assignment

October 8, 2023

[1]:
```
# Importing required libraries
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#read iris dataset
df = pd.read_csv('iris.csv')
```

1 Exploratory Data Analysis

[3]:
```
#viewing the schema of the dataset
df.head()
```

[3]:

	Sepal length	Sepal width	Petal length	Petal width	id	label
0	5.1	3.5	1.4	0.2	id_1	Iris-setosa
1	4.9	3.0	1.4	0.2	id_2	Iris-setosa
2	4.7	3.2	1.3	0.2	id_3	Iris-setosa
3	4.6	3.1	1.5	0.2	id_4	Iris-setosa
4	5.0	3.6	1.4	0.2	id_5	Iris-setosa

[4]:
```
#getting information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal length  150 non-null    float64
 1   Sepal width   150 non-null    float64
 2   Petal length  150 non-null    float64
 3   Petal width   150 non-null    float64
 4   id            150 non-null    object
 5   label         150 non-null    object
dtypes: float64(4), object(2)
memory usage: 7.2+ KB
```

Dataframe contains 4 numerical variables and 1 categorical variable(label). There are no missing values in the dataframe also.

[5]:
```
#getting statistical insight
df.describe()
```

[5]:

	Sepal length	Sepal width	Petal length	Petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Maximum sepal length of an Iris flower is 7.9cm while the minimum sepal length is 4.3cm.

[6]:
```
#checking the distribution of each species
df['label'].value_counts()
```

