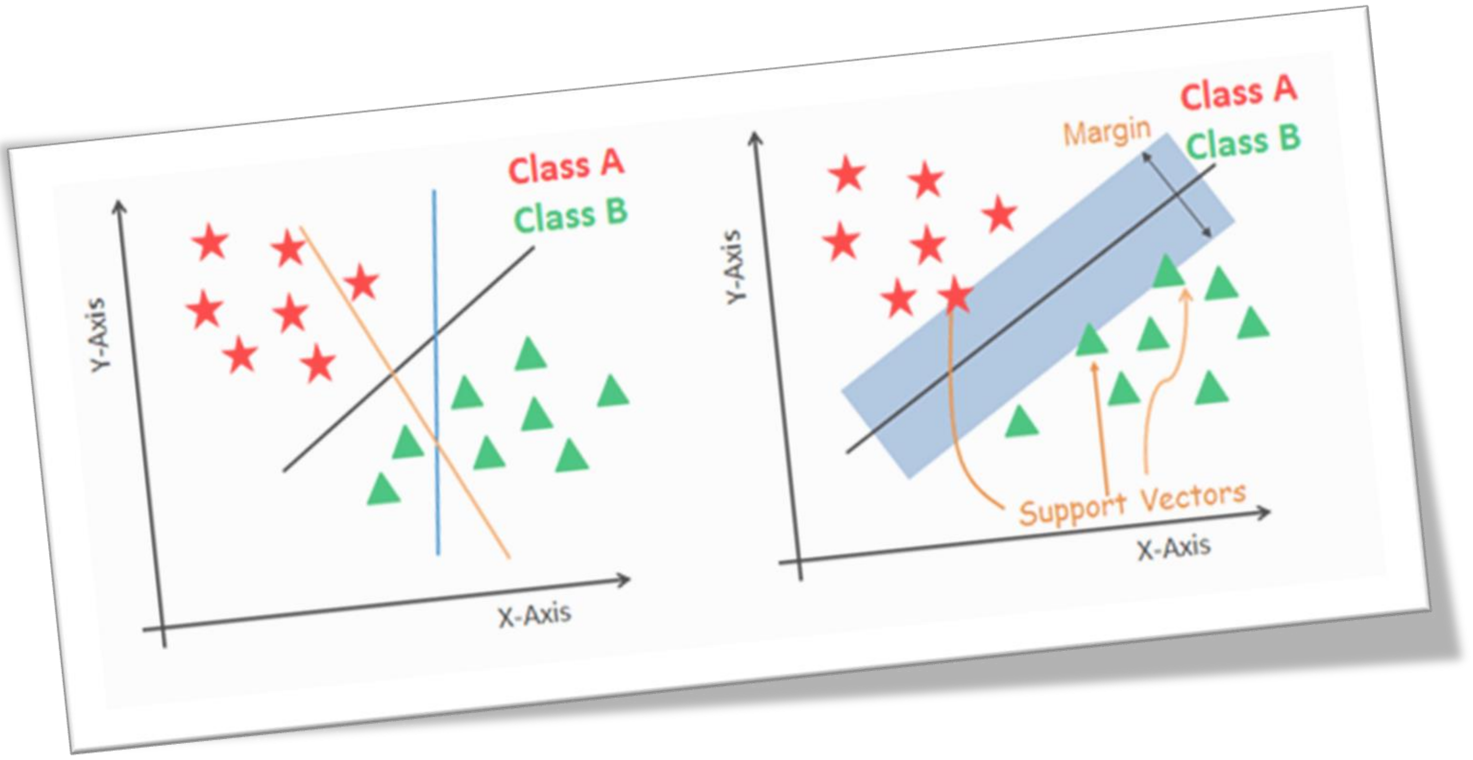


Credit Risk Analytics using SVM in Python

Created by- Dhruv Dixit

Abstract of Project

A key activity within the banking industry is to extend credit to customers, hence, credit risk analysis is critical for financial risk management. There are various methods used to perform credit risk analysis. In this project, we analyse German and Australian financial data from UC Irvine Machine Learning repository, reproducing results previously published in literature. Further, using the same dataset and various machine learning algorithms, we attempt to create better models by tuning available parameters, however, our results are at best comparable to published results.

In this report, we have explained the algorithms and mathematical framework that goes behind developing the machine learning models. We conclude with a discussion and comparison of summarizing the best approach to classify these datasets. K-Nearest Neighbours (KNN), Logistic Regression (LR), Naive Bayes Classification, Support Vector Machine (SVM), Classification Trees and Artificial Neural Networks (ANN) are the machine learning models used for this report.

Objective of Project

INTRODUCTION:

MOTIVATION

A significant activity of the banking industry is to extend credit to customers. Credit risk management evaluates available data and decides the credibility of a customer, with the intent of protecting the nancial institution against fraud.

DATASETS

The UC Irvine Machine Learning repository (UCI - ML) contains collection of datasets useful for evaluating machine learning algorithms. Two datasets rom the UCI – ML repository were used for this project: the Australian credit dataset [1], and other is the German credit dataset[2]. The German dataset has 20 attributes and 1,000 instances, while the Australian dataset has 14 characteristics and 690 instances. The response variable is a binary decision, whether a customer is credible or not. Both datasets have some common attributes such as the credit score, the purpose of the loan and customer information (occupation, salary, age and account duration).

METHODOLOGY

We apply six machine learning classification algorithms. A summary of the methods are as follows:

1. K-Nearest Neighbours (KNN)

The KNN algorithm is a non-parametric technique that classifies unknown data points based on a majority votes of it's K nearest neighbors.

2. Logistic Regression (LR)

Logistic Regression fits a logistic (sigmoid) function to the data. This is useful when the dependent variable is binary. To compute the weights, we will utilize the Iteratively Re-weighted Least Squares algorithm (IRLS).

