ADVANCED PRACTICAL COURSE DATA SCIENCE

FINAL REPORT

Task 1 – Wine Quality Prediction

Data Science Question: Which quality score will a wine get?

Exploratory Data Analysis And Data Preprocessing

Train Data Set

The train data set contains 13 features, Table 1 gives an overview.

Feature	Min	Median	Max	Datatype
fixed acidity	3.80	7.00	15.90	Float
volatile acidity	0.08	0.29	1.58	Float
citric acid	0.00	0.31	1.66	Float
residual sugar	0.60	2.90	65.80	Float
chlorides	0.01	0.05	0.61	Float
free sulfur dioxide	1.00	29.00	289.00	Float
total sulfur dioxide	6.00	118.00	440.00	Float
density	0.99	0.99	1.04	Float
pH	2.72	3.20	4.01	Float
sulphates	0.22	0.51	2.00	Float
alcohol	8.00	10.30	14.90	Float
type	1.00	2.00	2.00	Int
quality	**3.00**	**6.00**	**9.00**	**Int**

Table 1

There are 5150 observations and no missing vales. 1267 observations are of red wines and 3883 observations are of white wines. While there are more observations for white wines in the train data, there is a sufficient amount of observations for red wines. E.g. we don't run into a problem where there are fewer observations than features for a specific wine type. This finding would allow the separation of the data set and model into one for red wines and one for white wines. But should this be done? A principle component analysis can give more insight. 77% of the sum of the variances of all standardized features in the train data set is explained by the first 5 principle components, therefore the 5 principle components in figure 1 represent the train data set to a good amount. It is visible that red wines (red dots) and white wines (blue dots) are different clusters in the train data set. Additionally, early models suggested that the prediction improves with separate data sets and models. The decision to split the data set is now supported and kept.

