DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Project Report

Prudsys Data Mining Cup 2020

Kyrillus Aiad
Nicolas Bauer
Markus Böbel
Karin Frlic
Michael Hauer
Min Wu
Tobias Zeulner

Management Summary

Representing the Technical University of Munich, a group of seven Master's students tackled the 21st Prudsys Data Mining Cup. The Prudsys Data Mining Cup is an international student competition in which, in the year of 2020, 162 teams from 126 universities based in 35 countries took part [1].

For this year's competition, a prediction model for a retailer had to be developed. Considering possible product promotions, limited by six months of training data and anonymized training features, the goal of the task was to predict the number of sales for every item for a two-week prediction period [1].

Exploring different methods in econometrics, machine learning as well as deep learning, we ended up using two complementary *XGBoost* models referred to as the "split approach". One of those predicted sales on promotion days and the other on the remaining non-promotion days. Due to the large difference in values between sales on promotion and non-promotion days, this combined approach outperformed the previous models with the highest monetary value based on the competition's predefined evaluation metric.

Combined with our efforts to prepare the data, research about possible data sources and build useful features, we ended up reaching a monetary value of €3,507,974.84 and ranking 13th in the overall competition, not far from the first place with a monetary value of €4,305,017.89.

The achievement is attributed to efficient teamwork, additional motivation due to group internal competitions, and guidance by our tutors and professor.

We therefore want to thank Prof. Dr. Martin Bichler, Paul Sutterer, Stefan Heidekrüger and our colleagues from the Data Mining Seminar 2020 for their guidance and feedback during our journey.

Contents

1 Introduction

1.1 Task Description

This year's Prudsys Data Mining Cup was all about estimating customer demand. Therefore, a dataset describing past sales of an unknown retailer was given. The dataset consisted of information about 10,463 items, including their prices and promotions as well as the number of sales in the period between January 1st and June 29th, 2018. The goal of the competition was to predict the demand of each of the given items for a two-weeks period between June 30th and July 13th, 2018. The final submission was eventually evaluated, based on the monetary value M that the retailer would gain, when purchasing the predicted demands. Thus, in case the real demand d would be higher or equal to the prediction $\hat{d}$, the monetary value of an item with price p would equal to the realized revenue, calculated by the following formula:

$$M \underset{\hat{d} \leq d}{=} \hat{d} \times p \tag{1.1}$$

In case the actual demand d would be lower than the prediction $\hat{d}$, an overstocking fee would be applied and the monetary value M would be calculated as follow:

$$M \underset{\hat{d} > d}{=} d \times p - 0.6 \times p(\hat{d} - d) \tag{1.2}$$

The winner of the final competition was chosen from 162 participating teams according to the monetary value calculated by the aforementioned formula.

1.2 Structure of the Report

Firstly, we present the project plan we were using to tackle the data mining challenge. In Chapter 2, we then give an overview of the dataset. This includes a detailed descriptive data analysis and a summary of the created features.

In Chapter 3, we discuss our choice of models for the competition. Therefore, we first present our time series research results. Afterwards, we provide a brief introduction to sequence-to-sequence models. At last, we give an overview of the different ensemble methods as well as the implementation of the evaluation function.

In Chapter 4, we present the results of the competition. This includes a comparison of the applied models and a closer analysis of the finally selected model.

In the concluding discussion, we review the competition and summarize faced problems, challenges and the solutions we applied.

1.3 Project Plan

When tackling a challenge like the Prudsys Data Mining Cup with a team, the organization of the project is one of the keys to successful results. From a project management perspective, the main drive thereby was to develop models as fast as possible by splitting up tasks wisely and exchanging knowledge on a regular basis. Hence, we decided to follow a project plan based on eight sprints.

The first sprint was dedicated to data analysis and model research. Therefore, we split up the team into two subgroups. One group consisting of five group members *(Karin Frlic, Kyrillus Aiad, Markus Böbel, Min Wu, Nicolas Bauer)* were focusing on the structure of the data. The results of the data analysis are described in Chapter 2. The other sub-team did research on promising models *(Michael Hauer, Tobias Zeulner)*. The resulting models were eventually clustered into three groups: econometric models, machine learning models as well as deep learning based approaches.

Within the second sprint, we divided the team into three subgroups according to the resulting model clusters of sprint one. The objective was to gain deeper insights on the strengths and weaknesses of the respective model groups and their approaches as well as the development effort of the models. Thus, two members each were looking into econometric models *(Kyrillus Aiad, Nicolas Bauer)* and machine learning models (*Michael Hauer, Min Wu*). The remaining three team members worked on deep learning models *(Karin Frlic, Markus Böbel, Tobias Zeulner)*.

The subject of the third sprint was to create first working models. To gain independent approaches, the project group was divided into two teams, which competed against each other within the sprint. Hereby, the teams from the second sprint were reorganized in a way that each of the competing teams consisted of at least one specialist of each of the model groups. The first team *(Karin Frlic, Kyrillus Aiad, Markus Böbel, Min Wu)* worked on a solution based on the machine learning algorithm XGBoost (see Chapter 3) and a variety of new implemented features. The second team *(Nicolas Bauer, Michael Hauer, Tobias Zeulner)* focused additionally on the sequence-to-sequence (*Seq2Seq*) model based on long short-term memory (LSTM) layers.