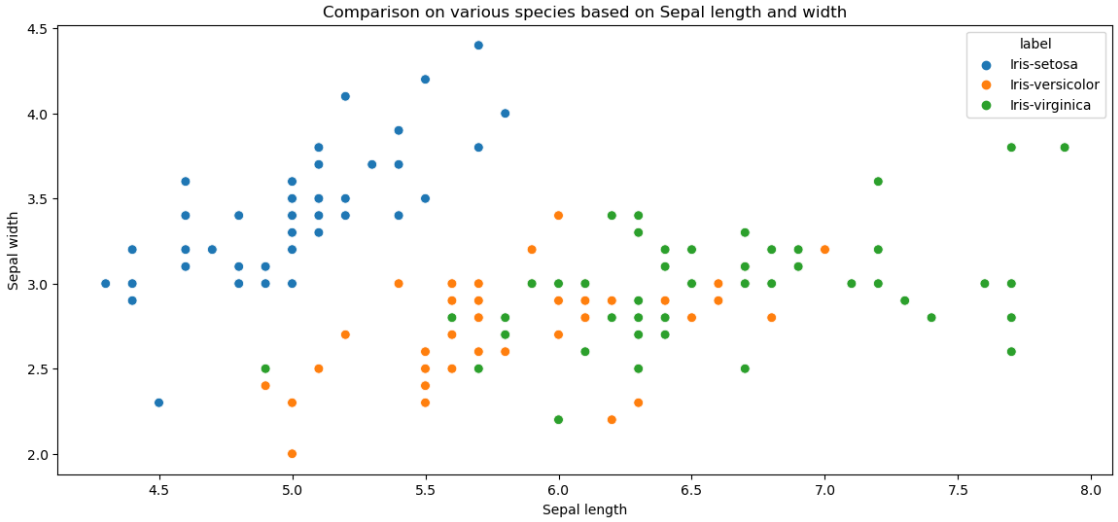
[6]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: label, dtype: int64
```

Each species of the iris flowers has 50 samples each. This means our data is balanced.

[7]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

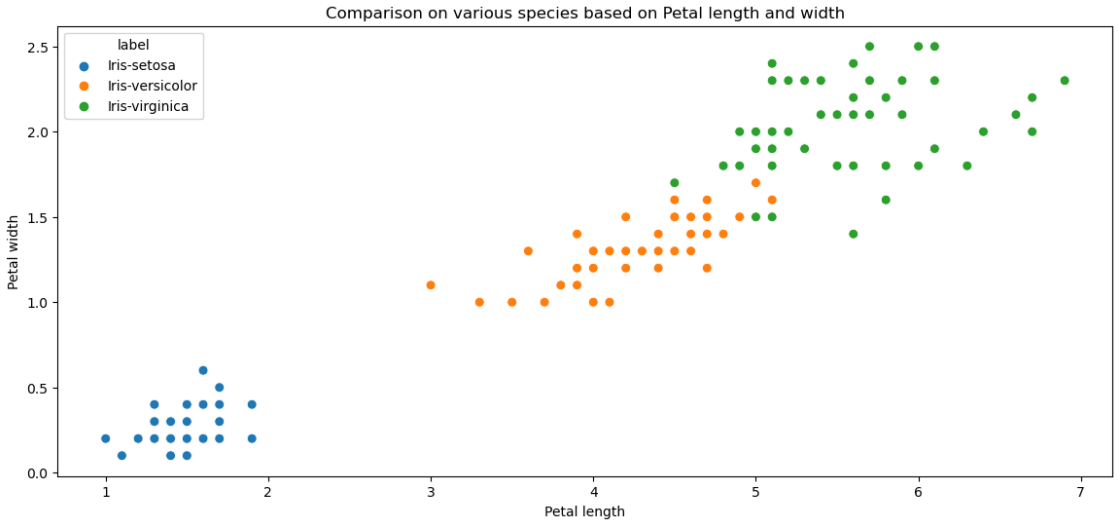
[8]:
```
# Comparison between various species based on sepal length and width
plt.figure(figsize=(14,6))
plt.title('Comparison on various species based on Sepal length and width')
sns.scatterplot(x=df['Sepal length'], y=df['Sepal width'], hue=df['label'],␣
 ↪s=50)
```

[8]:
```
<Axes: title={'center': 'Comparison on various species based on Sepal length and
width'}, xlabel='Sepal length', ylabel='Sepal width'>
```



[9]:
```
# Comparison between various species based on petal length and width
plt.figure(figsize=(14,6))
plt.title('Comparison on various species based on Petal length and width')
sns.scatterplot(x=df['Petal length'], y=df['Petal width'], hue=df['label'],␣
↪s=50)
```

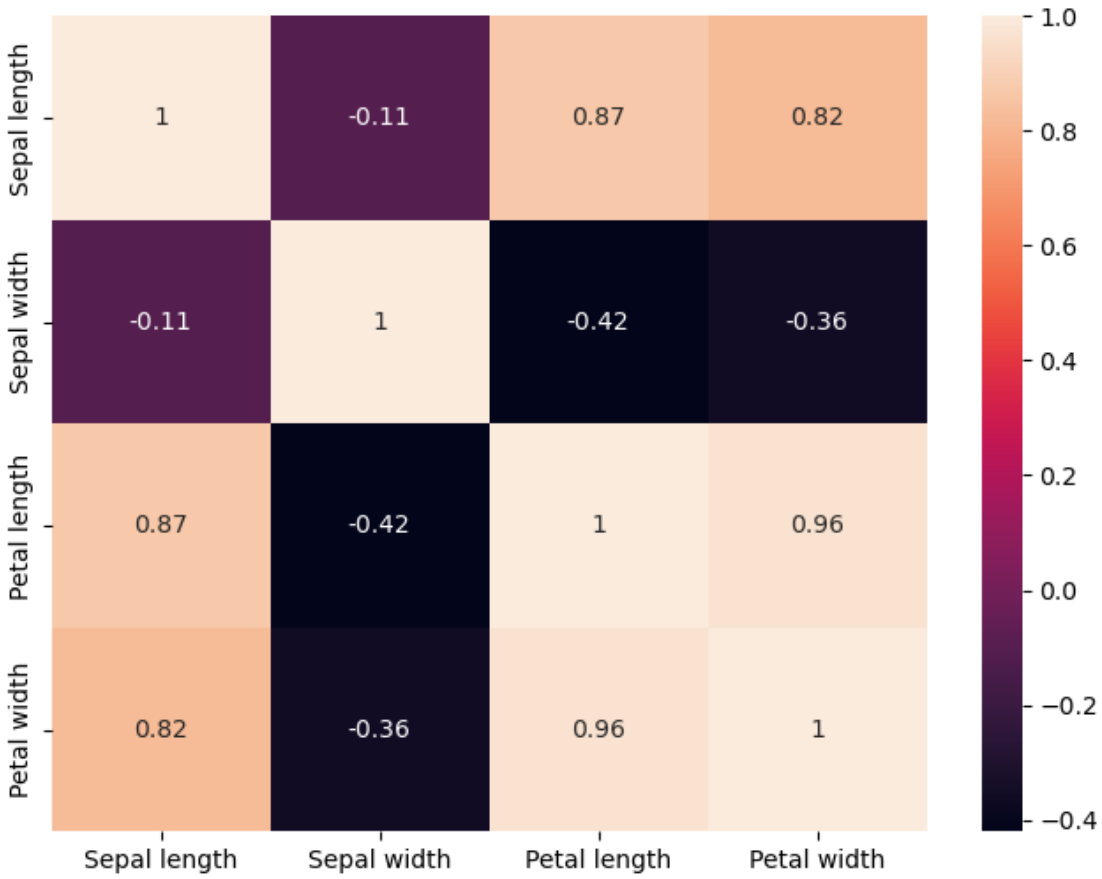
[9]:
```
<Axes: title={'center': 'Comparison on various species based on Petal length and
width'}, xlabel='Petal length', ylabel='Petal width'>
```



[10]:
```
#checking correlation
plt.figure(figsize=(8,6))
iris_corr = df.corr()
```

