EAI 6000: Fundamentals of Artificial Intelligence Final Project

Credit Card Approval Prediction

Emily (Zhaohui) Fan, Amber Moon, Hansol Kim

Northeastern University
Professor Kasun Samarasinghe
December 7, 2020

I. INTRODUCTION

A credit card is a convenient financial product that can be used for everyday purchases including gas, groceries, and other goods and services; big-ticket items such as TVs, travel packages, and jewelry because the funds for these items are not always immediately at our disposal; benefits such as sign-up bonus, cash back, reward points; and most importantly, building credit history. Many people apply for credit cards, but not everyone gets approval. Credit card companies might use applicants' financial details including credit history, income, credit score, monthly housing costs or ownership of realty to decide whether to approve or not. A denied credit card application does not hurt, but the hard inquiry from a submission can cause a decrease in the applicant's credit score, which is one of the determination factors for credit card company to decide whether to approve or disapprove.

In this research we targeted and predicted the credit card approval status of applicants for analysis. We used two datasets from Kaggle website that sharing the common IDs (primary key). To automate the process, the pre-processing module makes the data ready before processing and the modeling module is equipped with the number of classification modeling techniques for detailed analysis.

II. EXPLANATORY DATA ANALYSIS

In this section we provided an understanding of the data in the context of exploration and visualizations. Details about variables categories, how we manipulated the datasets will be covered in *III. Feature Engineering* section.

A. Credit status dataset

It contained 1,048,576 rows and 3 columns. There are 45,985 unique IDs, duplicated keys exist

because the credit status attached to these IDs varied each month.

B. Application dataset

It contained 438,558 rows and 18 columns. 438,510 unique IDs in applications dataset. Figure 1 showed income for different occupation types.

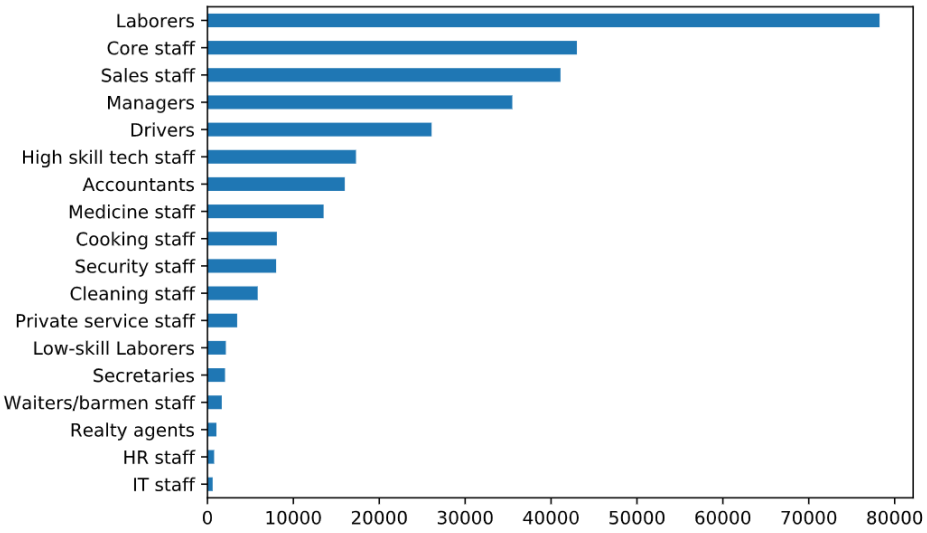

Figure 1. Total income VS occupation types

C. Complete dataset

Combine above two datasets, there are 36,457 unique IDs. "dep_value" means at any point in time whether someone had a balance 60+ days overdue, marked as risky. (*Feature Engineering*)

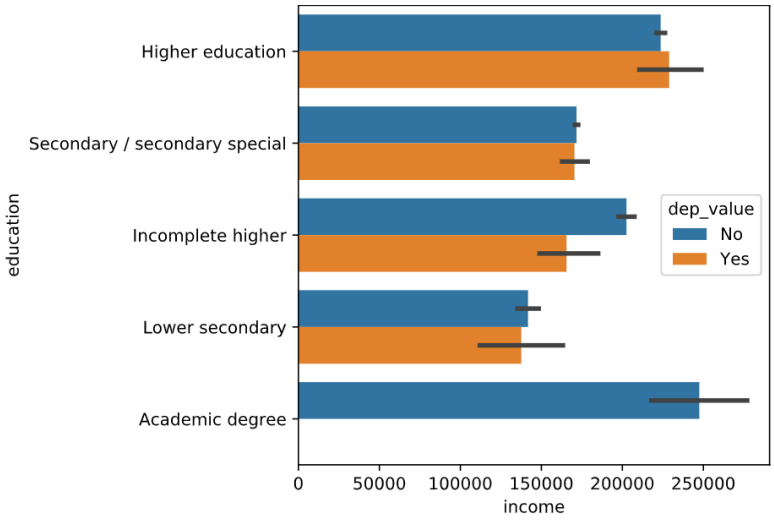

Figure 2. Income VS education levels and credit status

Figure 2 explored how total annual income relates to different education levels and credit status. Total annual income is significantly related to education levels, the higher the education level is, the higher the income is. **Higher education related to more risks than other levels.** People who a lower secondary degree had the least income and less 60+ days overdue than other degrees.

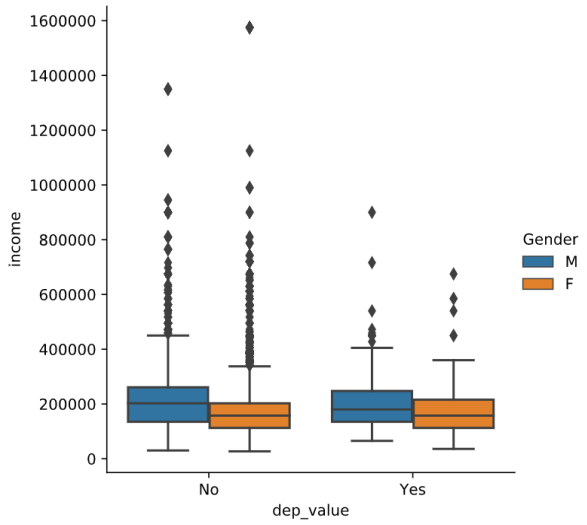

Figure 3. Income VS credit status and genders

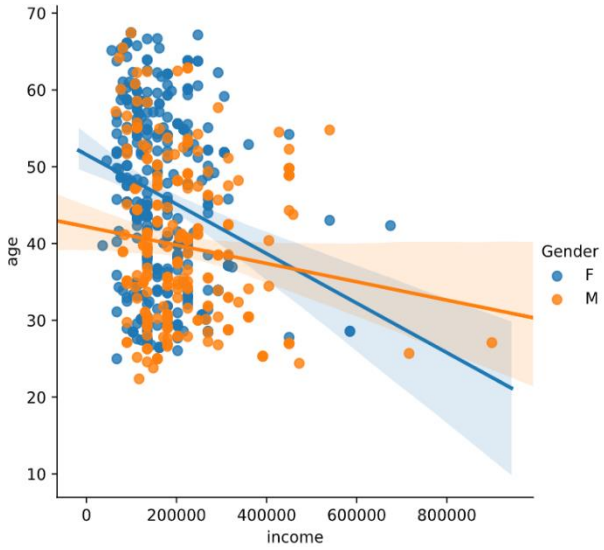

Figure 4. Income VS age and gender for risky people

Figure 3 revealed for both males and females, higher income associated with less risky but the income gaps in genders were more apparent when they had a risky record than not.

