MICRO CREDIT PROJECT

Submitted by:

Deepro Sengupta

ACKNOWLEDGMENT

This project would not have come to fruition without the kind guidance of our senior/mentor Mr. Sajid Choudhary at FlipRobo Technologies. His help and input have been invaluable to this project. Secondly, I would like to extend my gratitude to all mentors and teaches at DataTrained for training me to be a good data scientist. Lastly, I would like to thank my parents for their support.

INTRODUCTION

Micro Credits loans play an invaluable role in empowering people financially. This is particularly true for people from financially challenged background. However, extending micro credit loans can have its own challenges. Since the beneficiaries are usually from weak financial section of the society, there is a high probability that the loan might not be paid back and therefore, it becomes imperative to identify potential loan defaulters before loans are extended to them. This project tries to address the issue of identifying potential loan defaulters and calculating the probability of loan payback.

In case of this project, our client is an Indonesia based telecommunications firm looking to extend micro credit loans on talk-time in form of IDR 5 & 10 for which an amount of IDR 7 & 12 respectively is to be paid back. The aim of this project is predicting the probability of loan getting paid back within 5 days of loan issuance.

Analytical Problem Framing

The problem we have in this project is a supervised classification problem as the data provided has our target variable which we can use the first find train our algorithm and then use the trained model to predict the probability values.

The data for this project was provided directly by the client and contains various attributes which relates to their clientele. Initially, before any data pre-processing, the dataset had 37 columns and 2,09,593 rows.

The original data formats of the columns are were either int64 or float. The columns 'msisdn', 'pcircle' and 'pdate', however, were of the type 'object'.

In order to prepare the data for the project, some data pre-processing was needed. A quick check on the data revealed that there were no missing values in the dataset. There were also no duplicate rows in the data. The columns 'msisdn', 'pcircle' and 'pdate' which were of the type 'object' were dropped as it 'msisdn' was the mobile number of the customer while 'pcircle' was just one value all throughout the 2,00,000+ rows and 'pdate' was just dates. These columns, therefore, were not contributing significant information in the analysis.

There were several columns with huge outliers. This was evident on checking the summary statistics of the where the mean was substantially higher than the 50^{th} percentile value and the difference between the 75^{th} percentile and the maximum value was huge. The presence of outliers was further confirmed by the boxplots as well. The outliers were removed using the IQR-method where any datapoints outside 1.5 times outside the inter-quartile range (IQR) of the column is replaced with the values of 'upperBridge'(which is 75^{th} percentile value + 1.5 time the IQR) for values higher than the 'upperBridge' and values of the 'lowerBridge' (which is 25^{th} percentile value - 1.5 time the IQR) for values lower than the 'lowerBridge'.

There was skewness present in the data. However, removing them was no necessary as we are not using linear or logistic regression which assumes

that the independent variables have gaussian distribution. Classification algorithms are not sensitive to skewness.

The target variable, label, had an imbalanced distribution as the value '0' was just 12.5% of the total labels while the value '1' was 87.5% of the labels. In order to tackle this, oversampling was used to balance the dataset before model training.

This project uses the Anaconda Distribution of Python and Jupyter Notebook for analysis. The system requirements are:

- License: Free use and redistribution under the terms of the ../eula.
- Operating system: Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
- Minimum 5 GB disk space to download and install.

The following packages were used in this analysis:

1. NumPy: For basic mathematical operations
2. Pandas: For DataFrame manipulation
3. Matplotlib & Seaborn: For data visualization
4. Scikit-Learn: For data pre-processing, data modelling and model evaluation.

Model/s Development and Evaluation

The following classification algorithms were used in this analysis:

1. RandomForestClassifier
2. Support Vector Classifier
3. K-nearest Neighbor Classifier

Firstly, the best random-state for the dataset was calculated by looping through random-state from 1 to 100. The code for it is as below:

In [24]:
```
#Finding the best random state
maxAcc = 0
maxRs = 0

for randState in range(0,100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=randState)
    rfc = RandomForestClassifier()
    rfc.fit(x_train, y_train)
    predrfc = rfc.predict(x_test)
    acc = accuracy_score(y_test, predrfc)
    if acc>maxAcc:
        maxAcc=acc
        maxRs=randState
print(f"Best accuracy is {maxAcc*100}% on Random state {randState}")
```

```
Best accuracy is 91.31651770094469% on Random state 99
```

The best random-state was 99 with accuracy score of 91.31%.

In [29]:
```
rfc = RandomForestClassifier()
svc = SVC()
knn = KNeighborsClassifier()

model = [rfc, knn, svc]

for m in model:
    m.fit(x_trainOS, y_trainOS)
    m.score(x_trainOS, y_trainOS)
    predm = m.predict(x_test)
    print(f'Accuracy score of {m}:\n')
    print(accuracy_score(y_test, predm))
    print(confusion_matrix(y_test, predm))
    print(classification_report(y_test, predm))
    print(f1_score(y_test, predm))
    print('\n')
```

The snapshot above shows the coding done to implement the algorithms used in this analysis. Line 1 through 3 initialises the object for the classifier algorithms. The list ‘model’ contains all the classifier objects and is used in the for-loop below to fit and print the evaluation metrics of each of the algorithms one-by-one.

