UCS2612 Machine Learning Laboratory A5: K Nearest Neighbour Algorithm

Name : Kavin Siva S

Roll No : **3122 21 5001 306**

**----------------------------------------------------------------------------------------------**

Develop a python program to predict the Online Shoppers Purchasing Intention using K- Nearest Neighbour algorithm. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library.

**Code and Output:**

import pandas as pd shopping\_online=pd.read\_csv('/home/mllab2/Desktop/onlineShop/online

\_shoppers\_intention.csv')pip install pandas shopping\_online.info()

shopping\_online.isnull().sum() from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'for col in shopping\_online.select\_dtypes(include=['object','bool']).columns:

# Print the column name and the unique values print(f"{col}: {shopping\_online[col].unique()}")

# Loop over each column in the DataFrame where dtype is 'object'for col in shopping\_online.select\_dtypes(include=['object','bool']).columns:

# Handle missing values (replace with a default value or drop rows) shopping\_online[col].fillna('unknown', inplace=True)

# Convert all values to strings

shopping\_online[col] = shopping\_online[col].astype(str)

# Initialize a LabelEncoder object label\_encoder = preprocessing.LabelEncoder()

# Fit the encoder to the unique values in the column

label\_encoder.fit(shopping\_online[col].unique())

# Transform the column using the encoder

shopping\_online[col] = label\_encoder.transform(shopping\_online[col])

# Print the column name and the unique encoded valuesprint(f"{col}:

{shopping\_online[col].unique()}") pip install scikit-learn

import matplotlib.pyplot as plt import seaborn as sns

correlation\_matrix = shopping\_online.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)

plt.title('Correlation Heatmap') plt.show()

pip install matplotlib pip install seaborn

from sklearn import metrics

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score, cross\_validate, GridSearchCV

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report, confusion\_matrix,roc\_auc\_score, make\_scorer

x\_o=shopping\_online.drop('Revenue',axis=1) y\_o=shopping\_online['Revenue']

from sklearn.model\_selection import train\_test\_split x\_train\_o,x\_test\_o,y\_train\_o,y\_test\_o=train\_test\_split(x\_o,y\_o,test\_size

=0.2,random\_state=42,stratify=y\_o)

from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy\_score

# Initialize the KNN classifier

knn\_classifier = KNeighborsClassifier(n\_neighbors=5) # You can adjustthe number of neighbors

# Train the model knn\_classifier.fit(x\_train\_o, y\_train\_o)

# Predict on the test set

y\_pred\_o = knn\_classifier.predict(x\_test\_o)

# Evaluate the model

accuracy = accuracy\_score(y\_test\_o, y\_pred\_o) print("Accuracy:", accuracy)

import numpy as np class KNNClassifier:

def init (self, k=5):self.k =

k

def fit(self, X, y): self.X\_train = X self.y\_train = y

def predict(self, X):

y\_pred = [self.\_predict(x) for x in X] return np.array(y\_pred)

def \_predict(self, x):

distances = [np.linalg.norm(x - x\_train) for x\_train in self.X\_train]k\_indices = np.argsort(distances)[:self.k]

k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]most\_common = max(set(k\_nearest\_labels),

key=k\_nearest\_labels.count) return most\_common

# Example usage:

# Instantiate the classifier knn\_classifier = KNNClassifier(k=5)

# Train the classifier knn\_classifier.fit(x\_train\_o.values, y\_train\_o.values)

# Predict on the test set

y\_pred\_o = knn\_classifier.predict(x\_test\_o.values)

# Evaluate the model

accuracy = np.mean(y\_pred\_o == y\_test\_o.values) print("Accuracy:", accuracy)

# Example usage:

# Instantiate the classifier knn\_classifier = KNNClassifier(k=3)

# Train the classifier knn\_classifier.fit(x\_train\_o.values, y\_train\_o.values)

# Predict on the test set

y\_pred\_o = knn\_classifier.predict(x\_test\_o.values)

# Evaluate the model

accuracy = np.mean(y\_pred\_o == y\_test\_o.values) print("Accuracy:", accuracy)

import numpy as np class KNNClassifier:

def init (self, k=5, epochs=1):self.k = k self.epochs = epochs

def fit(self, X, y): self.X\_train = X self.y\_train = y

def predict(self, X):

y\_pred = [self.\_predict(x) for x in X] return np.array(y\_pred)

def \_predict(self, x):

distances = [np.linalg.norm(x - x\_train) for x\_train in self.X\_train]k\_indices = np.argsort(distances)[:self.k]

k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]most\_common = max(set(k\_nearest\_labels),

key=k\_nearest\_labels.count) return most\_common

# Iterate through epochs for \_ in range(self.epochs):

for x, y in zip(X, y): self.\_update\_weights(x, y)