Digging into people who had a 60+ days overdue balance at any point in time, Figure 5 looked for relationships within all variables, Figure 4 indicated how income associated with ages and genders.

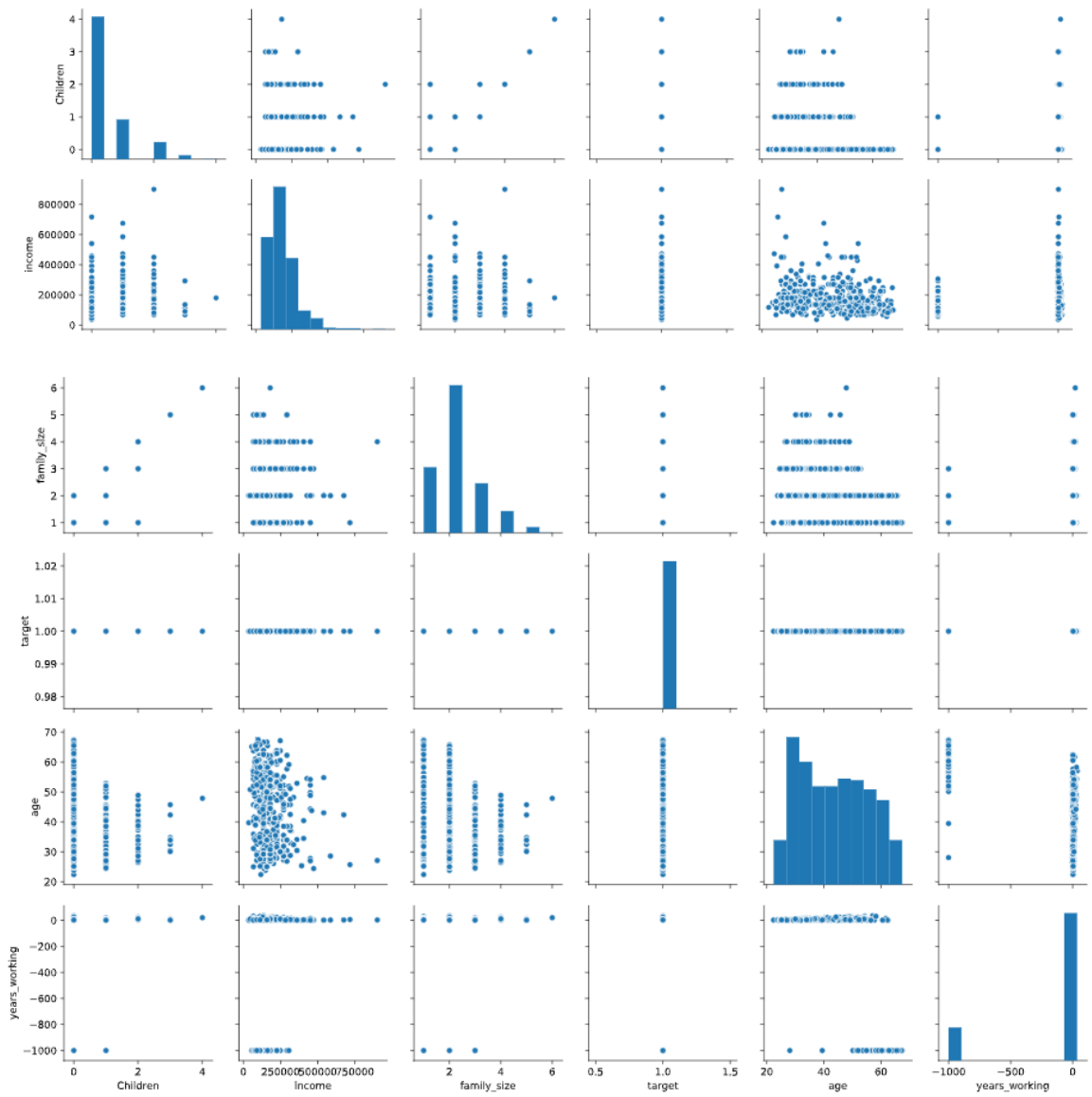

Figure 5. Joint distribution for high risky people

III. FEATURE ENGINEERING

The task here is to prepare all dataset needed from raw data dataset, before various modeling techniques processing to predict the credit card approval.

A. Feature selection

We had identified 12 features plus 1 target value for each user: Gender, Car, Reality, Children, income, income type, education, housing, family size, credit status, age, years of working, target.

B. Numerical Binning

We did numerical binning on credit status, income, children, family size, age, years of working to make the model more robust and prevent overfitting.

For credit status feature(dep_value), Status 0: 1-29 days overdue; Status 1: 30-59 days past due; Status 2: 60-89 days overdue; Status 3: 90-119 days overdue; Status 4: 120-149 days overdue; Status 5: Overdue or bad debts, write-offs for more than 150 days; Status C: paid off that month; Status X: No loan for the month. We marked credit status of 2, 3, 4 and 5 as risky and other status as not risky. This part was used in the EDA section.

C. One-hot encoding

We did one-hot encoding on features of income type, education, housing, income, children, family size, age and years of working. Using *get_dummies* function to maps all values in a column to multiple flag columns and assigns binary values. These binary values express the relationship between grouped and encoded columns.

D. Imbalanced Data Handling

We first checked if the data was imbalanced and found only 1.69% of the applicants were target users. Stratification is needed to maintain the same ratio across training and testing datasets.

```
# Checking if the data is imbalanced or not
sum(y)/ len(y)
# 1.69% of the applicants are target users. We need to make sure we maintain the same % across training and testing datasets.
# It's called "stratification": split the data to maintain the ratio.
```

```
0.016896617933455853
```

Some popular techniques to handle imbalanced data are resample the training set, K-fold Cross-Validation, ensemble different resampled datasets, resample with different ratios and cluster the abundant class. We used *Synthetic Minority Over-Sampling Technique (SMOTE)* to overcome sample imbalance problem and introduce stratification to split the data to maintain the ratio.

E. Split Dataset

We split 70% of the dataset into training sets and 30% as test sets.

```
# Using Synthetic Minority Over-Sampling Technique(SMOTE) to overcome sample imbalance problem.
X_balance,Y_balance = SMOTE().fit_sample(X,y)
X_balance = pd.DataFrame(X_balance, columns = X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_balance,Y_balance,
                                                    stratify=Y_balance, test_size=0.3,
                                                    random_state = 1024)
```

IV. MODELING TECHNIQUES

In this section we discussed various predictive modeling techniques and set performance metrics.