```
sns.heatmap(iris_corr, annot=True)
```

[10]: <Axes: >

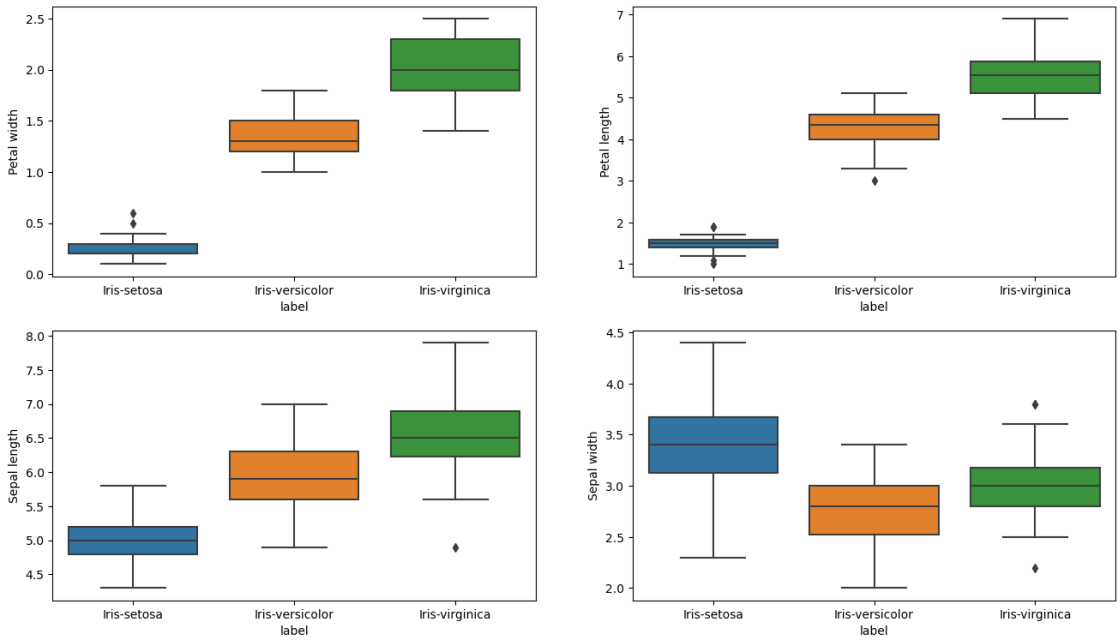


Petal length and petal width are highly correlated, as well as the Petal width and sepal length have a good correlation.

[11]:
```
#checking distribution of data points
fig, axes = plt.subplots(2,2, figsize=(16,9))

sns.boxplot(x='label', y='Petal width', data=df, orient='v', ax=axes[0,0])
sns.boxplot(x='label', y='Petal length', data=df, orient='v', ax=axes[0,1])
sns.boxplot(x='label', y='Sepal length', data=df, orient='v', ax=axes[1,0])
sns.boxplot(x='label', y='Sepal width', data=df, orient='v', ax=axes[1,1])
```

[11]: <Axes: xlabel='label', ylabel='Sepal width'>



1. Setosa species have some outliers in the Petal length and width(i.e data points that differs significantly from the majority of the data taken from a sample or population)
2. Setosa species have smaller features(dimensions) and are less distributed
3. Versicolor species are distributed in an average manner and average features(dimensions).
4. Virginica species are highly distributed with a large number of values and features(dimensions).

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

[12]:
```
#Calculating the mean vector of all species
sp_vector = {}
for species in df['label'].unique():
    species_data = df[df['label'] == species]
    mean_vector = species_data.iloc[:, 0:6].mean().values
    sp_vector[species] = mean_vector
```

[13]:
```
from itertools import combinations