Based on the insights gained in the internal competition, we combined the approaches. To achieve this, we split up the whole team again into three subgroups with dedicated tasks. Team one *(Markus Böbel)* prepared a pipeline, which enabled the team to create and test new models fast as well as to generate internal and final submissions. The second team *(Kyrillus Aiad, Nicolas Bauer, Michael Hauer, Min Wu)* worked on the creation of new features and the development of a suitable evaluation metric. The third team *(Karin Frlic, Tobias Zeulner)* developed the sequence-to-sequence model further including a fitting evaluation metric.

From the fifth sprint onward we focused on refining our best-working models. This included the application of ensemble methods like stacking of different models as well as the chosen "split approach" to train models based on the occurrence of promotions. Here the team was working as a whole, and tasks were split during the organizational calls which took place twice per week.

2 Data

In order to build good performing models, we first needed to have a clear picture about the given datasets. In this chapter, we first cover the descriptive data analysis performed on the original datasets, and then an overview of the completed feature engineering methods is described.

2.1 Descriptive Data Analysis

Three individual `.csv` files are provided by the competition organizers for this demand forecasting task. `item.csv` contains meta information about each item, `orders.csv` covers all transaction data within the 6-month training period, and finally `infos.csv` gives sales price and promotion information for all items during the simulation period. To better understand the structure and analysis conducted for each file, we introduce the analysis done for each dataset separately, and in the end, some identified problems are reported.

2.1.1 Items

As mentioned earlier, the items table contains meta information about each item. In the table below (Table 2.1), each column in the original items table is introduced, along with its basic analysis.

Features	Basic Analysis
itemID	10,463 items
brand	275 brands
manufacturer	253 manufacturers
customerRating	scale: 1-5; avg 4.4 (rated)
category1	8 categories
category2	52 categories
category3	8 categories
recommendedRetailPrice	avg €118

Table 2.1: Features in the items table and their descriptive analysis.

We noticed that out of 10,463 items, only 3,346 items have indicators in the "customerRating" column. Additionally, we also found out that the "category1" and "category2" features are hierarchically connected. In other words, the items are first categorized by "category1", then

each category within "category1" is further split to "category2". "category3", on the other hand, has no clear mapping between "category1" nor "category2".

2.1.2 Info

`infos.csv` indicates the price and promotion information for each item during the simulation period. The feature names and a basic analysis are also shown in the following table (Table 2.2):

Features	Basic Analysis
itemID	10,463 items
simulationPrice	avg €122
promotion	1,843 items promoted

Table 2.2: Features in the info table and their descriptive analysis.

Notice that an item can be promoted multiple times within the prediction period. After analyzing those promoted items in the simulation period and their prices, we found out cheaper items are promoted more frequently.

2.1.3 Orders

`orders.csv` includes all transaction data within the training period as shown below (Table 2.3):

Features	Basic Analysis
time	01-01-2018 until 06-29-2018 (180 days)
transactID	2,076,066 (unique); 2,181,955 (total rows)
itemID	9,840 unique items
order	avg 1.24 units sold per transaction
salesPrice	avg €36.7

Table 2.3: Features in the orders table and their descriptive analysis.

There exists an overall upward trend in the number of orders on the platform as time progresses (shown in Figure 2.1 and Figure 2.2). Notice the orders data is not aggregated on any level. It was suggested by the competition that we should aggregate the data on a desired level (e.g., daily, weekly, bi-weekly, etc) by ourselves in order to train and predict later.

We also realized the orders data for each item individually seems to show a "peaky" pattern consisting of many spikes. Furthermore, according to the transaction history, there are 6,126 items with fluctuations in "salesPrice" throughout the 6-month training period. An example of the fluctuations in "salesPrice" and number of sold orders can be seen in Figure 2.3 and Figure 2.4 for item 7851. The two figures also suggest that we could combine information from price and orders together to derive promotion dates for training, since the promotion dates are not

Figure 2.1: Trend showing the number of orders aggregated by week.

Figure 2.2: Trend showing the number of orders aggregated by month.

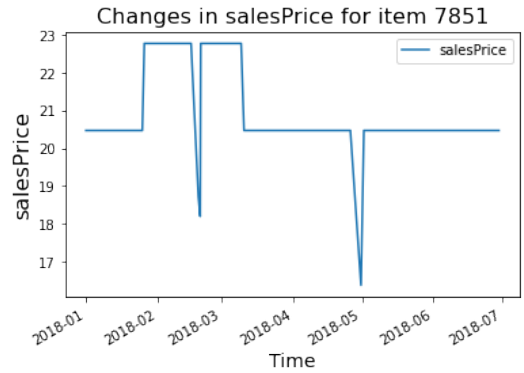

Figure 2.3: Pattern for changes in salesPrice during training period for item 7851.

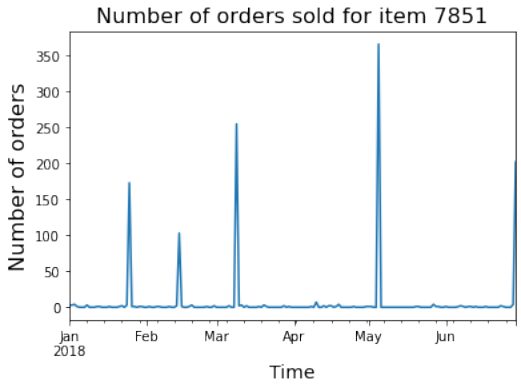

Figure 2.4: Pattern for changes in number of orders sold during training period for item 7851.

explicitly given in the training period and could potentially be the most important feature for our models.

