MAHARSHI DAYANAND UNIVERSITY

DELHI GLOBAL INSTITUTE OF TECHNOLOGY

MACHINE LEARNING LAB

PRACTICAL FILE

SUBJECT CODE: LC-CSE-421G

SUBMITTED BY : KAJAL CHAUHAN

REGISTRATION NO. : 191380207

INDEX

S.NO	TITLE	REMARKS
1.	Intro to data set . & How to run a data set on anaconda(spider). & How to do different operations on data set ex- conversion to matrix, matrix addition, subtraction, multiplications	
2.	Normalize the given matrix and find its covariance.	
3.	Implement the PCA on two datasets (Iris, a 500 row and 5 col)	
4.	How to implement KNN on a dataset and interpret its results	
5.	How to implement decision tree on a dataset	
6.	How to implement linear regression on a dataset	
7.	How to implement Support vector machine on a dataset	
8.	How to implement K-means algo on a dataset	
9.	Implement the one ensemble method on a dataset(bosting ,bagging, random forest)	
10.	Find the performance of all the above practical and compare different results(confusion, precision recall F1-score, Roc, Median absolute deviation)	

PRACTICAL NO – 1

Aim: Intro to data set and how to run a data set on anaconda(spider). How to do different operations on data set ex- conversion to matrix, matrix addition, subtraction, multiplications.

(1.1) Intro to data set :

A dataset is a collection of structured data that is used for machine learning or statistical analysis. Datasets can come in various formats, such as CSV, Excel, SQL databases, and more.

(1.2) How to run a data set on anaconda(spider):

1. Open Anaconda Navigator: Open the Anaconda Navigator by clicking on the Anaconda icon in your applications folder or start menu.

2. Create a new environment (optional): If you want to create a new environment for your dataset, click on the "Environments" tab and then click the "Create" button. Give your environment a name, choose your Python version, and select the packages you want to include.

3. Launch Spider: Click on the "Home" tab in Anaconda Navigator, then click on the "Launch" button under the "Spyder" icon to open the Spyder IDE.

4. Load the dataset: In the Spyder IDE, click on "File" in the top menu bar and select "Open". Navigate to the folder where your dataset is saved and select it.

5. Analyze the dataset: Once your dataset is loaded, you can start analyzing it using Python libraries such as NumPy, Pandas, Matplotlib, and more. For example, you can use Pandas to read the CSV file, clean and preprocess the data, and perform exploratory data analysis. You can also use machine learning libraries like Scikit-learn to build models and make predictions based on your dataset.

(1.3) How to do different operations on data set ex- conversion to matrix, matrix addition, subtraction, multiplications.

IRIS.cvs file

aa	bb	cc	dd
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5	3.6	1.4	0.2
5.4	3.9	1.7	0.4
4.6	3.4	1.4	0.3
5	3.4	1.5	0.2
4.4	2.9	1.4	0.2
5.4	3.7	1.5	0.2
4.8	3.4	1.6	0.2
4.8	3	1.4	0.1
4.3	3	1.1	0.1
5.8	4	1.2	0.2
5.7	4.4	1.5	0.4
5.4	3.9	1.3	0.4
5.1	3.5	1.4	0.3
5.7	3.8	1.7	0.3
5.1	3.8	1.5	0.3

```
import numpy as np
df = pd.read_csv(r'C:\Users\dell\OneDrive\Desktop\iris.csv')
print(df)
```

OUTPUT :

```
In [21]: runfile('C:/Users/de
kajal9.py', wdir='C:/Users/de
      aa   bb   cc   dd
0    5.1  3.5  1.4  0.2
1    4.9  3.0  1.4  0.2
2    4.7  3.2  1.3  0.2
3    4.6  3.1  1.5  0.2
4    5.0  3.6  1.4  0.2
..   ...  ...  ...  ...
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]
```

```
import pandas as pd
import numpy as np

# Create a sample data set
data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}
df = pd.DataFrame(data)

# Convert the data set to a matrix
matrix = df.to_numpy()

# Create another matrix
matrix2 = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])

# Matrix addition
matrix_addition = matrix + matrix2

# Matrix subtraction
matrix_subtraction = matrix - matrix2

# Matrix multiplication
matrix_multiplication = np.matmul(matrix, matrix2)

# Print the results
print("Original Data Set:\n", df)
print("Data Set as a Matrix:\n", matrix)
print("Matrix 2:\n", matrix2)
print("Matrix Addition:\n", matrix_addition)
print("Matrix Subtraction:\n", matrix_subtraction)
print("Matrix Multiplication:\n", matrix_multiplication)
```

