.21
Maximal number clicks	To first version	0.99	1.14	0.24
	To predecessor version	0.99	1.16	0.21
All features	To first version	0.72	0.87	0.21
	To predecessor version	0.72	0.80	0.24

Table 4: Runtime change of feature extraction

The main objective is to parallelize the calculation of the three features on the processor. The individual improvements build on each other. Each step uses the previously made changes and not the first version of the implementation. The first change is to remove the sort_values() function of pandas in average time clicks and maximal time clicks, and only execute it once before the functions will be executed. This reduces the runtime by almost 30%. The second change is to perform the aggregation by the apply function of pandas on the dataframe. This does not improve the runtime of the three features and the runtime extends again. However, an apply() function can be executed distributed, but an agg() function cannot. The third change implements parallel execution with the multiprocessing package of python. It uses data chunks to distribute the data on threads and then applies the aggregation on this data chunks parallel. I implement it with the three features. The Appendix A.2.1 shows the first implementation of average time clicks and the Appendix A.2.2 shows the parallel version of average time clicks with all three changes. The runtime has improved by 79 percent after the three steps.

The Table 4 shows the changes in the duration of the three features and the total duration of all features. For this purpose, the changes to the first implementation and to the previous step are considered. The changes are displayed with the percentage factor. I use hardware configuration 1 (see Appendix A.1) for the runtime measurement. After change 1, the runtime of average time clicks is 0.56 of the runtime of the first implementation. After change 2, 0.72 of the runtime of the first implementation and after change 3, 0.15. The next line shows the changes to the predecessor. From the first implementation to change 1, the percentage factor is 0.56. From change 1 to change 2, the percentage factor is 1.25, i.e. the runtime of average time clicks is 1.25 times longer with change 2 than with change 1.

I also tried to improve the calculation of features by choosing a suited data type for

each column in the click and buy dataset. Therefore, I used the categorical data type from pandas library for the Session ID, Item ID and so on. However, the runtime of the three features raised again, although this was done on the version with parallelization. The runtime of average time clicks was 3 times higher than the version with the three previously described changes. That is why I stayed with the previous data types (see Table 2).

3.4 Data Understanding of Features

Data Description The feature data has about 9.2 million rows and 14 columns. The pandas dataframe of the features needs 1.11 GB RAM. In Table 3 are the columns and their data types described. All features are numbers.

Data Exploration The buyer class has 509,696 samples. The non-buyer class has 8,740,033 samples. Only 5.5% of the samples are buyers, this could be a problem for the model training. Some modeling methods cannot learn to generalize from this data and always predict the majority class. This model has an accuracy of 94.5%, but labels every sample as a non-buyer.

The features are the knowledge for the model to learn the differences between the classes. The correlation expresses the cohesion between the features. The correlation of the features to the label is shown in table Table 5. The highest value is 0.19. The individual correlations are low to the label.

Data Quality Verification The features have no missing values. I calculated for all Session IDs the features, because the number of rows is the number of unique Session IDs.

3.5 Modeling

Modeling Technique Methods like collaborative filtering and content-based filtering cannot be used. The data does not allow to collect user information beyond sessions or use connections between items. I only have the Item ID and category, but cannot say if two items are similar with this information. I need to predict a buy with the click stream data. The click stream is represented with the calculated features as a state for every

Feature	Correlation
Session time	0.15
Average time clicks	0.05
Maximal time clicks	0.11
Day week	0.03
Month year	-0.01
Time day	-0.02
Number clicks	0.19
Maximal clicks item	0.18
Percentage popular items	0.06
Distinct items	0.13
Average price	-0.03
Percentage S	-0.01
Percentage items with category	-0.01
Label	1.00

Table 5: Correlation to the Label

Session ID. I use multilayer perceptron as model algorithm and implement models with Keras with TensorFlow backend.

Test Design I split the data into train and test data sets. I use 70% for model training and 30% for testing. The data sets are divided equally for all models. They learn and test on the same data. For evaluation, I use the accuracy and the loss. The expectation is that the loss will decrease during training. But the accuracy should increase during the training. Furthermore, I use a confusion matrix to visualize precision and recall.

