Machine Learning Engineer Nanodegree - Capstone Project

An Xia
December 26, 2018

[Kaggle Competition] Elo Merchant Category Recommendation: Understand Customer Loyalty Using Transactions Data

Definition

Project Overview

Elo is a payment solutions company in Brazil, who provides various credit card options to consumers and partners with merchants to offer promotions to cardholders. Elo adds value to cardholders and merchants when cardholders get discounts they highly value and end up purchasing goods and services from the relevant merchants. Therefore, it's highly important to provide targeted promotions to the most relevant group of cardholders. One way to differentiate between high value cardholders vs. the rest is to use loyalty scores, and Elo has called for the data science community on Kaggle to help predict customer loyalty scores given historical credit card transactions data.

Predicting customer loyalty has been a critical topic in marketing for decades. Previous research in this field has successfully applied machine learning techniques such as regression models and neural networks to find the relationship between historical purchases, customer satisfaction, perceived value, and customer loyalty (Buckinx, Verstraeten, & Poel, 2007; Ansari & Riasi, 2016). This project will explore the relationship between historical transactions and customer loyalty using Elo's internal data.

Problem Statement

The objective of this project is to predict cardholder (identified via card_id) loyalty score given historical credit card transactions data. Because loyalty score is a continuous numeric variable, regression models are good candidates for the prediction. Card_id level characteristics and credit card transactions data can be used as inputs to predict loyalty scores. Before model fitting, the raw, timestamped transactions data needs to be inspected for outliers and transformed into card_id level summary statistics. Elo has provided training datasets that contain information on 202K cardholders, their loyalty score, and up to 5 months of transactions data. Elo has also provided a test dataset that contains transactions data for 124K cardholders without loyalty score, and the the goal is to predict loyalty scores for these cardholders in the test data.

Metrics

The prediction model aims to minimize the Root Mean Squared Error (RMSE) between the true loyalty score and predicted loyalty score, which takes the root of the average squared difference between the predicted loyalty score for each card_id and the actual loyalty score for the same card_id. The lower the score, the better the prediction. Competition participants will upload the predicted loyalty score for cardholders in the test dataset and Elo will return the RMSE between the submitted score and the true scores. Another candidate metric is Mean Absolute Error (MAE), which takes the average of the sum of absolute delta between the predicted value and the true value. Because we are concerned about large outliers in predictions, RMSE is a more appropriate option for the problem at hand since it gives higher weights to larger errors (Aydöre, JJ)

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{i}(y_{true\ i} - y_{pred\ i})^2}$$

Analysis

Data Exploration & Exploratory Visualization

Elo has provided the following datasets as inputs on Kaggle (https://www.kaggle.com/c/elo-merchant-category-recommendation/data):

- train.csv - a training set that contains card_id features and loyalty_score for each card_id
- test.csv - a test set that contains card_id features without loyalty_score
- historical_transactions.csv - up to 3 months' worth of historical transactions for each card_id
- new_merchant_transactions.csv - 2 months' worth of data not presented in historical_transactions
- merchants.csv - additional information about all merchants

Number of observations in each dataset. See sample data in Fig 1:

- Training data: 201,917
- Test data: 123,623
- Historical transactions: 29,112,361
- New transactions: 1,963,031
- Merchants: 334,696

Fig 1. Sample data from train.csv, historical_transactions.csv, new_merchant_transactions.csv, and merchants.csv

Train

	first_active_month	card_id	feature_1	feature_2	feature_3	target	elapsed_time	target_win
0	2017-06-01	C_ID_92a2005557	5	2	1	-0.820283	245	-0.820283

Historical Transactions

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_category_id	merchant_id	month_lag	purchase_amount	purc
0	Y	C_ID_4e6213e9bc	88	N	0	A	80	M_ID_e020e9b302	-8	-0.703331	2

New Merchant Transactions

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_category_id	merchant_id	month_lag	purchase_amount	purc
0	Y	C_ID_415bb3a509	107	N	1	B	307	M_ID_b0c793002c	1	-0.557574	2

Merchants

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	numerical_2	category_1	most_recent_sales_range	most_recent_purc
0	M_ID_838061e48c	8353	792	9	-0.057471	-0.057471	N	E	

1 rows × 22 columns

To work towards a model solution, the first step is to identify what historical transactions information and what cardholder features are most impactful on the loyalty score. A good starting hypothesis is that high spenders, frequent spenders, spenders that don't get transactions declined, buyers of certain goodies, cardholders located in certain geo locations, longer-term cardholders, etc. are more likely to have higher customer loyalty scores. Except for loyalty score (i.e. target) in the test data, other features are not readily available, but can be engineered by transforming or combining different features that are already in the datasets.

