PENNSYLVANIA STATE UNIVERSITY

DATA MINING
STAT 557 - FALL 2022

PROJECT - 2

REPORT

Done by:

Denesh Kumar Manoharan [dzm5964@psu.edu]
Aadya Sanwal [afs6594@psu.edu]

Abstract:

In this project we are developing a classification model to determine if the customer of the bank will be able to pay his credit on time based on certain parameters like age, sex, education and his previous pay history, We will also be developing various machine learning model to perform classification and identify the model that has the highest accuracy. To perfume this classification we are using the “Default credit card clients” dataset from Kaggle dataset repository.

Exploring Data:

The dataset consists of 30000 observations with 22 features of customers. “Default payment next month” is the binary classifier that classifies if the customer can pay the credit on time represented by 1 and customer cannot pay the credit represented by 0. The 22 features of the dataset included amount of given credit, sex, education, age, repayment status of 6 month, amount of bill payment of 6 months, amount of previous payment of 6 months from September to April of 2005. This dataset does not contain any NaN values and all the features are numerical.

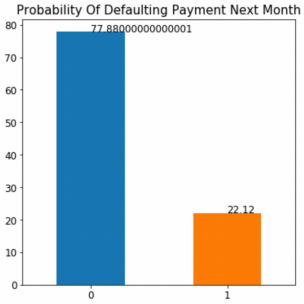


Fig1

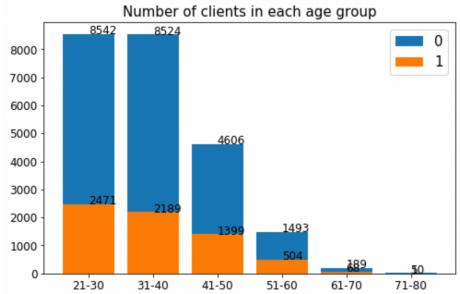


Fig2

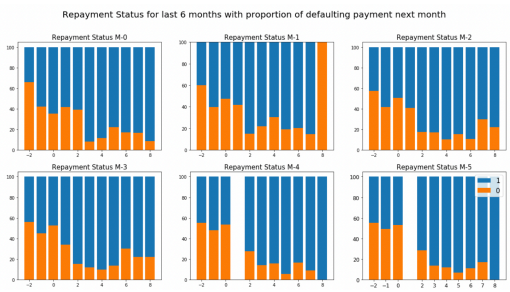


Fig3

From Fig 1 we can analyze that the dataset consists of 78% clients defaulting payment whereas 22% clients are paying their credit on time. From Fig 2 we see that most number of customers is in the age group of 21-30 and 31-40. Also we can observe that as the age increases the number of people repaying the credit on time decreases. So age is one of the key feature to perform the classification. The Fig3 plot visualizes the two classes of people defaulting payment and not default payment bases on repayment status of last 6 months.

Dimensionality Reduction:

The number of variables or features of a dataset is referred as dimensionality. Dimensionality reduction refers to a technique that reduces the number of input variables to the machine learning model. The reason for dimensionality reduction is high dimensional data may negatively affect the algorithms performances and high dimensionality might lead to poor generalization.

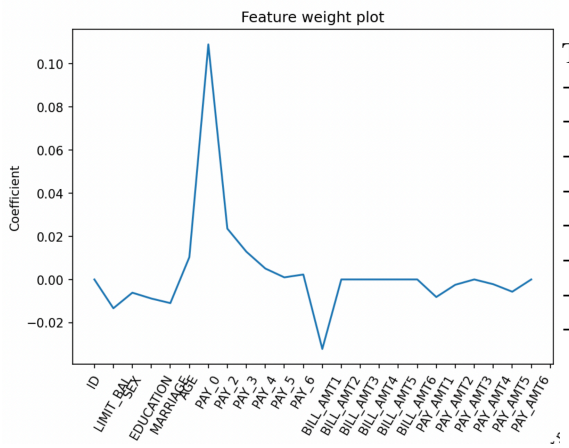
Logistic Regression + Lasso penalization for Dimensionality Reduction:

For dimensionality reduction we decided to proceed with Lasso penalization because this method Is good for feature interpretation whereas in PCA the resultant features would be combination of other feature.

Logistic Regression is a predictive analysis algorithm and based on the concept of probability. The name “logistic regression” is derived from the concept of the logistic cost function that is

uses, this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’. The value of this logistic function lies between zero and one.

The lasso regression allows us to shrink or regularize the feature coefficients to avoid overfitting and make them work better on different datasets. This type of regression is used when the dataset shows high multicollinearity or when we want to automate variable elimination and feature selection. Increasing or decreasing that regularization parameters slightly, it does regularization which with Lasso actually drops coefficients all the way to zero. And a coefficient of zero means the feature has been dropped. Before performing lasso regression on this dataset we standardized the dataset by making the mean of features to zero and variance as 1. This helps in scaling all the features that will prevent model from allocating more weight to particular feature.
We Initially performed grid search on lessor regression in our dataset to identify the best parameter of Alpha and found the best value of Alpha as [0.00170125]. On performing the lasso regression on best parameter value the features with zero coefficient is removed. This is one of the methods of performing dimensionality reduction. The coefficient of features is shown in Fig4.



