**CIS 432 Final Project Report**

**Overview**

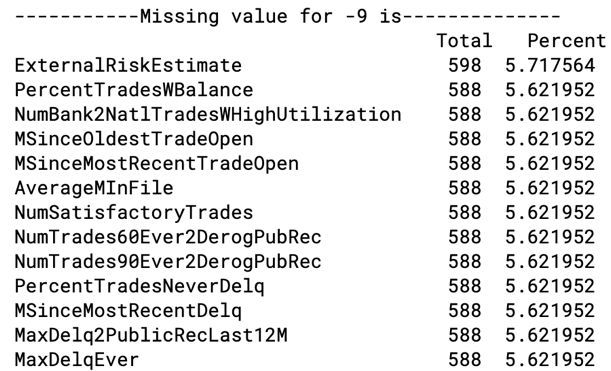
In this project, we developed a predictive model and a decision support system that evaluates the risk of Home Equity Line of Credit (HELOC) applications. We cleaned the data, did data pre-processing, and then selected best model by comparing nine machine learning models commonly used in finance industry. After that, we used streamlit to build an interface for users to predict a customer’s default risk by simply drawing the sidebars. Our interface is user-friendly enough so that those who has little technical proficiency can also understand prediction result in an intuitive way.

**Part1: Data Cleaning and Data Pre-processing**

In this case, there are 10459 observations, 23 features and target variable are ‘Bad’ and ‘Good’. Generally, the data for this case is relatively clean, but we still need to deal with missing values and categorical variables.

**Step1: Missing value analysis**

We firstly conducted the missing value analysis to observe the missing value distribution. This analysis can help us determine how we could deal with those value without losing too many records. For example, we found that for -9, the distribution is uniform among all the attributes, and the proportion is relatively small. So, we just delete all the rows containing this number. For -8, we found in feature ‘NetFractionInstallBurden’, the percentage of -8 is 32.6%, which is relatively large. So, we decided to delete this column and used imputer to fill the missing values with mean value. Note that we also delete the 'MSinceMostRecentDelq' and 'MSinceMostRecentInqexcl7days' but not used imputer method considering that simply imputing will bring lots of noise to this dataset (two reasons here: 1) NaN in variable means not satisfying the condition, NOT missing information, predicting a value for this variable will be inaccurate; 2) the proportions of NaN in those two variables are 44% and 17% respectively, which are relatively large). In the final part, we will briefly discuss how we can improve the method of dealing those two variables.



**Fig. Missing Value Analysis**

**Step2: Deal with Categorical Variables**

We also used LabelEncoder to deal with categorical variables 'MaxDelq2PublicRecLast12M' and 'MaxDelqEver'. Note that we abandon the one-hot code for the convenience of coding for interactive system.

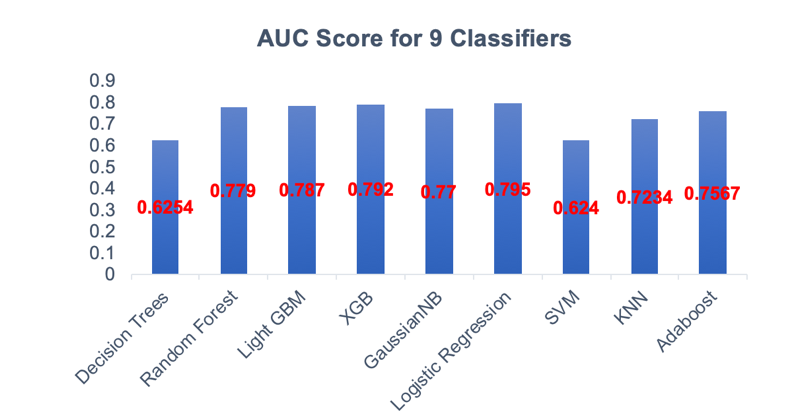
**Part2: Model Selection**

In this case, we have used 9 models: Decision Trees, Random Forest, Light GBM, XGB, GaussianNB, Logistic Regression, SVM, KNN, AdaBoost to train the data and selected the best one by comparing final AUC score. Note that we add the trial of Light GBM and XGB because those two classifiers can perform well using gradient algorithm and are extremely powerful to deal with categorical variables.

