Practical No: 1(a)

Aim: Design a simple machine learning model to train the training instances and test the same.

Theory:

Designing a simple machine learning model involves several steps, such as selecting a model, preprocessing data, training the model, and evaluating its performance. I'll provide a basic example using Python and scikit-learn, a popular machine learning library.

Let's assume you're working with a classification problem. We'll use a popular dataset, the Iris dataset, as an example. This dataset has four features (sepal length, sepal width, petal length, and petal width) and three classes (setosa, versicolor, and virginica).

Code:

```
# Import necessary libraries

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset

from sklearn.datasets import load_iris

iris = load_iris()

X, y = iris.data, iris.target

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Standardize the features (mean=0 and variance=1)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Choose a simple model (K-Nearest Neighbors)

knn_model = KNeighborsClassifier(n_neighbors=3)

# Train the model

knn_model.fit(X_train, y_train)

# Make predictions on the test set

y_pred = knn_model.predict(X_test)

# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

classification_rep = classification_report(y_test, y_pred)

# Print the results

print(f"Accuracy: {accuracy}")

print("Classification Report:\n", classification_rep)
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Discussion Results:

Simple machine learning algorithms are often used for tasks such as classification, regression, clustering, and pattern recognition. These algorithms are designed to identify patterns and relationships within data, allowing the system to make predictions or decisions based on new, unseen data.

Some examples of simple machine learning algorithms include:

1. **Linear Regression:** Used for predicting a continuous variable based on one or more input features.
2. **Logistic Regression:** Used for binary classification problems, where the output is a binary (yes/no) decision.
3. **Decision Trees:** Tree-like models that make decisions based on input features. They are used for both classification and regression tasks.
4. **k-Nearest Neighbors (k-NN):** A classification algorithm that classifies a data point based on the majority class of its k-nearest neighbors.
5. **Naive Bayes:** A probabilistic algorithm that is often used for classification problems. It assumes that the features are independent of each other.
6. **Support Vector Machines (SVM):** A supervised learning algorithm used for classification and regression tasks. It finds a hyperplane that best separates data points into different classes.

Practical No: 1(b)

Aim: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Theory:

The find-S algorithm finds the most specific hypothesis that fits all the positive examples.

We have to note here that the algorithm considers only those positive training example.

The find-S algorithm starts with the most specific hypothesis and generalizes this
hypothesis each time it fails to classify an observed positive training data.

Hence, the Find-S algorithm moves from the most specific hypothesis to the most general
hypothesis**.**

Code:

```
import pandas as pd
import numpy as np

# Read the data from the CSV file
data = pd.read_csv("weather_data.csv")
print(data, "\n")

# Extracting attributes
attributes = np.array(data)[:, :-1]
print("The attributes are: ", attributes, "\n")

# Segregating the target with positive and negative examples
target = np.array(data)[:, -1]
print("The target is: ", target, "\n")
```

```
# Training function to implement the find-s algorithm
def train(c, t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
    return specific_hypothesis

# Obtaining the final hypothesis
print ("The final hypothesis is:", train(attributes, target))
```

Output:

```
/Users/schau/OneDrive/Pictures/Machine-Learning-all-Practicals/prac1b-Find-S-Algorithms.py
       Data.Precipitation   Date.Full  Date.Month  ...  Data.Temperature.Min Temp  Data.Wind.Direction Data.Wind.Speed
0                    0.00  2016-01-03           1  ...                         32                   33            4.33
1                    0.00  2016-01-03           1  ...                         31                   32            3.86
2                    0.16  2016-01-03           1  ...                         41                   35            9.73
3                    0.00  2016-01-03           1  ...                         38                   32            6.86
4                    0.01  2016-01-03           1  ...                         29                   19            7.80
...                   ...         ...         ...  ...                        ...                  ...             ...
16738                0.08  2017-01-01           1  ...                         15                   23           19.98
16739                0.00  2017-01-01           1  ...                         21                   26           15.16
16740                0.00  2017-01-01           1  ...                          4                   26            1.65
16741                0.06  2017-01-01           1  ...                         13                   24           18.16
16742                0.10  2017-01-01           1  ...                          8                   23            7.51

[16743 rows x 14 columns]

The attributes are:  [[0.0 '2016-01-03' 1 ... 46 32 33]
 [0.0 '2016-01-03' 1 ... 47 31 32]
 [0.16 '2016-01-03' 1 ... 51 41 35]
 ...
 [0.0 '2017-01-01' 1 ... 29 4 26]
 [0.06 '2017-01-01' 1 ... 31 13 24]
 [0.1 '2017-01-01' 1 ... 34 8 23]]