The removed feature with zero coefficient:
-ID
-BILL_AMT2
-BILL_AMT3
-BILL_AMT4
-BILL_AMT5
-BILL_AMT5
-PAY_AMT3
-PAY_AMT6

Fig4

On removing these features the accuracy of logistic regression is 79.74% with a F1 score of 0.335 and precision score of 0.696. The AUC for the ROC is 0.71 in test data as seen in Fig 5.

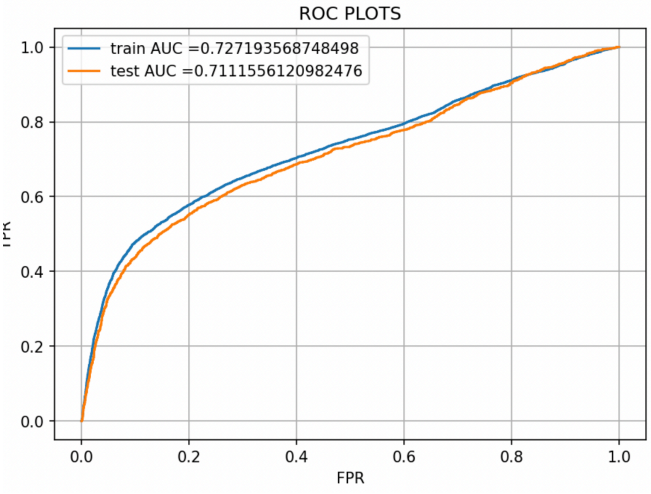


Fig5

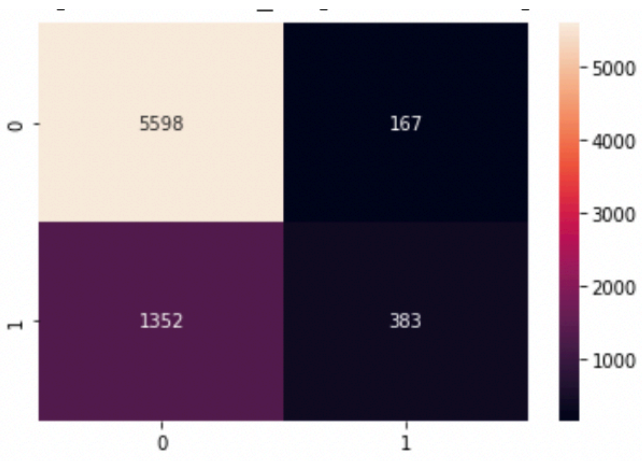


Fig6

Random Forest:

In random forest classification builds a number of decision trees on bootstrapped training samples. Compared with other bagging approaches, this method decor-relates the trees by forcing each split to consider only a subset of the predictors. The primary choice here is of predictor subset size m. Using a small value of m in building a random forest is typically helpful when we have a large number of correlated predictors.

In this case of Random forest we have implemented with bootstrap technique and we have also performed hyper parameter tuning (Gradient search) to obtain the best parameter value for the given dataset. We have obtained the best parameters as 700 trees with a depth of 6 nodes. To train this model it took a total time of 15.6 seconds in a MacBook M1 processor with 8GB ram.

The ROC curve for the training and testing dataset for Logistic Regression algorithm is shown in Fig7

