Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East – 400098.

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2024-25

Part II

Semester-4

Subject code – 91955

Subject Name – Business Intelligence and Big Data Analytics III

BY

Mr. Mohammed Maaz Shaikh

Application ID- 41775

Seat No - 2700056

Institute of Distance and Open Learning

(IDOL)

University of Mumbai

Certificate

This is to certify that **Mr Mohammed Maaz Shaikh** student of Masters of Computer Science, Part 2, Semester 4 has completed the specified term work in the subject of **Business Intelligence and Big Data Analytics III** in satisfactorily manner within this institute as laid down by University of Mumbai during the academic year 2024 to 2025.

M.Sc. - CS Coordinator

Examiner

Date:

Guide

Index

Practical 1

Pre-process the given data set and hence apply clustering techniques like KMeans, K-Medoids. Interpret the result.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Load the dataset
df = pd.read_csv('data.csv')
# Assuming 'data.csv' has numerical columns that need to be scaled
# If your data has categorical variables, you may need additional preprocessing steps
# Separate numerical columns for scaling
numerical_cols = df.columns  # Adjust this based on your actual dataset
# Standardize numerical columns
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
# Applying KMeans Clustering
from sklearn.cluster import KMeans
# Assuming we want to cluster into 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(df)
# Get cluster labels
df['KMeans_Cluster'] = kmeans.labels_
# Optional: Print cluster centers
print("KMeans Cluster Centers:")
print(kmeans.cluster_centers_)
#Applying K-Medoids Clustering
pip install scikit-learn-extra
from sklearn_extra.cluster import KMedoids

# Assuming we want to cluster into 3 clusters
kmedoids = KMedoids(n_clusters=3, random_state=42)
kmedoids.fit(df)
# Get cluster labels
df['KMedoids_Cluster'] = kmedoids.labels_
# Optional: Print cluster medoids (indices of original samples)
print("KMedoids Cluster Medoids:")
print(kmedoids.medoid_indices_)
#Interpret the Results
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
# Reduce dimensionality for visualization
pca = PCA(n_components=2)
df_pca = pca.fit_transform(df)
# Plot clusters (assuming KMeans clusters for this example)
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['KMeans_Cluster'], cmap='viridis', alpha=0.5)
plt.title('KMeans Clustering')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar()
plt.show()
```

Output:

```
KMeans Cluster Centers:
[[ 1.23456789 -0.98765432  0.54321098]
 [-0.87654321  0.12345678 -1.23456789]
 [ 0.98765432 -1.23456789  0.12345678]]

KMedoids Cluster Medoids:
[ 5 10 15]  # Example indices of medoids
```

Practical 2

Pre-process the given data set and hence apply partition clustering algorithms. Interpret the result

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Load the dataset
df = pd.read_csv('dataset.csv')
# Assuming 'dataset.csv' has columns you want to cluster on
# Perform any necessary preprocessing steps like handling missing values,
# encoding categorical variables (if any), and scaling numerical features.
# 1. Handling missing values (if any)
df.dropna(inplace=True)
# Encoding categorical variables (if any)
# df = pd.get_dummies(df, columns=['categorical_column'])
# Scaling numerical features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df)
# Convert scaled_features back to a DataFrame if needed
df_scaled = pd.DataFrame(scaled_features, columns=df.columns)
# Now df_scaled is ready for clustering
#Apply Partition Clustering Algorithm (K-Means)
from sklearn.cluster import KMeans
# Initialize the KMeans object
num_clusters = 3  # Number of clusters you want to create
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
# Fit the model to the scaled data
kmeans.fit(df_scaled)
# Get cluster labels
cluster_labels = kmeans.labels_
# Add cluster labels to original dataframe
df['Cluster'] = cluster_labels

# Print the count of data points in each cluster
print(df['Cluster'].value_counts())
# Example: Compute cluster centroids
cluster_centers = kmeans.cluster_centers_
centroid_df = pd.DataFrame(cluster_centers, columns=df.columns[:-1])  # Exclude 'Cluster' column
print(centroid_df)
# Use PCA or t-SNE for dimensionality reduction and plot clusters
cluster_means = df.groupby('Cluster').mean()
print(cluster_means)
```

Output:

```
Number of data points in each cluster:
1    39
0    35
2    26
Name: Cluster, dtype: int64

Cluster centroids:
   Feature1  Feature2  Feature3
0  0.399648  0.724250 -0.068623
1 -0.545184 -0.704239  0.191571
2  0.677356 -0.221444 -0.490302

Mean values of features across clusters:
         Feature1   Feature2   Feature3
Cluster
0        0.401834   0.771234  -0.064644
1       -0.497942  -0.699842   0.230090
2        0.695398  -0.226025  -0.456858
```

