KOÇ UNIVERSITY

Industrial Engineering Department
INDR 491 Industrial Engineering Design Project

Spring 2022

Modeling Personal Finance Credit Usage Characteristics of Credit Card Holders

Information about the team:

Name	ID	Email	Phone
1 Yeşim Demir	0064920	ydemir17@ku.edu.tr	+905545687850
2 Merve Sena Taşcı	0064654	mtasci17@ku.edu.tr	+905320672030
3 Firat Tamur	0060534	ftamur16@ku.edu.tr	+905418343471
4 Oya Suran	0069337	oyasuran18@ku.edu.tr	+905064888268
5 Bahar Öztürk	0064455	bozturk17@ku.edu.tr	+905346677876

Academic Supervisor:

Name	Signature
Mehmet Gönen	

Table of Contents

1. System Description

Finansbank is a private deposit bank established in 1987 [1]. It changed its name several times because of the acquisition of different companies. Finally, in 2016 its new name was changed to "QNB Finansbank" after it was acquired by the QNB Group.

It serves individual customers, small and medium-sized businesses, and commercial/corporate companies with approximately 10944 employees and 444 branches. Its services include cash management, opening/closing deposit/investment accounts, credit card transactions, project financing, and providing loans and insurance.

2. System Analysis

a. Problem Definition:

When customers need money, they resort to one of three ways which are taking credit, getting a cash advance from an ATM, or getting an instant loan. In the first way, when customers want to take out a loan, they have to go through many control mechanisms. Therefore, this is a time-consuming process for both customers and banks. Secondly, when customers choose cash advance from ATMs, customers are faced with a high-interest rate. Lastly, customers can choose to get an instant loan which is using the unused credit card limit as a loan. In this method, the interest rate is low and the control process is shorter. Because when a customer acquires a credit card, many security stages are deployed and passed without any issues. In addition, banks have to keep a partition of each credit card's limit in cash, thus when a customer does not max out their credit card the difference between this portion kept as cash and the unused limit has an opportunity cost on the bank. Therefore, an instant loan is the best for both the customer and the bank.

b. The objective and scope of the project:

QNB Finansbank aims to reach customers who do not reach their credit card limit by calling their employees at the call center and giving them instant loans. The aim of this project is to predict the

customers who are more likely to accept the loan by developing different models with the customer data accumulated by the company. Thus, the credit utilization rate will be increased, and agents will be able to use their time more effectively. Therefore, this project can be classified as a binary classification problem.

c. Analysis of data:

Data partitioned as train and test were transmitted by the company in excel format on March 4th. The train data contains 51 different columns of customer information, the content, and meaning of columns are not disclosed by the bank. There are 30000 rows. The test data contains 50 different columns of customer information, there are 15000 rows. We first look at the number of missing values in this data and whether we should impute it as Figure 1 shows there are missing values in 32 columns and in some columns the missing value amount is more than 50%.

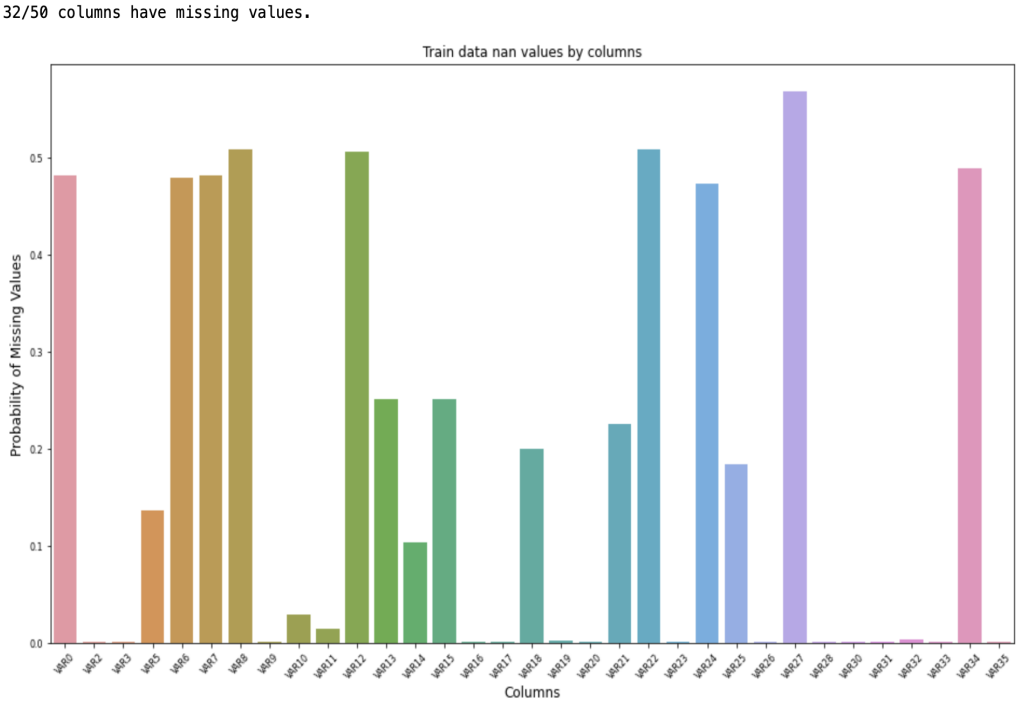


Figure1: Showing missing data amount in test data set.

To tackle this problem our supervisor advises us to test different methods of imputation for example using mean, median a constant value as well as predicting what the missing values can be from the other columns. We tried these methods, and more explanations and the results are

detailed in Section 4.

Furthermore, our data set is very imbalanced: we have 28509 rows labeled as the negative class, ‘0’, and 1491 rows labeled as the positive class, ‘1’. We also tried to visualize our data, mainly, we have boxplots for each column, count graphics to assess null values in each column, distribution graphics for each column, a heatmap to see the correlation between different columns, a covariance table, a scatter plot to see how data is spread and we have each columns target value subcategories where you can see in which sub-segment of a column value cast as positive or negative class. From these distribution graphs, some features had a very similar distribution as Figure 2 shows.

