UDACITY MACHINE LEARNING NANODEGREE CAPSTONE PROJECT SOLUTION

A PREPRINT

Thomas Rogenhofer
Udacity Machine Learning Nanaodegree
Udacity
Mountain View, CA 94040

June 8, 2019

ABSTRACT

In this paper, I present the results of the Udacity Machine Learning Nanodegree Capstone Project. In the beginning, I introduce the domain background of the project. This provides the reader with sufficient information to follow the problem statement. Once the problem is discussed, I describe the datasets and inputs and present a solution to the problem. At the end of this project, I present the results and a refelction on the findings.

***K*eywords** Capstone Project Solution · Google Analytics Customer Revenue Prediction

1 Definition

1.1 Project Domain Background

The Pareto Principle states that for many events, about 80% of the effects come from 20% of the causes [1]. This is an observation that holds true for many e-commerce platforms, as only a small percentage of customers generates most of the revenue. With this observation in mind, marketing managers must answer three questions in the context of a marketing campaign: how often to make an offer (frequency), when to make an offer (timing) and whom to contact (target group selection) [2].

This capstone project addresses the question of target group selection for a Google Merchandise Store, also known as GStore. In this store, customers can buy all sorts of Google related merchandise like T-shirts, bags, drinkware and other items. The GStore can be found at

`https://shop.googlemerchandisestore.com/`

1.2 Problem Statement

As discussed in the previous chapter, marketing managers of the GStore must identify those customers that generate most of the revenue. These customers can then be contacted in future marketing campaigns, or in real-time when visiting the GStore. Learning the user intent in real time has become a vivid area of research and personal visits to online stores often prove to be rather unsatisfactory in terms of customer experiences [3].

For this project, we need to predict the natural log of the transaction revenue for a single customer visit. This means that for every visit, the target is defined as follows:

$$target_{visit} = \ln(y_{transactionRevenue} + 1) \tag{1}$$

This formal description of the problem fulfills the requirement that the solution must be quantifiable, measurable and replicable. In addition, with this formal description in place, we can evaluate a number of different machine learning algorithms, in order to find the best solution to our problem.

1.3 Solution Statement

As a first step towards a solution, I performed an exploratory data analysis (EDA) in order to gain a better understanding on the data and relationships within. This was followed by a number of feature engineering tasks that prepared the data sets for a number of machine learning algorithms. Finally, I will create a benchmark model based on default parameters and compare this benchmark model to other machine learning algorithms in terms of performance.

The solution to this capstone project requires a model that uses historical customer data to make a prediction on future customer spending. As we are going to predict a continuous quantity, our solution can be classified as "regression predictive modeling". Generally speaking, this the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). I selected four machine learning algorithms that have proven effective for predictive regression modeling.

- LinearRegression (comparison model)
- AdaBoostRegressor (comparison model)
- SVR (comparison model)
- XGBRegressor (benchmark model)

2 Analysis

2.1 Data Sets and Inputs

The datasets for this capstone project are those, that are provided by the kaggle competition "Google Analytics Customer Revenue Prediction". They can be found at

`https://www.kaggle.com/c/ga-customer-revenue-prediction/data`

The dataset contains three different types of files and their purpose is as follows.

- *train_v2.csv*: a dataset for training the machine learning algorithms. It contains user transactions from August 1st, 2016 to April 30th, 2018.
- *test_v2.csv*: a dataset for testing the machine learning algorithms. It contains user transactions from May 1st, 2018 to October 15th, 2018.
- *sample_submission_v2.csv*: a list of customers for which a prediction of the sum of transaction is required in the context of the kaggle competition.

The participation in the kaggle competition is not within the scope of this project. Therefore, I will only use the first two datasets from the list above. In addition, the target value is not the sum of all transaction revenue per user, but the transaction revenue per visit. This approach reduces complexity and to focus on the performance of the models themselves.

Both *train_v2.csv* and *test_v2.csv* contain columns as listed in *Table* 1. The dataset has a number of characteristics that are important to note. First, each row in the dataset represents one visit of a customer to the store. Second, there are multiple columns in the dataset that contain JSON objects.

The target value *"transactionRevenue"* that I am going to predict is contained within the JSON object *"total"*. This will require data preprocessing prior to any rounds of model training and optimization. The technique for transforming JSON objects to usable machine learning data structures in described in *Section* 3.

A first look at the train data set shows a total number of 1708337 rows and 13 columns of data type *object* and *int64*. The *object* data types contain JSON objects with a number of data items that will prove relevant in the cause of training the machine learning algorithms.

The test data set contains a total number of 401589 rows and identical columns as in the train data set. We will see in *Section* 3, that the testing data set contains (after transforming the JSON objects) one feature less than in the training data set. This feature is *'trafficSource.campaignCode'*.