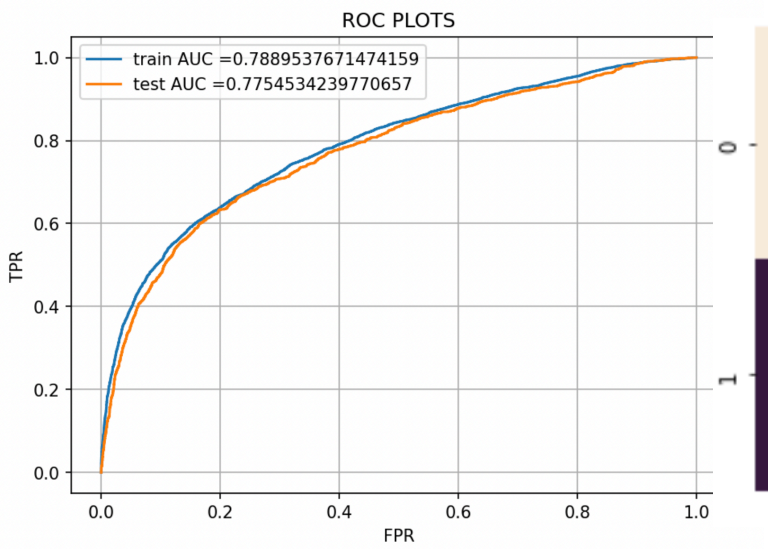


Fig7

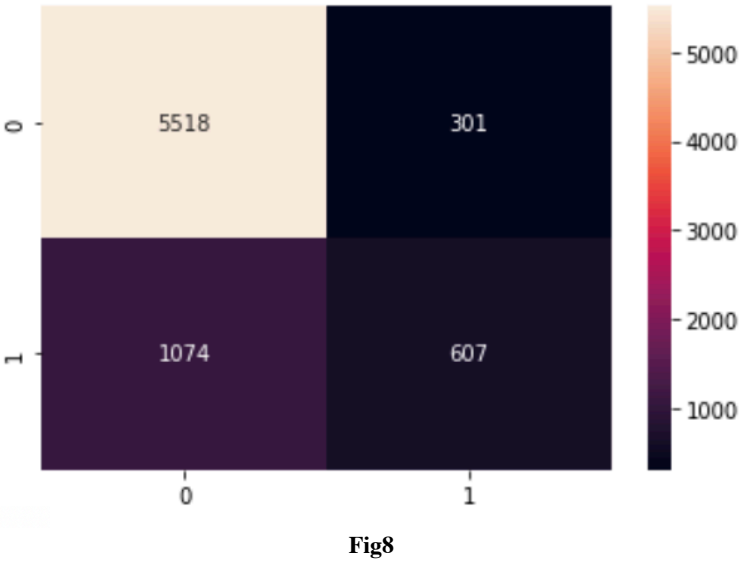


Fig8

The total computation time taken to train the algorithm using hyper-parameter optimization, is almost 3 hours. The accuracy of the Random Forest Classifier, evaluated on the test data set was 81.66%, and the precision of 66.85%,and a F1 score of 0.46. Also from the confusion matrix we can infer that the algorithm does not perform well on defaulters, in part, due to the unbalance in the training data set, that contains a majority of non-defaulters, and also suggesting that the features are not related enough to make the data linearly separable.

Support Vector Machine:

Support Vector Machines (SVMs) are supervised learning models utilized for performing supervised classification and regression. SVMs are effective models in cases with high dimensional data and even where the number of samples is smaller than the number of dimensions. In this we use SVM method to classify the data and find out which class a new data point will belong to.

First, using "GridSearchCV" we tried to optimize the parameters of the svm by cross-validated grid-search over a parameter-grid. The kernel type we used is "rbf". The parameters we optimized are "C", and "gamma". "C" is the regularization parameter, and "gamma" is the kernel coefficient. The optimized "C" is equal to 2 and best "gamma" is equal to 0.316. Then we trained the model using the training data and then used the trained model to predict the test data. The total computing time for SVM was 30.43 seconds and the accuracy is 81.64%.

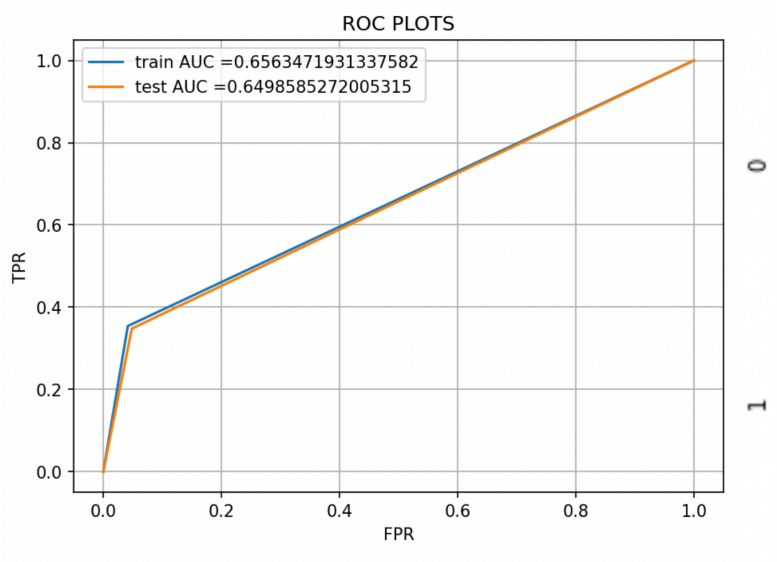


Fig9

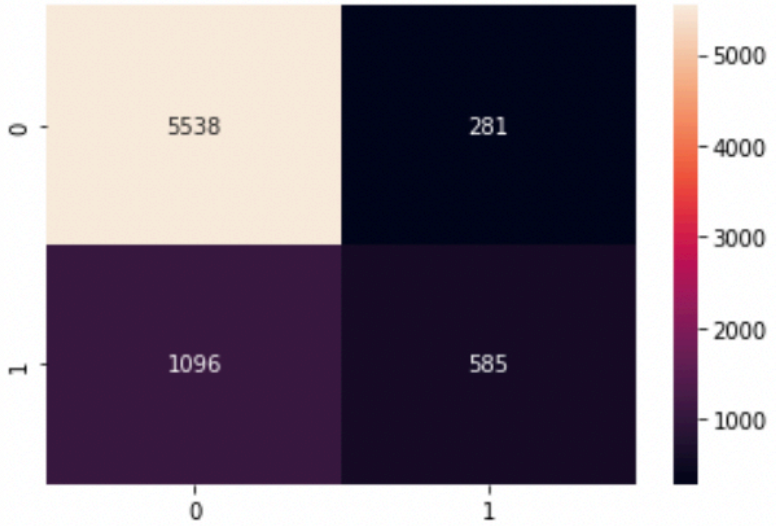


Fig10

The total computation time taken to train the algorithm using hyper-parameter optimization, is almost 2 hours. The accuracy of the SVM classifier, evaluated on the test data set was 81.64%, and the precision of 67.55%,and a F1 score of 0.45. The AUC for the ROC plot in test dataset is 0.649.