Loyalty Score (Target)

The main outcome variable is loyalty score, which is labeled as “target” in the dataset. It is unclear how Elo came up with the customer loyalty scores, but examining the dataset shows that customer loyalty score is a numeric value centered around 0 with outliers on the lower end (Fig. 2). The extreme outliers on the lower end suggests that winsorization should be done during data preprocessing.

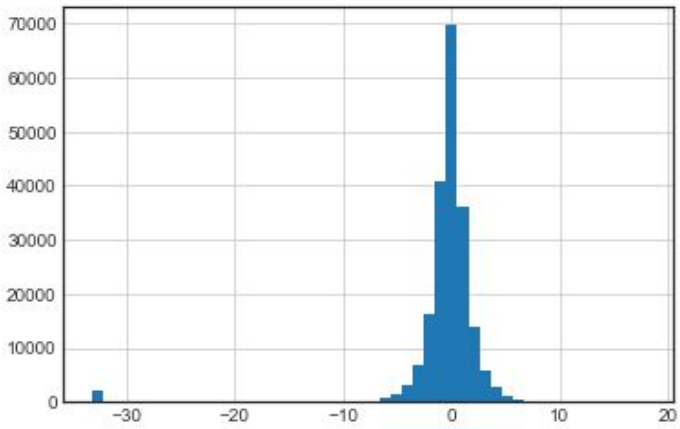

Fig 2. Distribution of customer loyalty score in training data

High Spenders, Frequent Spenders

High total purchase amount and high average purchase amount at card_id level can be used as proxies for high spenders. Similarly, high total number of purchases can high total number of purchases over a certain period can be used as proxies for frequent spenders. Checking the min, max and quantiles for purchase amount shows that extreme outliers exist at the upper end and suggests that winsorization is needed. Though counter-intuitive, majority of the purchase amount data is negative. This suggest that Elo may have made some data transformation before uploading the info on Kaggle to mask the true values and it's unnecessary to truncate negative values.

```
purchase_amount min and max:
min: -0.7469078
max: 6010603.9717525
purchase_amount percentile:
0.01   -0.743166
0.25   -0.719905
0.50   -0.687523
0.75   -0.600865
0.99    1.239597
```

Spenders that Don't Get Transactions Declined

Each transaction contains a "Yes/No" flag indicating whether the transaction is authorized. The higher proportion of authorized transactions, the more likely the customer is a better customer. More than 50% of the cardholders have 93% or more of their transactions authorized. Outliers exist at the lower end with 70% or even lower proportion of transactions authorized (Fig 3). This does not need to be further winsorized.

```
authorized_mean percentile
0.01    0.500000
0.25    0.866667
0.50    0.933333
0.75    0.975000
0.99    1.000000
```

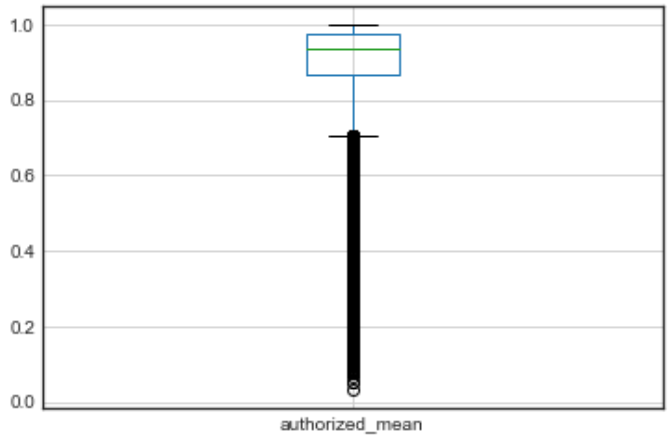

Fig 3. Distribution of cardholder level proportion of authorized transactions

Buyers of Certain Goodies
There are lots of categorical variables in the transactions dataset (e.g. category_1, category_2, category_3, merchant_category_id, subsector_id) that indicate the type of goods involved in the transactions. The true values of these categorical variables are replaced by generic category indicators (e.g. “A”, “B”, “C” etc) by Elo. As a result, it is unclear what these values actually represent, but could still be used in the models once transformed into dummy columns.