Table 1: Data Fields contained in the training and testing datasets

Column Name	Description
fullVisitorId	A unique identifier for each user of the Google Merchandise Store
channelGrouping	The channel via which the user came to the Store
date	The date on which the user visited the Store
device	The specifications for the device used to access the Store
geoNetwork	This section contains information about the geography of the user
socialEngagementType	Engagement type, either "Socially Engaged" or "Not Socially Engaged"
totals	This section contains aggregate values across the session
trafficSource	This section contains information about the Traffic Source from which the session originated
visitId	An identifier for this session This is part of the value usually stored as the _utmb cookie.
visitNumber	The session number for this user. If this is the first session, then this is set to 1
visitStartTime	The timestamp (expressed as POSIX time)
hits	This row and nested fields are populated for any and all types of hits.
customDimensions	This section contains any user-level or session-level custom dimensions that are set for a session.
totals	This set of columns mostly includes high-level aggregate data

2.2 Data Values and Quality

The transformation of JSON objects to features, or technically speaking pandas columns, extends the total numbers of features from 13 to 60 in the training data set and 59 in the testing data set, with *'trafficSource.campaignCode'* missing in the later one.

Except of column *'device.isMobile'*, all features originating from the JSON transformation are of type string and need to be converted to machine usable numeric values. Besides of this string to number conversions, a closer look at the data quality itself reveals the following findings. These findings hold true for the training and testing data set.

- Missing feature values by description: a number of features contain values like *'(not set)','not available in demo dataset','(not provided)','unknown.unknown','(none)'*
- Missing feature values by data type *pandas.NaN*: a number of features contains the pandas data type *NaN*.

Knowing that there a missing values, I continued the EDA by counting the totals of the missing values for each feature. Prior to this task, I transformed the missing values by description with techniques further outlined in *Section* 3. The result of counting the missing values for each feature is displayed in *Figure* 1.

Figure 1 shows that the target feature *"totals.transactionRevenue"* has a missing rate of 98.916256%. Additionally, the top 17 features from *Figure* 1 have no values at all. These features will dropped in the cause of feature engineering.

Another interesting finding from the EDA is that there are 16141 paying users (1323730 total) in the training data of which

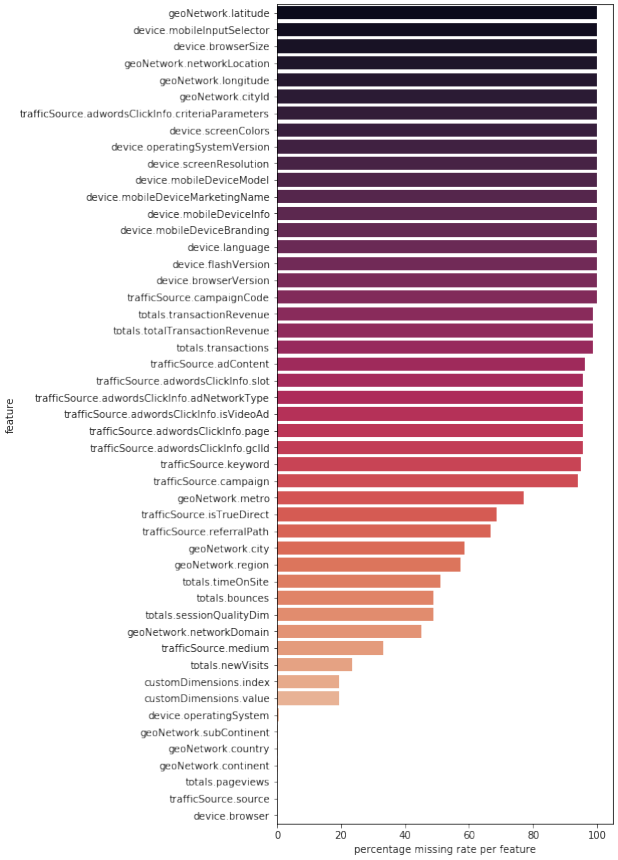


Figure 1: Missing rate for each feature

- 14600 users (90.4529% of paying) have 1 paid transaction,
- 1133 users (7.0194% of paying) have 2 paid transactions,
- 256 users (1.5860% of paying) have 3 paid transaction.

Only one user has (0.0062% of paying) 33 paid transactions and compared to the users, this user seems to be a real fan of Google merchandise. *Figure* 2 shows the small number of *log1* transformed transaction revenues for each user.