2.1.4 Identified Problems

After the initial descriptive data analysis, we identified some major problems with the data, namely:

- Out of 10,463 items, only 9,840 have transaction histories in the orders table.
- We only have training data for 6 months, and therefore we lack knowledge in the seasonal patterns or influence for the entire year, making predictions in unseen months more challenging.
- Promotion dates are not given for the training period.

- The overall trend is recognizable for the total amount of sales. However, each item shows the aforementioned peaky and unstable sales pattern, which makes it hard to predict.

Next, we tried to find solutions to address the above-mentioned problems in the feature engineering step.

2.2 Feature Engineering

A pipeline was set up for the whole project and the first two steps in the pipeline were for feature engineering. In the initial basic feature creation step the three input files: `items.csv`, `infos.csv` and `orders.csv` were processed to create a static feature dataframe and a dynamic feature dataframe. These features are separated into the static and dynamic dataframes because the input requirements differ between the models. In the final step, these two dataframes are used for the meta feature creation, such as one-hot encoding or lag features.

2.2.1 Dynamic Features

We aggregated the orders on a daily basis and arranged them per item. These orders need to be predicted at the end and therefore the "daily_sold" column serves as our label. The Table 2.4 shows that each row represents a unique item and date combination with the corresponding time and item dependent features. The training period as well as the prediction period were included in this dynamic features table (January 1st until July 13th, 2018).

itemID	date	daily_sold	feature 1	feature n
1	2018-01-01	0	5	...
...	...	...	...	...
10463	2018-07-13	23	10	...

Table 2.4: Structure of the dynamic features table.

Price Features

The sales price of an item is given by the recommended retail price in the training period and by the simulation price in the prediction period. The actual sales price changes throughout the training period and is taken into account by calculating average prices for a given day for a given item. This average sales price of the day was then used to calculate the relationship with the recommended retail price to get some information how relatively cheap or expensive an item is on this day.

Promotion Features

No promotion information is given in the training period, but we assumed that the peaks in the item sales patterns indicated promotions. The differences in the sales whether or not an

item is promoted are very significant. Therefore, identifying the correct promotion dates is a key part of the challenge. We determined the promotions for each day and item with the following formula:

$$log_2(\frac{sold\_on\_current\_day + lambda}{sold\_on\_previous\_day + lambda}) > threshold \tag{2.1}$$

Researching Prudsys clients, we discovered that 1-2-3.tv with their online-auction model fits a number of characteristics in the data perfectly, for example, the peaky sales pattern and the sales on each day of the week. Consequently, we created more promotion features that took into account 1-2-3.tv's online auction business model. A Boolean feature was created for indicating the two days after each promotion, because 1-2-3.tv still sells their products for 2 days after an auction with a discount. The total promotions per day were included as another feature, as well as the number of days since the last promotion for each item.

Sales Features

Furthermore, the sales of the last promotion day for each item were calculated. The overall sales per day were added as well. The information for this feature can be calculated in the training period but needs to be forecasted for the prediction period. The forecasting was implemented with Facebook's *Prophet* model.

Time Features

There were a number of time features regarding the date that were generated. This included the day of the year, week number, and month, as well as the day of the week. Additionally, a Boolean variable reflecting German holidays was added.

2.2.2 Static Features

The static features include all features concerning individual items that are constant over time. As can be seen in Table 2.5, each row in this table represents the features for one specific item.

item ID	feature 1	feature n
1	12	...
...	...	...
10463	7	...

Table 2.5: Structure of the static features table.

Price Features

Besides the given recommended retail price, a static price feature with the average sales price of an item over the whole training period was created.

Promotion Features

Due to the importance of the promotions, some static promotion features were also generated. There is a feature for each item with the average sales on promotion days as well on non-promotion days. In addition, the number of promotions for an item and whether it was promoted at all in the data was added.

Cluster Features

In the file `items.csv`, a lot of information was given for each item, such as customer rating, brand, manufacturer and the categories as mentioned previously. The category information was only provided numerically and it was difficult to make sense of it. Consequently, an approach to gain more item information was to categorize the items according to other features. One of those features included items that were often bought together. Moreover, using `pandas.qcut`, items were categorized corresponding to their prices, orders and the combinations of them. The combination yielded item clusters of high price with high orders or low price with low orders and so forth.

2.2.3 Meta Features

In this step, the already existing features were mutated to provide more value to the models. First, the variable types needed to be formatted accordingly to the model requirements. Categorical features were one-hot encoded and all Boolean features were converted to integers in order to fit the numerical input requirements of the boosting methods like *XGBoost*. Additionally, lag features are added to shift features by a certain number of days. These features have a suffix indicating how long the lag is, so e.g. t-1 is for the first lag. This lag includes time series information even for models that do not have them by design because the data of previous days would not be taken into account without the lags. In order to use those lags for the whole prediction period we predict on a daily basis and use the results for the lag features to predict the following days. The lags were done for the orders and the promotion features.