The target is:  [4.33 3.86 9.73 ... 1.65 18.16 7.51]
```

Discussion Results:

The Find-S algorithm assumes that the hypothesis language is a conjunction of constraints on the attribute values.

It is particularly suitable for learning from positive examples when the negative examples are not explicitly given.

The algorithm is guaranteed to find a consistent hypothesis if the true hypothesis exists in the hypothesis space.

It's important to note that the Find-S algorithm is a basic algorithm and may not perform well in complex scenarios or when dealing with noisy data.

More sophisticated algorithms, such as decision trees or support vector machines, are often used in practice for more accurate and robust learning.

Practical 2(a)

Aim: Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Theory:

Performing data loading, feature selection using Principal Component Analysis (PCA), and feature scoring and ranking are common tasks in data preprocessing and analysis. Here's a general outline of how you can approach these tasks using Python and popular libraries such as pandas, scikit-learn, and seaborn.

Adjust the code based on your specific dataset, target variable, and requirements. Additionally, you may need to handle missing values, encode categorical variables, and perform other preprocessing steps based on the characteristics of your data.

Code:

```
# Import necessary libraries

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import mutual_info_regression  # Change to mutual_info_regression for continuous target variable



# Step 1: Data Loading

# Load your dataset (replace 'your_dataset.csv' with your actual file path or URL)

data = pd.read_csv("C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-all-Practicals\\diabetes.csv")
```

```
# Display the first few rows of the dataset to inspect the data
print("Step 1: Data Loading")
print(data.head())

# Step 2: Feature Selection using Principal Component Analysis (PCA)
# Assuming 'X' contains your features and 'y' contains your target variable
X = data.drop('BMI', axis=1)  # Adjust 'target_variable' with the actual name
y = data['BMI']

# Standardize the features (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Specify the number of components you want to keep
n_components = 5  # Adjust as needed

# Apply PCA
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Display the explained variance ratio to understand how much variance is retained
print("\nStep 2: Feature Selection using PCA")
```

```
print("Explained Variance Ratio:", pca.explained_variance_ratio_)


# Step 3: Feature Scoring and Ranking

# Compute mutual information scores for each feature

mi_scores = mutual_info_regression(X, y)  # Change to mutual_info_regression for continuous target variable


# Create a DataFrame to display feature scores

feature_scores = pd.DataFrame({'Feature': X.columns, 'MI Score': mi_scores})

feature_scores = feature_scores.sort_values(by='MI Score', ascending=False)


# Display the feature scores

print("\nStep 3: Feature Scoring and Ranking")

print(feature_scores)
```

Output:

```
/Users/schau/OneDrive/Pictures/Machine-Learning-all-Practicals/prac2a-PCA.py
Step 1: Data Loading
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72             35        0  33.6                     0.627   50        1
1            1       85             66             29        0  26.6                     0.351   31        0
2            8      183             64              0        0  23.3                     0.672   32        1
3            1       89             66             23       94  28.1                     0.167   21        0
4            0      137             40             35      168  43.1                     2.288   33        1

Step 2: Feature Selection using PCA
Explained Variance Ratio: [0.2656751  0.21242685 0.13578651 0.10987471 0.09483919]

Step 3: Feature Scoring and Ranking
                    Feature  MI Score
3             SkinThickness  0.205891
7                   Outcome  0.115577
2             BloodPressure  0.110996
0               Pregnancies  0.056714
1                   Glucose  0.051769
6                       Age  0.035476
5  DiabetesPedigreeFunction  0.006989
4                   Insulin  0.005319
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Principal Component Analysis (PCA) is a dimensionality reduction technique widely used in machine learning and data analysis.

Its primary goal is to simplify the complexity in high-dimensional data while retaining trends and patterns.

PCA transforms the original features into a new set of uncorrelated features, which are linear combinations of the original ones, called principal components.

It's important to note that while PCA is a powerful technique, it assumes a linear relationship between variables.

In cases where the relationships are highly nonlinear, other techniques like t-SNE (t-distributed Stochastic Neighbor Embedding) might be more appropriate.

Practical No 2(b)

Aim: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output description of the set of all hypothesis consistent with the training examples.

Theory:

1) The candidate elimination algorithm incrementally builds the version space given a

hypothesis space H and a set E of examples.

