A Predictive Model of Credit Risk

Team Members: Isaac Revette, Jinren Liu, Zhuoyi Ying, Ruixuan Zhao, Zhikai Zhu

December 16th , 2019

Introduction

Our idea to this project is to develop an interpretable predictive model and make a decision support system that could help the users evaluate the risk of Home Equity Line of Credit applications in a simple, but effective way. Before we start building up the model, we recognize that there are some special numbers (-7, -8 and -9) that could affect the accuracy of our predictive model. To make our model more accurate, we decide to clean the data before any model training can be done. Next, to achieve our goal, we have divided our data into training and testing sets and tried several models. After carefully computation and evaluation, we find the Gradient Boosting model fits our data the best. The detailed description of our best-fitted model and the process of choosing the model will be discussed in more depth in the rest of the report. Although our model can be interpretable, our mission is to develop the simplest system that could help the users explain each prediction with little need for technical proficiency. To achieve this goal, we have transferred our code into an interface, via Streamlit , to turn the decisions of the predictive model into simple visualizations. In addition to our best-fitted model, our interface includes similar performing models and easy methods to manipulate the data. We will discuss the details of our interface in the following sections of the report.

Description of the Model

Data Cleaning

Our principle strategies for data cleaning is first to binarize the target column and then delete the rows that have all the information missing first (the rows that have all its column value equal to -7 or -8 or -9). Next, we delete the columns where missing values make up more than 90%, which only left 18 columns that we think are meaningful. To achieve this goal, we define two functions: *binarize_labels()* to binarize the y-labels and *NA_fixer()* to delete and fix the rows and columns by our requirement. Then, we create a Pipeline to do data transformation all at once. For the rest of the missing values, we impute the median of each respective numerical column and for categorical columns, we impute the most frequent value of the column. Also, we scaled them with Standardized scaler in our pipeline.

Model Building

To make our model more accurate, at the beginning of building models, we separate our data into a training set and testing set. We use the code from sklearn.model_selection import train_test_split to import the function train_test_split. This function helps us to randomly separate the data into 0.8:0.2 (80% of the data would be training data, and 20% of the data would be test data).After we have all our preparation done, we define a function called *class_model_test()*. In this way, we could easily put each model into the function and return its accuracy in a dataframe.

The code is shown as below:

```
#Test Model

def class_model_test(model):
    model.fit(X_train_complete,y_train)
    y_pred = model.predict(X_test_complete)
    prfs =  precision_recall_fscore_support(y_test, y_pred, average = 'micro')
    df = pd.DataFrame(list(precision_recall_fscore_support(y_test, y_pred, average = 'micro')),
                      index = ['Precision', 'Recall', 'F Score', 'Support'])
    return df
    #return prfs
```

With this function, we have tried 13 models with our data. And the accuracy of each model is shown as below:

	Model	F Score
0	SVC	0.715105
1	Logit	0.707935
2	Decision Tree	0.624283
3	Random Forest	0.689293
4	MLP	0.708891
5	Decision Tree Classifier	0.623327
6	KNN	0.676386
7	Ada Boost	0.710325
8	Gaussian NB	0.636711
9	Quadratic Discrimanant Analysis	0.618547
10	Gradient Boosting	0.715583
11	Bagging	0.673518
12	Extra Trees	0.686902

Figure1: model comparison

From the graph, it is obvious that the ***Gradient Boosting*** model has the highest F-score and we can conclude that it is the best model for our data among the models that we have tried. We also choose ***SVC*** and ***Logistic Regression*** as comparison models to show in our interface, as they also have relatively good F-score.

After choosing three models, we used GridSearch to tune the hyper-parameters separately for those three models. The code is as below(take SVC as example):

```
#Hyper Tune Models That Perform the Best

from sklearn.model_selection import GridSearchCV

# SVC = SVC()
# params = {'C': [1,5], 'gamma': [0.5, 0.1, .001], 'kernel': ['rbf','poly']} #'degree': [2,3,4]}
```

```
grid_search = GridSearchCV(SVC, params, cv = 5)
grid_search.fit(X_train_complete, y_train)
grid_search.best_estimator_
```

```
best_SVC = SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
               decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
               max_iter=-1, probability=False, random_state=None, shrinking=True,
               tol=0.001, verbose=False)
```

Finally, we save the three models separately with the pipeline in files: *Boost_Pipeline.sav*, *SVC_Pipeline.sav*, and *Logit_Pipeline.sav* for later use in building the interface.

Interface(Streamlit)

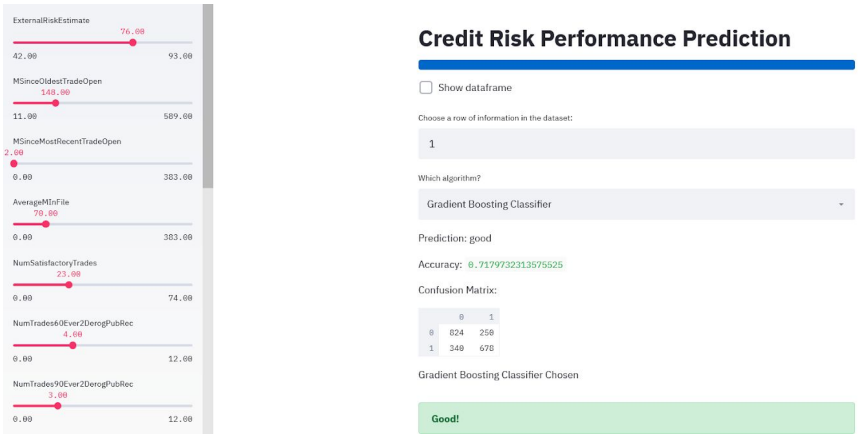

Figure2: Interface of the predictive model

Figure 2 shows how the interface of the predictive model looks. Here are the descriptions of the function of each part of the interface and how we realize them with streamlit.