* **model performance：**here we prefer to use AUC score rather than accuracy because we aimed to predict the default risk for each person rather than just using label 0 or 1. For the calculation of accuracy, the threshold setting will influence the result a lot but AUC score can avoid such problem.
* **Tune the hyperparameters:** here we employed two generally used method.

1. **Grid Search**: set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried, thus this method can be very inefficient.
2. **Automated Hyperparameter Tuning:** Use methods such as gradient descent, Bayesian Optimization, or evolutionary algorithms to conduct a guided search for the best hyperparameters. Cross validation and early stopping will be implemented using the LightGBM cv function. We will use 5 folds and 100 early stopping rounds. We tried this method to tune our Light GBM model and find the best hyperparameter ‘n\_estimators’.

According to final AUC score on the test dataset, we find that Logistic Regression, Light GBM and XGBoost perform best on this dataset.



**Fig2. AUC Scores Comparison**

**Part3: Interface System Design**

Based on 3 best models, we used streamlit package to build a interface for our users. The interface is an interactive system which contains a sidebar and the main menu. In the sidebar we can adjust several attributes like external risk estimates and months since oldest trade open etc. If the sales representative does not know the range of those attributes, he can click the ‘dataframe’ checkbox to have a general idea about those data. Additionally, he can also choose to see the performance comparison of 3 best models we apply and see what he want most. After he inputs those information, this system will automatically calculate the default risk based on the machine learning model he chooses, also the accuracy and AUC score based on that chosen model.