A. Models

1) Logistic Regression

Logistic regression is a classification algorithm used to find the probability of success or failure of certain event. It describes the relationship between one dependent variable and one or more independent variables. Logistic regression is easy to implement, interpret, and efficient to train. It performs well when the data set is linearly separable and can easily extend to multiple classes and a probabilistic view of class predictions.

2) Decision Tree

Decision Tree is a type of supervised machine learning where the data is continuously split according to a certain parameter. It shows the process of decision and easy to interpret and understand the outcome with an abridged description. Decision Tree works well with categorical variables without encoding. It is also useful handling missing values in training set or testing set as it categorizes missing values as one category.

2-a) Pruning

Pruning is a data compression technique in machine learning and search algorithms that reduces the size of decision trees by removing sections of the tree that are non-critical and redundant to classify instances. To improve bias-variance trade-off, we will compare different alphas against

accuracy for both the training and testing datasets to determine how much the Decision Tree should be pruned. Note that the maximum value for alpha will be omitted to avoid over-pruning.

3) Random Forest

Random Forest can be used for both regression and classification. It improves the accuracy of Decision Tree by building varieties of Decision Trees and passing-through data into each tree. When each tree classifies data and produces outcome, Random Forest chooses the final classification outcome from each tree's most voted outcome.

4) GBM

Gradient Boosting Machine (GBM) is a tree-based machine learning technique for regression and classification. It generates prediction models like other tree-based models but is affiliated with boosting while Random Forest uses bagging. GBM is one of the most accurate algorithms and it works with both numerical data and categorical data. GBM can optimize on different loss functions and provide users with several hyper parameter tuning options. We used lightGBM.

5) XGBoost

XGBoost is also a tree-based ML algorithm using a gradient boosting system. It is known for great performance compared to other machine learning algorithms. Instead of finding the optimal number of tress using K-Cross Validation, we used early stopping to stop the tree when the cost function no longer reduces. XGBoost will do the cross validation so we just need to specify the number of rounds to exhaust with no improvement before stopping.

6) Neural Network

Neural Networks are composed of simple elements, called neurons, each of which can make simple mathematical decisions. Together, the neurons can analyze complex problems, emulate almost any function including very complex ones, and provide accurate answers. A shallow

neural network has three layers of neurons: an input layer, a hidden layer, and an output layer. Neural Networks work effectively on nonlinear problems and self-learnings. It can produce outcomes beyond the input data. Unlike other Machine Learning algorithms, the consultation cost of Neural Networks is less because it analyzes through learning.

B. Performance Metrics

1) Accuracy score

Accuracy score is an important parameter for evaluating classification models. Accuracy is the number of correct predictions / total number of predictions.

2) Precision

Precision, also known as AUC, i.e., area under the receiver-operator curve, we considered as positive the predicted candidate receiving the credit card approval. It is represented as:

$$\text{Precision} = (TP) / (TP+FP)$$

where, TP is the number of true positives, and FP is the number of false positives.

3) Recall

$$\text{Recall} = (TP) / (TP+FN)$$

where, TP is the number of true positives, and FN is the number of false negatives.

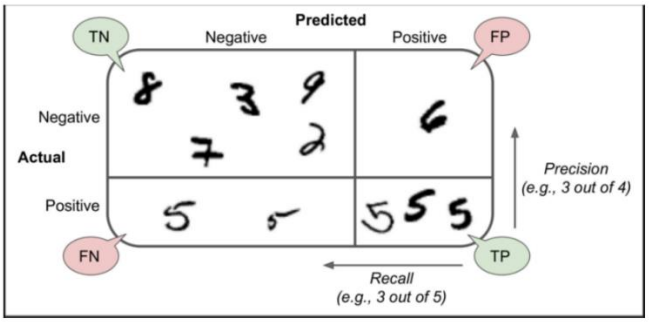

Figure 6. Performance metrics explanation

7) Run Time

It is the duration of the work performed describing the efficiency of the model. This measure includes the time to train the regressor and to evaluate the test cases.

8) Confusion matrix

It is a table describing the performance, errors and types of a classifier. The number of correct and incorrect predictions are summarized with count values and broken down by each class.

V. EXPERIMENT SETTINGS

For our experiment, we merged the credit status dataset and application datasets into a data frame and conducted feature engineering including numerical binning and one-hot encoding. We used Synthetic Minority Over-sampling Technique to overcome sample imbalance problem. Then we divided the cleaned data into two subsets using training data (70%) and testing data (30%, 21,505). The models that are used for experiments are Logistic Regression, Random Forest, Decision Trees, Gradient Boosting Machine (GBM), XGBoost and a simple Neural Network. Pruning for random forest is used to optimize the model. We used Python 3 and Jupyter Notebook implementations of these regressors.

In this section we only discuss some of the models during the training process. Performance measurements of all models will be covered in the *Result and Discussion* section.

1) Decision Tree

We used Decision Trees to find out patterns for classification and regression stored in Figure 6. A clearer version could be found in Appendix.

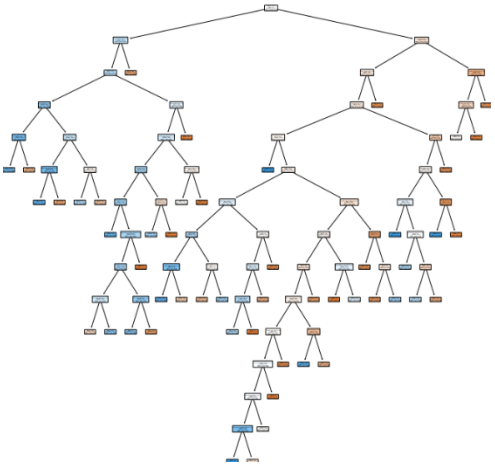

Figure 7. Decision Tree generated

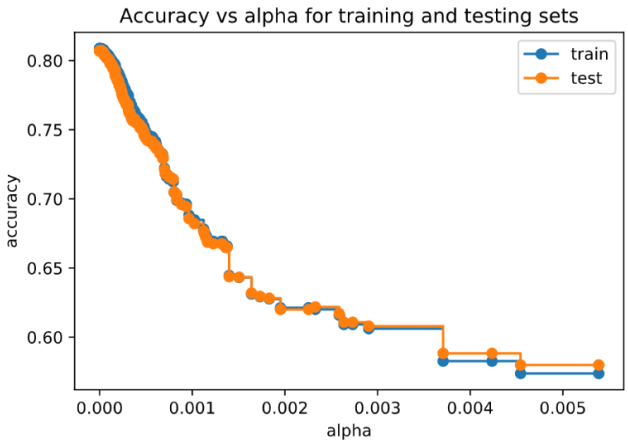

Figure 8. Pruning

2) Pruning for Decision Tree

Minimal cost complexity pruning recursively finds the node with the "weakest link" through an effective *alpha*, where the nodes with the smallest effective *alpha* are pruned first. We used cross validation to find the effective alphas at each step of the pruning process.

