Pattern Recognition Lab Assignment-2

Submitted by: Yukti Khurana, 2017UCP1234
Submitted to: Dr Deepak Ranjan Nayak
Semester-8 2021

DAY-5

1. Write a program to reduce the dimension of feature space for a given dataset using principal component analysis (PCA). (Load ionosphere dataset "ionosphere.mat")

CODE

PCA.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

print("Principal Component Analysis by Yukti Khurana")
df = pd.read_csv("Ionosphere.csv",header=None)
df = df[1:]
# split into training and testing sets
X = df.iloc[:, 1:].values
y = df.iloc[:, 0].values
# Splitting the training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.70, random_state = 41, stratify = y)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Standardize the features
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

# finding the Covariance Matrix of training data - X
cov_mat = np.cov(X_train_std.T)
# Eigen values and Eigen vectors using the covariance matrix
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

# calculate cumulative sum of explained variances
total_evals = sum(eigen_vals)
var_exp = [(i / total_evals) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)

# Feature Extraction
# Make a list of (eigenvalue, eigenvector) tuples
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i]) for i in
```

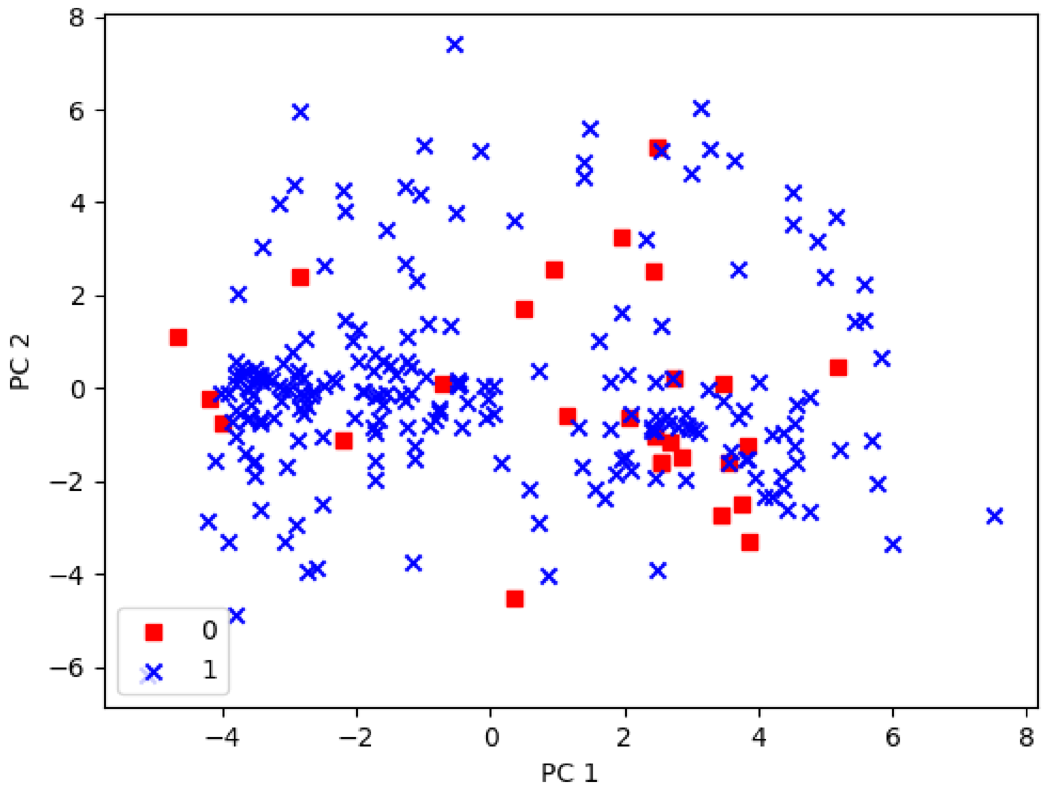
```
range(len(eigen_vals))]

# Sort the (eigenvalue, eigenvector) tuples from high to low (in decreasing order of eigen values)
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
#print("Sorted Eigen Value and Eigen vector pairs")
#print(eigen_pairs)
# we collect the two eigenvectors that correspond to the two largest eigenvalues,
w = np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:, np.newaxis]))
#print('Matrix W:\n', w)

# Using the projection matrix, we can now transform a sample x (represented as a 1 x 34-dimensional row vector)
# onto the PCA subspace (the principal components one and two) obtaining x′
# now a two-dimensional sample vector consisting of two new features
# x' = xW
X_train_std[0].dot(w)

# we can transform the entire 245 X 34-dimensional training dataset
# onto the two principal components by calculating the matrix dot product
# X' = XW
X_train_pca = X_train_std.dot(w)
print()
print("Training Dataset BEFORE Dimension Reduction : ")
print(X_train)
print()
print("Training Dataset AFTER reduction of Dimenstions : ")
print(X_train_pca)
# Hence the dimensions of dataset were reduced from 34 to 2
# visualizing the reduced Ionosphere training dataset
colors = ['r', 'b', 'g']
markers = ['s', 'x', 'o']
for l, c, m in zip(np.unique(y_train), colors, markers):
    plt.scatter(X_train_pca[y_train==l, 0],
                X_train_pca[y_train==l, 1],
                c=c, label=l, marker=m)
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.legend(loc='lower left')
plt.show()

print()
print("Dimensions Before PCA of Ionosphere Training dataset = ", X_train.shape)
print("Dimensions After PCA of Ionosphere Training dataset = ", X_train_pca.shape)
```

INPUT

- Ionosphere dataset is used. Any other dataset can also be used.
- The dimensions of the dataset are reduced from 34 to 4.

OUTPUT

```
C:\Users\lenovo\PycharmProjects\PR-Lab\venv\Scripts\python.exe C:/Users/lenovo/PycharmProjec
Principal Component Analysis by Yukti Khurana
(245, 34) (106, 34) (245,) (106,)

Training Dataset BEFORE Dimension Reduction :
[['0' '0.68729' '1' ... '0.56522' '0.23913' '0']
 ['0' '1' '0.45455' ... '0.72727' '0.27273' '1']
 ['0' '-0.00641' '-0.5' ... '0' '0' '0']
 ...
 ['0' '0.94653' '0.28713' ... '0.50376' '-0.0598' '1']
 ['0' '0.58459' '-0.35526' ... '0.44767' '-0.10309' '1']
 ['0' '0.89706' '0.38235' ... '-0.28676' '-0.56618' '1']]

Training Dataset AFTER reduction of Dimenstions :
[[-1.97119159  1.25626219]
 [-3.00958592 -0.01008441]
 [ 3.464617   -0.2630596 ]
 [ 3.7740374  -0.47154361]
 [-0.60403953  1.34903896]
 [-3.5397641  -0.15029433]
 [ 3.84821173 -1.23512379]
 [-1.65728483 -0.1496943 ]
 [-1.74915978 -0.85771307]
 [-2.30218521  0.14499792]
 [-2.03811133 -0.63524659]
 [ 2.04069883  0.27915954]
 [ 1.30957002 -0.81917652]
 [ 1.47028821  5.58801117]
 [ 5.76380016 -2.05424425]
 [-0.52273803  3.79050369]
 [-0.33593247 -0.29279171]
 [-1.54693493  3.41299961]
 [-3.62984943  0.01506607]
 [-3.41981118 -0.66666601]
 [ 4.42093929 -2.62986532]
 [ 3.48393342  0.10010655]
 [-1.23846608  0.5929103 ]
 [-3.50979829 -1.55047355]
 [ 4.36472352 -0.94392156]
 [-1.6088112  -0.34466074]
 [ 4.97585285  2.39167213]]

