MACHINE LEARNING

CREDIT APPROVAL PREDICTION

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

Guntreddi Rakesh Naidu (121810305023)
V V Mani Chandra Puppala (121810305032)
H N V R Shashidhar Kappera (121810305043)
Kokkirala Vishnu (121810305008)

Under the esteemed guidance of

Dr.J.Hyma

Associate Professor

DEPARTMENT OFCOMPUTER SCIENCE & ENGINEERING
GITAM
(Deemed to be University)
VISAKHAPATNAM
OCTOBER
2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "Credit Approval Prediction" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 3-11-2021

Registration No(s).	**Name(s)**	**Signature(s)**
121810305023	Rakesh Naidu	
121810305032	Mani Chandra	
121810305043	Shashidhar	
121810305008	Vishnu	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM INSTITUTE OF TECHNOLOGY
GITAM
(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**CREDIT APPROVAL PREDICTION**" is a bonafide record of work carried out by **Guntreddi Rakesh Naidu (121810305023), V V Mani Chandra Puppala (121810305032), H N V R Shashidhar Kappera (121810305043), Kokkirala Vishnu (121810305008)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	**Head of the Department**
Dr.J.Hyma	**Dr. R.Sireesha**
Associate Professor	**Professor**

TABLE OF CONTENTS

1	Abstract		5
2	Introduction		5
3	Literature Review		6
4	Problem Identification & Objectives		7
5	System Methodology		8
6	Overview of Technologies		8
7	Implementation		10
	7.1	Coding	10
	7.2	Testing	29
8	Results & Discussions		34
9	Conclusion and Future scope		34
10	References		34

Description

1. Abstract

The increased credit card defaulters have forced the companies to think carefully before the approval of credit applications. Credit card companies usually use their judgment to determine whether a credit card should be issued to the customer satisfying certain criteria. Some machine learning algorithms have also been used to support the decision. The performance of the built model is compared with the multiple traditional machine learning algorithms: Naive Baye's , K Nearest Neighbours, Decision Tree, Random Forest, XG Boost . Many other results show that the overall performance of our deep learning model is slightly better than that of the other models.

This Project is a machine learning solution to automate the process of credit card approval to an account. The model is trained with multiple attributes and the accuracy is measured with both train and test data.We will use the Credit Approval Dataset which is a collection of credit card applications and the credit approval decisions. The data is available from the UCI Machine Learning Repository. The techniques include data visualization, association rules, logistical regression, and decision trees. This analysis is organized as follows:

1. Generate several data visualizations to understand the underlying data
2. Perform data transformations as needed
3. Build Machine Learning models to the selected data
4. Compare accuracies and decide the best model

2. Introduction

The accurate assessment of consumer credit risk is of uttermost importance for lending organizations. Credit scoring is a widely used technique that helps financial institutions evaluates the likelihood for a credit applicant to default on the financial obligation and decide whether to grant credit or not. The precise judgment of the creditworthiness of applicants allows financial institutions to increase the volume of granted credit while minimizing possible losses. The credit industry has experienced a tremendous growth in the past few decades (Crook et al., 2007). The increased number of potential applicants impelled the development of sophisticated techniques that automate the credit approval procedure and supervise the financial health of the borrower. The large volume of loan portfolios also imply that modest improvements in scoring accuracy may result in significant savings for financial institutions (West, 2000). The goal of a credit scoring model is to classify credit applicants into two classes: the "good credit" class that is liable to reimburse the financial obligation and the "bad credit" class that should be denied credit due to the high probability of defaulting on the financial obligation. The classification is contingent on sociodemographic characteristics of the borrower (such as age, education level, occupation and income), the repayment Performance on previous loans and the type of loan. These models are also applicable to small businesses since these may be regarded as extensions of an individual costumer. In the last few decades, various quantitative methods were proposed in the literature to evaluate consumer loans and improve the credit scoring accuracy (for a review, see e.g. Crook et al., 2007). These

models can be grouped into parametric and non-parametric or data mining models. The most popular parametric models are the linear discriminant analysis and the logistic regression. Linear discriminant analysis was the first parametric technique suggested for credit scoring purposes (Reichert et al., 1983). This approach has attracted criticism due to the categorical nature of the data and the fact that the covariance matrices of the good credit and bad credit groups are typically distinct.

The goal here is to build an end to end automated Machine Learning solution where a user will be able to predict whether a bank customer should be approved for attaining the credit card or not. The user is only need to give the value of feature variables and the model will able to predict the binary outcome (Approve/ Not Approve). The model will be able take care of all intermediate functionalities like cross validation, hyper parameter tuning, algorithm selection etc.

This project shall be delivered in two phases:
Phase 1: All the functionalities with PyPi packages.
Phase 2: Integration of UI to all the functionalities.

Note: All the code will be written in python version 3.6

3. Literature Review

Credit card has evolved to a great level in banking industry. Each banking system consists of an enormous number of datasets to carry customer's transactions of their credit cards. So, banks would be in need of customer profiling. Customer Profiling in banks cognizes the issuer's decisions about whom to give banking facilities and what credit limit to be provided. In previous researches, profiling mainly depended on transaction data or demographic data, but in this research, both transaction and demographic data are merged in order to get more accurate results and minimize the possibility of risk occurrence

By using the best techniques, it leads to improvement in accuracy and helps banks to have high profitability through customer satisfaction by focusing on the valuable customer (companies) which are considered as the main engine in the bank's profitability. This study used k-mean, improved k-mean, fuzzy c-means and neural networks. The used dataset is labeled and for neural network classification creating a new label as a target becomes the main aspect of this study, which helps to reduce the execution time of clustering process and provide the best results with accuracy. Finally, by comparing the accuracy ratio the results show that the neural network is the best clustering technique which could give an accuracy percentage of about 98.08%.

The logistic regression (Wiginton, 1980) allows overcoming these deficiencies and became a common credit scoring tool of practitioners in financial institutions. Non-parametric techniques applied to credit scoring include the k-nearest neighbor (Henley and Hand, 1996), decision trees (Frydman et al., 1985; Davis et al., 1992), artificial neural networks (Jensen, 1992), genetic programming (Ong et al., 2005) and support vector machines (Baesens et al., 2003). More recently, research on hybrid data mining approaches has shown promising results (Lee et al., 2002; Hsieh, 2005; Lee and Chen, 2002). While the pursuit of better classifiers for credit scoring

applications is a crucial research effort, improved accuracies can be easily achieved by aggregating scores predicted by an ensemble of individual classifiers. West et al. (2005) found that the accuracy of an ensemble of neural networks is superior to that of a single neural network in credit scoring and bankruptcy prediction applications. Thanks to all the researchers who published their studies.

4. Problem Identification & Objectives

The proposed project is built end to end. Starting from Data Preprocessing to Deployment. This project includes the features like:

a) Statistical analysis
b) Hyper parameter tuning
c) Best algorithm selection
d) Deployment in Heroku using flask.

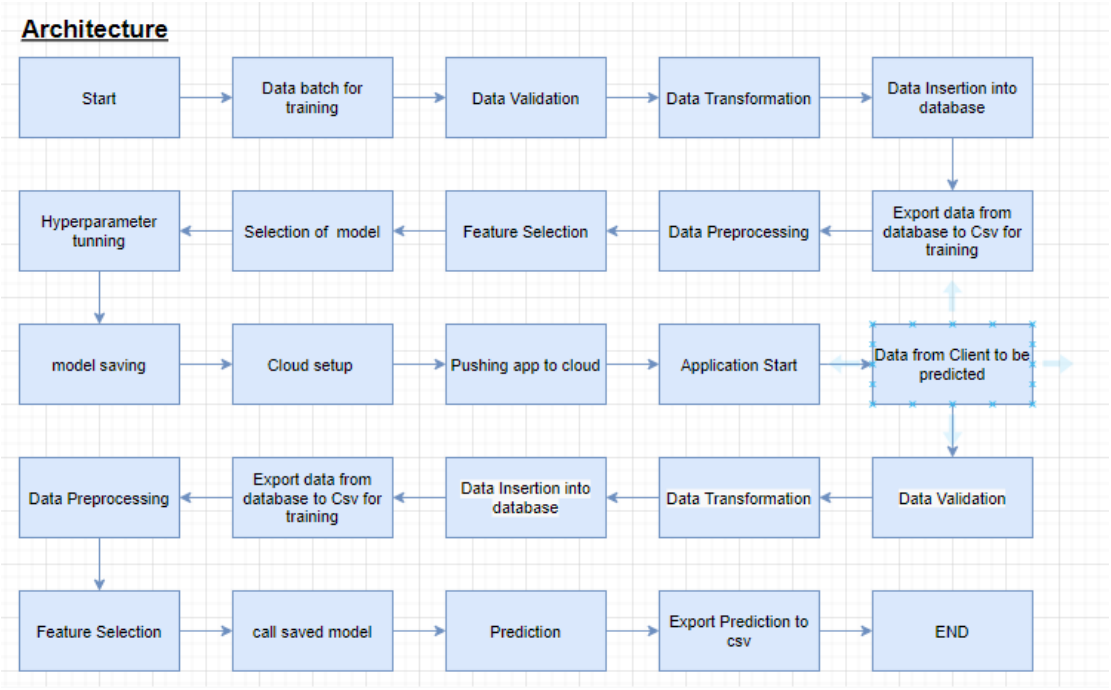
The main objectives of the proposed project are to:

a) Increase the accuracy
b) Do Exploratory data analysis
c) Test the model with different algorithms
d) Try different model selection criteria
e) Do Hyperparameter tuning
f) Deploy the project for easy use

Different technologies or libraries used in the project are:

a) numpy, pandas
b) matplotlib, seaborn
c) scipy, scikit learn
d) xgboost
e) html, css
f) flask, gunicorn

5. System Methodology



6. Overview of Technologies

6.1. Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

6.2. Pandas

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState's ActivePython.