Main part(right) :

- Show dataframe: The test data can be shown on the screen. (Figure)

	ExternalRiskEstimate	MSinceOldestTradeOpen	MSinceMostRecentTradeO…	AverageMInFile	NumSatisfactoryTrades	NumTrades60Ever2DerogP…	NumTrades90
0	68	15	13	14	3	0	
1	76	148	2	70	23	4	
2	64	165	1	71	15	1	
3	68	197	21	88	8	1	
4	57	184	25	80	11	4	
5	61	161	23	80	6	1	

Figure3: X_test data

- Choose one row of information in the dataset: Our user could choose one row in the test dataset by inputting the row number to check the prediction outcome.
- Which algorithm: Here we provide three best algorithms that we trained, which is Gradient Boosting Classifier, SVC, and Logistic Regression. Our user could choose as they like and check the prediction outcome, accuracy, and confusion matrix below.

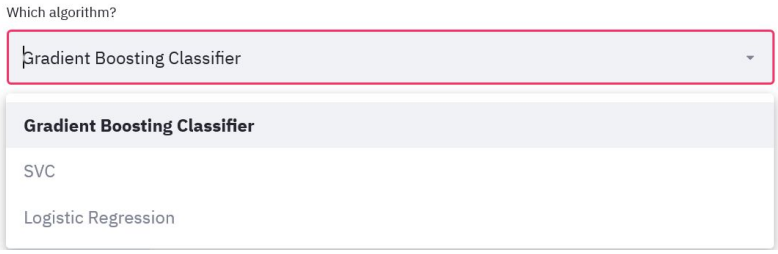

Figure4: Model selection

- Prediction, Accuracy, Confusion Matrix: The models and pipelines were saved as *.sav* files previously, so we load them first. Once our users choose the input data and the algorithm they would like to use, the model will make predictions and the result will show up automatically on the screen.
  The result includes: 1) The prediction of the risk: Good or Bad; 2) The prediction accuracy; 3) The confusion matrix. To make our interface more user-friendly, we use the code below to clearly show the results to our users.

```
if classifier == 'Gradient Boosting Classifier':
    # different trained models should be saved in pipe with the help pickle

    pipe = pickle.load(open('Boost_Pipeline.sav', 'rb'))

    arr = np.array([a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r]).reshape(1,18)
    df = pd.DataFrame(arr, columns = X_test.columns)

    res = pipe.predict(df)
    st.write('Prediction:  ', dic[int(res)])
    pred = pipe.predict(X_test)
    score = pipe.score(X_test, y_test)
    cm = metrics.confusion_matrix(y_test, pred)
    st.write('Accuracy: ', score)
    st.write('Confusion Matrix: ', cm)
    st.markdown('Gradient Boosting Classifier Chosen')
```

Figure5: code used for Gradient Boosting Classifier Predictor

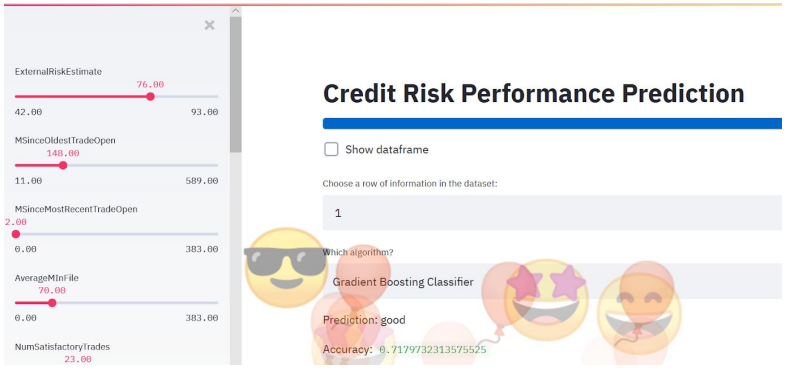

Figure6: example of predicting good

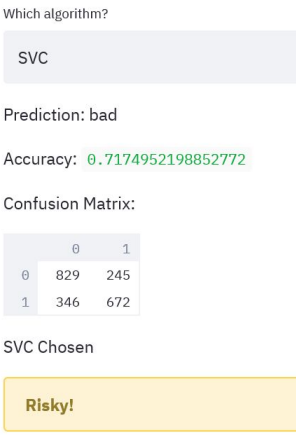

Figure7: example of predicting bad

- <u>Left control panel:</u>
  In order to make our interface more flexible and customizable, we set a sidebar for our users to adjust the value for prediction. Our user could drag the sidebar to any value in the range to build up a new customer profile, then our model would predict the risk and show the result as mentioned before.

```
if res == 1:
    st.spinner('Wait for it...')
    st.balloons()
    st.success('**Good!**')
else:
    st.spinner('Wait for it...')
    st.warning('**Risky!**')
```

Figure8: Sidebar

Summary

This report briefly shows our vision of a machine learning model. At the very beginning, we do clean the data to make sure the information used is meaningful. Then, with the cleaned data, we test and tune several models, selecting only the most effective. Last but not least, we input the models into Streamlit and cover them with an interface, which makes our model easier to use when predicting good or bad credit performance. The project helps us understand the applications of machine learning in the real world, and we look forward to working on more cases with what we have learned during this project.