Dimensions Before PCA of Ionosphere Training dataset =  (245, 34)
Dimensions After PCA of Ionosphere Training dataset =  (245, 2)

Process finished with exit code 0
```



For Visualization of Dataset features after Dimension reduction by PCA

2. Write a program to reduce the dimension of feature space for a given dataset using Linear Discriminant Analysis (LDA). (Load ionosphere dataset “ionosphere.mat”)

CODE

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

print("\nLinear Discriminant Analysis by Yukti Khurana")
df = pd.read_csv("Ionosphere.csv",header=None)
df = df[1:]

#grouping based on the class of the classification :
df_grouped = df.groupby(0)

df_group0 = df_grouped.get_group("0")
df_group1 = df_grouped.get_group("1")

X_group0 = df_group0.iloc[:, 1:].values
y_group0 = df_group0.iloc[:, 0].values

X_group1 = df_group1.iloc[:, 1:].values
```

```
y_group1 = df_group1.iloc[:, 0].values

X_train1, X_test1, y_train1, y_test1 = train_test_split(X_group1, y_group1,
train_size = 0.10, random_state = 41, stratify = y_group1)
X_train0, X_test0, y_train0, y_test0 = train_test_split(X_group0, y_group0,
train_size = 0.84, random_state = 41, stratify = y_group0)

sc = StandardScaler()
X_train_std1 = sc.fit_transform(X_train1)
X_test_std1 = sc.transform(X_test1)

X_train_std0 = sc.fit_transform(X_train0)
X_test_std0 = sc.transform(X_test0)

mean1 = np.mean(X_train_std1, axis=0)
mean0 = np.mean(X_train_std0, axis=0)

def Mean(samples):
    s = np.array(samples)
    m = np.mean(s, axis=0)
    return np.array(m)

def getZ(samples, mean):
    return np.array(samples) - np.array(mean)

print("Before Dimension Reduction of Class-0")
print(np.size(X_train_std0,0), np.size(X_train_std0,1))

print("\nBefore Dimension Reduction of Class-1")
print(np.size(X_train_std1,0), np.size(X_train_std1,1))
print()

u0 = getZ(X_train_std0, mean0)
u1 = getZ(X_train_std1, mean1)

print("Class-0 Data : ")
print(u0)
print("\nClass-1 Data : ")
print(u1)
print()

def getScatterMat(Z):
    return np.dot(np.array(Z).T, np.array(Z))


m0 = getScatterMat(u0)
m1 = getScatterMat(u1)
print("SCATTER MATRIX FOR CLASS 0 : ")
print(m0)
print("\nSCATTER MATRIX FOR CLASS 1 : ")
print(m1)
print()

#Sw is the sum of the scatter matrix for both the classes
sw = np.add(np.array(m0), np.array(m1))

t1 = np.subtract(np.array(mean0), np.array(mean1))
t2 = t1[np.newaxis]
print(np.size(t2,1))
t3 = t2.T
print(np.size(t3,0))
```

```
#Sb is the between class scatter calculated using the multiplation of
difference transpose and difference of the mean matrices
sb = np.matmul(t3, t2)
print(sb)

#Sb-1Sw for eigen vector calculation
req = np.matmul(np.linalg.inv(sw), sb)

#Calculation of eigen values and vectors
w, v = np.linalg.eig(req)
eigen_pairs = [(np.abs(w[i]), v[:, i]) for i in range(len(w))]

#sorting the eigen vectors based on eigen values
eigen_pairs.sort(key=lambda k: k[0], reverse=True)

#taking reduced dimensionality to 4
matw = np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:,
np.newaxis], eigen_pairs[2][1][:, np.newaxis], eigen_pairs[3][1][:,
np.newaxis]))

data0 = np.matmul(matw.T, u0.T)
data1 = np.matmul(np.array(matw).T, u1.T)

print("After Dimension Reduction of Class-0 ")
print(data0)
print("\nAfter Dimension Reduction of Class-1 ")
print(data1)
print("\nDimensions After LDA of Ionosphere dataset = ", np.size(data1,0))
```

INPUT

- Ionosphere dataset is used. Any other dataset can also be used.
- The dimensions of the dataset are reduced from 34 to 4.