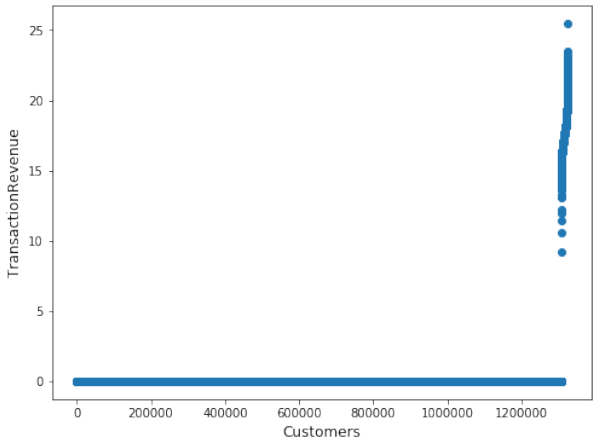


Figure 2: Total transaction revenue for each user

2.3 Target Value Distribution

Figure 3 shows that the target feature values are normally distributed. This observation holds true for the distribution of revenues per visit and for the distribution of total revenue for each user.

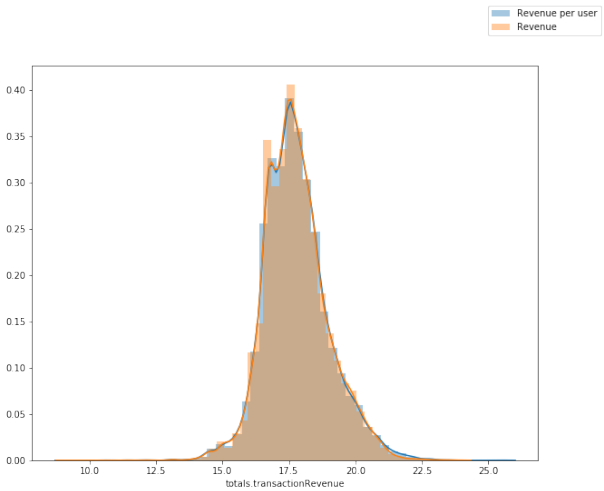
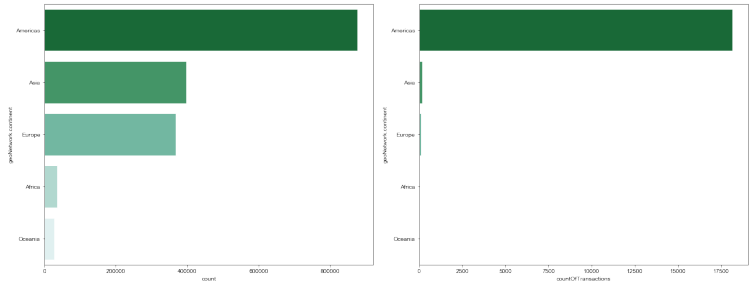


Figure 3: Distribution of single/total transaction revenue for each user

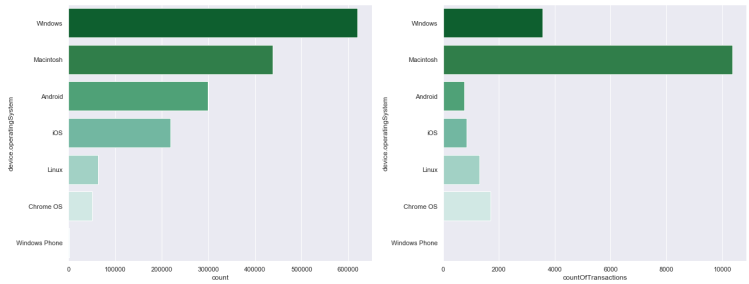
2.4 Google Store Customer Analysis

Looking deeper into who the customers of the GStore are, revealed some interesting details. These can be summarized as follows.

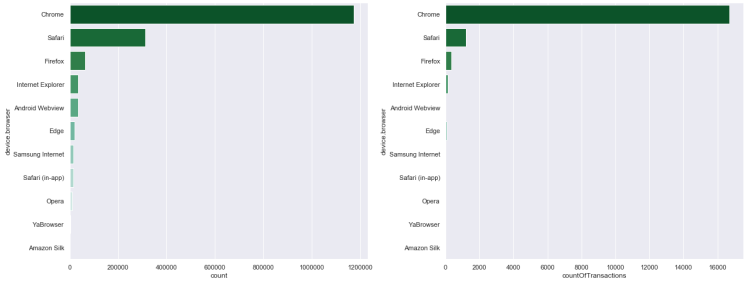
- *Figure* 4 (a) shows that most GStore traffic comes from North America, followed by Asia and Europe. When we further break down the traffic for each continent according to the actual revenue generated, we can see that almost all revenue is being generated from North America.
- *Figure* 4 (b) shows that in terms of a single operating system Macintosh OS generates most of the revenue.
- *Figure* 4 (c) shows that most traffic and revenue is being generated from Chrome browsers.
- Google is the main traffic source for the GStore, whereby most revenue is being generated by users who visit the GStore directly.



(a) Continent



(b) Operating System



(c) Device Browser

Figure 4: Transaction revenue grouped by different features

Taking all these findings into account, a GStore marketing team should focus on Mac Users as their most valuable source of customers. Once more, it seems to prove correct, that Mac Users represent a customer group with high purchasing power.