OUTPUT :

```
In [36]: runfile('C:/Users/del
kajal9.py', wdir='C:/Users/del
Original Data Set:
    A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
Data Set as a Matrix:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
Matrix 2:
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
Matrix Addition:
 [[ 2  4  7]
 [ 2  6  8]
 [ 3  6 10]]
Matrix Subtraction:
 [[0 4 7]
 [2 4 8]
 [3 6 8]]
Matrix Multiplication:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

PRACTICAL NO – 2

Aim: Normalize the given matrix and find its covariance

```
import pandas as pd
import numpy as np

# Create a sample DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}
df = pd.DataFrame(data)

# Normalize the DataFrame
df_norm = (df - df.mean()) / df.std()

# Find the covariance of the normalized DataFrame
covariance_matrix = df_norm.cov()

# Print the results
print("Original DataFrame:\n", df)
print("Normalized DataFrame:\n", df_norm)
print("Covariance Matrix:\n", covariance_matrix)
```

OUTPUT :

```
In [37]: runfile('C:/Use
kajal10.py', wdir='C:/Us
Original DataFrame:
    A  B  C
0  1  4  7
1  2  5  8
2  3  6  9
Normalized DataFrame:
      A    B    C
0 -1.0 -1.0 -1.0
1  0.0  0.0  0.0
2  1.0  1.0  1.0
Covariance Matrix:
      A    B    C
A  1.0  1.0  1.0
B  1.0  1.0  1.0
C  1.0  1.0  1.0
```

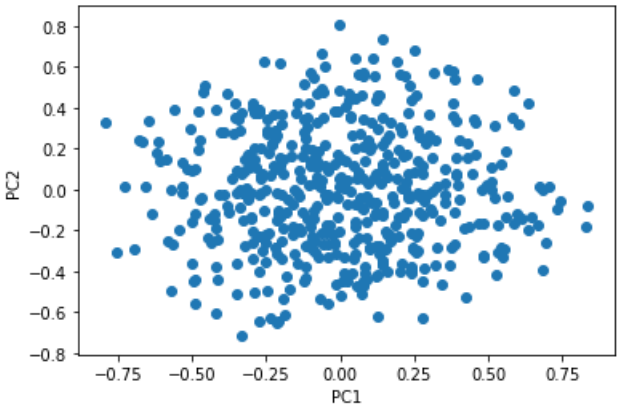
PRACTICAL NO – 3

Aim: Implement the PCA on two datasets(Iris, a 500 row and 5 col)

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Generate a random dataset with 500 rows and 5 columns
np.random.seed(42)
X = np.random.rand(500, 5)

# Perform PCA with two components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot the results
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

OUTPUT :

```
In [2]: runfile('C:/Users/dell/.spyder-py3/kajal.py'
```



PRACTICAL NO – 4

Aim: How to implement KNN on a dataset and interpret its results

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2,
random_state=42)

# Train the KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn.predict(X_test)

# Compute the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT :

```
In [4]: runfile('C:/Use
kajal2.py', wdir='C:/Us
Accuracy: 1.0
```

PRACTICAL NO – 5

Aim: How to implement decision tree on a dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2,
random_state=42)

# Train the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Compute the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT :

```
In [5]: runfile('C:
kajal3.py', wdir='C
Accuracy: 1.0
```

PRACTICAL NO – 6

Aim: How to implement linear regression on a dataset

```
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import numpy as np

# Load the Boston housing dataset
boston = load_boston()

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target,
test_size=0.2, random_state=42)

# Train the linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lr.predict(X_test)

# Compute the mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean squared error:", mse)

# Compute the coefficient of determination (R^2 score)
r2 = lr.score(X_test, y_test)
print("R^2 score:", r2)
```

OUTPUT :

```
In [6]: runfile('C:/Users/dell/.spyder-
kajal4.py', wdir='C:/Users/dell/.spyder
Mean squared error: 24.291119474973428
R^2 score: 0.6687594935356332
```

PRACTICAL NO – 7

Aim: How to implement Support vector machine on a dataset

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
cancer = load_breast_cancer()

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
test_size=0.2, random_state=42)

# Train the SVM classifier
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm.predict(X_test)

# Compute the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

OUTPUT :