2) The examples are added one by one; each example possibly shrinks the version space by

removing the hypotheses that are inconsistent with the example.

3) The candidate elimination algorithm does this by updating the general and specific

boundary for each new example.

 You can consider this as an extended form of Find-S algorithm.

 Consider both positive and negative examples.

 Actually, positive examples are used here as Find-S algorithm (Basically they are

generalizing from the specification).

 While the negative example is specified from generalize form.

Code:

```
import csv


a = []
```

```
with open("C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-all-Practicals\\data2.csv") as csvfile:
    next(csvfile)
    for row in csv.reader(csvfile):
        a.append(row)


for x in a:
    print(x)


print("\nThe total number of training instances are : ", len(a))


num_attribute = len(a[0]) - 1
print("\nThe initial hypothesis is : ")
hypothesis = ["0"] * num_attribute
print(hypothesis)


for i in range(0, len(a)):
    if a[i][num_attribute] == "yes":
        print("\nInstance ", i + 1, "is", a[i], " and is Positive Instance")


        for j in range(0, num_attribute):
            if hypothesis[j] == "0" or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]


            else:
                hypothesis[j] = "?"


        print(
```

```
            "The hypothesis for the training instance", i + 1, " is: ", hypothesis, "\n"
        )

    if a[i][num_attribute] == "no":
        print(
            "\nInstance ", i + 1, "is", a[i], " and is Negative Instance Hence Ignored"
        )

        print(
            "The hypothesis for the training instance", i + 1, " is: ", hypothesis, "\n"
        )

print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)
```

Output:

```
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals> & C:/Users/schau/AppData/
/Users/schau/Downloads/Practical2/Practical2/Prac2B.py
['many', 'big', 'no', 'one', 'yes']
['some', 'big', 'always', 'few', 'no']
['many', 'medium', 'no', 'many', 'yes']
['many', 'small', 'no', 'many', 'yes']

The total number of training instances are :  4

The initial hypothesis is :
['0', '0', '0', '0']

Instance  1 is ['many', 'big', 'no', 'one', 'yes']  and is Positive Instance
The hypothesis for the training instance 1  is:  ['many', 'big', 'no', 'one']


Instance  2 is ['some', 'big', 'always', 'few', 'no']  and is Negative Instance Hence Ignored
The hypothesis for the training instance 2  is:  ['many', 'big', 'no', 'one']


Instance  3 is ['many', 'medium', 'no', 'many', 'yes']  and is Positive Instance
The hypothesis for the training instance 3  is:  ['many', '?', 'no', '?']


Instance  4 is ['many', 'small', 'no', 'many', 'yes']  and is Positive Instance
The hypothesis for the training instance 4  is:  ['many', '?', 'no', '?']


The Maximally specific hypothesis for the training instance is  ['many', '?', 'no', '?']
```

Result Discussion:

In the context of machine learning, a hypothesis is a proposed explanation or model for a set of observations or data. The term is commonly used in the context of supervised learning, where the goal is to learn a mapping from input data to output labels or predictions. There are different types of hypotheses, depending on the learning algorithm and the nature of the problem. Here are some common types of hypotheses:

1. **Boolean Hypothesis:**
    - In binary classification problems, where the goal is to predict one of two possible classes (e.g., spam or not spam), a boolean hypothesis predicts the class label as either true or false.
2. **Numeric Hypothesis:**
    - In regression problems, where the goal is to predict a continuous numeric value, a numeric hypothesis outputs a real number as the predicted value.
3. **Linear Hypothesis:**
    - A linear hypothesis assumes a linear relationship between the input features and the output. For example, in a simple linear regression problem, the hypothesis might be a straight line equation.
4. **Polynomial Hypothesis:**
    - A polynomial hypothesis allows for non-linear relationships by including polynomial terms of the input features. This is used when a linear model is insufficient to capture the underlying patterns in the data.
5. **Neural Network Hypothesis:**
    - In the context of neural networks, the hypothesis involves a complex arrangement of interconnected nodes organized into layers. The activation of nodes in the output layer represents the predicted values.
6. **Decision Tree Hypothesis:**
    - Decision trees make predictions by recursively splitting the data into subsets based on the values of input features. The hypothesis is a tree structure where each leaf node corresponds to a predicted output.
7. **Support Vector Machine (SVM) Hypothesis:**
    - In SVM, the hypothesis is a decision boundary that maximally separates data points of different classes in the feature space. This boundary is defined by a set of support vectors.