2.5 Patterns in Time Series Data

The training and testing data sets represent a time series of visits to the GStore. Therefore, it is important to investigate, whether there are observable patterns in the training and test data sets. *Figure* 5 illustrates clearly a pattern of increased visits to the GStore before Christmas holidays and a sharp drop afterwards. The same holds true for the revenue generated as shown in *Figure* 6. There are also peaks of visits around May and October, with a slight but steady increase of visits from June to October. The testing data set spans from May to October and a similar tendency can be observed there as well.

We will see later in the training of the models that the observed patterns in the time series data have an immediate impact on the techniques applied for k-fold cross validation. Randomly shuffling data (in this case visits to the GStore) into training buckets is no longer an option, as we risk loosing important information like the drop of revenue after the Christmas season.

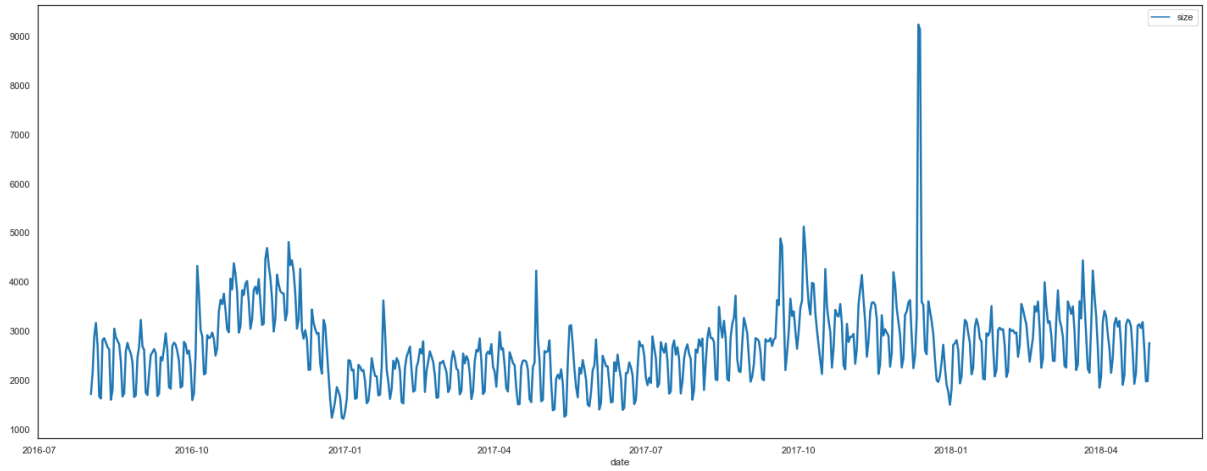


Figure 5: Total number of visitors

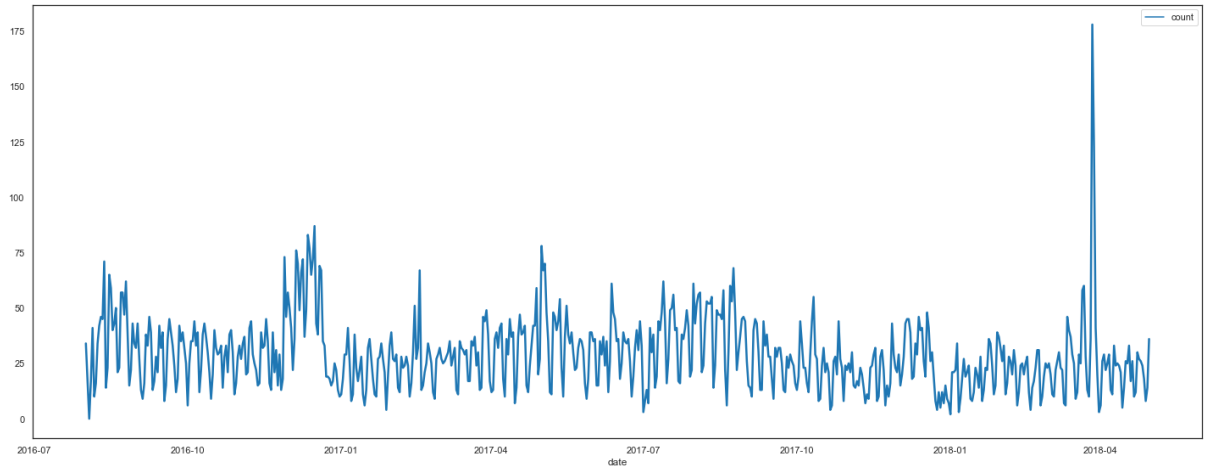


Figure 6: Total number of revenue

3 Feature Engineering

Feature engineering is an important building stone for the effectiveness of predictive models [6]. The effort required to engineer a useful data set can vary depending on the quality of the original data. In case of the GStore project, the data sets require a lot of engineering and careful consideration. This is mainly due to the encapsulation of data in JSON objects and the large number of missing values that were found during the EDA.

