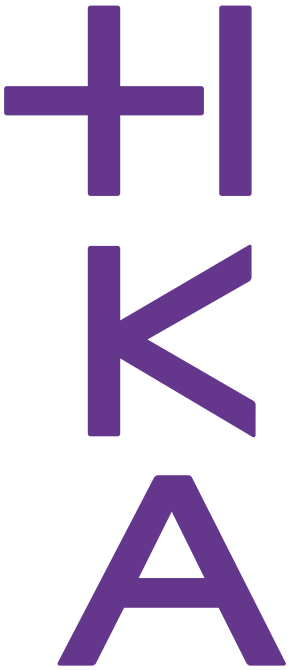
PREDICTING FUTURE SALES
using linear regression

Cédric Uden
cedric@uden.de

Computer Science and Business Information Systems
Hochschule Karlsruhe, University of Applied Sciences



September 15, 2021

The goal of this research project is to use the "Future Sales" dataset to create a performant prediction model. This paper covers the steps undertaken for the EDA, feature engineering and the training of the model. The focus is on a linear regression model.

Table of Contents

1 Introduction

The goal of this research project was to utilize the "Future Sales" dataset from the Kaggle competition "Predict Future Sales" to create a machine learning prediction model. The model was then used to predict the sales of the upcoming month for individual items, across selected branches spread within the country. The evolution of this project is documented on this GitHub.

The data consists of $2'935'849$ records of individual sales from different items, span across 2 ½ years[1], from a Russian software firm "1C Company". The task implied obtaining a grasp of the data by performing an exploratory data analysis (EDA) and using the newly gained domain knowledge to craft new features. The data consists not only of the items, shops and the sales itself, but also the name of the shops which include the cities name, the item price and their category.

After composing the feature engineered dataset, we then applied it to different linear regression models. These turned out to be well suited to tackle this problem. The models were then further examined with various data manipulation techniques such as scaling and regularization. Additionally, we inspected the impact of a time series analysis using ARIMA along the way.

To proceed, comparisons were made between observations originating from the EDA such as dropping certain outliers and removing shops which are exclusively considered in the training set. Finally, different hyperparameters and regressive models were compared using cross validation to evaluate their varying impacts on performance.

2 Exploratory Data Analysis

2.1 Brief overview

To get a feel for potential seasonal irregularities as well as for the general trend, we summed up the monthly sales of all shops and items combined.

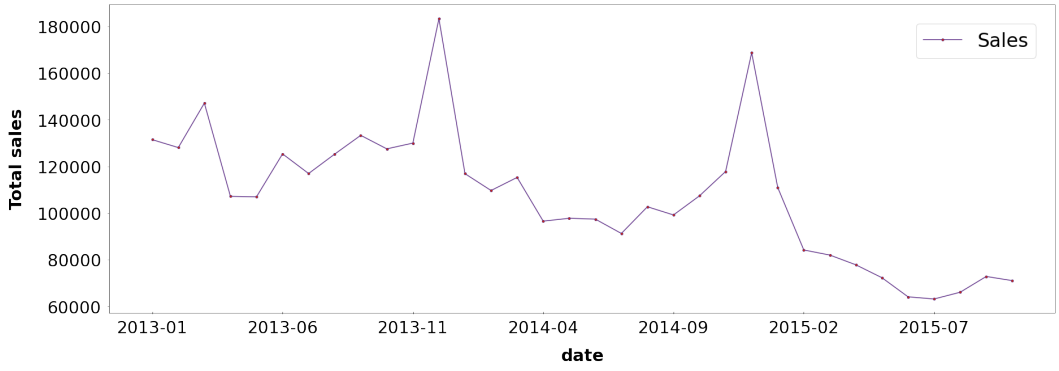


Figure 1: Total sales of the company

[1] 33 months to be precise, these are referenced as `date_block_num` in the project

In figure 1, we can observe that the sales peak in the month of December. This could be due to the increasing demand of gifts and disposable income from the population but could also indicate special Christmas sales which are common at this time of the year. Additionally, we observe a slight decline in demand over the timespan of the dataset.

