Problem Statement: Implement the Single Layer Perceptron Learning algorithm.

Introduction: Perceptron Learning algorithm is used for supervised learning for binary classification. In this problem, a single layer perceptron learning algorithm was used to perform classification of a given dataset. Single Layer Perceptron means only one perceptron (node) is used for classification. It performs a weighted sum of its inputs, adds a bias value, compares this to a defined internal threshold value and turns on only if the threshold value is exceeded. If not, it stays off. In this way the prediction is performed. Because the inputs are passed through the model perceptron to produce the output, this system is known as feedforward network. The formula to calculate the output is:

$$\mathbf{y}(\boldsymbol{t}) = \boldsymbol{f_h} \sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \boldsymbol{w_i}(\boldsymbol{t})\boldsymbol{x_i}(\boldsymbol{t}) + \boldsymbol{b_i}$$

Here, $\mathrm{y}(t)$ is the final output. f_h is the desired function where comparison with threshold value is performed, $w_i(t)$ is the weight, $x_i(t)$ is the input and b_i is the bias value.

As this is a supervised learning technique, the weight and bias values are updated. The formula for updating the values are:

$$\boldsymbol{W = W + (Y_0 - Y) * input * learning\_rate}$$

$$\boldsymbol{bias = bias + (Y_0 - Y) * learning\_rate}$$

here, W is the weight, Y_0 is the prediction, Y is the actual label.

Dataset Description: The dataset used for K-Nearest Neighbor classification is the "Social Network Ads" dataset from Kaggle. It consists 400 entries, each with 5 columns. The dataset represents the details of 400 persons about their User ID, Gender, Age, EstimatedSalary, and whether they have purchased products from social network ad or not. The first 5 entries of the dataset are demonstrated below for better understanding.

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Data Preprocessing: The dataset was processed before utilizing for classification. First the dataset was separated into data and label. The 2nd and 3rd index column was taken as values and the 4th index column was taken as labels. The original shape of dataset was (400, 5). After separating the shape of values was (400, 2) and the shape of labels was (400,). Later the entire dataset was split into train value, train label and test value, test label.

Methodology: Before performing the classification, the dataset was scaled using MinMax scaler in the range of 0 and 1. This makes the classification faster. The number of input features used was 2 and the learning rate was 0.001. The model was trained for 500 epochs to perform the classification.

Code:

```
from sklearn.metrics import confusion_matrix, roc_curve, auc

import itertools

from itertools import cycle

from sklearn import metrics

from scipy import interp

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (20.0, 10.0)

%matplotlib inline


df = pd.read_csv('../input/input-dataset/Social_Network_Ads.csv')

df.head()


X = df.iloc[:,2:4].values

y = df.iloc[:,4].values

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)


from sklearn.preprocessing import MinMaxScaler
```

```
sc_X = MinMaxScaler(feature_range=(0, 1))
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
print(X_train,"\n------------------------------\n",X_test)

class Perceptron_Learning(object):

    def __init__(self, no_of_features, learning_rate):
        self.learning_rate = learning_rate
        self.weights = np.zeros(no_of_features)
        self.bias = 0

    def predict(self, data):
        sum = np.dot(data, self.weights) + self.bias
        return np.where(sum>0,1,0)

    def fit_function(self, train_data, train_labels):
            for data, label in zip(train_data, train_labels):
                prediction = self.predict(data)
                self.weights += (label - prediction) * data * self.learning_rate
                self.bias += self.learning_rate * (label - prediction)



from sklearn.metrics import confusion_matrix,classification_report
test_acc=[]
train_acc=[]
model = Perceptron_Learning(2,0.001)
epochs = 500
for i in range(0, epochs):
  model.fit_function(X_train,y_train)

  train_pred = model.predict(X_train)
  train_cm = confusion_matrix(y_train,train_pred)
```

```
 train_accuracy = (train_cm[0][0]+train_cm[1][1])/(train_cm[0][0]+train_cm[0][1]+train_cm[1][0]+train_cm[1][1])
 train_acc.append(train_accuracy)

 y_pred = model.predict(X_test)
 cm = confusion_matrix(y_test,y_pred)
 accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
 test_acc.append(accuracy)



plt.plot(train_acc, label = 'Train Accuracy')
plt.plot(test_acc, label = 'Test Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend()
#plt.title("Learning rate =" + str(learning_rate))
plt.show()



train_accuracy = (train_cm[0][0]+train_cm[1][1])/(train_cm[0][0]+train_cm[0][1]+train_cm[1][0]+train_cm[1][1])*100
print('Train Accuracy = ',train_accuracy,'%')



accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])*100
print('Test Accuracy = ',accuracy,'%')

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
```

