University of Reading

Department of Computer Science

Bank Loan Eligibility Predictor by Classification

By: Faris Hanna

Module Code: CS3AI18

Assignment Title: Artificial Intelligence Coursework

Convenor Name: Dr. Yevgeniya Kovalchuk

Student Number: 27018697

Date of completion: 04/03/2022

Word Count: 1969

A report submitted in partial fulfilment of the requirements of the University of Reading for the degree of
Bachelor of Science in Computer Science

March 05, 2022

Abstract

In the banking sector, it is crucial to know if a customer applying for a loan is eligible. A machine learning algorithm will be implemented to predict whether a customer can be eligible for a loan. Processes such as data collecting, cleaning, exploratory data analysis, data preprocessing, feature selection, model implementation and evaluation will be shown. The model implemented in achieving our classification solution is the XGBoost Classifier Model. With optimizations such as using GridSearchCV and using a pipeline, the results have been phenomenal, reaching 99% accuracy. This study can highly aid the financial sector in concerns of customers avoiding loan payments, as this tool can reduce these risks.

Table of Contents

1. Introduction

1.1 Background

A loan is an amount of money borrowed for a set period within an agreed repayment schedule. (Bank Finance: Advantages and Disadvantages of Bank Loans, n.d.). Loans can be used to benefit both parties, as the lender will earn a profit, and the customer can have access to extra finance needed. However, in cases like this, trust is an important issue. It raises questions such as, how reliable is it for a customer to pay back their loans? Are there any factors that can determine how likely a customer can return their loans?

In the 2008 housing crisis, according to Investopedia, U.S. government-sponsored mortgage lenders Fannie Mae and Freddie Mac made **home loans accessible to borrowers** who had low credit scores and a higher risk of defaulting on loans (Kosakowski, 2022). This has resulted in an economy crisis as many borrowers were not able to pay back their loans. Which has led to more strict requirements when applying for a loan. Loans are extremely beneficial; however, they should not be distributed without caution.

1.2 Problem Statement

The specific problem that will be solved is determining whether a customer can be eligible for a loan. There are numerous factors that contribute to a customer being eligible, such as their income and age, however, these factors cannot solely predict whether a customer is eligible. Therefore, a machine learning model will be implemented to predict accurately. This will require a dataset of previous customers, with relevant information, that have applied for a loan, and whether they have been eligible. This solution can highly aid the banking industry as a tool in determining whether a customer is eligible or not, which can also lead to automatically processing and providing loans to customers that are eligible. This can also provide online bank customers the ability to apply for a loan online, and to instantaneously return an answer, resulting in saving time and resources.

2. Dataset Description

The dataset has been obtained from Kaggle. It contains data of 5000 customers from the Thera Bank. From the 5000 customers, only 480 customers have been eligible for a loan. Fortunately, the data is mostly clean with no NULL values and correct datatypes.

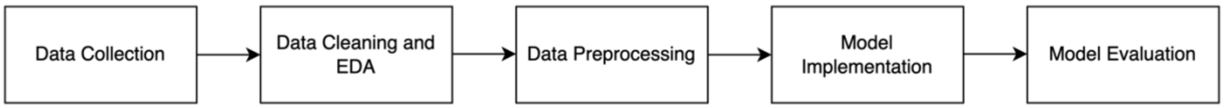
The dataset contains the following attributes:

Attribute	Description	Type
ID	Customer ID	Int
Age	Customer Age	Int
Experience	Customer years of professional experience	Int
Income	Customer Income	Int
ZIP Code	Customer ZIP Code	Int
Family	Number of family members	Int
CCAvg	Customer average spending of credit cards	Float
Education	Customer Education Level	Object encoded to Int
Mortgage	Mortgage house value (if any)	Int
Personal Loan	Did the customer accept this loan?	Boolean encoded to Int
Securities Account	Does Customer have securities account?	Boolean encoded to Int
CD Account	Does Customer have a certificate of deposit account?	Boolean encoded to Int
Online	Does Customer use banks online facilities?	Boolean encoded to Int
CreditCard	Does Customer have a credit card?	Boolean encoded to Int

3. Machine Learning Model

3.1 Summary of the Approach

The process of the approach will be in the following steps:



With the data already collected, the first step of the process will be to clean and perform EDA analysis on the data, which would include removing any outliers and making sure values are 'correct' and balanced. The data will be visualized to ensure this. The dataset will then be taken to the preprocessing step, which will include splitting the features and the target variable, balancing the dataset by the target variable, feature selection, and splitting the data into training and testing. Once the data is ready for learning, it will be implemented to the XGBClassifier model from xgboost, and the model will then be tested to return the accuracy scores and the confusion matrix. The final step is evaluating the model, which would include processes such as optimization by performing hyperparameter tuning and implementing a pipeline, and ensuring the model is working accurately by using cross validation techniques.