6.3. Matplotlib

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

6.4. Seaborn

Seaborn is an open-source Python library built on top of matplotlib. It is used for data visualization and exploratory data analysis. Seaborn works easily with dataframes and the Pandas library. The graphs created can also be customized easily. Below are a few benefits of Data Visualization.

6.5. Scipy

SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Olliphant.

6.6. Scikit learn

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:
- ✓ **Regression**, including Linear and Logistic Regression
- ✓ **Classification**, including K-Nearest Neighbors
- ✓ **Clustering**, including K-Means and K-Means++
- ✓ **Model selection**
- ✓ **Preprocessing**, including Min-Max Normalization

6.7. Xgboost

XGBoost is a tree based ensemble machine learning algorithm which is a scalable machine learning system for tree boosting. XGBoost stands for Extreme Gradient Boosting. It uses more accurate approximations to find the best tree model. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now.

6.8. HTML, CSS

HTML (the Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, CSS the (visual and aural) layout, for a variety of devices. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web Applications. Learn more below about:

6.9. Flask, gunicorn

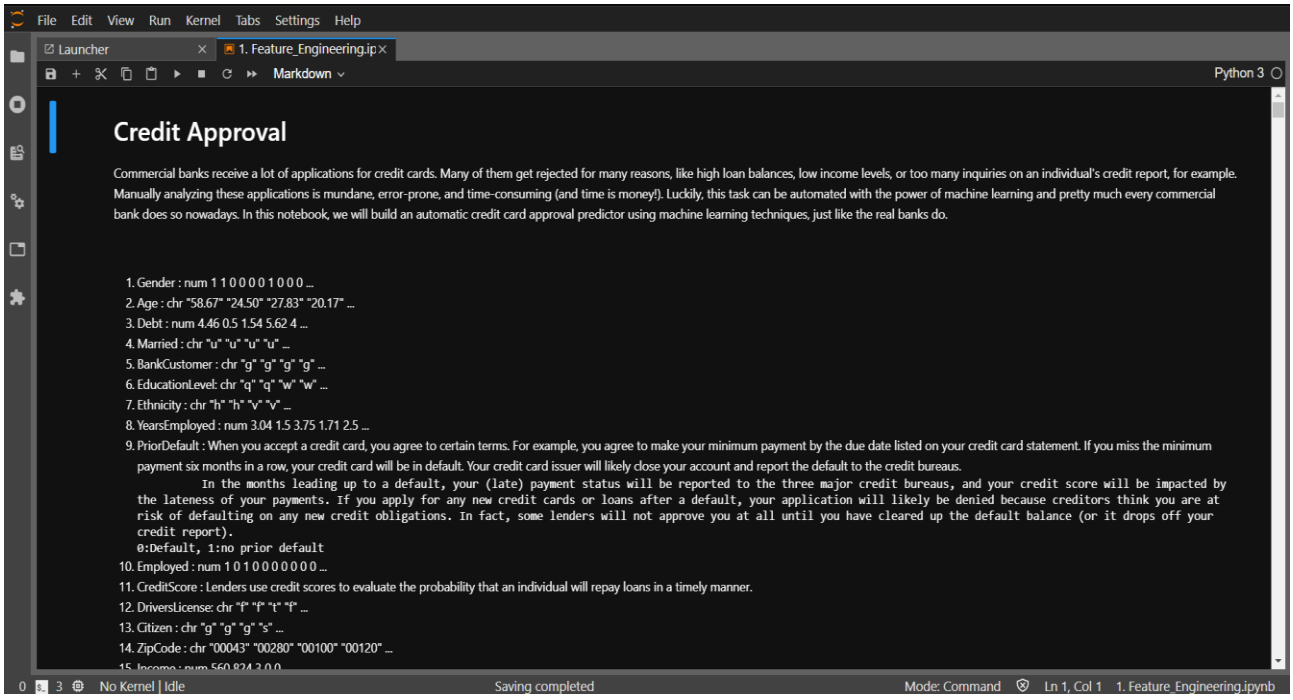
Flask is a Python-based microframework that is popular with web developers, given its lightweight nature and ease of use. Gunicorn is a Python WSGI HTTP Server that

uses a pre-fork worker model. By using gunicorn, we will be able to serve your Flask application on more than one thread.

7. Implementation

7.1 Coding

Feature Engineering



[4]:
```
dataset.columns
```

[4]:
```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], dtype='int64')
```

The output may appear a bit confusing at its first sight, but let's try to figure out the most important features of a credit card application. The features of this dataset have been anonymized to protect the privacy, but Analysis of Credit Approval Data blog gives us a pretty good overview of the probable features. The probable features in a typical credit card application are Gender, Age, Debt, Married, BankCustomer, EducationLevel, Ethnicity, YearsEmployed, PriorDefault, Employed, CreditScore, DriversLicense, Citizen, ZipCode, Income and finally the ApprovalStatus. This gives us a pretty good starting point, and we can map these features with respect to the columns in the output.

As we can see from our first glance at the data, the dataset has a mixture of numerical and non-numerical features. This can be fixed with some preprocessing, but before we do that, let's learn about the dataset a bit more to see if there are other dataset issues that need to be fixed.

[5]:
```
columns = ["Gender","Age","Debt","Married","BankCustomer","EducationLevel","Ethnicity","YearsEmployed","PriorDefault","Employed","CreditScore","Dr
```

[6]:
```
dataset.columns = columns
```

[7]:
```
dataset.head()
```

[7]:

	Gender	Age	Debt	Married	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	ZipCode	Income	Approved
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

replacing ? with nan

[13]:
```
dataset.replace('?', np.NaN, inplace=True)
```

[14]:
```
dataset.isnull().sum()
```

[14]:
```
Gender            12
Age               12
Debt               0
Married            6
BankCustomer       6
EducationLevel     9
Ethnicity          9
YearsEmployed      0
PriorDefault       0
Employed           0
CreditScore        0
DriversLicense     0
Citizen            0
ZipCode           13
Income             0
Approved           0
dtype: int64
```

[15]:
```
dataset['Gender'].value_counts()
```

[15]:
```
b    468
```

[19]:
```
## Here we will check the percentage of nan values present in each feature
## 1 -step make the list of features which has missing values
features_with_na=[features for features in dataset.columns if dataset[features].isnull().sum()>1]
## 2- step print the feature name and the percentage of missing values

for feature in features_with_na:
    print(feature, np.round(dataset[feature].isnull().mean(), 4),  ' % missing values')
```

```
Gender 0.0174  % missing values
Age 0.0174  % missing values
Married 0.0087  % missing values
BankCustomer 0.0087  % missing values
EducationLevel 0.013  % missing values
Ethnicity 0.013  % missing values
ZipCode 0.0188  % missing values
```

We replaced all the question marks with NaNs. This is going to help us in the next missing value treatment that we are going to perform.

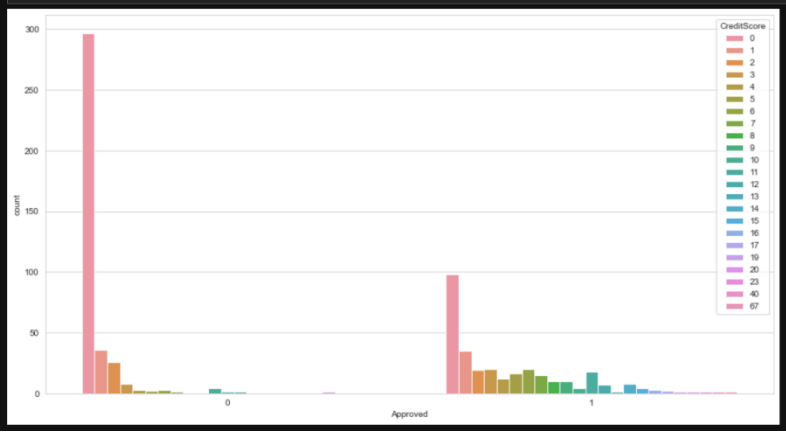
An important question that gets raised here is why are we giving so much importance to missing values? Can't they be just ignored? Ignoring missing values can affect the performance of a machine learning model heavily. While ignoring the missing values our machine learning model may miss out on information about the dataset that may be useful for its training. Then, there are many models which cannot handle missing values implicitly.

So, to avoid this problem, we are going to impute the missing values with a strategy called mean imputation.

[20]:
```
dataset['Age']=dataset['Age'].astype(float)
```

[21]:
```
dataset.info()
```

[36]:
```
for feature in discrete_feature:
    data=dataset.copy()
    plt.figure(figsize=(15,8))
    sns.set_style('whitegrid')
    sns.countplot(x='Approved',hue=feature,data=data)
```



[63]:
```
def detect_outliers(data):
    outliers=[]
    threshold=3
    mean = np.mean(data)
    std =np.std(data)


    for i in range(len(data)):
        z_score= (data[i] - mean)/std
        if np.abs(z_score) > threshold:
            outliers.append(i)
    return outliers
```

[64]:
```
outlier_pt=detect_outliers(data['Age'])
print(outlier_pt)
```

```
[130, 157, 206, 296, 405, 485, 539, 550, 585]
```

[65]:
```
outlier_pt1=detect_outliers(data['Debt'])
print(outlier_pt1)
```

```
[44, 69, 234, 241, 250, 305, 317, 529, 550, 586]
```

[66]:
```
outlier_pt2=detect_outliers(data['Income'])
print(outlier_pt2)
```

```
[]
```

handling 0 values

[79]:
```
dataset[dataset['Income']==0].shape[0]
```

[79]:
```
295
```

[80]:
```
dataset[dataset['CreditScore']==0].shape[0]
```

[80]:
```
395
```

[81]:
```
dataset["CreditScore"].nunique()
```

[81]:
```
23
```

[82]:
```
data['Income']=np.where(data['Income']==0,data['Income'].mean(),data['Income'])
data.head()
```

[82]:

	Gender_b	Married_l	Married_u	Married_y	BankCustomer_g	BankCustomer_gg	BankCustomer_p	PriorDefault_t	Employed_t	DriversLicense_t	Citizen_p	Citizen_s	Age	Debt	EducationLevel	Ethnicity	YearsEmployed	Cre
0	1	0	1	0	1	0	0	1	1	0	0	0	30.83	0.000	0.515625	0.423559	1.25	
1	0	0	1	0	1	0	0	1	1	0	0	0	58.67	4.460	0.653846	0.630435	3.04	
2	0	0	1	0	1	0	0	1	0	0	0	0	24.50	0.500	0.653846	0.630435	1.50	
3	1	0	1	0	1	0	0	1	1	1	0	0	27.83	1.540	0.515625	0.423559	3.75	
4	1	0	1	0	1	0	0	1	0	0	0	1	20.17	5.625	0.515625	0.423559	1.71	

[83]:
```
data.to_csv('feature-engineering.csv', index = False)
```

Feature Selection

Univariate Selection

[8]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

[9]:
```
### Apply SelectKBest Algorithm
ordered_rank_features=SelectKBest(score_func=chi2,k=15)
ordered_feature=ordered_rank_features.fit(X,y)
```

[10]:
```
dfscores=pd.DataFrame(ordered_feature.scores_,columns=["Score"])
dfcolumns=pd.DataFrame(X.columns)
```

[11]:
```
features_rank=pd.concat([dfcolumns,dfscores],axis=1)
```

[12]:
```
features_rank.columns=['Features','Score']
features_rank
```

[12]:

	Features	Score
0	Gender_b	0.175810
1	Married_l	2.495114
2	Married_u	4.908390
3	Married_y	18.817889
4	BankCustomer_g	4.908390
5	BankCustomer_gg	2.495114
6	BankCustomer_p	18.817889
7	PriorDefault_t	170.746388

Feature Importance

[14]:
```
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model=ExtraTreesClassifier()
model.fit(X,y)
```

[14]:
```
ExtraTreesClassifier()
```

[15]:
```
print(model.feature_importances_)
```

```
[0.01796896 0.00223785 0.00678839 0.00625776 0.00557448 0.00267097
 0.00763106 0.35663928 0.07549839 0.02085268 0.00599864 0.01473194
 0.06499268 0.06829546 0.08091447 0.04091068 0.07608249 0.07574094
 0.07021291]
```

[16]:
```
ranked_features=pd.Series(model.feature_importances_,index=X.columns)
lst=ranked_features.nlargest(10).plot(kind='barh')
plt.show()
lst=ranked_features.nlargest(6)
print(lst)
```

Multicollinearity

- variance inflation factor
- correlation

variance inflation factor

[17]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF Factor"]=[variance_inflation_factor(X.values,i) for i in range(X.shape[1])]
vif["features"]=X.columns
#Let's check the values
vif
```

[17]:

	VIF Factor	features
0	3.412023	Gender_b
1	inf	Married_l
2	inf	Married_u
3	inf	Married_y
4	inf	BankCustomer_g
5	inf	BankCustomer_gg
6	inf	BankCustomer_p
7	3.360099	PriorDefault_t
8	3.853341	Employed_t
9	1.952504	DriversLicense_t

correlation

```
[18]: import seaborn as sns
      corr=df.iloc[:,:-1].corr()
      top_features=corr.index
      plt.figure(figsize=(20,20))
      sns.heatmap(df[top_features].corr(),annot=True)
```

```
[18]: <AxesSubplot:>
```

Information Gain

```
[22]: from sklearn.feature_selection import mutual_info_classif
```

```
[23]: mutual_info=mutual_info_classif(X,y)
```

```
[24]: mutual_data=pd.Series(mutual_info,index=X.columns)
      mutual_data.sort_values(ascending=False)
```

```
[24]: PriorDefault_t      0.297249
      CreditScore         0.167713
      Employed_t          0.108918
      Income              0.083886
      EducationLevel      0.077240
      YearsEmployed       0.073430
      Ethnicity           0.051324
      Married_u           0.029247
      BankCustomer_gg     0.022781
      Citizen_p           0.020874
      BankCustomer_p      0.020765
      Debt                0.017180
      Married_y           0.012308
      Age                 0.007701
      Gender_b            0.000000
      Citizen_s           0.000000
      Married_l           0.000000
      BankCustomer_g      0.000000
      DriversLicense_t    0.000000
      dtype: float64
```

```
      Gender_b            0.000000
      Citizen_s           0.000000
      Married_l           0.000000
      BankCustomer_g      0.000000
      DriversLicense_t    0.000000
      dtype: float64
```

```
[25]: ## Not including Employed because Income is same as Employed
      df = df[['PriorDefault_t','YearsEmployed','CreditScore','Income','Approved']]
      df.head()
```

[25]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

```
[26]: df.to_csv('feature_selection.csv',index=False)
```

Training with KNN

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
df=pd.read_csv('feature_Selection.csv')
df.head()
```

[3]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

[4]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop('Approved',axis=1))
```

[5]:
```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

[5]:

	PriorDefault_t	YearsEmployed	CreditScore	Income

[6]:
```
#### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(scaled_features,df['Approved'],test_size=0.20,random_state=0)
```

[7]:
```
y_test.shape
```

[7]:
```
(138,)
```

[8]:
```
y_test.value_counts()
```

[8]:
```
0    78
1    60
Name: Approved, dtype: int64
```

[9]:
```
y_train.value_counts()
```

[9]:
```
0    305
1    247
Name: Approved, dtype: int64
```

[10]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier=KNeighborsClassifier(n_neighbors=1).fit(X_train,y_train)
prediction=knn_classifier.predict(X_test)
```

[11]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,roc_curve, roc_auc_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
print(f'Accuracy Score : {accuracy_score(y_test,prediction)}')
print(classification_report(y_test,prediction))
```

```
confusion matrix : [[65 13]
 [15 45]]
```

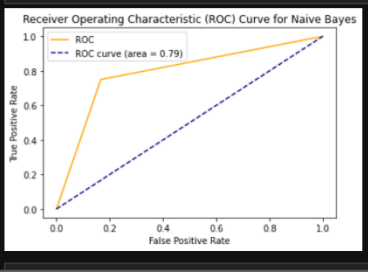
```
Name: Approved, dtype: int64
```

[13]:
```
auc = roc_auc_score(y_test, prediction)
auc
```

[13]:
```
0.7916666666666667
```

[14]:
```
fpr, tpr, thresholds = roc_curve(y_test, prediction)
```

[15]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
plt.legend()
plt.show()
```



[16]:
```
from sklearn.model_selection import RandomizedSearchCV
random_grid = { 'algorithm' : ['ball_tree', 'kd_tree', 'brute'],
               'leaf_size' : [18,20,25,27,30,32,34],
               'n_neighbors' : [3,5,7,9,10,11,12,13]
              }
```

[17]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
Classifier=KNeighborsClassifier()
```

[18]:
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
knn_random = RandomizedSearchCV(estimator = Classifier, param_distributions = random_grid, cv = 5, verbose=2)
```

[19]:
```
knn_random.fit(X_train,y_train)
```

```
...
```

[20]:
```
knn_random.best_params_
```

[20]:
```
{'n_neighbors': 5, 'leaf_size': 34, 'algorithm': 'kd_tree'}
```

[21]:
```
best_random_grid=knn_random.best_estimator_
```

[22]:
```
from sklearn.metrics import accuracy_score,accuracy_score,roc_curve, roc_auc_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

[28]:
```
knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print(accuracy_score(y_test,pred))
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
0.8913043478260869


[[85  1]
 [14 38]]


              precision    recall  f1-score   support

           0       0.86      0.99      0.92        86
```

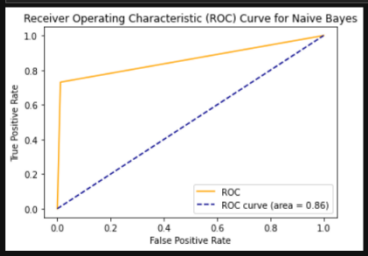
[29]:
```
# Area Under Curve
auc = roc_auc_score(y_test, pred)
auc
```

[29]:
```
0.8595706618962433
```

[30]:
```
fpr, tpr, thresholds = roc_curve(y_test, pred)
```

[30]:
```
fpr, tpr, thresholds = roc_curve(y_test, pred)
```

[31]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
plt.legend()
plt.show()
```



[32]:
```
import pickle
# save the model to disk
filename = 'finalized_model_knn.sav'
pickle.dump(knn,open(filename,'wb'))
```

Training with Naive Bayes

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
df=pd.read_csv('feature_Selection.csv')
df.head()
```

[3]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

[4]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop('Approved',axis=1))
```

[5]:
```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

[5]:

	PriorDefault_t	YearsEmployed	CreditScore	Income

[6]:
```
#### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(scaled_features,df['Approved'],test_size=0.20,random_state=10)
```

[7]:
```
from sklearn.naive_bayes import GaussianNB
NB=GaussianNB().fit(X_train,y_train)
prediction=NB.predict(X_test)
```

[8]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,roc_curve, roc_auc_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
print(f'Accuracy Score : {accuracy_score(y_test,prediction)}')
print(classification_report(y_test,prediction))
```

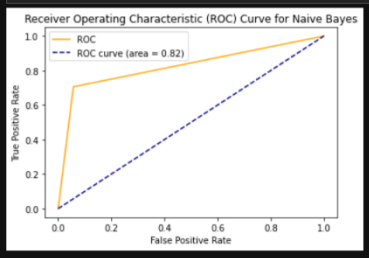
```
confusion matrix : [[66  4]
 [20 48]]
Accuracy Score : 0.8260869565217391
              precision    recall  f1-score   support

