**Iris does not reduce dimensionality**

1. **from** scipy.spatial.distance **import** mahalanobis
2. **from** sklearn.datasets **import** load\_iris
3. **from** sklearn.model\_selection **import** train\_test\_split
4. **import** numpy as np
5. # Load iris dataset
6. iris = load\_iris()
7. # Split dataset into training and testing datasets
8. X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.3, random\_state=1)
9. # Compute mean vectors and covariance matrix for original data
10. mean\_vectors = []
11. **for** i **in** range(3):
12. mean\_vectors.append(np.mean(X\_train[y\_train==i], axis=0))
13. cov\_matrix = np.cov(X\_train.T)
14. #  compute the minimum Mahalanobis distance classifier
15. **def** mmd(x):
16. distances = []
17. **for** i **in** range(3):
18. distance = mahalanobis(x, mean\_vectors[i], np.linalg.inv(cov\_matrix))
19. distances.append(distance)
20. **return** np.argmin(distances)
21. # Compute classification accuracy for minimum Mahalanobis distance classifier on original data
22. y\_pred = [mmd(x) **for** x **in** X\_test]
23. accuracy = np.sum(y\_pred == y\_test) / len(y\_test)
24. **print**("Classification accuracy (no dimensionality reduction): {:.2f}%".format(accuracy\*100))

**Breast cancer without dimension reduction**

1. **from** sklearn.datasets **import** load\_breast\_cancer
2. **from** sklearn.model\_selection **import** train\_test\_split
3. **from** scipy.spatial.distance **import** mahalanobis
4. **import** numpy as np
5. # Load the breast cancer dataset
6. breast\_cancer = load\_breast\_cancer()
7. X = breast\_cancer.data
8. y = breast\_cancer.target
9. # Split the dataset into training and testing sets
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)
11. # Compute the mean and covariance of each class in the training set
12. means = np.zeros((len(np.unique(y\_train)), X\_train.shape[1]))
13. covs = np.zeros((len(np.unique(y\_train)), X\_train.shape[1], X\_train.shape[1]))
14. **for** i **in** range(len(np.unique(y\_train))):
15. means[i] = np.mean(X\_train[y\_train == i], axis=0)
16. covs[i] = np.cov(X\_train[y\_train == i].T)
17. # Compute the inverse of each class covariance matrix
18. invcovs = np.zeros((len(np.unique(y\_train)), X\_train.shape[1], X\_train.shape[1]))
19. **for** i **in** range(len(np.unique(y\_train))):
20. invcovs[i] = np.linalg.inv(covs[i])
21. # Compute the minimum Mahalanobis distance for each test sample
22. y\_pred = np.zeros(y\_test.shape)
23. **for** i **in** range(X\_test.shape[0]):
24. distances = np.zeros(len(np.unique(y\_train)))
25. **for** j **in** range(len(np.unique(y\_train))):
26. distances[j] = mahalanobis(X\_test[i], means[j], invcovs[j])
27. y\_pred[i] = np.argmin(distances)
28. # Compute the classification accuracy
29. accuracy = np.sum(y\_pred == y\_test) / len(y\_test)
30. **print**(f"Classification accuracy using minimum Mahalanobis distance: {accuracy}")

**Iris using lda dimensionality reduction**

1. # Importing the necessary libraries and loading the dataset
2. **import** numpy as np
3. **import** pandas as pd
4. **from** sklearn.datasets **import** load\_iris
5. # Load the dataset
6. iris = load\_iris()
7. # Create a dataframe
8. df = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],
9. columns= iris['feature\_names'] + ['target'])
10. # Split the dataset into training and testing datasets
11. **from** sklearn.model\_selection **import** train\_test\_split
12. # Split the dataset into training and testing datasets
13. X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.iloc[:, :-1], df.iloc[:, -1], test\_size=0.3, random\_state=42)
14. # Data normalization
15. **from** sklearn.preprocessing **import** StandardScaler
16. # Perform data normalization
17. scaler = StandardScaler()
18. X\_train = scaler.fit\_transform(X\_train)
19. X\_test = scaler.transform(X\_test)
20. **from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis as LDA
21. # Compute LDA
22. lda = LDA(n\_components=2)
23. X\_train\_lda = lda.fit\_transform(X\_train, y\_train)
24. X\_test\_lda = lda.transform(X\_test)
25. # Train the minimum Mahalanobis distance classifier
26. **from** scipy.spatial.distance **import** mahalanobis
27. # Compute mean vectors and covariance matrix for LDA-transformed data
28. mean\_vectors\_lda = []
29. **for** i **in** range(3):
30. mean\_vectors\_lda.append(np.mean(X\_train\_lda[y\_train==i], axis=0))
31. cov\_matrix\_lda = np.cov(X\_train\_lda.T)
32. # Define function to compute the minimum Mahalanobis distance classifier
33. **def** mmd\_lda(x):
34. distances = []
35. **for** i **in** range(3):
36. distance = mahalanobis(x, mean\_vectors\_lda[i], np.linalg.inv(cov\_matrix\_lda))
37. distances.append(distance)
38. **return** np.argmin(distances)
39. # Compute classification accuracy for minimum Mahalanobis distance classifier on LDA-transformed data
40. y\_pred\_lda = [mmd\_lda(x) **for** x **in** X\_test\_lda]
41. accuracy\_lda = np.sum(y\_pred\_lda == y\_test) / len(y\_test)
42. **print**("Classification accuracy (LDA): {:.2f}%".format(accuracy\_lda\*100))

