Predictive Analytics/ Python
Instructor: Yaron Shaposhnik
December 16,2019

Team 5:
Xinyu Huang
Fanglei Su
Guilherme Plentz De Li
Yiqun Zhao

Project #2 Report

Overview

One important factor financial institutions use to decide whether or not to approve a loan is credit scores. The scores are designed to predict the likelihood of repayment of a loan, and customers can get explanations for their scores.

We have access to a dataset of Home Equity Line of Credit (HELOC) applications made by real homeowners. The customers in this dataset have requested a credit line in the range of $5,000 - $150,000. The dataset contains the information about the applicant in their credit report and their risk performance.

We developed a predict model and a decision support system that evaluates the credit risks of Home Equity Line of Credit applications. This model can be used to predict whether a customer will repay his or her HELOC account within 2 years and help decide whether the homeowner qualifies for a line of credit.

In the model building process, we first clean the data to exclude any missing value. Then we try different models and compare their accuracy to find the best model. We try logistic regression, K nearest neighbor, decision tree, random forest and adaboost. Finally we find that logistic regression model works best. After choosing the best model, we design and develop an interactive interface using the model we build. Sales representatives in the bank or credit card company can use this interface to decide on whether to accept or reject customers' applications.

Data Cleaning

Most of the Machine Learning algorithms cannot work with missing features, or any other kind of “dirty” data. Therefore it is incredibly important to create a few functions to be able to predict the best model. By using the function risk_data.describe() we can superficially see

that there are missing values based on the data-dictionary that was provided. The missing values are the -9, -8, -7.

For example:

Index	cternalRiskEstima	nceOldestTradeO	aMostRecentTrad	AverageMInFile	mSatisfactoryTrac	des60Ever2Derog	des90Ever2Derog	centTradesNeverD	inceMostRecentD	elq2PublicRecLas	MaxDelqEver
count	10459	10459	10459	10459	10459	10459	10459	10459	10459	10459	10459
mean	67.4258	184.205	8.54346	73.8433	19.4281	0.0427383	-0.142843	86.6615	6.76241	4.92829	5.51018
std	21.1216	109.684	13.3017	38.7828	13.0043	2.51391	2.3674	25.9996	20.5013	3.75628	3.97118
min	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9
25%	63	118	3	52	12	0	0	87	-7	4	5
50%	71	178	5	74	19	0	0	96	-7	6	6
75%	79	249.5	11	95	27	1	0	100	14	7	8
max	94	803	383	383	79	19	19	100	83	9	8

We had basically three options:

1. Get rid of the corresponding rows.
2. Get rid of the whole attribute.
3. Set the values to some value (zero, the mean, the median or mode).

Our choice was to first, replace with modes our categorical columns MaxDelq2PublicRecLast12M and MaxDelqEver with their respective modes. And after that we replaced the missing data with NaN to then replace it with the respective mean of every different attribute/ feature. The last step was changing our “Y” which was “Bad/Good” with 1/0 and then round our dataset so we have just integers.

We end up getting:

Index	RiskPerformance	cternalRiskEstima	nceOldestTradeO	aMostRecentTrad	AverageMInFile	mSatisfactoryTrac	des60Ever2Derog	des90Ever2Derog	centTradesNeverD	inceMostRecentD	elq2PublicRecLas
count	10459	10459	10459	10459	10459	10459	10459	10459	10459	10459	10459
mean	0.521943	72.057	200.787	9.61163	78.7906	21.1146	0.604934	0.363132	92.3397	21.9421	5.8278
std	0.499542	9.58541	93.9937	12.5941	33.0946	10.9986	1.20731	0.968959	11.4374	14.4313	1.62303
min	0	33	2	0	4	0	0	0	0	0	0
25%	0	65	140	3	58	13	0	0	90	16	6
50%	1	72	194	6	78	21	0	0	96	22	6
75%	1	79	249.5	11	95	27	1	0	100	22	7
max	1	94	803	383	383	79	19	19	100	83	9

Creating Train/Test Data:

We used the package sklearn.model_selection import train_test_split and split the data into training (model sees and learns from this data - 80% of data) and testing sets (sample of data used to provide an unbiased evaluation of a final model fit on the training dataset - 20% of data).

```
#3.Create the Train Set and Test Set


#For the decision tree model ,we split the data into train set and test set
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(risk_data, test_size=0.2, random_state=40)

train_set.head()
test_set.head()

Y_test = test_set.RiskPerformance
Y_train = train_set.RiskPerformance

test_set = test_set.drop(columns = 'RiskPerformance')
train_set = train_set.drop(columns = 'RiskPerformance')
#For other models, we split data into X_test, y_test, X_train and y_train

Y=np.array(risk_data['RiskPerformance'])
del risk_data['RiskPerformance']
X=np.array(risk_data)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=40)
```

(1)logistic regression

```
#A.logistics regression:
pipe_logistic = Pipeline([('minmax', MinMaxScaler()), ('lr',
                          LogisticRegression(penalty='l1',tol=0.00001,C=100))])
pipe_logistic.fit(X_train, y_train)

print('Accuracy_logisctic: ', pipe_logistic.score(X_test, y_test))
```

We first build a logistic regression model. Since logistic regression works better when features are on relatively similar scale and close to normally distributed, we use min-max scalar to preprocess the data. Then we build the logistics regression and tune the model. We try many different values for several parameters. We finally decide to use 'l1' norm in penalization, 0.00001 as tolerance for stopping and 100 as the inverse of regularization strength to specify a not that strong regularization.

```
Accuracy_logisctic:  0.72131931166348
```

We finally get an accuracy of 0.7213 for the logistic regression model.

(2) K-Nearest Neighbour

We then build a KNN model. For this model, we also use standard scalar to pre-process the data before building the model. After scaling, we first build the KNN model with k=1.

```
#B.KNN:
 #standard scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(1)
knn.fit(X_train_scaled, y_train)

print('Accuracy_knn: ',knn.score(X_test_scaled, y_test))
```

The accuracy for this model when k=1 is almost 0.64. Although we tried to tune the model and change the value of k, the highest accuracy we can reach with this model is 0.6396.

```
Accuracy_knn:  0.6395793499043977
```

(3) AdaBoost

Next, we tried the Adaptive Boosting Classifier. We first build an AdaBoost model based on the data pre-processed by standard scalar, then we tried different values for several parameters to tune the model, for example, the number of estimators and learning rate.

```
#C.AdaBoost:

from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=100,learning_rate=0.7)
abc = abc.fit(X_train_scaled,y_train)

print('Accuracy_adaboost: ',abc.score(X_test_scaled, y_test))
```

We finally decided to use 100 estimators and 0.7 as the learning rate for the AdaBoost model. The accuracy we finally got for this is 0.7180.

```
Accuracy_adaboost:  0.71797323135755525
```

(4) Random Forest

After trying the AdaBoost we tried the Random Forest model. For this model we tried different values for several parameters to tune the model. For example, n-estimators, criterion, max depth, minimum sample split, and max features.

```
#D. Random Forest
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10, criterion='gini',
                             max_depth=9, min_samples_split=8, max_features=5)
rfc.fit(X_train_scaled,y_train)

print('Accuracy: ',rfc.score(X_test_scaled, y_test))
```

After tunning the model, we finally get an accuracy of 0.7161.

```
Accuracy_randomforest:  0.7160611854684512
```

(5) Decision Tree

At last, we tried the Decision Tree Model. For this model, we tried to find out the best depth of decision tree using a for loop which tried different values of depth from 1 to 31, calculated the accuracy score for each depth, and chose the best depth with the highest accuracy score.

```
#E. Decision Tree:
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth =  4)
dtc.fit(train_set, Y_train)

from sklearn.metrics import accuracy_score
prediction = dtc.predict(test_set)
accuracy_score(Y_test, prediction)

bestDepth = pd.DataFrame({'Depth':np.repeat(0.111,31),'Accuracy':np.repeat(0.111,31)})

for d in range(1,31):
    dtc = DecisionTreeClassifier(max_depth=d)
    dtc = dtc.fit(train_set, Y_train)
    prediction = dtc.predict(test_set)
    bestDepth['Depth'][d] = d
    bestDepth['Accuracy'][d] = accuracy_score(Y_test,prediction)
```

After tuned, the decision tree model reached an accuracy of 0.7132.

```
0.7131931166347992
```

Interface

After building different models and comparing the accuracy, we come to the step of building the interface. We use streamlit to build the interface, for its convenience and beauty. With only two lines of codes, we can get access to streamlit and open it in the terminal.