Also from the confusion matrix we can infer that the algorithm does not perform well on defaulters, in part, due to the unbalance in the training data set, on comparing this with the random forest the accuracy is almost the same but the computation time is almost twice the time take by random forest and also this svm model is good in predicting non defaulters compared to random forest but bad in predicting defaulter compared to random forest.

Boosting:

The boosting method trains an arbitrary number of weak learners in a sequential manner, where the worst predicted samples are weighted more in the successive iterations to train new learners more focused on this observations. We have used the AdaBoostClassifier. Boosting works in a similar way as bagging, except that the trees are grown sequentially i.e., each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling instead each tree is fit on a modified version of the original data set. Boosting has three tuning parameters: the number of trees B, the shrinkage parameter λ, the number of splits in each tree, d. We used an AdaBoost classifier- which is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset. The weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

We have performed gradient search to find the best number of models (estimators) required to achieve best result. The best estimator is 100 and the learning rate is 1. This model has achieved an accuracy of 81.88 in train dataset. This has take 3.26 seconds by the model to compute the test data classification.

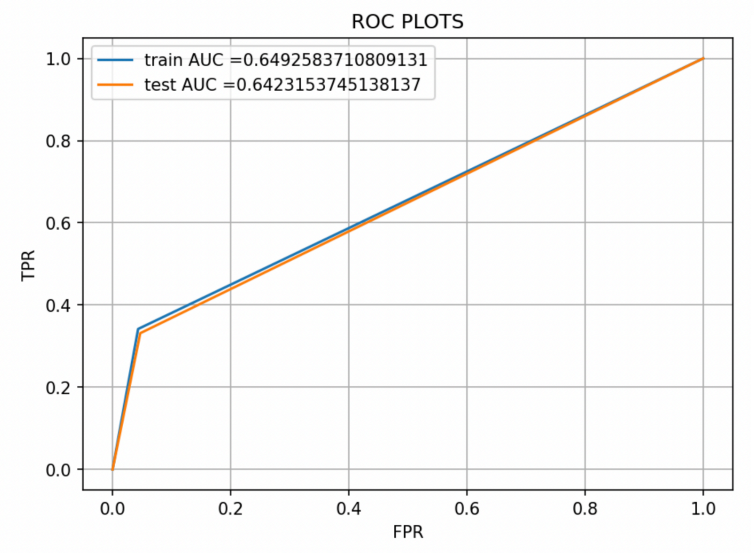


Fig11

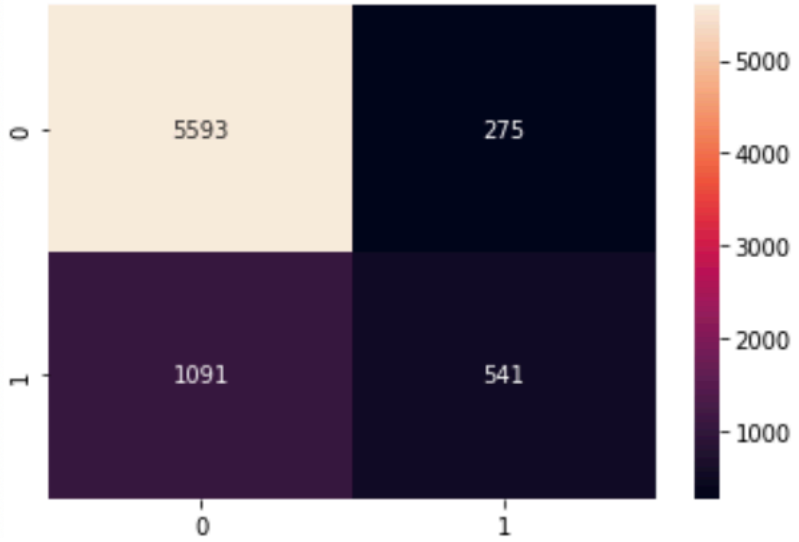


Fig12

The total computation time taken to train the algorithm using hyper-parameter optimization, is 8 min. The accuracy of the boosting classifier, evaluated on the test data set was 81.78%, and the precision of 66.29%,and a F1 score of 0.44. The AUC for the ROC plot in test dataset is 0.642.

Comparison between methods:

The results accuracy, error, precision, F1 score, AUC for ROC is all summarized for all the all the classification method in the table1. All the three algorithms perform similarly in terms of the accuracy and the precision in the positive labels that they offer. There is no much difference there than the timing it takes for the model to compute the classification. On comparing all the model we performed, SVM takes the highest computing time and boosting has the lowest computing time. Also on comparing the models, AdaBoosting has achieved the highest accuracy rate and also AdaBoosting is the only classifier which classifies non defaulter most of the cases.