**Breast cancer using lda dimensionality reduction**

1. **from** sklearn.datasets **import** load\_breast\_cancer
2. **from** sklearn.model\_selection **import** train\_test\_split
3. **from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis
4. **from** scipy.spatial.distance **import** mahalanobis
5. **import** numpy as np
6. # Load the breast cancer dataset
7. breast\_cancer = load\_breast\_cancer()
8. X = breast\_cancer.data
9. y = breast\_cancer.target
10. # Split the dataset into training and testing sets
11. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)
12. # Apply LDA for dimensionality reduction
13. lda = LinearDiscriminantAnalysis(n\_components=1)
14. X\_train\_lda = lda.fit\_transform(X\_train, y\_train)
15. X\_test\_lda = lda.transform(X\_test)
16. # Compute the mean and covariance of each class in the training set
17. means = np.zeros((len(np.unique(y\_train)), X\_train\_lda.shape[1]))
18. covs = np.zeros((len(np.unique(y\_train)), X\_train\_lda.shape[1], X\_train\_lda.shape[1]))
19. **for** i **in** range(len(np.unique(y\_train))):
20. means[i] = np.mean(X\_train\_lda[y\_train == i], axis=0)
21. covs[i] = np.cov(X\_train\_lda[y\_train == i].T)
22. # Compute the inverse of each class covariance matrix
23. invcovs = np.zeros((len(np.unique(y\_train)), X\_train\_lda.shape[1], X\_train\_lda.shape[1]))
24. **for** i **in** range(len(np.unique(y\_train))):
25. invcovs[i] = np.linalg.inv(covs[i])
26. # Compute the minimum Mahalanobis distance for each test sample
27. y\_pred = np.zeros(y\_test.shape)
28. **for** i **in** range(X\_test\_lda.shape[0]):
29. distances = np.zeros(len(np.unique(y\_train)))
30. **for** j **in** range(len(np.unique(y\_train))):
31. distances[j] = mahalanobis(X\_test\_lda[i], means[j], invcovs[j])
32. y\_pred[i] = np.argmin(distances)
33. # Compute the classification accuracy
34. accuracy = np.sum(y\_pred == y\_test) / len(y\_test)
35. **print**(f"Classification accuracy after LDA: {accuracy}")

**Iris using pca dimensionality reduction**

1. # Importing the necessary libraries and loading the dataset
2. **import** numpy as np
3. **import** pandas as pd
4. **from** sklearn.datasets **import** load\_iris
5. # Load the dataset
6. iris = load\_iris()
7. # Create a dataframe
8. df = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],
9. columns= iris['feature\_names'] + ['target'])
10. # Split the dataset into training and testing datasets
11. **from** sklearn.model\_selection **import** train\_test\_split
12. # Split the dataset into training and testing datasets
13. X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.iloc[:, :-1], df.iloc[:, -1], test\_size=0.3, random\_state=42)
14. # Data normalization
15. **from** sklearn.preprocessing **import** StandardScaler
16. # Perform data normalization
17. scaler = StandardScaler()
18. X\_train = scaler.fit\_transform(X\_train)
19. X\_test = scaler.transform(X\_test)
20. # Dimensionality reduction
21. **from** sklearn.decomposition **import** PCA
22. # Perform PCA
23. pca = PCA(n\_components=2)
24. X\_train\_pca = pca.fit\_transform(X\_train)
25. X\_test\_pca = pca.transform(X\_test)
26. # Train the minimum Mahalanobis distance classifier
27. **from** scipy.spatial.distance **import** mahalanobis
28. # Compute mean vectors and covariance matrix for PCA- data
29. mean\_vectors\_pca = []
30. **for** i **in** range(3):
31. mean\_vectors\_pca.append(np.mean(X\_train\_pca[y\_train==i], axis=0))
32. cov\_matrix\_pca = np.cov(X\_train\_pca.T)
33. # Define function to compute the minimum Mahalanobis distance classifier
34. **def** mmd\_pca(x):
35. distances = []
36. **for** i **in** range(3):
37. distance = mahalanobis(x, mean\_vectors\_pca[i], np.linalg.inv(cov\_matrix\_pca))
38. distances.append(distance)
39. **return** np.argmin(distances)
40. # Compute classification accuracy for minimum Mahalanobis distance classifier on PCA-transformed data
41. y\_pred\_pca = [mmd\_pca(x) **for** x **in** X\_test\_pca]
42. accuracy\_pca = np.sum(y\_pred\_pca == y\_test) / len(y\_test)
43. **print**("Classification accuracy (PCA): {:.2f}%".format(accuracy\_pca\*100))