Cardholders Located in Certain Geo Locations
Similarly, variables such as state_id, city_id etc. also provide information on where the transactions happen. These variables can be used in the models as once transformed into dummy columns.

Long-term Cardholders (Tenure)
Both the train and test datasets contain card_id level’s “first_active_month*”.* Inferring from transaction level “purchase_date” and “month_lag”, the current month in the dataset is “2018-02”. Calculating the difference between “first_active_month” and “2018-02-01” shows how many days have elapsed since the card_id’s first_active_month (Fig 4). Majority of card_ids were activated within one year (365 days). Long-term cardholders have been active for 5 years or more (1800+ days). The distribution on time elapsed looks similar for the train and test dataset.

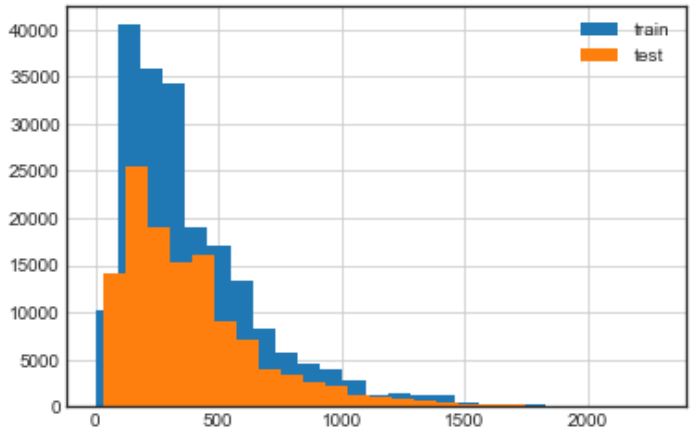

Fig 4. Distribution of cardholder level days since first_active_month (train and test dataset)

Algorithms and Techniques

Since the desired outcome of the model is to predict card_id level loyalty scores (continuous numeric values), linear and non-linear regression models are good candidates for this problem. The closer the prediction is to the actual loyalty score, the better the model. Linear Regression can be a good first step to try. Given the abundance in sample size and the likely non-linear relationship between the explanatory variables and the outcome variable, non-linear ensemble models such as AdaBoostRegressor and GradientBoostRegressor are good candidates as well:

Linear Regression (Yale University Course Note, Okello)

- **Strengths & when it performs well:** Linear regression attempts to model the relationship between the explanatory variable(s) and the dependent variable by fitting a linear equation to the data. The most common method for fitting is ordinary least squares, which calculates the best fitting line for the data by minimizing the sum of the square errors of each data point to the line. It computes fast and is a good technique when relationship can be represented by linear forms.
- **Weakness & when it performs poorly:** For complex relationship that is hard to approximate using linear forms, linear regressions don't perform as well. Moreover, other conditions need to be met such as the residuals need to be independent and the variance between the residuals needs to be constant.
- **Why good candidate for the problem/data:** The problem at hand is not too simplistic and also not too complex, so linear regression could be a good candidate. The fast computation speed is also a plus given the large amount of data involved.

AdaBoost (Freund & Schapire, 1996)

- **Strengths & when it performs well:** AdaBoost combines weak learners into a strong learner. It adaptively combines the weak learners by increasing the weights on data that were misclassified and aims to be good at learning those data in the next weak learner. It weights each weak learner based on accuracy. The higher the accuracy, the larger the weights.
- **Weakness & when it performs poorly**: Cannot perform well given with insufficient data, when the weak hypotheses are too weak, and seems to be susceptible to large numbers of outliers.
- **Why good candidate for the problem/data**: AdaBoost can be applied to regression problems. We have sufficient data and weak hypotheses seem to be good enough for our problem space (vs. more complex problem spaces).