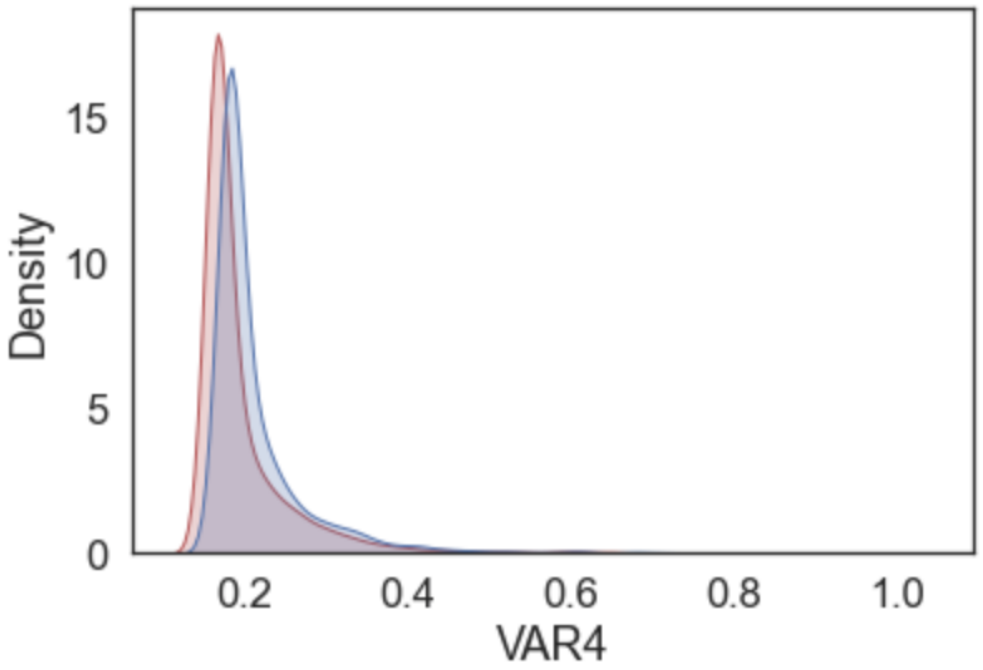


Figure2: Showing distributions of VAR1 and VAR2

Therefore, we thought 50 features was a lot and some of these features were both correlated and had similar distributions thus we thought feature selection was needed. We implemented different methods suggested for labeled data such as SelectKBest (Univariate feature selection) to select by statistical test values, RFE which looks at feature importance to select, and Sequential Feature Selection in the forward direction. Additionally, tree-based models select their own set of features. Initially, we gave selected features to these models as well but our supervisor showed us that by

doing so we are actually decreasing tree-based models' chances to give better predictions. Furthermore, we saw that we actually hurt our pipeline by selecting features and we did not need to do the feature selection. All plots are plotted in Jupyter Notebooks; since we have many different plots for all columns we did not include them in our report but the Appendix has a google drive link to our several different Jupyter Notebooks in pdf forms in notebooks there are also comments and explanations as to why we took the indicated steps. For data visualization please see Appendix, Jupyter notebook named Data Analysis & Visualization.

3. Literature Review and Sector Analysis

Our proposed system would be put into action internally this is why we are keeping sector analysis brief. There are 51 banks in Turkey. Out of 51, 9 of them are private banks like QNB Finansbank[2]. The financial sector in Turkey is growing in the past decade which pushes banks to give competitive interest rates to clients[3]. QNB Finansbank is among the 10 largest banks in Turkey[4].

There are several different ways to handle imbalance data:
One of them is Synthetic Minority Over-sampling Technique (SMOTE)[5] where data is synthetically generated for the less crowded class. This method yields better results in non-continuous data sets. Another applicable method is Random Undersampling, which refers to removing random samples from the majority class. There is also Random Oversampling which refers to adding the same random data from the minority class to the minority class over and over again. This technique is similar to tuning class_weigth parameters in many Machine Learning algorithms in Python[6]. Furthermore, undersampling and oversampling could be used together to attain better results. The combination of these techniques is used for credit card fraudulent transactions[7].

There are also several different techniques to handle missing-value imputations:
Some of the more straightforward methods are: filling missing values with mean, median, or mode [8]. Missing values can be predicted as well by K-nearest Neighbor (KNN)[9] or iterative approaches. There are also Tree methods like XGBoost[10], and LightGBM[11] which can handle

missing values automatically, with no need to preprocess them

We researched machine learning classifiers for proposals as well we have the same classification algorithms on tabular data which are:
Support Vector Machine (SVM)s, decision trees, random forests, KNN, and combining different classifiers like boostering algorithms. A set of high-dimensional features are spirited by hyperplanes in SVM [12], which yields maximum distance for data points between classes.

The Multilayer Perceptron Neural Network (MLP)[13] has one or more layers of neurons. Data is given to the base (input) layer and prediction is made on the output layer. Between input and output layers there could be hidden levels as well. MLP is a supervised learning technique that trains on a data set to learn a function that represents data. In Decision Trees [14] the goal is to learn decision rules from data features to predict the value of a target variable. Decision trees are a non-parametric supervised learning method that can be used for classification and regression problems. Random Forest [15] algorithms are meta estimators that employ averaging to increase predicted accuracy and control over-fitting by fitting several decision tree classifiers on various sub-samples of the data set. KNN [16] is a neighbors-based classification is a kind of instance-based learning, also known as non-generalizing learning, in which no attempt is made to build a general internal model, instead of storing samples of the training data. Classification is determined by a simple majority vote of each point's nearest neighbors: a query point is assigned to the data class having the most representatives among the point's nearest neighbors.

Boosting algorithms are built from a set of weak classifiers that output a single strong classifier. Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modeling problems. The AdaBoost [17] classifier is a meta-estimator that begins by fitting a classifier on the original data set and then fits additional copies of the classifier on the same data set but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. The LightGBM [18] is a GBDT open-source framework that enables highly efficient training over huge data sets while consuming little memory. Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) are two innovative approaches used by LightGBM. LightGBM can train each tree

with only a small portion of the complete data set thanks to GOSS. LightGBM handles high-dimensional sparse features significantly better than EFB. LightGBM also supports GPU-accelerated distributed training with reduced communication costs. XGBoost [19] is a scalable tree boosting technique commonly used by data scientists that produce cutting-edge outcomes on a variety of challenges. XGBoost is a theoretically justified weighted quantile sketch for approximate learning and a unique sparsity-aware technique for handling sparse data. For developing a scalable end-to-end system with tree boosting, XGBoost uses cache access patterns, data compression, and sharding.

4. System Design

A. System Architecture:

As Figure 3. shows our pipeline consists of several stages and components. The main input to our model is the data provided by QNB Finansbank. We have three major components, and our project is divided into stages inside these components. Our main components are Preprocessing, Training, and Prediction.
Preprocessing component encloses the first three boxes in Figure 3., the training component encloses three boxes after that, and lastly there is prediction.