```
In [11]: runfile('C:/Users/del
kajal5.py', wdir='C:/Users/del
Accuracy: 0.956140350877193
```

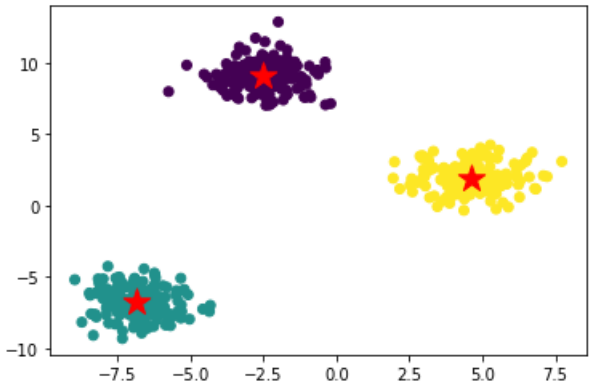
PRACTICAL NO – 8

Aim: How to implement K-means algo on a dataset

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Generate a random dataset with 500 samples and 2 features
X, y = make_blobs(n_samples=500, centers=3, random_state=42)

# Create a KMeans object with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the KMeans model to the data
kmeans.fit(X)

# Get the cluster labels and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Plot the data points and the centroids
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300, c='r')
plt.show()
```

OUTPUT :

```
In [12]: runfile('C:/Users/dell/.spyder-py3/
kajal6.py', wdir='C:/Users/dell/.spyder-py3')
```



PRACTICAL NO – 9

Aim: Implement the one ensemble method on a dataset(bosting ,bagging, random forest)

```
from sklearn.datasets import load_iris
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a decision tree classifier
clf = DecisionTreeClassifier()

# Create a bagging classifier with 10 estimators
bagging_clf = BaggingClassifier(base_estimator=clf, n_estimators=10,
random_state=42)

# Fit the bagging classifier to the training data
bagging_clf.fit(X_train, y_train)

# Predict the class labels for the test set
y_pred = bagging_clf.predict(X_test)

# Calculate the accuracy score of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

OUTPUT :

```
In [14]: runfile('C:
kajal7.py', wdir='C:
Accuracy: 1.0
```

PRACTICAL NO – 10

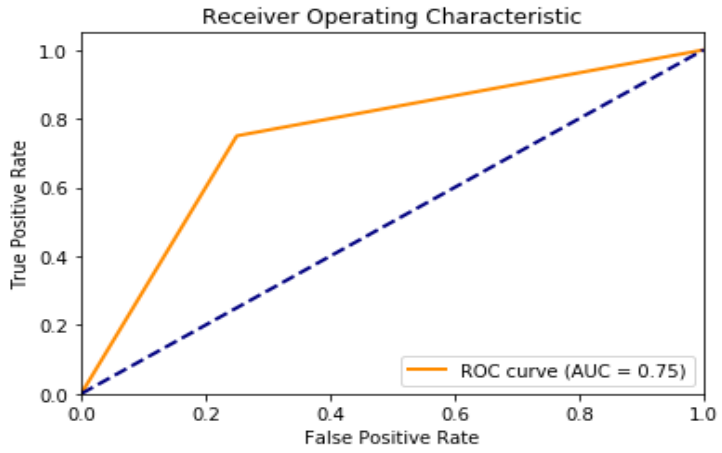
Aim: Find the performance of all the above practical and compare different results(confusion, precision recall F1-score, Roc, Median absolute deviation)

```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc, median_absolute_error
import matplotlib.pyplot as plt

# y_true is the true labels of the test set, and y_pred is the predicted labels of the test set
# you can obtain y_pred by calling the predict() method of your trained model
y_true = [0, 1, 0, 1, 0, 1, 1, 0]
y_pred = [1, 1, 0, 1, 0, 0, 1, 0]

# calculate the confusion matrix
conf_mat = confusion_matrix(y_true, y_pred)
print('Confusion Matrix:\n', conf_mat)

# calculate the precision, recall, F1-score, and support for each class
precision, recall, f1_score, support = precision_recall_fscore_support(y_true, y_pred, average=None)
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1_score)
print('Support:', support)

# calculate the ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_true, y_pred)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
# calculate the median absolute deviation
mad = median_absolute_error(y_true, y_pred)
print('Median Absolute Deviation:', mad)
```

OUTPUT :

```
In [16]: runfile('C:/Users/dell/.spyder-py3/kajal8.py
Confusion Matrix:
 [[3 1]
 [1 3]]
Precision: [0.75 0.75]
Recall: [0.75 0.75]
F1-score: [0.75 0.75]
Support: [4 4]
```




```
Median Absolute Deviation: 0.0
```