Gradient Boosting (Gandhi, Zhang, Sharma)

- **Strengths & when it performs well:** Similar to AdaBoost, Gradient Boosting also combines weak learners into a strong learner. Take optimization problems as an example. The goal of optimization problems is to minimize loss of a predefined loss function. Gradient boosting builds learners one at a time, where each new learner helps further reduce errors made by the previously trained learners. It is a iterative process that continues until a certain threshold on the minimized loss is reached.
- **Weakness & when it performs poorly:** Because Gradient Boosting algorithms builds and iterates on learners sequentially, it can be computationally demanding.
- **Why good candidate for the problem/data:** GradientBoosting can be applied to regression problems. We have sufficient data and the size is still viable for computation.

Benchmark

A naive benchmark model can be built by first splitting the train dataset into a intermediary train set and a intermediary test set (different from the final test dataset provided by Elo), with both sets containing the true loyalty scores. Then, the average loyalty_score can be taken from the intermediary train set and applied to each observation in the intermediary test set. Because we know the true loyalty_score for the intermediary test set, we can calculate the RMSE between the predicted scores (i.e. intermediary train set average score) and the true scores and use it as a benchmark. The benchmark RMSE is 2.25.

```
y_pred = [y_train.mean()] * y_test.shape[0]
mean_squared_error(y_test, y_pred) ** 0.5
2.2487432783994192
```

Methodology

Data Preprocessing

Given the discussions in “Data Exploration & Exploratory Visualization”:

- Loyalty_score (i.e. target) in the train dataset is winsorized at the lower bound
- Categorical variables such as purchase categories, state_id, city_id, etc., are transformed into dummy columns for modeling
- Purchase_amount is also winsorized to deal with extreme outliers
- Historical transactions and new transactions datasets are combined to represent a unified transactions dataset
- The unified transactions dataset is then used to compute card_id level characteristics such as total purchase amount, average purchase amount, number of transactions, number and percent of authorized transactions, number of purchases by categories, by merchant_id, state_id, city_id, subsector_id, and number and proportion of purchases with installments

The final output data contains one row per card_id, which includes card_id features such as tenure, as well as summary statistics across all available transactions.

Implementation and Refinement

Once data preprocessing is complete, the implementation step contains two stages:

- The first stage is to evaluate various linear and non-linear models in terms of speed and its performance on the primary metric (i.e. RMSE)
- The second stage is to pick a model and further tune hyperparameters for better performance

During the first stage, the first step is to split the card_id level training dataset into a train/test set with 80/20 split for model fitting and testing. The second step is to develop a pipeline to evaluate training time, test time, training RMSE, and testing RMSE for all 3 models (Linear Regression, AdaBoostRegressor, GradientBoostRegressor) and compare the results.

Because the prediction speed is not exponentially slower, the performance is the best (i.e. lowest RMSE), and there is more room for parameter refinement compared to Linear Regressions, GradientBoostRegressor is picked for further tuning and improvements (Fig 5).

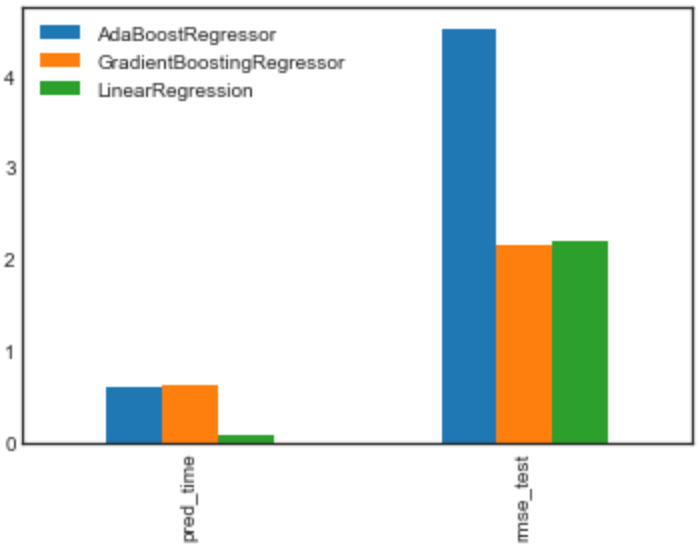

Fig 5. Model prediction time and test dataset RMSE for AdaBoost, GradientBoosting, and Linear regressors

For further tuning and refinement, GridSearch was attempted on the GradientBoostRegressor model with different levels selected for n_estimators, learning_rate, max_depth, min_samples_leaf (Table 1). However, given the size of the data and the available computing power, it could only perform GridSearch when a maximum of 4 scenarios are selected (e.g. two options for n_estimators, two options for learning_rate). This defeats the purpose of applying an extensive GridSearch over various scenarios. Moreover, the selected 4 scenarios did not show better performance compared to the default setting.