OUTPUT

```
C:\Users\lenovo\PycharmProjects\PR-Lab\venv\Scripts\python.exe C:/Users/lenovo/PycharmProjects/PR-L

Linear Discriminant Analysis by Yukti Khurana
Before Dimension Reduction of Class-0
31 34

Before Dimension Reduction of Class-1
31 34

Class-0 Data :
[[ 0.          0.98361184 -1.35622245 ...  1.48804762 -1.59140131
   0.        ]
 [ 0.          0.98361184  1.04622875 ...  0.         -0.09644856
   0.        ]
 [ 0.          0.98361184 -1.35622245 ...  1.48804762  1.39850418
   0.        ]
 ...
 [ 0.         -1.45574553  1.04622875 ...  0.         -0.09644856
   0.        ]
 [ 0.          0.98361184 -1.35622245 ...  1.48804762 -1.59140131
   0.        ]
 [ 0.         -0.23606684 -0.15499685 ...  0.         -0.09644856
   0.        ]]
```

```
Class-1 Data :
[[ 0.         -1.36131556 -0.02179408 ... -0.72484688 -0.3158487
  -2.04124145]
 [ 0.         -1.02841518 -3.15499426 ... -0.71348483 -0.15184373
  -2.04124145]
 [ 0.          0.51494797 -0.26626361 ... -0.4634983   1.0578293
   0.48989795]
 ...
 [ 0.          0.78655769  2.55349296 ... -0.71348483 -0.15184373
  -2.04124145]
 [ 0.         -2.19230889 -0.2206242  ... -1.36481499 -1.02428576
   0.48989795]
 [ 0.         -0.23381897 -0.42098446 ...  0.99330561 -0.13137558
   0.48989795]]

SCATTER MATRIX FOR CLASS 0 :
[[  0.           0.           0.         ...   0.           0.
    0.        ]
 [  0.          31.           4.72615891 ...   3.62987993   6.58763047
    0.        ]
 [  0.           4.72615891  31.         ... -10.72488536  -0.46342594
    0.        ]
 ...
```

```
 ...
 [  0.          3.62987993 -10.72488536 ...  31.          -8.89824351
    0.        ]
 [  0.          6.58763047  -0.46342594 ...  -8.89824351  31.
    0.        ]
 [  0.          0.           0.         ...   0.           0.
    0.        ]]

SCATTER MATRIX FOR CLASS 1 :
[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.         31.         13.01505277 ... 10.09782547  8.10232686
  14.15025297]
 [ 0.         13.01505277 31.         ...  0.2836182   7.78560142
   7.31378133]
 ...
 [ 0.         10.09782547  0.2836182  ... 31.         -1.89974498
   3.81572361]
 [ 0.          8.10232686  7.78560142 ... -1.89974498 31.
   8.08673181]
 [ 0.         14.15025297  7.31378133 ...  3.81572361  8.08673181
  31.        ]]

34
```

```
After Dimension Reduction of Class-0
[[ 3.43906159+0.j         -0.56837193+0.j          2.32523475+0.j
  -0.97293169+0.j          2.58796265+0.j         -0.78872532+0.j
  -0.19233561+0.j         -0.6067053 +0.j         -0.56573965+0.j
   0.93860887+0.j          2.30103602+0.j         -2.32068661+0.j
   1.12297947+0.j         -1.11186696+0.j         -2.44620174+0.j
  -2.53524917+0.j         -1.18845658+0.j         -0.20277092+0.j
   3.90598946+0.j          0.59553258+0.j          0.48209836+0.j
  -0.73261506+0.j         -2.02941968+0.j          0.90992842+0.j
  -2.36533197+0.j         -1.45813261+0.j         -1.08627942+0.j
   1.07514496+0.j         -2.43262672+0.j          4.70959513+0.j
  -0.78872532+0.j        ]
 [-1.43474246+0.j         -0.7066018 +0.j         -1.30727291+0.j
   1.52725582+0.j         -1.36888666+0.j          0.31834009+0.j
   0.2430081 +0.j          1.69280751+0.j          1.3841668 +0.j
   1.42347255+0.j         -1.64778118+0.j          0.46486915+0.j
  -0.10717643+0.j          1.43246616+0.j         -0.58317329+0.j
  -0.41132409+0.j          0.3953076 +0.j         -0.76936086+0.j
  -1.51518806+0.j         -0.78544669+0.j          0.50382719+0.j
   0.3266211 +0.j          0.92766708+0.j         -1.35872549+0.j
   1.70896846+0.j         -0.75251057+0.j          0.31334701+0.j
  -0.69875346+0.j          1.66912447+0.j         -1.20264526+0.j
   0.31834009+0.j        ]
 [-0.04758233+0.69071358j  0.1550775 +1.48690494j  0.05059018-0.80347427j
```

```
After Dimension Reduction of Class-1
[[-3.54379417+0.j         -1.70437239+0.j          2.46326214+0.j
   3.26509419+0.j          2.07240406+0.j          2.2953561 +0.j
   2.49817244+0.j          2.8130433 +0.j         -2.86721965+0.j
   0.64608739+0.j         -2.66069683+0.j         -0.81050672+0.j
  -3.82989282+0.j          0.64902193+0.j         -0.4452248 +0.j
   1.9688272 +0.j         -1.71800401+0.j          2.11813202+0.j
   2.19006881+0.j          0.74393707+0.j          0.09766266+0.j
   1.67804118+0.j          1.67591738+0.j         -0.89541062+0.j
  -2.18597632+0.j         -0.21156416+0.j          3.18304053+0.j
  -3.41293098+0.j         -3.56969858+0.j         -4.16303692+0.j
   1.66026058+0.j        ]
 [ 1.47108112+0.j          1.02492738+0.j         -0.51991575+0.j
  -0.99687364+0.j         -0.73219112+0.j         -0.94038859+0.j
  -0.98441609+0.j         -0.70002528+0.j          1.85524887+0.j
  -0.1321803 +0.j         -0.08055448+0.j         -0.08835282+0.j
   0.96536415+0.j          0.25948621+0.j         -1.59608115+0.j
  -1.03679584+0.j          2.63341474+0.j         -0.5797331 +0.j
  -0.96600591+0.j         -0.16390562+0.j         -0.01427796+0.j
  -0.25619357+0.j         -0.43858468+0.j          1.88249638+0.j
  -1.03886215+0.j         -0.41132604+0.j         -0.92552244+0.j
   0.25606647+0.j         -0.41847278+0.j          2.61151634+0.j
   0.06105762+0.j        ]
```

```
  0.9555365 +1.00314889j -0.04078162-0.29811594j  0.62126819+0.75878151j
 -0.00452697-0.39004699j -0.26686584-0.31245161j -0.17416867-0.52244562j
 -0.06957142-0.29975367j -1.25307252+0.13245056j -0.18755214-0.1640439j
  0.17435454-0.41187786j  0.20467867-0.01872351j  0.16372134+0.20826564j
 -0.16804236-0.35142115j -0.125027  -0.24189313j -0.34242833-0.6307473j
 -1.07426586-0.02957087j  0.79977521+0.86173174j  0.3956865 +0.67155629j
  0.10227088-0.42248155j -0.27521866+1.14812091j -0.5081156 -0.18479905j
 -0.30923637-0.24439443j]
[-0.29533126-0.24652751j  0.3652637 +0.75084111j  0.44940632-0.5417936j
  0.20020765-0.37415314j  0.01170436+0.1283376j   0.13672781-0.2430396j
  0.11471571+0.18111072j  0.64539669-0.82950473j -0.24650945+1.4360181j
  0.9555365 -1.00314889j -0.04078162+0.29811594j  0.62126819-0.75878151j
 -0.00452697+0.39004699j -0.26686584+0.31245161j -0.17416867+0.52244562j
 -0.06957142+0.29975367j -1.25307252-0.13245056j -0.18755214+0.1640439j
  0.17435454+0.41187786j  0.20467867+0.01872351j  0.16372134-0.20826564j
 -0.16804236+0.35142115j -0.125027  +0.24189313j -0.34242833+0.6307473j
 -1.07426586+0.02957087j  0.79977521-0.86173174j  0.3956865 -0.67155629j
  0.10227088+0.42248155j -0.27521866-1.14812091j -0.5081156 +0.18479905j
 -0.30923637+0.24439443j]]

Dimensions After LDA of Ionosphere dataset =  4

Process finished with exit code 0
```

DAY-6

1. Load the "Iris" dataset and perform k-nearest neighbor classification. Plot the accuracy/error w.r.t different k values. Compare the accuracy with the built-in function of k-NN.

CODE

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from collections import Counter
import matplotlib.pyplot as plt

print("K-Nearest Neighbours ")
# HELPER FUNCTIONS

def Euclidean_distance(a, b):
    # just in case, if the instances are lists or tuples:
    a = np.array(a)
    b = np.array(b)
    #ed = np.sqrt(np.sum((a - b)**2)) # or ed = np.linalg.norm(a - b)
    ed = np.linalg.norm(a - b)
    return ed

def get_K_neighbours(X_train, y_train, test_sample, k):
    # calculate the distances of the test sample from each training sample
    distances = [Euclidean_distance(x, test_sample) for x in X_train]

    # sort the distances and get the k neighbours with smallest distance
    k_nearest = np.argsort(distances)[:k]

    # get the class labels of all k-nearest neighbours
    k_nearest_labels = [y_train[i] for i in k_nearest]
    return k_nearest_labels

def get_majority_class(k_nearest_labels):
    # select the most frequent class labels amongst the k-nearest 
neighbours of the test sample
    most_common_class = Counter(k_nearest_labels).most_common(1)[0][0]
    return most_common_class

def knn_predict(X_train, y_train, X_test, k=3):
    y_pred = []
    for test_sample in X_test:
        k_nearest_labels = get_K_neighbours(X_train, y_train,test_sample,
k)
        prediction = get_majority_class(k_nearest_labels)
        y_pred.append(prediction)
    return np.array(y_pred)

def sklearn_knn_algo(X_train, y_train, X_test, k=3):
    # using the sklearn library for comparison
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    return y_pred
```

```
def accuracy(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    a =  correct / float(len(actual)) * 100.0
    return round(a,3)



# Loading the Dataset- iris
dataset = datasets.load_iris()
target_iris_names =  list(dataset.target_names)
X = dataset.data  # input features
y = dataset.target # target features