To compare and confirm the above statements, we plotted the monthly revenue of the company:

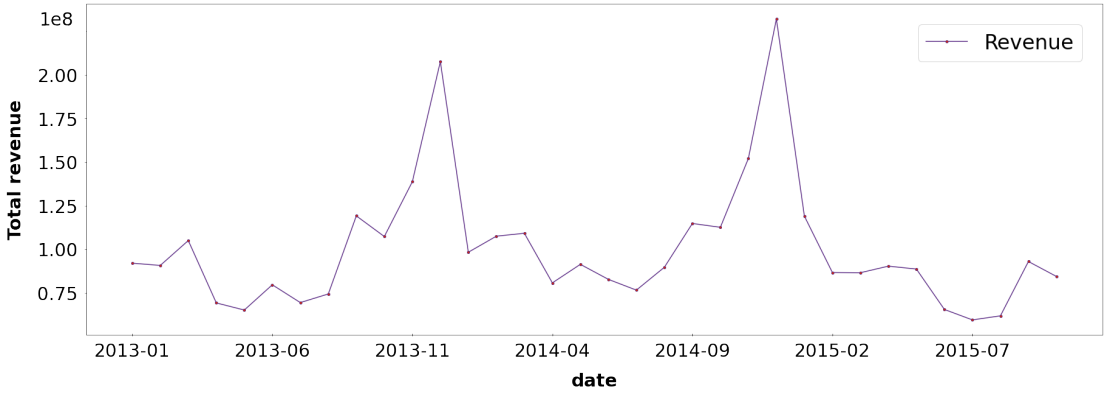


Figure 2: Total revenue of the company

Figure 2 confirms the seasonality of the data. Meanwhile, the downwards trend does not appear to be of relevance for the company. While having fewer sales, the revenue stream has a relatively foreseeable development over the years. We should hereby take note of the trend of having fewer sales, which consequently implies more expensive items.

To finish off the brief overview, we are looking at the correlation between the features and the label in figure 3.

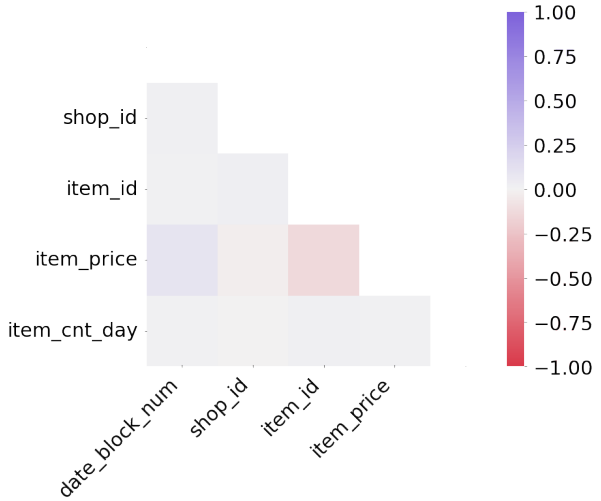


Figure 3: Correlation matrix

As we can observe, the label `item_cnt_day` has no strong correlation between any of the given features. This most likely leads to difficult to predict models, as no clear characteristics of the label is found in relation to the various features.

Additionally, no high correlation between features is observed. Therefore, we are not worried about multicollinearity now. [1]

2.2 Inspect outliers

In this section, we are going to examine potential outliers in the training data and observe their distribution. As the competition guidelines suggest to clip the true target values to 20, we can safely ignore the outliers from the label as these will be clipped either way.

Figure 4: Outliers of the feature `item_price`

Figure 4 shows an unorthodox looking boxplot. This is due to the quartiles, median and whiskers all being concentrated around the median price of 890. This plot helps us identify any outliers. After inspecting the single outlier, we can be sure that the record is not faulty: the item in question, item nr. 6066, is a single record of a software license for 522 people.[2]

Fortunately, upon further examination, we were able to verify that the item in question is not present in the testing data and therefore not relevant for the final prediction.

2.3 Distribution of sales among shops and across item categories

We are now going to examine the distribution of the total sales in regard to the different shops as well as the different item categories. The goal is, again, to detect potential outliers and to get an understanding of how the sales record came about.