Step 1: Build the title of the interface. In addition, for the purpose of reference of the previous user information, we also leave a piece of space for the dataframe.

```
2 # title
3 st.title('Credit Risk Assessment')
4 # show data
5 if st.checkbox('Show dataframe'):
6     st.write(X_test)
7
```

Step 2: Build the control panel of the interface. In this model ,we have 23 features when predicting the default risk of the applicants. Therefore, we decide to use the sidebar to adjust these parameters, which is easy for bank staff to op

```
# Create four sliders in the sidebar
a = st.sidebar.slider('ExternalRiskEstimate', 0.0,94.0, values[0], 1.0)
b = st.sidebar.slider('MSinceOldestTradeOpen', 0.0,803.0, values[1], 1.0)
c = st.sidebar.slider('MSinceMostRecentTradeOpen', 0.0,383.0, values[2], 1.0)
d= st.sidebar.slider('AverageMInFile', 0.0,383.0, values[3], 1.0)
e= st.sidebar.slider('NumSatisfactoryTrades', 0.0,79.0, values[4], 1.0)
f= st.sidebar.slider('NumTrades60Ever2DerogPubRec', 0.0,19.0, values[5], 1.0)
g= st.sidebar.slider('NumTrades90Ever2DerogPubRec', 0.0,19.0, values[6], 1.0)
h= st.sidebar.slider('PercentTradesNeverDelq', 0.0,100.0, values[7], 1.0)
i= st.sidebar.slider('MSinceMostRecentDel', 0.0,83.0, values[8], 1.0)
j= st.sidebar.slider('MaxDelq2PublicRecLast12M', 0.0,9.0, values[9], 1.0)
k= st.sidebar.slider('MaxDelqEver', 0.0,8.0, values[10], 1.0)
l= st.sidebar.slider('NumTotalTrades', 0.0,104.0, values[11], 1.0)
m= st.sidebar.slider('NumTradesOpeninLast12M', 0.0,19.0, values[12], 1.0)
n= st.sidebar.slider('PercentInstallTrades', 0.0,100.0, values[13], 1.0)
o= st.sidebar.slider('MSinceMostRecentInqexcl7days', 0.0,24.0, values[14], 1.0)
p= st.sidebar.slider('NumInqLast6M', 0.0,66.0, values[15], 1.0)
q= st.sidebar.slider('NumInqLast6Mexcl7days', 0.0,66.0, values[16], 1.0)
r= st.sidebar.slider('NetFractionRevolvingBurden', 0.0,232.0, values[17], 1.0)
s= st.sidebar.slider('NetFractionInstallBurden', 0.0,471.0, values[18], 1.0)
t= st.sidebar.slider('NumRevolvingTradesWBalance', 0.0,32.0, values[19], 1.0)
u= st.sidebar.slider('NumInstallTradesWBalance', 0.0,23.0, values[20], 1.0)
v= st.sidebar.slider('NumBank2NatlTradesWHighUtilization', 0.0,18.0, values[21], 1.0)
w= st.sidebar.slider('PercentTradesWBalance', 0.0,100.0, values[22], 1.0)
```

erate. If they want to change the parameter(such as External Risk Estimate), what they need to do is just drag the bars to the figure they want.

Above codes are for the purpose of building sidebars. While constructing them, what we need to know is the max and min figure of the parameter and leave it as the data of the selected user if no changes are made.

Step 3: Apply the model into the interface. Since the highest accuracy is logistics regression model, we will combine this model into the interface.