# Stratified split of training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.80, random_state = 41, stratify = y)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Initializing the count of each of all classes of flowers -
Versicolor,Setosa,Virginica
# let the total number of classes be 'C'
C = len(np.unique(np.array(y_test)))

# set the total number of features/ dimensions
M = X.shape[1]

print("Number of Classes = ",C)
print("Number of Features = ",M)

# K-NEAREST NEIGHBOURS ALGORITHM
# set the value of k
k = 3
# predicting the class labels for test data
y_pred = knn_predict(X_train, y_train, X_test, k)
my_accuracy = accuracy(y_test, y_pred)

# Running Knn sklearn library function for comparison
sklearn_ypred = sklearn_knn_algo(X_train, y_train, X_test, k)
sklearn_accuracy = accuracy(y_test, sklearn_ypred)

# Comparing Results
print("Predicted Class Labels by My KNN = \n" + str(y_pred))
print()
print("Predicted Class Labels by Sklearn KNN = \n" + str(sklearn_ypred))
print()
print("My KNN Accuracy  = ",my_accuracy, "%")
print("Sklearn library KNN Accuracy = ", sklearn_accuracy, "%")
print("\n")

"""
for i in range(len(y_pred)):
    print("Test data = ", X_test[i])
    print("Class label = ", y_pred[i])
    print(i," Predicted Value = iris ",target_iris_names[int(y_pred[i])])
"""

# plotting the accuracy vs k values
```

```
k_values = [i for i in range(2,15)]
y_preds = []
accuracies = []
for i in range(2,15):
    cur_pred = knn_predict(X_train, y_train, X_test, i)
    y_preds.append(cur_pred)
    a = accuracy(y_test, cur_pred)
    accuracies.append(a)
print(accuracies)
plt.plot(k_values, accuracies)
plt.title("KNN Algorithm - K-value V/S Accuracy")
plt.xlabel("k-value")
plt.ylabel("Accuracy")
plt.show()
```

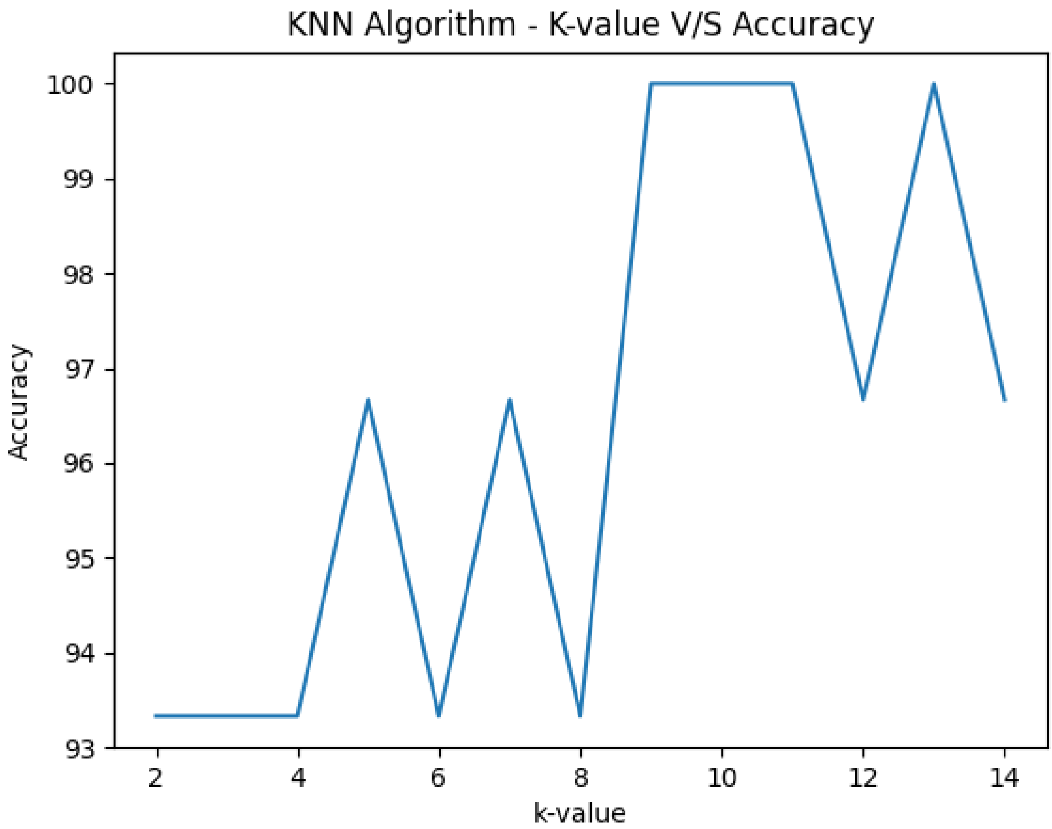
INPUT

K-value for running algorithm = 3 (by default)

K-values from 2 to 15 to plot the graph between k and accuracy of algorithm

OUTPUT

```
C:\Users\lenovo\PycharmProjects\PR-Lab\venv\Scripts\python.exe C:/Users/lenovo/PycharmProjects/PR-Lab/
K-Nearest Neighbours
(120, 4) (30, 4) (120,) (30,)
Number of Classes =  3
Number of Features =  4
Predicted Class Labels by My KNN =
[1 1 1 0 0 2 2 2 1 1 1 1 2 2 2 2 0 1 0 1 0 0 2 1 0 0 0 0 2 2]

Predicted Class Labels by Sklearn KNN =
[1 1 1 0 0 2 2 2 1 1 1 1 2 2 2 2 0 1 0 1 0 0 2 1 0 0 0 0 2 2]

My KNN Accuracy  =  93.333 %
Sklearn library KNN Accuracy =  93.333 %


[93.333, 93.333, 93.333, 96.667, 93.333, 96.667, 93.333, 100.0, 100.0, 100.0, 96.667, 100.0, 96.667]
```



DAY-7

1. Load the "fisheriris.mat" dataset and design a linear classifier using both perceptron batch and sequential algorithm.

CODE

Perceptron_sequential.py

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from matplotlib.colors import ListedColormap

print("Perceptron Implementation using Python by Yukti Khurana")
# loading dataset
df = pd.read_csv("Iris.csv",
usecols=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Spe
cies'])
df.columns = range(df.shape[1])
X = df.iloc[0:100, [0,2]].values

# Visualizing the dataset
plt.scatter(X[:50, 0], X[:50, 1], label = 'setosa',marker='x',
```

```
color='purple')
plt.scatter(X[50:100, 0], X[50:100, 1], label = 'versicolor',color='green')
plt.title("Iris Dataset Visualization by Yukti Khurana")
plt.xlabel("SepalLengthCm")
plt.ylabel("PetalLengthCm")
plt.show()

y = df.iloc[0:100, 4].values
# class label for setosa flower = -1
# class label for versicolor flower = 1
y = np.where(y == 'Iris-setosa', -1, 1)

# Initialising the model parameters
learn_rate = 0.001
epochs = 50
errors = []
# initializing an array for weights which will get updated in each
iteration
weights = np.zeros(1 + X.shape[1])

# function for summing the given matrix inputs and their corresponding
weights.
def net_input(x, weights):
  return np.dot(x, weights[1:]) + weights[0]

# prediction function
def predict(x, weights):
  return np.where(net_input(x, weights) >= 0.0, 1, -1)

# model fitting
# creating an numpy array of the size of train data
weights = np.zeros(1 + X.shape[1])
for i in range(epochs):
  err = 0
  for xi, target in zip(X,y):
    update = learn_rate * (target - predict(xi, weights))
    weights[1:] += update*xi
    weights[0] += update
    err += int(update != 0)
  errors.append(err)
print(errors)

# observing the drop in misclassification error after each epoch
plt.plot(range(1, len(errors) + 1), errors, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()

# Visualization of the perceptron model
resolution = 0.02
# setup marker generator and color map
markers = ('x', 'o')
colors = ('purple', 'green')
cmap = ListedColormap(colors[:len(np.unique(y))])

# plot the decision surface
x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
np.arange(x2_min, x2_max, resolution))
```

```
Z = predict(np.array([xx1.ravel(), xx2.ravel()]).T, weights)
Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())

class_names = ['setosa','versicolor']
# plot class samples
for idx, cl in enumerate(np.unique(y)):
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.7,
c=cmap(idx),marker=markers[idx], label=class_names[idx])

plt.title("Perceptron Model on Iris dataset")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Petal Length (cm)")
plt.legend()
plt.show()
```