# Example usage:

# Instantiate the classifier

knn\_classifier = KNNClassifier(k=5, epochs=3) # Set number of epochs to3

# Train the classifier knn\_classifier.fit(x\_train\_o.values, y\_train\_o.values)

# Predict on the test set

y\_pred\_o = knn\_classifier.predict(x\_test\_o.values)

# Evaluate the model

accuracy = np.mean(y\_pred\_o == y\_test\_o.values) print("Accuracy:", accuracy)

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import pairwise\_distances

class KNNClassifier:

def init (self, k=5, distance='euclidean'):self.k = k self.distance = distance

def fit(self, X, y): self.X\_train = X self.y\_train = y

def predict(self, X):

distances = pairwise\_distances(X, self.X\_train, metric=self.distance)y\_pred

= [self.\_predict(dist) for dist in distances] return np.array(y\_pred)

def \_predict(self, distances):

k\_indices = np.argsort(distances)[:self.k] k\_nearest\_labels

= [self.y\_train[i] for i in k\_indices]most\_common = max(set(k\_nearest\_labels),

key=k\_nearest\_labels.count)

return most\_common

# Function to evaluate accuracy for a given k and distance metric

def evaluate\_knn(k, distance\_metrics, x\_train, y\_train, x\_test, y\_test):accuracies

= []

for distance\_metric in distance\_metrics:

knn\_classifier = KNNClassifier(k=k, distance=distance\_metric) knn\_classifier.fit(x\_train, y\_train)

y\_pred = knn\_classifier.predict(x\_test) accuracy = np.mean(y\_pred == y\_test) accuracies.append(accuracy)

return accuracies

# Define the value of k and distance metricsk = 20 distance\_metrics = ['euclidean', 'manhattan', 'minkowski']

# Evaluate accuracy for the specified k value and distance metricsaccuracies = evaluate\_knn(k, distance\_metrics, x\_train\_o.values, y\_train\_o.values, x\_test\_o.values, y\_test\_o.values)

print(accuracies)#

Plot results plt.figure(figsize=(10, 6))

plt.bar(distance\_metrics, accuracies)

plt.title(f'Accuracy for k={k} and Different Distance Metrics')plt.xlabel('Distance Metric')

plt.ylabel('Accuracy')

plt.ylim(0, 1) # Set y-axis limits between 0 and 1 plt.grid(axis='y')

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

class KNNClassifier:

def init (self, k=5, distance='euclidean'):self.k = k self.distance = distance

def fit(self, X, y): self.X\_train = X self.y\_train = y

def predict\_proba(self, X):

distances = pairwise\_distances(X, self.X\_train, metric=self.distance)y\_probs

= []

for dist in distances:

k\_indices = np.argsort(dist)[:self.k] k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]

class\_probs = [k\_nearest\_labels.count(c) / self.k for c in np.unique(self.y\_train)]

y\_probs.append(class\_probs) return np.array(y\_probs)

# Function to plot ROC curve

def plot\_roc\_curve(y\_true, y\_prob, title): fpr, tpr, \_ = roc\_curve(y\_true, y\_prob) roc\_auc

= auc(fpr, tpr)

plt.plot(fpr, tpr, lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

# Function to plot ROC curves for training and testing data on the samegraph def plot\_roc\_curves(classifier, x\_train, y\_train, x\_test, y\_test, title):

classifier.fit(x\_train, y\_train)

y\_train\_prob = classifier.predict\_proba(x\_train)[:, 1] y\_test\_prob = classifier.predict\_proba(x\_test)[:, 1]

plt.figure(figsize=(8, 6))

plot\_roc\_curve(y\_train, y\_train\_prob, f'Training {title} ROC Curve') plot\_roc\_curve(y\_test, y\_test\_prob, f'Testing {title} ROC Curve')

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('ROC Curves') plt.legend(loc="lower right") plt.show()

# Instantiate KNN classifier

knn\_classifier = KNNClassifier(k=20, distance='euclidean')