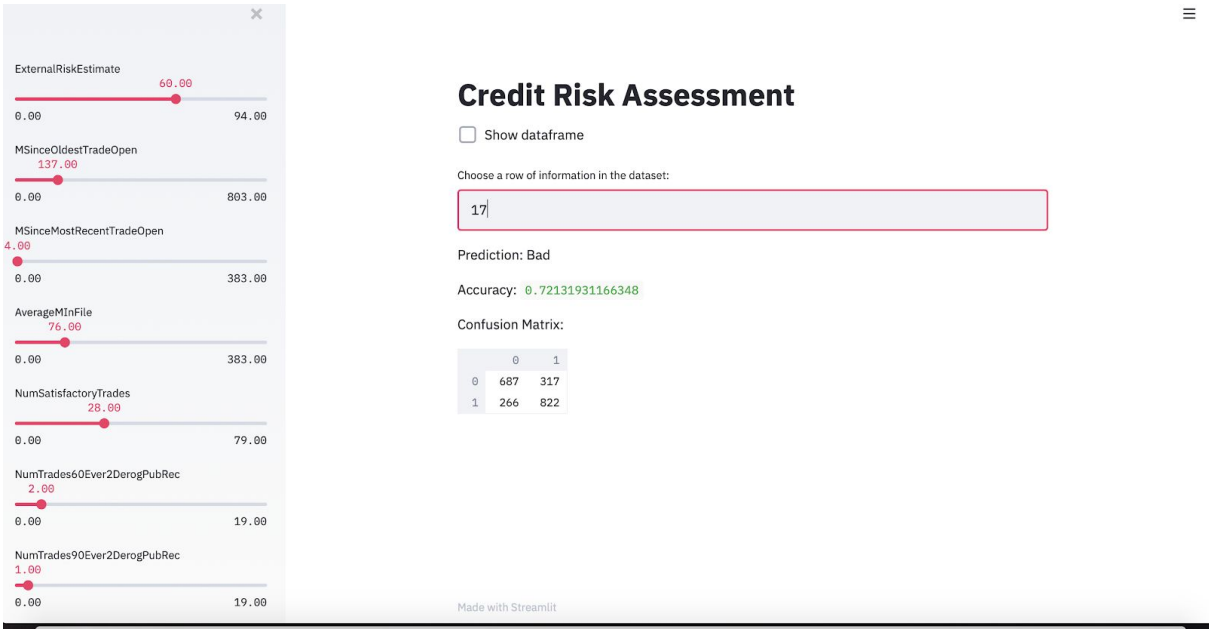
```
    pipe = pickle.load(open('pipe_logistic.sav', 'rb'))
    res = pipe.predict(np.array([a, b, c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w]).reshape(1, -1))[0]
    st.write('Prediction:  ', dic[res])
    pred = pipe.predict(X_test)
    score = pipe.score(X_test, y_test)
    cm = metrics.confusion_matrix(y_test, pred)
    st.write('Accuracy: ', score)
    st.write('Confusion Matrix: ', cm)
```

Step 4: Set an input text frame to get the record of previous guests. Considering the cases that someone wants to analyze the previous guests data, we create an opportunity for them. Following codes are for them to choose the specific number of previous guests who have applied for their credit card and getting their data.

```
# Input the index number
number = st.text_input('Choose a row of information in the dataset:', 17)

test_demo(int(number))  # Run the test function
```

The final interface looks like this:



Sidebars on the left side can be adjusted as you like. Other than the prediction results, you can also see the accuracy of prediction and the confusion matrix. You can choose not to show the data frame if you don't want to retrieve the previous data.

Summary:

After using the dataset from Home Equity Line of Credit (HELOC) applications made by real homeowners, which contains more than 10,000 rows of information about the applicants, their credit report and risk performance. We can come to the conclusion that by using our model we can run predictions, much more accurately than simply doing a "coin toss", that is why our world is going through this analytic revolution where companies want to be data driven. It is way more efficient for a company to make a decision by using our model (which will be correct more than 70% of the time) than just by people trying to guess what is right or wrong.

We developed the model and a decision support system (interface), which makes it easier even for people who don't have the technical skills to run code. They can simply understand the logic behind using a model (data, tuning, choosing a model) and make the best choice based on the accuracy.

Our biggest lesson is that for models we need to try out many different ones we can, and tune them in as many ways we can. Some models predict best under certain situations while others don't, therefore, the best thing we can do is run as many options we can until we get the best accuracy.