Machine Learning Algorithms

Exercise 3

Name: Chethan Kashyap Bangalore Muralidhara

Date: 30/03/2023

Completed Exercises: 1,2,3,4,5,6(All)

Solutions:

1. Code in Python

In [5]:
```
###Question 1

import pandas as pd
from sklearn.naive_bayes import GaussianNB

# Load the data from the observation table
data = pd.DataFrame({
    'Day': ['weekday', 'weekday', 'weekday', 'weekday', 'saturday', 'weekday', 'holiday', 'sunday', 'weekday', 'weekday', 'saturc
    'Season': ['spring', 'winter', 'winter', 'winter', 'summer', 'autumn', 'summer', 'summer', 'winter', 'summer', 'spring', 'sum
    'Wind': ['none', 'none', 'none', 'high', 'normal', 'normal', 'high', 'normal', 'high', 'none', 'high', 'high', 'normal', 'hig
    'Rain': ['none', 'slight', 'slight', 'heavy', 'none', 'none', 'slight', 'none', 'heavy', 'slight', 'heavy', 'slight', 'none',
    'Class': ['on time', 'on time', 'on time', 'late', 'on time', 'very late', 'on time', 'on time', 'very late', 'on time', 'car
})

# Convert categorical variables to numerical values using one-hot encoding
data = pd.get_dummies(data, columns=['Day', 'Season', 'Wind', 'Rain'])

# Split the data into features and target
X = data.drop('Class', axis=1)
y = data['Class']

# Create and fit the Naive Bayes classifier
classifier = GaussianNB()
classifier.fit(X, y)
```

```
# Define the case to predict
case = pd.DataFrame({
    'Day_weekday': [1],
    'Day_saturday': [0],
    'Day_sunday': [0],
    'Day_holiday': [0],
    'Season_spring': [0],
    'Season_winter': [1],
    'Season_summer': [0],
    'Season_autumn': [0],
    'Wind_none': [0],
    'Wind_normal': [0],
    'Wind_high': [1],
    'Rain_none': [0],
    'Rain_slight': [0],
    'Rain_heavy': [1]
})

# Predict the class for the given case
prediction = classifier.predict(case)

# Print the predicted class
print(prediction)
```

```
[very late]
```

2.

In [10]:
```python
###Question 2

import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal

# Load the data from the Excel file
data = pd.read_excel('data3.xlsx', header=None)

# Split the data into the two classes
c1_data = data.iloc[:100]
c2_data = data.iloc[100:200]

# Calculate the mean and covariance for each class
c1_mean = np.mean(c1_data, axis=0)
c1_cov = np.cov(c1_data.T)
c2_mean = np.mean(c2_data, axis=0)
c2_cov = np.cov(c2_data.T)

# Construct the 2D Gaussian models for each class
c1_model = multivariate_normal(c1_mean, c1_cov)
c2_model = multivariate_normal(c2_mean, c2_cov)

# Classify the remaining 20 points
test_data = data.iloc[200:]
predictions = []
for index, row in test_data.iterrows():
    c1_prob = c1_model.pdf(row)
    c2_prob = c2_model.pdf(row)
    if c1_prob > c2_prob:
        predictions.append(1)
    else:
        predictions.append(2)
```

```python
# Calculate the accuracy, specificity, and sensitivity of the classifier
true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy = np.mean(np.array(predictions) == np.array(true_labels))
c1_true = np.sum(np.array(predictions)[:10] == 1)
c1_false = np.sum(np.array(predictions)[:10] == 2)
c2_true = np.sum(np.array(predictions)[10:] == 2)
c2_false = np.sum(np.array(predictions)[10:] == 1)
specificity = c2_true / (c2_true + c2_false)
sensitivity = c1_true / (c1_true + c1_false)

print("Accuracy:", accuracy)
print("Specificity:", specificity)
print("Sensitivity:", sensitivity)
```

```
Accuracy: 1.0
Specificity: 1.0
Sensitivity: 1.0
```