	Random Forest	SVM	Boosting
Accuracy	81.66%	81.64%	81.78%
Error	18.33%	18.35%	18.21%
Precession Y=1	66.85%	67.55%	66.29%
F1 score Y=1	0.468	0.459	0.441
Test AUC	0.775	0.649	0.642
Computing Time (sec)	15.67	30.43	3.26

Conclusion:

1. We have performed a comprehensive analysis of a credit card data set to predict clients probable to default. The best accuracy achieved was 81.78%, with a low recall for the defaulters

2. The lasso regression played a significant role in dimensionality reduction which helped in reducing the computational complexity without hugely impacting the accuracy, the features removed by lasso was also verified from the correlation plot

3. If a bank requires a classier that performs good in classifying defaults then Random forest classifier is the best choice, If the bank required a classifier that performs good in classifying Non defaults then logistic regression is the best choice, If a bank needs the best model that classifies both defaults and non defaults then AdaBoosting is the best choice

Individual contribution:

Denesh Kumar Manoharan: Logistic Regression + Lasso penalization, Random Forest and Boosting

Aadya Sanwal: Data Analysis Visualization, Data standardization, Support Vector Machine

NOTE:

As suggested by professor during class presentation to Implement Random forest method to identify the feature that is less significant and perform dimensionality reduction

On performing the above task we have obtained the weights for all the feature and have plated as seen if Fig 13

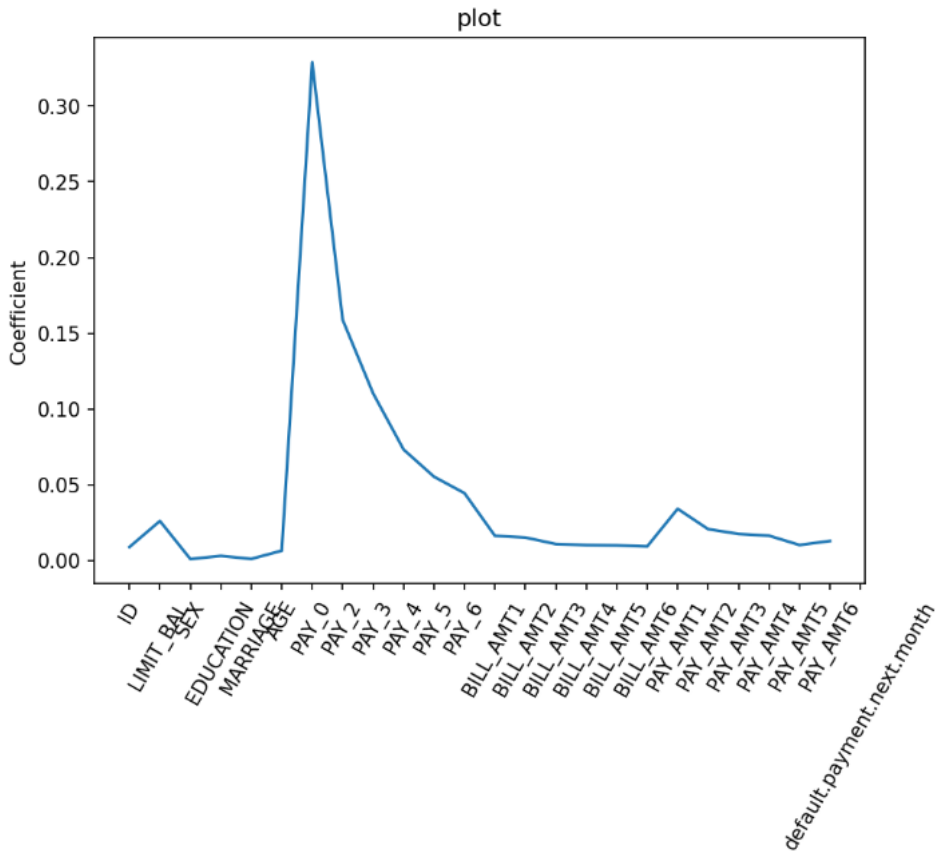


Fig13

As we can visualize from the plot the weights of the feature at Bill_AMT2, Bill_AMT3, Bill_AMT4, Bill_AMT5, PAY_AMT5, PAY_AMT6 are almost zero compared to weights of other feature, which means these features are less significant and can be removed from dataset as dimensionality reduction.