Table 1. Attempted GridSearch Hyperparameters for GradientBoostRegresser

Hyperparameter	**List of Values**
n_estimator	[100, 120, 150, 200]
learning_rate	[0.1, 0.5]
max_depth	[1, 3, 5]
min_samples_leaf	[1, 2, 3]

A less desired alternative in tuning than GridSearch was to pick a few scenarios with hyperparameters that are likely to be better than the default options and examine the results. This is more achievable given limited computing power. Two versions of hyperparameters were ran with variations on n_estimator and min_samples_leaf. Both versions outperformed the default model option by lowering RMSE on the test dataset.

Results

Model Evaluation and Validation

The final model selected is a GradientBoostRegressor with:

- learning_rate=0.1
- max_depth=3
- min_samples_leaf=1
- min_samples_split=2
- n_estimators=150
- And all other default options

The RMSE score is 2.1381 for the final model, which is an improvement compared to the unoptimized model and the benchmark (Table 2).

Table 2. Model performance on RMSE: Benchmark, Unoptimized, Optimized Model

Metric	Benchmark	Unoptimized Model	Optimized Model
RMSE	2.2487	2.1436	2.1381

The 3-fold cross validation on the optimized model yields an average RMSE of 2.1365, which corroborates that the model is robust.

```
RMSE for 3-fold cv: [2.16762  2.11905169  2.12290882]
Average RMSE for 3-fold cv: 2.136526837978261
```

Justification

Comparing the RMSE, it shows that the final GradientBoostRegressor model is an improvement over the benchmark mode and the unoptimized version. Moreover, the benchmark model is unrealistic in real life, because it assigns the same loyalty score to all cardholders, and thus defeats the purpose of segmenting cardholders. The final model enables Elo to segment cardholders by loyalty score and apply targeted promotions and marketing campaigns to the most relevant groups.

Conclusion

Free-Form Visualization

Elo has also provided a final test dataset for final model performance, where the loyalty score is unknown to Kaggle competition participant. Though the true loyalty score is unknown, we can still use the GradientBoostRegressor model obtained above to predict loyalty scores for this final test dataset and compare its distribution with the known loyalty scores among all train dataset cardholders (Fig 6). The distributions look very similar in shape, especially in the interval centered around zero, but the train dataset contains more observations around the more negative and more positive intervals. This indicates that the model could be further improved to perform more differentiated predictions on loyalty score versus having all predictions clustered around a small interval.

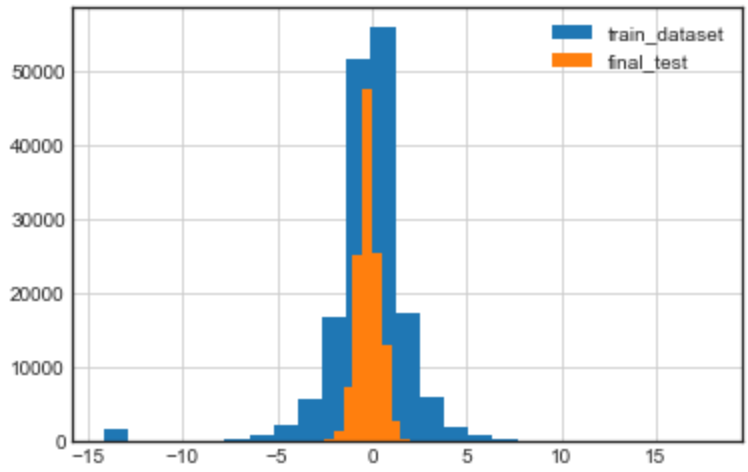

Fig 6. Distribution of predicted loyalty scores for cardholders in the final test dataset and for cardholders in the train dataset

Reflection

To work towards a model to predict cardholder loyalty scores, we first came up with a hypothesis that high spenders, frequent spenders, spenders that don't get transactions declined, buyers of certain goodies, cardholders located in certain geo locations, longer-term cardholders, etc. are more likely to have higher customer loyalty scores.