Lastly, with the help of *featuretools*, a Python framework which can perform automated feature engineering, we created more automated features based on the generated static feature table and its relationships to the original orders table. With defined relationships between various tables, *featuretools* can apply "deep feature synthesis", or in other words, it can stack basic primitive features together to form new ones. Here, quite a few statistical features were generated, for example, the standard deviation of an item's sales price or the skewness of an item's orders during the training period, and so on.

In the end, we generated more than 600 features in total in the static and dynamic feature dataframes.

3 Models

In the following, we go into more detail about the models used in our project. Prior to that, we explain the general topics in the area of time series forecasting.

3.1 Research

We started our project by conducting some research into time series in general and common models used to forecast such a time series. These two parts will be discussed separately.

3.1.1 Time Series

A time series is a sequence of observations taken sequentially in time [2]. Such a series is either univariate or multivariate [3]. The former one is based on a single time-dependent variable. The latter one, on the other hand, is based on at least two variables that vary over time.

Furthermore, such a series can be decomposed into three parts (see Figure 3.1) [4]: the trend, seasonal, and residual data. The trend is the increasing or decreasing value, seasonality is the repeating short-term cycle, and the residuals are the random variations in the series. Adding all three parts together retrieves the originally observed data.

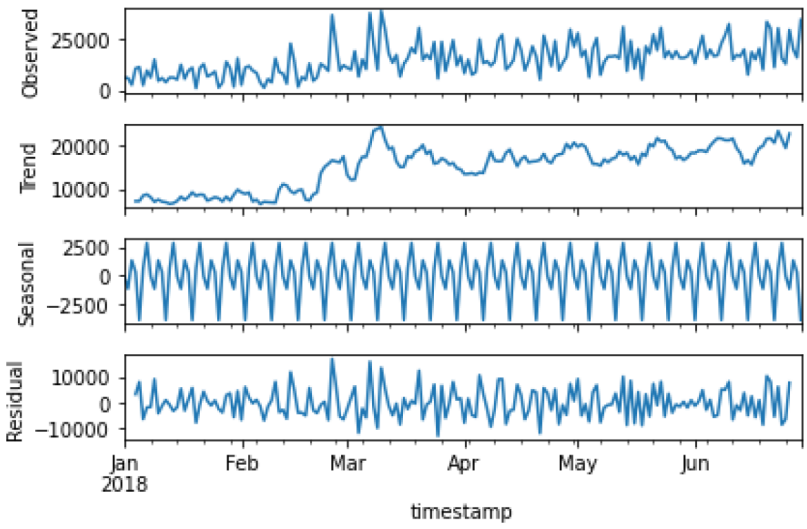

Figure 3.1: Decomposition of our time series (overall amount of orders of all items) in three components.

Finally, one has to differentiate between a stationary and non-stationary time series (see Figure 3.2) [5]. If the mean (case b), the variance (case c), or the covariance of the i-th and the $(i+m)$-th term (case d) is a function of time, then the time series is non-stationary.

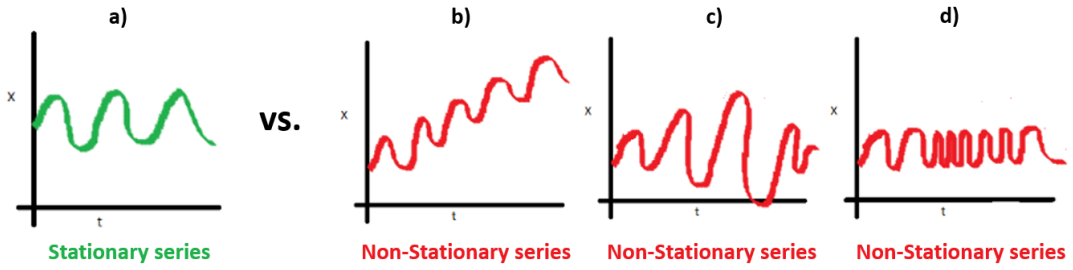

Figure 3.2: Visual difference between a stationary and a non stationary time series [6].

This is important as some forecasting models require the data to be stationary, without a trend or seasonality, or can only handle a univariate series. Hence, depending on the model, the data has to be preprocessed.

3.1.2 Models

There exist several commonly used methods for how to do time series forecasting. These methods can be grouped into three main types of models (see Figure 3.3): econometric models, machine learning models, and neural networks, which will be explained in the following.