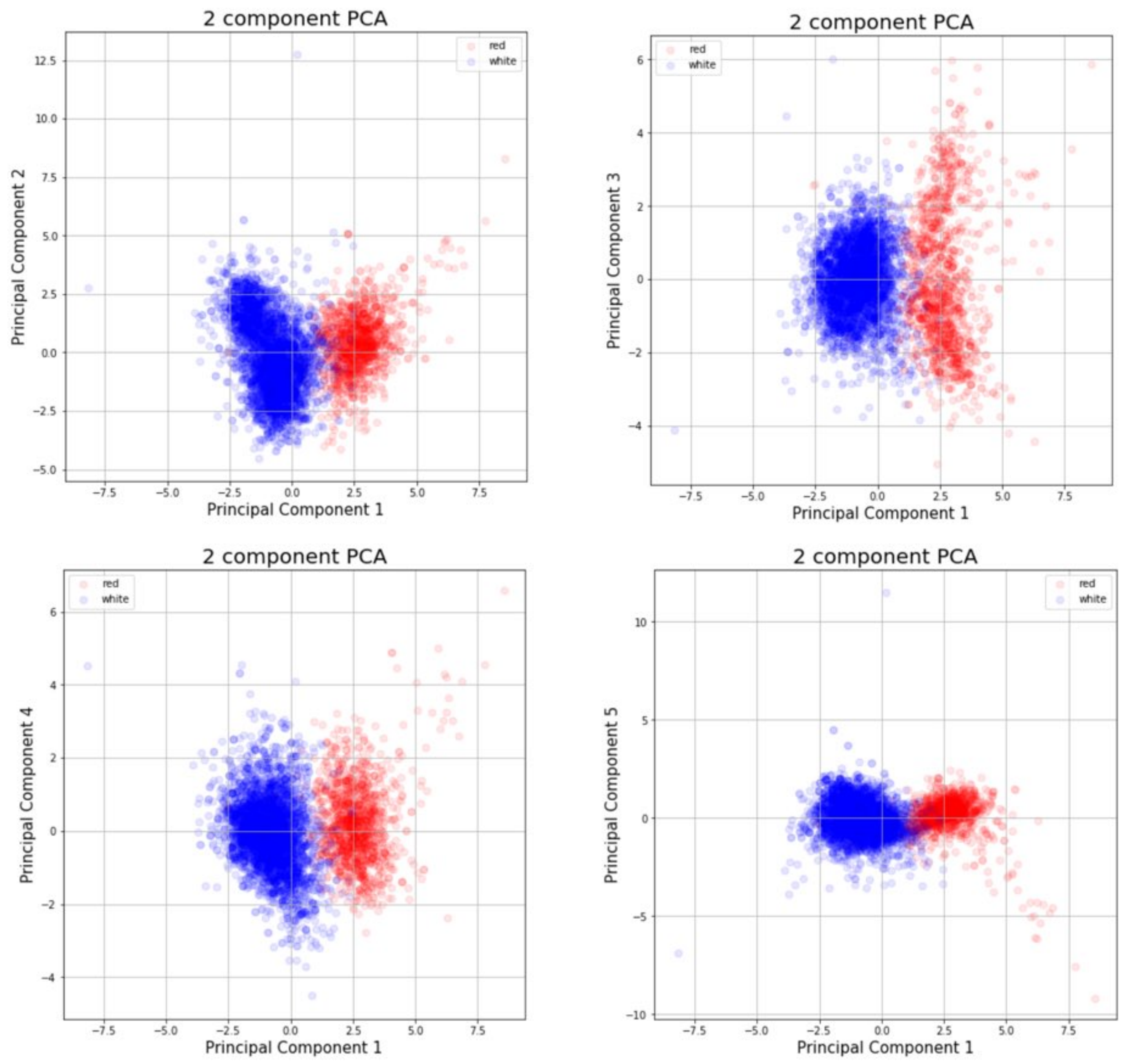


Figure 1

<u>Test Data Set</u>

There are 1347 observations and no missing values. The test data set has the same features as the train data set but excluding the target feature ‘quality’ and including the feature ‘id’, which is needed to identify each tuple on Kaggle. 332 observations are of red wines, 1015 observations are of white wines. There are more white wines than red wines whose qualities have to be predicted, but it is not the case that e.g. one wine type makes only 1% of the wines to be predicted. This shows that it’s worth to aim for wine type specific predictions.

The target feature quality can theoretically be an integer value between 1 and 10. Bar plots of the frequencies of the quality levels have shown that there is a roughly bell shaped distribution, with qualities of 1, 2 and 10 not occurring at all, 3 and 9 rarely and 5 and 6 very commonly. Analysis of the other features has shown that there are outliers, but removal of those did not improve the prediction. The features are also on different scales, but the models used do not require normalization (unlike the principle component analysis conducted above). There were no missing values. In conclusion, no feature engineering was considered to be required, the only action taken was to split the train and test data sets into two sets for red and white wine each.

Model Building

All features from the train data set that were also in the test data set were used as predictors. A Scikit Learn Gradient Boosting Regressor was used. As a model benchmark, a 3-fold CV score on the train data set was used (within limits, more folds would have led to a more consistent score, but they would have been more computationally expensive). The public Kaggle leaderboard score was mostly ignored for the following reasons: The train data set contains 5150 observations. The test data set contains 1347 observations, 30% of that, or 404 observations were used for the public leaderboard score on Kaggle. Therefore, a CV score based on the train data set is based on ten times more values and it was additionally computed using the cross-validation method. This means that there is a lower probability that such a score would be misleading and/or lead into overfitting. Using a grid search with GridSearchCV(), the following parameters were found to lead to the best prediction I could find:

Red wine model	White wine model

n_estimators = 2500
learning_rate = 0.033
loss = "huber"
max_depth = 7
random_state = 1
alpha = 0.95
subsample = 0.78
	n_estimators = 3500
	learning_rate = .033
	loss = "huber"
	max_depth = 8
	random_state = 1
	alpha = 0.95
	subsample = 0.78

Table 2

Also tried but with less promising results:
- Scitkit Learn Support Vector Machine
- Scikit Learn Random Forest.

Result

Kaggle public score: 0.34653
Kaggle private score: 0.32555
public rank on Kaggle: 7th
private rank on Kaggle: 2nd (shared with another participant)
This shows that relying on a train data set based CV score lead to less overfitting than relying on the Kaggle public leaderboard score.

Task 2 – NYC Taxi Fare Prediction

Data Science Question: How much did a customer pay in total for a taxi ride in New York City?

Exploratory Data Analysis And Feature Engineering

The Data

The original train data set has 11,135,470 observations of 19 features, it is computationally expensive (regarding CPU and memory) to work with all of it. Therefore, a random subset of 100,000 observations was drawn and used further on, including model fitting. This subset is from now on referred to as the train data set. The full 11 million observations were only used at the very end to slightly optimize the prediction.

The used train data set: 100,000 observations, no missing values, 19 features, target feature is 'total_amount'. Table 3 gives an overview of all features in the train data set.

Feature	Min	Median	Max	Type
VendorID	1	2	2	Int
tpep_pickup_datetime	-	-	-	Object
tpep_dropoff_datetime	-	-	-	Object
passenger_count	0	1	6	Int
trip_distance	0.00	1.71	46.24	Float
pickup_longitude	-75.71	-73.98	0.00	Float
pickup_latitude	0.00	40.75	41.00	Foat
RatecodeID	1	1	99	Int
store_and_fwd_flag	-	-	-	Object
dropoff_longitude	-75.71	-73.98	0.00	Float
dropoff_latitude	0.00	40.75	41.19	Float
payment_type	1	1	4	Int
fare_amount	-75.00	10.00	240.00	Float
extra	-4.50	0.00	4.50	Float
mta_tax	-0.50	0.50	0.50	Float
tip_amount	-11.46	1.35	202.00	Float
tolls_amount	0.00	0.00	55.63	Float
improvement_surcharge	-0.30	0.30	0.30	Float
total_amount	**-75.30**	**12.30**	**312.39**	**Float**

Table 3

The test data set has 64,000 observations and no missing values. It has the same features as the train data set, excluding 'tota_amount', 'fare_amount', 'extra', 'mta_tax', 'tip_amount' and including 'id', which is used by Kaggle to identify the rows.