3.

```
In [9]: ###Question 3

from scipy.spatial.distance import mahalanobis

# Classify the remaining 20 points using Mahalanobis distance
test_data = data.iloc[200:]
predictions = []
for index, row in test_data.iterrows():
    c1_distance = mahalanobis(row, c1_mean, np.linalg.inv(c1_cov))
    c2_distance = mahalanobis(row, c2_mean, np.linalg.inv(c2_cov))
    if c1_distance < c2_distance:
        predictions.append(1)
    else:
        predictions.append(2)

# Calculate the accuracy, specificity, and sensitivity of the classifier
true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy = np.mean(np.array(predictions) == np.array(true_labels))
c1_true = np.sum(np.array(predictions)[:10] == 1)
c1_false = np.sum(np.array(predictions)[:10] == 2)
c2_true = np.sum(np.array(predictions)[10:] == 2)
c2_false = np.sum(np.array(predictions)[10:] == 1)
specificity = c2_true / (c2_true + c2_false)
sensitivity = c1_true / (c1_true + c1_false)

print("Accuracy:", accuracy)
print("Specificity:", specificity)
print("Sensitivity:", sensitivity)
```

```
Accuracy: 1.0
Specificity: 1.0
Sensitivity: 1.0
```

4.

```
In [8]: ###Question 4

from sklearn.neighbors import KNeighborsClassifier

# Create the training set and labels
X_train = np.vstack([c1_data, c2_data])
y_train = np.hstack([np.ones(100), np.ones(100) * 2])

# Create the test set
X_test = data.iloc[200:]

# Create the k-NN classifier with k=1 and Euclidean distance measure
knn_1 = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn_1.fit(X_train, y_train)
predictions_1 = knn_1.predict(X_test)

# Create the k-NN classifier with k=3 and Euclidean distance measure
knn_3 = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_3.fit(X_train, y_train)
predictions_3 = knn_3.predict(X_test)

# Calculate the accuracy of the classifiers
true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy_1 = np.mean(predictions_1 == true_labels)
accuracy_3 = np.mean(predictions_3 == true_labels)

print("Accuracy (k=1, Euclidean):", accuracy_1)
print("Accuracy (k=3, Euclidean):", accuracy_3)
```

```
Accuracy (k=1, Euclidean): 1.0
Accuracy (k=3, Euclidean): 1.0
```

In [14]:
```python
knn_manhattan = KNeighborsClassifier(n_neighbors=1, metric='manhattan')
knn_manhattan.fit(X_train, y_train)
predictions_manhattan = knn_manhattan.predict(X_test)
print(predictions_manhattan)

knn_cosine = KNeighborsClassifier(n_neighbors=1, metric='cosine')
knn_cosine.fit(X_train, y_train)
predictions_cosine = knn_cosine.predict(X_test)
print(predictions_cosine)

knn_chebyshev = KNeighborsClassifier(n_neighbors=1, metric='chebyshev')
knn_chebyshev.fit(X_train, y_train)
predictions_chebyshev = knn_chebyshev.predict(X_test)
print(predictions_chebyshev)

true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy_manhattan = np.mean(predictions_manhattan == true_labels)
accuracy_cosine = np.mean(predictions_cosine == true_labels)
accuracy_chebyshev = np.mean(predictions_chebyshev == true_labels)

print("Accuracy Manhattan:", accuracy_manhattan)
print("Accuracy Cosine:", accuracy_cosine)
print("Accuracy Chebyshev:", accuracy_chebyshev)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
[1. 1. 1. 1. 2. 2. 1. 1. 1. 2. 1. 2. 2. 2. 2. 1. 2. 2. 2. 2.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
Accuracy Manhattan: 1.0
Accuracy Cosine: 0.75
Accuracy Chebyshev: 1.0
```

5.

In [15]:
```python
###Question 5

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Create the LDA classifier
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
predictions_lda = lda.predict(X_test)

# Calculate the accuracy of the classifier
true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy_lda = np.mean(predictions_lda == true_labels)

print("Accuracy (LDA):", accuracy_lda)
```

```
Accuracy (LDA): 1.0
```

6.

In [16]:
```
###Question 6

from sklearn.naive_bayes import GaussianNB

# Create the Naive Bayes classifier
nb = GaussianNB()
nb.fit(X_train, y_train)
predictions_nb = nb.predict(X_test)

# Calculate the accuracy of the classifier
true_labels = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
accuracy_nb = np.mean(predictions_nb == true_labels)

print("Accuracy (Naive Bayes):", accuracy_nb)
```

```
Accuracy (Naive Bayes): 1.0
```