3. Naive Bayes Classification

This algorithm is based on the Bayes' Theorem. It computes the probability of each potential classification and selects the one with higher probability.

4. Support Vector Machine (SVM)

SVM identifies hyperplanes to separate the data based on labels. To handle nonlinear data, projections using various kernels can be used.

5. Classification Trees

We will implement two types of classification trees: Decision Tree and Random Forest. Decision Tree is similar to a flowchart and is very easy to visualize. Every terminal node represents the output. Random Forest is a collection of decision trees, where the majority vote is taken for prediction.

6. Artificial Neural Networks (ANN)

Neural Network is an algorithm which learns from the data. The process of learning is similar to the human brain. Neural networks are used in many realworld application like classify images, predicting the weather, voice detection etc.

Process of Project

Linear SVM

Support Vector Machine is a supervised machine learning algorithm. The idea is to select the optimal hyperplane that partitions the data. There are indeed many hyperplanes that can partition the data, see fig. 1; the SVM method selects the optimal hyperplane, which we define below. First, the equation of a hyperplane is given by

$w^T x + b = 0$ --eq. (1)

where w is a weight vector, x is the input vector and b is the bias. The optimal hyperplane we wish to find is the one which maximizes the margin to the hyperplane, that is, finding a hyperplane that maximizes the distance to the nearest training-data point of any class.

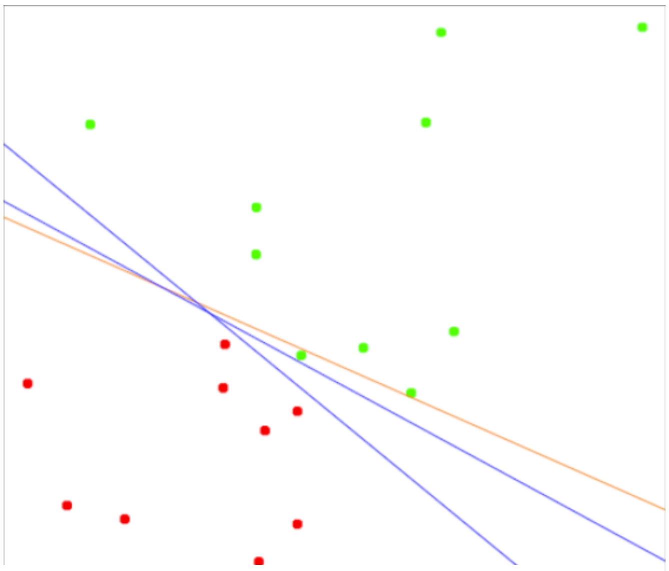


Figure 1: Example of the possible hyperplanes

The distance between a data point xi to a hyperplane defined by eq. (1) is given by

$|w^T x_i + b|/||w||$ --eq. (2)

We can scale w arbitrarily without changing its direction, so without loss of generality, we can scale w so that the numerator of eq. (2) is one. Hence, the distance between the data point xi to a hyperplane will be given by 1/||w||. The data point(s) in Class "+1" closest to the hyperplane will obey $w^T x_i + b = 1$, and the points in Class "-1" closest to the hyperplane will obey $w^T x_i + b = -1$. An example of the optimal margins given in fig. (2). The total width between the hyperplanes and the two classes is thus given by

$$\frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}.$$

If the data is linearly separable, the following must hold true:

$$w^T x_i + b \geq 1, \qquad \forall x_i \in C_{+1},$$

$$w^T x_i + b \leq -1, \qquad \forall x_i \in C_{-1}.$$

--eq. 3

We note that maximizing the margin is $\frac{1}{2}\|w\|^2$ equivalent to minimizing ||w||. For convenience, we minimize. If we desire that all the samples are correctly classified i.e., eq. (3) is satisfied, then we wish to satisfy

$$y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N.$$

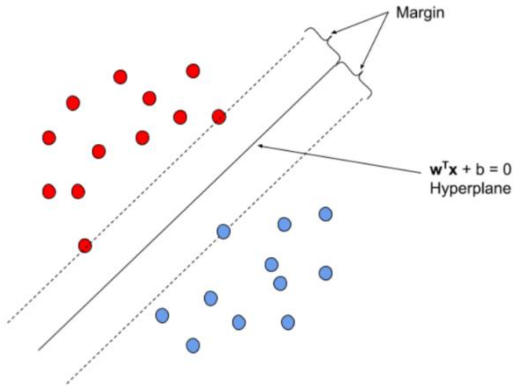


Figure 2: Example of optional margin

The constrained minimization problem can then be posed:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad i = 1, \ldots, N.$$

Using Lagrange multipliers to find the minima,

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i (y_i(w^T x_i + b) - 1).$$

--eq. 4

Due to the inequality constraints, the solution to this must meet the following Karush-Kuhn-Tucker (KKT) conditions,

$$\alpha_i \geq 0, \quad i = 1, \ldots, N;$$

$$\alpha_i (y_i(w^T x_i + b) - 1) = 0, \quad \forall i$$

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \implies w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0$$

Substituting these KKT conditions, eq. (5), into the Lagrangian equation, eq. (4), leads to the Wolfe dual problem,

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k \, y_i \, y_k \, x_i^T x_k$$
$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N.$$

The dual problem is a maximization problem of a cost function quadratic in. So, we'll minimize the negative of cost function,

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k \, y_i \, y_k \, x_i^T x_k - \sum_{i=1}^{N} \alpha_i$$
$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N.$$

To put this in standard QP form, we`ll express it in matrix/vector form,

$$\min_{\alpha} \frac{1}{2} \alpha^T \begin{bmatrix} y_1 \, y_1 \, x_1^T \, x_1^T & y_1 \, y_2 \, x_1^T \, x_1^T & \cdots & y_1 \, y_N \, x_1^T \, x_N^T \\ y_2 \, y_1 \, x_2^T \, x_1^T & y_2 \, y_2 \, x_2^T \, x_1^T & \cdots & y_2 \, y_N \, x_2^T \, x_N^T \\ \vdots & \vdots & \ddots & \vdots \\ y_N \, y_1 \, x_N^T \, x_1^T & y_N \, y_2 \, x_N^T \, x_1^T & \cdots & y_N \, y_N \, x_N^T \, x_N^T \end{bmatrix} \alpha - 1^T \alpha,$$

$$\text{subject to} \quad y^T \alpha = 0, \quad 0 \leq \alpha \leq \infty.$$

This minimization problem can be solved by any QP solver. In Python, the CVXOPT library has QP solvers, and MATLAB has the quadprog() function.