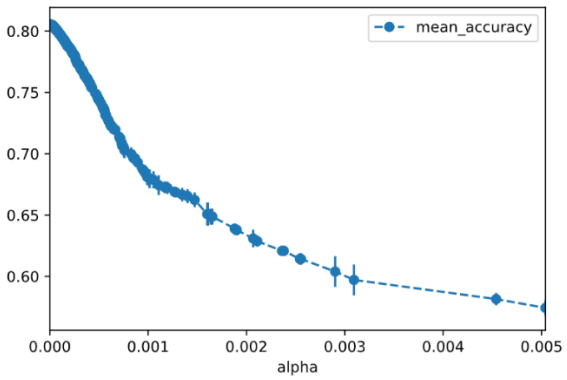

Figure 9. Mean accuracy VS effective alphas

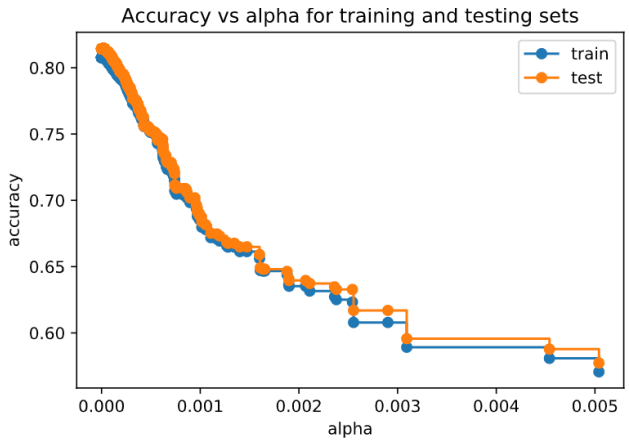

Figure 10. Accuracy vs alpha for training and testing sets

As alpha increases, more of the tree is pruned, which increases the total impurity of its leaves. The ideal value for alpha we found was 0.01 for this study.

3) GBM

Based on the GBM model we ranked the feature importance below:

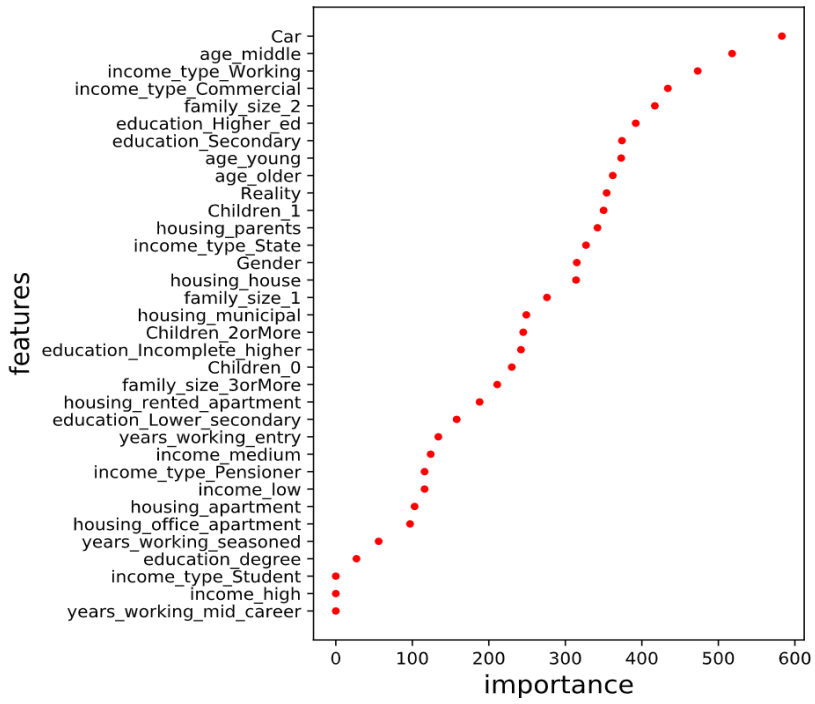

Figure 11. Feature importance ranking by GBM

4) XGBoost

We use evaluation metric as Area Under Precision-Recall Curve, XGBoost will train until validation_0-aucpr did not improve in 10 rounds. During the training, after building trees for 99 rounds, the model had no improvement before stopping. The base score is 0.71323 and the final score is 0.91073. Based on XGBoost model, we ranked the feature importance in Figure 11:

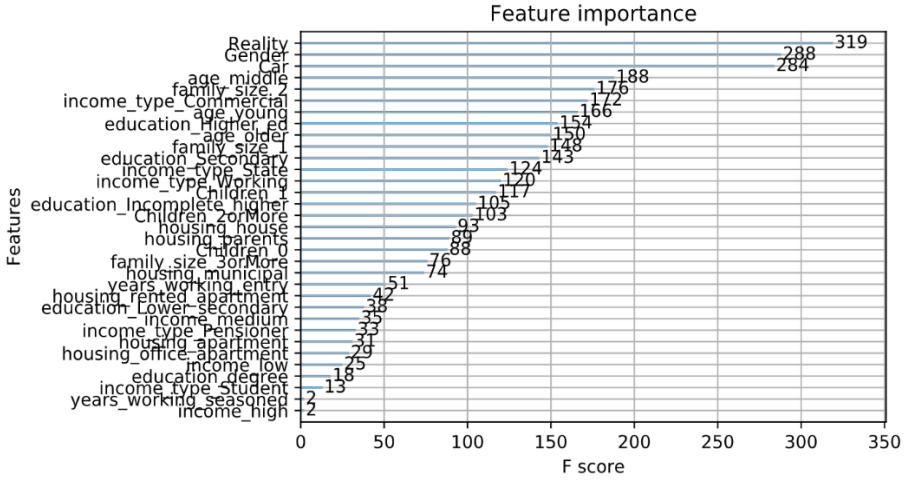

Figure 12. Feature importance ranking by XGBoost

5) Neural Network

We used TensorFlow Keras to build the NN model, and configured the training procedure using *compile()*. 10 epochs, 10% split as validation set and *Adam* optimizer was used. During the training, Loss, binary accuracy, validation loss and validation binary accuracy was calculated during the training. The model had no improvement before stopping after 1412 rounds.

```
Epoch 7/10
1412/1412 [==============================] - 2s 1ms/step - loss: 0.3735 - binary_accuracy: 0.8019 - v
al_loss: 0.3664 - val_binary_accuracy: 0.8091
Epoch 8/10
1412/1412 [==============================] - 2s 1ms/step - loss: 0.3705 - binary_accuracy: 0.8028 - v
al_loss: 0.3697 - val_binary_accuracy: 0.8035
Epoch 9/10
1412/1412 [==============================] - 2s 1ms/step - loss: 0.3682 - binary_accuracy: 0.8039 - v
al_loss: 0.3641 - val_binary_accuracy: 0.8061
Epoch 10/10
1412/1412 [==============================] - 2s 1ms/step - loss: 0.3675 - binary_accuracy: 0.8045 - v
al_loss: 0.3667 - val_binary_accuracy: 0.7995
Neural network Accuracy Score is 0.79842
```

VI. RESULT AND DISCUSSION

The experimentation had been performed on variety of classification models.