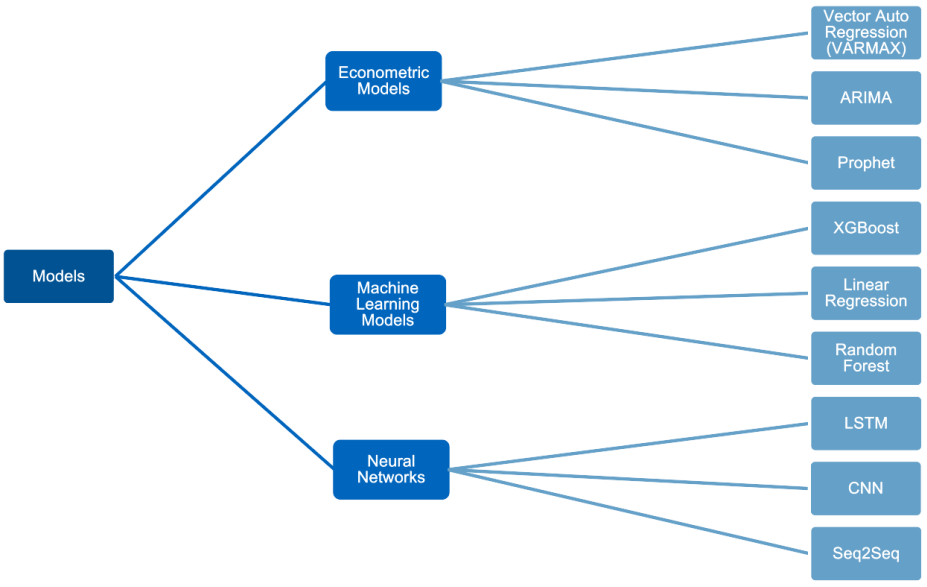

Figure 3.3: Overview of the three main types how to do time series forecasting with some common examples.

Statistical models in econometric analysis have strong theoretical foundations and often strict assumptions about their dependent and independent variables [7]. This enables inferencing and interpreting the results as well as the coefficients of the model. As the time series for a single item has no seasonality or trend, just peaks in changing intervals, these models are not recommended for our task. However, it can be used to predict the overall amount of orders of all the items, because this series includes seasonality and trend (see Figure 3.1) and use it as an additional feature. For this task we used *Prophet*.

Common examples of forecasting a time series with machine learning methods are standard models like linear regression or ensemble learning methods like random forest or *XGBoost*. Latter techniques have proven to be particularly effective for time series analysis in the last years [8], therefore we decided to focus on them. *XGBoost* and *CatBoost* were able to achieve the best performance out of all of the trained and tested algorithms.

The third approach to do forecasting is using neural networks. There are simple models like an LSTM or CNN, but also more complex ones like a *Seq2Seq* model. The winner of the Kaggle challenge "Web Traffic Time Series Forecasting" in 2017 used such a *Seq2Seq* model [9], which is why we decided to use it in our project and see if we can achieve similar performance with our dataset.

3.2 Seq2Seq

Sequence to sequence learning (*Seq2Seq*) was first described in 2014 in [10] as a model for machine translation. It uses an encoder-decoder framework, mapping an arbitrary long input sequence to an arbitrary long output sequence with an intermediate encoded state. The encoded state represents the entire history of the sequence providing the context the decoder needs to generate an output sequence. This understanding is produced by iteratively reading each of the input values using an LSTM architecture, which produces the final state vector. The decoder LSTM then takes this vector for its initial state, iteratively writing each output and updating its internal state. Although the original application was machine translation, recent work showed that it can easily generalize to sequences from many domains, including time series forecasting (see Figure 3.4) [11].

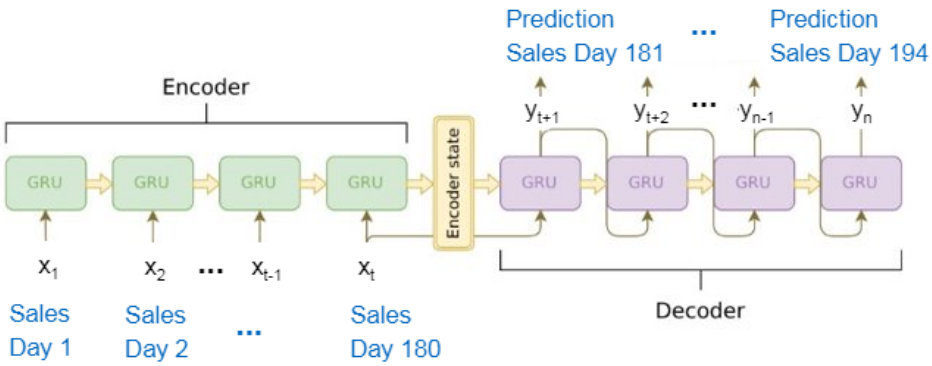

Figure 3.4: Visualization of a *Seq2Seq* model for a time series prediction [12].

We implemented our own *Seq2Seq* model with the functional API from Keras (see Figure 3.5). The encoder encodes the multivariate time series in a hidden state using the dynamic features. The hidden state is then passed to the decoder, where it is used together with the outputs of the decoder to implement the attention mechanism. This helps the network to cope with long input sequences. The output tensor of that mechanism is concatenated with the static features, which are passed through a previous dense layer. To calculate the final output (amount of orders for one item per day), it is fed again through a dense layer.

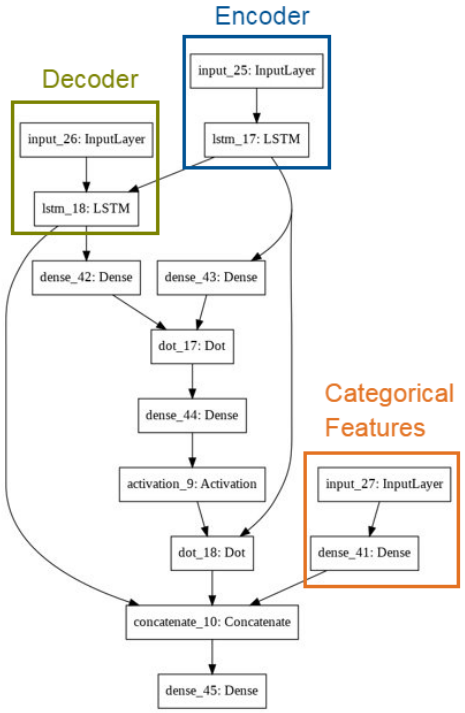

Figure 3.5: Our own *Seq2Seq* implementation with the functional API from Keras.