Summary of added features to both the train and the test data set (details in the following paragraphs):

Feature name	Description
‘duration’	Duration of the ride in seconds
‘averagespeed’	Average speed of the ride in mph
‘pick00’ – ‘pick99’	Dummy variables for 100 tiles in which the map of New York City and surroundings were divided
‘drop00’ – ‘drop99’	Dummy variables for 100 tiles in which the map of New York City and surroundings were divided
‘h00’ – ‘h23’	Dummy variables for the 24 hours of the day
‘mon’ – ‘sun’	Dummy variables for the weekdays

Table 4

Time

All rides in the train data took place in June 2016. All rides in the test data took place in May 2016. In both cases, all days of the respective months were represented. The number of cab rides taking place each day had some but no strong fluctuation. E.g. there are no two days in the data sets where one day had twice as many rides as the other day. There seems to be a weekly seasonality in the train data, but it does not show as clearly in the test data. A time series analysis could have been done to analyze this further, but it was not clear how much these insights would help to find the total amounts payed per taxi rides (and not the number of taxi rides per day). Figure 2 depicts the number of observations per each day.

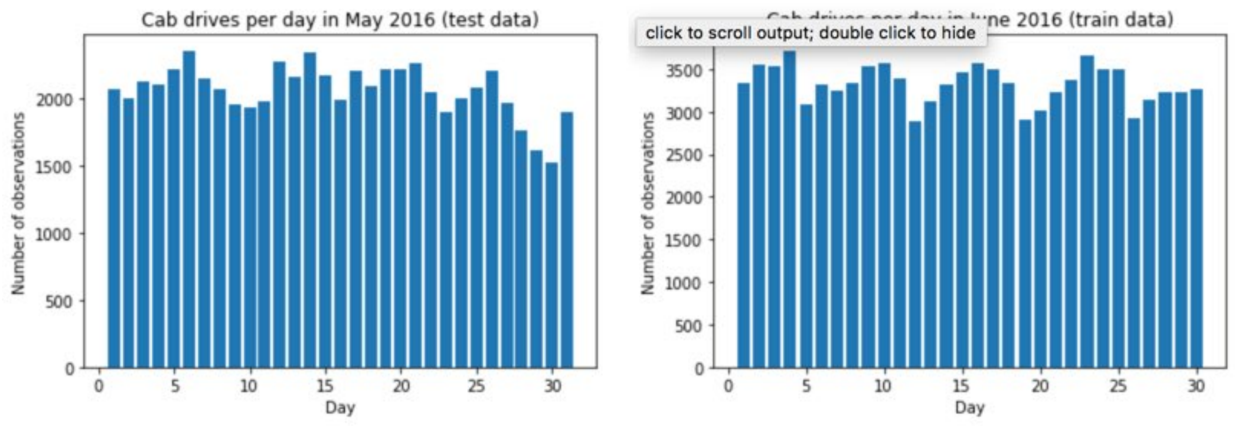


Figure 2

Speculation: maybe a pattern can be seen that people give less tips during the end of a month. A single integer column might be sufficient for such a relationship. 31 dummy variables would allow for modelling more complex relationships, but this could also lead to strong overfitting. Just because people gave a lot of tips on June 14 does not mean that they did the same exactly on May 14. This is why a single integer column for the month was added, as it only allows for modelling simpler, but also more general relationships.

7 dummy variables for the weekdays were added in the hope that customers give different tips on different days (e.g. difference between weekend and work days). 24 dummy variables for the hour of the day were added in the hope that tips may change with the hour (e.g. maybe there is different behavior between people driving to work at 7 am and people driving to a party at 1 am).

The following public holidays and other special days took place in May and June 2016 in NYC:

- May 8, Mother's Day (not a public holiday)
- May 30, Memorial Day (public holiday)
- June 19, Father's Day (not a public holiday).

Conclusion: There were very few of those days. We only have Father's Day in June to "learn" from it. But it would be a big assumption to say that people tip in the same way on Memorial Day as they did on Father's Day. There is a high risk of overfitting when trying to learn from public holidays or special days. The only thing relevant here is that the $1 surcharge from 4pm to 8pm in RatecodeID 1 does not occur on public holidays. Therefore, $1 was manually subtracted from the predictions for May 30 at the respective times.

There was the idea to add a Boolean 'hurry' feature, which is true when the minute of the drop off is within a given interval of a full hour (e.g. all minutes between xx:55 and xx:05). The speculation behind this feature was that people in a hurry might tip differently, and that people that arrive somewhere close to a full hour might be in a hurry. It turned out that this feature does not improve the prediction so it was not part of the data sets or models later on.