Non-Linear SVM

Suppose the data is non-linearly separable. SVM can still be applied to separate data with the use of a kernel trick (a kernel is used to transform the data to a higher dimensions) which is then separable with hyperplanes. Radial Basis

Functions (RBF) and polynomial kernels are two popular kernel functions used in the SVM community.

† RBF Kernel: $K(x_i, x_k) = e^{-\gamma \|x_i - x_k\|^2}$

† d-th degree Polynomial Kernel: $K(x_i, x_k) = \left(1 + x_i^T x_k\right)^d$

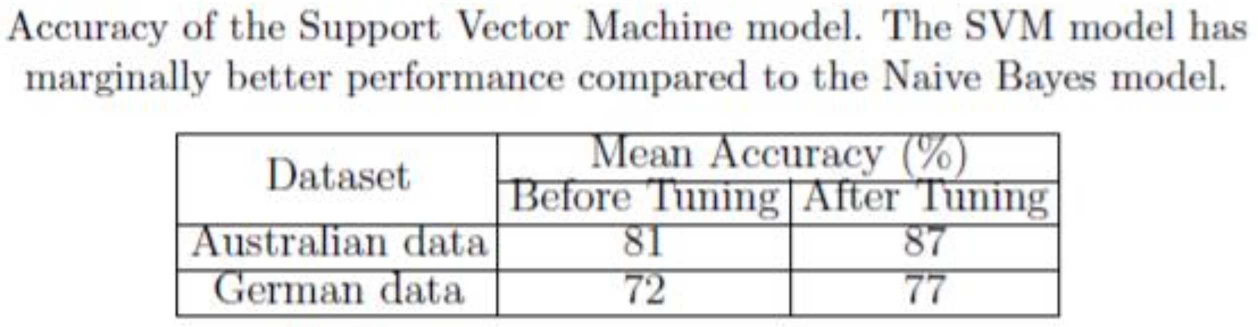
We can rewrite the Wolfe dual minimization problem using the kernel functions

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \, \alpha_k \, y_i \, y_k \, K(x_i, x_k) - \sum_{i=1}^{N} \alpha_i \qquad \text{--eq. 5}$$
$$\text{subject to} \sum_{i=1}^{N} \alpha_i y_i = 0, \quad \alpha_i \geq 0, \quad i = 1, \ldots, N$$

Results:

By applying Kernel SVM to German and Australian credit data sets, the accuracies are obtained shown in table (1). The default parameters for SVM is set to be polynomial kernel with degree 2. The other tuning values for degree are from 1 to 7 and for kernel, RBF is also considered as different kernel. However, for Australian data set, it gives even good accuracy by setting the polynomial kernel with degree 1. While for German data, it improves the accuracy with RBF kernel.

Figures (3) and (4) are the confusion matrix of German data and Australian data respectively. It seems that it has more false positives which are 33 and 19 for German and Australian data respectively. This model has good accuracies, but due to a greater number of false-positives, this model is risky for these data sets.



Accuracy of the Support Vector Machine model. The SVM model has marginally better performance compared to the Naive Bayes model.

Dataset	Mean Accuracy (%)	
	Before Tuning	After Tuning
Australian data	81	87
German data	72	77

Table 1: Accuracy of the Support Vector Machine model. The SVM model has marginally better performance compared to the Naive Bayes model.

Figures (5) and (6) are the decision boundary plot of this method. We can see that in German data set the data are separated by the Gaussian curve, while a straight line separates Australian data because for this data set the poly kernel with degree 1 is used.

Figures 7 and 8 are the learning curves for the German and Australian data respectively. In fig. (7) the training score is very high; however the validation score is not improving, which shows that the SVM model for German data is over-fitting. In fig. (8) the gap between the training score and validation score is very low which shows under-fitting.

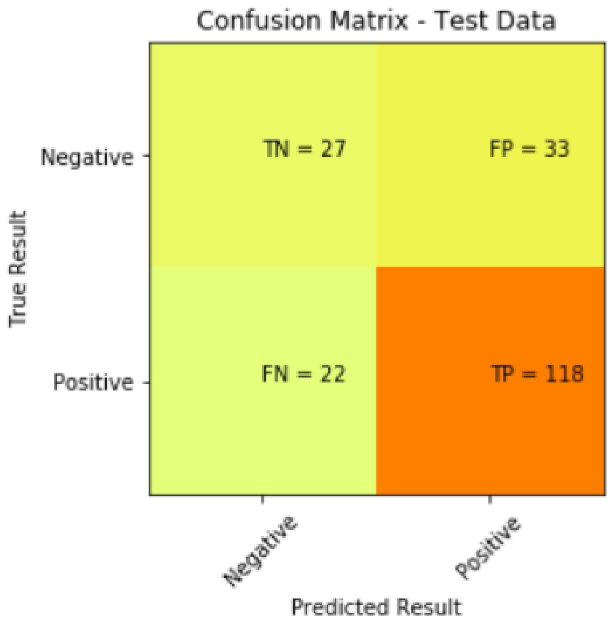


Figure 3: Confusion matrix of Support Vector Machine using German data set. There is an unacceptably large number of false positives.