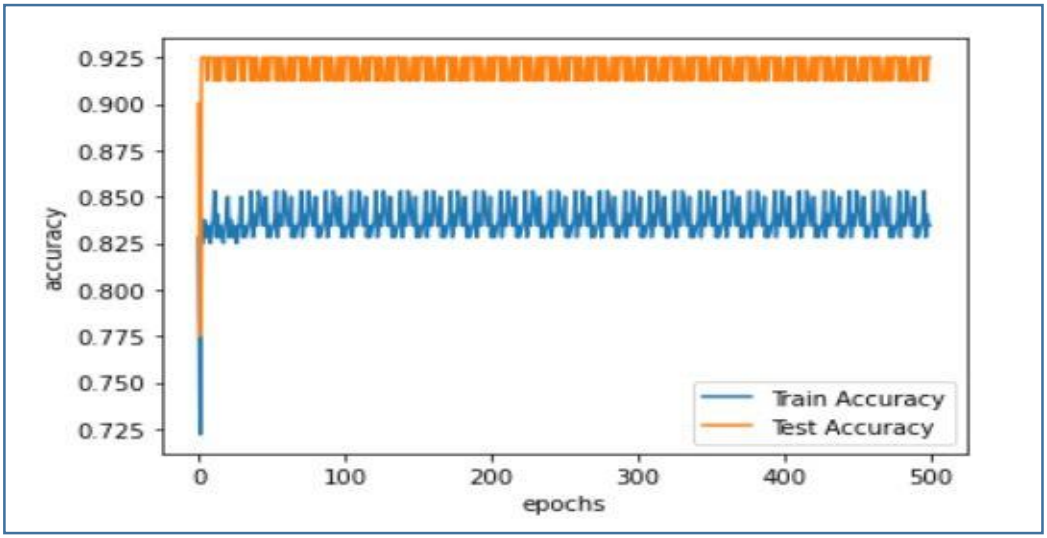
```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    #plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


cm_plot_labels = ['PURCHASED', 'NOT_PURCHASED']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='')

lw = 2
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
```

```
roc_auc = dict()
for i in range(2):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_pred)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_pred.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(2)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(2):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= 2

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(1)
plt.plot(fpr["micro"], tpr["micro"],
        label='micro-average ROC curve (area = {0:0.2f})'
              ''.format(roc_auc["micro"]),
        color='deeppink', linestyle=':', linewidth=4)
```

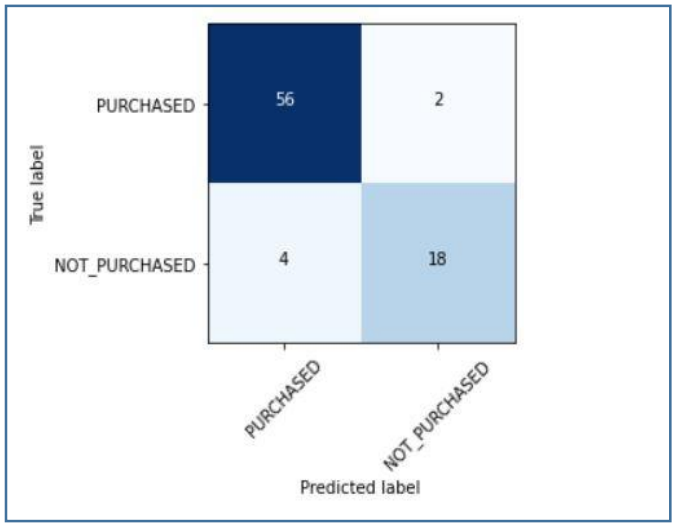
```
plt.plot(fpr["macro"], tpr["macro"],
        label='macro-average ROC curve (area = {0:0.2f})'
            ''.format(roc_auc["macro"]),
        color='navy', linestyle=':', linewidth=4)


colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(2), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
            label='ROC curve of class {0} (area = {1:0.2f})'
            ''.format(i, roc_auc[i]))


plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
#plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()


print(classification_report(y_test, y_pred))
```

Experimental Result:

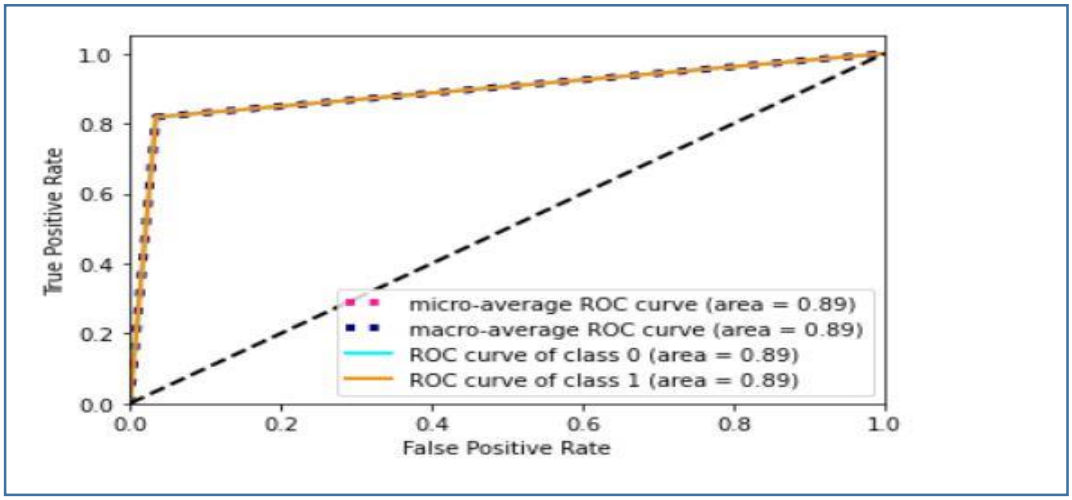
Accuracy Curve:



Confusion Matrix:



Receiver Operating Characteristics Curve:



Classification Report:

	precision	recall	f1-score	support
0	0.93	0.97	0.95	58
1	0.90	0.82	0.86	22
accuracy			0.93	80
macro avg	0.92	0.89	0.90	80
weighted avg	0.92	0.93	0.92	80

Conclusion: Using this technique, a training accuracy of 83.43% and testing accuracy of 92.50% was achieved. As the training accuracy is less, the model has an underfitting problem also known as bias problem. This can be solved using a larger amount of data.