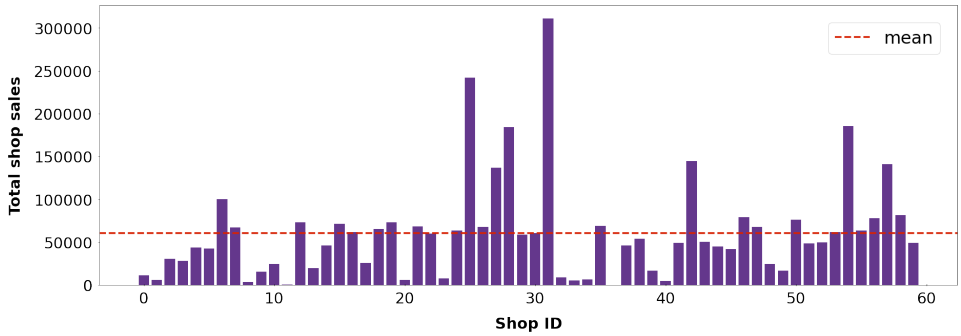


Figure 5: Distribution of total item sales across the different shops

[2]Translation - Software

As figure 5 shows, the shops in the center of the graph appear to be very prominent. This makes sense when we take into consideration that the shops with ID's 20–32, are all located in Moscow, the capital of the country. The next prominent shop in the graph, shop nr. 55, is again easily understood after finding out that this is the digital shop of the company. The digital shop is common to all stores across the country which explains its high sales count.

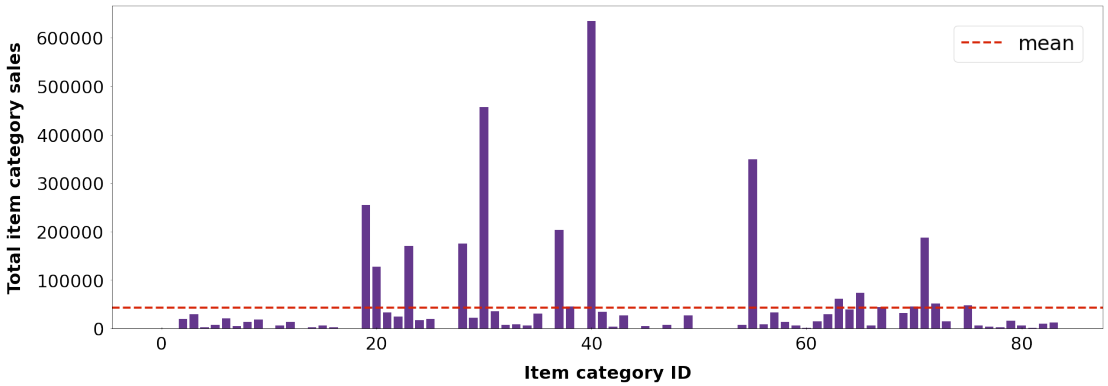


Figure 6: Distribution of total item sales within each item category

Now, when inspecting figure 6, the distribution of sales within a given item category, we can see that the space division is uneven. When diving into the details, we can relatively easily make sense as of why this came across. The three most prominent item categories are namely: PC games, movies & music. With the importance of the entertainment industry, these results are relatable.

In contrast to the best-selling items, we have inspected the categories which appear to be non-existent at first glance. After making sure that no category is missing any records, the worst selling item categories are simply explained after finding out what they are all about: a mix of niche and novelty items, as well as books. The lowest selling categories, with less than 10 sales each, are: accessories for an outdated gaming console, games for the Apple Mac OS platform and various categories of books.

2.4 Understanding the test data

The test data contains a total of $214'200$ entries to be predicted. Considering the total amount of records in the train data, the relation of 13.7 train records per test record does not appear to be as significant as one might presume. This surprising low number highlights the importance of a thorough feature engineering in order to create a robust model.

Before that, we are looking at how the test data came about. After isolating the `shop_id` and `item_id` duplicates, the number $214'200$ is quickly demystified: there are 5100 unique items

and 42 (of 60) unique shops. This factors up neatly to $214'200$. In conclusion, this tells us that there are exactly 5100 items in each (present) shop to be predicted.

Finally, the description of the Kaggle competition hints to inspect if there are any new items in the test data (which are at no point in time present in the train data). Indeed, after subtracting two sets of the total items from both datasets, we can identify that there are 363 new items, hence a total of $363 \cdot 42 = 15'246$ entries that have never been seen by the trained model.

3 Feature Engineering

3.1 Create zero sales

The training dataset tells us exactly which item got sold when and where. At the same time, this tells us exactly which shop did not sell any item on a given date. With this information, we can craft the possibly most important feature of them all: add the records from a possible combination of a given month, all shops and all items. Then fill the sale record with 0. This operation was dubbed "create zero sales". The mathematical function of this operation can be expressed with the following cartesian product:

$$\{\texttt{date\_block\_num}_i\} \quad \times \quad \{\texttt{shop\_id}\} \quad \times \quad \{\texttt{item\_id}\} \quad | \quad i \cap [0;33], \quad i \in \mathbb{N}$$

This operation was implemented using the Python function `product` from the `itertools` package from the Python standard library.[3]
N.B. that this ignores the months which have not sold anything in the considered time period.