#calculating the euclidean distance
distances = {}
sp_combination = list(combinations(df['label'].unique(), 2))
for species1, species2 in sp_combination:
    vector1 = sp_vector[species1]
    vector2 = sp_vector[species2]
```

```
    distance = np.linalg.norm(vector1 - vector2)
    distances[(species1, species2)] = distance
```

[14]:
```
distance_table = pd.DataFrame({'Species Pair': sp_combination, 'Euclidean␣
↪Distance': list(distances.values())})
distance_table
```

[14]:

	Species Pair	Euclidean Distance
0	(Iris-setosa, Iris-versicolor)	3.205175
1	(Iris-setosa, Iris-virginica)	4.752592
2	(Iris-versicolor, Iris-virginica)	1.620489

3 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).

[15]:
```
#standardizing the column vectors
species_data = {}
for species in df['label'].unique():
    species_data[species] = df[df['label'] == species].iloc[:, 0:4]



standardized_data = {}
for species, data in species_data.items():
    print(species)
    mean = data.mean()
    std_dev = data.std()
    standardized_data[species] = (data - mean) / std_dev


#Calculating the mean vector of all species
sp_vectors = {}
for species, data in standardized_data.items():
    mean_vector = data.mean().values
    sp_vectors[species] = mean_vector


#calculating the euclidean distance
distances = {}
sp_combination = list(combinations(df['label'].unique(), 2))
for species1, species2 in sp_combination:
    vector1 = sp_vectors[species1]
    vector2 = sp_vectors[species2]
```

```
    distance = np.linalg.norm(vector1 - vector2)
    distances[(species1, species2)] = distance



distance_table = pd.DataFrame({'Species Pair': sp_combination, 'Euclidean␣
 ↪Distance': list(distances.values())})
distance_table
```

```
Iris-setosa
Iris-versicolor
Iris-virginica
```

[15]:

	Species Pair	Euclidean Distance
0	(Iris-setosa, Iris-versicolor)	1.477819e-15
1	(Iris-setosa, Iris-virginica)	3.700422e-15
2	(Iris-versicolor, Iris-virginica)	3.612884e-15

4 Same as 1 except use Mahalanobis Distance.

[16]:
```
from scipy.spatial import distance
```

[17]:
```
species_data = {}
for species in df['label'].unique():
    species_data[species] = df[df['label'] == species].iloc[:,0:4]


species_mean_vectors = {}
species_cov_matrices = {}
for species, data in species_data.items():


    numeric_data = data.apply(pd.to_numeric, errors='coerce').dropna()

    mean_vector = numeric_data.mean().values
    cov_matrix = np.cov(numeric_data, rowvar=False)

    species_mean_vectors[species] = mean_vector
    species_cov_matrices[species] = cov_matrix

#calculating the mahalanobis distance
distances = {}
sp_combination = list(combinations(df['label'].unique(), 2))
for species1, species2 in sp_combination:
    mean_vector1 = species_mean_vectors[species1]
    mean_vector2 = species_mean_vectors[species2]
```

```
    cov_matrix1 = species_cov_matrices[species1]
    cov_matrix2 = species_cov_matrices[species2]


    mahalanobis_dist = distance.mahalanobis(mean_vector1, mean_vector2, np.
 ↪linalg.inv((cov_matrix1 + cov_matrix2) / 2))
    distances[(species1, species2)] = mahalanobis_dist


distance_table = pd.DataFrame({'Species Pair': sp_combination, 'Mahalanobis␣
 ↪Distance': list(distances.values())})

distance_table
```

[17]:

	Species Pair	Mahalanobis Distance
0	(Iris-setosa, Iris-versicolor)	10.193677
1	(Iris-setosa, Iris-virginica)	13.994860
2	(Iris-versicolor, Iris-virginica)	3.770794

5 Same as 1 except use Cosine Similarity.

[18]:
```
from sklearn.metrics.pairwise import cosine_similarity
species_data = {}
for species in df['label'].unique():
    species_data[species] = df[df['label'] == species].iloc[:, 0:4]


species_mean_vectors = {}
for species, data in species_data.items():
    mean_vector = data.mean().values
    species_mean_vectors[species] = mean_vector