# Plot both ROC curves

plot\_roc\_curves(knn\_classifier, x\_train\_o.values, y\_train\_o.values,x\_test\_o.values, y\_test\_o.values, 'KNN Classifier')

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc, pairwise\_distances

class KNNClassifier:

def init (self, k=5, distance='euclidean'):self.k = k self.distance = distance

def fit(self, X, y): self.X\_train = X self.y\_train = y

def predict\_proba(self, X):

distances = pairwise\_distances(X, self.X\_train, metric=self.distance)y\_probs

= []

for dist in distances:

k\_indices = np.argsort(dist)[:self.k] k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]

class\_probs = [k\_nearest\_labels.count(c) / self.k for c in np.unique(self.y\_train)]

y\_probs.append(class\_probs) return np.array(y\_probs)

# Function to plot ROC curve

def plot\_roc\_curve(y\_true, y\_prob, title): fpr, tpr, \_ = roc\_curve(y\_true, y\_prob) roc\_auc

= auc(fpr, tpr)

plt.plot(fpr, tpr, lw=2, label=f'{title} (area = %0.2f)' % roc\_auc)

# Function to plot ROC curves for training and testing data on the samegraph def plot\_roc\_curves(x\_train, y\_train, x\_test, y\_test, k=20):

distance\_metrics = ['euclidean', 'manhattan', 'minkowski'] plt.figure(figsize=(8, 6))

for distance\_metric in distance\_metrics:

knn\_classifier = KNNClassifier(k=k, distance=distance\_metric) knn\_classifier.fit(x\_train, y\_train)

y\_train\_prob = knn\_classifier.predict\_proba(x\_train)[:, 1]y\_test\_prob

= knn\_classifier.predict\_proba(x\_test)[:, 1]

plot\_roc\_curve(y\_train, y\_train\_prob, f'Training

{distance\_metric.upper()}')

plot\_roc\_curve(y\_test, y\_test\_prob, f'Testing

{distance\_metric.upper()}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

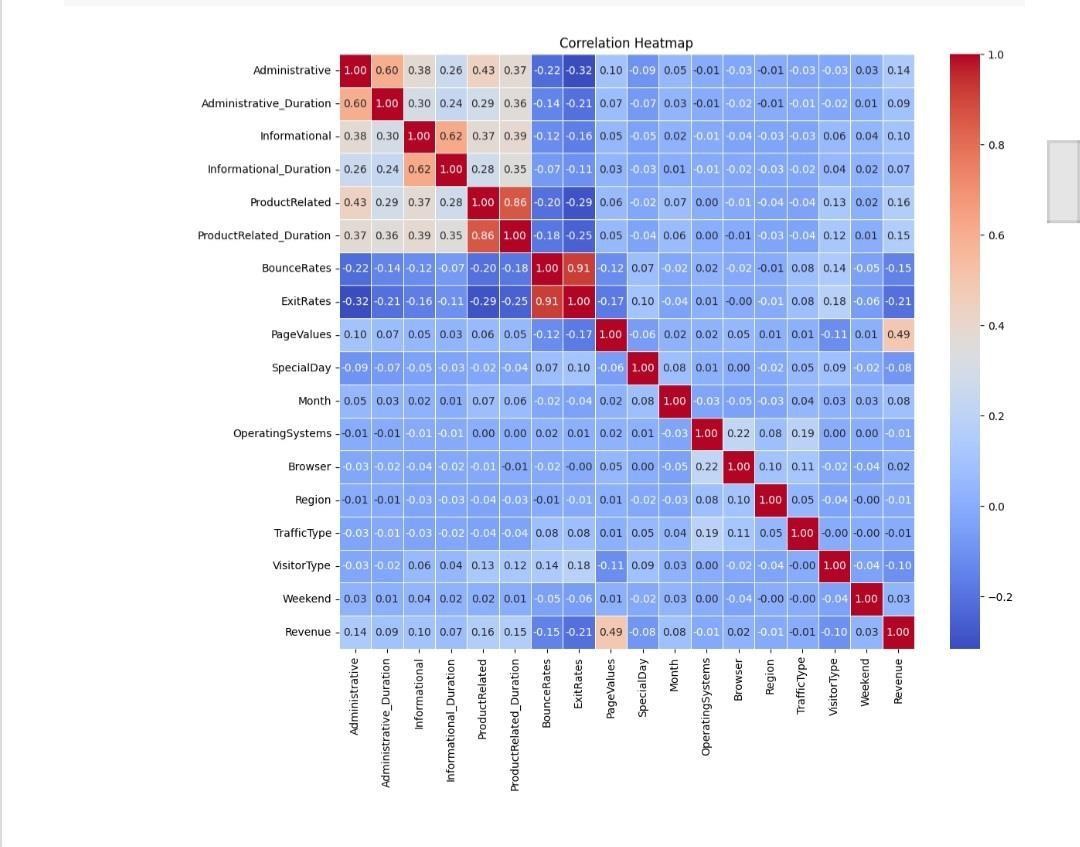
plt.xlim([0.0, 1.0])

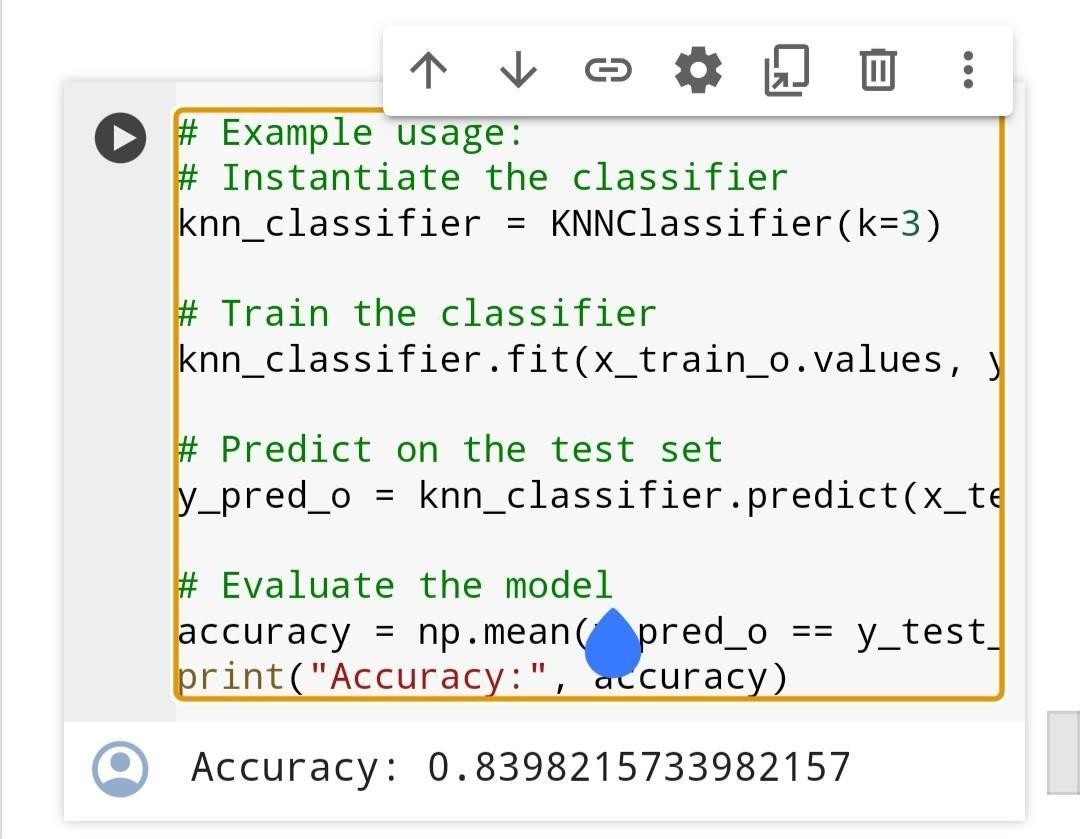
plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate')

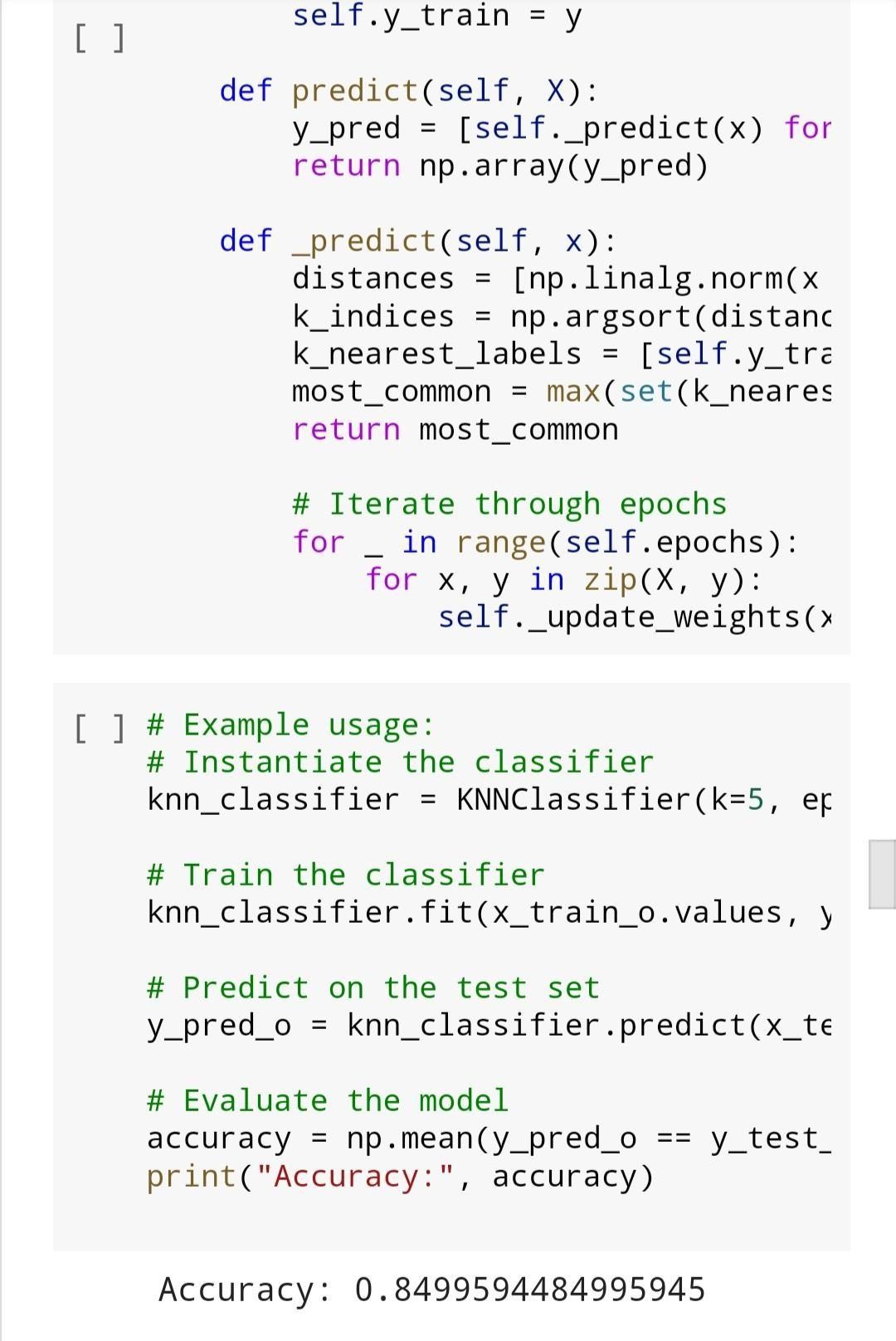
plt.title('ROC Curves for Different Distance Metrics') plt.legend(loc="lower right")

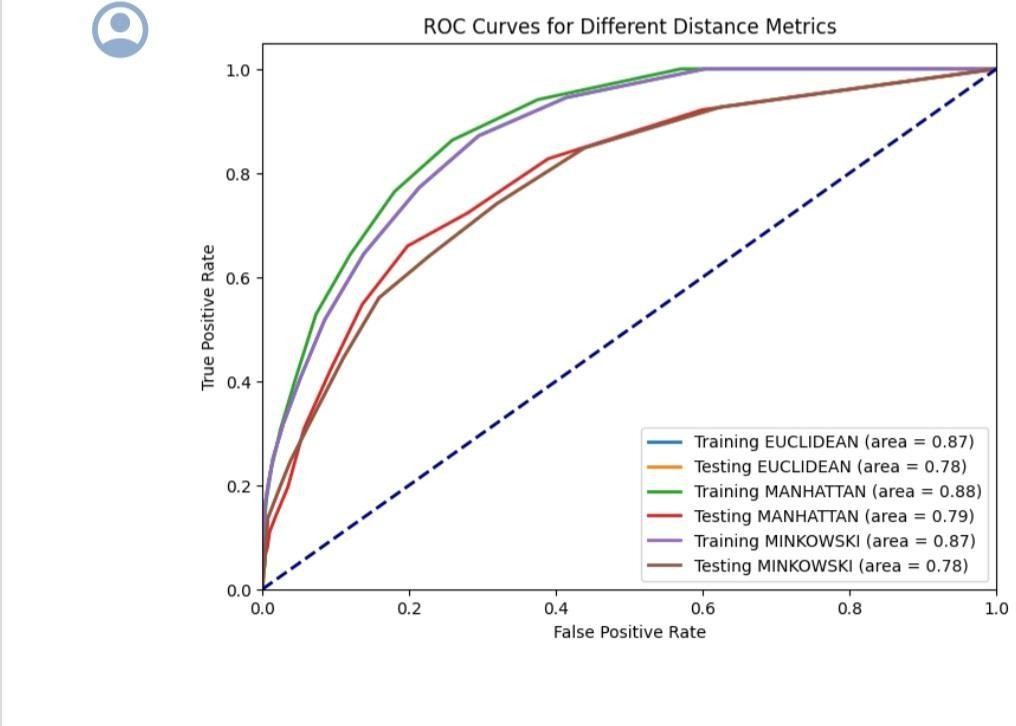
plt.show()

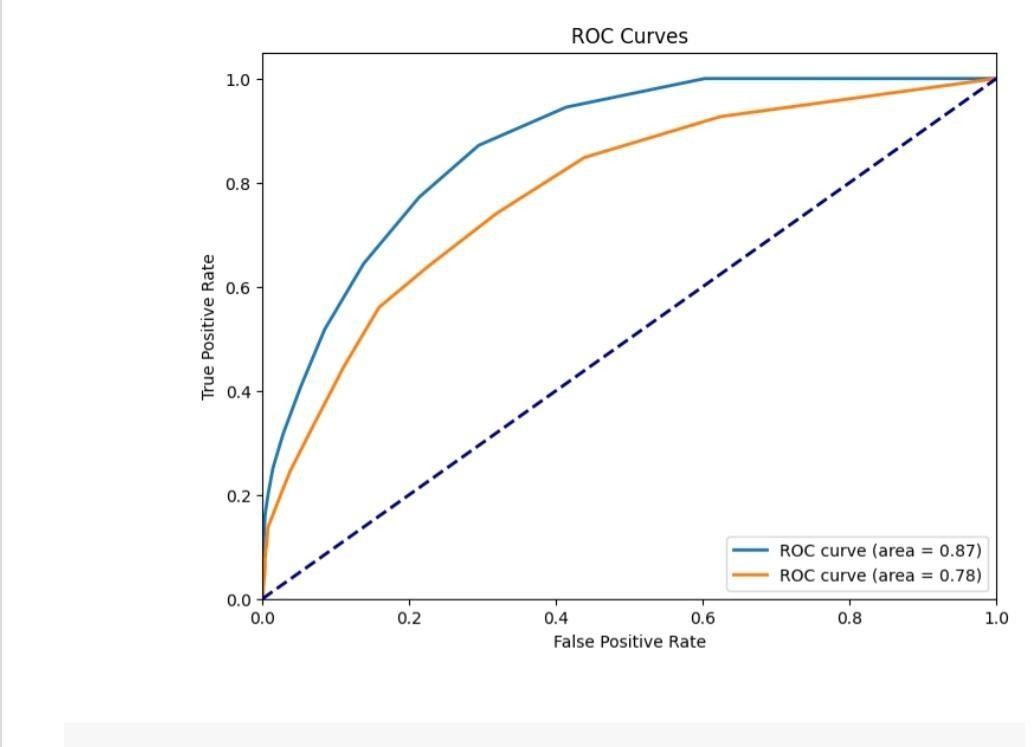
# Plot ROC curves for all distance metrics plot\_roc\_curves(x\_train\_o.values, y\_train\_o.values, x\_test\_o.values,y\_test\_o.values)











**Learning Outcome:-**

* Better Understanding about the K Nearest Neighbours
* The K Nearest Neighbors was performed using built in functions.
* The model worked well for k value of 5
* From the above results I conclude **that K Nearest Neighbors (KNN) Workswell for the dataset**