Model Description I use multilayer perceptron for our models. All models have an input layer with 13 dimensions for all developed features. I use 2 dense layers with 64 neurons and relu as activation function. Between these layers is a dropout. Then our models have an output layer with one dimension for the predicted label. The activation function of the output layer is a sigmoid function. I use 20 epochs and a batch size of 128. The loss function is binary cross entropy and the optimizer is rmsprop.

The first model is trained on 70% of input data. These are 6,474,810 samples with 362,650 buyers and 6,112,160 non-buyers. The second model is trained on 725,300 samples with 362,650 buyers and non-buyers each. I sample the data with RandomUnderSampler from sklearn. It is an undersampling method for data balancing. The third model is trained on

			Actual class			
			Buyer		Non-buyer	
			Absolute	Relative [%]	Absolute	Relative [%]
Predicted class	Model I	Buyer	0	0.00	0	0.00
		Non-buyer	147,046	5.30	2,627,873	94.70
	Model II	Buyer	96,512	3.48	608,733	21.94
		Non-buyer	50,534	1.82	2,019,140	72.76
	Model III	Buyer	91,231	3.29	530,552	19.12
		Non-buyer	55,815	2.01	2,097,321	75.58

Table 6: Confusion matrix

12,224,320 samples with 6,112,160 buyers and non-buyers each. I generate new data for buyers with SMOTE from sklearn. It is an oversampling method for data balancing.

Model Assessment The first model does not generalize well. It only predicts non-buyers, see Table 6. It has not learn the different classes from the data. It is not useful to look only at the accuracy which is 94.7%. The data set is unbalanced, and the mayor class is 94.5% of all data. Therefore, I consider the measures like recall and precision, see Table 7. The recall is 0, because the buyer class is never predicted.

Model II and Model III have less accuracy than Model I, but they predict the buyer class. They learn something from the data. Recall or sensitivity is the proportion of real positive cases that are correctly predicted positive. In our case it is how often the buyer class is correctly predicted from all buyer cases. A value near 1 means that I recognize the buyers well.

Precision or confidence denotes the proportion of predicted positive cases that are correctly real positives. In our case it is how often the buyer class is correctly predicted from all predicted buyer cases. A value near 1 means that I can distinguish the buyers well from the non-buyers.

Both models recognize the buyers well, but they do not distinguish well between the two classes. One possibility is that the features are not good enough to distinguish the two classes well. In addition, the model may have inappropriate parameters, e.g. too few layers or neurons. Both data balancing methods perform equally well with the data and model procedure.

	Accuracy	Precision	Recall	F1 score	Jaccard index
Model I	0.95	-	0	-	0
Model II	0.76	0.14	0.66	0.23	0.13
Model III	0.79	0.15	0.62	0.24	0.13