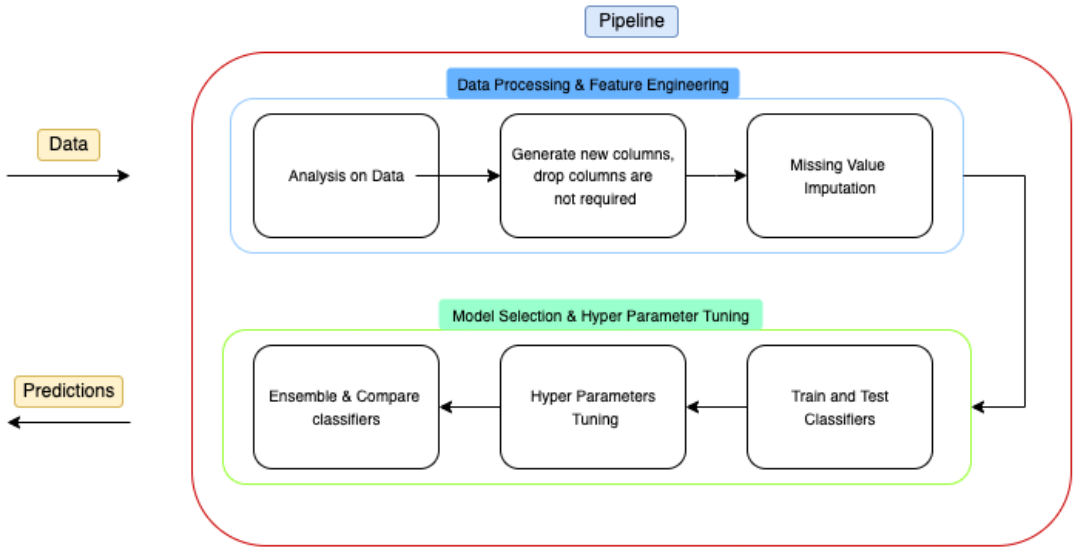


Figure3: Showing our pipeline for this project

Preprocessing component has 3 stages:

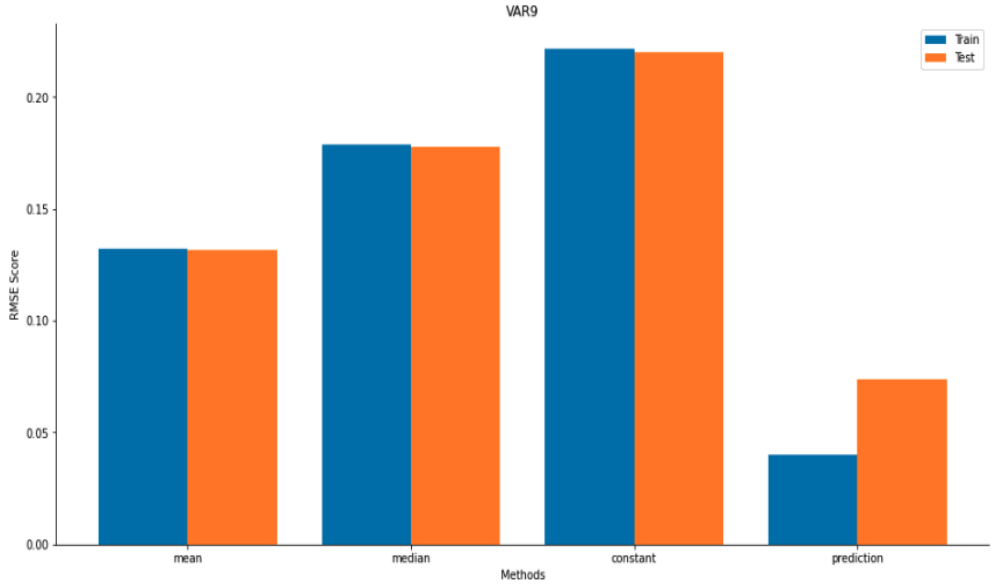
1. Analysis of Data:

   To understand what needs to be done, for example in this stage we saw that our data was imbalanced and missing many values. Visualizing our data was also a part of this stage. We have boxplots, count graphics to see null values in each column, distribution graphics, a heatmap to see the correlation between different columns, a covariance table, and a scatter plot for each column. Please see the Appendix, Jupyter notebook named Data Analysis & Visualization.

2. Handling Missing Values:

In this stage after we talked to our supervisor, we tried different methods for imputation and chose the best method to impute missing data in a specific column. To find the best method we looked at the RMSE score of each column imputed by different methods. Figure 4 shows an example of our results from imputation: For VAR9 and all other variable columns we tried imputing by using the mean value of the column, median of the column, a constant value, and by predicting the what value would be by training on filled rows of that column. We used XGBoost as our model for prediction. Please see the Appendix, Jupyter notebook named Imputation V2. For imputing the test data we used the

same method selected for that variable column in train data as the method for the test data variable column. If the best method was mean we used mean value for training set variable column to impute the test set's variable column's missing values. Because in the real world we will not offer instant loan options to 15000 people at the same time. Thus we won't have their data at the same time, and trying to impute test data by its own mean value would create this delusion.




```
Col: VAR9, Best RMSE: 0.07 with Method: prediction.
```

Figure4:

Imputation methods RMSE score for VAR9

1. Feature Selection:

   In this stage we tried different methods to select features. We used SelectKbest, Recursive Feature Elimination (RFE), Select from Model, Sequential Feature Selection.

Training component has 3 stages:

1. Preparation to train:

This stage included OneHotEncoding for categorical variables. This stage was needed because we used models like GaussianNB, KNeighbour, Logistic Regression, as well as tree based models, and ensemble classifiers like XGBoost and LightGBM. For example, KNN is based on distance and using numeric categorical values can lead to unintended results. This is why transforming categorical variables is necessary. We thought that OneHotEncoding would hurt the tree based model's prediction but our supervisor corrected us by saying if we increased the number of levels allowed in tree based models we can prevent this negative effect.

2. Training:

   We train different models and test them on our validation set which has the size of 20% of our training data. In Section 5 we show our initial results from various models on our imputed and One Hot Encoded data set. These results are our baselines to improve upon with hyperparameter tuning.

3. Hyperparameter Tuning:

   We wrote a wrapper class for all of our classifiers to make our implementation of training, testing, tuning faster and easier to use. For hyperparameter tuning we use RandomizedSearch to find the better values for hyperparameters. We used *sklearn* library's [20] "RandomizedSearchCV " function.

Prediction component has 2 stages:

1. Prediction:

This is the stage where we got our best results from each classifier we tuned in the Training component.

2. Performance comparison:

In this stage we looked at performance of each classifier from different metrics and saw how they compare. Additionally, we compare our results to QNB Finansbank's results as well.

B. Model Development and Validation:

We used different methods for different tasks and from the results of these tasks we have different classifiers to do binary classification. Firstly, we have different methods for imputation and we compare their RMSE scores to pick which one works best for each column. These methods are: filling missing values by the mean, median, and constant value, and predicting missing values for each column. Then we also deployed different methods for feature selection: SelectKBest (Univariate feature selection) to select by statistical test values, RFE which looks at feature importance to select, and Sequential Feature Selection in the forward direction. Additionally, tree-based models select their own set of features. Even though in the end we did not eliminate any features we moved on to the Training component with all the features. Then we have 7 different classifiers: GAUSSIANNB, Decision Tree, KNN, Logistic Regression, RandomForest, XGBoost, and LightGBM. Each of these classifiers gets its hyperparameters tuned by RandomCV. We check how they perform on our validation set again and compare their non-tuned results with tuned results. After that, we move on to our last component prediction, where we give our test data and see how all these classifiers perform. We compare their area under the roc curve, Receiver Operating Characteristics (ROC), and we also look at confusion metrics and compare precisions. Finally, QNB Finansbank ROC Curve result of QNB Finansbank is 0.8 and Gini Index is 0.6.