Also,
The features selected above by random forest is exact set of features that was selected when we performed logistic regression with lasso penalization as seen in Fig 4.

```
import time
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.linear_model import Lasso
from sklearn.metrics import roc_curve, auc\

import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score,
confusion_matrix, roc_curve, auc

from sklearn import svm
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier

# Import Credit Card Data

data_credit_card = pd.read_csv("UCI_Credit_Card.csv")

data_credit_card.head()

# Dimension of the dataset, we have 24 Features and 30000 data points

data_credit_card.shape

# Check for Null or missing values in dataset, and we do not have any Null points

data_credit_card.isnull().sum()

# Splitting Feature and the Prediction variable from dataset as x and y respectively

x = data_credit_card.drop('default.payment.next.month', axis=1)
y = data_credit_card['default.payment.next.month']

"""# Splitting data for Training and Testing"""

#By default the data gets split into 75-25 rule

x_train, x_test, y_train, y_test = train_test_split(x, y)

"""# Standardization the Features"""

# The values of some feature are in higher value range and some are in lower value range, so the
classification model can assume the feature with higher value is
# more important and assign more weight to that feature, so we perform standardization to bring all
feature to same level

model_standardization=StandardScaler()
```

```
model_standardization.fit(x_train)
x_train=model_standardization.transform(x_train)
x_test=model_standardization.transform(x_test)

"""# Feature Selection by Lasso Regression"""

# Performing Gridsearch to find the best value of alpha

lasso_model=Lasso()
param=np.logspace(-4, 4, 40)
parameters={'alpha':param}
lasso_classifier=GridSearchCV(lasso_model,parameters,cv=5)
lasso_classifier.fit(x_train,y_train)

print(lasso_classifier.best_params_)

best_model_lasso=Lasso(alpha=lasso_classifier.best_params_['alpha'])
best_model_lasso.fit(x_train,y_train)
best_model_lasso.coef_

name = data_credit_card.columns

# This plot is to visualize how the weights of each feature varies, the features with zero weights can be
removed as they are not significant for prediction

plt.figure(figsize=(7,5),dpi=150)
plt.title("Feature weight plot")
plt.plot(best_model_lasso.coef_)
_=plt.xticks(range(len(name)),name,rotation=60)
_=plt.ylabel("Coefficient")

coeffi = best_model_lasso.coef_!=0
non_significant=[]
for i in range(len(data_credit_card.columns)-1):
  if coeffi[i]==False:
    non_significant.append(data_credit_card.columns[i])

# These are the fesatures that is currently being removed

non_significant

# Removing features with zero weights as they are not significant for our prediction

x_train=x_train[:,best_model_lasso.coef_!=0]
x_test=x_test[:,best_model_lasso.coef_!=0]

# The feature has been reduced and this is the new dimention of out dataset

x_train.shape

"""# Logistic Regression"""

# Performing Gridserach for the best value of Regularization strength and Penalty
```

```
start0=time.time()
logisticRegression_model=LogisticRegression()
parameters = {'C':np.logspace(-1, 3, 50)}
logistic_regression= GridSearchCV(logisticRegression_model, parameters, cv=6, scoring='accuracy')
logistic_regression.fit(x_train,y_train)
end0=time.time()

print(f"Time taken gradient search for logistic regression {end0-start0} seconds")

# The identified best parameters from gradient search

print(logistic_regression.best_params_)
print(logistic_regression.best_score_)

# Thus found the best parameters, we are using this for our final logistic regression model

start1=time.time()
c_best=logistic_regression.best_params_['C']
#penalty_best=logistic_regression.best_params_['penalty']
logistic_regression_best_model=LogisticRegression(C=c_best)
logistic_regression_best_model.fit(x_train, y_train)
pred_y_train = logistic_regression_best_model.predict_proba(x_train)[:,1]
pred_y_test = logistic_regression_best_model.predict_proba(x_test)[:,1]
end1=time.time()

print(f"Time taken by logistic regression to run is {end1-start1} seconds")

"""# ROC Plot"""

tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train, pred_y_train)
te_fpr, te_tpr, te_thresholds = roc_curve(y_test, pred_y_test)

plt.figure(figsize=(7,5),dpi=150)
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()

"""# Confusion Matrix::"""

print("Confusion matrix")
sns.heatmap(confusion_matrix(y_test,
logistic_regression_best_model.predict(x_test)),annot=True,fmt='d')

accuracy_logisticRegression=accuracy_score(y_test, logistic_regression_best_model.predict(x_test))
error_logisticRegression  = 1 - accuracy_logisticRegression
precision_logisticRegression = precision_score(y_test, logistic_regression_best_model.predict(x_test),
average='binary')
f1_logisticRegression=f1_score(y_test, logistic_regression_best_model.predict(x_test))
print(f"Accuracy of logistic regression for this dataset is {accuracy_logisticRegression}")
```

```
print(f"Error of logistic regression for this dataset is {error_logisticRegression}")
print(f"Precision score of logistic regression for this dataset is {precision_logisticRegression}")
print(f"F1 score of logistic regression for this dataset is {f1_logisticRegression}")

"""# Random Forest """