Practical No: 3 (a)

Aim: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Theory:

The Naive Bayes classifier is a probabilistic machine learning algorithm that is based on Bayes' theorem. It is known as "naive" because it makes the assumption of independence between features, which means it assumes that the presence or absence of a particular feature is independent of the presence or absence of any other feature given the class variable. This assumption simplifies the computation and makes it computationally efficient.

Here are the key components and steps of the Naive Bayes classifier:

Bayes' Theorem: The algorithm is based on Bayes' theorem, which describes the probability of a hypothesis (class) given some evidence (features).

P(Class|Features)=P(Features|Class)×P(Class)P(Features)P(Class|Features)=P(Features)P(Features|Class)×P(Class).

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report



# Load the dataset

data = pd.read_csv("C:/Users/schau/OneDrive/Pictures/Machine-Learning-all-Practicals/diabetes.csv")

# Assuming the last column is the target variable and the rest are features

X = data.iloc[:, :-1]
```

```
y = data.iloc[:, -1]

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Naive Bayes classifier

classifier = GaussianNB()

# Train the classifier on the training data

classifier.fit(X_train, y_train)

# Make predictions on the test data

y_pred = classifier.predict(X_test)

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

# Display additional classification metrics

print("\nClassification Report:")

print(classification_report(y_test, y_pred))
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

           1       0.66      0.71      0.68        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154

PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Results Discussion:

Bayes' Theorem:

P(A|B)=P(B|A)·P(A)P(B)P(A|B)=P(B)P(B|A)·P(A)

- P(A|B)P(A|B) is the probability of event A occurring given that event B has occurred.
- P(B|A)P(B|A) is the probability of event B occurring given that event A has occurred.
- P(A)P(A) and P(B)P(B) are the probabilities of events A and B occurring, respectively.

Naive Bayes Classifier:

The Naive Bayes algorithm makes a "naive" assumption that the features used to describe an observation are independent given the class label. This simplifying assumption makes the computations more tractable.

In the context of a classification problem, where you have a set of features XX and a class label CC, Naive Bayes can be expressed as:

P(C|X)=P(X|C)·P(C)P(X)P(C|X)=P(X)P(X|C)·P(C)

- P(C|X)P(C|X) is the probability of class CC given the observation XX.
- P(X|C)P(X|C) is the likelihood of observing XX given class CC.
- P(C)P(C) is the prior probability of class CC.
- P(X)P(X) is the probability of observing XX.

The "naive" part comes from assuming that the features are conditionally independent given the class label:

P(X|C)=P(x1|C)·P(x2|C)·…·P(xn|C)P(X|C)=P(x1|C)·P(x2|C)·…·P(xn|C)

where x1,x2,…,xnx1,x2,…,xn are the features.

Naive Bayes is particularly popular for text classification tasks, such as spam filtering or sentiment analysis. It's computationally efficient and often performs well, even with the independence assumption. However, if the features are strongly dependent on each other, the performance of Naive Bayes may suffer.

Practical 3(b)

Aim: Write a program to implement Decision Tree and Random forest with Prediction, Test

Score and Confusion Matrix

Theory:

Decision Tree:

1) Decision Tree is a supervised learning algorithm used in machine learning. It operated in

both classification and regression algorithms. As the name suggests, it is like a tree with

nodes.

2) The branches depend on the number of criteria. It splits data into branches like these till it

achieves a threshold unit. A decision tree has root nodes, children's nodes, and leaf nodes.

3) Recursion is used for traversing through the nodes. You need no other algorithm. It

handles data accurately and works best for a linear pattern. It handles large data easily and

takes less time.

Advantages

1. Easy

2. Transparent process

3. Handle both numerical and categorical data

4. Larger the data, the better the result

5. Speed

Disadvantages

1. May overfit

2. Pruning process large

3. Optimization unguaranteed

4. Complex calculations

5. Deflection high

Random-Forest-

1) It is also used for supervised learning but is very powerful. It is very widely used. The

basic difference being it does not rely on a singular decision. It assembles randomized

decisions based on several decisions and makes the final decision based on the majority.

2) It does not search for the best prediction. Instead, it makes multiple random predictions.

Thus, more diversity is attached, and prediction becomes much smoother.

3) You can infer Random-forest to be a collection of multiple decision trees! Decision trees

are very easy as compared to the random forest. A decision tree combines some decisions,

whereas a random forest combines several decision trees. Thus, it is a long process, yet slow.