Once the three classification algorithms were trained, their f1-scores were used to measure their performance. The reason f1-scores were chosen as evaluation metrics is because it is a harmonic mean of ‘Precision’ and ‘Recall’ and therefore presents a better insight on the model’s performance. The snapshots below summarise the results for each of the algorithms:

```
Accuracy score of RandomForestClassifier():

0.9072648621139349
[[ 3896  3937]
 [ 1894 53151]]
              precision    recall  f1-score   support

           0       0.67      0.50      0.57      7833
           1       0.93      0.97      0.95     55045

    accuracy                           0.91     62878
   macro avg       0.80      0.73      0.76     62878
weighted avg       0.90      0.91      0.90     62878

0.9479992508895686
```

```
Accuracy score of KNeighborsClassifier():

0.7870956455357995
[[ 5027  2806]
 [10581 44464]]
              precision    recall  f1-score   support

           0       0.32      0.64      0.43      7833
           1       0.94      0.81      0.87     55045

    accuracy                           0.79     62878
   macro avg       0.63      0.72      0.65     62878
weighted avg       0.86      0.79      0.81     62878

0.8691589698480183
```

```
Accuracy score of SVC():

0.7192181685168103
[[ 6314  1519]
 [16136 38909]]
              precision    recall  f1-score   support

           0       0.28      0.81      0.42      7833
           1       0.96      0.71      0.82     55045

    accuracy                           0.72     62878
   macro avg       0.62      0.76      0.62     62878
weighted avg       0.88      0.72      0.77     62878

0.8150786086118588
```

The last score in each of the snapshots are the respective f1-scores of each of the algorithms. We can see that Random Forest Classifier has the highest f1-score. However, this might also be due to overfitting and so the cross-validation scores and difference between the actual f1-score and cross-validation f1-scores of each of the algorithms were calculated. They are given in the snapshot below:

```
In [37]:
print(f'Difference between Accuracy and Cross Validation Score for RandomForestClassifier is {(srcRfc.mean()*100)-(0.9479992
print(f'Difference between Accuracy and Cross Validation Score for KNN is {(srcKnn.mean()*100)-(0.8691589698480183*100)}')
print(f'Difference between Accuracy and Cross Validation Score for SVC is {(srcSvc.mean()*100)-(0.8150786086118588*100)}')

Difference between Accuracy and Cross Validation Score for RandomForestClassifier is 3.0623779542091825
Difference between Accuracy and Cross Validation Score for KNN is -0.4282586894100433
Difference between Accuracy and Cross Validation Score for SVC is -5.397419750315194
```

Based on the above score difference, the RandomForestClassifier was chosen as the final model as the other algorithm score difference was negative indicating that these models performance dropped after cross-validation.

Finally, hyperparameter tuning was done on the RandomForestClassifier with the code below to further improve the f1-score:

```
In [81]:
rscv = RandomizedSearchCV(rfc, param_distributions = parameters, n_jobs = -1, cv = 5)

In [82]:
rscv.fit(x_trainOS, y_trainOS)

Out[82]:
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'criterion': ['entropy'],
                                        'max_features': ['sqrt', 'log2'],
                                        'n_estimators': [10, 150, 200, 500,
                                                         1000]})

In [84]:
rscv.best_params_

Out[84]:
{'n_estimators': 1000, 'max_features': 'sqrt', 'criterion': 'entropy'}

In [31]:
rfc1 = RandomForestClassifier(n_estimators = 1000, max_features = 'sqrt', criterion = 'entropy')
rfc1.fit(x_trainOS, y_trainOS)

Out[31]:
RandomForestClassifier(criterion='entropy', max_features='sqrt',
                       n_estimators=1000)
```

The final deployment-ready model was trained with the optimal parameters found using RandomizedSearchCV and the final f1-score was 0.95(approx.) which was a slight improvement from the initial model.

The final probabilities are calculated using the predict_proba function as given in the code below:

```
In [35]:
predRfc1 = rfc1.predict_proba(x_test)

In [36]:
predRfc1

Out[36]:
array([[0.028, 0.972],
       [0.   , 1.   ],
       [0.018, 0.982],
       ...,
       [0.468, 0.532],
       [0.003, 0.997],
       [0.986, 0.014]])
```

The final probability calculations are stored in the array ‘predRfc1’ which can be saved to MS Excel if needed.

CONCLUSION

From the probability calculations, we see that the probability of the borrower of paying back the loan ranges vastly with around 2% in the lower side to as high as 98% on the higher side of the spectrum.

However, this study suffers from a few limitations. Firstly, the vast size of the data makes it very difficult for model training as each algorithm takes a huge amount of time to train. In this case, a laptop with 9^{th}-Generation i7 processor with 6-cores and 16GBs of RAM struggled greatly to train the models with the fast algorithm taking 2 hours to train and the slowest one taking as much as 5 hours to train. Therefore, due to hardware limitations, only three models could be tested. More accurate results may be obtained by trying other classification algorithms on a more powerful system or on cloud-based computation environment like Google Colab.