Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.6

Clustering & Classification

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 6

Clustering & Classification

(Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to K-Means Clustering

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into clusters or groups. It aims to group similar data points into clusters while minimizing the within-cluster variance.

Here's how K-means clustering works:

- **Initialization:** Start by selecting K initial cluster centroids. These centroids can be randomly chosen data points or based on some other criteria.
- **Assignment:** Assign each data point to the nearest centroid, creating K clusters. The "nearest" centroid is typically determined by using a distance metric, such as Euclidean distance.
- **Update:** Recalculate the centroids of the K clusters. This is done by taking the mean of all data points in each cluster.
- **Repeat:** Steps 2 and 3 are repeated iteratively until convergence criteria are met. Common convergence criteria include a maximum number of iterations, a minimal change in cluster assignments, or when the centroids no longer change significantly.
- **Output:** The algorithm provides a final set of K cluster centroids, and each data point is associated with the cluster to which it is closest.

The choice of K (the number of clusters) is a critical parameter in K-means clustering and often requires some domain knowledge or techniques like the Elbow Method or Silhouette Analysis to find the optimal value.

K-means clustering has several applications, including image segmentation, customer segmentation in marketing, anomaly detection, and more. However, it has limitations, such as sensitivity to the initial cluster centroids and the assumption that clusters are spherical and equally sized

K-means Clustering with Python

Here's a Python example of how to perform K-means clustering using the popular scikit-learn library:

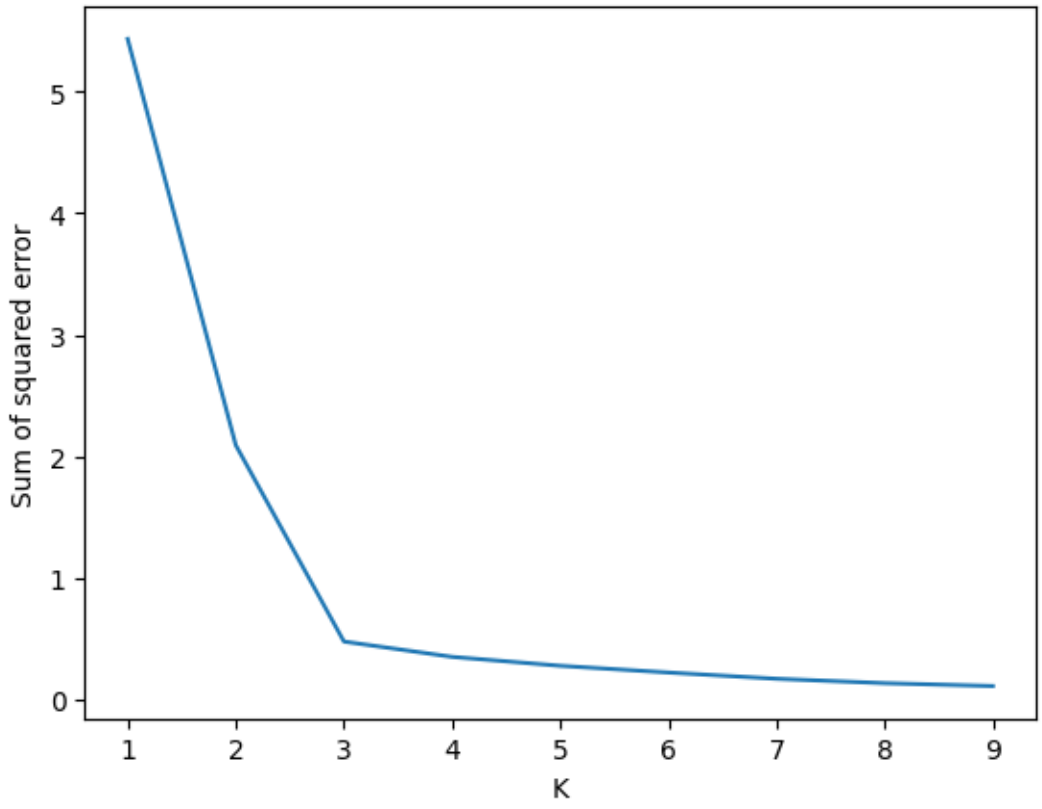
```
#Import Libraries
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
#Import Dataframe
df = pd.read_csv("income.csv")
df.head()
#Scattering
plt.scatter(df.Age,df['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
#Defining the number of clusters (K)
km = KMeans(n_clusters=3)
```

```
km
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
df['cluster']=y_predicted
df.head()
#Making clusters
km.cluster_centers_
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='pur
ple',marker='*',label='centroid')
plt.xlabel('Age')
plt.ylabel('Income ($)')
plt.legend()
#Preprocessing using min max scaler
scaler = MinMaxScaler()
scaler.fit(df[['Income($)']])
df['Income($)'] = scaler.transform(df[['Income($)']])
scaler.fit(df[['Age']])
df['Age'] = scaler.transform(df[['Age']])
#Data Analysis
plt.scatter(df.Age,df['Income($)'])
#Prediction
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df[['Age','Income($)']])
y_predicted
df['cluster']=y_predicted
df.head()
km.cluster_centers_
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1.Age,df1['Income($)'],color='green')
plt.scatter(df2.Age,df2['Income($)'],color='red')
plt.scatter(df3.Age,df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='pur
ple',marker='*',label='centroid')
plt.legend()
```

Elbow Method:

The Elbow Method is a graphical approach for finding the optimal number of clusters (K) in a K-means clustering problem. It helps you choose the right value for K by analyzing the variance explained as a function of the number of clusters. The "elbow" of the curve in the plot represents a point where increasing the number of clusters doesn't significantly improve the model's fit.

```
sse = []
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['Age','Income($)']])
    sse.append(km.inertia_)
plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```



Introduction to Naive Bayes

Naive Bayes is a classification algorithm based on Bayes' theorem with the "naive" assumption of independence among the features. It is a probabilistic algorithm used for classification tasks, such as spam detection, sentiment analysis, document categorization, and more. The "naive" assumption simplifies the modeling process, making it computationally efficient and often surprisingly effective, especially in text and natural language processing applications.

Here's a high-level overview of how Naive Bayes works:

- **Bayes' Theorem:** Naive Bayes is based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. In the context of classification, it calculates the probability of a data point belonging to a particular class based on the features of the data point.
- **Independence Assumption:** The "naive" assumption in Naive Bayes is that the features used for classification are conditionally independent given the class label. In reality, this assumption is rarely true, but it simplifies the calculations and often works well in practice.
- **Training:** To train a Naive Bayes classifier, you collect a labeled dataset where each data point is associated with a class label. The algorithm estimates the probabilities of each feature occurring within each class and the prior probability of each class. This information is used to build a probabilistic model for each class.
- **Prediction:** To classify a new data point, the algorithm calculates the probability of it belonging to each class based on the observed feature values. It then assigns the class with the highest probability as the predicted class for that data point.

The three common types of Naive Bayes classifiers are:

- **Multinomial Naive Bayes:** This is typically used for text classification tasks where features represent the frequency or occurrence of words or tokens in a document. It is especially useful for document categorization and spam detection.
- **Gaussian Naive Bayes:** This is used for continuous data where features are assumed to follow a Gaussian (normal) distribution. It is suitable for numerical features and works well when the feature distributions are approximately normal.
- **Bernoulli Naive Bayes**: This is used for binary data where features are either present or absent. It's commonly used in document classification tasks, like spam detection.

Naive Bayes is known for its simplicity, speed, and efficiency, making it a good choice for many real-world applications. However, it may not perform as well as more complex models like Support Vector Machines, Random Forests, or deep learning models on highly complex tasks. The performance of Naive Bayes can also be influenced by the quality and quantity of training data.

Naive Bayes with Python

Here's a Python example of how to perform Naïve Bayes using the popular scikit-learn library

```
#Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score
#Import Dataset
df = pd.read_csv("titanic.csv")
df.head()
#Data Analysis
df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked
'],axis='columns',inplace=True)
df.head()
# Set input & target
inputs = df.drop('Survived',axis='columns')
target = df.Survived
dummies = pd.get_dummies(inputs.Sex)
dummies.head(3)
inputs = pd.concat([inputs,dummies],axis='columns')
inputs.head(3)
# Dropping Column
inputs.drop(['Sex','male'],axis='columns',inplace=True)
inputs.head(3)
#checking column that contain NaN value
inputs.columns[inputs.isna().any()]
inputs.Age[:10]
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs.head()
#Test Train Split
X_train, X_test, y_train, y_test =
train_test_split(inputs,target,test_size=0.3)

#Importing GaussianNB Model and fitting this model
model = GaussianNB()
model.fit(X_train,y_train)
#Prediction
model.predict(X_test[0:10])
#Accuracy
model.score(X_test,y_test)

#Importing MultinomialNB Model and fitting this model
model = MultinomialNB()
model.fit(X_train,y_train)
model.score(X_test,y_test)
model.predict(X_test[0:10])

#Cross Validation
```

```
cross_val_score(GaussianNB(),X_train, y_train, cv=5)
cross_val_score(MultinomialNB(),X_train, y_train, cv=5)
```

Introduction to K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple and intuitive supervised machine learning algorithm used for classification and regression tasks. KNN makes predictions based on the similarity of data points, where "K" represents the number of nearest neighbors to consider. Here's how KNN works:

For Classification:

- **Training:** In the training phase, KNN stores the entire dataset with its labels.
- **Prediction:** When you want to classify a new data point, KNN calculates the distances (commonly Euclidean distance) between that point and all other data points in the training set.
- **Choosing K:** You specify the value of K, which is the number of nearest neighbors to consider. A smaller K value can lead to a more sensitive model, while a larger K value can make the model more robust but potentially less precise.
- **Voting:** For classification, the KNN algorithm identifies the K-nearest data points (neighbors) to the new point and counts how many of them belong to each class. The class that appears the most among the K-nearest neighbors is assigned as the predicted class for the new data point.

For Regression:

The process for regression using KNN is similar to classification, with a few differences:

- **Training:** The training phase is the same, storing the dataset with labels.
- **Prediction:** When predicting a new data point's continuous value, KNN calculates the distances to all other data points in the training set.
- **Choosing K:** You specify the value of K.
- **Averaging:** For regression, instead of voting, KNN calculates the average (or weighted average) of the labels of the K-nearest neighbors. This average value is assigned as the predicted continuous value for the new data point.

Key considerations when using KNN:

- **Distance Metric:** The choice of distance metric (e.g., Euclidean, Manhattan, or others) can significantly impact the algorithm's performance. It should be selected based on the nature of the data and the problem.
- **Scaling Data:** It's crucial to normalize or standardize features when using KNN, as it's sensitive to the scale of the data. Features with larger scales can dominate the distance calculation.
- **Choosing K:** Selecting the right K value is essential for the model's performance. Smaller K values can lead to more noise and sensitivity to outliers, while larger K values can lead to oversmoothing and a loss of detail.
- **Computational Cost:** KNN can be computationally expensive, as it requires calculating distances to all data points in the training set. This can be an issue for large datasets.

KNN is straightforward and easy to understand, making it a good starting point for many classification and regression problems, especially when you have a small to moderately-sized dataset. However, its performance can be influenced by the choice of K, distance metric, and feature scaling, and it might not work well for high-dimensional data due to the "curse of dimensionality.".

K-Nearest Neighbors with Python

Here's a Python example of how to perform KNN using the popular scikit-learn library

```
#Import Libraries
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sn
from sklearn.metrics import classification_report
#Import Dataset
iris = load_iris()
#Data Analysis
iris.feature_names
iris.target_names
#Dataframe
df = pd.DataFrame(iris.data,columns=iris.feature_names)
df.head()
df['target'] = iris.target
# Applying Lamda Function
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
# Data Visualization
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(df0['sepal length (cm)'], df0['sepal width
(cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width
(cm)'],color="blue",marker='.')
#Test Train Split
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)
#KNN Classifier & Model Fitting
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
```

```
#Prediction
knn.predict([[4.8,3.0,1.5,0.3]])
#Accuracy
knn.score(X_test, y_test)

# Confusion Matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Lab Task:

1. Use iris flower dataset from sklearn library and try to form clusters of flowers using petal width and length features. Drop other two features for simplicity.
    - Figure out if any preprocessing such as scaling would help here
    - Draw elbow plot and from that figure out optimal value of k
2. Use wine dataset from sklearn.datasets to classify wines into 3 categories. Load the dataset and split it into test and train. After that train the model using Gaussian and Multinominal classifier and post which model performs better. Use the trained model to perform some predictions on test data.

Home Task:

1. From sklearn.datasets load digits dataset and do following
    - Classify digits (0 to 9) using KNN classifier. You can use different values for k neighbors and need to figure out a value of K that gives you a maximum score. You can manually try different values of K or use gridsearchcv
    - Plot confusion matrix
    - Plot classification report