C. Consideration of alternatives:

We had different alternatives in different steps, our alternative methods for imputation and feature selection are already discussed in previous sections. However, we considered different alternatives for other parts of our project as well. For example, to get rid of imbalance we considered 3 different methods: undersampling the negative class, oversampling the positive class, and using the SMOTE method to generate more positive class data. After we discussed SMOTE with our supervisor we did not use it, because our data set contains discrete features. Moreover, for imbalance issues, at the training point in the project, we repeated positive class data 19 times, which is the proportion of the negative

class to the positive class. After repetition, we gave this new data set to our classifiers and trained them. The results of this are shown in Figure 5.

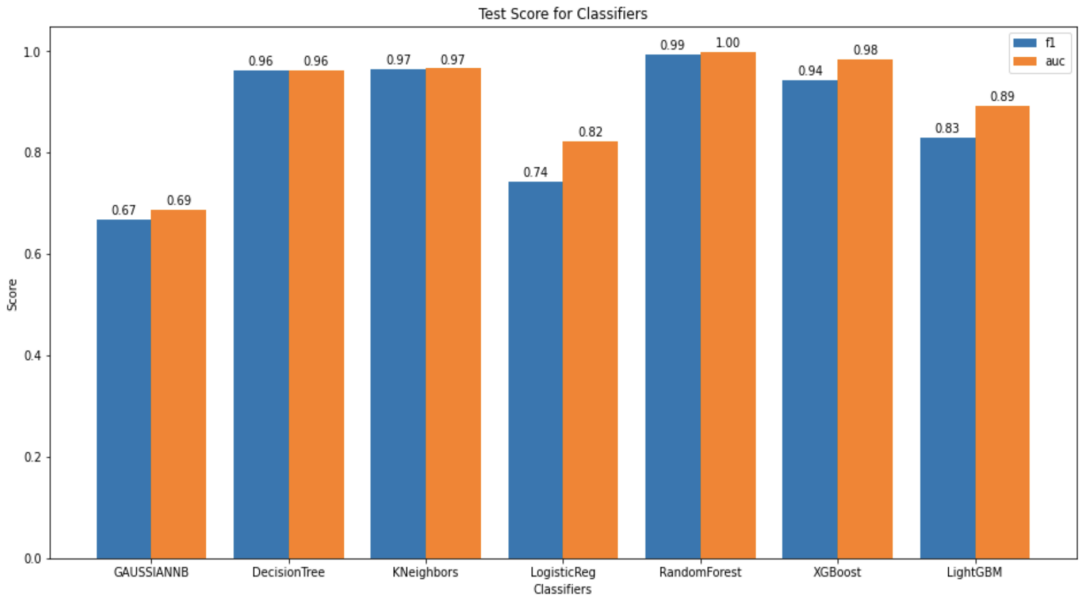


Figure5: Showing Area under ROC, F_1-Score

As you can see these results were too good to be true. Our supervisor warned us about simplifying the problem by repeating the data over and over again and also doing this repeating process for the validation set where we were testing our classifiers and getting these results. Thus these results did not reflect the truth and we needed to correct this error. Our supervisor suggested that we use class weight parameters when we train our classifier. This parameter directly makes the model know that there is an imbalance between classes and its rate is specified by the parameter. Furthermore, at the beginning phases of our project, we made HeatMap see correlations between the features. There was a high correlation and we thought we might eliminate some of these highly correlated features because we were planning to use Logistic Regression and multicollinearity can harm logistics models. However, this thinking was incorrect since we used nonlinear models like XGBoost as well. Furthermore, in the hyperparameter tuning part, we used Randomized Search but seeing that there was no increase in XGBoost AUC scores we used optimization on the hyperparameters to get better results. To do this we used the *hyperopt* library [21].

D. Solution methodologies:

We have a binary classification problem in our hands. In each stage mentioned in part, we have different approaches but essentially we want to train the best predicting models. At first, we needed to prepare the data and understand the underlying relationships that paralleled our initial step of Preprocessing. In which we draw plots to see if there is a direct correlation between different features as well as to see the amount of data we were missing. In this step, we realized that we needed to impute the missing data which was done by a couple of different methods to make sure that we had the best possible imputation method for each column. For example, for the prediction method, we look at a null value in a specific column, and we train XGBRegressor by different columns and this specific column's non-null value for each specific row. Then we divide that column into 3 piece tests, validation, and training. We train and validate our model's prediction by comparing its prediction to the true value and finding the RMSE score. After each method is tried on that specific column we look at their RMSE scores to select the best method. After this step, we moved on to Feature Selection. This was also done with different methods like Wrapper Methods such as RFE which was combined with cross-validation to find out how many features we need to give RFE to get the best result as well as filtering methods like Forward Selection. However, after we saw that we hurt our models by giving only a selected number of features, and consulting with our supervisor we proceeded with using all the features given to us by QNB. Then we moved on to the second component of our pipeline where we train our models. Before doing so since there are categorical variables 15 columns to be exact we did OneHotEncoding to give our data correctly to methods such as KNN. Then we started to train our selected 6 models. After the training, we used RandomCSV for some hyperparameters of each model. Random search yields not the optimal tuning but better tuning for each model. Then we trained our newly tuned models again to compare the new results. In the end, we were ready to train our model and predict on the test set which was 20% of our training set not shown to the models before this step. In Section 5 we show the results of our model's predictions.

Then we moved on to understanding the performance of each model. For this purpose, we looked at their AUC results, and Gini Index as well as the confusion matrix. We mistakenly looked at their accuracy but realizing that our data is imbalanced accuracy would not make a good measurement of performance.

5. Implementation

Our implementation of the project resides in the Jupyter Notebooks link given in the Appendix. We have a data folder where data provided by QNB, the data inputted by best methods, data encoded by one-hot encoding, and under and oversampling of training data reside. Under the main folder, there is the Preprocessing folder in which we did all the steps specified in Section 4 subpart a. There are two different versions of imputation version 2 is the most recent version in which we call the imputation method specified in utils_imputation python file where we have imputation by mean, median, constant value as well as by predicting the value with XGBRegressor. Then imputation compare methods look at the min RMSE score and select that method for imputation. We also put these findings into graphs. As Figure 4. showed prediction can give the best result for some variables but other methods can also give the best results as well. In this folder, we also have initial test scores where imputation is done by "0" for all the missing values as Figure 6 shows. We wanted to see what the initial results were.