           0       0.77      0.94      0.85        70
           1       0.92      0.71      0.80        68

    accuracy                           0.83       138
   macro avg       0.85      0.82      0.82       138
weighted avg       0.84      0.83      0.82       138
```

[9]:
```
# Area Under Curve
auc = roc_auc_score(y_test, prediction)
auc
```

[9]:
```
0.8243697478991597
```

[10]:
```
fpr, tpr, thresholds = roc_curve(y_test, prediction)
```

[11]:
```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve for Naive Bayes')
plt.legend()
plt.show()
```



[12]:
```
import pickle
# save the model to disk
filename = 'finalized_model_nb.sav'
pickle.dump(NB, open(filename, 'wb'))
```

Training with Decision Tree

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
df=pd.read_csv('feature_Selection.csv')
df.head()
```

[3]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

[4]:
```
#### Independent And Dependent features
X=df.drop('Approved',axis=1)
y=df['Approved']
```

[5]:
```
#### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

[6]:
```
from sklearn.tree import DecisionTreeClassifier
```

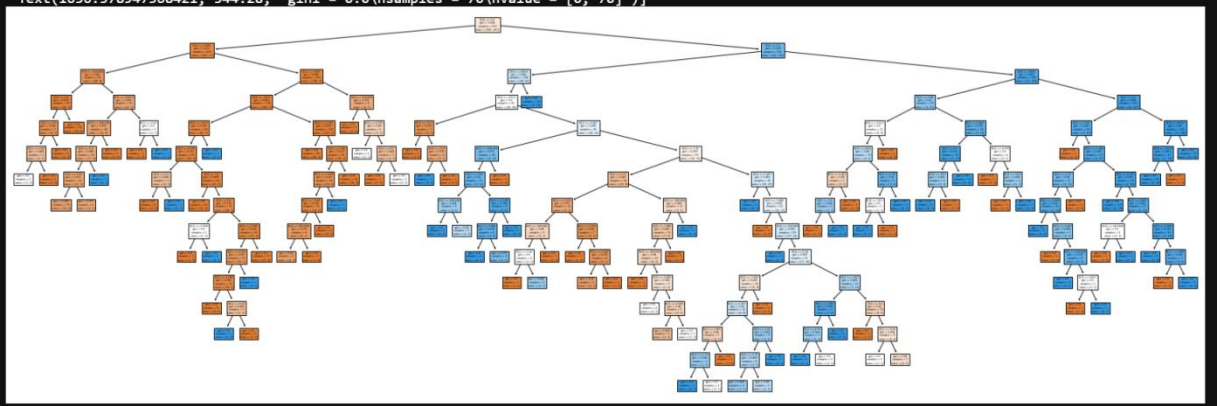
[6]:
```
from sklearn.tree import DecisionTreeClassifier
df_classifier=DecisionTreeClassifier().fit(X_train,y_train)
prediction=df_classifier.predict(X_test)
```

[7]:
```
from sklearn import tree
```

[8]:
```
plt.figure(figsize=(30,10))
tree.plot_tree(df_classifier,filled=True)
```

[8]:
```
[Text(670.4259868421053, 525.48, 'X[0] <= 0.5\ngini = 0.494\nsamples = 552\nvalue = [305, 247]'),
 Text(268.72105263157897, 489.24, 'X[1] <= 0.062\ngini = 0.123\nsamples = 259\nvalue = [242, 17]'),
 Text(114.53684210526316, 453.0, 'X[3] <= 310.846\ngini = 0.245\nsamples = 56\nvalue = [48, 8]'),
 Text(70.48421052631579, 416.76, 'X[3] <= 3.0\ngini = 0.074\nsamples = 26\nvalue = [25, 1]'),
 Text(52.863157894736844, 380.52, 'X[2] <= 0.5\ngini = 0.245\nsamples = 7\nvalue = [6, 1]'),
 Text(35.242105263157896, 344.28, 'X[1] <= 0.02\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(17.621052631578948, 308.04, 'gini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(52.863157894736844, 308.04, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(70.48421052631579, 344.28, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(88.10526315789474, 380.52, 'gini = 0.0\nsamples = 19\nvalue = [19, 0]'),
 Text(158.58947368421053, 416.76, 'X[3] <= 1347.0\ngini = 0.358\nsamples = 30\nvalue = [23, 7]'),
 Text(123.34736842105264, 380.52, 'X[3] <= 336.346\ngini = 0.337\nsamples = 28\nvalue = [22, 6]'),
 Text(105.72631578947369, 344.28, 'X[2] <= 0.5\ngini = 0.386\nsamples = 23\nvalue = [17, 6]'),
 Text(88.10526315789474, 308.04, 'X[1] <= 0.02\ngini = 0.351\nsamples = 22\nvalue = [17, 5]'),
 Text(70.48421052631579, 271.8, 'gini = 0.346\nsamples = 18\nvalue = [14, 4]'),
 Text(105.72631578947369, 271.8, 'gini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(123.34736842105264, 308.04, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(140.96842105263158, 344.28, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(193.83157894736843, 380.52, 'X[1] <= 0.02\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(176.21052631578948, 344.28, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(211.45263157894738, 344.28, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
```

```
 Text(1638.7578947368422, 380.52, 'X[3] <= 357.0\ngini = 0.02\nsamples = 101\nvalue = [1, 100]'),
 Text(1621.1368421052632, 344.28, 'X[3] <= 337.0\ngini = 0.062\nsamples = 31\nvalue = [1, 30]'),
 Text(1603.5157894736842, 308.04, 'gini = 0.0\nsamples = 30\nvalue = [0, 30]'),
 Text(1638.7578947368422, 308.04, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(1656.378947368421, 344.28, 'gini = 0.0\nsamples = 70\nvalue = [0, 70]')]
```



[9]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
```

[9]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
print(f'Accuracy Score : {accuracy_score(y_test,prediction)}')
print(classification_report(y_test,prediction))
```

```
confusion matrix : [[67 11]
 [16 44]]
Accuracy Score : 0.8043478260869565
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        78
           1       0.80      0.73      0.77        60

    accuracy                           0.80       138
   macro avg       0.80      0.80      0.80       138
weighted avg       0.80      0.80      0.80       138
```

[10]:
```
from sklearn.model_selection import RandomizedSearchCV
random_grid={
 "splitter"    : ["best","random"] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_samples_leaf" : [ 1,2,3,4,5 ],
 "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4],
 "max_features" : ["auto","log2","sqrt",None ],
 "max_leaf_nodes":[None,10,20,30,40,50,60,70]
}
```

[11]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
Classifier=DecisionTreeClassifier()
```

[12]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
df_random = RandomizedSearchCV(estimator = Classifier, param_distributions = random_grid, n_iter = 100, cv = 5, verbose=2, random_state=10, n_jobs
```

[13]:
```
df_random.fit(X_train,y_train)
```

```
...
```

[14]:
```
df_random.best_params_
```

[14]:
```
{'splitter': 'random',
 'min_weight_fraction_leaf': 0.3,
 'min_samples_leaf': 4,
 'max_leaf_nodes': None,
 'max_features': None,
 'max_depth': 6}
```

[15]:
```
best_random_grid=df_random.best_estimator_
```

[16]:
```
print(best_random_grid)
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=4,
                       min_weight_fraction_leaf=0.3, splitter='random')
```

[15]:
```
best_random_grid=df_random.best_estimator_
```

[16]:
```
print(best_random_grid)
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=4,
                       min_weight_fraction_leaf=0.3, splitter='random')
```

[17]:
```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score: {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[64 14]
 [ 6 54]]
Accuracy Score: 0.855072463768116
Classification report:               precision    recall  f1-score   support

           0       0.91      0.82      0.86        78
           1       0.79      0.90      0.84        60

    accuracy                           0.86       138
   macro avg       0.85      0.86      0.85       138
weighted avg       0.86      0.86      0.86       138
```

[18]:
```
import pickle
# save the model to disk
filename='finalized_model_DT.sav'
pickle.dump(df_random, open(filename, 'wb'))
```

Training with Random Forest and Hyper parameter tuning

All Techniques Of Hyper Parameter Optimization

1.GridSearchCV
2.RandomizedSearchCV
3.Bayesian Optimization -Automate Hyperparameter Tuning (Hyperopt)
4.Sequential Model Based Optimization(Tuning a scikit-learn estimator with skopt)
5.Optuna- Automate Hyperparameter Tuning
6.Genetic Algorithms (TPOT Classifier)

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
df=pd.read_csv('feature_Selection.csv')
df.head()
```

[2]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

[3]:
```
#### Independent And Dependent features
X=df.drop('Approved',axis=1)
y=df['Approved']
```

[4]:
```
#### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

[5]:
```
from sklearn.ensemble import RandomForestClassifier
rf_classifier=RandomForestClassifier(n_estimators=10).fit(X_train,y_train)
prediction=rf_classifier.predict(X_test)
```

[6]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
print(f'Accuracy Score : {accuracy_score(y_test,prediction)}')
print(classification_report(y_test,prediction))
```

```
confusion matrix : [[67 11]
 [14 46]]
Accuracy Score : 0.8188405797101449
              precision    recall  f1-score   support

           0       0.83      0.86      0.84        78
           1       0.81      0.77      0.79        60

    accuracy                           0.82       138
   macro avg       0.82      0.81      0.81       138
weighted avg       0.82      0.82      0.82       138
```

Manual Hyperparameter Tuning

[7]:
```
### Manual Hyperparameter Tuning
model=RandomForestClassifier(n_estimators=300,criterion='entropy',
                             max_features='sqrt',min_samples_leaf=10,random_state=100).fit(X_train,y_train)
predictions=model.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(accuracy_score(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[67 11]
 [10 50]]
0.8478260869565217
              precision    recall  f1-score   support

           0       0.87      0.86      0.86        78
           1       0.82      0.83      0.83        60

    accuracy                           0.85       138
   macro avg       0.84      0.85      0.85       138
weighted avg       0.85      0.85      0.85       138
```

Randomized Search Cv

[8]:
```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 120, 230, 340, 450, 560, 670, 780, 890, 1000], 'min_samples_split': [2, 5, 10, 14], 'min_samples_leaf': [1, 2, 4, 6, 8], 'criterion': ['entropy', 'gini']}
```