In order to transform the original GStore data sets from the Kaggel competition, I have applied the following techniques.

3.1 Feature Split

I have borrowed the algorithm for feature splitting from [7]. This algorithm extended the feature set for the training set to 60 features and for the testing set to 59 features (with missing feature *'trafficSource.campaignCode'* from training set).

The following features from the original training and data sets were subject to the feature splitting:

['channelGrouping', 'date', 'device', 'fullVisitorId', 'geoNetwork', 'socialEngagementType', 'totals', 'trafficSource', 'visitId', 'visitNumber', 'visitStartTime', 'customDimensions']

3.2 Imputation

In statistics, imputation is the process of replacing missing data with substituted values [9]. In case of the GStore data sets, the process consisted of various steps. I encountered during the EDA that missing values were represented by a number of different values. These could be of pandas data type NaN, but also in textual form like *'(not set)','not available in demo dataset','(not provided)','unknown.unknown','(none)'*.

Therefore, as a first step I transformed the textual missing data types to a *pandas.NaN* data type. This enabled me to use pandas count functions on *pandas.NaN*. As a second step, I transformed the *pandas.NaN* to number value 0, because machine learning algorithms perform better with the value 0 in comparison to *pandas.NaN*.

3.3 Deleting Features

At the end of the EDA it became clear that there were a number of features that had to be dropped for the following reasons:

- Some features were user session related and had no relevance for the prediction problem.
- The feature *'trafficSource.campaignCode'* was missing in the test data set and replacing it with approximation values was not an option.
- Some features had only *pandas.NaN* as value.

The decision on which feature to drop depends on a number of factors and can be considerably influenced by the expertise of the machine learning (ML) engineer and the ML algorithms to be applied. In other words, dropping features is not an exact science and the final prove awaits the ML engineer once the performance of the model has been evaluated. For this project, I decided to drop the following features:

["trafficSource.adContent", "trafficSource.adwordsClickInfo.adNetworkType", "trafficSource.adwordsClickInfo.gclId", "trafficSource.adwordsClickInfo.isVideoAd", "trafficSource.adwordsClickInfo.page", "trafficSource.adwordsClickInfo.slot", "visitId", "visitStartTime", "totals.bounces", "trafficSource.isTrueDirect", "trafficSource.medium", "trafficSource.campaignCode", "trafficSource.isTrueDirect", "customDimensions.index", "socialEngagementType", "geoNetwork.networkDomain", "customDimensions.value", "totals.totalTransactionRevenue", "totals.transactions", "geoNetwork.city", "fullVisitorId","geoNetwork.metro","totals.sessionQualityDim", "trafficSource.campaign", "trafficSource.keyword","trafficSource.referralPath","trafficSource.source","geoNetwork.region"]

3.4 Log Transformation

Log Transformation is a widely used technique for feature engineering. The advantages of this technique are amongst others:

- It reduces skewness in the data set, so that the distribution turns closer to normal.
- It normalizes the magnitude difference between different features.
- It reduces the negative effects of outliners due to the normalization of magnitude outliners.

The following features in the testing and training data set were *log1* transformed:

["visitNumber", "totals.hits", "totals.newVisits", "totals.pageviews", "totals.timeOnSite", "totals.visits"]

3.5 Cleaning Feature Values

I learned in the cause of the EDA, that the feature *'device.browser'* had a high number of meaningless data. As I assumed that feature *'device.browser'* might be relevant for performance of the models, I cleaned this feature off the irrelevant data. The following browser types were kept from the original data set:

['Chrome', 'Internet Explorer', 'Safari (in-app)', 'Edge', 'Safari', 'Firefox', 'YaBrowser', 'Opera', 'Android Webview', 'Opera Mini', 'UC Browser', 'Samsung Internet', 'Amazon Silk', 'Mozilla Compatible Agent', 'Coc Coc','Maxthon',

'Android Browser', 'Puffin', 'Playstation Vita Browser', 'Nokia Browser', 'Mozilla', '+Simple Browser','BlackBerry', 'BrowserNG', 'SeaMonkey', 'Nintendo Browser', 'Iron', 'GemiusSDK']

This cleaning task of removing approximate over 100 irrelevant data items also helped to keep the feature set small after One-Hot Encoding.

3.6 One-Hot Encoding

One-Hot Encoding is a technique for transforming categorical values to numeric ones. One of the challenges for one-hot encoding in this context was to keep the features of the training and testing sets identical. I implemented the following algorithm for One-Hot Encoding:

- One hot encoding of categorical features the in training data set.
- Store newly created columns in variable for later use with the testing set.
- One hot encoding of categorical features in the testing data set.
- Remove additional columns in test data set that are not in the train test data set.
- Add columns to the testing data set that are present in the training data set and not in the testing data set.