**Breast cancer using pca dimensionality reduction**

1. **from** sklearn.datasets **import** load\_breast\_cancer
2. **from** sklearn.model\_selection **import** train\_test\_split
3. **from** sklearn.decomposition **import** PCA
4. **from** scipy.spatial.distance **import** mahalanobis
5. **import** numpy as np
6. breast\_cancer = load\_breast\_cancer()
7. X = breast\_cancer.data
8. y = breast\_cancer.target
9. # Split the dataset into training and testing sets
10. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)
11. # Perform PCA for dimensionality reduction
12. pca = PCA(n\_components=20)
13. X\_train\_pca = pca.fit\_transform(X\_train)
14. X\_test\_pca = pca.transform(X\_test)
15. # Compute the mean and covariance of each class in the training set after PCA
16. means = np.zeros((len(np.unique(y\_train)), X\_train\_pca.shape[1]))
17. covs = np.zeros((len(np.unique(y\_train)), X\_train\_pca.shape[1], X\_train\_pca.shape[1]))
18. **for** i **in** range(len(np.unique(y\_train))):
19. means[i] = np.mean(X\_train\_pca[y\_train == i], axis=0)
20. covs[i] = np.cov(X\_train\_pca[y\_train == i].T)
21. # Compute the inverse of each class covariance matrix
22. invcovs = np.zeros((len(np.unique(y\_train)), X\_train\_pca.shape[1], X\_train\_pca.shape[1]))
23. **for** i **in** range(len(np.unique(y\_train))):
24. invcovs[i] = np.linalg.inv(covs[i])
25. # Compute the minimum Mahalanobis distance for each test sample after PCA
26. y\_pred = np.zeros(y\_test.shape)
27. **for** i **in** range(X\_test\_pca.shape[0]):
28. distances = np.zeros(len(np.unique(y\_train)))
29. **for** j **in** range(len(np.unique(y\_train))):
30. distances[j] = mahalanobis(X\_test\_pca[i], means[j], invcovs[j])
31. y\_pred[i] = np.argmin(distances)
32. # Compute the classification accuracy after PCA
33. accuracy = np.sum(y\_pred == y\_test) / len(y\_test)
34. **print**(f"Classification accuracy after PCA using minimum Mahalanobis distance: {accuracy}")

**View the scatterplot of each principal component**

1. #Import required libraries
2. **import** numpy as np
3. **import** pandas as pd
4. **import** seaborn as sns
5. **import** matplotlib.pyplot as plt
6. **from** sklearn.decomposition **import** PCA
7. # read related dataset
8. data = sns.load\_dataset('iris')
9. data.head(8)
10. #Simple processing of the data to see the relationship between the dimensions
11. values = data.iloc[:, :4] #Read the first 4 columns of data
12. correlation = values.corr() #Correlation coefficient between columns
13. fig, ax = plt.subplots(figsize=(12, 10))
14. sns.heatmap(correlation, annot=True, annot\_kws={'size':16}, cmap='Reds', square=True, ax=ax) #heatmap
15. sns.pairplot(data, hue='species') #Scatter plot
16. plt.show()
17. #Analyze the cumulative results of each principal component of the data set
18. pca = PCA(n\_components=4)
19. pc = pca.fit\_transform(values)
20. **print**("explained variance ratio: %s" % pca.explained\_variance\_ratio\_) #Output the proportion of each principal component
21. plt.plot(range(1, 5), np.cumsum(pca.explained\_variance\_ratio\_)) # draw principal component cumulative proportion map
22. plt.scatter(range(1,5),np.cumsum(pca.explained\_variance\_ratio\_))
23. plt.xlim(0, 5)
24. plt.ylim(0.9, 1.02)
25. plt.xlabel("number of components")
26. plt.ylabel("cumulative explained variance")
27. plt.show()
28. #Select the first two principal components to analyze
29. pca1 = PCA(n\_components=2)
30. pc1 = pca1.fit\_transform(values)
31. pc1\_df = pd.DataFrame(pc1, columns=['pc\_1', 'pc\_2'])
32. pc1\_df['species'] = data['species']
33. pc1\_df.head(8)
34. **print**(pca1.components\_)
35. #Reduce the data from four dimensions to two dimensions and plot
36. setosa = pc1\_df[pc1\_df['species']=='setosa']
37. virginica = pc1\_df[pc1\_df['species']=='virginica']
38. versicolor = pc1\_df[pc1\_df['species']=='versicolor']
39. fig, ax = plt.subplots(figsize=(10, 8))
40. plt.scatter(setosa['pc\_1'], setosa['pc\_2'], alpha=0.7, color = 'red', label='Setosa') #Draw the scatterplot of Setosa
41. plt.scatter(virginica['pc\_1'], virginica['pc\_2'], alpha=0.7, color = 'green', label='Virginica')
42. plt.scatter(versicolor['pc\_1'], versicolor['pc\_2'], alpha=0.7, color = 'blue',  label='Versicolor')
43. plt.legend(loc='best')
44. plt.xlabel('principal component 1')
45. plt.ylabel('principal component 2')
46. plt.show()