A. Models performance Comparison

	Logistic Regression	Decision Tree	Decision Tree -- Pruning	Random Forest	GBM	XGBoost	Neural Network
Run time	N/A	375 ms	288 ms	9.93 s	1.3 s	5.56 s	10 s
Accuracy	0.66682	0.66682	0.81428	0.79233	0.78614	0.81009	0.79842
Precision	N/A	0.71975	0.77265	0.76856	0.7724	0.77099	
Recall	N/A	0.54645	0.89064	0.8366	0.8144	0.88227	

Table 1. Models performance comparison

B. Confusion Matrix

1) Decision Tree

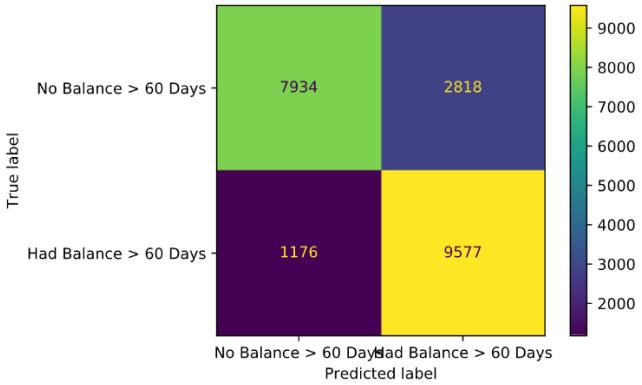

Figure 13. Confusion Matrix of Decision Tree

Using Decision Tree, 7,934 + 2818 = 10,752 people that did not have an overdue balance > 60 days, 7,934 (74%) were correctly classified. And of the 1,176 + 9,577 = 10,753 people that did have an overdue balance > 60 days, 9,577 (89%) were correctly classified.

1-A) Pruning

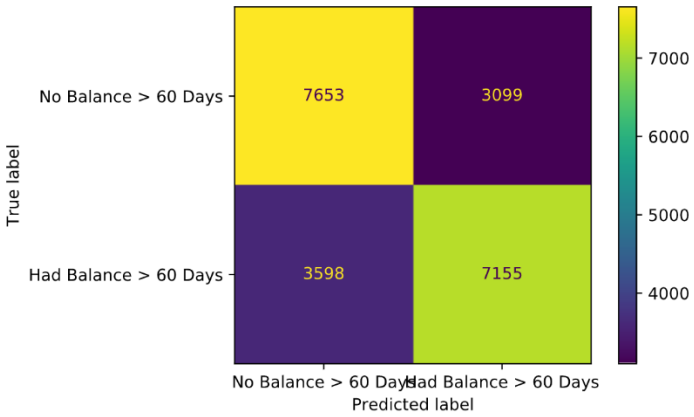

Figure 14. Confusion Matrix of Pruned Decision Tree

Using Pruning Decision Tree, 7,653 + 3,099 = 10,752 people that did not have an overdue balance > 60 days, 7,653 (71%) were correctly classified. And of the 3,598 + 7,155 = 10,753 people that did have an overdue balance > 60 days, 7,155 (67%) were correctly classified.

2) *Random Forest* (see below)

3) GBM

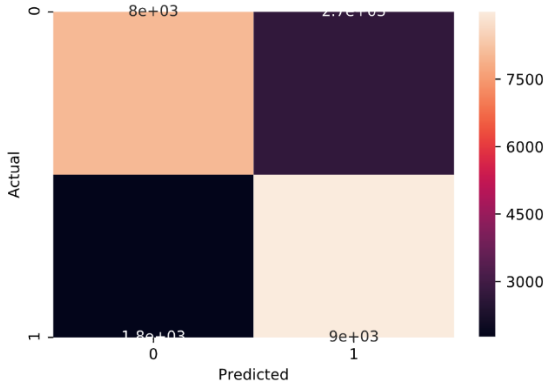

Figure 15. Heatmap of Random Forest

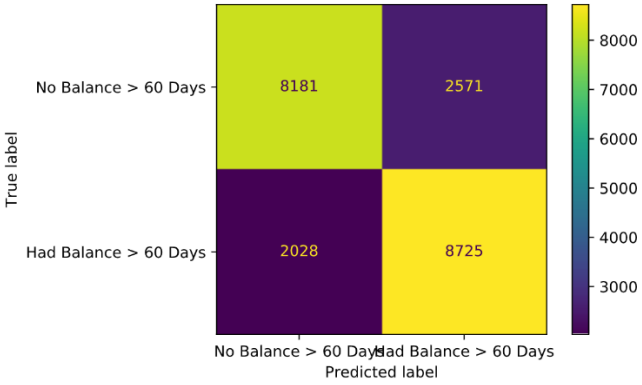

Figure 16. Confusion Matrix of GBM

Using Gradient Boosting, 8,181 + 2,571 = 10,752 people that did not have an overdue balance > 60 days, 8,181 (76%) were correctly classified. And of the 2,028 + 8,725 = 10,753 people that did have an overdue balance > 60 days, 8,725 (81%) were correctly classified.

4) XGBoost

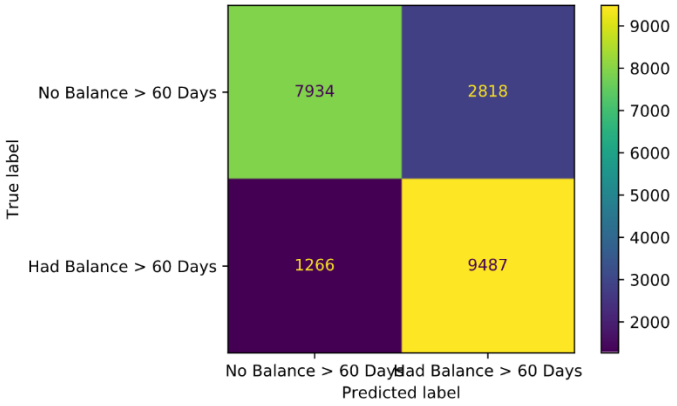

Figure 17. Confusion Matrix of XGBoost

Using XGBoost, 7,934 + 2,818 = 10,752 people that did not have an overdue balance > 60 days, 7,934 (74%) were correctly classified. And of the 1,266 + 9,487 = 10,753 people that did have an overdue balance > 60 days, 9,487 (88%) were correctly classified.

VII. CONCLUSION AND FUTURE SCOPE

This report performs the credit card approval prediction using classification models and machine learning techniques based on the applicant's information. We examined the decision trees and neural network using a training dataset and came to the conclusion that the credit card application approvals can be modelled to make the future prediction. In our analysis, we used decision trees and neural networks and found that XGBoost perform better than the neural networks for this credit card approval prediction process. Another study that is made in this paper is to measure the effect of pruning and we found it had improved the prediction accuracy whereas saving the evaluation time.