The following features were subject to one-hot encoding:

['channelGrouping','device.browser', 'device.deviceCategory', 'device.operatingSystem', 'geoNetwork.continent', 'geoNetwork.country', 'geoNetwork.subContinent', 'device.isMobile']

4 Methodology

For this capstone project, the selection of an appropriate model evaluation technique depends a lot on the time series nature of the training and testing data. Randomly splitting the training and testing data sets into k-folds (as outlined in my capstone project proposal) is not an option, because we might loose important information when training and validating the estimators.

Therefore, I have decided to choose a two-fold validation approach, depending on whether we are validating the benchmark model, or the *"competitor"* models.

For the *XGBRegressor()* benchmark model, I utilized the option of training and validating the performance of the model with the *XGBRegressor().fit()* built in parameter *eval_set*. While iterating through the total number of *n_estimators*, i.e. the number of trees to fit, the *XGBRegressor()* will generate a training and testing score. This approach has the advantage, that the time series data is kept in sequence and that I immediately receive feedback on the performance of the benchmark model.

The comparison machine learning algorithms will be subject to another validation strategy. They will be validated via the technique of k-fold Cross-Validation. The *sklearn.model_selection.TimeSeriesSplit* class supports splitting of time series data and can be used within *sklearn.model_selection.GridSearchCV* for hyper-parameter tuning. It splits the training data into so-called k-partitions or folds in sequential order without any random shuffling. For this capstone project, I extended the *sklearn.model_selection.GridSearchCV* from [8] with a custom-built RMSE scorer and prediction based on *sklearn.model_selection.GridSearchCV.best_estimator_* that is used by *sklearn.model_selection.GridSearchCV.predict()* .

One could argue, that the benchmark model should also be trained with *sklearn.model_selection.TimeSeriesSplit* and *sklearn.model_selection.GridSearchCV* for reasons of comparability. Nevertheless, I decided to use this two-fold approach, as I was ionterested to see, whether training the *XGBRegressor()* within *sklearn.model_selection.GridSearchCV* and numerous k-folds would lead to a better model performance.

4.1 Performance Metrics

I will apply the model performance measure of "Root Mean Squared Error" (RMSE) after each round of training and testing for the benchmark and comparison models. RMSE provides the average amount of error made on the test-set in the units of the output variable and can be formalized as follows.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2}, \tag{2}$$

where y hat is the natural log of the predicted revenue for a customer and y is the natural log of the actual summed revenue value plus one.

The function *XGBRegressor().fit()* provides the parameter *eval_metric='rmse'* for outputting the training and testing score as RMSE. For obtaining the training and testing scores of *sklearn.model_selection.GridSearchCV* as RMSE, I have utilised the class *sklearn.metrics.make_scorer* to custom-build a RMSE scorer.

4.2 Benchmark Model

I have selected the *XGBRegressor()* as benchmark model for this capstone project, because of the following characteristics [4].

- Execution Speed. XGBoost is really fast, when compared to other gradient boosting.
- Model Performance. XGBoost performs very well on regression predictive modeling problems.

I have trained and evaluated the *XGBRegressor()* with the following default parameters on the entire training and testing data set with the results as shown in *Figure* 7.

XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,colsample_bynode=1, colsample_bytree=1, gamma=0, importance_type='gain', learning_rate=0.1, max_delta_step=0, max_depth=3, min_child_weight=1, missing=None, n_estimators=100, n_jobs=1, nthread=None, objective='reg:linear', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None, silent=None, subsample=1, verbosity=1)

Figure 7 illustrates that the best RSME performance score has been achieved at the total number of *n_estimators=30* with RSME = 1.78241. The model shows clear signs of over-fitting, as it memorizes the training data very well and performs very poorly on the testing data. It will be interesting to see, whether training the models in *sklearn.model_selection.GridSearchCV* will achieve better performance scores.