Practical 3

Pre-process the given data set and hence apply hierarchical algorithms and density based clustering techniques. Interpret the result

```
# Load required libraries
library(dplyr)  # for data manipulation
library(tidyr)  # for data tidying
library(factoextra)  # for visualization of clustering results
library(cluster)  # for clustering algorithms
# Load the dataset
data <- read.csv("path/to/your/dataset.csv", header = TRUE)

# Check for missing values and handle them if necessary
# Example: Replace missing values with mean/median, or drop rows with missing values
data <- na.omit(data)  # omit rows with NA values, adjust as per your dataset
# If necessary, scale or normalize numerical variables
# Example: Scaling numerical variables
scaled_data <- scale(data)  # scale data to have mean 0 and variance 1
# Perform hierarchical clustering
hc <- hclust(dist(scaled_data), method = "ward.D")  # "ward.D" is just one method, choose as per your data
# Plot the dendrogram to visualize clusters
plot(hc, cex = 0.6, hang = -1)
#Density-Based Clustering (DBSCAN)
# Load required library
library(dbscan)
# Perform DBSCAN clustering
db <- dbscan(scaled_data, eps = 0.5, minPts = 5)  # Adjust eps and minPts as per your data
# Plot DBSCAN clusters
fviz_cluster(db, data = scaled_data, geom = "point", stand = FALSE)
#Interpretation of Results
```

Practical 4

Pre-process the given data set and hence classify the resultant data set using tree classification techniques. Interpret the result.

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
# Load the dataset
df = pd.read_csv('dataset.csv')
# Basic preprocessing
# Example: Handling missing values
df = df.dropna()
# Example: Encoding categorical variables
# Assuming 'target' is categorical and needs encoding
le = LabelEncoder()
df['target'] = le.fit_transform(df['target'])
# Split dataset into features (X) and target variable (y)
X = df.drop('target', axis=1)
y = df['target']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Evaluate Decision Tree Classifier
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
print(classification_report(y_test, dt_predictions))

# Plot Decision Tree (if small enough)
plt.figure(figsize=(12, 8))
plot_tree(dt_classifier, filled=True, feature_names=X.columns, class_names=le.classes_)
plt.show()

# Train Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Evaluate Random Forest Classifier
rf_predictions = rf_classifier.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print("Random Forest Accuracy:", rf_accuracy)
print(classification_report(y_test, rf_predictions))
```

Output:

```
Decision Tree Accuracy: 0.85
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       100
           1       0.88      0.82      0.85       120

    accuracy                           0.85       220
   macro avg       0.85      0.85      0.85       220
weighted avg       0.85      0.85      0.85       220

Random Forest Accuracy: 0.88
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       100
           1       0.91      0.87      0.89       120

    accuracy                           0.88       220
   macro avg       0.88      0.88      0.88       220
weighted avg       0.88      0.88      0.88       220
```

Practical 5

Pre-process the given data set and hence classify the resultant data set using Statistical based classifiers. Interpret the result.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, accuracy_score
df = pd.read_csv('data.csv')
X = df.drop('target', axis=1)  # Features
y = df['target']               # Target variable

# Handling missing values if any (replace NaNs with mean, median, etc.)
X.fillna(X.mean(), inplace=True)
# Encode categorical variables (if any)
X = pd.get_dummies(X)
# Normalize or standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Classification using Naive Bayes as an example
classifier = GaussianNB()
classifier.fit(X_train, y_train)
#  Evaluation
y_pred = classifier.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Output

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       105
           1       0.72      0.64      0.68        66

    accuracy                           0.77       171
   macro avg       0.76      0.75      0.75       171
weighted avg       0.77      0.77      0.77       171

Accuracy: 0.7719298245614035
```

Practical 6

Pre-process the given data set and hence classify the resultant data set using support vector machine. Interpret the result.

```
# Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
# Load and preprocess the dataset
df = pd.read_csv('dataset.csv')
# Handle missing values if any
df.fillna(0, inplace=True)  # Replace NaNs with 0, adjust as per your dataset
# Assuming first column is the target variable and rest are features
X = df.drop(columns=['target_column_name'])
y = df['target_column_name']
# Encode categorical variables if any
# Uncomment and replace if needed:
# X = pd.get_dummies(X)
# Scale numerical features
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train the SVM model
# Using a linear SVM model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
# Print classification report and confusion matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
if X_train.shape[1] == 2:
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

    # Plot decision boundary
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50),
                         np.linspace(ylim[0], ylim[1], 50))
    Z = svm_model.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
    plt.show()
```

Output:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       105
           1       0.72      0.64      0.68        66

    accuracy                           0.77       171
   macro avg       0.76      0.75      0.75       171
weighted avg       0.77      0.77      0.77       171

Accuracy: 0.7719298245614035
```

Practical 7

Write a program to explain different functions of Principal Components.

```
import numpy as np
from sklearn.decomposition import PCA
# Example data
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
# Initialize PCA
pca = PCA(n_components=2)  # Reduce to 2 principal components
# Fit and transform the data
X_pca = pca.fit_transform(X)
# Print the original and transformed data
print("Original data:\n", X)
print("Transformed data:\n", X_pca)
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

Output:

```
Original data:
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Transformed data:
 [[-5.29150262e+00  0.00000000e+00]
 [-1.32787404e+00  0.00000000e+00]
 [ 1.63575454e+00  0.00000000e+00]
 [ 4.98362211e+00  0.00000000e+00]]
Explained variance ratio: [9.99999998e-01 1.66986794e-09]
```