Exploring Feature Engineering Process with Combinations of Three Approaches

Member 1: Yixin Xie Member 2: Zehao Li Member 3: Renyu Yang

Member 1 Contribution: Data preprocessing, Embedded methods

Member 2 Contribution: Data preprocessing, Filter methods

Member 3 Contribution: Data preprocessing, Wrapper methods

1. Introduction:

In this project, the general objective is to incorporate different feature engineering techniques on the dataset to examine their prediction accuracy. Based on our preliminary research,we focus on three approaches, which are filter methods, wrapper methods, and embedded methods. By combining these methods together, we hope to learn about how different approaches interact with each other, and we expect to achieve maximum improvements in prediction accuracy through merging methods. This study will serve as a reference for real-estate appraisers to determine the value of a house. To be more specific, the real-estate appraisers can save time collecting only necessary information that helps with predicting the house price. For house owners who wish to sell their houses, this study will give them a rough idea how to determine the price for their houses. Most importantly, this report would also provide suggestions toward beginners in data science, for which they would learn whether incorporating feature engineering methods together is applicable in statistical prediction problems.

Our final results show that the filter methods could potentially reduce performance of wrapper or embedded methods, and the embedded method outperforms the other alternatives in general. As a result, embedded methods should be the best option in prediction problems with large datasets, and we should avoid removing large amounts of columns using other methods before applying it.

2. Methodology

Algorithms: First, We start with a data-preprocessing process by dealing with “NA” values and preparing categorical data for prediction. Second, We build k-best selection, constant or quasi-constant removal; these are filter methods that reduce the number of variables. Third, we construct wrappers methods, these include forward selection, backward selection, and stepwise selection. Then we construct embedded methods: Lasso regression, Ridge regression, Random Forest , XGBoost and their tuned models. At the final stage, we grouped all algorithms into 2 levels, for which in the first level we could either apply all filter methods or ignore filter methods for comparison; in the second level, we use wrapper methods and embedded methods to predict the house price using the data that is passed from the first level. In total, there are 18 models to be tested resulting from different possible combinations.

Dataset: The house price prediction dataset contains 34 numerical and 47 categorical variables. Under the numerical category, we have integer variables, such as year of construction, number of restrooms, number of pools, etc. We also have float variables, which are length and area. If we observe an “NA” value in numerical features, this means the value is missing. However, if we observe an “NA” value in categorical features, it could mean either missing data or the house does not have certain features. For example, an “NA” value in a feature fence means the house doesn’t have a fence. These “NA”s contain useful information regarding the data, thus they should be treated carefully

3. Implementation Details

Pre-processing: Our first task is to handle missing values. Using the data description, we create lists of variables that use NA’s to represent a level in categories, then we replace these NA’s with “Don't Have” in order to select the rows with true missing values. Next, we have two options: dropping observations with missing values or replacing the missing values. Because dropping NA’s would reduce the number of observations from 2919 to 2259, we believe that it would have a significant influence on the prediction result through reducing the information. Thus we eventually select the replacement approach, for which we attempt to compute the missing values while not having an effect on the data distribution. Through a boxplot of all normalized numerical variables, we find that some variables are extremely skewed. Therefore we replace the missing values using column median, which are less-susceptible to skewness and outliers, meanwhile it would not change the integer values to float. For the categorical variables, we replace the missing values with mode. For processing categorical data in prediction problems, a common approach is to convert categorical variables into dummy variables with values 1 and 0, where 1 represents presence and 0 represents absence.

Filter methods

The filter method starts by looking at the variance of every numerical feature. Features with low variance indicate that the value in those features are roughly the same or exactly the same. When we hope to predict house price, features with extremely low variance will not bring us much help compared to other features with higher variances. In addition, we also observe that some of the values for a certain feature are significantly larger than the value for other features. Features with large values will more likely to result in higher values of variance compared to features with small values. As a result, features with large values are less likely to be discarded. In order to treat features with low values and high values equally, we normalize the values for all numerical features. The user-determined threshold allows us to drop features with variance under that threshold. Here, we need to determine the value of the variance threshold. We generate the bar plot of the variance for all numerical features. From the bar plot attached in the supplement material page, we notice that Lot Area has the highest variance. For the rest of the features, they seem to be divided into two groups. A huge difference in variance exists between "YearBuilt" and "GrLivArea". For features with variance smaller than the variance of "BsmtUnfSF", they don't have much difference in terms of variance level. Thus, we decided that features with variance greater than 0.002 can be considered as significant.