Some implausible findings related to time:
In the train data set, 84 cab rides had a duration of 0 seconds and 4 rides had negative durations. While such values are implausible, the good news is that there are not a lot of these cases. One goal of this analysis was to check if all kinds of cancellations, errors and/or missing values occur as 0-second-duration-rides in the data, which fortunately is not the case. Negative and zero value durations have been removed from the test and train data sets. In the train data, there are 149 rides that took longer than 2 hours (some even more than 5 hours, some almost 24 hours). The duration is remarkably long, but the distance is only 5 miles on average in these cases, 40 miles at max. The taxi drivers should have gotten $60 for a 2 hour wait alone, but only got $20 on average. Most have a RatecodeID of 1, meaning they used the standard rate (no negotiated fare). There are also 135 cases, where the ride took over an hour (note that only 1.2% of all rides take longer than an hour) but were the 'fare_amount' was below $30. The vast majority of these observations turned out to be implausible at a closer look, so they were removed. The longest ride in the test data set took 2 h 45 min. That seems realistic. My Colleague removed some more implausible rows.

Location

Coordinates of the pickup location and the drop off location were given in the data sets. There are 1239 observations where the pickup coordinates are 0 (both for latitude and longitude). There are 1121 observations where the drop off coordinates are 0 (both for latitude and longitude). Because they are only a few relative to the overall size of the dataset and because there is no way to reconstruct the coordinates, all rows in train data which contain 0 values for coordinates were removed. All coordinates in the test data set were non-zero.

The coordinates are just floats and the models we used can't learn much from them. Idea: put a 10x10 mesh over New York's map to divide it into 100 tiles. The result are 100 dummy variables for the pickup tile and 100 dummy variables for the drop off tile. Figure 3 depicts the drop off tiles. We did not use quadratic and evenly sized tiles. Instead, we used 10%-quantiles of the longitude and latitude coordinates to determine the mesh. This means that there are tiles with an area of many square kilometers in remote locations (not many cab rides started, ended or took place in remote areas far away from Manhattan, this means that the extreme quantiles of e.g. the longitude are much further apart compared to the more central quantiles). There were also tiles in Manhattan that are only a few hundred meters across. This does not mean that exactly 1% of the observations are from each of the 100 tiles, but that is roughly the idea behind it. Each tile does not cover exactly 1% of the observations because it is in most cases impossible to divide a given map like this with 9 vertical and 9 horizontal lines. Evenly sized quadratic tiles covering exactly 1% each of the arbitrary area that is spanned by the two spatially most extreme observations would have been a division by area instead of dividing by the number of observations. This was considered inferior, because there would have been only 1 tile for the whole inner city. Another approach to classify the spatial locations could have been to have dummy variables for all of New York's districts. However, it's more elaborate to assign the districts to the coordinate values properly and a fine mesh of rectangles has the potential to lead to a better prediction than the rather big districts.

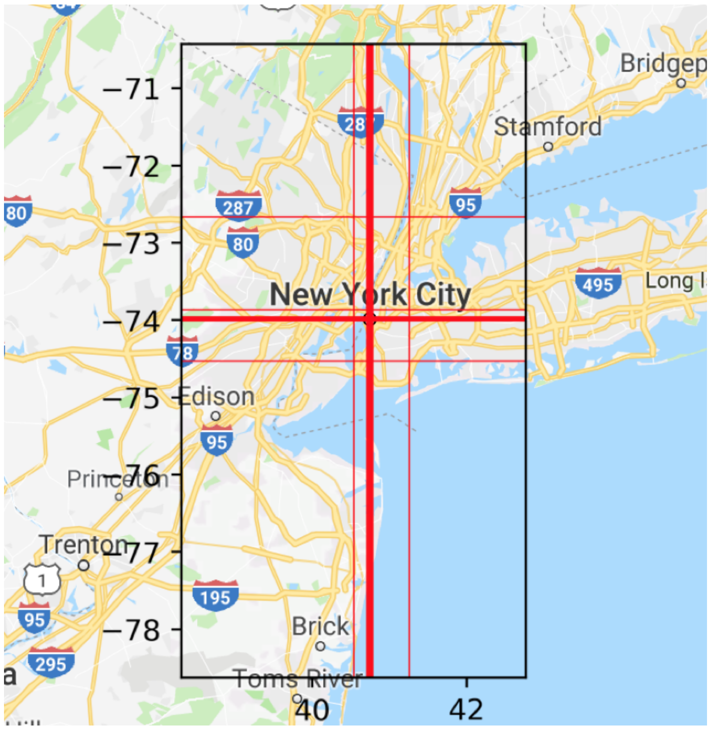


Figure 3

RatecodeID

There are different taxi rates which determine how the 'fare_amount' is calculated, 'RatecodeID' is the corresponding feature to these rates. In the train data, the RatecodeIDs 1 to 6, as well as 99 occur. RatecodeID 99 is not documented and only occurs 2 times in the used train data set (272 times in the original train data set with 11 million observations), zero times in the test data set. RatecodeID 6 occurs only 1 time in the train data set (only 117 times in the original train data set with 11 million observations) and zero times in the test data set. Rows with RatecodeID 6 and 99 were removed from the train data.