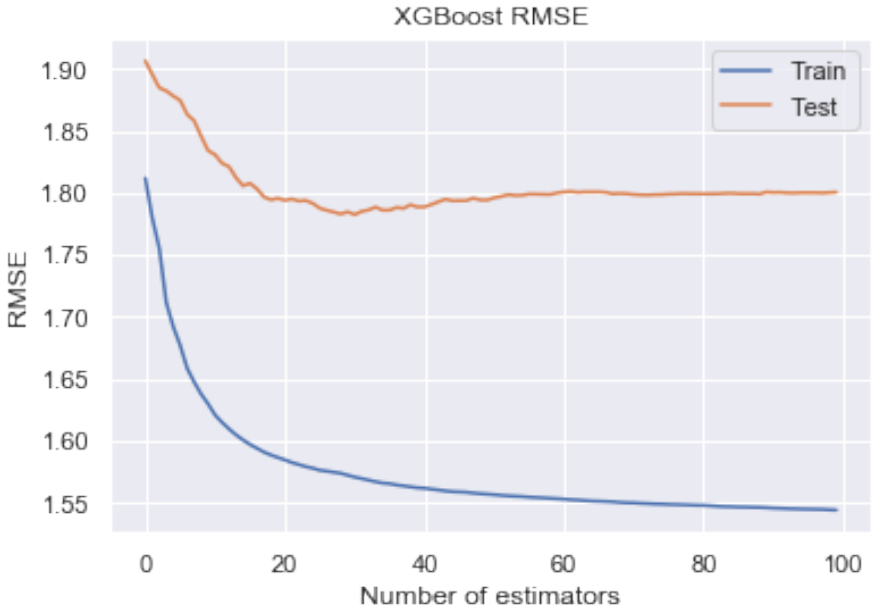


Figure 7: Training Testing Scores of Benchmark Model

5 Results

5.1 Model Evaluation and Validation

Model evaluation and validation for this project revealed a number of interesting findings and challenges. As stated in *Section* 1.3 I intended to use the following model setup:

- LinearRegression (comparison model)
- AdaBoostRegressor (comparison model)
- SVR (comparison model)
- XGBRegressor (benchmark model)

With the *SVR()* regressor, I encountered the problem of very long training run times. I performed two rounds of training. Each round, I had to stop via interrupting the kernel after 23 hours, as the kernel seemed to run endlessly. Obviously, the processing power of my Macbook Pro with 4 kernels and 2.7 GHz did not perform well with the training data containing a total number of 1708337 rows and 328 columns. The SVR kernels that I used were of type *'rbf' and 'poly'*. Further investigation of this problem revealed, that the long run-times could be caused by the *SVR()* kernels to solve an optimization problem of quadratic order. The following source [10] describes this problem in more detail as follows:

"The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to datasets with more than a couple of 10000 samples. For large datasets consider using sklearn.linear_model.LinearSVR or sklearn.linear_model.SGDRegressor instead, ..."

Following this approach, I used *sklearn.linear_model.LinearSVR* instead of *sklearn.linear_model.SVR*. This solved the problem of very long runtimes and produced the following results as outlined in *Figure* 8.

	estimator	min_score	mean_score	max_score	std_score	C
1	LinearSVR	2.02765	1.84375	1.57286	-0.195588	1
0	LinearSVR	2.02764	1.84374	1.57286	-0.195586	0.5

Figure 8: GridSearchCV Results for LinearSVR

Figure 9 shows the results of the k-fold Cross-Validation for the machine learning algorithms *AdaBoostRegressor* and *XGBRegressor*. While the *AdaBoostRegressor* shows little improvement in the training scores, the *XGBRegressor* shows much better training scores.

	estimator	min_score	mean_score	max_score	std_score	early_stopping_rounds	learning_rate	max_depth	n_estimators
3	AdaBoostRegressor	2.25042	2.16228	2.07725	-0.070731	NaN	1	NaN	16
1	AdaBoostRegressor	2.24844	2.1704	2.06332	-0.0783145	NaN	0.5	NaN	32
2	AdaBoostRegressor	2.292	2.18176	2.03819	-0.106265	NaN	0.5	NaN	64
4	AdaBoostRegressor	2.25368	2.16816	2.03396	-0.0960762	NaN	1	NaN	32
5	AdaBoostRegressor	2.2577	2.18045	2.02975	-0.106575	NaN	1	NaN	64
0	AdaBoostRegressor	2.26164	2.09722	1.93535	-0.13322	NaN	0.5	NaN	16
11	XGBRegressor	1.64418	1.55271	1.38535	-0.118516	10	0.1	9	NaN
6	XGBRegressor	1.67962	1.5642	1.38077	-0.13114	10	0.05	3	NaN
8	XGBRegressor	1.64127	1.54725	1.37918	-0.119119	10	0.05	9	NaN
9	XGBRegressor	1.66536	1.5559	1.3768	-0.127684	10	0.1	3	NaN
10	XGBRegressor	1.64093	1.54389	1.3724	-0.121612	10	0.1	6	NaN
7	XGBRegressor	1.64587	1.5454	1.37144	-0.123499	10	0.05	6	NaN

Figure 9: GridSearchCV Results for AdaBoostRegressor and XGBRegressor

Table 2 compares the training and testing performance scores of the benchmark and comparison models. The results can be summarized as follows:

- *AdaBoostRegressor* is the only learning algorithm that performs better on the testing data set than on the training data set.
- *AdaBoostRegressor* and *LinearSVR*, both perform worse on the testing data set than the benchmark performance algorithm, i.e. *XGBRegressor*.
- *XGBRegressor* performs best on the testing data set and also performs better than the benchmark model.
- The k-fold CRoss-Validation on time series data improves the performance of the *XGBRegressor* in comparison to the benchmark model.