Perceptron_batch.py

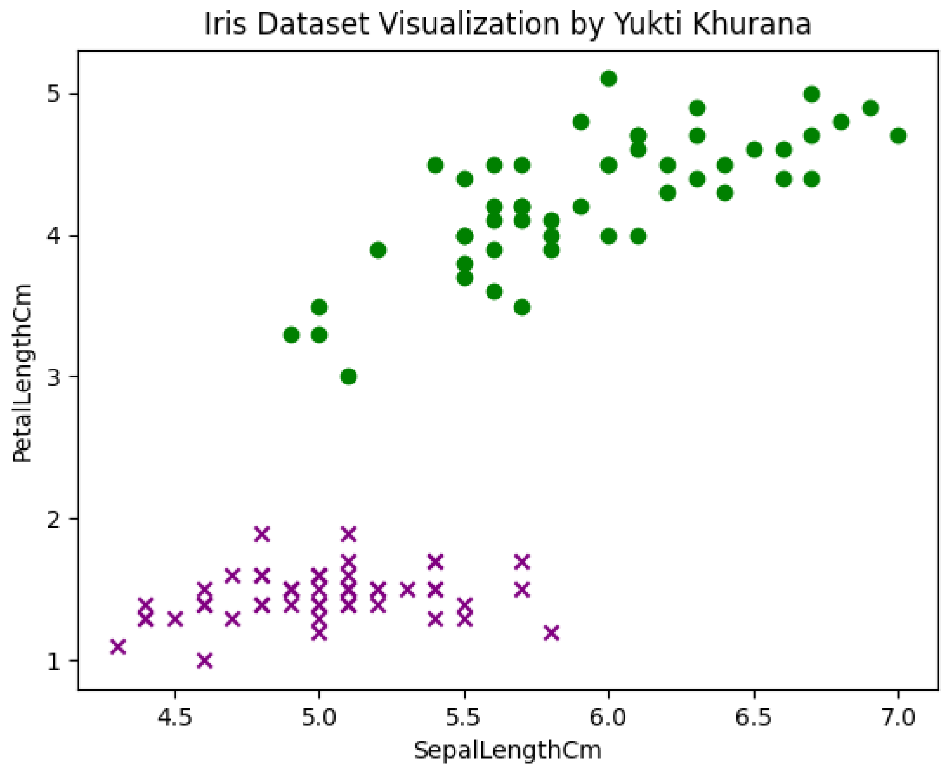
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from matplotlib.colors import ListedColormap

print("Perceptron Implementation using Python by Yukti Khurana")
# loading dataset
df = pd.read_csv("Iris.csv",
usecols=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Spe
cies'])
df.columns = range(df.shape[1])
X = df.iloc[0:100, [0,2]].values

# Visualizing the dataset
plt.scatter(X[:50, 0], X[:50, 1], label = 'setosa',marker='x',
color='purple')
plt.scatter(X[50:100, 0], X[50:100, 1], label = 'versicolor',color='green')
plt.title("Iris Dataset Visualization by Yukti Khurana")
plt.xlabel("SepalLengthCm")
plt.ylabel("PetalLengthCm")
plt.show()

y = df.iloc[0:100, 4].values
# class label for setosa flower = -1
# class label for versicolor flower = 1
y = np.where(y == 'Iris-setosa', -1, 1)

# Initialising the model parameters
learn_rate = 0.001
epochs = 40
errors = []
# initializing an array for weights which will get updated in each
iteration
weights = np.zeros(1 + X.shape[1])

# function for summing the given matrix inputs and their corresponding
weights.
def net_input(x, weights):
```

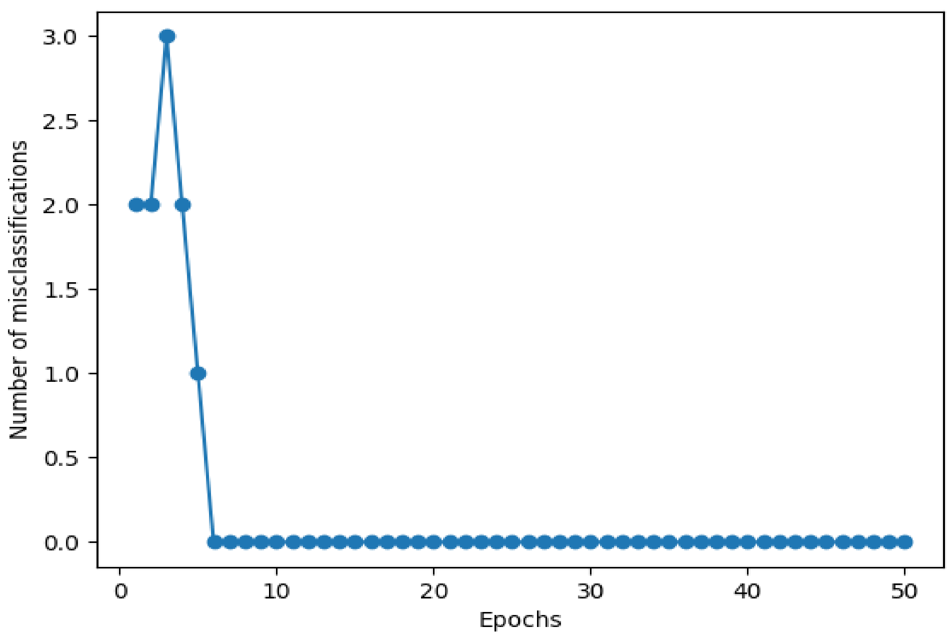
```
  return np.dot(x, weights[1:]) + weights[0]

# prediction function
def predict(x, weights):
  return np.where(net_input(x, weights) >= 0.0, 1, -1)

# model fitting
# creating an numpy array of the size of train data
weights = np.zeros(1 + X.shape[1])
errors = []
for i in range(epochs):
    error = y - net_input(X, weights)
    weights[1:] += learn_rate * X.T.dot(error)
    weights[0] += learn_rate * error.sum()
    cost = (error ** 2).sum() / 2.0
    errors.append(cost)
print(errors)

# observing the drop in misclassification error after each epoch
plt.plot(range(1, len(errors) + 1), errors, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()

# Visualization of the perceptron model
resolution = 0.02
# setup marker generator and color map
markers = ('x', 'o')
colors = ('purple', 'green')
cmap = ListedColormap(colors[:len(np.unique(y))])