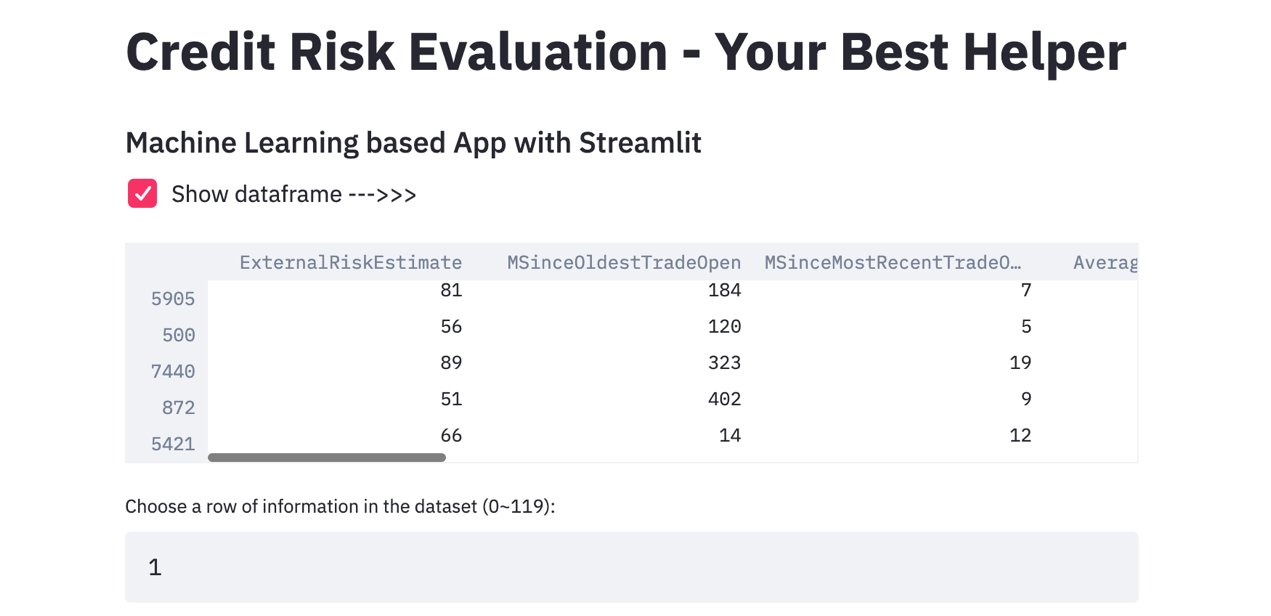


Fig4. Choose Dataframe to See

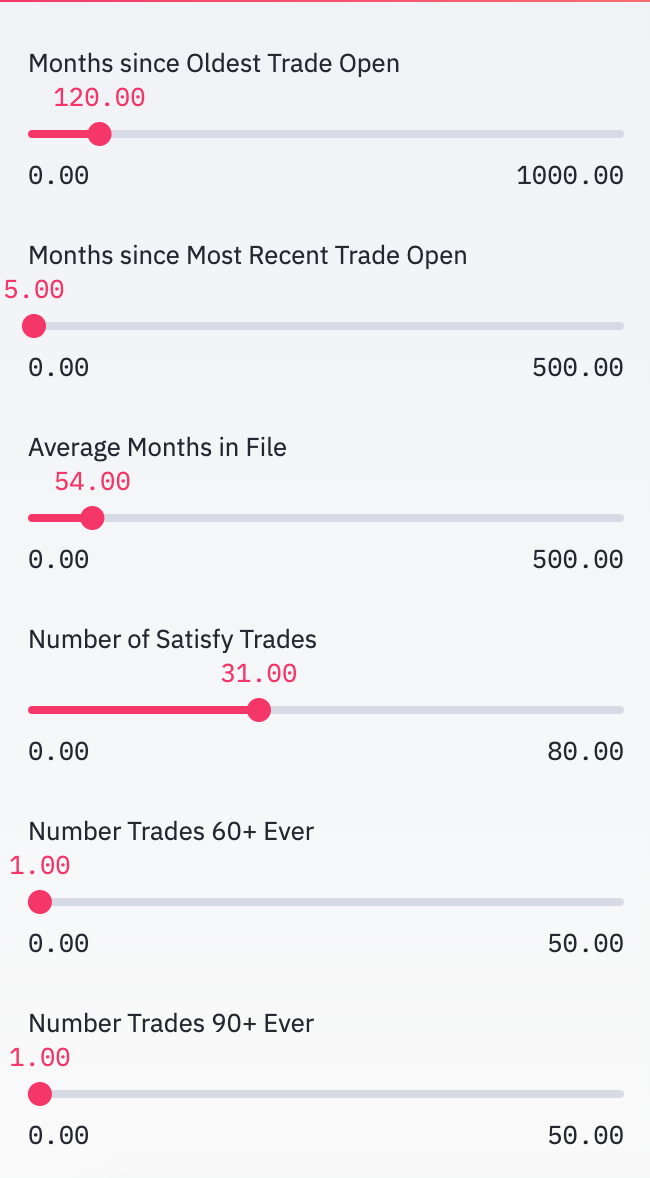


Fig3. User Input Interface

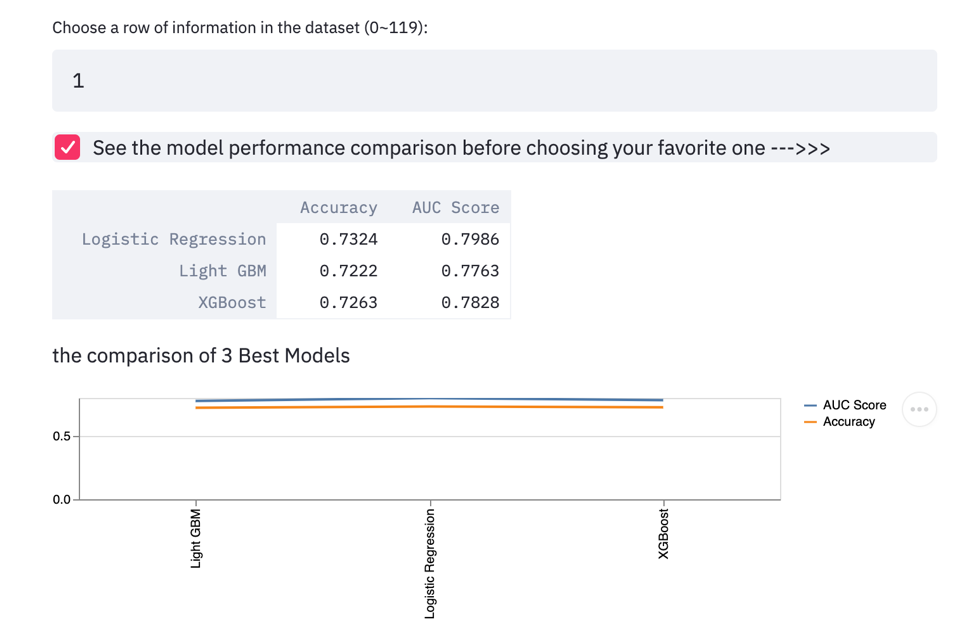


Fig5. See the model performance comparison

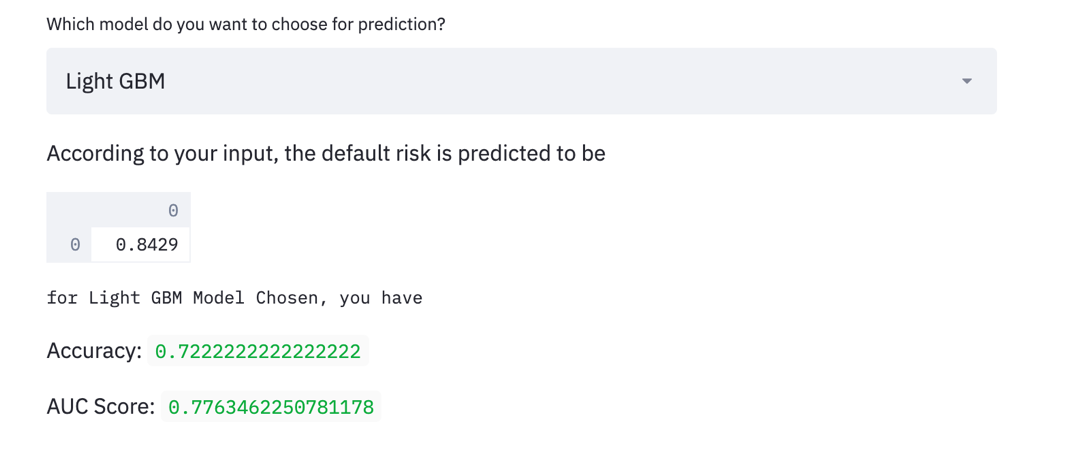


Fig6. See the predicted default risk and model performance for chosen model

**Part4: Some Reflections**

* **Data Preprocessing:**

we just delete the 'MSinceMostRecentDelq' and 'MSinceMostRecentInqexcl7days'. However, we consider to use some categories to deal with those variables. For example, the ‘no such inquiries and not satisfying’ users can be attribute to category1, 0-5 months users can be attribute to category 2, and so on. Such method can save the precious data records and may improve the final performance for our model.

* **Model Selection：**

We notice that although Light GBM and XGBoost are excellent classifiers for such classification problems, it seems they work not so well for our dataset. Firstly, we infer it may be because those classifiers can perform well for bigger dataset (our credit dataset has merely 10459 records, relatively small.) Additionally, we employ the Automated Hyperparameter Tuning to select the best hyperparameters. However, this method help us to find the ‘n\_estimators’, but other parameters cannot be set the optimized one. Later, we want to explore how we improve this method to find more optimized parameters (also a method which can save computing time!)