Besides feature selection with variance, we also implement k-best feature selection. With user-determined k, we are able to select the k highest F-statistic scores. F-statistic scores is the hypothesis test result with null hypothesis stating that all of the regression coefficients are equal to zero. With high F-statistic scores, we are more likely to reject the null hypothesis and state that at least one regression coefficient is significant. In terms of k-best feature selection, F statistic scores are calculated based on correlation of every input feature and the target variable (house price in our dataset). After we determine the number of k, we only keep the k numerical features with the first kth highest F statistic scores and remove the rest of the numerical features. Again, we generate the bar plot of F statistic scores for all features in the supplementary material page. "GrLivArea" has the largest F statistic score. A huge difference in F statistic scores is observed between "GarageYrBlt" and "BsmtFinSF1". For features with F statistic scores smaller than "BsmtFinSF1", they don't have much difference in terms of F statistic score. We decided that F statistic scores greater than 200 can be considered as significant. Therefore, we believe that setting k=12 is reasonable.

Wrapper methods

We applied wrapper methods in our feature selection part. We compared the performance of three methods: forward selection, backward elimination, and stepwise selection. Depending on different inferences of wrapper methods, we will compare performance of each feature to decide whether to add or remove the feature and then train a model using the subset features with optimal performance. For the forward selection method, we start with an empty model and select one feature at a time that increases the R-squared of the model the most until we reach a certain number of features. For the backward elimination, we start with the model with all features and eliminate the feature that reduces the R-squared the most. For the stepwise method, we perform backward elimination during the process of forward selection, we test for the R-squared of each feature in the newly added model and eliminate the one that increases the R-squared in the remaining model. The reason we used R-squared for our feature selection scoring criteria is that we have passed the test of the regression diagnostic test. Then, we can use R-squared to determine how well the features fit the model. And we use linear regression for our feature selection model.

Before operating the feature selection function, we convert our categorical variables to dummy variables containing only zero and 1 to obtain most of the information in our dataset. Based on the method combination of our methods, we obtain 325 variables for the complete dataset and 300 variables after k-best selection and constant removal. We also need to find out what is the optimal number of features. Hit and trial is a common method to find the optimal number of features. The method tests the performance of different numbers of features based on the negative mean squared errors. Note that mean squared error is a measure that is less biased than the R-squared, the less mean squared error, the better performance the model is. From the graph in the additional material, we observe that 40 is a reasonable choice of feature numbers since the performance of models with features more than 40 is not significant. Finally, since our target variable is the SalePrice, we would fit a regression model with the optimal features to predict SalePrice.

Embedded methods

These methods contain built-in feature selection in the algorithms, and they are more widely used in large data regression problems compared to the previous two approaches.

Lasso and Ridge regularize complex models through adding penalty terms. Lasso has a L1 norm penalty and thus it leads to significant parameter shrinkage. Ridge on the other hand, has a L2 norm penalty and thus it can not reduce features to zeros. For both models, the penalty terms are not fair with variables of different magnitudes. Therefore we first standardize the numerical columns before fitting Lasso or Ridge.We also standardize the test set and the training set separately to avoid data leakage. Next, we are fitting a linear regression to the data with Lasso or Ridge regularization. Notice that we could transform the x columns and add polynomial features if we want polynomial regressions, yet we only apply linear regression here due to time constraints. The next important step is to find the penalization term lambda: while lambda equals to zero, we only have the linear regression model. We find lambda with the highest k-fold cross-validation score from the lambda grid. Using this lambda, we fit our models to the training set then use it to predict the testing Y. If there are only a small number of variables that actually influence the response, we will expect the Lasso to have a better performance than Ridge, and vice versa.