# plot the decision surface
x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
np.arange(x2_min, x2_max, resolution))

Z = predict(np.array([xx1.ravel(), xx2.ravel()]).T, weights)
Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
plt.xlim(xx1.min(), xx1.max())
plt.ylim(xx2.min(), xx2.max())

class_names = ['setosa','versicolor']
# plot class samples
for idx, cl in enumerate(np.unique(y)):
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],alpha=0.7,
c=cmap(idx),marker=markers[idx], label=class_names[idx])

plt.title("Perceptron Model on Iris dataset")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Petal Length (cm)")
plt.legend()
plt.show()
```

INPUT

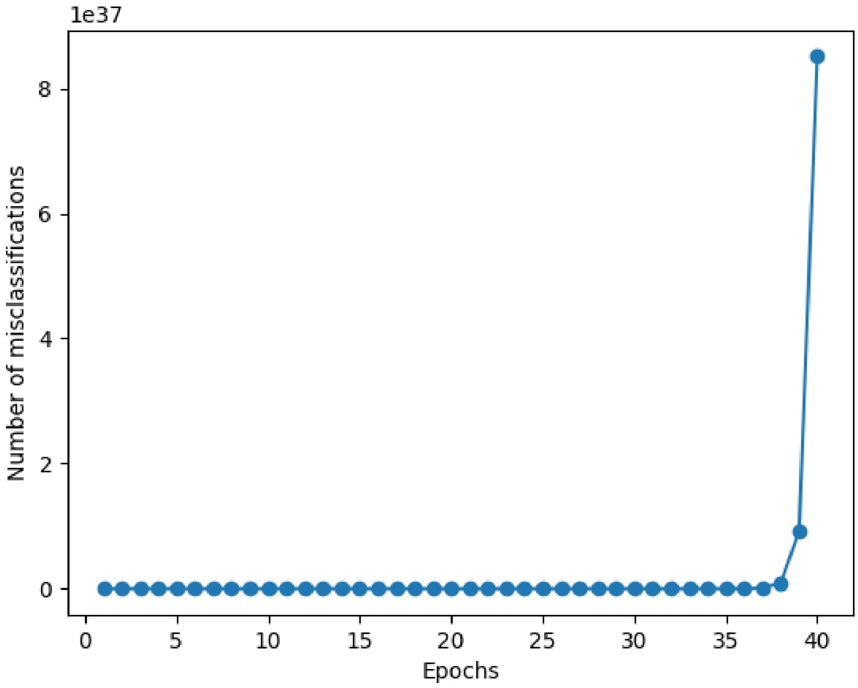
Iris dataset is used. Any other dataset can also be used.

OUTPUT

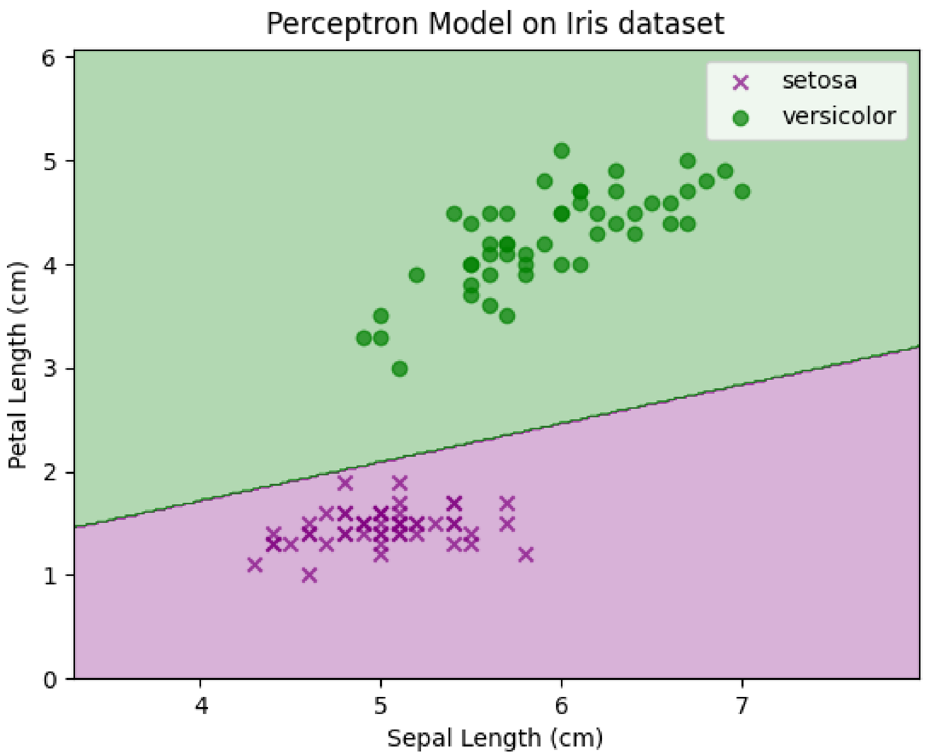




For Perceptron sequential.py



For Perceptron batch.py



Visualization of Classification of Iris dataset using Perceptron

2. Design a linear classifier using relaxation criteria on the same dataset.

CODE

```
import matplotlib.pyplot as plt
import numpy as np

def augment_vector(X):
    bias = np.ones((len(X), 1))
    return np.hstack((X, bias))

def normalise_train_set(X, Y):
    X = np.asarray(X)
    Y = np.asarray(Y)
    train_set = augment_vector(X)
    labels_ = np.unique(Y)
    idx = Y == labels_[1]  # Select one class as negative class
    train_set[idx] = -train_set[idx]  # Make the x coordinate of selected
class as negative
    dim = train_set.shape[1]  # Return Dimensionality of feature space //
train set
    return X, Y, train_set, dim

def linear_relaxation_algo(X, Y, learning_rate, margin):
    # Obtained the values as required (Augmented and negated)
    X, Y, train_set, dim = normalise_train_set(X, Y)
    weights = [0, 0, 1]
    k = -1
    i = 0
    count = 0
    while i != len(train_set):
        k = (k + 1) % len(train_set)
        if np.dot(train_set[k], weights) <= margin:
            i = 0
            temp1 = (margin - np.dot(train_set[k], weights))
            temp2 = np.dot(train_set[k], train_set[k])
            temp3 = float((float(temp1) / float(temp2)) * 2)
            temp = np.dot(temp3, train_set[k])
            weights = weights + (learning_rate * temp)
        else:
            i += 1
            count += 1