Table 7: Metrics of models

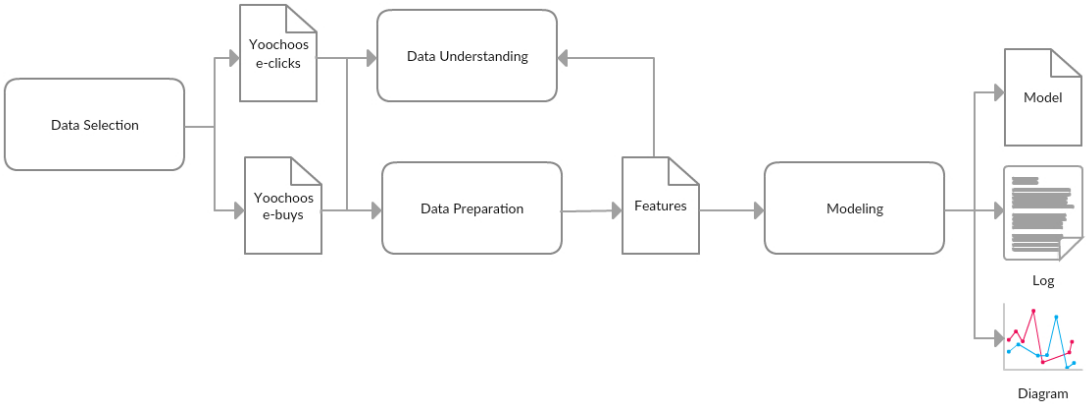

Figure 1: Development environment for buy prediction models

3.6 Evaluation

The objectives were achieved. An environment with the desired requirements was implemented. In the following, the implemented environment is summarized, and which technologies were used is shown. In the Data Selection module, a jupyter notebook was implemented to load data to pandas and saved it to parquet files. In the Data Understanding phase, a jupyter notebook for the clicks and purchases was implemented, and one notebook for understanding the features with pandas and matplotlib. In the Data Preparation, I used python and pytest to calculate features and the label of the clicks and purchases. In the Modeling phase I implemented three models with Keras. I used matplotlib and scikit-learn to generate evaluation measures. I produced log data of model training and saved the weights of the best model of each training phase. The individual modules can be exchanged or adapted.

I did without an automatic processing line, because the data basis does not change and the implementation of the features and/or the models changes depending on the research question. However, it is not necessary to recalculate the features with each

model training. The modules can be executed manually on the PC or via a docker container on hardware configuration 1 (Appendix A.1). It is possible to run the modules on hardware configuration 2 (Appendix A.1) with a docker container or on any other PC.

Thus, there is an environment for further experiments that meets the requirements. It runs on the desired systems and uses their available resources. I have gone through the CRISP-DM and created first buy prediction models.

4 Conclusion

Creating an environment for experiments for machine learning projects is something very practical and not scientific. I did not find any suitable literature and therefore oriented myself on CRISP-DM and implemented it practically for my application. In this project, I used the work of Cohen et al. [4] as orientation, and the findings were used to test the environment. The goal of the project was to build an environment for further experiments and not to create the best model for buy prediction. These experiments will follow in the next project and different methods to create features will be implemented in the resulting environment and compared with each other.

Acknowledgements

The work was developed within the context of the machine learning working group ML AG at HAW Hamburg. I would like to thank Prof. Dr. Kai von Luck and M.Sc. Tobias Eichler, who actively support the working group, encourage the exchange between the students and supervise our projects. I would especially like to thank Matthias Nitsche, Stephan Halbritter and Timo Lange for the professional exchange in this group and for the work on the systems to provide a platform for machine learning projects. I would also like to thank Henrik Wortmann from the CSTI, who are driving the development of an infrastructure for machine learning projects at the HAW Hamburg with Tobias Eichler. I would also like to thank all members of the ML AG who are not named by name for the exchange and participation which contributed to this work.

References

[1] Bhagat, Rahul ; Muralidharan, Srevatsan ; Lobzhanidze, Alex ; Vishwanath, Shankar: Buy It Again. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '18*. New York, New York, USA : ACM Press, 2018, S. 62–70. – URL http://dl.acm.org/citation.cfm?doid=3219819.3219891. – ISBN 9781450355520

[2] Bhattacharyya, Siddhartha ; Jha, Sanjeev ; Tharakunnel, Kurian ; Westland, J. C.: Data mining for credit card fraud: A comparative study. In: *Decision Support Systems* (2011). – ISBN 0167-9236

[3] Chapman, Pete ; Clinton, Julian ; Kerber, Randy ; Khabaza, Thomas ; Reinartz, Thomas ; Shearer, Colin ; Wirth, Rudiger: Crisp-Dm 1.0. In: *CRISP-DM Consortium* (2000), S. 76. – ISBN 9780769532677

[4] Cohen, Nadav ; Gerzi, Adi ; Ben-Shimon, David ; Shapira, Bracha ; Rokach, Lior ; Friedmann, Michael: In-House Solution for the RecSys Challenge 2015. In: *Proceedings of the 2015 International ACM Recommender Systems Challenge on - RecSys '15 Challenge* (2015), S. 1–4. – URL http://dl.acm.org/citation.cfm?doid=2813448.2813519. ISBN 9781450336659

[5] Inc., Docker: *Docker*. 2018. – URL https://www.docker.com/. – Zugriffsdatum: 2018-11-16