Random Forest is a popular supervised learning algorithm that builds an ensemble of decision trees. The trees run in parallel and their results would be aggregated into final predictions, this usually leads to high accuracy, however it also contains risk of overfitting. In addition, the method requires hyperparameters that are crucial to learning, in this project we include both default and tuned random forest. Since our goal is to compute 18 combinations of models, tuning each parameter in a large parameter space is time-consuming and it is not likely to be achieved in an iterative process. As a result, we use a random search approach with cross validation instead of gridsearch. The reason is that the random search does not go through every combination of hyperparameters but rather jumping between random combinations. Due to runtime concerns, we only pick 5 hyperparameters to be tuned, this results in a low-dimensional parameter space and makes the random search more efficient. The key parameters include the number of trees before taking average over predictions, the maximum features being considered when splitting a node, and the minimum number of leaves when splitting an internal node. After tuning hyperparameters, we also do feature selection based on Gini importance. The Gini importance measures how on average each feature reduces the impurity of the split. We picked this method due to its advantage in computation speed compared to the Mean Decrease Accuracy. Eventually, the random forest model with tuned parameters is fitted on the selected features, and we use this model to predict the Y test.

XGBoosting is a supervised learning algorithm using parallel tree boosting. It is selected because it often outperforms the other methods in prediction. In this project, we use both default XGB and XGB with tuned parameters. The procedure of XGBoosting is very similar to the random forest model. We use random search with cross validations to tune the five most important hyperparameters. These include the step shrinkage used to prevent overfitting, the max_depth of the tree(which could determine the complexity), minimum sum of instance weight needed in a child(it determines whether the algorithm is conservative), and the subsample ratio when constructing each tree, etc. The parameter space for each variable is constructed by adding finite numbers into a list, for which each number is a result from incrementing a certain amount to a starting value. For example, the learning rate is a number between 0 and 1, thus we could try the following parameter space [0.1, 0.2 … 0.6]. There are no correct answers for the selection of numbers. While random search would randomly evaluate combinations of parameters from their parameter space. adding more numbers into the parameter space could create more combinations for selection, yet it also significantly increases the runtime. As a result, we only input several numbers.. After the tuning process is finished, it is fitted to the train data and is used to predict the Y test.

4. Results

By going over the performance score of the 18 combinations of models. We have the following discoveries. First, the performance of procedures with pre-processing filter methods is dominated by the procedures without it. Second, in the group without filter methods, we have the following ranking of prediction performance from highest to lowest: XGB, Lasso, Ridge, default XGB, default Random Forest, Pairwise Selection, Forward Selection, Backward Selection, and Random Forest with tuned parameters. Notice that the results are generated from performance score of Kaggle competition after submitting the prediction result; despite a low score means better performance based on the description, we don't exactly know the computation process of this score, thus this ranking only provides ordinal information. The detail of the performance score of each combination is in the supplementary page.

Interpretation of Results

Regarding Filter Methods: In this case, it is possible that we remove too much information in the first level. Yet, it does not necessarily indicate a mistake in the filter method procedure. Recall that we have 32 numerical variables in total. We decided the k best and constant removal threshold strictly based on graphs, for which we only keep variables with high variance and high F statistics. This generates highly overlapping results, for which the prior reduces numerical variables to 12, and the latter reduces it to 7. This shows that the other variables are likely to be considered as noise using the filter method standard. Nevertheless, the wrapper method and embedded may still be able to extract information out of those variables. This shows that we may need to adjust standards while incorporating filter methods with the other two approaches. As a result, we may need to replace the strict threshold by a mild threshold which only excludes the features with the extreme low variance and extreme low F-statistic score. From the bar plot of feature variances, features with variances less than variance of "Lot Frontage" are very close to zero. Therefore, we won't lose much information by dropping these features. In this case, we will have to set our variance threshold to 0.00000013. However, it merely suggests a direction for which we can improve, and we need further testing to study whether filter methods with a reduced standard provide an improvement for wrapper and embedded methods.

Regarding XGB and Random Forest: The XGB and the Random Forest method aggregate results from multiple trees, thus they are less susceptible to noises in the data. As a result, deleting excessive numbers of variables in filter methods could reduce the information for the two methods. On the other hand, the tuned Random Forest method performs poorly compared to the default in both groups; this might suggest that the model overfitted the data and it is not likely due to filter methods. To fix the overfitting issues, we could increase the number of trees in parameter space while tuning the parameters.