#calculating the cosine similarity
similarities = {}
sp_combination = list(combinations(df['label'].unique(), 2))
for species1, species2 in sp_combination:
    mean_vector1 = species_mean_vectors[species1].reshape(1, -1)
    mean_vector2 = species_mean_vectors[species2].reshape(1, -1)


    similarity = cosine_similarity(mean_vector1, mean_vector2)[0][0]
    similarities[(species1, species2)] = similarity


similarity_table = pd.DataFrame({'Species Pair': sp_combination, 'Cosine␣
 ↪Similarity': list(similarities.values())})
```

```
similarity_table
```

[18]:

	Species Pair	Cosine Similarity
0	(Iris-setosa, Iris-versicolor)	0.924848
1	(Iris-setosa, Iris-virginica)	0.888438
2	(Iris-versicolor, Iris-virginica)	0.995713

6 Same as 1 except use Pearson Correlation.

[19]:
```
species_mean_vectors = {}
for species, data in species_data.items():
    mean_vector = data.mean().values
    species_mean_vectors[species] = mean_vector

#calculating the pearson ccorrelation
correlations = {}
sp_combination = list(combinations(df['label'].unique(), 2))
for species1, species2 in sp_combination:
    mean_vector1 = species_mean_vectors[species1]
    mean_vector2 = species_mean_vectors[species2]


    correlation = np.corrcoef(mean_vector1, mean_vector2)[0, 1]
    correlations[(species1, species2)] = correlation


correlation_table = pd.DataFrame({'Species Pair': sp_combination, 'Pearson␣
↪Correlation': list(correlations.values())})
correlation_table
```

[19]:

	Species Pair	Pearson Correlation
0	(Iris-setosa, Iris-versicolor)	0.763854
1	(Iris-setosa, Iris-virginica)	0.618438
2	(Iris-versicolor, Iris-virginica)	0.979489

7 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.

1. Euclidean Distance: Pros: Euclidean distance is like a straightforward ruler to measure the distance between two points. Smaller numbers mean things are closer or more similar. Cons: It gets confused when the ruler is in different units, like one thing measured in inches and another in feet. It also doesn't think about how things relate to each other or if there are too many things to measure.

2. Z-Score: Pros: Z-score makes different things (like apples and oranges) more easily comparable, even if they started with different sizes.It's like putting all your fruits on the same healthy-eating scale. It's not easily fooled by weird fruits that are way bigger or smaller than the rest.It helps to handle fruits that are a little strange (outliers) without making your scale go crazy. Cons: After using Z-score, your attributes are no longer in their original scales, which can be confusing if you're used to thinking about them in their original measuring convention (cm, inches, foot). Sometimes, Z-score works best if your attributes act like they're supposed to (follow a normal pattern). If your fruits are very different from that, it might not be the best choice.

3. Mahalanobis Distance: Pros: Mahalanobis distance is like having a special ruler that takes into account how different things relate to each other. It's great when your data has things that work together or influence each other. It doesn't get confused when you're measuring things that are in different units, like measuring heights in feet and weights in pounds. It's like having a ruler that can handle both. Cons: Using Mahalanobis distance can be a bit like doing a hard math problem because it involves some tricky calculations, like turning your data into a special shape. It works best when your data behaves in a very specific way (like a normal pattern), but sometimes real-world data doesn't act that way, so it might not be the best choice for all situations.

4. Cosine Similarity: Pros: Measures how things point in different directions, especially useful when there are lots of dimensions. Doesn't care about how long things are, only where they're pointing. Good for things like words. Cons: Doesn't work if things can go backward or if negative relationships are important. Ignores the "size" of things.

5. Pearson Correlation: Pros: Pearson correlation checks if things go up or down together in a straight line. It tells us how strong and in which direction this connection is. It's great for comparing things, even if they use different measurements. Cons: This only understands straight-line connections, so it might miss some wavy relationships. It gets easily confused by strange data points, especially when there isn't much data. And it assumes that things always act like they're "normal," which isn't always true in real life.

8 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.

[20]:
```
from scipy.stats import chi2

species_data = {}
for species in df['label'].unique():
    species_data[species] = df[df['label'] == species].iloc[:, 0:4]
```