[9]:
```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,param_distributions=random_grid,n_iter=100,cv=3,verbose=2,
                               random_state=100,n_jobs=-1)
```

[10]:
```
rf_randomcv.best_params_
```

[10]:
```
{'n_estimators': 1400,
 'min_samples_split': 10,
 'min_samples_leaf': 8,
 'max_features': 'auto',
 'max_depth': 450,
 'criterion': 'gini'}
```

[11]:
```
rf_randomcv
```

[11]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [10, 120, 230, 340, 450,
                                                      560, 670, 780, 890,
                                                      1000],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4, 6, 8],
                                        'min_samples_split': [2, 5, 10, 14],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=100, verbose=2)
```

[12]:
```
best_random_grid=rf_randomcv.best_estimator_
```

[13]:
```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
```

```
[[69  9]
 [10 50]]
Accuracy Score 0.8623188405797102
Classification report:              precision    recall  f1-score   support

           0       0.87      0.88      0.88        78
           1       0.85      0.83      0.84        60

    accuracy                           0.86       138
   macro avg       0.86      0.86      0.86       138
weighted avg       0.86      0.86      0.86       138
```

GridSearch CV

[14]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf']+4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split']-2,
                          rf_randomcv.best_params_['min_samples_split']-1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split']+1,
                          rf_randomcv.best_params_['min_samples_split']+2],
```

[15]:
```
#### Fit the grid_search to the data
rf=RandomForestClassifier()
grid_search=GridSearchCV(estimator=rf,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(X_train,y_train)
```

```
Fitting 10 folds for each of 75 candidates, totalling 750 fits
```

[15]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini'], 'max_depth': [450],
                         'max_features': ['auto'],
                         'min_samples_leaf': [8, 10, 12],
                         'min_samples_split': [8, 9, 10, 11, 12],
                         'n_estimators': [1200, 1300, 1400, 1500, 1600]},
             verbose=2)
```

[16]:
```
grid_search.best_estimator_
```

[16]:
```
RandomForestClassifier(max_depth=450, min_samples_leaf=10, min_samples_split=9,
                       n_estimators=1200)
```

[17]:
```
best_grid=grid_search.best_estimator_
```

[18]:
```
best_grid
```

[18]:
```
RandomForestClassifier(max_depth=450, min_samples_leaf=10, min_samples_split=9,
                       n_estimators=1200)
```

[19]:
```
y_pred=best_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[68 10]
```

Automated Hyperparameter Tuning

Automated Hyperparameter Tuning can be done by using techniques such as

- Bayesian Optimization
- Gradient Descent
- Evolutionary Algorithms

Bayesian Optimization

Bayesian optimization uses probability to find the minimum of a function. The final aim is to find the input value to a function which can gives us the lowest possible output value.It usually performs better than random,grid and manual search providing better performance in the testing phase and reduced optimization time. In Hyperopt, Bayesian Optimization can be implemented giving 3 three main parameters to the function fmin.

- Objective Function = defines the loss function to minimize.
- Domain Space = defines the range of input values to test (in Bayesian Optimization this space creates a probability distribution for each of the used Hyperparameters).
- Optimization Algorithm = defines the search algorithm to use to select the best input values to use in each new iteration.

[20]:
```
from hyperopt import hp,fmin,tpe,STATUS_OK,Trials
```

[21]:
```
space = {'criterion': hp.choice('criterion', ['entropy', 'gini']),
        'max_depth': hp.choice('max_depth',[10, 120, 230, 340, 450, 560, 670, 780, 890, 1000]),
        'max_features': hp.choice('max_features', ['auto', 'sqrt','log2', None]),
        'min_samples_leaf': hp.uniform('min_samples_leaf', 0, 0.5),
        'min_samples_split' : hp.uniform ('min_samples_split', 0, 1),
        'n_estimators' : hp.choice('n_estimators', [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000])
    }
```

[22]:
```
def objective(space):
    model = RandomForestClassifier(criterion = space['criterion'], max_depth = space['max_depth'],
                                   max_features = space['max_features']
```

[23]:
```
from sklearn.model_selection import cross_val_score
trials = Trials()
best = fmin(fn= objective,
            space= space,
            algo= tpe.suggest,
            max_evals = 80,
            trials= trials)
best
```

```
100%|██████████| 80/80 [08:56<00:00,  6.71s/trial, best loss: -0.8585913185913185]
```

[23]:
```
{'criterion': 1,
 'max_depth': 0,
 'max_features': 1,
 'min_samples_leaf': 0.08453684676276399,
 'min_samples_split': 0.11006204582812613,
 'n_estimators': 2}
```

[24]:
```
crit = {0: 'entropy', 1: 'gini'}
depth = [10, 120, 230, 340, 450, 560, 670, 780, 890, 1000]
feat = {0: 'auto', 1: 'sqrt', 2: 'log2', 3: None}
est = {0: 200, 1: 400, 2: 600, 3: 800, 4: 1000, 5:1200, 6:1400,7:1600,8:1800,9:2000}

print(crit[best['criterion']])
print(feat[best['max_features']])
print(est[best['n_estimators']])
```

```
gini
sqrt
600
```

[25]:
```
best['min_samples_leaf']
```

[26]:
```
trainedforest = RandomForestClassifier(criterion = crit[best['criterion']], max_depth = depth[best['max_depth']],
                                       max_features = feat[best['max_features']],
                                       min_samples_leaf = best['min_samples_leaf'],
                                       min_samples_split = best['min_samples_split'],
                                       n_estimators = est[best['n_estimators']]).fit(X_train,y_train)
predictionforest = trainedforest.predict(X_test)
print(confusion_matrix(y_test,predictionforest))
print(accuracy_score(y_test,predictionforest))
print(classification_report(y_test,predictionforest))
acc5 = accuracy_score(y_test,predictionforest)
```

```
[[64 14]
 [ 6 54]]
0.855072463768116
              precision    recall  f1-score   support

           0       0.91      0.82      0.86        78
           1       0.79      0.90      0.84        60

    accuracy                           0.86       138
   macro avg       0.85      0.86      0.85       138
weighted avg       0.86      0.86      0.86       138
```

[27]:
```
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
param = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(param)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 120, 230, 3
40, 450, 560, 670, 780, 890, 1000], 'min_samples_split': [2, 5, 10, 14], 'min_samples_leaf': [1, 2, 4, 6, 8], 'criterion': ['entropy', 'gini']}
```

[28]:
```
from tpot import TPOTClassifier

tpot_classifier = TPOTClassifier(generations= 5, population_size= 24, offspring_size= 12,
                                 verbosity= 2, early_stop= 12,
                                 config_dict={'sklearn.ensemble.RandomForestClassifier': param},
                                 cv = 4, scoring = 'accuracy')
```

```
                population_size=24, scoring='accuracy', verbosity=2)
```

[29]:
```
accuracy = tpot_classifier.score(X_test, y_test)
print(accuracy)
```

```
0.8478260869565217
```

Optimize hyperparameters of the model using Optuna

The hyperparameters of the above algorithm are n_estimators and max_depth for which we can try different values to see if the model accuracy can be improved. The objective function is modified to accept a trial object. This trial has several methods for sampling hyperparameters. We create a study to run the hyperparameter optimization and finally read the best hyperparameters.

[30]:
```
import optuna
import sklearn.svm
def objective(trial):
    classifier = trial.suggest_categorical('classifier', ['RandomForest', 'SVC'])

    if classifier == 'RandomForest':
        n_estimators = trial.suggest_int('n_estimators', 200, 2000,10)
        max_depth = int(trial.suggest_float('max_depth', 10, 100, log=True))

        clf = sklearn.ensemble.RandomForestClassifier(
            n_estimators=n_estimators, max_depth=max_depth)
    else:
        c = trial.suggest_float('svc_c', 1e-10, 1e10, log=True)
        clf = sklearn.svm.SVC(C=c, gamma='auto')
    return sklearn.model_selection.cross_val_score(
        clf,X_train,y_train, n_jobs=-1, cv=3).mean()
```

[31]:
```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

trial = study.best_trial

print('Accuracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

```
[I 2021-11-07 09:21:53,119] A new study created in memory with name: no-name-d1f2e607-527d-4c27-ad91-968e601eb6d4
[I 2021-11-07 09:21:58,012] Trial 0 finished with value: 0.8278985507246377 and parameters: {'classifier': 'RandomForest', 'n_estimators': 1860,
'max_depth': 52.831636492821204}. Best is trial 0 with value: 0.8278985507246377.
[I 2021-11-07 09:22:03,753] Trial 1 finished with value: 0.6721014492753623 and parameters: {'classifier': 'SVC', 'svc_c': 274357.6778465932}. Bes
t is trial 0 with value: 0.8278985507246377.
[I 2021-11-07 09:22:06,332] Trial 2 finished with value: 0.8333333333333334 and parameters: {'classifier': 'RandomForest', 'n_estimators': 700, 'm
ax_depth': 66.28649973761492}. Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:22:07,029] Trial 3 finished with value: 0.8278985507246377 and parameters: {'classifier': 'RandomForest', 'n_estimators': 420, 'm
ax_depth': 21.09713286033926}. Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:45,166] Trial 4 finished with value: 0.5887681159420289 and parameters: {'classifier': 'SVC', 'svc_c': 5861527765.6813545}. Be
st is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:45,223] Trial 5 finished with value: 0.7101449275362318 and parameters: {'classifier': 'SVC', 'svc_c': 37.30576591726301}. Bes
t is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:45,261] Trial 6 finished with value: 0.552536231884058 and parameters: {'classifier': 'SVC', 'svc_c': 2.091714942429609e-08}.
Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:47,807] Trial 7 finished with value: 0.8315217391304347 and parameters: {'classifier': 'RandomForest', 'n_estimators': 1620,
'max_depth': 17.311885144567185}. Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:50,104] Trial 8 finished with value: 0.8278985507246377 and parameters: {'classifier': 'RandomForest', 'n_estimators': 1450,
'max_depth': 10.254902923002449}. Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:50,738] Trial 9 finished with value: 0.8333333333333334 and parameters: {'classifier': 'RandomForest', 'n_estimators': 380, 'm
ax_depth': 98.70403999854032}. Best is trial 2 with value: 0.8333333333333334.
[I 2021-11-07 09:25:52,429] Trial 10 finished with value: 0.8278985507246377 and parameters: {'classifier': 'RandomForest', 'n_estimators': 1000,
'max_depth': 54.859407243484796}. Best is trial 2 with value: 0.8333333333333334.
```

```
Accuracy: 0.8333333333333334
Best hyperparameters: {'classifier': 'RandomForest', 'n_estimators': 700, 'max_depth': 66.28649973761492}
```