We decided to separate the data sets by RatecodeID. This was done by My Colleague, he also cleaned the data further. The result were the following data sets:

5 train data sets:
20,000 observations each of rides under RatecodeID 1, 2, 3, 5.
5,000 observations of rides under RatecodeID 4.

5 test data sets:
50,000 obervations with RatecodeID 1
7,000 obervations with RatecodeID 2
4,000 obervations with RatecodeID 3
1,000 obervations with RatecodeID 4
2,000 obervations with RatecodeID 5

Correlations with tip_amount

To get a better idea of what determines the tip that a customer gives, we calculated the correlations of all features to the 'tip_amount'-feature. Correlations with itself, with 'total_amount' and with 'fare_amount' are irrelevant, because they are unknown for us in the prediction. 'trip_distance' has a noticeable correlation (0.52). We now know that the taxi driver tends to get a higher tip when the trip distance was long. That was expected because long rides are usually more expensive and the tip is often given as a fraction of the amount to pay. A not so obvious thing to note: the correlation with tolls_amount (0.42). While this is nice to know, we did not further make use of the insights gained. The main insight is probably that most features don't have a noticeable correlation with 'tip_amount', meaning there are unfortunately no simple and strong linear relationships with the tip amount.

Model Building

The benchmark: we used the average MAE scores of a 3-fold cross validation. We found out that the scores were very close to each other when using 10 and 3 folds. We decided to use 3 folds further on for runtime reasons. Our CV-based MAE was consistent with the public Kaggle score, meaning that an improved CV-based MAE usually meant an increased Kaggle score. We therefore relied on our CV-based MAE and did not need to use up the Kaggle submission limits.

The target variable 'total_amount' is the sum of 'tip_amount' and 'fare_amount'. 'fare_amount' should be deterministic, as it is calculated by the taximeters according to publicly known algorithms (the different rates whose rules can be found online). Only 'tip_amount' should be non-deterministic. The first idea was to have a (possibly manually created, very accurate) model for 'fare_amount' and a (machine learning-) model for 'tip_amount' and simply add the predictions of them. It turned out that it was impossible to get the exact fare_amounts, as we don't have all data that the taximeters have. We only know the average speed but not the actual speeds in every second of the ride. The actual speed is used by the taximeter to switch between charging per minute and charging per 1/5th mile. The hope was that a machine learning model (we used a Scikit Learn Greadient Boosting Regressor) could still predict the 'fare_amount' very accurately. It turned out that having one model to predict 'total_amount' lead to better predictions than using the sum of the predictions of separate models for 'fare_amount' and 'tip_amount'. We therefore continued with only one model.

While the exploratory data analysis took place with a random sample of 100,000 observations drawn from the 11 million observations in the original train data set, My Colleague now created 5 train data sets for each of the RatecodeIDs present in the test data set. We did this to be able to fit separate models for each RatecodeID and to make sure that there is a sufficient amount of observations for each RatecodeID (the previously used random sample had less than 300 observations each for the RatecodeIDs 3-5).

We used a Scikit Learn Greadient Boosting Regressor and used all features from the train data set that were also in the test data set as predictors, except for the 'X_datetime'-features, because they were adequately replaced by other features during feature engineering. The 'store_and_fwd_flag'-feature was also not included. Our models couldn't read it but it was also hard to see how this feature is meaningful so we didn't bother to find a way to include it in the model. Early predictions have shown that using the same parameters as in task 1, just with 200 estimators (instead of 2500 or 3500) already lead to good results. As the prediction was more computationally expensive than in task 1, we did not do a lot of search for the optimal parameters. If time wouldn't be so scarce we could have done a grid search to improve the prediction, but we already scored high on Kaggle with just 200 estimators and a random sample of 100,000 observations. We later improved our score by using the 5 different train data sets and models according to the RatecodeIDs and using 3000 estimators.

Result

Kaggle public score: 1.92145
Kaggle private score: 1.90164
Kaggle public rank: 2nd
Kaggle private rank: 3rd
This indicates that there was not much overfitting.

Task 3 – Online Advertisement Click Prediction

Data Science Question: Will a user click on an online ad or not?

Exploratory Data Analysis And Data Preprocessing

The Data

There were 4 tables available that could be used for training: 'train', 'ad_feature', 'user_profile' and 'behavior_log'. The test data set made available was called 'test'. The data sets 'train'/'test', 'ad_feature' and 'user_profile' could be merged together relatively easily without creating too many NaNs and with the number of observations remaining the same as the 'train'/'test' data sets have (2,079,023/60,652). This was done and called 'bigtrain'/'bigtest'.