```
def detect_outliers(data, significance_level=0.55):

    mean_vector = data.mean().values
    cov_matrix = np.cov(data, rowvar=False)


    degrees_of_freedom = len(data.columns)
    critical_value = chi2.ppf(significance_level, df=degrees_of_freedom)

    is_outlier = []

    for i in range(len(data)):
        data_point = data.iloc[i, :].values
        mahalanobis_dist = np.sqrt((data_point - mean_vector).T @ np.linalg.
 ↪inv(cov_matrix) @ (data_point - mean_vector))


        is_outlier.append(mahalanobis_dist > critical_value)

    return is_outlier

outliers_by_species = {}
for species, data in species_data.items():
    is_outlier = detect_outliers(data)
    outliers_by_species[species] = is_outlier

for species, is_outlier in outliers_by_species.items():
    print(f"Outliers in {species}: {sum(is_outlier)}")
```

```
Outliers in Iris-setosa: 0
Outliers in Iris-versicolor: 0
Outliers in Iris-virginica: 1
```

9 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.

[21]:
```
species_data = {}
for species in df['label'].unique():
    species_data[species] = df[df['label'] == species].iloc[:, 0:4]

def detect_outliers(data, multiplier=1.5):

    mean_vector = data.mean().values
```

```
    euclidean_distances = np.linalg.norm(data - mean_vector, axis=1)

    Q1 = np.percentile(euclidean_distances, 25)
    Q3 = np.percentile(euclidean_distances, 75)
    #calculating the interquartile range
    IQR = Q3 - Q1
    threshold_upper = Q3 + multiplier * IQR
    threshold_lower = Q1 - multiplier * IQR

    is_outlier = (euclidean_distances > threshold_upper) | (euclidean_distances␣
 ↪< threshold_lower)

    return is_outlier

outliers_by_species = {}
for species, data in species_data.items():
    is_outlier = detect_outliers(data)
    outliers_by_species[species] = is_outlier

for species, is_outlier in outliers_by_species.items():
    print(f"Outliers in {species}: {sum(is_outlier)}")
```

```
Outliers in Iris-setosa: 2
Outliers in Iris-versicolor: 4
Outliers in Iris-virginica: 2
```

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

Fewer outliers were observed in case of Mahalanobis distance when compared to the number of outliers detected by Euclidean distance measure. The choice between Euclidean and Mahalanobis distance depends on the specific characteristics of your data and your objectives. Mahalanobis distance takes into account the covariance (correlation) structure of the data. It is also less sensitive to differences in variable scales, making it more robust when dealing with data with varying units or scales. Euclidean distance is better to use when the variables in a dataset are not correlated with each other, and there is no meaningful relationship between them. Euclidean distance is also a simpler and more intuitive approach for outlier detection which is less computationally expensive as well.

11 **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 contingency table reveals and what we can conclude from it.**

[22]:
```
species_mean_vectors = {}
for species in df['label'].unique():
    species_mean_vectors[species] = df[df['label'] == species].iloc[:,0:4].
 ↪mean().values

def mahalanobis_distance(x, y, cov_matrix):
    delta = x - y
    return np.sqrt(delta.T @ np.linalg.inv(cov_matrix) @ delta)

species_labels = df['label'].unique()
contingency_table = np.zeros((len(species_labels), len(species_labels)),␣
 ↪dtype=int)

for index, row in df.iterrows():
    true_species = row['label']
    data_point = row.iloc[0:4].values

    min_distance = float('inf')
    predicted_species = None
    for species, mean_vector in species_mean_vectors.items():
        cov_matrix = np.cov(df[df['label'] == species].iloc[:, 0:4],␣
 ↪rowvar=False)
        mahalanobis_dist = mahalanobis_distance(data_point, mean_vector,␣
 ↪cov_matrix)
        if mahalanobis_dist < min_distance:
```

```
            min_distance = mahalanobis_dist
            predicted_species = species

    true_species_index = np.where(species_labels == true_species)[0][0]
    predicted_species_index = np.where(species_labels ==␣
 ↪predicted_species)[0][0]
    contingency_table[true_species_index][predicted_species_index] += 1

print("Contingency Table : ")
contingency_table
```

```
Contingency Table :
```

[22]:
```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  0, 50]])
```

[ ]:
```

```