Table 2: Comparison - Best Score RMSE

Best Performance Score		
Learning algorithm	RSME - Train	RSME - Test
Benchmark XGBRegressor	1.78241	1.8008
LinearSVR	1.57286	1.9047
AdaBoostRegressor	1.93535	1.9083
XGBRegressor	1.37145	1.6063

Training and testing the *sklearn.linear_model.LinearRegression* learning algorithm, produced a RMSE in the range of one million. Running the other machine learning algorithms on the same training and testing data sets, resulted in an RMSE around, or below 2.0. For this reason, I considered the results from *sklearn.linear_model.LinearRegression* as flawed. A first investigation in this issue showed, that multi-variate linear regression can be a challenging task, in terms of preparing data sets. In order to keep in line with the available time-budget for this project, I decided to no longer pursue *sklearn.linear_model.LinearRegression* and instead focus on the other three machine learning algorithms.

6 Conclusion

6.1 Reflection

This capstone project showed that the *XGBRegressor* represents a machine learning algorithm, that is fast in processing and effective in terms of prediction performance. The findings of this capstone project are in line with the fact, that *XGBRegressor* is a very popular winning algorithm in Kaggle competitions.

This capstone project provided a number of challenges that also induced a steep learning curve in number of fields. The first learning curve was related to the complexity of the training and testing data sets with reference to their sizes and quality. Succeeding in the training of the models, also depended on the right selection of machine learning algorithm, that could deal with a training data set of approximate 1.7 millions rows.

Another challenge was the data analysis, for which I had to make a number of decisions on how to transform, or delete feature values, or entire features themselves.

Feature engineering itself was a more straight-forward task, as it depended a lot on the quality of the data analysis. One-hot encoding the training and testing data sets was definitely the most challenging part in the feature engineering section.

The definition of an effective training and validation methodology was another challenge for this project. One of the key-driving factors of complexity was the time series nature of the data sets. As I had mentioned on the sections above, the most approaches of randomly shuffling data for k-fold Cross-Validation was no longer an option. Finding an alternative and assuring that that this alternative keeps the time series in line with the project requirements proved complex.

The first rounds of training the machine learning algorithms represented the next steep learning curve and provided a number of surprises. The first results of the *LinearRegression* were completely flawed. It became clear that these flawed results would require a thorough and time-intensive exploration of the problem; easily expanding the time budget of this capstone project beyond reason. Another interesting finding was the speed of the *XGBRegressor* and the very long run-times of *SVR*.

XGBRegressor, *SVR* and *AdaBoostRegressor*, represent very powerful machine learning algorithms with a considerable number of parameters for fine-tuning. One of my conclusions for this capstone project is, that a thorough understanding of these algorithms requires a deep understanding of the scientific theories underpinning these algorithms.

6.2 Improvement

The following improvements could result form the structure and the findings of this project.

Choosing three machine learning algorithms and not four, as it was the case at the beginning of this project, would leave more time in exploring specific algorithm tuning parameters in depth.

Another area of improvement could be to thoroughly investigate the cause of the flawed results of the *LinearRegression*. A first step in this process could be to investigate linear correlations between the features in the training data set.

References

[1] Stan Lipovetsky, Pareto 80/20 law: derivation via random partitioning, In *International Journal of Mathematical Education in Science and Technology,* pages 271–277., 2009.

[2] Stefan Lessmann, Kristof Coussement, Koen W. De Bock, Johannes Haupt. Targeting customers for profit: An ensemble learning framework to support marketing decision making, *SSRN Electronic Journal*, 2018.

[3] Amy Wenxuan Ding, Shibo Li, Patrali Chatterjee, Learning User Real-Time Intent for Optimal Dynamic Web Page Transformation, In *Information Systems Research,* 2015.

[4] Jason Brownlee, A Gentle Introduction to XGBoost for Applied Machine Learning, Accessed on *https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/* 2016.

[5] Jason Brownlee, How To Estimate The Performance of Machine Learning Algorithms in Weka, Accessed on *https://machinelearningmastery.com/estimate-performance-machine-learning-algorithms-weka/* 2016.

[6] Jason Brownlee, Discover Feature Engineering, How to Engineer Features and How to Get Good at It, Accessed on *https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/* 2016.

[7] Dimitre Oliveira, LGBM - Google Analytics Customer Revenue Prediction Accessed on *https://www.kaggle.com/dimitreoliveira/lgbm-google-store-revenue-prediction* 2018.

[8] David Batista, Hyperparameter optimization across multiple models in scikitlearn Accessed on *http://www.davidsbatista.net/blog/2018/02/23/model_optimization* 2018.

[9] Wikipedia Imputation (statistics) Accessed on *https://en.wikipedia.org/wiki/Imputation_(statistics)* 2018.

[10] scikit learn Epsilon-Support Vector Regression Accessed on *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html* 2018.