The outcome of this work is an Artificial Intelligence prototype for credit card approval prediction which can be further enhanced by using category-based predictors or a hybrid set of regressors for better modeling. For applicants, this prototype could be useful to consider inquiring about credit cards and minimize the unnecessary credit score decrease. For credit card companies, this research provided a data-driven decision-making support in credit card applications inquiries and saved lots of human efforts, which leads to saving expenses for companies while maintaining the efficiency and accuracy.

REFERENCES

[1] Data source: Kaggle. (2019) *Credit Card Approval Prediction.* Retrieved from https://www.kaggle.com/rikdifos/credit-card-approval-prediction

[2] Song, X. (2019) Data Dictionary. Retrieved from https://www.kaggle.com/rikdifos/credit-cardapproval-prediction/discussion/119320

[3] XGBoost Parameters. Retrieved from https://xgboost.readthedocs.io/en/latest/parameter.html

[4] Gutierrez, D. (2014). *Data Munging, Exploratory Data Analysis, and Feature Engineering.* Retrieved from https://insidebigdata.com/2014/06/05/data-munging-exploratory-data-analysis-feature-engineering/

[5] Acharya, M., Armaan, A., Antony, A. (2019). *A Comparison of Regression Models for Prediction of Graduate Admissions.* IEEE International Conference on Computational Intelligence in Data Science.

[6] Ways To Handle Categorical Column Missing Data & Its Implementations. Retrieved from https://medium.com/analytics-vidhya/ways-to-handle-categorical-column-missing-data-its-implementations-15dc4a56893

[7] https://www.kaggle.com/storrinha/st-00-data-exploration-v1

APPENDIX: Jupyter Notebook

In [ ]:
```
# import libraries and data
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
import itertools

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix, precision_score, recall_scc
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier


import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers
```

In [ ]:
```
applications_df = pd.read_csv('application_record.csv')
credit_df = pd.read_csv('credit_record.csv')
```

In [ ]:
```
 plt.rcParams['figure.facecolor'] = 'white'
```

Exploratory Analysis & Preprocessing

In [ ]:
```
# For Applications dataset:
# View shape, nulls, cardinality, duplication (unique count of ID)
applications_df.shape
```

In [ ]:
```
applications_df.isnull().sum()/applications_df.shape[0]
```

In [ ]:
```
# For Credit Records dataset:
# View shape, nulls, cardinality, duplication (unique count of ID)
credit_df.shape
```

In [ ]:
```
credit_df.isnull().sum()
```

In [ ]:
```
print("Unique ID's in credit records dataset:")
print(credit_df['ID'].nunique())
print('')
print("Unique ID's in applications dataset:")
print(applications_df['ID'].nunique())
```

Addressing Null Values

In [ ]:
```
# 30% of Occupation Type column is null
applications_df['OCCUPATION_TYPE'].value_counts().sort_values().plot(kind='barh', figsize=(7,5))
```

Because Occupation Type is a categorical variable, some possible options for addressing nulls are:

1) Replace nulls with the mode

2) Ignore observations

3) Create a new category within variable

4) Predict the observation

However, since our dataset includes a comparable variable called Income Type, we will drop Occupation Type altogether.

In [ ]:
```
applications_df.drop('OCCUPATION_TYPE', axis=1, inplace=True)
```

We also see above that the Credit Records dataset has a long structure with 45,985 unique values in 1,048,575 rows, while the Applications dataset has minimal duplication.

Therefore, to merge the two into one de-duplicated dataset with our target prediction variable we will:

1) Identify how many IDs are common to both datasets (36,457)

2) Reshape Credit Records by Grouping by ID

3) Identify IDs where at any point in time there was a balance 60+ days overdue

In [ ]:
```
len(set(applications_df['ID']).intersection(set(credit_df['ID'])))
id_index = list(set(applications_df['ID']).intersection(set(credit_df['ID'])))
applications_df = applications_df[applications_df['ID'].isin(id_index)]
print(applications_df.shape)
```

In [ ]:
```
credit_df['dep_value'] = None
credit_df['dep_value'][credit_df['STATUS'] =='2']='Yes'
credit_df['dep_value'][credit_df['STATUS'] =='3']='Yes'
credit_df['dep_value'][credit_df['STATUS'] =='4']='Yes'
credit_df['dep_value'][credit_df['STATUS'] =='5']='Yes'
```

In [ ]:
```
resp=credit_df.groupby('ID').count()
resp['dep_value'][resp['dep_value'] > 0]='Yes'
resp['dep_value'][resp['dep_value'] == 0]='No'
resp = resp[['dep_value']]
```

In [ ]:
```
df=pd.merge(applications_df,resp,how='inner',on='ID')
df['target']=df['dep_value']
df.loc[df['target']=='Yes','target']=1
df.loc[df['target']=='No','target']=0
df.head().T
```

In [ ]:
```
#rename columns
df.rename(columns={'CODE_GENDER':'Gender','FLAG_OWN_CAR':'Car','FLAG_OWN_REALTY':'Reality','CNT_CHILDREN':'Child
'NAME_EDUCATION_TYPE':'education','NAME_HOUSING_TYPE':'housing','FLAG_EMAIL':'email','NAME_INCOME_TYPE':'income_
},inplace=True)
```

Visualization & Descriptive Analysis

In [ ]:
```
# How do income and education relate to credit status?
plt.figure(figsize=(5,5))
sns.barplot(x="income", y="education", hue="dep_value", data=df)
```

In [ ]:
```
plt.figure(figsize=(5,5))
sns.stripplot(x="dep_value", y="income", data=df, jitter=True)
```

In [ ]:
```
plt.figure(figsize=(5,15))
sns.catplot(x="dep_value", y="income", hue="Gender", kind="box", data=df)
```

In [ ]:
```
# add column for age in years
df['age'] = df['DAYS_BIRTH']/-365
# add column for years in workforce
df['years_working'] = df['DAYS_EMPLOYED']/-365
# drop columns we aren't going to use
df = df.drop(columns={'ID', 'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHONE', 'email', 'DAYS_BIRTH', 'DAYS_EMPLOYED
```

In [ ]:
```
# create subset risky_df to look for relationships within only "high risk"
# "high risk" = at any point in time had status 60 days or more overdue
risky = ['1']
risky_df = df.loc[df['target'].isin(risky)]

# Within risky subset, where can relationships be identified?
plt.figure(figsize=(3,3))
sns.pairplot(risky_df)
```

In [ ]:
```
# Within risky subset, what are the relationships between income, age, gender and credit status?
plt.figure(figsize=(10, 10))
sns.lmplot(x="income", y="age", hue="Gender", data=risky_df)
```

In [ ]:
```
# Income by gender of applicants in dataset
sns.catplot(data=df.sort_values("income"), orient="h", kind="box", x="income", y="Gender", height=3, aspect=2)
```