3.2 Data visualization, preprocessing, feature selection

The first step will involve cleaning the dataset. To help in this, *df.info()* will be used for showing any null values and each columns datatype while *df.describe()* will show the data statistics. The data has no null values and correct datatypes, but it is still not clean.

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000	5000.000000	5000.000000
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.498800	0.096000	0.104400	0.06040	0.596800	0.294000
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.713802	0.294621	0.305809	0.23825	0.490589	0.455637
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.00000	1.000000	0.000000
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.00000	1.000000	1.000000
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.00000	1.000000	1.000000

Figure 3.1: Data statistics using df.describe()

In figure 3.1, it is shown that -3 is the minimum value of Experience. The negative values can be converted to positive values by running the following script:

```
df['Experience'] = df['Experience'].abs()
```

There are possibly outliers in the Income, CCAvg, and Mortgage columns as their max values are far apart from the 75% percentile. By using the Seaborn library, these columns will be plotted as a boxplot, showing the following:

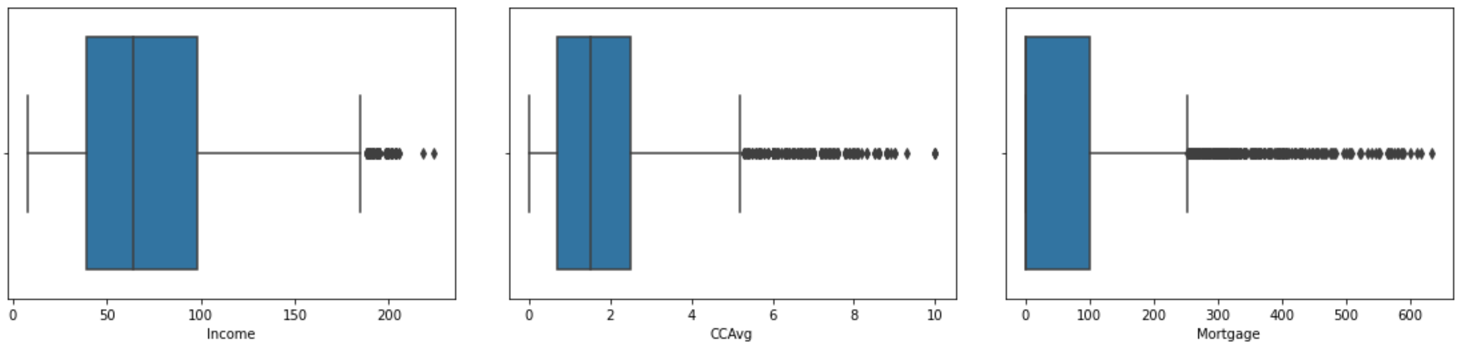


Figure 3.2: Boxplots for Income, CCAvg, and Mortgage

As shown in Figure 3.2, the Income and CCAvg boxplots have outliers. These will be removed to prevent any model bias. The Mortgage column can be ignored as there are many values that are zero, which results in most non-zero values as outliers. Only 4 total outliers have been removed.

The Age and Experience columns are both by unit years. By using the Matplotlib library, it would be best to plot a histogram to view the distribution between the ages and years of experience.

In figure 3.3, the ages of customers are between 30-60 years old, while the minority are in their 20's or 60's. Experience years are between 0-40. Overall, the two columns look evenly distributed.