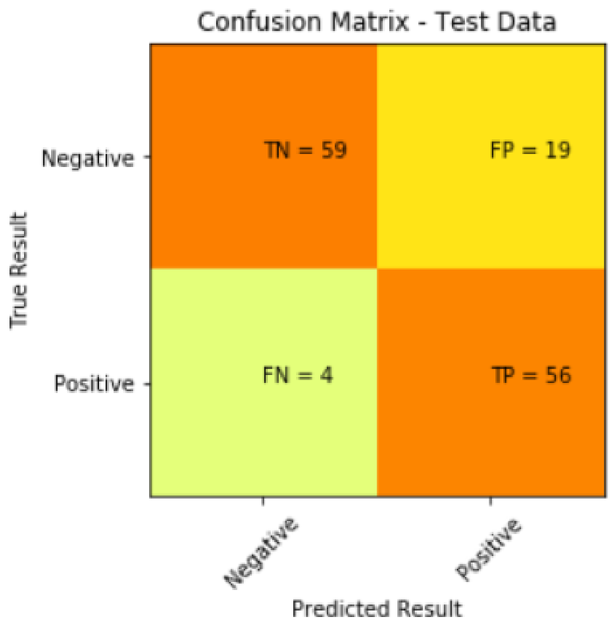


Figure 4: Confusion matrix of Support Vector Machine using Australian data set. There is an unacceptably large number of false positives.

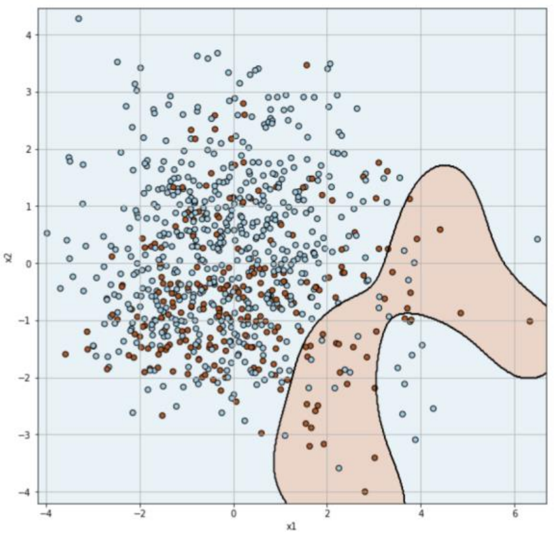


Figure 5: Decision boundary plot using Support Vector Machine of German data set. Observe that the decision boundary is a curve for this model.

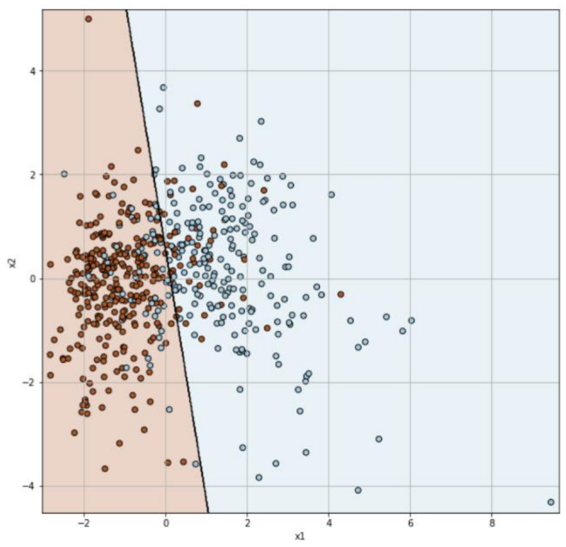


Figure 6: Decision boundary plot using Support Vector Machine of Australian data set. Observe that the decision boundary is a curve for this model.

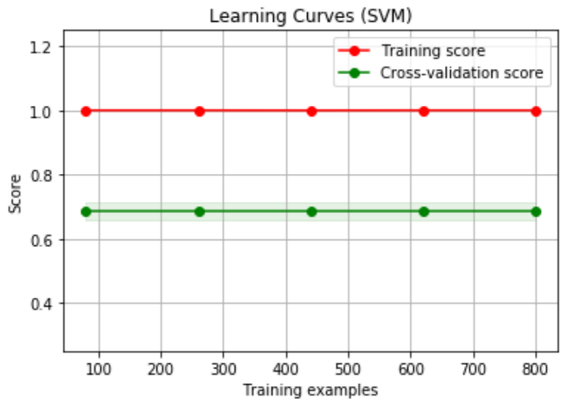


Figure 7: Learning curve of Support Vector Machine using the German data set. Training score is very high and cross validation score is not improving, indicating that the Support Vector Machine model is overfitting

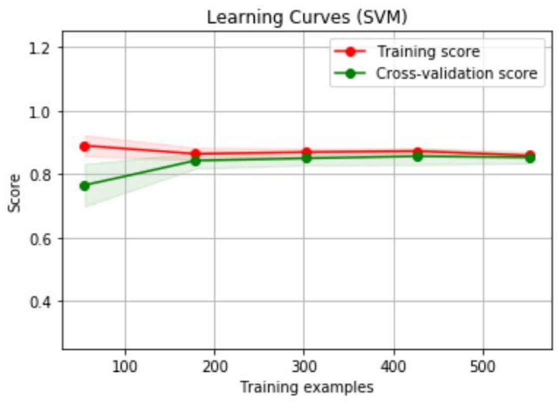


Figure 8: Learning curve of Support Vector Machine using the Australian data set. No gap is observed as increases the samples, indicating that the Support Vector Machine model is under-tting this Australian data set.