    plot_boundary(weights, train_set, X, Y)
    return weights


def plot_boundary(weights, train_set, X, Y):
    # plot data-points
    x_points = X[:, 0]
    y_points = X[:, 1]
    length = len(x_points)
    length = int(length)
    x_points_1 = x_points[0:int(length / 2)]
    x_points_2 = x_points[int(length / 2):length]
    y_points_1 = y_points[0:int(length / 2)]
    y_points_2 = y_points[int(length / 2):length]
```

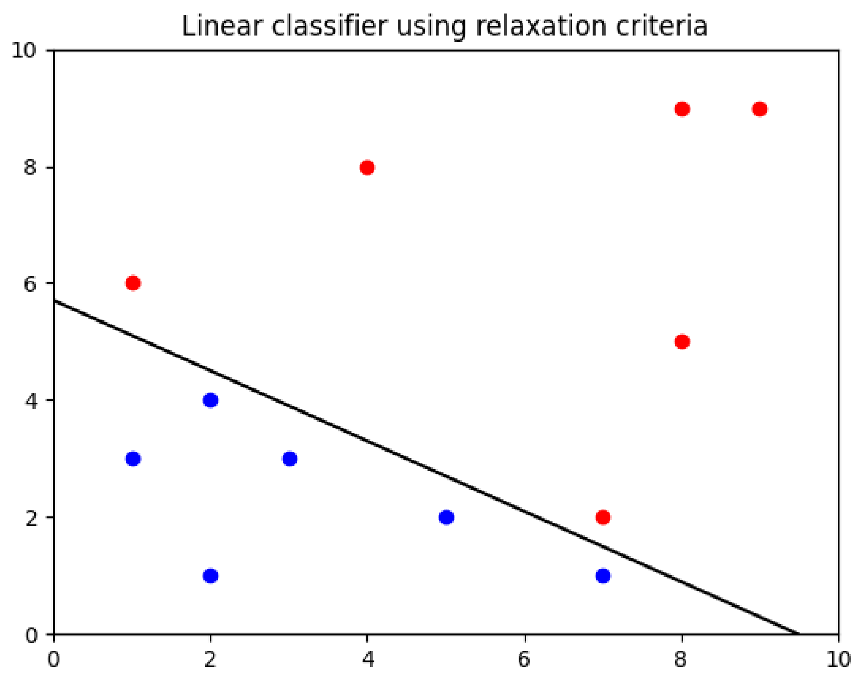
```
    plt.plot(x_points_1, y_points_1, 'ro');
    plt.axis([0, 10, 0, 10])
    plt.plot(x_points_2, y_points_2, 'bo');

    a, b, c = weights
    xchord_1 = 0
    xchord_2 = -(float(c)) / (float(a))
    ychord_2 = 0
    ychord_1 = -(float(c)) / (float(b))
    plt.title("Linear classifier using relaxation criteria")
    plt.plot([xchord_1, xchord_2], [ychord_1, ychord_2], 'black')
    plt.show()


def predict(test_set, weights):
    test_set = augment_vector(test_set)
    pred_list = []
    for i in range(len(test_set)):
        if np.dot(test_set[i], weights) < 0:
            pred_list.append(2)
        elif np.dot(test_set[i], weights) > 0:
            pred_list.append(1)
    return pred_list


def compute_accuracy(pred_labels_, Y_test):
    count = 0
    length = len(pred_labels_)
    for k in range(len(pred_labels_)):
        if pred_labels_[k] == Y_test[k]:
            count = count + 1
    accuracy = (float(count) / float(length)) * 100
    return round(accuracy,3)

def main():
    print("\nLinear classifier using relaxation criteria \n")
    X = [(1, 6), (7, 2), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3),
(2, 4), (7, 1), (1, 3), (5, 2)]
    Y = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2]
    testset = [(0, 1), (8, 1), (2, 6), (2, 4.5), (6, 1.5), (4, 3)]
    test = [(6, 4), (6, 6), (9, 4), (0, 0), (0, -2), (1, 1)]
    Y_test = [1, 1, 1, 2, 2, 2]
    Xnew = [(1, 6), (7, 6), (8, 9), (9, 9), (4, 8), (8, 5), (2, 1), (3, 3),
(2, 4), (7, 1), (1, 3), (5, 2)]
    X_no_sep = [(2, 1), (7, 2), (2, 4), (9, 9), (4, 8), (5, 2), (1, 6), (3,
3), (8, 9), (7, 1), (1, 3), (8, 5)]

    learning_rate = 1
    margin = 1.0
    weights = linear_relaxation_algo(X, Y, learning_rate, margin)
    pred_labels = predict(test, weights)
    print("Predicted Classes: ")
    print(pred_labels)
    accuracy = compute_accuracy(pred_labels, Y_test)
    print('Accuracy for the test data set is  {}%'.format(accuracy))


main()
```

INPUT

Given Dataset is used. Any other may also be used.

OUTPUT

```
C:\Users\lenovo\PycharmProjects\PR-Lab\venv\Scripts\python.exe C:/Users/lenovo/PycharmProjects/PR-Lab/

Linear classifier using relaxation criteria

Predicted Classes:
[1, 1, 1, 2, 2, 2]
Accuracy for the test data set is  100.0%

Process finished with exit code 0
```



DAY-8

1. Load the "fisheriris.mat" dataset and perform classification using support vector machine (SVM). (Take the first 100 samples out of 150 samples)

CODE

SVM.py

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import numpy as np

print("SUPPORT VECTOR MACHINE CLASSIFICATION BY YUKTI KHURANA")

df = pd.read_csv("Iris.csv")
df = df.drop(['Id'], axis=1)
# taking the first 100 samples out of 150 samples
# converting this into a binary classification problem by considering only
two classes
# two classes - setosa and versicolor
df = df[0:100]
# visualising the two classes
x = df['SepalLengthCm']
y = df['PetalLengthCm']
setosa_x = x[:50]
setosa_y = y[:50]
versicolor_x = x[50:]
versicolor_y = y[50:]
plt.figure(figsize=(8, 6))
plt.scatter(setosa_x, setosa_y, label = 'setosa',marker='+',
color='purple')
plt.scatter(versicolor_x, versicolor_y,label='versicolor', marker='_',
color='blue')
plt.xlabel("SepalLengthCm")
plt.ylabel("PetalLengthCm")
plt.title("Visualizing the two classes of Iris dataset")
plt.legend()
plt.show()

# Drop unnecessary features
df = df.drop(['SepalWidthCm', 'PetalWidthCm'], axis=1)
# constructing the class labels
y = []
target = df['Species']
for val in target:
    if (val == 'Iris-setosa'):
        y.append(-1)
    else:
        y.append(1)

df = df.drop(['Species'], axis=1)
X = df.values.tolist()
split_size = 0.80
# Splitting the training and testing data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=split_size, random_state=41, stratify=y)
print(len(X_train), len(y_train), len(X_test), len(y_test))

X_train, X_test, y_train, y_test = np.array(X_train), np.array(X_test),
np.array(y_train), np.array(y_test)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

y_train = y_train.reshape(X_train.shape[0], 1)
y_test = y_test.reshape(X_test.shape[0], 1)
print(y_train.shape, y_test.shape)

# extracting the training features
# feature 1
train_f1 = X_train[:, 0]
# feature 2
train_f2 = X_train[:, 1]


# extracting the test data features
test_f1 = X_test[:, 0]
test_f2 = X_test[:, 1]

test_f1 = test_f1.reshape(X_test.shape[0], 1)
test_f2 = test_f2.reshape(X_test.shape[0], 1)

train_f1 = train_f1.reshape(X_train.shape[0], 1)
train_f2 = train_f2.reshape(X_train.shape[0], 1)
print(train_f1.shape, train_f2.shape)

w1 = np.zeros((X_train.shape[0], 1))
w2 = np.zeros((X_train.shape[0], 1))

# no of epochs
epochs = 1
# learning rate
alpha = 0.0001

# the regularization parameter λ is set to 1/epochs.
# Therefore, the regularizing value reduces the number of epochs increases.
print("\nModel Training begins.....\n")
while (epochs < 10000):
    y = w1 * train_f1 + w2 * train_f2
    prod = y * y_train
    count = 0
    print("Epoch - ", epochs)
    for val in prod:
        if (val >= 1):
            # no misclassification
            # cost will be zero
            # w = w - alpha * (2*lambda*w)
            cost = 0
            w1 = w1 - alpha * (2 * 1 / epochs * w1)
            w2 = w2 - alpha * (2 * 1 / epochs * w2)
        # When there is a misclassification, i.e our model make a mistake
on the prediction of the class of our data point, we include the loss along
with the regularization parameter to perform gradient update.
        else:
            # misclassification occurs
            # so gradient updation will involve cost
            # w  = w+ alpha * (yi * xi - (2*lambda*w))
```