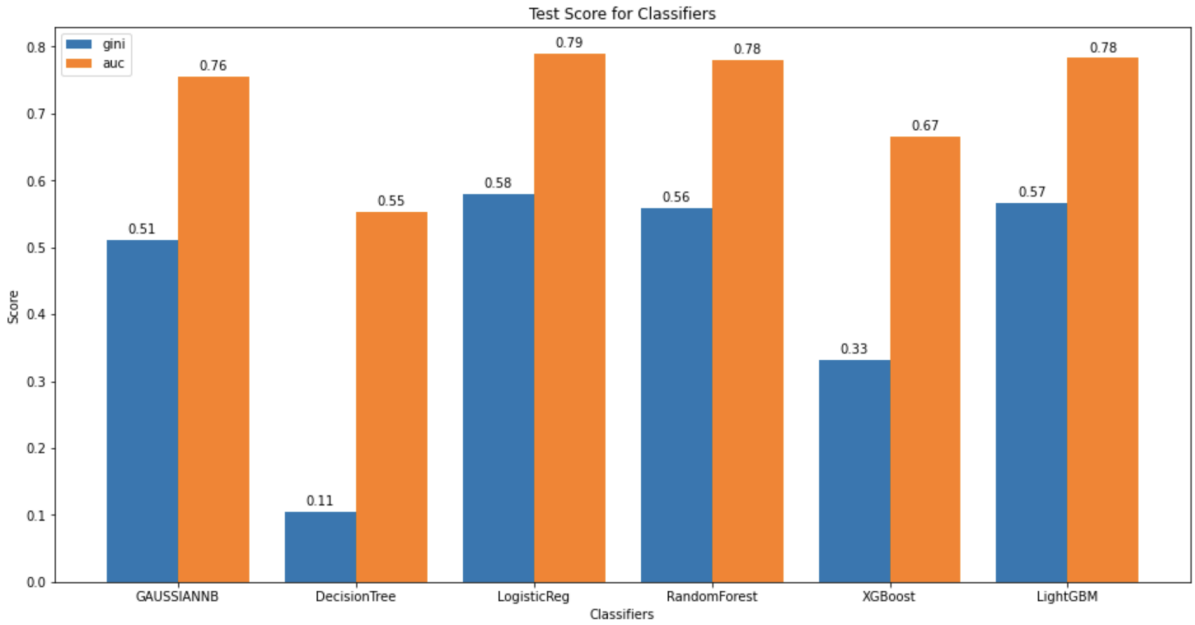


Figure6: Showing test score for initial case (imputed with 0) test size 0.2

Then we impute each column with the best method. Then we have move to Feature Selection for our data set we tried specified methods for feature selection by calling *sklearn* library's "SelectFromModel", "SequentialFeatureSelector", "RFECV" and "SelectKBest", "chi2", "f_classif". "Chi2 " and "f_classif" are used in the SelectKBest method as statistical tests. In our Jupyter Notebook named "Train & Test Models - Imputed & Feature Selection" we compute each method's selected features and how much these features affect our test and train scores for our selected models. Initially we make each feature selection method to select 10 features and get testing results from selected 10 features for each classifier model. Then we increment the feature by then and do the same process again. Figure 7 shows the results for selecting 40 features from the method "SelectFromModel", and training our classifiers with these features. Only Figure 7 is shown since it has the best results even though its results are worse then Figure 11 where no feature selection was done. Thus we moved on with no feature selection.

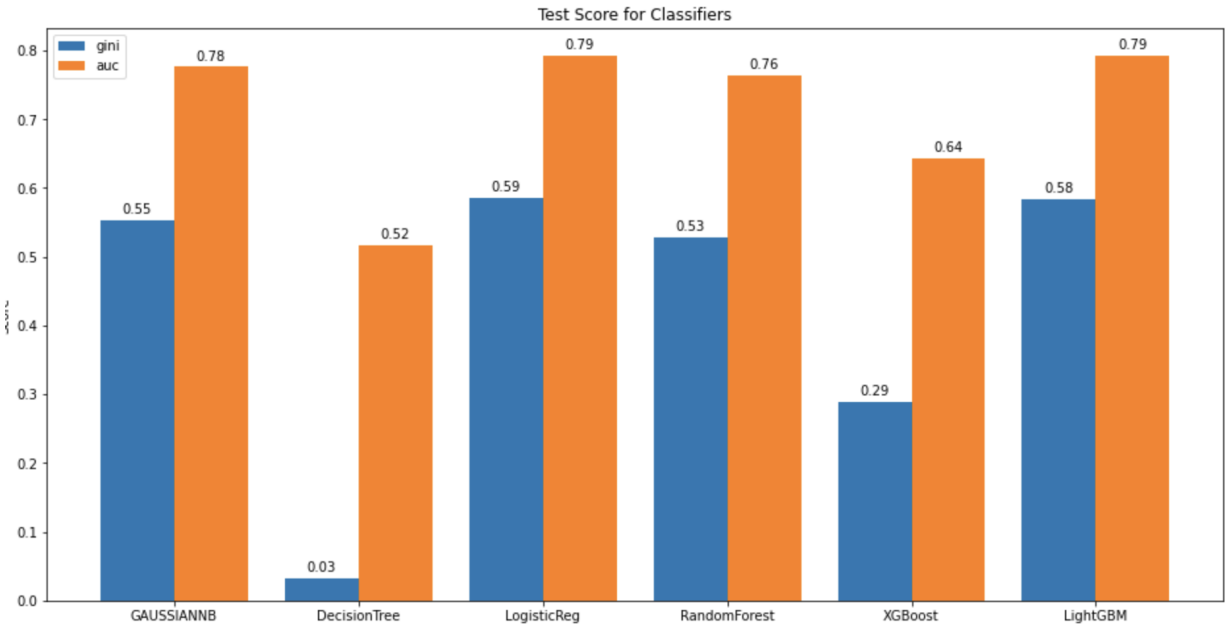


Figure7: Showing test score for 40 selected feature for each classifier initial test size 0.2

This endeavor showed us problems with this size of training data and variable amounts we may not need feature selection.

Then we have a OneHotEncodingCategoricalColumns notebook where we use *sklearn* library's "OneHotEncoder" on categorical columns named as cat_columns and we transform them. Then we move onto the training part of our implementation. To make our models implement hyperparameter tuning training and predicting processes easier to use and relying on the same API we created our custom_ml.py class this file can be found in the model's folder. In this file we have functions which will need to be implemented for each classifier. There might be some alterations for each classifier thus they are abstract methods. Then we have a custom_ml_classifiers.py file where we have all of our selected models: GaussianNB, DecisionTree, KNN, LogisticRegression, RandomForest, XGBoot and LightGBM. These models need to tune different hyper parameters thus inside their class definitions we alter hyperparameter_search function according to their specifications. For example, Figure 8. Shows alterations for LightGBM, we found our initial hyperparameter settings from different websites for example since we are doing binary classification adjusting the objective parameter of LightGBM to 'binary' helps us to improve our predictions in this model.

```
def hyperparams_search(self, n_iter = 100):

    #https://neptune.ai/blog/lightgbm-parameters-guide
    #https://github.com/Microsoft/LightGBM/issues/1339
    #https://towardsdatascience.com/hyper-parameter-tuning-in-python-1923797f124f
    """

    Hyperparameter search for the self.model.

    :param X  : test input features data
    :param y  : test input target data

    :return   : score for each metric

    """

    parameters = {
                  'num_leaves'             : [20, 40, 60, 80, 100],
                  'min_child_samples'      : [5, 10, 15],
                  'max_depth'              : [-1, 5, 10, 20],
                  'learning_rate'          : [0.05, 0.1, 0.2],
                  'reg_alpha'              : [0, 0.01, 0.03],
                  'objective'              : 'binary',
                  'metric'                 : 'auc',
                  'is_unbalance'           : True,
                  'boosting'               : 'gbdt',
                  'early_stopping_rounds'  : 30
                 }

    self.model = RandomizedSearchCV(self.model, parameters, scoring='accuracy', n_iter=100)
```

Figure8:

Showing alterations in hyperparams_seacrh function for LightGBM

Then we have a wrapper class named classifier.py where we have a list of custom classifiers and we apply the same functions over and over again. We did our training procedure in this way so that we can add and remove different classifiers easily for ease of use. In our main folder there are several jupyter notebooks where we trained our custom classifiers. We made some mistakes then corrected them in this part. Initially we thought we should do oversampling for positive class class and train our models by having equal amounts of positive and negative class which are but imputatted and encoded.