Images/Video Links

- ✓ **Cover Page Image:** https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
- ✓ **Cover Page Python Logo:** https://commons.wikimedia.org/wiki/File:Python-logo-notext.svg
- ✓

Source Code of Program

```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

# Function for Plotting:

def plot_classification(X_train,X_test,y_train,y_test,↩
   y_pred):
    from sklearn.decomposition import PCA
    pca = PCA(n_components = 2)
    X_train = pca.fit_transform(X_train)
    X_test = pca.transform(X_test)
    explained_variance = pca.explained_variance_ratio_


    comparision = pd.DataFrame.join(pd.DataFrame(y_test,↩
       columns=['y_test']),pd.DataFrame(y_pred,dtype=int,↩
       columns=['y_pred']))
    comparision['Comparision'] = comparision.apply(↩
       lambda x: 0 if x[0] == x[1] else 1, axis=1)

    plotdata = pd.DataFrame.join(pd.DataFrame(X_test,↩
       columns=['0','1']),comparision['Comparision'])


    # Plot misclassification

    fig3, ax3 = plt.subplots(figsize = (14,7))

    ax3.scatter(x = plotdata[plotdata.Comparision ==0]['↩
       0'],y = plotdata[plotdata.Comparision == 0]['1'], ↩
       marker = 'o',color = 'red')
    ax3.scatter(x = plotdata[plotdata.Comparision ==1]['↩
       0'],y = plotdata[plotdata.Comparision == 1]['1'], ↩
       marker = 'o',color = 'blue')

    ax3.legend(['Classified','Misclassified'])
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.title('Classification')
    plt.show()
```

```
# Function for Confusion Matrix:

def con_mat_plot(y_test,y_pred):
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.↩
       Wistia)
    classNames = ['Negative','Positive']
    plt.title('Confusion Matrix - Test Data')
    plt.ylabel('True Result')
    plt.xlabel('Predicted Result')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]

    for i in range(2):
        for j in range(2):



            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j↩
              ]))
        plt.show()

    # Function k-fold cross validation
    def k_fold_cross(classifier, X_train, y_train):
        from sklearn.model_selection import cross_val_score
        accuracies = cross_val_score(estimator = classifier,↩
            X = X_train, y = y_train, cv = 10)
        return accuracies.mean()

    # Data Preprocessing
    def data_preprocessing(dataset):
        listt = dataset.select_dtypes(include=['category', ↩
           object]).columns
        X = dataset.iloc[:, :-1].values
        y = dataset.iloc[:, -1].values

        for i in range(0,len(y)):
            if y[i] == 2:
                y[i] = 0


        listn = np.empty((len(listt),1))
        for i in range(0,len(listt)):
            listn[i,0] = dataset.columns.get_loc(listt[i])

        # Categorial data
        from sklearn.preprocessing import LabelEncoder, ↩
           OneHotEncoder
        labelencoder_X = LabelEncoder()

        for i in range(0,len(listn)):
```

```
        X[:,int(listn[i,0])] = labelencoder_X.↩
            fit_transform(X[:,int(listn[i,0])])
        onehotencoder        = OneHotEncoder(↩
            categorical_features = [int(listn[i,0])])

    # Splitting the dataset into the Training set and ↩
        Test set
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(↩
        X, y, test_size = 0.2)


    # Feature Scaling
    from sklearn.preprocessing import StandardScaler
    sc      = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test  = sc.transform(X_test)
    return X_train,X_test,y_train,y_test,X,y

# Function for selecting the best parameters:

def choosing_parameters(parameters,classifier,X_train,↩
    y_train):
    from sklearn.model_selection import GridSearchCV
    grid_search = GridSearchCV(estimator = classifier,
                               param_grid = parameters,
                               scoring = 'accuracy',
                               cv = 10,
                               n_jobs = -1)
    grid_search = grid_search.fit(X_train, y_train)
    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_
    best_para = np.array(best_parameters.values())
    return best_para
```

```
# Function for decision boundary plot


def plot_decision_boundary(clf, X, Y, X_test, cmap='↩
   Paired_r'):
    clf.predict(X_test)
    h = 0.02
    x_min, x_max = X[:,0].min() - 10*h, X[:,0].max() + ↩
       10*h
    y_min, y_max = X[:,1].min() - 10*h, X[:,1].max() + ↩
       10*h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,10))
    plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.25)
    plt.contour(xx, yy, Z, colors='k', linewidths=0.7)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.grid()
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=cmap, ↩
       edgecolors='k');

# Function for learning curve

def plot_learning_curve(estimator, title, X, y, ylim=↩
   None, cv=None,
                        n_jobs=None, train_sizes=np.↩
                           linspace(.1, 1.0, 5)):

    plt.figure()
plt.title(title)
if ylim is not None:
    plt.ylim(*ylim)
plt.xlabel("Training examples")
plt.ylabel("Score")
train_sizes, train_scores, test_scores = ↩
   learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs, ↩
       train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - ↩
   train_scores_std,
                 train_scores_mean + ↩
                    train_scores_std, alpha=0.1,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - ↩
   test_scores_std,
                 test_scores_mean + test_scores_std,↩
                     alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color↩