'bigtrain' And 'bigtest'

The features of 'bigtrain' are:

Feature	Min	Median	Max	Datatype
user	10	573474	1141723	Int
time_stamp	1494000001	1494313923	1494626395	Int
adgroup_id	13	560107	846803	Int
pid	-	-	-	Object
nonclk	0	1	1	Int
clk	**0**	**0**	**1**	**Int**
cms_segid	0	0	96	Int
cms_group_id	0	4	12	Int
final_gender_code	1	2	2	Int
age_level	0	3	6	Int
pvalue_level	1	2	3	Float
shopping_level	1	3	3	Int
occupation	0	0	1	Int
new_user_class_level	1	2	4	Float
cate_id	1	5858	12960	Int
campaign_id	3	205346	423435	Int
customer	1	105620	255874	Int
brand	2	224985	461321	Float
price	0.01	168	99999999	Float

Table 5

The features 'pvalue_level', 'new_user_class_level' and 'brand' are categorical and could also be represented by integers, they just happen to be read in as floats and it does not hurt to leave it that way. 'bigtest' has the same features, excluding the target feature 'clk' and the corresponding 'nonclk' feature and including an 'id'-feature which helps Kaggle to identify the rows. The only NaNs in 'bigtrain'/'bigtest' are in the 'pvalue_level'-feature (1,086,193/ 32,415 NaNs) and in the 'new_user_class_level'-feature (554,429/17,048 NaNs). Additionally, 'bigtest' has 15,559 NaNs for the feature 'brand', where 'bigtrain' has 642,770 NANs. Observations in 'bigtrain' are from May 5-11 and a part of May 12 2017, observations in 'bigtest' are from a part of May 12 and from May 13 2017.

The feature ‘nonclk’ is just the negation of ‘clk’, it was removed because of redundant information. The ‘datetime’ feature was added to ‘bigtrain’ and ‘behavior_log’ and is just the ‘time_stamp’ feature given as an datetime object instead of an epoch integer.’datetime’ was then used to generate further time related features, as the models we used can’t read datetime objects. 24 hour dummy variables ‘h00’, …, ‘h23’ were added. The thought behind this was that users might click differently, depending on the time of the day and the compartment of life they are in (e.g. work, free time).

Further data preparation (e.g. clustering) and feature engineering (e.g. feature selection or feature imputation) was closely related to model building and can be found in the section “Model Building And Feature Selection”.

‘behavior_log’

‘behavior_log’ has substantially more observations than ‘bigtrain’ (64,795,089 vs 2,079,023) and it was not clear how to merge it with ‘bigtrain’ or how to use the information in it at all. Approaches to include ‘behavior_log’ are discussed in the “Model Building” section. The features in ‘behavior_log’ are given in table 6.

Feature	Min	Median	Max	Datatype
user	10	579526	1141723	Int
time_stamp	-2101140178	1493815716	1727020881	Int
btag	-	-	-	Object
cate	1	6183	12976	Int
brand	2	204697	461527	Int

Table 6

The 4 possible values of the ‘btag’-feature (used in the section “Model Building”):

‘btag’ value	Meaning	Frequency
pv	Just browsing the website without further action related to the advertised product	61,699,576 (95.2%)
buy	Buying the advertised product	832,753 (1,3%)
cart	Putting the advertised product in the shopping cart	1,428,473 (2,2%)
fav	Saving the advertised product as a favorite	834,287 (1,3%)

Table 7

Aside from observations which make up less than one one-thousandth of all observations, the observations in ‘behavior_log’ are from the months April and May 2017. We removed rows with other months, because many have a date in the future or decades ago, so they were not considered plausible. For ‘behavior_log’, data preparation and model building are closely related and based on one idea, which is explained in the section “Model Building And Feature Selection”.

Model Building And Feature Selection

The Benchmark

The public Kaggle score was calculated with 50% of the data, meaning 30,326 observations. Relying on this Kaggle score means less risk of overfitting than in task 1, where the Kaggle score was based on only 404 observations. Furthermore, it seems that the day to be predicted (May 13, and partly May 12 2017) is very different from the days in the train data set (May 5-12 2017). When uploading an all-zero submission to Kaggle, the MAE is 21.559%. Assuming that the other 50% of the test data which were not used for this score are similar, there must be roughly 22% clicks (13,076 in total) and 78% no-clicks (47,576 in total) in the test data. On the other hand, the train data had a relative click frequency of only roughly 5%. For some reason the users were much more tended to click on an ad on May 13 (or maybe also the night of May 12). In order to not be reliant on the limit of Kaggle submissions, we tried to generate an MAE of our predictions using May 5-10 for training and May 11-12 for testing. It was not consistent with the public Kaggle score, possibly for the reason that the test and the train data are very different. A model that was an improvement on Kaggle got a worse 'bigtrain'-based MAE and vice versa, meaning it was misleading. It was furthermore time consuming, as every model fit took several minutes in this task. We abandoned our own MAE and solely relied on the public Kaggle score in this task.