[6] Mobley, R K.: Predictive Maintenance. In: *Plant Engineer's Handbook*. 2001. – ISBN 978-0-8493-3598-3; 978-1-4200-5618-1

[7] Nyce, Charles: Predictive Analytics White Paper. In: *American Institute for Chartered Property Casuality Underwriters* (2007), S. 16. – URL http://www.theinstitutes.org/doc/predictivemodelingwhitepaper.pdf{%}5Cnhttp://ieg-sites.s3.amazonaws.com/sites/4e70a00a3723a839c1000042/contents/content{_}instance/4ec268ce3723a856ba00015c/files/PredictiveModelingWhitepaper.pdf

[8] Romov, Peter ; Sokolov, Evgeny: RecSys Challenge 2015. In: *Proceedings of the 2015 International ACM Recommender Systems Challenge on - RecSys '15 Challenge*. New York, New York, USA : ACM Press, 2015, S. 1–4. – URL http://dl.acm.org/citation.cfm?doid=2813448.2813510. – ISBN 9781450336659

[9] Xia, Qiaolin ; Jiang, Peng ; Sun, Fei ; Zhang, Yi ; Wang, Xiaobo ; Sui, Zhifang: Modeling Consumer Buying Decision for Recommendation Based on Multi-Task Deep Learning. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management - CIKM '18* (2018), S. 1703–1706. – URL http://dl.acm.org/citation.cfm?doid=3269206.3269285https://doi.org/10.1145/3269206.3269285. ISBN 9781450360142

A Appendices

A.1 Hardware Appendix

Those are the hardware resources of the ML AG at HAW Hamburg I used in this project. There are two computers with the following equipment.
Configuration 1:

- RAM: 64 GB
- CPU: Intel Core i7 CPU @ 4.2 GHz with 8 threads
- GPU: 2 x GeForce GTX 1080ti

Furthermore, there are two compute clusters with following equipment.
Configuration 2:

- RAM: 396 GB
- CPU: 2 x Intel Xeon E5-2697V4 @ 2.3 GHz with 36 threads
- GPU: 10 x Nvidia Quadro P6000

A.2 Feature Engineering: Parallelized Execution

A.2.1 Feature Calculation

This is my first implementation of the feature average time clicks with Python. For each session an average of the difference between two clicks is calculated in seconds.

```
def average_time_between_two_clicks(df_clicks):
series = df_clicks.groupby(['Session ID']). \
          agg({'Timestamp': lambda group: group. \
              sort_values().diff().mean()})
series['Timestamp'] = round(series['Timestamp']. \
                      dt.total_seconds(), 2)
series.fillna(value=0, inplace=True)
return series['Timestamp']
```

A.2.2 Feature Calculation with Parallelization

This is the parallel version of the feature average time clicks with three major changes to the first implementation. The sort_values() function is no longer called in the method, but a sorted dataframe of the clicks is passed. So that this only happens at one point in the source code and not again and again in the methods. An apply() is introduced, because apply() can be parallelized. Here it can be found in the method func_group_apply_avg_time(). The apply() calls the method with the agg() to calculate the average of the differences. Here it is the method group_function_avg_time(). The data chunk column divides the data set into different parts. The method apply_parallel() is used to parallelize the apply() per data chunk and execute it on the available threads in parallel.

```
def average_time_between_two_clicks(df_clicks):
    df_clicks["data_chunk"] = df_clicks["Session ID"]. \
                              astype('int64'). \
                              mod(cpu_count() * 1000)
    series = apply_parallel(df_clicks.groupby("data_chunk"), \
                            func_group_apply_avg_time)
    series.fillna(value=pd.Timedelta('0 days'), inplace=True)
    return round(series.dt.total_seconds(), 2)


def group_function_avg_time(group):
    return group.diff().agg('mean')
```

```
def func_group_apply_avg_time(df):
    return df.groupby(['Session_ID'])['Timestamp']. \
            apply(group_function_avg_time)


def apply_parallel(df_grouped, func):
    with Pool(cpu_count()) as p:
        ret_list = p.map(func, \
                    [group for name, group in df_grouped])
        ret_list = pd.concat(ret_list)
        return pd.Series(ret_list)
```