3.2 Shop information

To add detailed information about the shops, we first identified the city in which a shop is located. To continue from there, we researched geographic and demographic data. This information was added to a spreadsheet and exported as comma-separated values (CSV) file in a similar manner to the original dataset. The following features have been added: `zip_code`, `region`, `population`, `population_growth` and `region_gdp`.

The population growth was considered, on a city level, from a time frame roughly between 2010–2018 (varying according to different sources from discovered materials).

The population and gdp (per capita) information were primarily taken from the year 2018. Again, this was due to the abundance of information publicly available for this period.

[3] `itertools` - official documentation

3.3 Category information

Compared to the shop information, the process of arranging detailed features for the various categories was relatively arbitrary. After translating all 84 categories, some were very specific and hereby intuitive, for instance the category 12 “Game consoles - PS4”. This could easily be split into a type (“game console”), a dedicated device (“ps4”), platform (“playstation”), manufacturer (“sony”) and a medium (“physical”).

The same distinctions were harder to make for other categories like “Gifts - Stuffed Toys” or “Service”. As these can be interpreted very differently, we attempted to create meaningful splits to our best intentions. The resulting features can be found in the CSV file on GitHub.

The feature `category_is_fancy`, evolved from the distinction in certain existing categories like “PC Games - Standard Editions” and “PC Games - Collector’s Editions” which expects the latter to have a narrower target audience for a same title.

3.4 Lag features

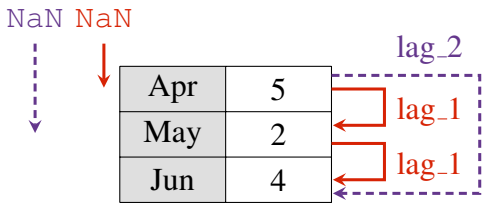


Figure 7: Creating lag features

In the next step, we have created a set of lag features. The adjoining figure 7 illustrates their implementation. The idea is that, in a time series, every label y_t, in function of its time t, is dependent on its preceding value y_{t-n}. n is chosen to suit the time interval of the problem, for instance $n \cap \{1; 2; 3; 6; 12\}$ to consider the past two months, trimester, semester and year in a monthly spaced time series. Consequently, the lag_n series shifts the label for n intervals forward in time. To apply this feature on several different existing features, the following function has been implemented:

```python
def compute_lag_feature(df, lags, features):
    id_columns = ['date_block_num', 'shop_id', 'item_id']
    # N.B. `cl` concatenates elements from a lists with other elements
    for n in lags:
        shifted = df[cl(id_columns, features)].copy()
        shifted.columns = cl(id_columns, f"{features}_lag_{str(n)}")
        shifted['date_block_num'] += n
        df = pd.merge(df, shifted, on=id_columns, how='left')
    return df
```

Listing 1: Compute Lag Features (in Python)

The function seen in listing 1 receives following parameters: a DataFrame `df`, the intervals `lags` in which the features are intended to be lagged and the features which are lagged. Then, the individual lags are iterated over: begin by storing the relevant columns from the DataFrame, add a new column, shift the date `n` forward and merge the newly created feature back onto the passed DataFrame.

3.5 Price related features

Primarily to avoid throwing the new items from the test set under the bus (identified in section 2.4), a variety of average prices has been calculated. This includes the average price of a given item overall, as well as the average price an item has in each shop and the average price of items in a given category. This is intended to nudge the model in a certain direction.

On top of that, a trend feature has been added. This is effectively a lag of the item price. This indicates the direction of item prices. This accounts for the trend observed in figure 1 and figure 2, which imply fewer sales with rising item prices over time.

3.6 Additional features

To finish off the feature engineering, a few selected features have been crafted representing additional information gathered from the time data. A few are simple features, such as the month of the year. This is intended to take the seasonality into account. A feature which holds the number of days in a month, with intents to explain the difference in sales when having a month with 31 days or 28 days, thus having roughly 10% more days where the shops are open for business.[4]

Another set of features, which has been a little bit trickier to implement, is the record of when an item has been initially and most recently sold on a companywide and on a given shop basis. To achieve this, we created a set for each feature respectively, and stored a key pair `(item_id, shop_id)` with the resulting dates. Finally, the set has been matched to the corresponding rows in the DataFrame.