In the Jupyter Notebook named “Train & Test Models - Imputed - OverSampled - OneHot” we give train data in which positive class labeled data was repeated 19 times to match the number of negative class labeled data. Then this data was undergone OneHotEncoding and then given to our selected models to train and test their precision results in our validation set which has 20% of the train set as Figure 9 shows.

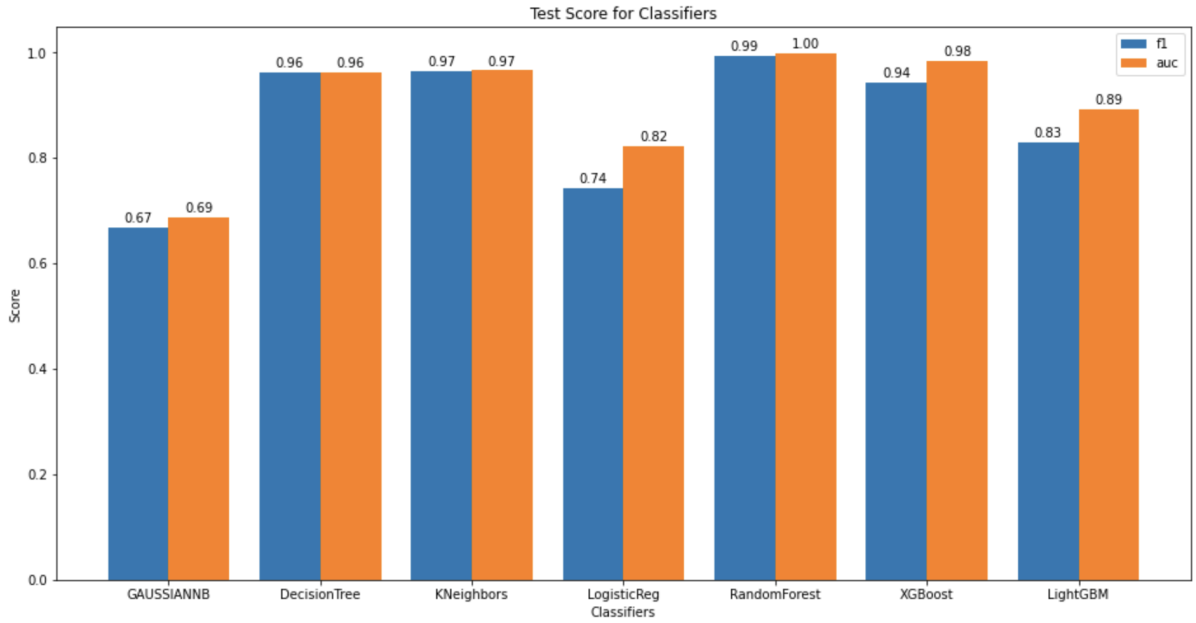


Figure9: Showing test score for oversampling for selected models

This result was too good to be true because we messed up our validation set too, which was one of the mistakes we made in this part. Then we tried undersampling in our data set. In the jupyter notebook named "Train & Test Models - Imputed - UnderSampled - OneHot" we took 5.26% of our negative class which was shuffled and this data was also undergone one hot encoding and then given to our selected models to train and test their precision results in our validation set which has 10% of train set. Figure 10. Shows our results for the validation set.

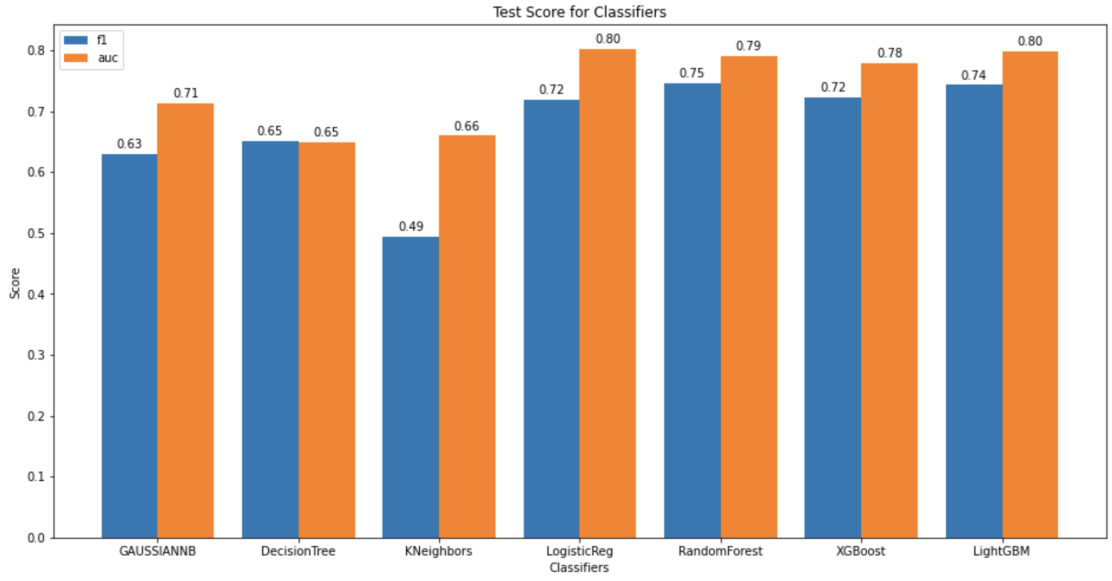


Figure10: Showing test score for undersampling for selected models

However, over and undersampling did not solve our problems and consulting with our supervisor we changed class_weight parameters in our custom_classifiers to represent imbalance in our data set. After making this correction our results for test size 20% of the train set are shown below in Figure 11.