Regarding Wrapper Methods, LASSO, and Ridge: The process of filter methods eliminates plenty of useful information for the wrapper methods. Thus, we would observe a decrease of performance of the result when we use filter-processed dataset for wrapper methods compared with the result using complete dataset. Although in our project, all of the three wrapper methods, LASSO and ridge, use linear regression to fit our model, LASSO has better performance and is more efficient than the rest. Since the wrapper methods are greedy search algorithms that consider all the possible subset of features to improve performance, they are computationally expensive in high dimensional dataset. Moreover, we sacrifice accuracy for computation time when we choose the optimal number of features in the wrapper methods. Thus, we would observe that LASSO performs better than the wrapper methods. We observe that LASSO has a better performance than ridge since we have a small number of variables that are influential to the response. This observation makes sense since we expanded all the categorical variables to dummy variables, which made many of the variables insignificant to our response variable.

Supplements

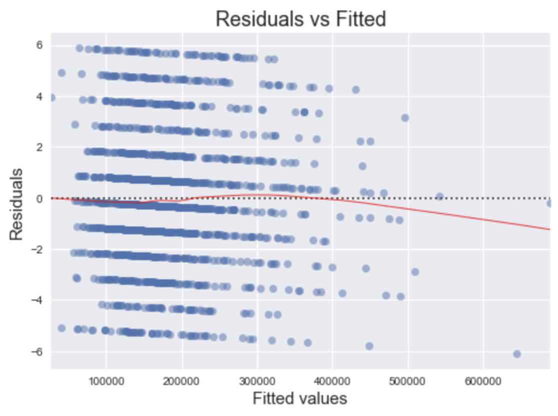
Explanation Regarding The Procedure:

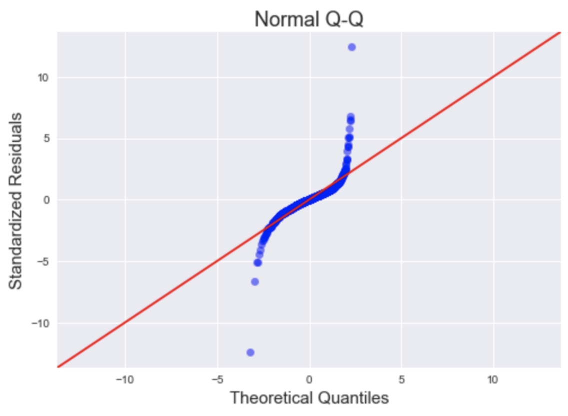
Our original plan is to test all combinations of methods in filter, wrapper and embedded methods. However, we find it to be not applicable due to 2 reasons. First, 2 filter methods, 3 wrapper methods and 6 embedded methods create at least 84 possible combinations (if we consider not applying any methods in an approach as an option), it is not possible to be implemented due to the time constraints. Secondly, the wrapper methods, such as the backward selection, are very time consuming algorithms; meanwhile, the tuning of parameters for the embedded methods is also a time consuming process. If we apply them sequentially, it eventually turns into a waiting game. Thus eventually, we decide to put embedded methods and wrapper methods on the same level, and see how filter methods influence their performance.

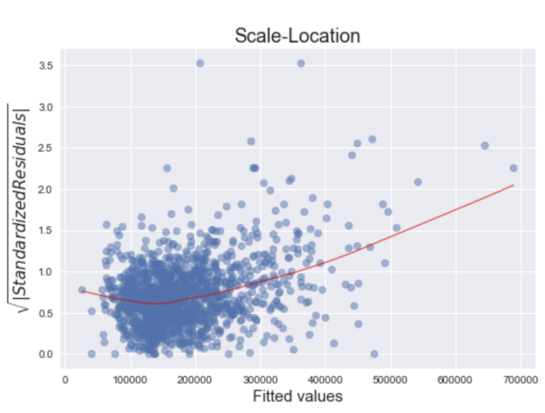
Scoring Board:

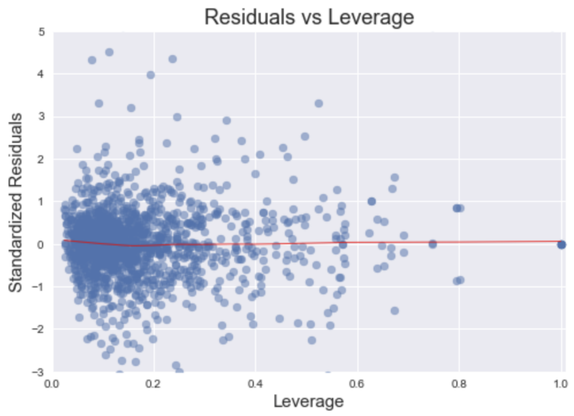
group	score	type	filter method
1	0.33049	forward	yes
2	0.21133	backward	yes
3	0.33049	pairwise	yes
4	0.15085	lasso	yes
5	0.15373	ridge	yes
6	0.15958	default random forest	yes
7	0.15103	default xgb	yes
8	0.2229	tuned random forest	yes
9	0.13821	tuned xgb	yes
10	0.16515	forward	no
11	0.17039	backward	no
12	0.16491	pairwise	no
13	0.14507	lasso	no
14	0.15096	ridge	no
15	0.15808	default random forest	no
16	0.15483	default xgb	no
17	0.21896	tuned random forest	no
18	0.13405	tuned xgb	no

<u>Graphs</u>

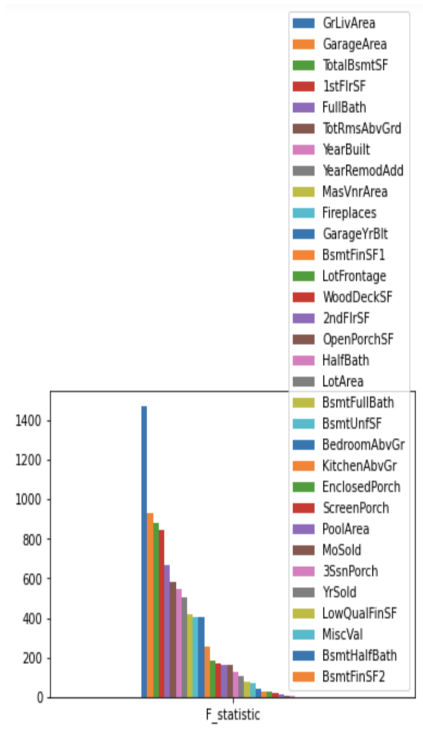


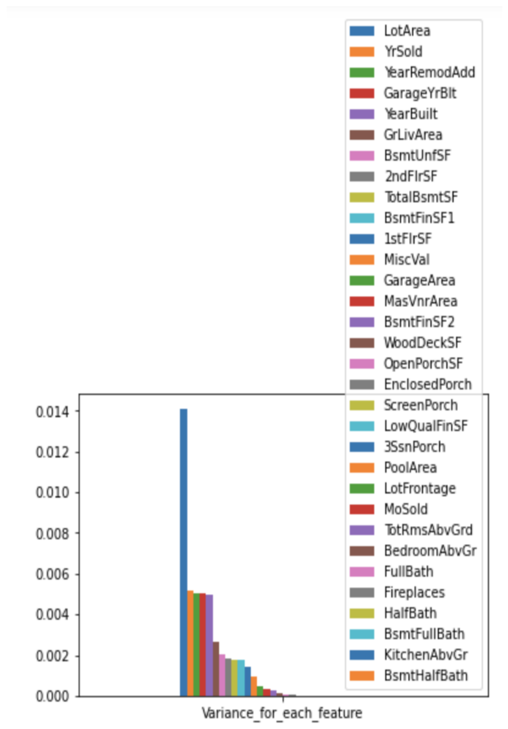




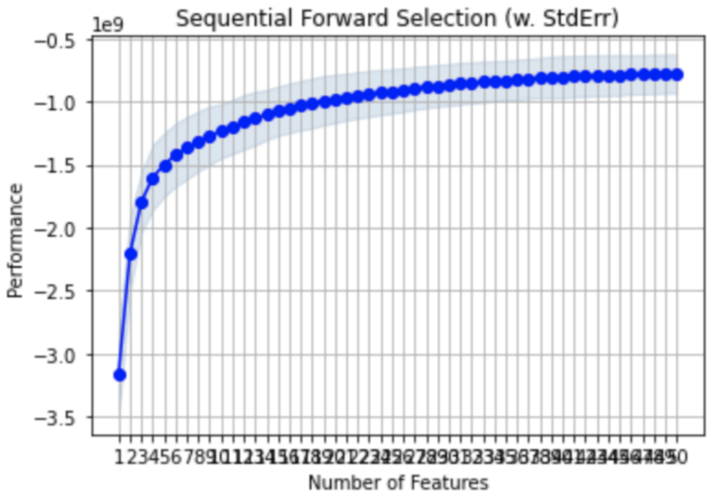


The above plots are the regression diagnostic test.