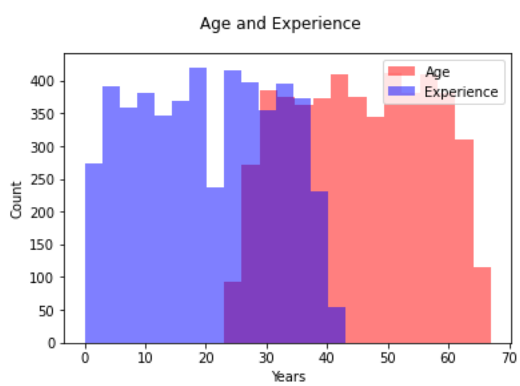
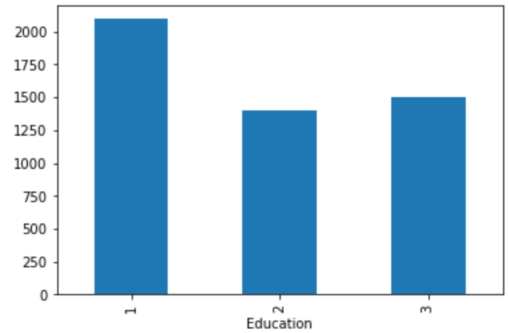


Figure 3.3: Histogram plots between Age and Experience columns

The Education and Family columns are categorized into 3-4 values. Education presenting the level, and Family shows how many family members of the customer. It would be best to plot bar charts of the counts to ensure that the values are evenly distributed. In figure 3.4, values are evenly distributed, and the columns are acceptable.



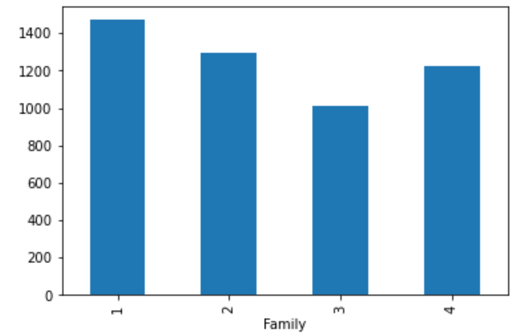


Figure 3.4: Bar chart of counts of Education and Family columns

The rest of the columns in the data are in the form of Boolean. There are only two values of either 1 (Yes) or 0 (No). In the case of this, it would be best to plot the counts of the 1's and 0's for each column to determine if there is any imbalanced data. It is already known that there is an imbalance of the Personal Loan column since most customers were not given a loan. It would be best to check the rest of the columns for any imbalances to know what to expect if the model should have any errors. In figure 3.5, a bar chart comparing the 1's and 0's is represented, and the Securities Account and CD Account are also imbalanced. For now, these columns will stay, and the Personal Loan column will be rebalanced (demonstrated later).

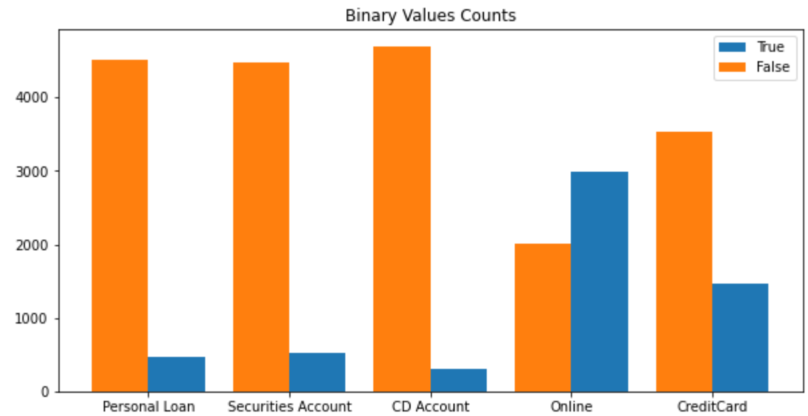


Figure 3.5: Bar charts of the counts of the binary Boolean columns.

The final step of the visualization process is producing a heatmap of the data. This will show the correlation between all the columns. This will help in determining the features that contribute most to the Personal Loan column. In figure 3.6, the Personal Loan column is correlated with Income, CCAvg, Family, Education, Mortgage and CD Account. These columns are highly likely to be the most important features, but the selection process will be shown later.