[32]:
```
trial
```

[32]:
```
FrozenTrial(number=2, values=[0.8333333333333334], datetime_start=datetime.datetime(2021, 11, 7, 9, 22, 3, 754986), datetime_complete=datetime.dat
etime(2021, 11, 7, 9, 22, 6, 331670), params={'classifier': 'RandomForest', 'n_estimators': 700, 'max_depth': 66.28649973761492}, distributions=
{'classifier': CategoricalDistribution(choices=('RandomForest', 'SVC')), 'n_estimators': IntUniformDistribution(high=2000, low=200, step=10), 'max
_depth': LogUniformDistribution(high=100.0, low=10.0)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=2, state=TrialState.COMPL
ETE, value=None)
```

[33]:
```
study.best_params
```

[33]:
```
{'classifier': 'RandomForest',
 'n_estimators': 700,
 'max_depth': 66.28649973761492}
```

[38]:
```
rf=RandomForestClassifier(n_estimators=study.best_params['n_estimators'],max_depth=study.best_params['max_depth'])
rf.fit(X_train,y_train)
```

[38]:
```
RandomForestClassifier(max_depth=66.28649973761492, n_estimators=700)
```

[39]:
```
y_pred=rf.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[66 12]
 [14 46]]
0.8115942028985508
              precision    recall  f1-score   support
```

[39]:
```
y_pred=rf.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[66 12]
 [14 46]]
0.8115942028985508
              precision    recall  f1-score   support

           0       0.82      0.85      0.84        78
           1       0.79      0.77      0.78        60

    accuracy                           0.81       138
   macro avg       0.81      0.81      0.81       138
weighted avg       0.81      0.81      0.81       138
```

[40]:
```
import pickle
# save the model to disk
filename = 'finalized_model_RF.sav'
pickle.dump(rf_randomcv, open(filename, 'wb'))
```

Training with Xgboost

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
import pandas as pd
df=pd.read_csv('feature_Selection.csv')
df.head()
```

[3]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

[4]:
```
#### Independent And Dependent features
X=df.drop('Approved',axis=1)
y=df['Approved']
```

[5]:
```
#### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

[9]:
```
import xgboost as xgb
xgb_classifier=xgb.XGBClassifier().fit(X_train,y_train)
prediction=xgb_classifier.predict(X_test)
```

```
[16:25:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
ation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.
```

[10]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(f'confusion matrix : {confusion_matrix(y_test,prediction)}')
print(f'Accuracy Score : {accuracy_score(y_test,prediction)}')
print(classification_report(y_test,prediction))
```

```
confusion matrix : [[66 12]
 [14 46]]
Accuracy Score : 0.8115942028985508
              precision    recall  f1-score   support

           0       0.82      0.85      0.84        78
           1       0.79      0.77      0.78        60

    accuracy                           0.81       138
   macro avg       0.81      0.81      0.81       138
weighted avg       0.81      0.81      0.81       138
```

[11]:
```
### Manual Hyperparameter Tuning
model=xgb.XGBClassifier(n_estimators=300,learning_rate =0.05,
                        max_depth=5,subsample=0.7,min_child_weight=3).fit(X_train,y_train)
predictions=model.predict(X_test)
print(confusion_matrix(y_test,predictions))
print(accuracy_score(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[16:25:50] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evalu
ation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.
[[69  9]
 [13 47]]
0.8405797101449275
              precision    recall  f1-score   support

           0       0.84      0.88      0.86        78
           1       0.84      0.78      0.81        60

    accuracy                           0.84       138
   macro avg       0.84      0.83      0.84       138
weighted avg       0.84      0.84      0.84       138
```

Randomized Search CV

[12]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Various learning rate parameters
```

[13]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
Classifier=xgb.XGBClassifier()
```

[14]:
```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
xg_random = RandomizedSearchCV(estimator = Classifier, param_distributions = random_grid,scoring='roc_auc', n_iter = 100, cv = 5, verbose=2, rando
```

[15]:
```
xg_random.fit(X_train,y_train)
```

•••

[17]:
```
xg_random.best_params_
```

[17]:
```
{'subsample': 0.6,
 'n_estimators': 100,
 'min_child_weight': 6,
 'max_depth': 15,
 'learning_rate': '0.05'}
```

[18]:
```
best_random_grid=xg_random.best_estimator_
```

[19]:
```
from sklearn.metrics import accuracy_score
y_pred=best_random_grid.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(y_test,y_pred)))
print("Classification report: {}".format(classification_report(y_test,y_pred)))
```

```
[[69  9]
 [11 49]]
Accuracy Score 0.855072463768116
```

Bayesian Optimization

[20]:
```
from hyperopt import hp,fmin,tpe,STATUS_OK,Trials
```

[21]:
```
space = {
    'max_depth' : hp.choice('max_depth', range(5, 30, 1)),
    'learning_rate' : hp.quniform('learning_rate', 0.01, 0.5, 0.01),
    'n_estimators' : hp.choice('n_estimators', range(20, 205, 5)),
    'gamma' : hp.quniform('gamma', 0, 0.50, 0.01),
    'min_child_weight' : hp.quniform('min_child_weight', 1, 10, 1),
    'subsample' : hp.quniform('subsample', 0.1, 1, 0.01),
    'colsample_bytree' : hp.quniform('colsample_bytree', 0.1, 1.0, 0.01)}
```

[22]:
```
def objective(space):
    import xgboost as xgb
    model = xgb.XGBClassifier(n_estimators = space['n_estimators'],
                              max_depth = int(space['max_depth']),
                              learning_rate = space['learning_rate'],
                              gamma = space['gamma'],
                              min_child_weight = space['min_child_weight'],
                              subsample = space['subsample'],
                              colsample_bytree = space['colsample_bytree']
                                   )



    accuracy = cross_val_score(model, X_train, y_train, cv = 5).mean()
```

[27]:
```
from tpot import TPOTClassifier
from xgboost import XGBClassifier

tpot_classifier = TPOTClassifier(generations= 5, population_size= 24, offspring_size= 12,
                                 verbosity= 2, early_stop= 12,
                                 cv = 4, scoring = 'accuracy')
tpot_classifier.fit(X_train,y_train)
```

```
Optimization Progress:   0%|          | 0/84 [00:00<?, ?pipeline/s]

Generation 1 - Current best internal CV score: 0.8605072463768115

Generation 2 - Current best internal CV score: 0.8605072463768115

Generation 3 - Current best internal CV score: 0.8641304347826086

Generation 4 - Current best internal CV score: 0.8641304347826086

Generation 5 - Current best internal CV score: 0.8641304347826086

Best pipeline: ExtraTreesClassifier(VarianceThreshold(input_matrix, threshold=0.001), bootstrap=False, criterion=entropy, max_features=0.1, min_sa
mples_leaf=5, min_samples_split=4, n_estimators=100)
```

[27]:
```
TPOTClassifier(cv=4, early_stop=12, generations=5, offspring_size=12,
               population_size=24, scoring='accuracy', verbosity=2)
```

[28]:
```
accuracy = tpot_classifier.score(X_test, y_test)
print(accuracy)
```

```
0.855072463768116
```

[29]:
```
import pickle
# save the model to disk
```

Comparing Bag of all 5 models vs KNN

```
[1]: import pickle
     models = []
     models.append(pickle.load(open('finalized_model_nb.sav','rb')))
     models.append(pickle.load(open('finalized_model_knn.sav','rb')))
     models.append(pickle.load(open('finalized_model_DT.sav','rb')))
     models.append(pickle.load(open('finalized_model_RF.sav','rb')))
     models.append(pickle.load(open('finalized_model_xgb.sav','rb')))

[2]: import numpy as np
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()

[3]: feat = np.array([[0,1.25,0,322]])

[4]: preds = []
     for model in models:
         preds.append(model.predict(feat))

[5]: preds

[5]: [array([1], dtype=int64),
      array([1], dtype=int64),
      array([0], dtype=int64),
      array([0], dtype=int64),
      array([0], dtype=int64)]

[6]: for model in models:
         print(model)

     GaussianNB()
```

```
[7]: def predict(feat,scaled_feat):
         preds = []
         i=0
         for model in models:
             if i<2:
                 preds.append(model.predict(scaled_feat))
             else:
                 preds.append(model.predict(feat))
             i+=1
         o = preds.count(1)
         z = preds.count(0)
         if(o>z):
             return 1
         else:
             return 0

[8]: import pandas as pd
     df = pd.read_csv("feature_selection.csv")

[9]: X = df.drop(['Approved'],axis=1)
     scaled_features = scaler.fit_transform(df.drop('Approved',axis=1))
     y = df['Approved']

[10]: df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
      df_feat.head()

[10]:
```

	PriorDefault_t	YearsEmployed	CreditScore	Income
0	0.95465	-0.300502	-0.324711	-0.288910

```
[11]: pred = []
      print(X.shape[0])
      for i in range(X.shape[0]):
          feat = np.array([X.iloc[i].values])
          scaled_feat = np.array([df_feat.iloc[i].values])
          pred.append(predict(feat,scaled_feat))