4 Model Training

After creating a variety of different features, we are now going to apply them on different types of models and compare their performance.[5] To best compare the results, a common metric has been

[4] Shops in Russia are generally open 7 days a week

[5] A few results listed below date back to a previous version of the feature engineered dataset and may vary slighly in the final version. The conclusions drawn from the impact of a particular method remain intact.

chosen (in particular to match the Kaggle competitions evaluation method): root mean squared error (RMSE).

4.1 Linear Regression

In this section we are incrementally going to inspect the various parameters and inputs applied to the model and examine their direct impact. All steps have been documented in separate Git commits and linked to the respective commit for reference.

4.1.1 Raw data ⇒ *Git link*

To start off, we determined a base score to proceed from. We simply took the train data as is, provided at the start of the competition and applied a basic `LinearRegression`[6] on the data.

RMSE : **8.10506**

4.1.2 Clip raw data ⇒ *Git link*

Now, as noted in section 2.2, we will apply the intended clip to the $[0; 20]$ range to the train values. After this step and reapplying them onto the very same input data, the score increased significantly.

RMSE : **2.56222**

4.1.3 Add zero sales ⇒ *Git link*

In this step, we have added the zero sales, referenced in section 3.1, onto the training data. We can again see a significant increase in the model's performance.

RMSE : **1.22789**

4.1.4 Add feature engineering ⇒ *Git link*

Finally, the — long anticipated — feature engineered data. How will the model perform now? We can again observe a significant leap in performance.

RMSE : **0.80112**

[6] Link to scikit-learn documentation

4.1.5 Applying `cross_val_score` ⇒ *Git link*

In order to get a more realistic approximation of the final score on the test data, we have now switched to the more sophisticated `cross_val_score`[7] evaluation. This action is significantly heavier on execution time, as the same input data is split into various validation sets internally (5 by default). We can see that our `train_test_split` selection got lucky before, as we observe a small dent in performance.

RMSE : **`0.82563`**

4.1.6 Dropping superfluous shops ⇒ *Git link*

As observed in section 2.4, about 30% of the shops present in the train data are irrelevant to the final test data. In theory, this should not change anything when performing a split on the train data. But, maybe the authors of the competition had certain irregularities (such as the detected price outliers in section 2.2 which is confirmed to be irrelevant for the test prediction) in mind when setting the environment for the competition. Our result was satisfying:

RMSE : **`0.81432`**

4.1.7 Applying LASSO regression ⇒ *Git link*

Next up, we attempted to apply the data to a LASSO model. The result was very shocking: the performance worsens significantly.

RMSE : **`0.98846`**

4.1.8 Applying Ridge regression ⇒ *Git link*

Now, onto the Ridge model. We were now back to our original performance:

RMSE : **`0.81432`**

4.1.9 Applying Ridge cross-validation ⇒ *Git link*

As a final evaluation of different regularization attempts, we tried a cross-validation to determine the best hyperparameters. To our surprise, this did not yield any improvement either:

RMSE : **`0.81432`**

[7] `cross_val_score` documentation

4.1.10 Scaling the data ⇒ *Git link*

To finish the comparisons, we examined the effects of applying a scaler to the data prior to the model training. This did not have any significant impact on the performance.

RMSE : **0.81148**

4.2 ARIMA forecast

Another part of this project was to understand the handling of a forecasting model in a time series. For this, the autoregressive integrated moving-average (ARIMA) model was used. The ARIMA acronym is divided into three parts: [2]

- **Autoregressive**
  Describes the dependency among successive observations.
- **Integrated**
  Number of differentiations needed to make a nonstationary time series stationary.
- **Moving Average**
  Describes the impact of a random shock from one observation to the next.

With ARIMA being intended to be used for univariate problems [3], we had to modify the data a little bit and create our own narrative. Therefore, we analyzed the data and isolated the item, item nr. `5822`, that had the most sales across the time series.