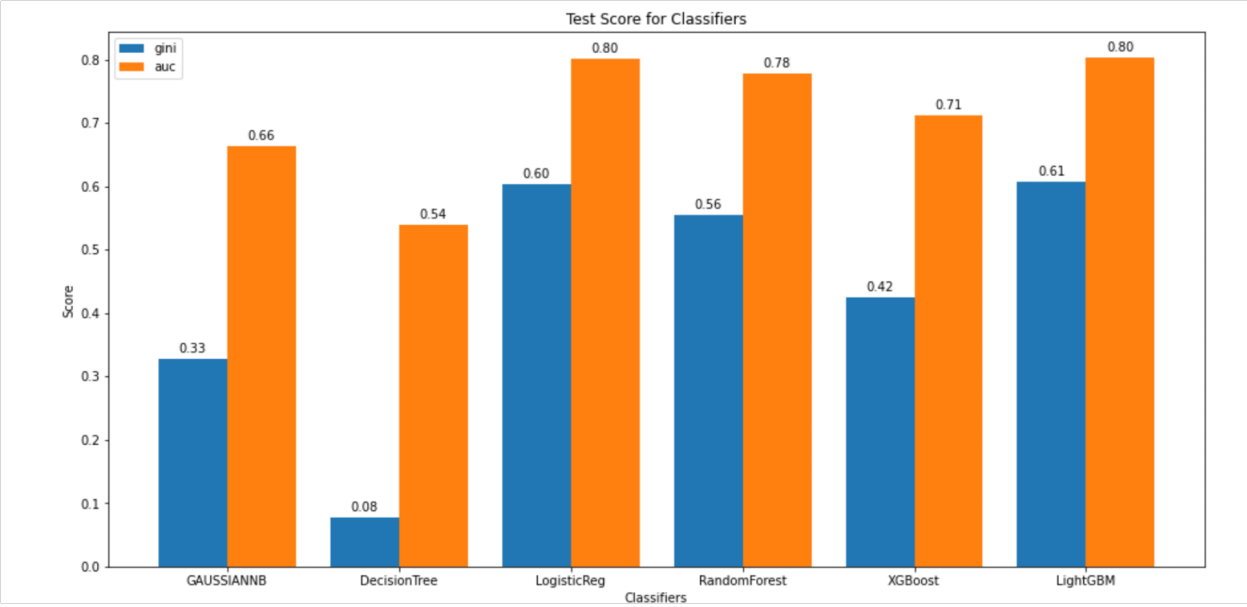


Figure11: Showing test score (auc,gini index) where models adjusted with class weight

Additionally, there is a confusion matrix for each of our models, represented in Figure 12.

Then we wanted to see how hyper parameter tuning changed our ROC curve results and how our Gini Index is doing. Hyperparameter tuning helped nearly all of our models. However Randomized Search does not give globally optimal results thus we wanted to try an optimization library on tuning as well. In a Jupyter Notebook named "Hyperopt on XGBoost" we tried the Hyperopt library and it gave us best hyper parameter results from its trial runs. Figure 13 shows the results comparing AUC curve of XGBoost hyperparameter tuned with Randomized Search vs by Hyperopt optimization.

```
Testing GAUSSIANNB...
 ###### Confusion Matrix #######
[[ 302 5400]
 [  13  285]]
 ##############################
 executed in 0.1 seconds.
Testing DecisionTree...
 ###### Confusion Matrix #######
[[5419  283]
 [ 255   43]]
 ##############################
 executed in 0.0 seconds.
Testing LogisticReg...
 ###### Confusion Matrix #######
[[5690   12]
 [ 288   10]]
 ##############################
 executed in 0.0 seconds.
Testing RandomForest...
 ###### Confusion Matrix #######
[[5701    1]
 [ 297    1]]
 ##############################
 executed in 0.5 seconds.
Testing LightGBM...
 ###### Confusion Matrix #######
[[5700    2]
 [ 295    3]]
 ##############################
 executed in 0.0 seconds.
```

Figure12:Confusion matrix for test set for all models

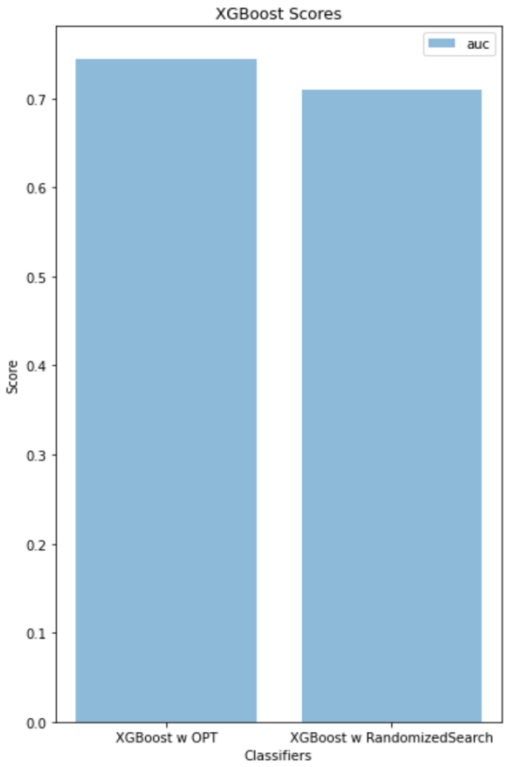
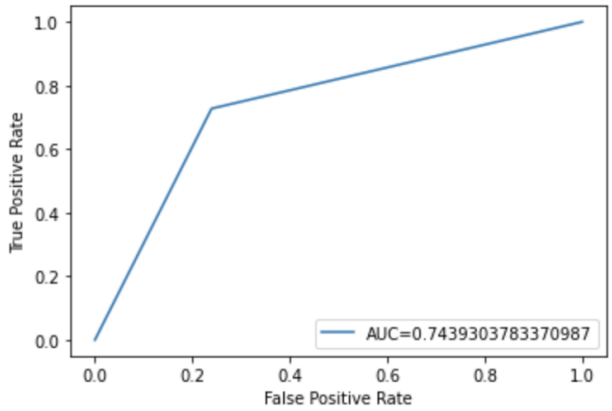


Figure 13: Showing results of XGBoost

For the XGBoost AUC graph and confusion matrix result is shown in Figure 14.




```
auc: 0.7439303783370987
```

```
[[4346 1368]
 [  78  208]]
```

Figure14: Showing auc score when XGBoost's hyperparameters are tuned with optimizer

Hyperparameter tuning with RandomizedSearchCV function results are shown in Figure 15.