With that in mind, we then looked into the datasets to identify which variables can be engineered to become proxies for such features, explored the distributions of the relevant features, and performed feature engineering on those with extreme outliers. In the last stage of feature engineering, we collapsed the unified transactions data and joined it with card_id level feature data to arrive at the final training dataset that contains one row per card_id, with intrinsic card_id level characteristics and summary statistics derived from all available transactions.

After the feature selection and engineering steps, we split the training dataset into an 80% training set and a 20% test set for model selection and tuning. Once training and test datasets are ready, the next step was to select appropriate models to fit the data and run the prediction. Regression models are good options, since the desired output is a continuous numeric variable. Three relevant linear and non-linear regression models were considered: Linear Regression, AdaBoost, and Gradient Boosting. Comparing the prediction speed, the RMSE on test dataset, and room for further tuning, the GradientBoostRegressor was selected for further tuning. The final tuned model shows improvement over the untuned version and the naive benchmark model and is meaningful in the real world for Elo to differentiate between cardholders and come up with targeted treatments.

One challenge during model tuning is that computing speed is a real constraint on applying GridSearch. Initially, the plan was to apply GridSearch on all relevant hyperparameters to search for the best combination of hyperparameters for the dataset at hand. However, due to the constraint from CPU, we could only test on a few scenarios that may improve the model versus comprehensively search through all the combinations of hyperparameters. This highlights how important efficient modeling practices are and how important computing power is in applied machine learning.

Improvement

A few improvements can be made to make better predictions on loyalty scores:

- **Additional feature engineering**: How data can best represent a cardholder's behavior is the most crucial part of this project. We have tried a quite comprehensive list of feature engineering, but more could be done. For example, the average time it takes between transactions, how often the geo-location changes for the cardholder, and other underlying features that could be built using PCA are good additional features to add to the model. Additionally, some classifications could be done one cardholders based on their characteristics and different models could then be applied to each group of cardholders.
- **Transforming loyalty score (i.e. target) to a larger interval**: Loyal score in its current form is a small numeric value clustered around zero. Scaling the loyalty score to a larger range may enable models to better capture its variations so it's no longer clustered around a small interval as seen in the Free-From visualization.
- **Explore other model forms**: We have used linear and non-linear supervised learning regression models to approach the problem, but haven't look into more advanced techniques such as Neural network. It is unclear whether it will for sure improve the model performance or not, but it will be interesting to test out.

References

Buckinx, W., Verstraeten, G., & Poel, D. V. (2007). Predicting customer loyalty using the internal transactional database. Expert Systems with Applications,32(1), 125-134. doi:10.1016/j.eswa.2005.11.004

Ansari, A., & Riasi, A. (2016). Modelling and evaluating customer loyalty using neural networks: Evidence from startup insurance companies. Future Business Journal, 2(1), 15-30. doi:10.1016/j.fbj.2016.04.001

Sergül Aydöre, What is the difference between squared error and absolute error, https://www.quora.com/What-is-the-difference-between-squared-error-and-absolute-error, Accessed 2018.12.26

JJ, MAE and RMSE - Which Metric is Better, https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d, Accessed 2018.12.26

Yale University, Linear Regression course note, http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm, Accessed 2018.12.16

Erick Okello, Simple Linear Regression-Pros and Cons, http://erickokellostatcentre.blogspot.com/2016/06/simple-linear-regression-pros-and-cons.html, Accessed 2018.12.26

Freund, Y., & Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. Machine Learning: Proceedings of the Thirteenth International Conference. https://people.cs.pitt.edu/~milos/courses/cs2750/Readings/boosting.pdf, Accessed 2018.12.26

Rohith Gandhi, Gradient Boosting and XGBoost, https://hackernoon.com/gradient-boosting-and-xgboost-90862daa6c77, Accessed 2018.12.26

Zhaojun Zhang, What is the difference between gradient boosting and adaboost, https://www.quora.com/What-is-the-difference-between-gradient-boosting-and-adaboost, Accessed 2018.12.26

Bhuvan Sharma, What are the advantages/disadvantages of using gradient boosting over random forests, https://www.quora.com/What-are-the-advantages-disadvantages-of-using-Gradient-Boosting-over-Random-Forests, Accessed 2018.12.26