To get a comparable evaluation to the linear regression, we applied the model on the raw data: this gave us a RMSE of `5.85736`. After analyzing the data and finetuning the ARIMA parameters, we were able to increase the performance up to `5.64160`. Very disenchanting compared to the linear regression model. The final forecast can be observed in figure 8:

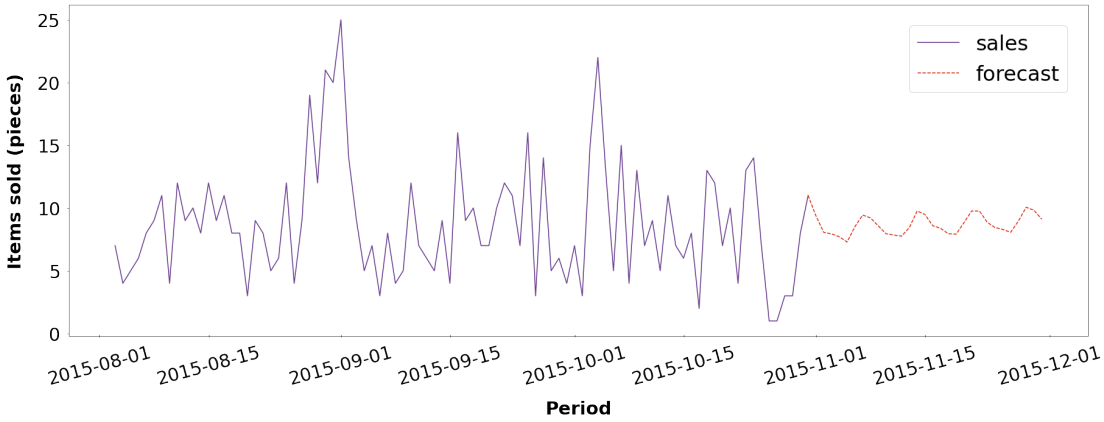


Figure 8: ARIMA forecast of item `5822`

The step-by-step procedure on how to finetune the ARIMA parameters can be found here.

5 Next Steps

The following steps were acknowledged, but unfortunately would breach the scope of this research project at the time being.

This includes simpler steps, like extending the zero sales from its current state, which only matches item and shop combinations on a given month, to the entire timespan. This would inflate the current number of rows, approximately 11 million — which are already now cumbersome to handle — by 4 times.

Next, we would take a closer look at the final features and potentially drop features that are having a negative impact on the model's performance. For this, we would have to reevaluate the correlations between features to detect potential multicollinear values. Additionally, verifying the feature importance of individual features would be beneficial.

Then, we would aim for a more throughout evaluation of different modelling techniques such as decision trees or LSTM. Or applying different frameworks such as `xgboost` or `LightGBM`. These predictions can then be utilized using ensemble learning in order to increase the performance even further. This will have to be postponed for the time being.

6 Conclusion

Dealing with the numerous challenges of handling large datasets along the road of this project has been incredibly frustrating at times. In the end, sticking to the goal and overcoming these obstacles has been immensely rewarding.

The most notable takeaways from the project were the importance of an EDA, how to craft new meaningful features in a time series problem and the impact of various regularization techniques.

The EDA was a perfect showcase of the impact of gaining domain knowledge on a given problem. Understanding how the records of the outliers came about and spotting the train data which is irrelevant towards the test data. These all led to solid increases in performance.

New concepts like lag features were discovered and opened up a realm of possibilities when working with time series in the future. These were of particular importance when real world data brought previously unknown items into the mix.

Finally, we learned a lot when applying various regularization models, which surprisingly revealed that hyperparameter tuning does not always imply improvements in performance.

A Bibliography

[1] B. G. Tabachnick and L. S. Fidell. *Using multivariate statistics*. 7th ed. Pearson, 2018, pp. 88–91. ISBN: 9389342236.

[2] B. G. Tabachnick and L. S. Fidell. *Using multivariate statistics*. 7th ed. Pearson, 2018, Chapter 18–2. ISBN: 9389342236.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed. O'Reilly, 2017, pp. 671–673. ISBN: 1492032646.

B Glossary

DataFrame Datatype from the Pandas library. Two-dimensional, size-mutable, potentially heterogeneous tabular data.... Link to the full documentation. 7

Future Sales Name of the dataset from the Kaggle competition. Link. I, 1

hyperparameter Parameter that is set before the learning process. These can be tuned according to the statistical models that are employed and has a direct impact on the performance. 1, 9, 11

scikit-learn Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning... Link to the full documentation. 8

C Acronyms

ARIMA autoregressive integrated moving-average. 1, 10

CSV comma-separated values. 5, 6

EDA exploratory data analysis. I, 1, 11

LASSO least absolute shrinkage and selection operator. 9

LightGBM light gradient boosting machine. 11

LSTM long short-term memory. 11

RMSE root mean squared error. 8–10

D List of Figures

E List of Listings