Even with different scaling variants for the input features, dimensionality reduction on categorical features, and hyperparameter tuning, the model was still unable to produce reliable results. Especially for the flat regions in the time series, the prediction error was high and fluctuated a lot. The most likely explanation for this is the size of the training set. Training such a model requires more than merely six-month time series data.

3.3 Boosting and Ensemble Methods

3.3.1 Evaluation Metric

Our final submission was evaluated based on the monetary value[1] that we were able to generate. To integrate this requirement into the boosting models, we derived and implemented

[1] Defined in Section 1.1.

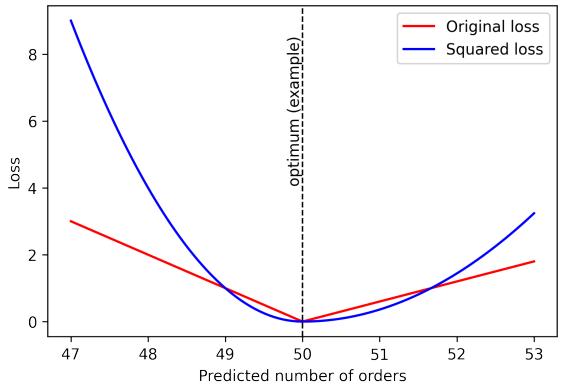

Figure 3.6: The loss based on how much more revenue we could generate (red) was changed to its squared version (blue) to ensure more stable training.

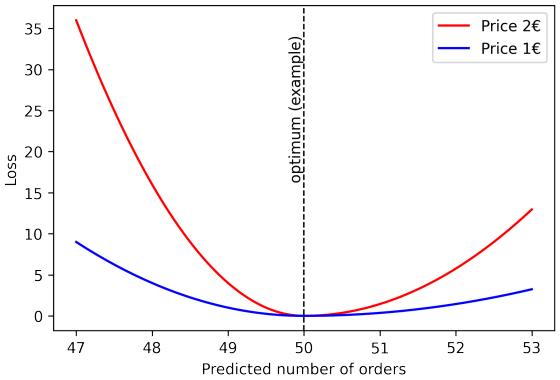

Figure 3.7: The defined loss reflects that the correct prediction of expensive items is more important than the prediction of cheaper items.

the custom evaluation and objective functions.

The *evaluation function* is used to evaluate the model, in particular to select the best model among the models trained with different hyperparameters. Our implementation directly corresponds to the monetary value, i.e., the metric provided by the organizers of the competition.

The *objective function* provides the model with the information that guides the learning: the first and the second derivative of the loss function. We based this function on a self-derived loss, namely how much more revenue the model would generate, if it predicted the demand correctly. This loss is asymmetric, as predicting one item too few results in a missed opportunity to sell an item, so the loss equals the item price, while predicting one item too many causes the overstocking fee of only 60% of the item price. An example is shown as the red line in Figure 3.6.

This, however, was not the final version because an oscillating behaviour was observed during training. The models were constantly jumping from too high to too low predictions, and vice versa. The reason for that were the gradients of the loss which were constant. Therefore, they did not contain any information about how far from the optimum the current solution is. The issue was solved by squaring the loss, which is visualized as the blue line in Figure 3.6. As a consequence, the gradients now increase with the increasing distance from the optimal value, which makes the training more stable. It is also worth noting that due to the definition of the monetary value, it is more important to predict correctly the demand for the expensive items. The loss is therefore influenced by the item price, as shown in Figure 3.7. The final formulas used in the objective function are written in Table 3.1.

	$prediction \leq demand$	$prediction > demand$
Loss	$((demand - prediction) \times price)^2$	$(0.6 \times (prediction - demand) \times price)^2$
1st deriv.	$2 \times price^2 \times (prediction - demand)$	$0.72 \times price^2 \times (prediction - demand)$
2nd deriv.	$2 \times price^2$	$0.72 \times price^2$

Table 3.1: Defined custom loss and its first and second derivative with respect to the prediction.

3.3.2 XGBoost

XGBoost (eXtreme Gradient Boosting - XGB) is one of the most prominent machine learning frameworks and therefore one of the first models we tried our data on. The gradient boosting library is known for its performance and speed especially in data science competitions. Due to the easy implementation, it is a good choice for initial tests on the data. Another reason for our choice is the possibility to extract feature importance, which helped us to understand which information is more useful for predicting future sales [13].

One downside of *XGBoost* is that it only accepts numerical values [13]. Categorical data needs to be one-hot encoded before being fed into the algorithm, which leads to higher numbers of features (in our case over 600), thus making it difficult to keep a clear overview.

Nevertheless, *XGBRegressor* outperformed most of the time the other methods we tried. In contrast to *XGBClassifier*, *XGBRegressor* is used to predict continuous outcomes so called "regression tasks" rather than categorical outcomes. In our case, *XGBRegressor* was the clear solution to our sales prediction problem.

Using hyperparameter tuning with grid search, we achieved promising results in the end. The performance and results compared to other methods are illustrated in Chapter 4.