The above bar plots include the F statistic scores for all numerical features and variance for all numerical features starting from the highest value to the lowest value.



The above plot explores the relationship between the number of features and performance of the wrapper method model. Note: the performance indicates the negative mean squared errors.

Reference

Alvarez, R. (n.d.). *Creating diagnostic plots in Python*. Robert Alvarez - A blog about ML tidbits. Retrieved March 18, 2022, from https://robert-alvarez.github.io/2018-06-04-diagnostic_plots/

Bergstra, J., & Bengio, Y. (1970, January 1). *Random search for hyper-parameter optimization*. Journal of Machine Learning Research. Retrieved March 18, 2022, from https://www.jmlr.org/papers/v13/bergstra12a.html

Brownlee, J. (2020, August 20). *How to choose a feature selection method for machine learning*. Machine Learning Mastery. Retrieved March 18, 2022, from https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

Coelho, A. (n.d.). *Narrowing the search: Which hyperparameters really matter?* Blog. Retrieved March 18, 2022, from https://blog.dataiku.com/narrowing-the-search-which-hyperparameters-really-matter

DataTechNotes. (2021, February 11). *SelectKBest feature selection example in Python*. SelectKBest Feature Selection Example in Python. Retrieved March 18, 2022, from https://www.datatechnotes.com/2021/02/seleckbest-feature-selection-example-in-python.html

Jan Kirenz. (2021, December 27). *Lasso regression with python*. Jan Kirenz. Retrieved March 18, 2022, from https://www.kirenz.com/post/2019-08-12-python-lasso-regression-auto/

Kang, C. (2020, July 7). *Fine-tuning your xgboost model*. Chan`s Jupyter. Retrieved March 18, 2022, from https://goodboychan.github.io/python/datacamp/machine_learning/2020/07/07/02-Fine-tuning-your-XGBoost-model.html#Grid-search-with-XGBoost

Liyenhsu. (2017, November 18). *Feature selection and ensemble of 5 Models*. Kaggle. Retrieved March 18, 2022, from https://www.kaggle.com/liyenhsu/feature-selection-and-ensemble-of-5-models

Pocs, M. (2021, May 16). *Hyperparameter tuning in Lasso and Ridge regressions*. Medium. Retrieved March 18, 2022, from https://towardsdatascience.com/hyperparameter-tuning-in-lasso-and-ridge-regressions-70a4b158ae6d

prashant111. (2019, December 23). *Random forest classifier + feature importance*. Kaggle. Retrieved March 18, 2022, from https://www.kaggle.com/prashant111/random-forest-classifier-feature-importance

prashant111. (2020, November 18). *Comprehensive guide on feature selection*. Kaggle. Retrieved March 18, 2022, from https://www.kaggle.com/prashant111/comprehensive-guide-on-feature-selection

prashant111. (2020, November 18). *Comprehensive guide on feature selection*. Kaggle. Retrieved March 18, 2022, from https://www.kaggle.com/prashant111/comprehensive-guide-on-feature-selection#2.-Filter-Methods-

Random Forest hyperparameter tuning in Python: Machine learning. Analytics Vidhya. (2020, April 20). Retrieved March 18, 2022, from https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Regularization Tutorial: Ridge, Lasso & Elastic Net Regression. DataCamp Community. (n.d.). Retrieved March 18, 2022, from https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net

T., B. (2021, April 11). *How to use variance thresholding for robust feature selection*. Medium. Retrieved March 18, 2022, from https://towardsdatascience.com/how-to-use-variance-thresholding-for-robust-feature-selection-a4503f2b5c3f

Tsai, C.-F., & Hsiao, Y.-C. (2010, August 21). *Combining multiple feature selection methods for stock prediction: Union, Intersection, and multi-intersection approaches*. Decision Support Systems. Retrieved March 18, 2022, from https://www.sciencedirect.com/science/article/pii/S0167923610001521#s0055

XGBOOST parameters: XGBoost parameter tuning. Analytics Vidhya. (2020, November 23). Retrieved March 18, 2022, from https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/