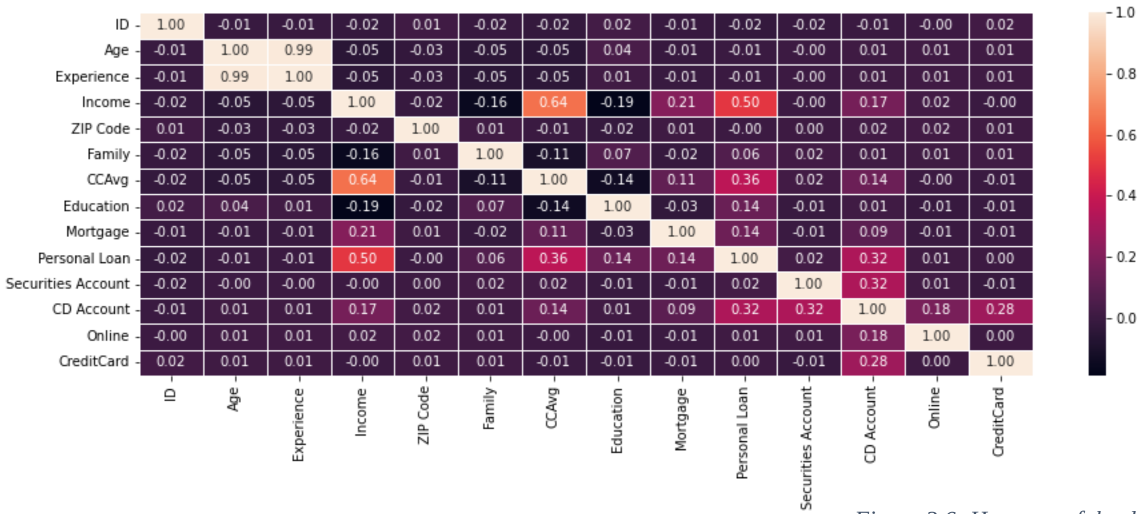


Figure 3.6: Heatmap of the data

For preprocessing, ID and ZIP Code will be removed as they are likely irrelevant. It will be removed as follows: `data = df.drop(columns=['ID', 'ZIP Code'])`

Now, the features and the target variable must be separated to split the data for training and testing. It will be done as follows:

```
features = data.drop(columns=['Personal Loan']).values
target = data['Personal Loan']
```

Before splitting the training and testing data, the data must be rebalanced first as it is known that only 9.8% of the loans are accepted. Running *target.value_counts()* will confirm the imbalance. To rebalance the data, using SMOTE (short for Synthetic Minority Oversampling Technique *(ML | Handling Imbalanced Data with SMOTE and Near Miss Algorithm in Python - GeeksforGeeks, 2022)*) will help in achieving this by running the following code:

```
smote = SMOTE()
features, target = smote.fit_resample(features, target)
```

The next step of the preprocessing will be feature selection. Going back to figure 3.6, it is known that there are 6 features that are highly relevant to the model. Therefore, using SelectKBest will return these features if given k value of 6. However, it has been decided to select the features during the pipeline stage and therefore will not be done at this stage.

The next step will be to split the training and testing data. To ensure the model is not overfitting or biased, cross validation will be used confirm the results. Therefore, by using KFold from sci-kit-learn, the training data will be split into 10 splits. The training and testing data will be generated by running the following code:

```
kf = KFold(n_splits=10, shuffle=True)
for train, test in kf.split(features):
      features_train, features_test, target_train, target_test =
      features[train], features[test], target[train], target[test]
```

The data is now ready to be implemented to the model. However, to justify not scaling the features, XGBoost does not need to be standardized or normalized since it is not sensitive to monotonic transformations of its features for the same reason that decision trees and random forests are not *(What are the implications of scaling the features to xgboost?, 2022)*.

3.3 Model Training and Evaluation

With the features and target split into training and testing data, the model is now ready for training. The model will be initialized and then fitted with the training data. By using the test data, the test score and cross validation scores will be calculated using the *accuracy_score()* and *cross_val_score()* functions from sci-kit-learns' metrics. The cross-validation results will then be plotted in a box plot, and a confusion matrix of the model will be plotted. A function was made for this step since the same process will be repeated when getting the scores of the pipeline and grid search optimizations. The following code (next page) uses model as an argument, and a grid Boolean argument that shows the best parameters if grid search is used:

```python
# Prints scores of a specified model
def model_scores(model, grid=False):
    model.fit(features_train, target_train) # Fit model
    target_pred = model.predict(features_test) # Predict test features

    accuracy = round(accuracy_score(target_test, target_pred, normalize = 
True)*100, 2) # Accuracy
    cv_scores = cross_val_score(model, features, target, cv=kf, n_jobs=-1) # 
CV Score
    cv_mean = round(np.mean(cv_scores)*100, 2) # Mean of CV Scores
    print('Test Results:', f'{accuracy}%') # Print accuracy
    print('CV Results Mean:', f'{cv_mean}%')  # Print CV Results

    # Print best params if gridsearch is used
    if(grid==True):
        print('Best Parameters: ', model.best_params_)
    # Boxplot of cv scores and confusion matrix
    sns.boxplot(x=cv_scores)
    plt.show()
    plot_confusion_matrix(model, features_test, target_test)
    plt.show()
```