Feature Engineering

Binary Data

In [ ]:
```
# replace M/F with 0/1
df['Gender'] = df['Gender'].replace(['F','M'],[1,0])
print(df['Gender'].value_counts())
# replace car flag with 0/1
df['Car'] = df['Car'].replace(['N','Y'],[0,1])
print(df['Car'].value_counts())
# replace realty with 0/1
df['Reality'] = df['Reality'].replace(['N','Y'],[0,1])
print(df['Reality'].value_counts())
```

Categorical Data

In [ ]:
```
# Convert continuous income to category
df['income'] = pd.cut(df['income'], bins=3, labels=["low", "medium", "high"])
# Convert continuous Children to category
df.loc[df['Children'] >= 2,'Children']='2orMore'
# Convert continuous Family Size to category
df.loc[df['family_size'] >= 3,'family_size']='3orMore'
# Convert continuous Age to category
df['age'] = pd.cut(df['age'], bins=3, labels=["young", "middle_age", "older"])
# Convert continuous Years Working to category
df['years_working'] = pd.cut(df['years_working'], bins=3, labels=["entry", "mid_career", "seasoned"])
df.head(2)
```

One Hot Encoding

In [ ]:
```
# convert all categories to binary
df = pd.get_dummies(df, columns=['income_type',
                                 'education',
                                 'housing',
                                   'income',
                                   'Children',
                                   'family_size',
                                   'age',
                                   'years_working'])
df.head(2)
```

In [ ]:
```
# rename columns
df = df.drop(columns={'dep_value'})
df.columns = ['Gender','Car','Reality','target','income_type_Commercial','income_type_Pensioner','income_type_St
              'income_type_Student','income_type_Working','education_degree','education_Higher_ed',
              'education_Incomplete_higher','education_Lower_secondary','education_Secondary','housing_apartment
              'housing_house','housing_municipal','housing_office_apartment','housing_rented_apartment','housing
              'income_low','income_medium','income_high','Children_0','Children_1','Children_2orMore','family_si
              'family_size_2','family_size_3orMore','age_young','age_middle','age_older','years_working_entry',
              'years_working_mid_career','years_working_seasoned']
```

In [ ]:
```
df['income_type_Commercial'] = df.income_type_Commercial.astype('int64')
df['income_type_Pensioner'] = df.income_type_Pensioner.astype('int64')
df['income_type_State'] = df.income_type_State.astype('int64')
df['income_type_Student'] = df.income_type_Student.astype('int64')
df['income_type_Working'] = df.income_type_Working.astype('int64')
df['education_degree'] = df.education_degree.astype('int64')
df['education_Higher_ed'] = df.education_Higher_ed.astype('int64')
df['education_Incomplete_higher'] = df.education_Incomplete_higher.astype('int64')
df['education_Lower_secondary'] = df.education_Lower_secondary.astype('int64')
df['education_Secondary'] = df.education_Secondary.astype('int64')
df['housing_apartment'] = df.housing_apartment.astype('int64')
df['housing_house'] = df.housing_house.astype('int64')
df['housing_municipal'] = df.housing_municipal.astype('int64')
df['housing_office_apartment'] = df.housing_office_apartment.astype('int64')
df['housing_rented_apartment'] = df.housing_rented_apartment.astype('int64')
df['housing_parents'] = df.housing_parents.astype('int64')
df['income_low'] = df.income_low.astype('int64')
df['income_medium'] = df.income_medium.astype('int64')
df['income_high'] = df.income_high.astype('int64')
df['Children_0'] = df.Children_0.astype('int64')
df['Children_1'] = df.Children_1.astype('int64')
df['Children_2orMore'] = df.Children_2orMore.astype('int64')
df['family_size_1'] = df.family_size_1.astype('int64')
df['family_size_2'] = df.family_size_2.astype('int64')
df['family_size_3orMore'] = df.family_size_3orMore.astype('int64')
df['age_young'] = df.age_young.astype('int64')
df['age_middle'] = df.age_middle.astype('int64')
df['age_older'] = df.age_older.astype('int64')
df['years_working_entry'] = df.years_working_entry.astype('int64')
df['years_working_mid_career'] = df.years_working_mid_career.astype('int64')
df['years_working_seasoned'] = df.years_working_seasoned.astype('int64')
```

Split data and fix imbalance using SMOTE

In [ ]:
```
# columns of data we will use to make classifications
X = df.drop('target', axis=1).copy()

# what we want to predict
y = df['target'].copy()
```

In [ ]:
```
# Checking if the data is imbalanced or not
sum(y)/ len(y)
# 1.69% of the applicants are target users. We need to make sure we maintain the same % across training and test
# It's called "stratification": split the data to maintain the ratio.
```

In [ ]:
```
# Using Synthetic Minority Over-Sampling Technique(SMOTE) to overcome sample imbalance problem.
X_balance,Y_balance = SMOTE().fit_sample(X,y)
X_balance = pd.DataFrame(X_balance, columns = X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_balance,Y_balance,
                                                    stratify=Y_balance, test_size=0.3,
                                                    random_state = 1024)
```

Logistic Regression

In [ ]:
```
logistic_model = LogisticRegression(C=0.8,
                            random_state=0,
                            solver='lbfgs')
logistic_model.fit(X_train, y_train)
y_predict = logistic_model.predict(X_test)

print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

Decision Tree

In [ ]:
```
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# precision = (TP) / (TP+FP)
print('Precision Score is {:.5}'.format(precision_score(y_test, y_predict)))
# recall = (TP) / (TP+FN)
print('Recall Score is {:.5}'.format(recall_score(y_test, y_predict)))
```

In [ ]:
```
%%time
clf_dt = DecisionTreeClassifier(random_state=1024)
clf_dt = clf_dt.fit(X_train, y_train)
y_predict = clf_dt.predict(X_test)

plot_confusion_matrix(clf_dt, X_test, y_test, display_labels=["No Balance > 60 Days", "Had Balance > 60 Days"])
```

In the confusion matrix, we see that of the 8,105 + 2,647 = 10,752 people that **did not have an overdue balance > 60 days**, 8,105 **(75%)** were correctly classified. And of the 1,435 + 9,318 = 10,753 people that **did have an overdue balance > 60 days**, 24 **(73%)** were correctly classified.

In [ ]:
```
path = clf_dt.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
ccp_alphas = ccp_alphas[:-1]

clf_dts = []
for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=1024, ccp_alpha=ccp_alpha)
    clf_dt.fit(X_train, y_train)
    clf_dts.append(clf_dt)
```

In [ ]:
```
# plot accuracy
train_scores = [clf_dt.score(X_train, y_train) for clf_dt in clf_dts]
test_scores = [clf_dt.score(X_test, y_test) for clf_dt in clf_dts]

fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train", drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test", drawstyle="steps-post")
ax.legend()
plt.show()
```

In [ ]:
```
%%time
# use cross validation to find best alpha
alpha_loop_values = []
for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    scores = cross_val_score(clf_dt, X_train, y_train, cv=5)
    alpha_loop_values.append([ccp_alpha, np.mean(scores), np.std(scores)])