   ="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color=↩
   "g",
         label="Cross-validation score")

plt.legend(loc="best")
return plt
```

```
#dataset = pd.read_excel('German_Credit_Data.xlsx')
#dataset.drop(["Acc","Telephone","Gender"],axis = 1, ↩
   inplace = True)
dataset = pd.read_excel('Australian_Credit_Data.xlsx')

# Data Pre-Processing(Splitting data, Categorical data, ↩
   Feature Scaling)
from Functions import data_preprocessing
X_train,X_test,y_train,y_test,X,y = data_preprocessing(↩
   dataset)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression()
```

```
classifier_lr.fit(X_train, y_train)

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors = 5, ↩
   metric = 'euclidean')
classifier_knn.fit(X_train, y_train)

# Fitting Kernel SVM to the Training set
from sklearn.svm import SVC
classifier_svc = SVC(kernel = 'poly', degree = 2)
classifier_svc.fit(X_train, y_train)

# Fitting Decision Tree Classification to the Training ↩
   set
from sklearn.tree import DecisionTreeClassifier
classifier_dt = DecisionTreeClassifier(criterion = 'gini↩
   ', min_samples_split = 10)
classifier_dt.fit(X_train, y_train)

# Fitting Random Forest Classification to the Training ↩
   set
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(n_estimators = ↩
   10, criterion = 'entropy', min_samples_split = 2)
classifier_rf.fit(X_train, y_train)

# Fitting Naive Bayes Classification to the Training set
from sklearn.naive_bayes import BernoulliNB
classifier_nb = BernoulliNB(alpha = 1, binarize = 0.0 )
classifier_nb.fit(X_train, y_train)

# Predicting the Test set results
y_pred_lr  = classifier_lr.predict(X_test)
```

```
y_pred_knn = classifier_knn.predict(X_test)
y_pred_svc = classifier_svc.predict(X_test)
y_pred_dt  = classifier_dt.predict(X_test)
y_pred_rf  = classifier_rf.predict(X_test)
y_pred_nb  = classifier_nb.predict(X_test)

# Plot Confusion Matrix before tuning
from Functions import  con_mat_plot
print('Confusion Matrix for Logistic Regression')
con_mat_plot(y_test,y_pred_lr)
print('Confusion Matrix for KNN')
con_mat_plot(y_test,y_pred_knn)
print('Confusion Matrix for SVM')
con_mat_plot(y_test,y_pred_svc)
print('Confusion Matrix for Decision Tree')
con_mat_plot(y_test,y_pred_dt)
print('Confusion Matrix for Random Forest')
con_mat_plot(y_test,y_pred_rf)
print('Confusion Matrix for Naive Bayes')
con_mat_plot(y_test,y_pred_nb)


# Applying K-Fold Cross Validation
from Functions import  k_fold_cross
accuracies_lr  = k_fold_cross(classifier_lr, X_train, ↩
   y_train)
accuracies_knn = k_fold_cross(classifier_knn, X_train, ↩
   y_train)
accuracies_svc = k_fold_cross(classifier_svc, X_train, ↩
   y_train)
accuracies_dt  = k_fold_cross(classifier_dt, X_train, ↩
   y_train)
accuracies_rf  = k_fold_cross(classifier_rf, X_train, ↩
   y_train)
```

```
accuracies_nb  = k_fold_cross(classifier_nb, X_train, ↩
   y_train)

print('LR: Mean Accuracy before tuning',accuracies_lr.↩
   mean()*100.)
print('KNN :Mean Accuracy before tuning',accuracies_knn.↩
   mean()*100.)
print('SVC :Mean Accuracy before tuning',accuracies_svc.↩
   mean()*100.)
print('Decision Tree :Mean Accuracy before tuning',↩
   accuracies_dt.mean()*100.)
print('Random Forest :Mean Accuracy before tuning',↩
   accuracies_rf.mean()*100.)
print('Naive Bayes :Mean Accuracy before tuning',↩
   accuracies_nb.mean()*100.)

# Applying Grid Search to find the best model and the ↩
   best parameters
from Functions import  choosing_parameters


# Best parameters for Logistic Regression

parameters_lr = [{'C': [1, 5, 10], 'tol': [1e-4,1e-5,1e↩
   -6,1e-10]}]
best_para_lr  = choosing_parameters(parameters_lr,↩
   classifier_lr,X_train,y_train)
C_lr          = best_para_lr[0]
tole_lr       = best_para_lr[1]

# Best parameters for KNN
parameters_knn = [{'n_neighbors': [3, 5, 7, 3, 9, 11, ↩
   13], 'metric': ['minkowski'],
                   'p': [1,2,3,4,5,6,7]}]
```

```
best_para_knn  = choosing_parameters(parameters_knn,↩
   classifier_knn,X_train,y_train)
N_knn          = best_para_knn[0]
metric_knn     = best_para_knn[1]
p_knn          = best_para_knn[2]

# Best parameters for SVM
parameters_svc = [{'kernel': ['rbf','poly'], 'degree': ↩
   [1, 2, 3, 4, 5, 6, 7]}]
best_para_svc  = choosing_parameters(parameters_svc,↩
   classifier_svc,X_train,y_train)
ker_svc        = best_para_svc[0]
deg_svc        = best_para_svc[1]

# Best parameters for Decision Tree
parameters_dt = [{'criterion': ['gini','entropy'], '↩
   min_samples_split': [2,4,6,8,10,15]}]
best_para_dt  = choosing_parameters(parameters_dt,↩
   classifier_dt,X_train,y_train)
criteria_dt   = best_para_dt[1]
min_split_dt  = best_para_dt[0]


# Best parameters for Random Forest
parameters_rf    = [{'criterion': ['gini','entropy'],'↩
   min_samples_split': [2,5, 10, 15, 20],
                    'n_estimators': [10, 20, 30, 40, ↩
                       50,100,150]}]
best_para_rf     = choosing_parameters(parameters_rf,↩
   classifier_rf,X_train,y_train)
min_samp_spl_rf = best_para_rf[0]
n_est_rf         = best_para_rf[1]
criterion_rf     = best_para_rf[2]
```

```
# Best parameters for Naive bayes
parameters_nb = [{'alpha': [1, 5, 7, 10, 20, 50, 60, ↩
   100],
                 'binarize': [0.0, 0.1, 0.2, 0.3, 0.4, ↩
                    0.5]}]
best_para_nb  = choosing_parameters(parameters_nb,↩
   classifier_nb,X_train,y_train)
alfa_nb       = best_para_nb[1]
bi_nb         = best_para_nb[0]

#Selecting the best parameters and apply the ↩
   classification methods

# Logistic Regression
classifier_lr = LogisticRegression(C = int(C_lr), tol = ↩
   float(tole_lr) )
classifier_lr.fit(X_train, y_train)