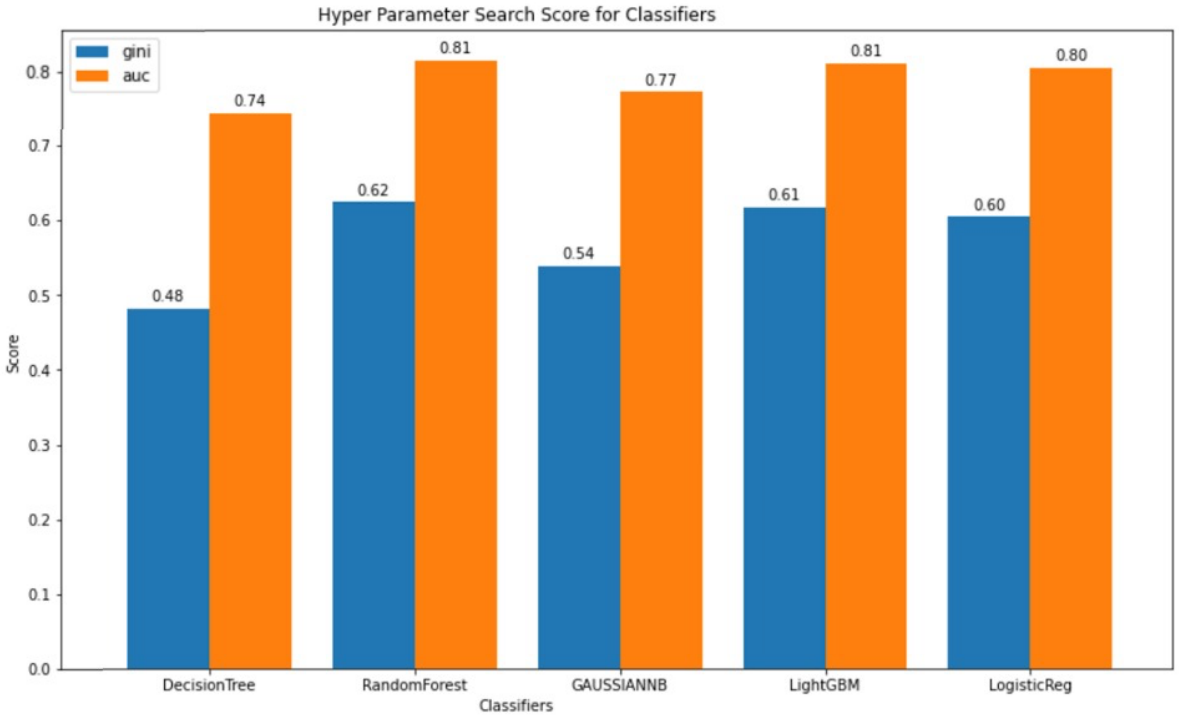


Figure15: Showing test score when model's hyperparameters are tuned

It seems we achieved the best results in LightGBM and Random Forest. However, looking at Figure11 shows that our best results are in GaussianNB. At this point, we considered doing an ensemble classifier but considering LightGBM and XGBoost are ensemble classifiers we opted against it. Our results are close to QNB Finansbank results however, while we were trying to see how we performed we looked closely into how our classifiers learn each class. It seems they learn negative classes better as is expected since there are many more examples of this class.

Our project deliverables are our custom models which were trained and tuned for this binary classification problem. We also have wrapper class and custom_ml class, which can be used easily to adapt and modify our models. Our models can be improved upon especially if the hyperparameter search function was changed from RandomSearch to GridSearch which is a sure way to find the optimal parameter. Furthermore, our initial starting point and our search space for RandomizedSearch can be adjusted for better results. Through this project, QNB finance bank can moderately predict whether a customer would accept the instant loan option the bank offer. If our

models can be improved upon, maybe the bank can directly find customers who will accept this offer with a higher probability. If this is the case the bank can decrease its opportunity cost of cash money they need to keep for some portion of the unused credit card limit.

References

[1]“Hakkımızda,” Qnbfinansbank.com. [Online]. Available: https://www.qnbfinansbank.com/qnb-finansbanki-taniyin/hakkimizda. [Accessed: 04-Mar-2022].

[2]Tbb.org.tr. 2022. TBB. [online] Available at: <https://www.tbb.org.tr/en/modules/banka-bilgileri/banka_Listesi.asp?tarih=7/4/2022> [Accessed 8 April 2022].

[3]Bstdb.org. 2022. [online] Available at: <https://www.bstdb.org/BSTDB_Overview%20of%20the%20financial%20sector_TURKEY.pdf> [Accessed 8 April 2022].

[4]Corporate Finance Institute. 2022. Top Banks in Turkey. [online] Available at: <https://corporatefinanceinstitute.com/resources/careers/companies/top-banks-in-turkey/> [Accessed 8 April 2022].

[5]Arxiv.org. 2022. [online] Available at: <https://arxiv.org/pdf/1106.1813.pdf> [Accessed 8 April 2022].

[6] W&B. 2022. Weights & Biases. [online] Available at: <https://wandb.ai/authors/class-imbalance/reports/Simple-Ways-to-Tackle-Class-Imbalance--VmlldzoxODA3NTk> [Accessed 8 April 2022].

[7] Ieeexplore.ieee.org. 2022. Combining oversampling and undersampling techniques for imbalanced classification: A comparative study using credit card fraudulent transaction data set. [online] Available at: <https://ieeexplore.ieee.org/document/9264517> [Accessed 8 April 2022].

[8] scikit-learn. 2022. sklearn.impute.SimpleImputer. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html#sklearn.impute.SimpleImputer> [Accessed 8 April 2022].

[9] scikit-learn. 2022. sklearn.impute.KNNImputer. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html> [Accessed 8 April 2022].

[10] Arxiv.org. 2022. [online] Available at: <https://arxiv.org/pdf/1603.02754v2.pdf> [Accessed 8 April 2022].

[11] Lightgbm.readthedocs.io. 2022. Advanced Topics — LightGBM 3.3.2.99 documentation. [online] Available at: <https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html> [Accessed 8 April 2022].

[12] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks" (PDF). *Machine Learning*. 20 (3): 273–297. CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018. S2CID 206787478. Cortes, C. and Vapnik, V., 1995. Support-vector networks. *Machine Learning*, 20(3), pp.273-297.

[13] scikit-learn. 2022. *1.17. Neural network models (supervised)*. [online] Available at: <https://scikit-learn.org/stable/modules/neural_networks_supervised.html> [Accessed 8 April 2022].

[14] "1.10. decision trees," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html. [Accessed: 04-Mar-2022].

[15] "Sklearn.ensemble.randomforestclassifier," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: 04-Mar-2022].

[16] "1.6. nearest neighbours," *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html#classification. [Accessed: 04-Mar-2022].

[17] Y. Freund, R. Schapire, "A Decision-Theoretic Generalisation of on-Line Learning and an Application to Boosting", 1995.

[18] "Lightgbm," *Microsoft Research*, 21-Nov-2021. [Online]. Available: https://www.microsoft.com/en-us/research/project/lightgbm/. [Accessed: 04-Mar-2022].

[19] "XGBoost: A scalable tree boosting system - arxiv." [Online]. Available: https://arxiv.org/pdf/1603.02754. [Accessed: 04-Mar-2022].

[20] "sklearn.model_selection.RandomizedSearchCV" [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. [Accessed: 04-Mar-2022].

[21] "Hyperopt: Distributed Asynchronous Hyper-parameter Optimization" [Online]. Available: http://hyperopt.github.io/hyperopt/. [Accessed: 20-May-2022].

Appendix

Github Link: https://github.com/firattamur/QNBFinansBank

Drive link for Jupyter Notebook pdf: https://drive.google.com/drive/folders/1xIlztFCEG2VscuGhtguS-fPQR0QX_bji?usp=sharing