Inclusion Of 'behavior_log'

The idea was that the btag actions 'buy', 'cart' and 'fav' all require a click, while just browsing the website in which the ad is shown (btag equal to 'pv') would mean no click. For more information about the possible 'btag'-values, please see table 6.

Step 1: Generate a 'clk' feature for 'behavior_log', which is 0 if the 'btag'-feature is equal to 'pv' and 1 in all other cases.

Step 2: Train a model using only 'behavior_log'. The newly created 'clk'-feature is the target, the features 'user', 'cate' and 'brand' are the predictors.

Step 3: Use the model that was trained with behavior_log to predict the 'clk'-feature of 'bigtest', using the 'user', 'cate' and 'brand' features of 'bigtest' as the input.

Step 4: Build a meta model that combines the predictions of a model that is trained with bigtrain and the model that is trained with 'behavior_log'.

To be able to do this, we needed to impute the NaNs in 'brand' in 'bigtest'. The first approach was to fill the NaNs with 0. The reason behind this was that one possible interpretation of the NaNs is that the products came from a no-name brand. The imputed 0 would then stand for the ID of the brand "no-name". No promising prediction came out of this. Another interpretation is that the advertised products are from certain brands but they were not entered into the system. My Colleague imputed the NaNs using a LightGBM Gradient Boosting Regressor which was trained with the rows in 'bigtest' and 'bigtrain' where 'brand' was not NaN, with 'brand' being the target feature.

Unfortunately, we could not get any promising predictions when using 'behavior_log'.
The documentation on the 'btag'-value 'pv' is a bit scarce and ambiguous. We later found out that another interpretation is that the whole 'behavior_log' table is about what happened after a click on an ad, meaning that 'pv' or 'browse' refers to a user browsing on the website that opens after an ad was clicked, not browsing the website in which the ad is shown. If this is the case, the approach can't work because the generated 'clk'-feature is wrong. Because we already got promising predictions using only 'bigtrain', we moved on (although My Colleague tried more things with 'behavior_log' and a lightGBM) and 'behavior_log' was not used in the final models that were selected on Kaggle.

Further Approaches To Improve The Prediction

One other approach to improve the prediction was to cluster the users. In theory, it would have been best to have a model for each user, but given the high number of users, this is way too expensive. We therefore clustered the users so we could fit a model per user cluster. My Colleague did a cluster analysis, an elbow plot suggested that it would be reasonable to divide the customers into 7 clusters. My Colleague then split 'bigtrain' and 'bigtest' into 7 subsets each according to the cluster in which the users were in. We then fitted 7 different models and predicted with them. We could not get a better prediction using this approach, so we abandoned clustering.

Another approach to improve the prediction was to create and use a train data set that has a 22% click frequency like the test data set must approximately have and not like the 5% that the given train data set has. Rows with no click were randomly removed from 'bigtrain' so that the click frequency was 22% in ‘bigtrain’ as well. ‘bigtrain’ has over 2,000,000 rows and roughly 1,500,000 of them were selected randomly. Because of these big numbers, we did not expect patterns to occur in the removal (e.g. that only rows with a certain brand would have been removed by chance). When looking at the number of observations for each day, it looks like the rows were evenly removed across the days, as expected. The prediction with this new train data set was worse than the trivial solution of only zeros. We abandoned the idea of creating a balanced train data set. If it had looked more promising, a further possible optimization could have been to use stratification according to some properties of ‘bigtest’ (e.g. also trying to get similar brand frequencies compared to ‘bigtest’).

Feature Selection

The original set of features used as predictors in an early but comparatively successful prediction (better Kaggle score than the trivial solution:

‘user’,
‘time_stamp’,
‘adgroup_id’,
‘pid’,
‘cms_segid’,
‘cms_group_id’,
‘final_gender_code’,
‘age_level’,
‘shopping_level’,
‘occupation’,
‘cate_id’,
‘campaign_id’,
‘customer’,
‘price’

Features have been added and removed to the original set of features and the public Kaggle score was generated to evaluate whether this feature selection was an improvement or deterioration, as depicted in figure 4.