# KNN
classifier_knn = KNeighborsClassifier(n_neighbors = int(↩
   N_knn), metric = str(metric_knn), p = int(p_knn))
classifier_knn.fit(X_train, y_train)

# SVM
classifier_svc = SVC(kernel = str(ker_svc), degree = int↩
   (deg_svc))
classifier_svc.fit(X_train, y_train)

# Decision Tree Classification
classifier_dt = DecisionTreeClassifier(criterion = str(↩
   criteria_dt), min_samples_split = int(min_split_dt))
classifier_dt.fit(X_train, y_train)

# Random Forest Classification
```

```
classifier_rf = RandomForestClassifier(n_estimators = ↩
   int(n_est_rf), criterion = str(criterion_rf),
                                        min_samples_split = ↩
                                           int(↩
                                           min_samp_spl_rf))
classifier_rf.fit(X_train, y_train)

# Naive Bayes Classification
classifier_nb = BernoulliNB(alpha = int(alfa_nb), ↩
   binarize = float(bi_nb))
classifier_nb.fit(X_train, y_train)


# Predict the Test set results
y_pred_lr = classifier_lr.predict(X_test)
y_pred_knn = classifier_knn.predict(X_test)
y_pred_svc = classifier_svc.predict(X_test)
y_pred_dt = classifier_dt.predict(X_test)
y_pred_rf = classifier_rf.predict(X_test)
y_pred_nb = classifier_nb.predict(X_test)

# Create Confusion Matrix
from Functions import  con_mat_plot
print('Confusion Matrix for Logistic Regression')
con_mat_plot(y_test,y_pred_lr)
print('Confusion Matrix for KNN')
con_mat_plot(y_test,y_pred_knn)
print('Confusion Matrix for SVM')
con_mat_plot(y_test,y_pred_svc)
print('Confusion Matrix for Decision Tree')
con_mat_plot(y_test,y_pred_dt)
print('Confusion Matrix for Random Forest')
con_mat_plot(y_test,y_pred_rf)
print('Confusion Matrix for Naive Bayes')
```

```
con_mat_plot(y_test,y_pred_nb)

# Apply K-Fold Cross Validation
from Functions import  k_fold_cross
accuracies_lr  = k_fold_cross(classifier_lr, X_train, ↩
   y_train)
accuracies_knn = k_fold_cross(classifier_knn, X_train, ↩
   y_train)
accuracies_svc = k_fold_cross(classifier_svc, X_train, ↩
   y_train)
accuracies_dt  = k_fold_cross(classifier_dt, X_train, ↩
   y_train)
accuracies_rf  = k_fold_cross(classifier_rf, X_train, ↩
   y_train)
accuracies_nb  = k_fold_cross(classifier_nb, X_train, ↩
   y_train)
print('LR: Mean Accuracy after tuning',accuracies_lr.↩
   mean()*100.)
print('KNN :Mean Accuracy after tuning',accuracies_knn.↩
   mean()*100.)
print('SVM :Mean Accuracy after tuning',accuracies_svc.↩
   mean()*100.)
print('Decision Tree :Mean Accuracy after tuning',↩
   accuracies_dt.mean()*100.)
print('Random Forest :Mean Accuracy after tuning',↩
   accuracies_rf.mean()*100.)
print('Naive Bayes :Mean Accuracy after tuning',↩
   accuracies_nb.mean()*100.)

from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Logistic Regression
classifier_lr = LogisticRegression(C = int(C_lr), tol = ↩
   float(tole_lr) )
classifier_lr.fit(X_train, y_train)

# KNN
classifier_knn = KNeighborsClassifier(n_neighbors = int(↩
   N_knn), metric = str(metric_knn), p = int(p_knn))
classifier_knn.fit(X_train, y_train)

# SVM
classifier_svc = SVC(kernel = str(ker_svc), degree = int↩
   (deg_svc))
classifier_svc.fit(X_train, y_train)

# Decision Tree Classification
classifier_dt = DecisionTreeClassifier(criterion = str(↩
   criteria_dt), min_samples_split = int(min_split_dt))
classifier_dt.fit(X_train, y_train)

# Random Forest Classification
classifier_rf = RandomForestClassifier(n_estimators = ↩
   int(n_est_rf), criterion = str(criterion_rf),
                                       min_samples_split = ↩
                                          int(↩
                                          min_samp_spl_rf))
classifier_rf.fit(X_train, y_train)

# Naive Bayes Classification
classifier_nb1 = BernoulliNB(alpha = int(alfa_nb), ↩
   binarize = float(bi_nb))
classifier_nb1.fit(X_train, y_train)

from sklearn.naive_bayes import GaussianNB
```

```
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)

# Decision boundary plot

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_lr ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_knn ,X_train, y_train,↩
    X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_svc ,X_train, y_train,↩
    X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_dt ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_rf ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

from Functions import plot_decision_boundary
plot_decision_boundary(classifier_nb ,X_train, y_train, ↩
   X_test, cmap='Paired_r')

# Learning curve

from sklearn.model_selection import ShuffleSplit
cv = ShuffleSplit(test_size=0.2, random_state = 0)
```

```
from Functions import plot_learning_curve
title = "Learning Curves (Logistic Regression)"
estimator = classifier_lr
plot_learning_curve(estimator, title, X, y, ylim=(0.25, ↩
   1.25), cv=cv, n_jobs=4)

title = "Learning Curves (KNN)"
estimator = classifier_knn
plot_learning_curve(estimator, title, X, y, (0.25, 1.25)↩
   , cv=cv, n_jobs=4)

title = "Learning Curves (SVM)"
estimator = classifier_svc
plot_learning_curve(estimator, title, X, y, (0.25, 1.25)↩
   , cv=cv, n_jobs=4)

title = "Learning Curves (Decision Tree)"
estimator = classifier_dt
plot_learning_curve(estimator, title, X, y, (0.25, 1.25)↩
   , cv=cv, n_jobs=4)

title = "Learning Curves (Random Forest)"
estimator = classifier_rf
plot_learning_curve(estimator, title, X, y, (0.25, 1.25)↩
   , cv=cv, n_jobs=4)

title = "Learning Curves (Naive Bayes)"
estimator = classifier_nb1
plot_learning_curve(estimator, title, X, y, (0.25, 1.25)↩
   , cv=cv, n_jobs=4)
```

```