3.3.3 CatBoost

In order to benchmark *XGBoost*, we trained multiple different boosting models. Thereof, *CatBoost* (Categorical Boost) turned out to be the most competitive one and even outperformed *XGBoost* when implementing without further techniques like our stacking and split approach (see Section 4.2).

CatBoost is a modern boosting approach developed by the Russian tech company *Yandex* in 2017. It is known for its good performance when using many categorical features due to the ordered boosting approach. By default, one-hot-encoding is applied if - and only if - a categorical feature has two different categories. This ensures user-friendly handling of the categorical features, reduces the number of features, and accelerates training time. Another specification of *CatBoost* is its low prediction time, achieved by building symmetric trees.[14].

Similar to *XGBoost*, *CatBoost* is offered in a publicly available package with an extensive documentation on *catboost.ai*. Therefore, we were able to apply grid search for hyperparameter tuning and use our own evaluation and objective functions. As can be seen in the next chapter, the results were promising, hence *CatBoost* will definitely be one of our first choices when tackling any future data mining problem.

3.3.4 Stacking

Beside the mentioned boosting methods, *stacking* is an often used ensemble technique to combine the strength of single models. This is achieved by connecting different models in series. When stacking two models for a regression task, the first model create the estimation on the original input data. The second model then take the estimations from the first model, and correct them according to the given labels. Here, it is possible to not only feed in the estimations of the first model, but also to feed in the input features of the initial model.

Within the competition, we tried two variations of the stacked modeling. First, we predicted the cumulative total demand of all the items during the prediction period, and used the predictions as additional inputs for the latter *XGBoost* model. Second, we trained two *XGBoost* models as well as a *CatBoost* model on our input space. Eventually, an *XGBoost* model was built on top of the three ground models. The predictions by the stacking models were slightly less accurate comparing to the final chosen model as introduced in the next section.

3.3.5 Split

The last variation tested was a model composed of two independent submodels: the first one predicting the demand for an item only on promoted days, and the second one predicting only on non-promoted days. Outputs of these submodels were then merged accordingly to produce the final prediction.

Both *XGBoost* and *CatBoost* models were used with this approach. The described modification of the *XGBoost* model turned out to be our best performing model, as explained in the next chapter.

4 Model Evaluation

This chapter describes our approach to test and compare different models, and presents the final results.

4.1 Model Comparison

We first split the dataset into multiple parts, as shown in the upper row in Figure 4.1. After training and validating, each model predicted the demand for the first test period from June 2nd to June 15th (Test 1). This prediction was used to compare the models and select the one generating the highest monetary value. Figure 4.2 shows the results of this comparison. To check that the selected model (*XGBoost - split*) is not overfitting to the first test period (which could have happened since the model was selected solely based on its performance on this period), we tested it on the second one from June 16th to June 29th (Test 2). The model generated the monetary value of €4.73M out of the possible €7.68M, and, therefore, we concluded that it generalized well.

Figure 4.1: Train, validation, and test periods during model selection and during training of the final model.

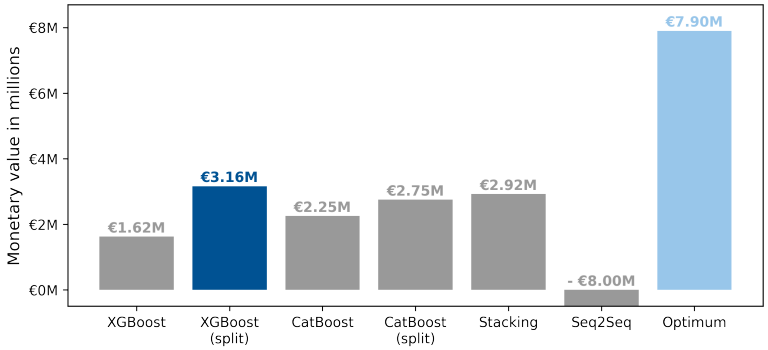

Figure 4.2: Monetary value in millions generated for the test period June 2nd - June 15th

To obtain the final submission, the time from June 30th till July 13th was set as the test period[1], while the two weeks prior (June 16th till June 29th) were treated as validation, and the rest as the train period. The final splits are shown in the bottom row in Figure 4.1.

4.2 Analysis of the Selected Model

To get some more insights into our final model, the differences between the model predictions and the true demand in the validation period are visualized in Figure 4.3. The plot depicts the comparison for the first 50 items[2] where either the prediction or the true demand was greater than 0. The predictions do not match the true demand exactly, but one can observe that the model was able to capture the main trend.

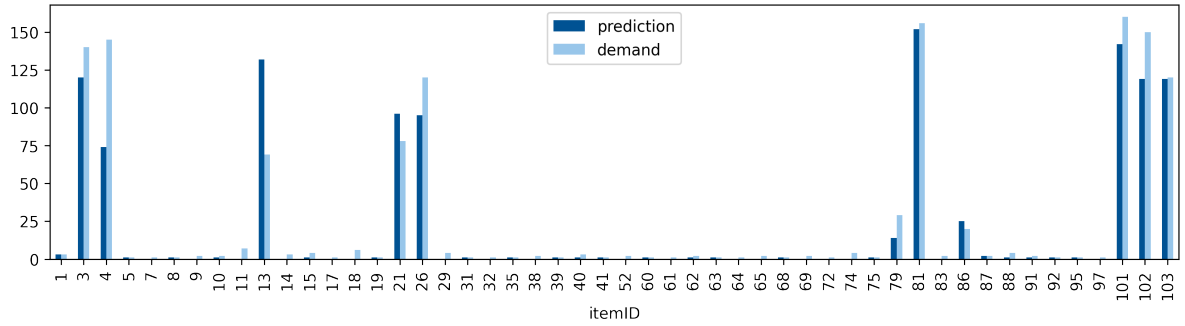

Figure 4.3: Comparison of the predicted and the true demand for some selected items.