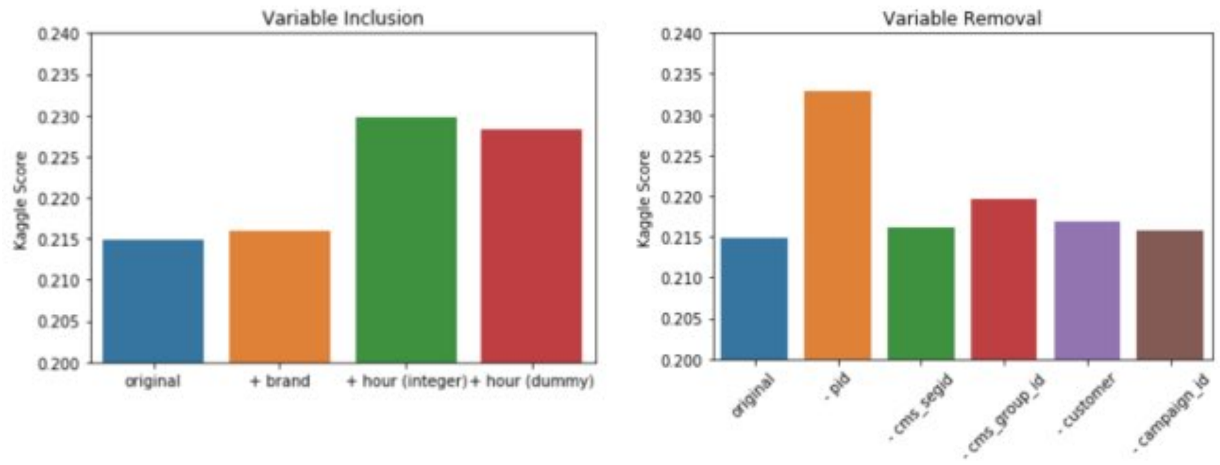
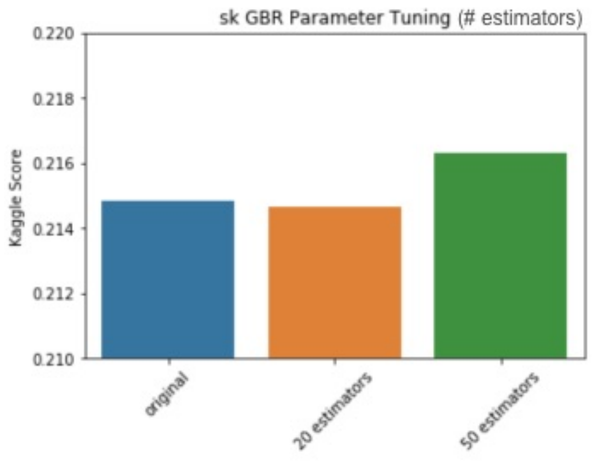


Figure 4

Figure 4 shows that including the ‘brand’-feature with NaNs imputed as zeroes lead to a worse Kaggle score, as well as including the hour of the day in either dummy or integer format. Removing ‘pid’, ‘cms_segid’, ‘cms_group_id’, ‘customer’ and ‘campaign_id’ from the set of predictors also lead to a worse Kaggle score. The insight here is that we already had a good feature selection in the beginning (at least for the local Kaggle score minimum that we found with our specific model and parameters) and that the ‘pid’-feature, the scenario and location of the ad on the screen, is important to predict the clicking behavior.

Parameter Tuning

Using the same Scikit Learn Gradient Boosting Regressors with similar parameters compared to task 2 and 3 already led to a prediction that was better than the trivial solution. Of all tasks, model fitting took the longest here, we therefore used a very low number of estimators on the order of dozens compared to thousands in task 1 and 2. Because of the long runtime, we did not set up a grid search, but we manually tried out many configurations and evaluated the Kaggle scores. Figure 5 depicts some things that we tried with the parameters ‘n_estimators’ and ‘learnrate’. We also tried more values of these parameters and also changing various other parameters but the usual finding was that it did not improve the predictions. Apparently, we already found something close to a local Kaggle score minimum with one of the first models we’ve build and we couldn’t find out how to improve the Kaggle score further. The only thing that slightly improved the score was to use 20 estimators instead of 10, which was done for our final Kaggle submissions.



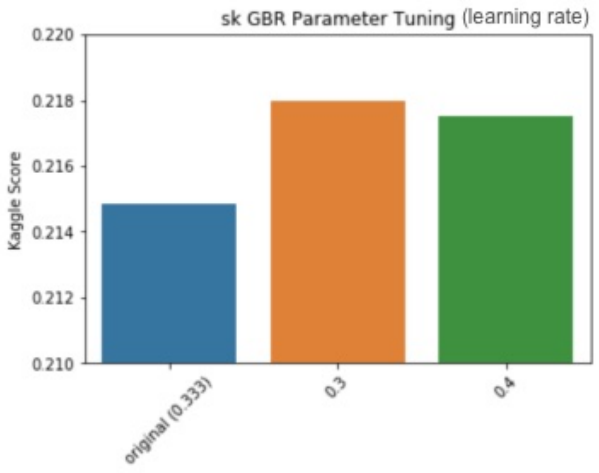


Figure 5

Result

Kaggle public score: 0.21466
Kaggle private score: 0.21470
Kaggle public rank: 3rd
Kaggle private rank: 3rd
This indicates that there was not much overfitting. We can also see that it was hard to predict substantially better than the trivial solution of only zeroes, which has a private Kaggle score of 0.21588.