# Import the dataset

dataset = pd.read_excel('German_Credit_Data.xlsx')
#dataset = pd.read_excel('Australian_Credit_Data.xlsx')

# Data Pre-Processing(Splitting data, Categorical data, ↩
   Feature Scaling)
from Functions import data_preprocessing
X_train,X_test,y_train,y_test,X,y = data_preprocessing(↩
   dataset)


# Make the ANN!

# Importing the Keras libraries and packages

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu', input_dim = np.↩
       shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))
    # Adding the output layer
    classifier.add(Dense(1, kernel_initializer = '↩
      uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = '↩
      binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
classifier = KerasClassifier(build_fn = ann_model, ↩
   batch_size = 25, epochs = 100)

# Cross validation
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

kfold = StratifiedKFold(n_splits=10, shuffle=True, ↩
   random_state = 2)
results = cross_val_score(classifier, X_test, y_test, cv↩
   =kfold)
print(results.mean())


# Making predictions and evaluating the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

from Functions import  con_mat_plot
```

```
con_mat_plot(y_test,y_pred)



# Tuning the ANN
import keras
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(units = 10, kernel_initializer ↩
      = 'uniform', activation = 'relu', input_dim = np.↩
      shape(X_test)[1]))
    classifier.add(Dropout(p = 0.1))
    classifier.add(Dense(units = 10, kernel_initializer ↩
      = 'uniform', activation = 'relu'))
    classifier.add(Dropout(p = 0.1))
    classifier.add(Dense(units = 1, kernel_initializer =↩
       'uniform', activation = 'sigmoid'))
    classifier.compile(optimizer = optimizer, loss = '↩
      binary_crossentropy', metrics = ['accuracy'])
    return classifier


classifier = KerasClassifier(build_fn = build_classifier↩
   )
parameters = {'batch_size' : [25, 50],
              'epochs' : [100, 200],
              'optimizer' : ['adam' , 'rmsprop']}

grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, y_train)
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
print('Best Accuracy = ',best_accuracy)
best_parameters = np.array(best_parameters.values())
epo             = int(best_parameters[0])
opt             = str(best_parameters[1])
bs              = int(best_parameters[2])

# Defining model with tuned parameteres

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
      uniform', activation = 'relu', input_dim = np.↩
      shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
      uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
      uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(1, kernel_initializer = '↩
      uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = opt, loss = '↩
      binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
classifier = KerasClassifier(build_fn = ann_model, ↩
   batch_size = bs, epochs = epo)
kfold = StratifiedKFold(n_splits=10, shuffle=True, ↩
   random_state = 2)
results = cross_val_score(classifier, X_test, y_test, cv↩
   =kfold)
print(results.mean())


# Making predictions and evaluating the model
classifier.fit(X_train, y_train)

# Predicting the Test set results

y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

# Confusion matrix

from Functions import  con_mat_plot
con_mat_plot(y_test,y_pred)

# Applying PCA
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

def ann_model():

    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu', input_dim = np.↩
       shape(X_test)[1]))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the second hidden layer
    classifier.add(Dense(10, kernel_initializer = '↩
       uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(1, kernel_initializer = '↩
       uniform', activation = 'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = '↩
       binary_crossentropy', metrics = ['accuracy'])

    return classifier

# Fitting the ANN to the Training set
```

```
classifier = KerasClassifier(build_fn = ann_model, ↩
   batch_size = 25, epochs = 100)
classifier.fit(X_train, y_train)

# Decision boundary

def plot_decision_boundary(clf, X, Y, X_test, cmap='↩
   Paired_r'):
    clf.predict(X_test)
    h = 0.02
    x_min, x_max = X[:,0].min() - 10*h, X[:,0].max() + ↩
       10*h
    y_min, y_max = X[:,1].min() - 10*h, X[:,1].max() + ↩
       10*h
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                          np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,10))
    plt.contourf(xx, yy, Z, cmap=cmap, alpha=0.25)
    plt.contour(xx, yy, Z, colors='k', linewidths=0.7)
    plt.ylabel('x2')
    plt.xlabel('x1')
    plt.grid()
    plt.scatter(X[:,0], X[:,1], c=Y, cmap=cmap, ↩
       edgecolors='k');

plot_decision_boundary(classifier ,X_train, y_train, ↩
   X_test, cmap='Paired_r')
```

References

- ✓ UC Irvine Machine Learning Repository Australian Credit Data. http://archive.ics.uci.edu/ml/datasets/Statlog+%28Australian+Credit+Approval%29
- ✓ UC Irvine Machine Learning Repository German Credit Data. https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
- ✓ Principal Component Analysis. https://en.wikipedia.org/wiki/Principal_component_analysis/.
- ✓ Suman Kumar Mohapatra Trilok Nath Pandey, Alok Kumar Jagadev and Satchidananda Dehuri. Credit Risk Analysis using Machine Learning Classifiers. IEEE, 2017.
- ✓ Artificial Intelligence - All in One by Andrew Ng. https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN/.
- ✓ Kaushik Roy Prajesh P. Anchalia. The k-Nearest Neighbor Algorithm Using MapReduce Paradigm. IEEE, 2014.