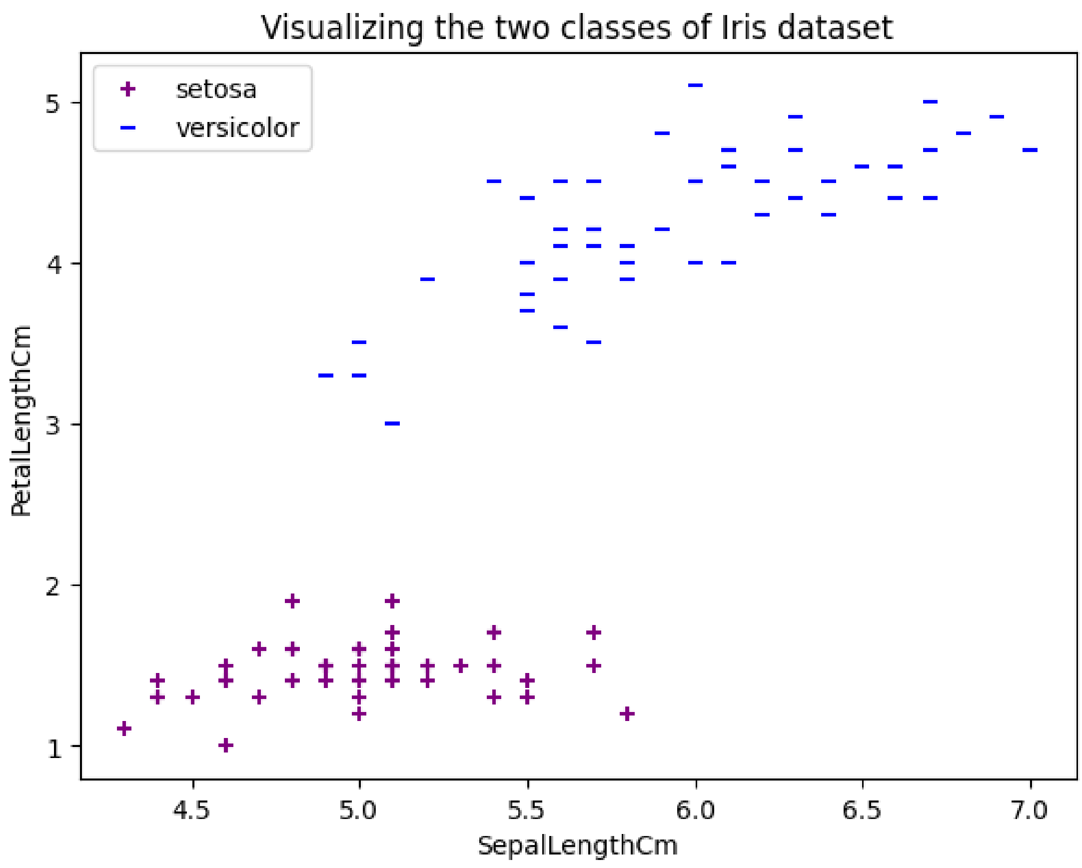
```
            cost = 1 - val
            w1 = w1 + alpha * (train_f1[count] * y_train[count] - 2 * 1 /
epochs * w1)
            w2 = w2 + alpha * (train_f2[count] * y_train[count] - 2 * 1 /
epochs * w2)
        count += 1
    epochs += 1
print("\nModel Training ends....\n")

def accuracy(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0


# Clipping the weights
index = list(range(X_test.shape[0], X_train.shape[0]))
w1 = np.delete(w1, index)
w2 = np.delete(w2, index)

# reshaping the weights
w1 = w1.reshape(X_test.shape[0], 1)
w2 = w2.reshape(X_test.shape[0], 1)

# Prediction
y_pred = w1 * test_f1 + w2 * test_f2
predictions = []
for val in y_pred:
    if (val > 1):
        predictions.append(1)
    else:
        predictions.append(-1)

pred_flower = []
for i in predictions:
    if i == -1:
        pred_flower.append("iris-setosa")
    else:
        pred_flower.append("iris-versicolor")

print("Classification according to my SVM - ")
print(predictions)
print(pred_flower)
print("Accuracy of my SVM classification = ", accuracy(y_test,
predictions), "%")
print("\n")


print("SVM Classification using sklearn library for comparison\n")
svc_clf = SVC(kernel='linear')
svc_clf.fit(X_train,np.ravel(y_train,order='C'))
sklearn_ypred = svc_clf.predict(X_test)
print("Classification according to Sklearn built-in SVM -")
print(sklearn_ypred.tolist())
print("The accuracy of built-in sklearn SVM classification =
",accuracy(y_test,sklearn_ypred), "%")
```

INPUT

- Iris Dataset has been used for SVM implementation. Any other dataset may also be used
- epochs = 10000
- learning rate = 0.0001

OUTPUT

s



```
C:\Users\lenovo\PycharmProjects\PR-Lab\venv\Scripts\python.exe C:/Users/lenovo/PycharmProjects/PR-Lab/
SUPPORT VECTOR MACHINE CLASSIFICATION BY YUKTI KHURANA
80 80 20 20
(80, 2) (20, 2) (80,) (20,)
(80, 1) (20, 1)
(80, 1) (80, 1)

Model Training begins.....

Epoch -  1
Epoch -  2
Epoch -  3
Epoch -  4
Epoch -  5
Epoch -  6
Epoch -  7
Epoch -  8
Epoch -  9
Epoch -  10
Epoch -  11
Epoch -  12
```

```
Epoch -  9994
Epoch -  9995
Epoch -  9996
Epoch -  9997
Epoch -  9998
Epoch -  9999

Model Training ends....

Classification according to my SVM -
[1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, -1, -1]
['iris-versicolor', 'iris-setosa', 'iris-versicolor', 'iris-versicolor', 'iris-versicolor', 'iris-versicolor', 'iris-setosa', 'iris-setosa', 'iris
Accuracy of my SVM classification =  100.0 %


SVM Classification using sklearn library for comparison

Classification according to Sklearn built-in SVM -
[1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, -1, -1]
The accuracy of built-in sklearn SVM classification =  100.0 %

Process finished with exit code 0
```

Classification according to my SVM -

[1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, -1, -1]

['iris-versicolor', 'iris-setosa', 'iris-versicolor', 'iris-versicolor', 'iris-versicolor', 'iris-versicolor', 'iris-setosa', 'iris-setosa', 'iris-versicolor', 'iris-setosa', 'iris-versicolor', 'iris-versicolor', 'iris-setosa', 'iris-setosa', 'iris-versicolor', 'iris-setosa', 'iris-versicolor', 'iris-setosa', 'iris-setosa', 'iris-setosa']

Accuracy of my SVM classification = 100.0 %

SVM Classification using sklearn library for comparison

Classification according to Sklearn built-in SVM -

[1, -1, 1, 1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, -1, -1]

The accuracy of built-in sklearn SVM classification = 100.0 %