      690

[12]: from sklearn.metrics import confusion_matrix,classification_report,accuracy_score,roc_curve, roc_auc_score

[13]: accuracy_score(y,np.array(pred))

[13]: 0.8724637681159421

[14]: confusion_matrix(y,np.array(pred))

[14]: array([[331,  52],
             [ 36, 271]], dtype=int64)

[15]: print(classification_report(y,np.array(pred)))

                   precision    recall  f1-score   support

                0       0.90      0.86      0.88       383
                1       0.84      0.88      0.86       307

         accuracy                           0.87       690
        macro avg       0.87      0.87      0.87       690
     weighted avg       0.87      0.87      0.87       690
```

```
[16]: model = pickle.load(open('finalized_model_knn.sav','rb'))

[17]: pred=[]
      for i in range(X.shape[0]):
          scaled_feat = np.array([df_feat.iloc[i].values])
          pred.append(model.predict(scaled_feat))

[18]: accuracy_score(y,np.array(pred))

[18]: 0.8826086956521739

[19]: confusion_matrix(y,np.array(pred))

[19]: array([[351,  32],
             [ 49, 258]], dtype=int64)

[20]: print(classification_report(y,np.array(pred)))
```

	precision	recall	f1-score	support
0	0.88	0.92	0.90	383
1	0.89	0.84	0.86	307
accuracy			0.88	690
macro avg	0.88	0.88	0.88	690
weighted avg	0.88	0.88	0.88	690

Therefore KNN with 3 neighbours has given best accuracy than bag of models.

Saving mean and standard deviations to use them for scaling

```
[1]: import pandas as pd

[2]: df = pd.read_csv("feature_selection.csv")

[3]: df.head()
```

[3]:

	PriorDefault_t	YearsEmployed	CreditScore	Income	Approved
0	1	1.25	1	321.692754	1
1	1	3.04	6	560.000000	1
2	1	1.50	0	824.000000	1
3	1	3.75	5	3.000000	1
4	1	1.71	0	321.692754	1

```
[4]: meanstd = pd.DataFrame([[df['PriorDefault_t'].mean(),df['PriorDefault_t'].std()],
                            [df['YearsEmployed'].mean(),df['YearsEmployed'].std()],
                            [df['CreditScore'].mean(),df['CreditScore'].std()],
                            [df['Income'].mean(),df['Income'].std()]],columns=["mean","std"],
                            index=["PriorDefault_t","YearsEmployed","CreditScore","Income"])

[5]: meanstd
```

[5]:

	mean	std
PriorDefault_t	0.523188	0.499824
YearsEmployed	2.040920	2.633902
CreditScore	2.156522	3.564275
Income	459.228061	476.394710

```
[6]: meanstd.reset_index(inplace=True)

[7]: meanstd.to_csv("meanstd.csv",index=False)

[8]: data = pd.read_csv("meanstd.csv")

[9]: data.head()
```

[9]:

	index	mean	std
0	PriorDefault_t	0.523188	0.499824
1	YearsEmployed	2.040920	2.633902
2	CreditScore	2.156522	3.564275
3	Income	459.228061	476.394710

```
[10]: data = pd.read_csv("meanstd.csv",index_col="index")
```

7.2 Testing

app.py

```
import numpy as np
import pandas as pd
from flask import Flask, request, jsonify, render_template
from flask_cors import CORS,cross_origin
import pickle

app = Flask(__name__)
meanstd = pd.read_csv("meanstd.csv",index_col="index")
model = pickle.load(open('finalized_model_knn.sav','rb'))

@app.route('/', methods=['GET'])
@cross_origin()
def index():
    return render_template('index.html')
```

```
def scale(lis):
    scaled = []
    ans = (lis[0]-meanstd.loc['PriorDefault_t']['mean'])/meanstd.loc['PriorDefault_t']['std']
    scaled.append(ans)
    ans = (lis[1]-meanstd.loc['YearsEmployed']['mean'])/meanstd.loc['YearsEmployed']['std']
    scaled.append(ans)
    ans = (lis[2]-meanstd.loc['CreditScore']['mean'])/meanstd.loc['CreditScore']['std']
    scaled.append(ans)
    ans = (lis[3]-meanstd.loc['Income']['mean'])/meanstd.loc['Income']['std']
    scaled.append(ans)
    # print(scaled)
    return scaled
```

```
@app.route('/predict', methods=['POST'])
@cross_origin()
def predict():
    if request.method == 'POST':
        try:
            features = [float(x) for x in request.form.values()]
            scaled_feat = scale(features)
            final_feat = [np.array(scaled_feat)]
            prediction = model.predict(final_feat)
            # print(prediction)
            if prediction==1:
                return render_template('index.html', prediction_text = "Credit Card Approved 🥳")
            else:
                return render_template('index.html', prediction_text = "Credit Card Not Approved 😞")
        except Exception as e:
            print('The Exception message is: ',e)
            return 'something is wrong'
    else:
        return render_template('index.html')

if __name__ == "__main__":
    app.run(debug=True)
```

HTML and CSS

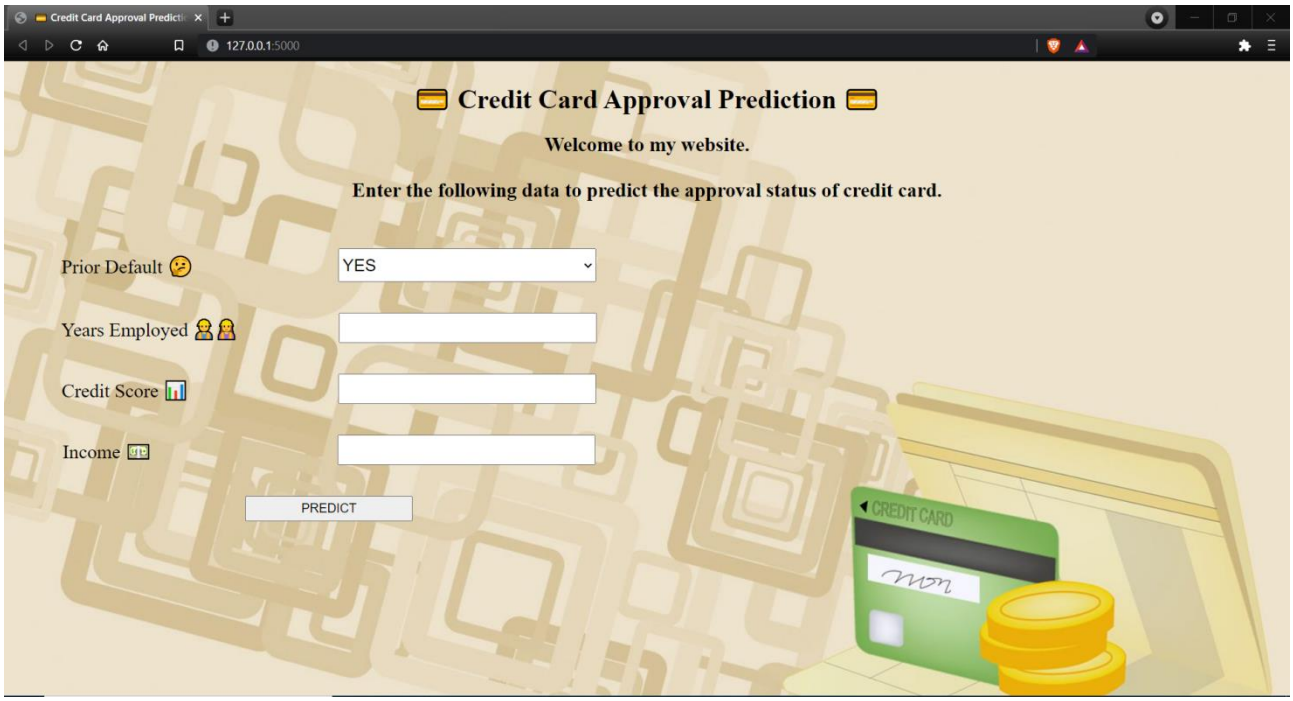
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>💳 Credit Card Approval Prediction 💳</title>
    <link rel="stylesheet" href="static/css/cssfile.css">
</head>
<body>
    <div class="login">
        <center><h1>💳 Credit Card Approval Prediction 💳</h1></center>
        <center>
            <h2>
                Welcome to my website.
                <br>
                <br>
                Enter the following data to predict the approval status of credit card.
            </h2>
        </center>
        <br><br>
        <form action="{{url_for('predict')}}" method="post" class="form-container">
            <div class="div">
                <label>
                    Prior Default 🤔         
            </label>
```