parameters = { 'bootstrap': [True],
    'n_estimators': [550,600,700],
    'max_features': ['auto', 'sqrt'],
    'max_depth' : [6,7],
    'criterion' :['gini', 'entropy']
}

start2=time.time()
random_forest=RandomForestClassifier()
Random_Forest_Classifier_model= GridSearchCV(random_forest, parameters, cv=5,
scoring='accuracy',return_train_score=True,n_jobs=-1)
Random_Forest_Classifier_model.fit(x_train,y_train)
end2=time.time()

print(f"Time taken for gradient search Random Forest Classifier to run is {end2-start2} seconds")

print(Random_Forest_Classifier_model.best_params_)
print(Random_Forest_Classifier_model.best_score_)

start3=time.time()
best_n_estimators=Random_Forest_Classifier_model.best_params_['n_estimators']
best_max_features=Random_Forest_Classifier_model.best_params_['max_features']
best_max_depth=Random_Forest_Classifier_model.best_params_['max_depth']
best_criterion=Random_Forest_Classifier_model.best_params_['criterion']
best_RandomForestClassifier_model=RandomForestClassifier(n_estimators=best_n_estimators,max_fea
tures=best_max_features,max_depth=best_max_depth,criterion=best_criterion)
best_RandomForestClassifier_model.fit(x_train, y_train)
end3=time.time()

print(f"Time taken be Random Forest Classifier to run is {end3-start3} seconds")

pred_y_train = best_RandomForestClassifier_model.predict_proba(x_train)[:,1]
pred_y_test = best_RandomForestClassifier_model.predict_proba(x_test)[:,1]

tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train, pred_y_train)
te_fpr, te_tpr, te_thresholds = roc_curve(y_test, pred_y_test)

plt.figure(figsize=(7,5),dpi=150)
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()

print("confusion matrix")
```

```
sns.heatmap(confusion_matrix(y_test,
best_RandomForestClassifier_model.predict(x_test)),annot=True,fmt='d')

accuracy_randomforest_model=accuracy_score(y_test,
best_RandomForestClassifier_model.predict(x_test))
error_randomforest_model = 1 - accuracy_randomforest_model
precision_randomforest_model = precision_score(y_test,
best_RandomForestClassifier_model.predict(x_test), average='binary')
f1_randomforest_model=f1_score(y_test, best_RandomForestClassifier_model.predict(x_test))
print(f"Accuracy of Random Forest Classifier for this dataset is {accuracy_randomforest_model}")
print(f"Error of Random Forest Classifier for this dataset is {error_randomforest_model}")
print(f"precision of Random Forest Classifier for this dataset is {precision_randomforest_model}")
print(f"F1 score of Random Forest Classifier for this dataset is {f1_randomforest_model}")

"""# SVM"""

start4=time.time()
parameters = {'C':[0.5,1,2], 'gamma':np.logspace(-3, 3, 5)}
svm = svm.SVC()
svm_model = GridSearchCV(svm, parameters, cv=5,
scoring='accuracy',return_train_score=True,n_jobs=-1)
svm_model.fit(x_train,y_train)
svm_model.best_estimator_
end4=time.time()

print(f"Time taken for gradient search SVM Classifier to run is {end4-start4} seconds")

start5=time.time()
best_c=svm_model.best_params_['C']
best_gamma=svm_model.best_params_['gamma']
svm_best_model = svm.SVC(C=best_c, random_state=0, gamma=best_gamma)
svm_best_model.fit(x_train, y_train)
end5=time.time()

print(f"Time taken by SVM model to run is {end5-start5} seconds")

print(best_c)

print(best_gamma)

y_train_pred = svm_best_model.predict(x_train)
y_test_pred = svm_best_model.predict(x_test)

tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
te_fpr, te_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.figure(figsize=(7,5),dpi=150)
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
```

```
plt.show()

print("Confusion matrix")
sns.heatmap(confusion_matrix(y_test, y_test_pred),annot=True,fmt='d')

accuracy_svm=accuracy_score(y_test,y_test_pred)
error_svm = 1 - accuracy_svm
precision_svm_model = precision_score(y_test,svm_best_model.predict(x_test), average='binary')
f1_svm=f1_score(y_test, y_test_pred)
print(f"Accuracy of SVM Classifier for this dataset is {accuracy_svm}")
print(f"Error of SVM Classifier for this dataset is {error_svm}")
print(f"Precision of SVM Classifier for this dataset is {precision_svm_model}")
print(f"F1 score of SVM Classifier for this dataset is {f1_svm}")

"""# Boosting """

start6=time.time()
parameters = {'n_estimators':[50,90,100], 'learning_rate':[1]}
AdaBoosting_model = AdaBoostClassifier()
clf = GridSearchCV(AdaBoosting_model, parameters,cv=5, scoring='accuracy')
clf.fit(x_train,y_train)
clf.best_estimator_
end6=time.time()

print(f"Time taken AdaBoosting gradient search to run is {end6-start6} seconds")

print(clf.best_params_)
print(clf.best_score_)

start7=time.time()
best_n_estimators=clf.best_params_['n_estimators']
best_learning_rate=clf.best_params_['learning_rate']
bst = AdaBoostClassifier(base_estimator=None, n_estimators=best_n_estimators, random_state=0,
learning_rate=best_learning_rate)
bst.fit(x_train, y_train)
end7=time.time()

print(f"Time taken to run AdaBoosting Classifier to run is {end7-start7} seconds")

y_train_pred = bst.predict(x_train)
y_test_pred = bst.predict(x_test)

tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
te_fpr, te_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.figure(figsize=(7,5),dpi=150)
plt.plot(tr_fpr, tr_tpr, label="train AUC ="+str(auc(tr_fpr, tr_tpr)))
plt.plot(te_fpr, te_tpr, label="test AUC ="+str(auc(te_fpr, te_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