alpha_results = pd.DataFrame(alpha_loop_values,
                             columns=['alpha', 'mean_accuracy', 'std'])

alpha_results.plot(x='alpha',
                   y='mean_accuracy',
                   yerr='std',
                   marker='o',
                   linestyle='--')
```

In [ ]:
```
# Find exact value
alpha_results[(alpha_results['alpha'] > 0.000)
              &
              (alpha_results['alpha'] < 0.001)]
```

In [ ]:
```
# Store ideal value for alpha
ideal_ccp_alpha = 0.001
```

In [ ]:
```
%%time
clf_dt_pruned = DecisionTreeClassifier(random_state=1024,
                                       ccp_alpha=ideal_ccp_alpha)
clf_dt_pruned = clf_dt_pruned.fit(X_train, y_train)
y_predict_pruned = clf_dt_pruned.predict(X_test)

print(pd.DataFrame(confusion_matrix(y_test,y_predict_pruned)))
```

In [ ]:
```
# confusion matrix
plot_confusion_matrix(clf_dt_pruned,
                      X_test,
                      y_test,
                      display_labels=["No Balance > 60 Days", "Had Balance > 60 Days"])
```

In [ ]:
```
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# precision = (TP) / (TP+FP)
print('Precision Score is {:.5}'.format(precision_score(y_test, y_predict)))
# recall = (TP) / (TP+FN)
print('Recall Score is {:.5}'.format(recall_score(y_test, y_predict)))
```

In [ ]:
```
# Draw Decision Tree
plt.figure(figsize=(12,12))
plot_tree(clf_dt_pruned,
           filled=True,
         rounded=True,
          class_names=["No Overdue", "Yes Overdue"],
          feature_names=X.columns)

plt.savefig('decision_tree.png')
```

Random Forest

In [ ]:
```
%%time
clf_rf = RandomForestClassifier(n_estimators=250,
                                max_depth=50,
                                min_samples_leaf=16
                                )
clf_rf.fit(X_train, y_train)
y_predict = clf_rf.predict(X_test)

print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

In [ ]:
```
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# precision = (TP) / (TP+FP)
print('Precision Score is {:.5}'.format(precision_score(y_test, y_predict)))
# recall = (TP) / (TP+FN)
print('Recall Score is {:.5}'.format(recall_score(y_test, y_predict)))
```

In [ ]:
```
clf_rf_matrix = pd.crosstab(y_test, y_predict, rownames=['Actual'], colnames=['Predicted'])
sns.heatmap(clf_rf_matrix, annot=True)
```

GBM —— lightGBM

In [ ]:
```
%%time
from lightgbm import LGBMClassifier
clf_gbm = LGBMClassifier(num_leaves=35,
                         max_depth=8,
                         learning_rate=0.02,
                         n_estimators=250,
                         subsample = 0.8,
                         colsample_bytree =0.8
                        )
clf_gbm.fit(X_train, y_train)
y_predict = clf_gbm.predict(X_test)

print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

In [ ]:
```
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# precision = (TP) / (TP+FP)
print('Precision Score is {:.5}'.format(precision_score(y_test, y_predict)))
# recall = (TP) / (TP+FN)
print('Recall Score is {:.5}'.format(recall_score(y_test, y_predict)))
```

In [ ]:
```
# Confusion Matrix on the test data
plot_confusion_matrix(clf_gbm,
                      X_test,
                      y_test,
                      values_format='d',
                      display_labels=["No Balance > 60 Days", "Had Balance > 60 Days"])
```

In [ ]:
```
#Showing important features:

def plot_importance(classifer, x_train, point_size = 25):
    #plot feature importance
    values = sorted(zip(x_train.columns, classifer.feature_importances_), key = lambda x: x[1] * -1)
    imp = pd.DataFrame(values,columns = ["Name", "Score"])
    imp.sort_values(by = 'Score',inplace = True)
    b = sns.scatterplot(x = 'Score',y='Name', linewidth = 0,
                  data = imp,s = point_size, color='red')
    fig = plt.gcf()
    fig.set_size_inches(5, 7) # Change plot size
    b.set_xlabel("importance",fontsize=15)
    b.set_ylabel("features",fontsize=15)
    b.tick_params(labelsize=10)

plot_importance(clf_gbm, X_train,20)

plt.savefig('gbm_feature_importance.png')
```

XG Boost

In [ ]:
```
%%time
import xgboost as xgb

# Creating the XGBClassfier shell
clf_xgb = xgb.XGBClassifier(objective='binary:logistic', missing=None, seed=42)

'''
Instead of finding the optimal number of tress using K-Cross Validation,
we use early stopping to stop the tree when the cost function no longer reduces.
XGBoost will do the cross validation; we just have to specifiy the
number of rounds to exhaust with no improvement before stopping.
We use evaluation metric as Area Under Precision-Recall Curve
'''

clf_xgb.fit(X_train,
            y_train,
            verbose=True,
            early_stopping_rounds=10,
            eval_metric='aucpr',
            eval_set=[(X_test, y_test)])

# After building 99 trees, the model doesn't improve any longer. base_score=0.5
# So we check for the next 10 iterations and stop.

# make predictions for test data
y_predict = clf_xgb.predict(X_test)

# evaluate predictions
print(pd.DataFrame(confusion_matrix(y_test,y_predict)))
```

In [ ]:
```
print('Accuracy Score is {:.5}'.format(accuracy_score(y_test, y_predict)))
# precision = (TP) / (TP+FP)
print('Precision Score is {:.5}'.format(precision_score(y_test, y_predict)))
# recall = (TP) / (TP+FN)
print('Recall Score is {:.5}'.format(recall_score(y_test, y_predict)))
```

In [ ]:
```
# Confusion Matrix on the test data
plot_confusion_matrix(clf_xgb,
                      X_test,
                      y_test,
                      values_format='d',
                      display_labels=["No Balance > 60 Days", "Had Balance > 60 Days"])
```

In [ ]:
```
xgb.plot_importance(clf_xgb)
plt.rcParams['figure.figsize'] = [5, 7]
plt.show()

plt.savefig('xgb_feature_importance.png')
```

Neural Network

In [ ]:
```
model = keras.Sequential([layers.Dense(64, activation='relu'),
                          layers.Dense(128, activation='relu'),
                          layers.Dense(128, activation='relu'),
                          layers.Dense(1, activation='sigmoid')])

model.compile(keras.optimizers.Adam(), keras.losses.BinaryCrossentropy(), metrics=[keras.metrics.BinaryAccuracy(

model.fit(np.array(X_train), np.array(y_train), validation_split=0.1, epochs=10)
y_predict = model.predict(np.array(X_test))
met = keras.metrics.BinaryAccuracy()
met(np.array(y_test), y_predict)
print('Neural network Accuracy Score is {:.5}'.format(met.result()))
```

References & Citations

Project on GitHub: https://github.com/atxmoon/eai6000_group_4_final_project (https://github.com/atxmoon/eai6000_group_4_final_project)