22AIE213- Machine Learning

Lab Assignment-3

**A1. Evaluate the intraclass spread and interclass distances between the classes in your dataset. If your data deals with multiple classes, you can take any two classes**

**Code:**

import numpy as np

import pandas as pd

# Define the file path and load the dataset

file\_path = "data.csv"

df = pd.read\_csv(file\_path)

# Split data based on class labels (assuming binary classification: 0 and 1)

class\_0 = df[df["target"] == 0].drop(columns=["target"])

class\_1 = df[df["target"] == 1].drop(columns=["target"])

# Calculate the mean (centroid) for each class

centroid\_0 = class\_0.mean(axis=0)

centroid\_1 = class\_1.mean(axis=0)

# Determine the spread (standard deviation) within each class

spread\_0 = class\_0.std(axis=0)

spread\_1 = class\_1.std(axis=0)

# Measure the Euclidean distance between the centroids

distance\_between\_centroids = np.linalg.norm(centroid\_0 - centroid\_1)

# Print results

print(f"Centroid of Class 0:\n{centroid\_0}\n")

print(f"Centroid of Class 1:\n{centroid\_1}\n")

print(f"Spread (Standard Deviation) of Class 0:\n{spread\_0}\n")

print(f"Spread (Standard Deviation) of Class 1:\n{spread\_1}\n")

print(f"Euclidean Distance Between Centroids:\n{distance\_between\_centroids}")

**Output:**

Class 0 Centroid:

age 52.706667

sex 0.553333

cp 1.820000

trestbps 128.866667

chol 244.213333

fbs 0.153333

restecg 0.860000

thalach 158.333333

exang 0.153333

oldpeak 0.622667

slope 0.400000

ca 0.286667

thal 1.373333

dtype: float64

Class 1 Centroid:

age 56.591667

sex 0.833333

cp 2.616667

trestbps 134.441667

chol 256.466667

fbs 0.141667

restecg 1.225000

thalach 138.858333

exang 0.550000

oldpeak 1.584167

slope 0.816667

ca 1.150000

thal 2.383333

dtype: float64

Class 0 Spread (Standard Deviation):

age 9.509830

sex 0.498813

cp 0.927362

trestbps 16.457660

chol 54.019085

fbs 0.361516

restecg 0.990085

thalach 19.283357

exang 0.361516

oldpeak 0.800851

slope 0.590757

ca 0.648557

thal 0.755709

dtype: float64

Class 1 Spread (Standard Deviation):

age 8.116273

sex 0.374241

cp 0.779823

trestbps 19.095424

chol 47.969166

fbs 0.350170

restecg 0.974140

thalach 23.130719

exang 0.499580

oldpeak 1.282067

slope 0.564843

ca 1.034286

thal 0.890504

dtype: float64

Interclass Distance:

24.071995636354426

**A2. Take any feature from your dataset. Observe the density pattern for that feature by plotting the histogram. Use buckets (data in ranges) for histogram generation and study. Calculate the mean and variance from the available data.**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

file\_path = r"data.csv"  # Path to the uploaded file

df = pd.read\_csv(file\_path)

# Select a numerical feature for analysis

feature = "age"  # Modify this to analyze a different feature

# Extract and clean the data (remove missing values if any)

data = df[feature].dropna()

# Compute histogram data

hist\_counts, bin\_edges = np.histogram(data, bins=10)  # Dividing data into 10 intervals

# Calculate statistical properties

mean\_val = data.mean()

variance\_val = data.var()

# Generate the histogram plot

plt.figure(figsize=(8, 5))

plt.hist(data, bins=10, color='royalblue', edgecolor='black', alpha=0.75)

plt.xlabel(feature)

plt.ylabel("Frequency")

plt.title(f"Distribution of {feature}")

plt.grid(axis='y', linestyle='--', alpha=0.6)

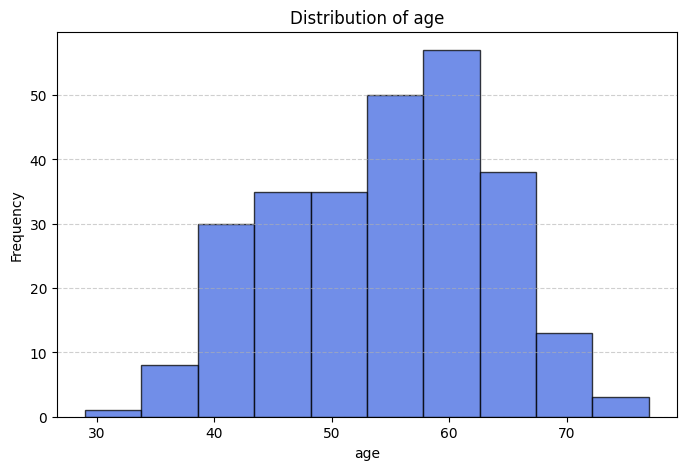
plt.show()

**Output:**

Feature Analyzed: age

Mean: 54.43

Variance: 82.98



**A3. Take any two feature vectors from your dataset. Calculate the Minkwoski distance with r from 1 to 10. Make a plot of the distance and observe the nature of this graph.**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Load dataset

file\_path = r"data.csv"  # Adjust if necessary

df = pd.read\_csv(file\_path)

# Select two numerical features

feature\_1 = "age"  # Modify as needed

feature\_2 = "trestbps"  # Modify as needed

# Extract and clean the data (remove missing values)

data\_1 = df[feature\_1].dropna().values

data\_2 = df[feature\_2].dropna().values

# Ensure both features have the same number of values

min\_len = min(len(data\_1), len(data\_2))

data\_1, data\_2 = data\_1[:min\_len], data\_2[:min\_len]

# Compute Minkowski distance for r values from 1 to 10

r\_values = np.arange(1, 11)

minkowski\_distances = [np.sum(np.abs(data\_1 - data\_2) \*\* r) \*\* (1 / r) for r in r\_values]

# Plot the distances

plt.figure(figsize=(8, 5))

plt.plot(r\_values, minkowski\_distances, marker='o', linestyle='-', color='darkorange', markersize=6)

plt.xlabel("Minkowski Parameter (r)")

plt.ylabel("Minkowski Distance")

plt.title(f"Minkowski Distance between {feature\_1} and {feature\_2}")

plt.xticks(r\_values)

plt.grid(alpha=0.6, linestyle='--')

plt.show()

# Display calculated distances

for r, dist in zip(r\_values, minkowski\_distances):

    print(f"Minkowski Distance (r={r}): {dist:.4f}")

**Output:**

****

**A4. Divide dataset in your project into two parts – train & test set. To accomplish this, use the traintest\_split() function available in SciKit**

**Code :**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

file\_path = "data.csv"

df = pd.read\_csv(file\_path)

# Ensure binary classification (filtering only classes 0 and 1 if needed)

df = df[df["target"].isin([0, 1])]

# Split features and target variable

X = df.drop(columns=["target"])

y = df["target"]

# Split data into train (70%) and test (30%) sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Display dataset shape

print("Training set shape:", X\_train.shape, y\_train.shape)

print("Testing set shape:", X\_test.shape, y\_test.shape)

**Output:**

Training set shape: (189, 13) (189,)

Testing set shape: (81, 13) (81,)

**A5. Train a kNN classifier (k =3) using the training set obtained from above exercise.**

**Code:**

from sklearn.neighbors import KNeighborsClassifier

# Initialize kNN with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the model

knn.fit(X\_train, y\_train)

print("kNN model trained successfully!")

**A6. Test the accuracy of the kNN using the test set obtained from above exercise.**

**Code:**

accuracy = knn.score(X\_test, y\_test)

print(f"Accuracy of kNN (k=3) on test set: {accuracy:.4f}")

**Output:**

Accuracy of kNN (k=3) on test set: 0.6667

**A7. Use the predict() function to study the prediction behavior of the classifier for test vectors. >>> neigh.predict(X\_test) Perform classification for a given vector using neigh.predict(<>). This shall produce the class of the test vector (test\_vect is any feature vector from your test set).**

**Code:**

# Predict class labels for test set

y\_pred = knn.predict(X\_test)

# Display first 10 predictions

print("First 10 predictions:", y\_pred[:10])

**Output:**

First 10 predictions: [1 1 0 0 1 1 0 0 0 1]

**A8.**  **Make k = 1 to implement NN classifier and compare the results with kNN (k = 3). Vary k from 1 to 11 and make an accuracy plot.**

**Code:**

import matplotlib.pyplot as plt

k\_values = range(1, 12) # k from 1 to 11

accuracies = []

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train, y\_train)

acc = knn.score(X\_test, y\_test)

accuracies.append(acc)

# Plot accuracy vs k

plt.figure(figsize=(8, 5))

plt.plot(k\_values, accuracies, marker='o', linestyle='-', color='b')

plt.xlabel("k (Number of Neighbors)")

plt.ylabel("Test Set Accuracy")

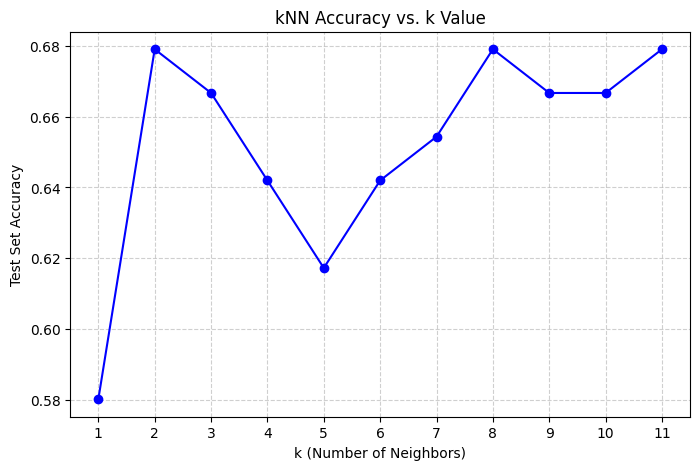
plt.title("kNN Accuracy vs. k Value")

plt.xticks(k\_values)

plt.grid(alpha=0.6, linestyle="--")

plt.show()

**Output:**



**A9. Please evaluate confusion matrix for your classification problem. From confusion matrix, the other performance metrics such as precision, recall and F1-Score measures for both training and test data. Based on your observations, infer the models learning outcome (underfit / regularfit / overfit).**

**Code:**

from sklearn.metrics import confusion\_matrix, classification\_report

# Compute confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

# Compute precision, recall, F1-score

report = classification\_report(y\_test, y\_pred)

print("Classification Report:\n", report)

**Output:**

Confusion Matrix:

[[34 15]

[12 20]]

Classification Report:

precision recall f1-score support

0 0.74 0.69 0.72 49

1 0.57 0.62 0.60 32

accuracy 0.67 81

macro avg 0.66 0.66 0.66 81

weighted avg 0.67 0.67 0.67 81