For pipelining, there will be two stages which consists of SelectKBest as the feature selection technique, and the XGBClassifier as the classifier. The pipeline will help in choosing which are the best parameters to use on the model and how many features to use, which should return the best possible model. This will be done using sci-kit-learns' Pipeline class. The pipeline will be initialized as follows:

```python
pipe = Pipeline([
('selection', SelectKBest(chi2, k=6)), # Initial value of 6
('classifier', XGBClassifier(use_label_encoder=False, eval_metric='logloss'))
])
```

And to check the scores of the pipeline, the *model_scores* function will do this.

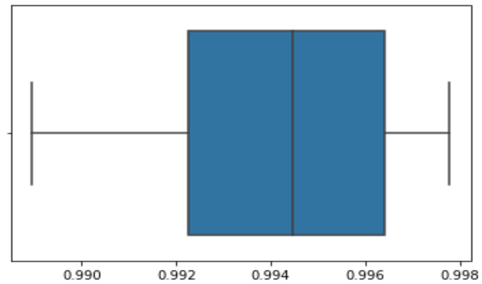
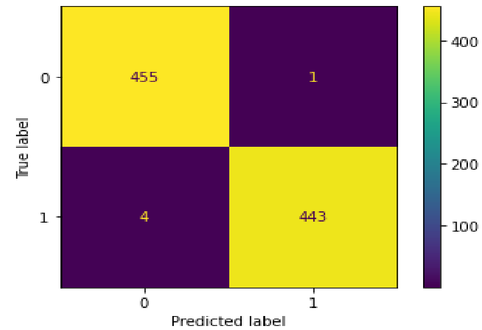
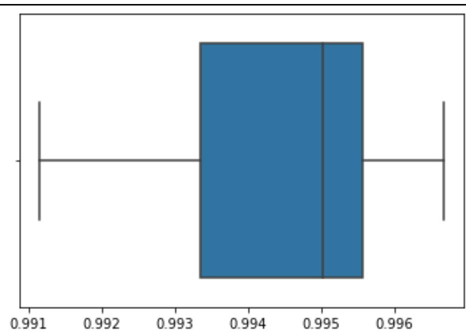
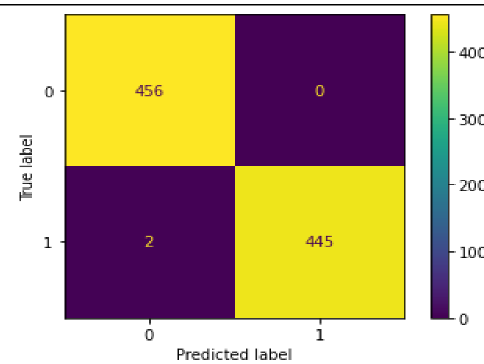
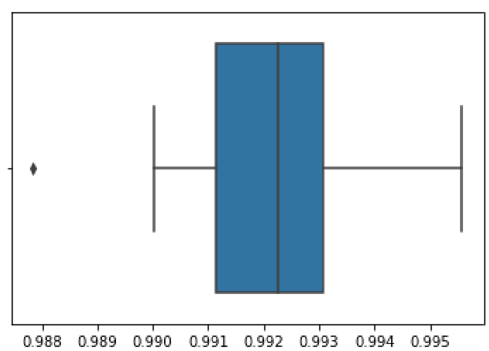
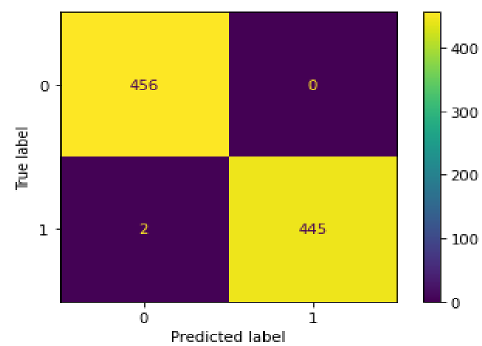
The final step of the implementation is using GridSearchCV. The parameters that will be used are:

- K in selection
- Learning rate in classifier
- Max depth of classifier
- N estimators of classifier

These parameters will be in a config file and will be used on GridSearchCV, which returns the final scores and best parameters to use.

3.4 Results and Discussion

The following table is a summary of all the results:

Model	CV Mean Score	Test Score	Boxplot of Cross-Validation	Confusion Matrix
Original	99.4%	99.45%		
Pipeline	99.44%	99.78%		
GridSearch	99.2%	99.78%		

The original model is already very accurate with a score of 99.45% and only 5 total errors. The pipeline model has had an accuracy of 99.78%, with only 2 total errors. The GridSearch did not have much effect, however the interquartile range as shown in the boxplot has a smaller spread, meaning it is more consistent when predicting. The best parameters of the GridSearch are k=8, learning_rate=0.3, max_depth=6, and n_estimators=1000.

4. Conclusion

4.1 Recommendations

It would be highly recommended to use more than one machine learning algorithm to obtain the highest possible result, such as using a logistic regression, decision tree, KNN, and possibly a neural network. Other feature selection methods can be used to compare the best selection method, such as trying the VarianceThreshold function, or by trying every combination of features that gets the best results, but it would be time-consuming. Other parameters can be used in the GridSearchCV model to get higher possible results, but this would take a lot of energy and time.

4.2 Future Work

This work can still be developed further to have a use-case. A very beneficial use-case, which is already applied in many banks is to create a user-friendly form within the bank application or website where users can apply for a loan online anytime without the need to visit the bank in person.

References

Nibusinessinfo.co.uk. 2022. Advantages and disadvantages of bank loans | nibusinessinfo.co.uk. [online] Available at: https://www.nibusinessinfo.co.uk/content/advantages-and-disadvantages-bank-loans#:~:text=A%20loan%20is%20an%20amount,start%2Dup%20capital .

Kosakowski, P., 2022. The Fall of the Market in the Fall of 2008. [online] Investopedia. Available at: https://www.investopedia.com/articles/economics/09/subprime-market-2008.asp

Dataset obtained from Kaggle: https://www.kaggle.com/krantiswalke/bank-personal-loan-modelling

GeeksforGeeks. 2022. *ML | Handling Imbalanced Data with SMOTE and Near Miss Algorithm in Python - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/

Stack Exchange. 2022. *What are the implications of scaling the features to xgboost?*. [online] Available at: https://stats.stackexchange.com/questions/353462/what-are-the-implications-of-scaling-the-features-to-xgboost .

Appendix

Python source code: model.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 16 21:40:23 2022

@author: farishanna
"""

#to manipulate data as a dataframe
import pandas as pd
# To make any needed calculations
import numpy as np

#to visualise data and results
import matplotlib.pyplot as plt
import seaborn as sns

# For managing imbalanced data
from imblearn.over_sampling import SMOTE
#to split data into training and testing, cross-validation, and finding the
best parameters for most accurate model
from sklearn.model_selection import KFold, GridSearchCV, cross_val_score
#to select best features
from sklearn.feature_selection import SelectKBest, chi2

# To use XGBClassifer algorithm
from xgboost import XGBClassifier
# Creating pipeline
from sklearn.pipeline import Pipeline

#to calculate accuracy score and plot confusion matrix
from sklearn.metrics import accuracy_score, plot_confusion_matrix


df = pd.read_csv('Bank_Personal_Loan_Modelling.csv') # Reading the data into
a pandas dataframe
print(df.head()) # Displays first 5 rows
print(df.info()) # Checking for null values and datatypes
print(df.describe()) # Getting data statistics

#  -- from https://stackoverflow.com/questions/29077188/absolute-value-for-
column-in-python
df['Experience'] = df['Experience'].abs() # Turning negative values postive

# Checking for any outliers using a boxplot
for column in ['Income', 'CCAvg', 'Mortgage']:
```

```
    sns.boxplot(x=df[column])
    plt.show()

# Removing outliers
before = len(df)
df = df.loc[df['Income'] < 220]
df = df.loc[df['CCAvg'] < 10]
print('\nOutliers removed:', before-len(df), '\n')

# Checking histograms of Age and Experience    Obtained from --
https://matplotlib.org/3.1.1/gallery/statistics/histogram_multihist.html
fig, ax = plt.subplots()
# Hist representing Age and Experience
ax.hist(df["Age"], bins=15, alpha=0.5, color="red", label="Age")
ax.hist(df["Experience"], bins=15, alpha=0.5, color="blue",
label="Experience")
# Adding labels, subtitles and legend
ax.set_xlabel("Years")
ax.set_ylabel("Count")
fig.suptitle("Age and Experience")
ax.legend();

# Checking for imbalances
for column in ['Family', 'Education']:
    pivot = pd.pivot_table(df, values='ID', index=[column], aggfunc='count')
    pivot.plot(kind='bar', legend='')
    plt.show()

# Checking for the counts of the binary columns -- used from:
https://www.geeksforgeeks.org/plotting-multiple-bar-charts-using-matplotlib-
in-python/
# To store counts of true and false
true_count = []
false_count = []
# Appends lists
for column in df.columns[9:]:
    true_count.append(len(df.loc[df[column] == 1]))
    false_count.append(len(df.loc[df[column] == 0]))
# Create appropriate x axis
X_axis = np.arange(len(df.columns[9:]))
fig = plt.figure(figsize=(10, 5))
# Creating bar charts with true or false labels
ab_bar_list = [
               plt.bar(X_axis+0.2, true_count, align='edge', width= 0.4,
label = 'True'),
               plt.bar(X_axis-0.2, false_count, align='edge', width= 0.4,
label = 'False')
               ]
# X axis names
plt.xticks(X_axis+0.2, df.columns[9:])
# Title and legend
plt.title("Binary Values Counts")
plt.legend()
```

```
plt.show()

# Heatmap  --style used by: https://www.kaggle.com/yamanizm/personal-loan-
eda-ml-98-iamdatamonkey
plt.figure(figsize=(15,5))
sns.heatmap(df.corr(),annot=True,linewidths=.5,fmt='.2f')
plt.show()


# Splitting feature variables with the target variable
features = df.drop(columns=['Personal Loan']).values
target = df['Personal Loan']

# Checking for imbalanced data on the target
print(target.value_counts())

# Balancing data using SMOTE
smote = SMOTE()
features, target = smote.fit_resample(features, target)

# K fold splitting training and testing data
kf = KFold(n_splits=10, shuffle=True)
for train, test in kf.split(features):
    features_train, features_test, target_train, target_test =
features[train], features[test], target[train], target[test]

# Prints scores of a specified model
def model_scores(model, grid=False):
    model.fit(features_train, target_train) # Fit model
    target_pred = model.predict(features_test) # Predict test features

    accuracy = round(accuracy_score(target_test, target_pred, normalize =
True)*100, 2) # Accuracy
    cv_scores = cross_val_score(model, features, target, cv=kf, n_jobs=-1) #
CV Score
    cv_mean = round(np.mean(cv_scores)*100, 2) # Mean of CV Scores
    print('Test Results:', f'{accuracy}%') # Print accuracy
    print('CV Results Mean:', f'{cv_mean}%')  # Print CV Results

    # Print best params if gridsearch is used
    if(grid==True):
        print('Best Parameters: ', model.best_params_)

    # Boxplot of cv scores and confusion matrix
    sns.boxplot(x=cv_scores)
    plt.show()
    plot_confusion_matrix(model, features_test, target_test)
    plt.show()

# Printing scores for XGBoost model
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
print('\nXGBoost Classifier Scores:')
model_scores(model)
```

```python
# Pipeline --guidance by: https://machinelearningmastery.com/modeling-pipeline-optimization-with-scikit-learn/
pipe = Pipeline([
('selection', SelectKBest(chi2)),
('classifier', XGBClassifier(use_label_encoder=False, eval_metric='logloss'))
])
# Printing pipeline results
print('\nPipeline Model Scores:')
model_scores(pipe)

# Reading parameters in the config file
file = "parameters.config"
parameters = eval(open(file).read())
# Optimizing pipeline using GridSearchCV
grid = GridSearchCV(pipe, parameters, cv=kf)
# Printing gridsearch results
print('\nGridSearchCV Model Scores:')
model_scores(grid, grid=True)
```

Parameters config file: parameters.config

```
{
'selection__k': [6, 8],
'classifier__learning_rate': [0.1, 0.3, 1],
'classifier__max_depth': [4, 6, 8],
'classifier__n_estimators': [10, 100, 1000]
}
```