```
print("confusion matrix")
sns.heatmap(confusion_matrix(y_test, bst.predict(x_test)),annot=True,fmt='d')

accuracy_AdaBoosting=accuracy_score(y_test,y_test_pred)
error_AdaBoosting = 1- accuracy_AdaBoosting
precision_AdaBoosting = precision_score(y_test, y_test_pred, average='binary')
f1_AdaBoosting=f1_score(y_test, y_test_pred)
print(f"Accuracy of AdaBoosting Classifier for this dataset is {accuracy_AdaBoosting}")
print(f"error of AdaBoosting Classifier for this dataset is {error_AdaBoosting}")
print(f"precision of AdaBoosting Classifier for this dataset is {precision_AdaBoosting}")
print(f"F1 score of AdaBoosting Classifier for this dataset is {f1_AdaBoosting}")
```

CODE FOR DATA ANALYSIS VISUALIZATION:

```
data_credit_card = pd.read_csv("UCI_Credit_Card.csv")
# Splitting Feature and the Prediction variable from dataset as x and y respectively

x = data_credit_card.drop('default.payment.next.month', axis=1)
y = data_credit_card['default.payment.next.month']

##Exploratory Data Analysis

data_credit_card.rename(columns={'default.payment.next.month':'def_pay'}, inplace=True)
def_cnt = round((data_credit_card.def_pay.value_counts(normalize=True)*100),3)
def_cnt.plot.bar(figsize=(6,6))
plt.xticks(fontsize=12, rotation=0)
plt.yticks(fontsize=12)
plt.title("Probability Of Default Next Month", fontsize=15)
for x,y in zip([0,1],def_cnt):
    plt.text(x,y,y,fontsize=12)
plt.show()

data_credit_card['def_pay'].value_counts()

class_0 = data_credit_card.loc[data_credit_card['def_pay'] == 0]["LIMIT_BAL"]
class_1 = data_credit_card.loc[data_credit_card['def_pay'] == 1]["LIMIT_BAL"]
plt.figure(figsize = (10,4))
plt.title('Default amount of credit limit  - grouped by Payment Next Month (Density Plot)')
sns.set_color_codes("pastel")
sns.distplot(class_1,kde=True,bins=200, color= "red")
sns.distplot(class_0,kde=True,bins=200, color= "cyan")
plt.show()

subset2 = data_credit_card[['PAY_0', 'PAY_2', 'PAY_3', 'PAY_4',
            'PAY_5', 'PAY_6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5',
'PAY_AMT6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
'LIMIT_BAL', 'AGE', 'def_pay']]

#  looking at correlations matrix, defined via Pearson function
```

```
corr = subset2.corr() # .corr is used to find corelation
f,ax = plt.subplots(figsize=(8, 7))
sns.heatmap(corr, cbar = True,  square = True, annot = False, fmt= '.1f',
            xticklabels= True, yticklabels= True
            ,cmap="coolwarm", linewidths=.5, ax=ax)
plt.title('CORRELATION MATRIX - HEATMAP', size=18);

# Creating a new dataframe with categorical variables
subset = data_credit_card[['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4',
                'PAY_5', 'PAY_6', 'def_pay']]
subset.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'}, 'EDUCATION' : {1 : 'graduate school', 2 : 'university', 3
: 'high school', 4 : 'others', 5 : 'others', 6 : 'others',  0 : 'others'}, 'MARRIAGE' : {1 : 'married', 2 : 'single',
3 : 'others', 0 : 'others'}}, inplace = True)

f, axes = plt.subplots(3, 3, figsize=(20, 13), facecolor='white')
f.suptitle('FREQUENCY OF CATEGORICAL VARIABLES (BY DEFAULT)')
ax1 = sns.countplot(x="SEX", hue="def_pay", data=subset, palette="bright", ax=axes[0,0])
ax2 = sns.countplot(x="EDUCATION", hue="def_pay", data=subset, palette="bright",ax=axes[0,1])
ax3 = sns.countplot(x="MARRIAGE", hue="def_pay", data=subset, palette="bright",ax=axes[0,2])
ax4 = sns.countplot(x="PAY_0", hue="def_pay", data=subset, palette="bright", ax=axes[1,0])
ax5 = sns.countplot(x="PAY_2", hue="def_pay", data=subset, palette="bright", ax=axes[1,1])
ax6 = sns.countplot(x="PAY_3", hue="def_pay", data=subset, palette="bright", ax=axes[1,2])
ax7 = sns.countplot(x="PAY_4", hue="def_pay", data=subset, palette="bright", ax=axes[2,0])
ax8 = sns.countplot(x="PAY_5", hue="def_pay", data=subset, palette="bright", ax=axes[2,1])
ax9 = sns.countplot(x="PAY_6", hue="def_pay", data=subset, palette="bright", ax=axes[2,2]);
```