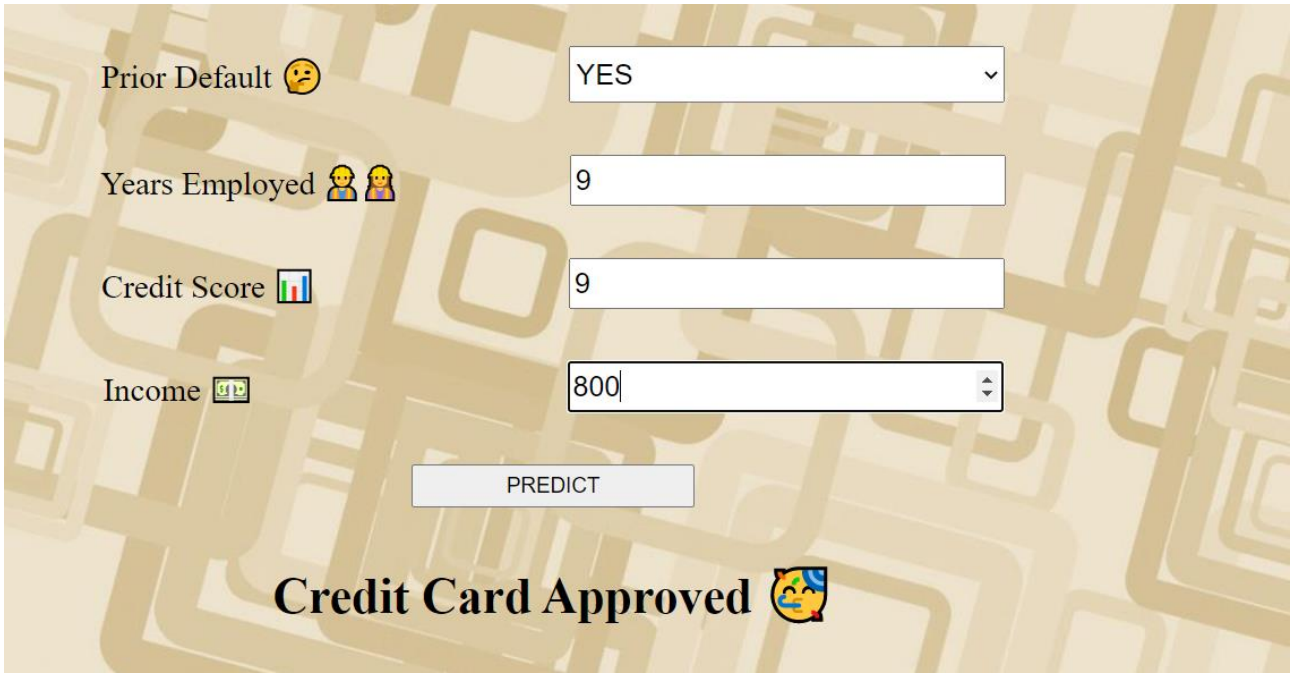
```
                                     
                </label>
                <label>
                <input type="number" min=0 max=2000 step=0.01 name="income" required="required">
                </label>
            </div>
            <br>
            <br>
            <div class="div">
            <button type="submit" class="btn btn-primary btn-block btn-large">
            PREDICT
            </button>
            </div>
            <br>
            <br>
        <div class="q">
            <strong style="font-size:35px">{{ prediction_text }}</strong>
        </div>
        </form>
        <br>
        <br>
    </div>
</body>
</html>
```

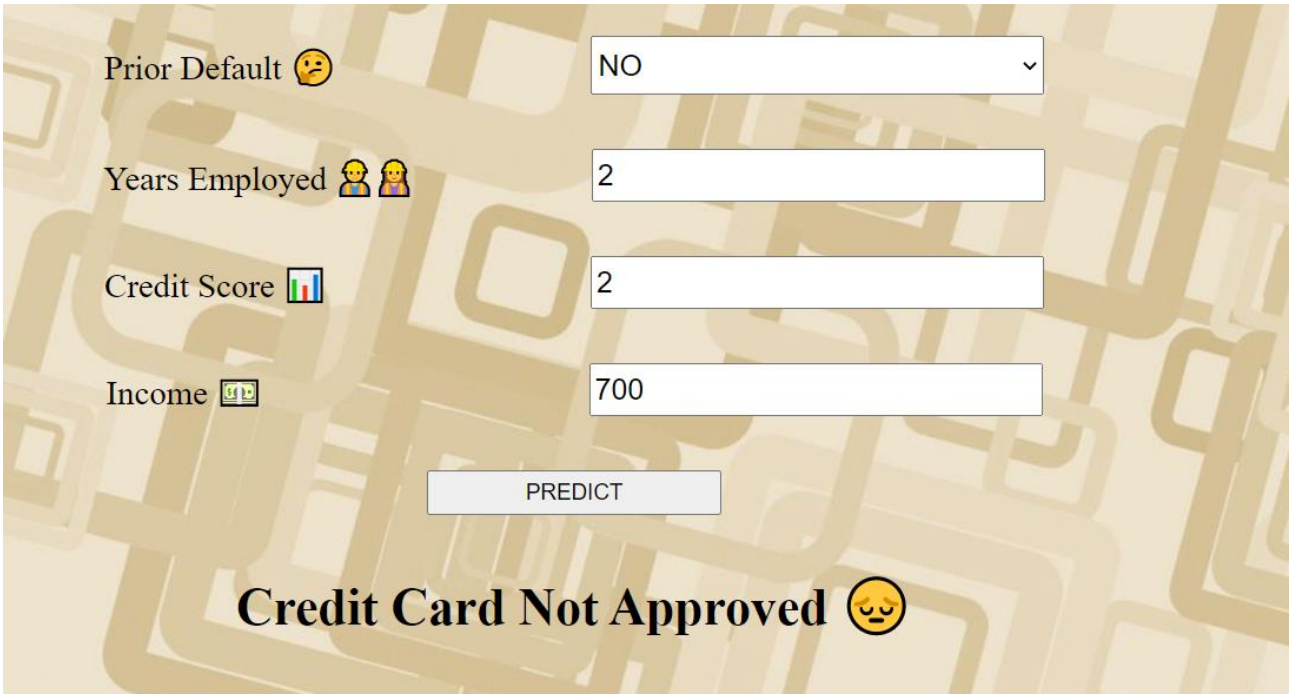
```css
body {
    background-image: url('h2.jpg');
    height: 100%;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
}
input {
    width: 300px;
    height: 30px;
    font-size: 20px;
}
select {
    width: 308px;
    height: 40px;
    font-size: 20px;
}
label {
    font-size: 23px;
    color: #000000;
}
button {
    width: 200px;
    height: 30px;
```

```css
button {
    width: 200px;
    height: 30px;
    font-size: 15px;
}
.form-container {
    width: 50%;
}
.div {
    display: flex;
    align-items: center;
    justify-content: space-around;
}
.q {
    display: flex;
    align-items: center;
    justify-content: space-around;
}
```

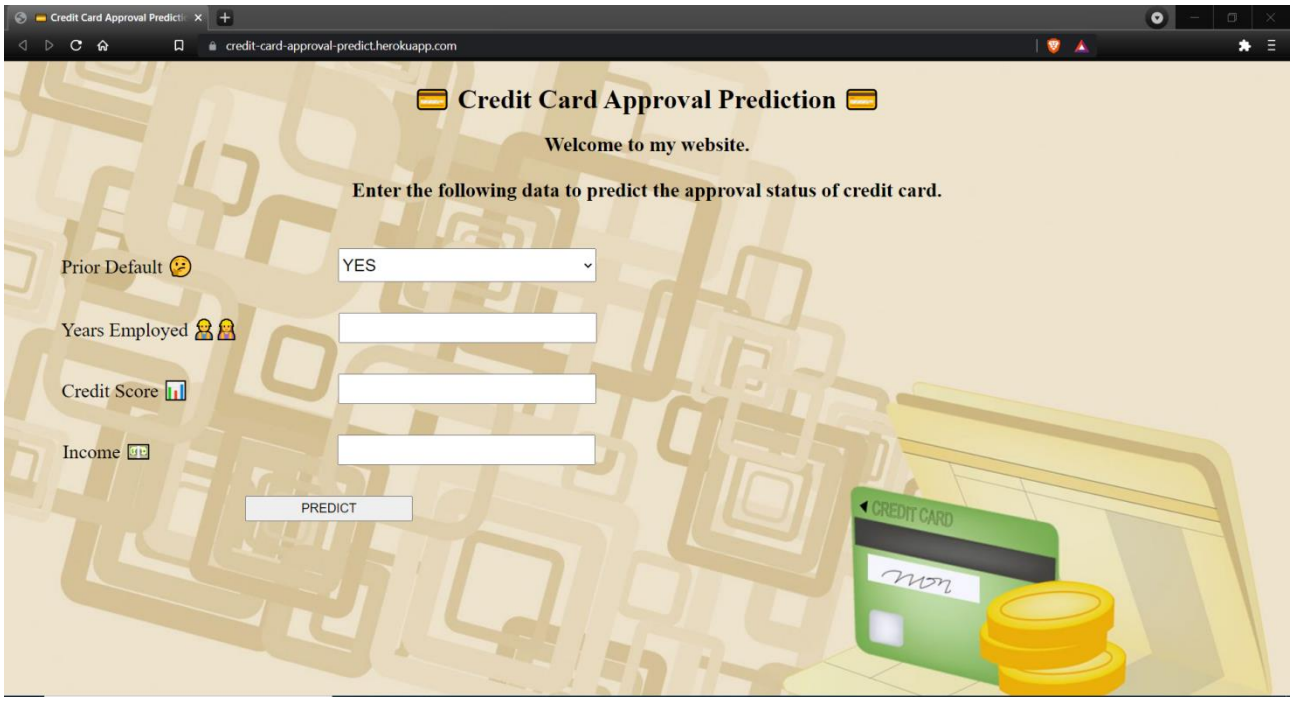
Testing in Local Host







Deployed web page



8. Results and Discussions

The accuracy of predicting the credit card is approved and credit card is not approved are almost the same 0.88 and 0.89. The classifier conservatively predicts that the status of credit card approval which will decrease a lost of work for the banks.

Performance measures of proposed algorithm

Accuracy Measure	Value
Specificity	0.91
Recall	0.84
Precision	0.88
F-Measure	0.86
Accuracy	0.88

The proposed algorithm returned an accuracy of 88.2% with performance measures: specificity, recall, precision and f-measure with values 0.91, 0.84, 0.88 and 0.86 respectively.

9. Conclusion and Future Scope

As of now the project has the models like KNN, Naive Bayes, Decision Tree, Random Forest, Xgboost. Neural networks can be used to get a better accuracy model. Neural network models can lead to give accuracies close to 98%.

10. References

A) https://www.ijrar.org/papers/IJRAR190B030.pdf
B) https://www.researchgate.net/publication/321002603_Credit_Approval_Analysis_using_R
C) https://www.ijeat.org/wp-content/uploads/papers/v9i4/D7293049420.pdf
D) https://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html
E) https://www.kaggle.com/echo9k/uci-credit-approval-data-set
F) https://archive.ics.uci.edu/ml/datasets/Credit+Approval