4.3 Final Results of the Prudsys Data Mining Cup

With the achieved monetary value of €3,507,974.84, we ranked 13th in the Prudsys Data Mining Cup. The winner of the competition achieved a monetary value €4,305,017.89, however we did not receive any information about the theoretically highest possible monetary value.

[1]This time range was set by the Prudsys Data Mining Cup as the final evaluation period.
[2]Sorted by item ID.

5 Discussion

The Prudsys Data Mining Cup is an awesome opportunity to learn more about data science techniques, and apply them in a real world business context. Our main takeaways are definitely the importance of feature engineering and that different models perform differently depending on the exact use case. The latter implies to focus on getting models running early to be able to select the final model in a goal-oriented way.

In feature engineering, the main challenge was to define the ideal promotion feature in order to account for the peaky sales pattern. Therefore, one should always conduct research about the business context of the data mining use case. For example, our feature engineering process profited enormously from knowing that our retailer was an online auction shop.

Implementing a custom evaluation and objective function helped us to understand deeply how the applied models work. Using the competition's evaluation metric instead of RMSE, enabled us to apply the models in an optimized way on our use case.

As mentioned in Section 3.2, we trained other models like a *Seq2Seq* deep learning model. Our learning is that although deep learning is known to be very powerful, it might not work always in time-series settings with a lower amount of training data.

Nevertheless, the amount of data was large enough to cause memory issues on our private machines. Using a virtual machine from the LRZ supercomputing centre at TUM helped us to process all the data in the end, while we worked on a reduced dataset initially.

To recap on our final results, even at the last presentation at TUM, our predictions did not perform as well as we intended. We were confident that this was caused by the unusual peaky sales patterns, as our predictions were often too far off on those peak days. Applying the split (and stacking) approach as well as building separate models for promotion days and non-promotion days turned out to be the best performing approach. This way we were able to more than double the monetary value compared to the result presented in our final presentation a few days prior to the final submission.

Finally, an administrative challenge was definitely to organize a team of seven students. We approached this with our iterative team rotation. Due to the fun task, all team members were motivated to code by themselves and contributed proactively to our project. Therefore, our team is already looking forward to apply to other data mining courses at TUM like the Data Innovation Lab.

Although missing a finish in the top 10, being 13^{th} in the overall competition is still a solid result considering the close gap to the winning teams. Something we could improve in the future is to use more time in the end to remove less important features and reduce overfitting. All in all, we gained much experience in how machine learning can help businesses to optimize their returns. We look forward to applying our knowledge in the future to help companies gain valuable insights by unleashing the full potential of their data.

List of Figures

List of Tables

Bibliography

[1] prudsys AG. *Data Mining Cup*. 2020. URL: https://www.data-mining-cup.com/ (visited on 07/25/2020).

[2] G. Tunnicliffe Wilson. *Time Series Analysis: Forecasting and Control,5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1*. Vol. 37. Mar. 2016, p. 1.

[3] W. Wei. *Multivariate Time Series Analysis and Applications*. Feb. 2019, p. 1. ISBN: 9781119502852.

[4] N. Bajpai. *Business Statistics*. 2009, p. 574. ISBN: 9788131726020.

[5] E. ROSCA. "STATIONARY AND NON-STATIONARY TIME SERIES". In: *The Annals of the "Stefan cel Mare" University of Suceava. Fascicle of The Faculty of Economics and Public Administration* 10 (Jan. 2011), pp. 177–186.

[6] Jagan. *Time series Basics : Exploring traditional TS*. 2018. URL: https://www.kaggle.com/jagangupta/time-series-basics-exploring-traditional-ts/log (visited on 07/21/2020).

[7] X. Wang, K. Smith-Miles, and R. Hyndman. "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series". In: *Neurocomputing* 72 (June 2009), pp. 2581–2594.

[8] H. Allende and C. Valle. "Ensemble Methods for Time Series Forecasting". In: vol. 349. Oct. 2017, pp. 217–232. ISBN: 978-3-319-48316-0.

[9] Arturus. *Kaggle Web Traffic Time Series Forecasting*. 2017. URL: https://github.com/Arturus/kaggle-web-traffic (visited on 07/21/2020).

[10] I. Sutskever, O. Vinyals, and Q. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems* 4 (Sept. 2014).

[11] V. Kuznetsov and Z. Mariet. "Foundations of Sequence-to-Sequence Modeling for Time Series". In: (May 2018).

[12] Arturus. *How it works*. 2017. URL: https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md (visited on 07/21/2020).

[13] T. Chen and C. Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[14] P. et al. "CatBoost: unbiased boosting with categorical features". In: *Moscow Institute of Physics and Technology, Dolgoprudny, Russia* (2017), pp. 1–11. URL: http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf.