prdm-hw1

May 3, 2024

Pujitha kallu - w1653660

The goal is to explore how different proximity measures impact various types of analyses on the Iris dataset. Proximity measures are crucial for tasks like clustering, anomaly detection, and supervised classification. By using real data from the Iris dataset, we can uncover valuable insights.

To start, we'll define the species mean vector for each Iris species. This involves computing the mean of the four-dimensional points associated with each species label. The dataset contains six columns, with one representing the label and one representing the ID, while the remaining four columns constitute the four dimensions of the data.

[ ]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

1. Compute the Euclidean distance between all pairs of "species mean vectors" and report them in a table.

[ ]:
```
import pandas as pd
from scipy.spatial.distance import pdist, squareform

# Step 1: Read the data and filter out unnecessary columns

file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'

# Read the CSV file
df = pd.read_csv(file_path)

df = df.drop(columns=["id"])  # Drop the 'id' column

# Step 2: Compute the mean vector for each species
mean_vectors = df.groupby('label').mean()


print("Mean species vectors:", mean_vectors )


# Step 3: Compute the Euclidean distance between each pair of mean vectors
distances = pdist(mean_vectors, metric='euclidean')
```

```
# Step 4: Report the distances in a table
distance_matrix = squareform(distances)
distance_df = pd.DataFrame(distance_matrix, columns=mean_vectors.index,␣
 ↪index=mean_vectors.index)

# Print the distance table
print("Euclidean distance between species mean vectors:")
print(distance_df)
```

```
Mean species vectors:                 Sepal length  Sepal width  Petal length
Petal width
label
Iris-setosa              5.006        3.418         1.464        0.244
Iris-versicolor          5.936        2.770         4.260        1.326
Iris-virginica           6.588        2.974         5.552        2.026
Euclidean distance between species mean vectors:
label            Iris-setosa  Iris-versicolor  Iris-virginica
label
Iris-setosa         0.000000         3.205175        4.752592
Iris-versicolor     3.205175         0.000000        1.620489
Iris-virginica      4.752592         1.620489        0.000000
```

---

2. Same as 1 except that standardize each of the column vectors (of the 4 dimensions first). Use the z-score standardization, i.e. subtract the column vector's mean from the value and divide by the column vector's standard deviation).

[ ]:
```
import pandas as pd
from scipy.spatial.distance import pdist, squareform
from sklearn.preprocessing import StandardScaler

# Step 1: Read the data and filter out unnecessary columns

file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'

# Read the CSV file
df = pd.read_csv(file_path)
df = df.drop(columns=["id"])  # Drop the 'id' column

# Standardize the data using z-score standardization
scaler = StandardScaler()
df_std = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]), columns=df.
 ↪columns[:-1])
df_std['label'] = df['label']

# Compute the mean vector for each species
```

```
mean_vectors = df_std.groupby('label').mean()

# Compute the Euclidean distance between each pair of mean vectors
distances = pdist(mean_vectors, metric='euclidean')

# Report the distances in a table
distance_df = pd.DataFrame(squareform(distances), columns=mean_vectors.index,␣
 ↪index=mean_vectors.index)
print("Euclidean distance between species mean vectors (after z-score␣
 ↪standardization):")
print(distance_df)
```

```
Euclidean distance between species mean vectors (after z-score standardization):
label            Iris-setosa  Iris-versicolor  Iris-virginica
label
Iris-setosa         0.000000         2.840756        3.952601
Iris-versicolor     2.840756         0.000000        1.494566
Iris-virginica      3.952601         1.494566        0.000000
```

---

3. Same as 1 except use Mahalanobis Distance.

[ ]:
```
#for each pair of species
import pandas as pd
import numpy as np

# Step 1: Read the data and filter out unnecessary columns
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'
df = pd.read_csv(file_path)
df = df.drop(columns=["id"])  # Drop the 'id' column

# Step 2: Compute the covariance matrix for each species
cov_matrices = {}

for species in df['label'].unique():
    subset = df[df['label'] == species].iloc[:, :-1]  # Exclude the label column
    cov_matrix = np.cov(subset, rowvar=False)
    cov_matrices[species] = cov_matrix

# Step 3: Calculate the Mahalanobis distance for each pair of species
mahalanobis_distances = pd.DataFrame(index=cov_matrices.keys(),␣
 ↪columns=cov_matrices.keys())

for species_i, cov_matrix_i in cov_matrices.items():
    for species_j, cov_matrix_j in cov_matrices.items():
        if species_i != species_j:
            # Compute the mean vectors
```

```
            mean_vector_i = df[df['label'] == species_i].iloc[:, :-1].mean().
 ↪values
            mean_vector_j = df[df['label'] == species_j].iloc[:, :-1].mean().
 ↪values

            # Compute the Mahalanobis distance
            mean_diff = mean_vector_i - mean_vector_j
            mahalanobis_dist = np.sqrt(np.dot(np.dot(mean_diff, np.linalg.
 ↪inv(cov_matrix_i)), mean_diff.T))
            mahalanobis_distances.loc[species_i, species_j] = mahalanobis_dist

# Print the Mahalanobis distances
print("Mahalanobis Distances between species:")
print(mahalanobis_distances)
```

```
Mahalanobis Distances between species:
                Iris-setosa Iris-versicolor Iris-virginica
Iris-setosa             NaN       18.194879      26.905889
Iris-versicolor   10.131561             NaN       4.226902
Iris-virginica    12.972528        3.720048            NaN
```

[ ]:
```
#for entire dataset
import pandas as pd
from scipy.spatial.distance import mahalanobis
import numpy as np

# Step 1: Read the data and filter out unnecessary columns
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'
df = pd.read_csv(file_path)
df = df.drop(columns=["id"])  # Drop the 'id' column

# Step 2: Compute the mean vector for each species
mean_vectors = df.groupby('label').mean()

# Step 3: Compute the covariance matrix for the entire dataset
cov_matrix = np.cov(df.iloc[:, :-1], rowvar=False)

# Step 4: Calculate the Mahalanobis distance between each pair of mean vectors
mahalanobis_distances = pd.DataFrame(index=mean_vectors.index,␣
 ↪columns=mean_vectors.index)

for species1 in mean_vectors.index:
    for species2 in mean_vectors.index:
        mean_diff = mean_vectors.loc[species1] - mean_vectors.loc[species2]
        mahalanobis_dist = np.sqrt(np.dot(np.dot(mean_diff, np.linalg.
 ↪inv(cov_matrix)), mean_diff.T))
        mahalanobis_distances.loc[species1, species2] = mahalanobis_dist
```

```
print("Mahalanobis Distance between species mean vectors:")
print(mahalanobis_distances)
```

```
[[ 0.68569351 -0.03926846  1.27368233  0.5169038 ]
 [-0.03926846  0.18800403 -0.32171275 -0.11798121]
 [ 1.27368233 -0.32171275  3.11317942  1.29638747]
 [ 0.5169038  -0.11798121  1.29638747  0.58241432]]
Mahalanobis Distance between species mean vectors:
label           Iris-setosa Iris-versicolor Iris-virginica
label
Iris-setosa             0.0        1.844099        2.35485
Iris-versicolor    1.844099             0.0        1.29136
Iris-virginica      2.35485         1.29136            0.0
```

---

4. Same as 1 except use Cosine Similarity.

[ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity

# Step 1: Read the data and filter out unnecessary columns
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'

# Read the CSV file
df = pd.read_csv(file_path)
df = df.drop(columns=["id"])  # Drop the 'id' column

# Step 2: Compute the mean vector for each species
mean_vectors = df.groupby('label').mean()

# Step 3: Compute the cosine similarity between each pair of mean vectors
similarities = cosine_similarity(mean_vectors)

# Step 4: Report the similarities in a table
similarity_df = pd.DataFrame(similarities, index=mean_vectors.index,␣
 ↪columns=mean_vectors.index)

# Print the similarity table
print("Table of Cosine Similarities:")
print(similarity_df)
```

```
Table of Cosine Similarities:
label            Iris-setosa  Iris-versicolor  Iris-virginica
label
Iris-setosa         1.000000         0.924848        0.888438
Iris-versicolor     0.924848         1.000000        0.995713
```

```
Iris-virginica      0.888438          0.995713          1.000000
```

5.Same as 1 except use Pearson Correlation.

[ ]:
```python
import pandas as pd
import numpy as np

# Define the file path
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'

# Function to read the data and drop unnecessary columns
def read_and_drop_columns(file_path):
    df = pd.read_csv(file_path)
    return df.drop(columns=["id"])

# Function to standardize the data
def standardize_data(df):
    standardized_df = df.copy()
    for column in df.columns[:-1]:  # Exclude the label column
        mean = df[column].mean()
        std = df[column].std()
        standardized_df[column] = (df[column] - mean) / std
    return standardized_df

# Function to compute covariance matrices for each species
def compute_corr_matrices(df):
    corr_matrices = {}
    for species in df['label'].unique():
        subset = df[df['label'] == species].iloc[:, :-1]  # Exclude the label␣
 ↪column
        corr_matrices[species] = subset.corr()
    return corr_matrices

# Read the data and drop unnecessary columns
df = read_and_drop_columns(file_path)

# Step 2: Standardize the data
standardized_df = standardize_data(df)

# Step 3: Compute the correlation matrix for each species
corr_matrices = compute_corr_matrices(standardized_df)

# Print the correlation matrices for each species
for species, corr_matrix in corr_matrices.items():
    print(f"Correlation matrix for {species}:")
    print(corr_matrix)
```

```
Correlation matrix for Iris-setosa:
              Sepal length  Sepal width  Petal length  Petal width
Sepal length      1.000000     0.746780      0.263874     0.279092
Sepal width       0.746780     1.000000      0.176695     0.279973
Petal length      0.263874     0.176695      1.000000     0.306308
Petal width       0.279092     0.279973      0.306308     1.000000
Correlation matrix for Iris-versicolor:
              Sepal length  Sepal width  Petal length  Petal width
Sepal length      1.000000     0.525911      0.754049     0.546461
Sepal width       0.525911     1.000000      0.560522     0.663999
Petal length      0.754049     0.560522      1.000000     0.786668
Petal width       0.546461     0.663999      0.786668     1.000000
Correlation matrix for Iris-virginica:
              Sepal length  Sepal width  Petal length  Petal width
Sepal length      1.000000     0.457228      0.864225     0.281108
Sepal width       0.457228     1.000000      0.401045     0.537728
Petal length      0.864225     0.401045      1.000000     0.322108
Petal width       0.281108     0.537728      0.322108     1.000000
```

[ ]:
```python
import pandas as pd
from scipy.stats import pearsonr

# Step 1: Read the data and filter out unnecessary columns
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'
df = pd.read_csv(file_path)
df = df.drop(columns=["id"])  # Drop the 'id' column

# Step 2: Compute the mean vector for each species
mean_vectors = df.groupby('label').mean()

# Step 3: Compute the Pearson correlation between each pair of mean vectors
correlation_df = pd.DataFrame(index=mean_vectors.index, columns=mean_vectors.
 ↪index)
for species1 in mean_vectors.index:
    for species2 in mean_vectors.index:
        if species1 != species2:
            correlation, _ = pearsonr(mean_vectors.loc[species1], mean_vectors.
 ↪loc[species2])
            correlation_df.loc[species1, species2] = correlation

# Replace NaN values with 0
correlation_df = correlation_df.fillna(0)

# Print the correlation table
print("Pearson correlation between species mean vectors:")
print(correlation_df)
```

```
Pearson correlation between species mean vectors:
```

label	Iris-setosa	Iris-versicolor	Iris-virginica
label			
Iris-setosa	0.000000	0.763854	0.618438
Iris-versicolor	0.763854	0.000000	0.979489
Iris-virginica	0.618438	0.979489	0.000000

---

6. You now have five tables. Explain the pros and cons of each of the measures for the purpose of quantifying how similar pairs of species are.

Ans:

1. Euclidean Distance:
   - Pros:
     - Straightforward calculation: Easy to compute and understand, providing a simple measure of distance between points in a multidimensional space.
     - Intuitive interpretation: Smaller distances imply greater similarity, while larger distances indicate greater dissimilarity.
   - **Cons**:
     - Sensitive to scale: Treats all dimensions equally, making it sensitive to differences in scale between features. Standardizing the data can mitigate this issue.
     - Not robust to outliers: Outliers can significantly influence the Euclidean distance, especially in high-dimensional spaces.
2. Standardized Euclidean Distance:
   - Pros:
     - Scale-invariant: Standardizing the data ensures that all features have a mean of 0 and standard deviation of 1, making the distance measure less sensitive to differences in scale.
     - Compensates for variable scales: Provides a more accurate measure of similarity by removing the influence of variable scales.
   - Cons:
     - May lose interpretability: After standardization, the resulting distances may be more challenging to interpret in the original feature space.
3. Mahalanobis Distance:
   - Pros:
     - Accounts for feature correlations: Takes into account the correlations between features, providing a more accurate measure of similarity for correlated data.
     - Scale-invariant and robust to outliers: Adjusts for differences in scale and is less sensitive to outliers compared to the Euclidean distance.
   - Cons:
     - Requires estimation of covariance matrix: Relies on estimating the covariance matrix from the data, which can be challenging, especially with limited samples.
     - Assumes multivariate normality: Assumes that the data follows a multivariate normal distribution, which may not hold true for all datasets.
4. Cosine Similarity:
   - Pros:
     - Ignores magnitude: Focuses on the direction of vectors rather than their magnitude, making it suitable for comparing documents or text data.
     - Effective for high-dimensional sparse data: Well-suited for high-dimensional datasets

where many features are zero or missing.

- Cons:
  - Limited to non-negative data: Requires non-negative data as negative values can affect similarity calculations.
  - Does not consider feature correlations: Ignores correlations between features, which may be important in some contexts.

5. Pearson Correlation:
   - Pros:
     - Measures linear relationship: Quantifies the strength and direction of the linear relationship between two variables, providing insight into linear associations between features.
     - Widely applicable: Commonly used in various statistical analyses and machine learning tasks.
   - Cons:
     - Assumes linear relationship: Assumes that the relationship between variables is linear, which may not hold true for all datasets.
     - Sensitive to outliers: Outliers can significantly affect correlation coefficients, potentially skewing similarity measures.

---

7. For each of the species, find if there are outliers in the species. For this purpose compare the 4-dimensional vector of any particular object (row in the csv file) with the 4-dimensional mean vector of the species. Use Mahalanobis Distance as the proximity measure. You will need to choose appropriate thresholds for discerning whether a data point is anomalous or not. Explain how you did this.

Ans:To determine the threshold for identifying outliers using the Mahalanobis distance, we utilize the Chi-Square distribution, which provides critical values for a given significance level (alpha) and degrees of freedom. Here's the theoretical explanation of how we calculate this threshold:

1. Significance Level (alpha): The significance level (alpha) is a predefined threshold that determines the probability of rejecting the null hypothesis when it is actually true. In the context of outlier detection, it represents the maximum tolerable probability of falsely identifying a non-outlying point as an outlier. Common choices for alpha include 0.05 (5% significance level) or 0.01 (1% significance level), but it can be adjusted based on the specific requirements and the desired stringency of the outlier detection process.

2. Degrees of Freedom: The degrees of freedom correspond to the number of dimensions in the dataset. In our case, since we are dealing with a 4-dimensional dataset (4 features), the degrees of freedom are set to 4.

3. Chi-Square Distribution: The Chi-Square distribution is a probability distribution that arises in statistical inference, particularly in hypothesis testing and confidence interval estimation for variance. The critical values of the Chi-Square distribution depend on the degrees of freedom and the significance level (alpha).

4. Chi-Square Critical Value: Using the `chi2.ppf()` function from the SciPy library, we calculate the Chi-Square critical value corresponding to the desired significance level (alpha) and degrees of freedom. This critical value represents the threshold beyond which data points are considered outliers. For instance, if we choose a 5% significance level (alpha = 0.05), we

obtain the Chi-Square critical value such that 95% of the distribution falls below this value.

5. Threshold for Mahalanobis Distance:The Mahalanobis distance is calculated for each data point in the dataset. If the Mahalanobis distance of a data point exceeds the Chi-Square critical value, it is considered an outlier. The threshold derived from the Chi-Square distribution ensures that the outlier detection process adheres to the chosen significance level, thereby controlling the rate of false positives.

[ ]:
```
import pandas as pd
import numpy as np
from scipy.stats import chi2
from numpy.linalg import inv
from scipy.spatial.distance import mahalanobis

# Read the data
iris_data = pd.read_csv('/content/drive/My Drive/Pujitha_kallu_PRDM/
 ↪Assignment-1/iris.csv')

# Mean vectors for each species
mean_vectors = {}
for species in iris_data['label'].unique():
    mean_vectors[species] = iris_data[iris_data['label'] == species][['Sepal␣
 ↪length', 'Sepal width', 'Petal length', 'Petal width']].mean().values

# Degrees of freedom (number of dimensions)
degrees_of_freedom = 4

# Significance level (1 - alpha) for outlier detection
alpha = 0.45

# Calculate the Chi-Square critical value for the given alpha and degrees of␣
 ↪freedom
chi2_critical_value = chi2.ppf(1 - alpha, degrees_of_freedom)

# Dictionary to store outliers for each species
outliers = {}

# Detect outliers for each species using Mahalanobis distance
for species, mean_vector in mean_vectors.items():
    # Check if there are enough data points for this species
    if len(iris_data[iris_data['label'] == species]) < degrees_of_freedom + 1:
        print(f"Not enough data points for species {species}. Skipping outlier␣
 ↪detection.")
        continue

    # Calculate the inverse of the covariance matrix for the current species
    species_cov_matrix = iris_data[iris_data['label'] == species][['Sepal␣
 ↪length', 'Sepal width', 'Petal length', 'Petal width']].cov()
```

```
    inv_species_cov_matrix = inv(species_cov_matrix)

    # Calculate Mahalanobis distance for each data point of the current species
    distances = []
    for _, row in iris_data[iris_data['label'] == species][['Sepal length',␣
 ↪'Sepal width', 'Petal length', 'Petal width']].iterrows():
        data_point = row.values
        mahalanobis_distance = mahalanobis(data_point, mean_vector,␣
 ↪inv_species_cov_matrix)
        distances.append(mahalanobis_distance)

    # Determine outliers based on the threshold
    outliers[species] = iris_data[iris_data['label'] == species][distances >␣
 ↪chi2_critical_value]

# Print outliers for each species
for species, outlier_data in outliers.items():
    print(f"Outliers for {species}:")
    print(outlier_data.to_string(index=True))
```

```
Outliers for Iris-setosa:
Empty DataFrame
Columns: [Sepal length, Sepal width, Petal length, Petal width, id, label]
Index: []
Outliers for Iris-versicolor:
Empty DataFrame
Columns: [Sepal length, Sepal width, Petal length, Petal width, id, label]
Index: []
Outliers for Iris-virginica:
     Sepal length  Sepal width  Petal length  Petal width      id
label
118           7.7          2.6           6.9          2.3  id_119  Iris-
virginica
```

---

8. Same as 7 except use Euclidean distance as the proximity measure. Do not standardize the column vectors. You will need to choose appropriate thresholds for discerning whether a data point is anomalous or not. Explain how you did this.

Ans: When using Euclidean distance as the proximity measure for outlier detection without standardizing the column vectors, we typically rely on statistical measures such as the Interquartile Range (IQR) to determine appropriate thresholds for identifying anomalous data points. Here's how this process works:

1. **Interquartile Range (IQR):** The IQR is a measure of statistical dispersion, representing the range between the first quartile (Q1, 25th percentile) and the third quartile (Q3, 75th percentile) of a dataset. It provides insights into the spread of the data, particularly in the middle 50% of the distribution, making it robust against outliers.

2. **Euclidean Distance Calculation:** For each data point in the dataset, we compute its Euclidean distance from the centroid or mean vector of its respective group (e.g., species in the Iris dataset). The Euclidean distance is a direct measure of the straight-line distance between two points in a multi-dimensional space.

3. **Threshold Determination:** We use the IQR of the Euclidean distances for each group (e.g., species) to establish thresholds for identifying outliers. The IQR serves as a robust measure of variability, capturing the spread of the distances within the dataset.

4. **Outlier Detection:** Any data point with a Euclidean distance greater than Q3 + k × IQR or less than Q1 - k × IQR can be considered an outlier, where k is a multiplier representing the desired level of stringency in outlier detection. Common choices for k include 1.5 or 2, with larger values indicating a higher tolerance for extreme outliers.

5. **Adjusting Thresholds:** The choice of k allows us to adjust the sensitivity of the outlier detection algorithm. Higher values of k result in broader thresholds, capturing more data points as outliers, while lower values of k lead to narrower thresholds, identifying only the most extreme outliers.

[22]:
```
import pandas as pd
from scipy.spatial.distance import euclidean

# Read the data
iris_data = pd.read_csv('/content/drive/My Drive/Pujitha_kallu_PRDM/
 ↪Assignment-1/iris.csv')

# Mean vectors for each species
mean_vectors = {}
for species in iris_data['label'].unique():
    mean_vectors[species] = iris_data[iris_data['label'] == species][['Sepal␣
 ↪length', 'Sepal width', 'Petal length', 'Petal width']].mean().values

# Dictionary to store outliers for each species
outliers = {}

# Constant for determining the threshold (k times IQR)
k = 1.5

# Dictionary to store outlier counts for each species
outlier_counts = {}

# Detect outliers for each species using Euclidean distance without␣
 ↪standardization
for species, mean_vector in mean_vectors.items():
    # Calculate Euclidean distance for each data point of the current species
    distances = []
    for _, row in iris_data[iris_data['label'] == species][['Sepal length',␣
 ↪'Sepal width', 'Petal length', 'Petal width']].iterrows():
        data_point = row.values
```

```
        euclidean_distance = euclidean(data_point, mean_vector)
        distances.append(euclidean_distance)
    # Calculate quartiles and IQR
    q1 = pd.Series(distances).quantile(0.25)
    q3 = pd.Series(distances).quantile(0.75)
    iqr = q3 - q1
    # Reset index of the DataFrame
    species_data = iris_data[iris_data['label'] == species].
↪reset_index(drop=True)
    # Determine outliers based on the threshold (Q3 + k * IQR)
    outliers[species] = species_data[pd.Series(distances) > (q3 + k * iqr)]
    # Count outliers
    outlier_counts[species] = len(outliers[species])

# Print outliers and their counts for each species
for species, outlier_data in outliers.items():
    print(f"Outliers for {species}:")
    print(outlier_data.to_string(index=True))
    print(f"Number of outliers: {outlier_counts[species]}")
```

```
Outliers for Iris-setosa:
    Sepal length  Sepal width  Petal length  Petal width     id        label
15           5.7          4.4           1.5          0.4  id_16  Iris-setosa
41           4.5          2.3           1.3          0.3  id_42  Iris-setosa
Number of outliers: 2
Outliers for Iris-versicolor:
    Sepal length  Sepal width  Petal length  Petal width     id            label
7            4.9          2.4           3.3          1.0  id_58  Iris-versicolor
10           5.0          2.0           3.5          1.0  id_61  Iris-versicolor
43           5.0          2.3           3.3          1.0  id_94  Iris-versicolor
48           5.1          2.5           3.0          1.1  id_99  Iris-versicolor
Number of outliers: 4
Outliers for Iris-virginica:
    Sepal length  Sepal width  Petal length  Petal width      id           label
6            4.9          2.5           4.5          1.7  id_107  Iris-virginica
18           7.7          2.6           6.9          2.3  id_119  Iris-virginica
Number of outliers: 2
```

---

9. Analyze your anomaly detection results. Does Mahalanobis distance work better than Euclidean distance? Or the other way around? Or the results are inconclusive?

Ans: Certainly! Let's elaborate further on the comparison between Mahalanobis distance and Euclidean distance for outlier detection in the Iris dataset.

Mahalanobis Distance:

Advantages: - Mahalanobis distance accounts for variable correlations, which is beneficial for datasets where variables are correlated. It provides a more accurate measure of dissimilarity in

such cases. - By considering the covariance between variables, Mahalanobis distance can offer insights into the true distance between data points.

Limitations: - It assumes that the data follows a multivariate normal distribution. If the data significantly deviates from this assumption, Mahalanobis distance might not perform well. - Mahalanobis distance is sensitive to outliers in the dataset, which can skew the estimation of the covariance matrix and affect the accuracy of the distance calculations.

Euclidean Distance:

Advantages: - Euclidean distance is a straightforward and computationally simple measure of dissimilarity between data points. - It works well when variables are not highly correlated and when the data distribution is not multivariate normal.

Limitations: - Euclidean distance does not take into account variable correlations. This can be a disadvantage when dealing with datasets where variables are correlated, as it may provide inaccurate distance measures.

Exploring the Results:

- When using Mahalanobis distance for outlier detection with ideal significance levels such as alpha = 0.05 or 0.1, no outliers were identified. However, to detect the first outlier, a high significance level of alpha = 0.45 had to be used, indicating a high false positive rate.
- In contrast, Euclidean distance identified 8 outliers using a threshold based on quartiles and interquartile range (IQR). This approach is less stringent compared to Mahalanobis distance.
- To determine which method is better, the validity of the outliers detected by Euclidean distance needs to be verified by domain experts. This validation ensures that the identified outliers are meaningful and relevant to the dataset.
- Ultimately, the decision on which method is better depends on the specific characteristics of the dataset and the importance of accounting for variable correlations in outlier detection.

---

10. Evaluate the efficacy of the following (nearest-neighbors-like) classifier. First calculate the mean vector of each species as described at the beginning of this assignment. Next, for each point in the data set (i.e. row in the csv file restricted to the first 4 columns) predict the species of that point to be the species of the mean vector that is the closest to that point. Use Mahalanobis Distance as the proximity measure. Report your results as a 3-by-3 contingency table T, where T[i][j] is the number of instances in which the true species is i and the predicted species is j. Attach a brief interpretation of what this table reveals about how well your classifier has done. Note that we are well aware that this evaluation is being done on the training set; its not meant for drawing conclusions about predictive accuracy, rather only to get an exploratory sense for what the cont9ngency table reveals and what we can conclude from it.

[ ]:
```
import pandas as pd
import numpy as np
from scipy.spatial.distance import mahalanobis
from numpy.linalg import inv

# Read the data
file_path = '/content/drive/My Drive/Pujitha_kallu_PRDM/Assignment-1/iris.csv'
```

```
iris_data = pd.read_csv(file_path)

# Drop unnecessary columns
iris_data = iris_data.drop(columns=["id"])

# Compute the mean vector for each species
mean_vectors = iris_data.groupby('label').mean()

# Initialize the contingency table
contingency_table = np.zeros((3, 3), dtype=int)

# Calculate the Mahalanobis Distance for each data point
for _, row in iris_data.iterrows():
    data_point = row[['Sepal length', 'Sepal width', 'Petal length', 'Petal␣
 ↪width']]
    true_species = row['label']

    min_distance = float('inf')
    predicted_species = None

    # Calculate Mahalanobis Distance to each mean vector
    for species, mean_vector in mean_vectors.iterrows():
        species_cov_matrix = iris_data[iris_data['label'] == species][['Sepal␣
 ↪length', 'Sepal width', 'Petal length', 'Petal width']].cov()
        inv_species_cov_matrix = inv(species_cov_matrix)
        distance = mahalanobis(data_point, mean_vector, inv_species_cov_matrix)

        # Update predicted species if closer
        if distance < min_distance:
            min_distance = distance
            predicted_species = species

    # Update contingency table
    true_index = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'].
 ↪index(true_species)
    predicted_index = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'].
 ↪index(predicted_species)
    contingency_table[true_index][predicted_index] += 1

# Display contingency table
print("Contingency Table:")
print(contingency_table)
```

```
Contingency Table:
[[50  0  0]
 [ 0 47  3]
 [ 0  0 50]]
```

Let's analyze the interpretation of each row:

1. Row 1: [50, 0, 0]
   - True Species: Iris-setosa
   - Interpretation: All 50 instances of Iris-setosa in the training data were correctly classified as Iris-setosa by the classifier. There are no misclassifications for this species. This row indicates perfect accuracy for Iris-setosa.
2. Row 2: [0, 47, 3]
   - True Species: Iris-versicolor
   - Interpretation: Out of 50 instances of Iris-versicolor, 47 were correctly classified as Iris-versicolor. However, the classifier misclassified 3 instances of Iris-versicolor as a different species. This suggests a slight difficulty in distinguishing some Iris-versicolor samples from other species.
3. Row 3: [0, 0, 50]
   - True Species: Iris-virginica
   - Interpretation: Similar to Iris-setosa, all 50 instances of Iris-virginica were correctly classified as Iris-virginica. There are no misclassifications for this species. This row indicates perfect accuracy for Iris-virginica.

References: 1. Class notes :PPT slides->proxmity measures 2. https://www.sciencedirect.com/topics/computer-science/pearson-correlation 3. https://www.datastax.com/guides/what-is-cosine-similarity 4. https://www.sciencedirect.com/science/article/abs/pii/S0031320308002057#:~:text=The%20Mahalanobis%20dis 5. https://www.sciencedirect.com/topics/mathematics/euclidean-distance

---