Advantages:

1. Powerful and highly accurate

2. No need to normalizing

3. Can handle several features at once

4. Run trees in parallel ways

Disadvantages:

1. They are biased to certain features sometimes

2. Slow

3. Cannot be used for linear methods

4. Worse for high dimensional data

Confusion Matrix:

1) A confusion matrix is a technique for summarizing the performance of a classification

algorithm.

2) Classification accuracy alone can be misleading if you have an unequal number of

observations in each class or if you have more than two classes in your dataset.

Code:

```
from sklearn.datasets import load_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import pandas as pd

import numpy as np


# Loading the data

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

```
df['is_train'] = np.random.uniform(0, 1, len(df)) <= 0.75  # Fix the bug in the threshold value


# Creating dataframes with test rows and train rows

train, test = df[df['is_train'] == True], df[df['is_train'] == False]


# Show the number of observations for the test and train dataframes

print('No. of observations in the train data: ', len(train))

print('No. of observations in the test data: ', len(test))


features = df.columns[:4]


# Converting each species into digits

y = pd.factorize(train['species'])[0]


# Creating a random forest classifier

clf = RandomForestClassifier(n_jobs=2, random_state=0)


# Training the classifier

clf.fit(train[features], y)


# Making predictions

predictions = clf.predict(test[features])
```

```
# Mapping names for the plants for each predicted plant

preds = iris.target_names[clf.predict(test[features])]


# Displaying actual vs. predicted species

print("Actual vs. Predicted Species:")

print(pd.DataFrame({'Actual Species': test['species'], 'Predicted Species':
preds}).head())


# Creating confusion matrix

conf_matrix = pd.crosstab(test['species'], preds, rownames=['Actual Species'],
colnames=['Predicted Species'])

print("\nConfusion Matrix:")

print(conf_matrix)


# Displaying additional metrics

print("\nClassification Report:")

print(classification_report(test['species'], preds))


# Calculating and displaying accuracy score

accuracy = accuracy_score(test['species'], preds)

print('\nThe accuracy score is:', accuracy)
```

Output:

```
ictures/Machine-Learning-all-Practicals/prac3b-Decission-Tree.py
No. of observations in the train data:  111
No. of observations in the test data:  39
Actual vs. Predicted Species:
   Actual Species Predicted Species
1          setosa            setosa
12         setosa            setosa
15         setosa            setosa
17         setosa            setosa
19         setosa            setosa

Confusion Matrix:
Predicted Species  setosa  versicolor  virginica
Actual Species
setosa                 13           0          0
versicolor              0          14          0
virginica               0           2         10

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.88      1.00      0.93        14
   virginica       1.00      0.83      0.91        12

    accuracy                           0.95        39
   macro avg       0.96      0.94      0.95        39
weighted avg       0.96      0.95      0.95        39


The accuracy score is: 0.9487179487179487
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Decision Trees:

Decision Trees are a versatile and widely used machine learning algorithm for both classification and regression tasks. The key advantages include:

1. **Interpretability:** Decision Trees are easy to understand and visualize. You can easily trace the decision-making process and interpret the rules used for classification.

2. **Handling Non-Linearity:** They can capture non-linear relationships in the data, making them suitable for a variety of datasets.

3. **No Data Normalization Required:** Decision Trees do not require data normalization or scaling, making them robust to different types of data.

4. **Feature Importance:** Decision Trees provide a feature importance score, which helps in understanding which features are more influential in making decisions.

Random Forest:

Random Forest is an ensemble learning method that builds multiple Decision Trees and merges their predictions. Here are the key points:

1. **Improved Generalization:** Random Forest reduces overfitting compared to individual Decision Trees by aggregating predictions from multiple trees. This often leads to better generalization on unseen data.

2. **High Accuracy:** Random Forest tends to provide high accuracy in a wide range of applications. It is considered a robust and powerful algorithm.

3. **Handling Missing Values:** Random Forest can handle missing values in the dataset and maintain its predictive performance.

Practical 4 (a)

Aim: For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

Theory:

1) The least-squares method is a form of mathematical regression analysis used to determine the

line of best fit for a set of data, providing a visual demonstration of the relationship between the

data points.

2) Each point of data represents the relationship between a known independent variable and an

unknown dependent variable.

3) This method of regression analysis begins with a set of data points to be plotted on an x- and y-

axis graph. An analyst using the least-squares method will generate a line of best fit that explains

the potential relationship between independent and dependent variables.

4) The least-squares method provides the overall rationale for the placement of the line of best fit

among the data points being studied.

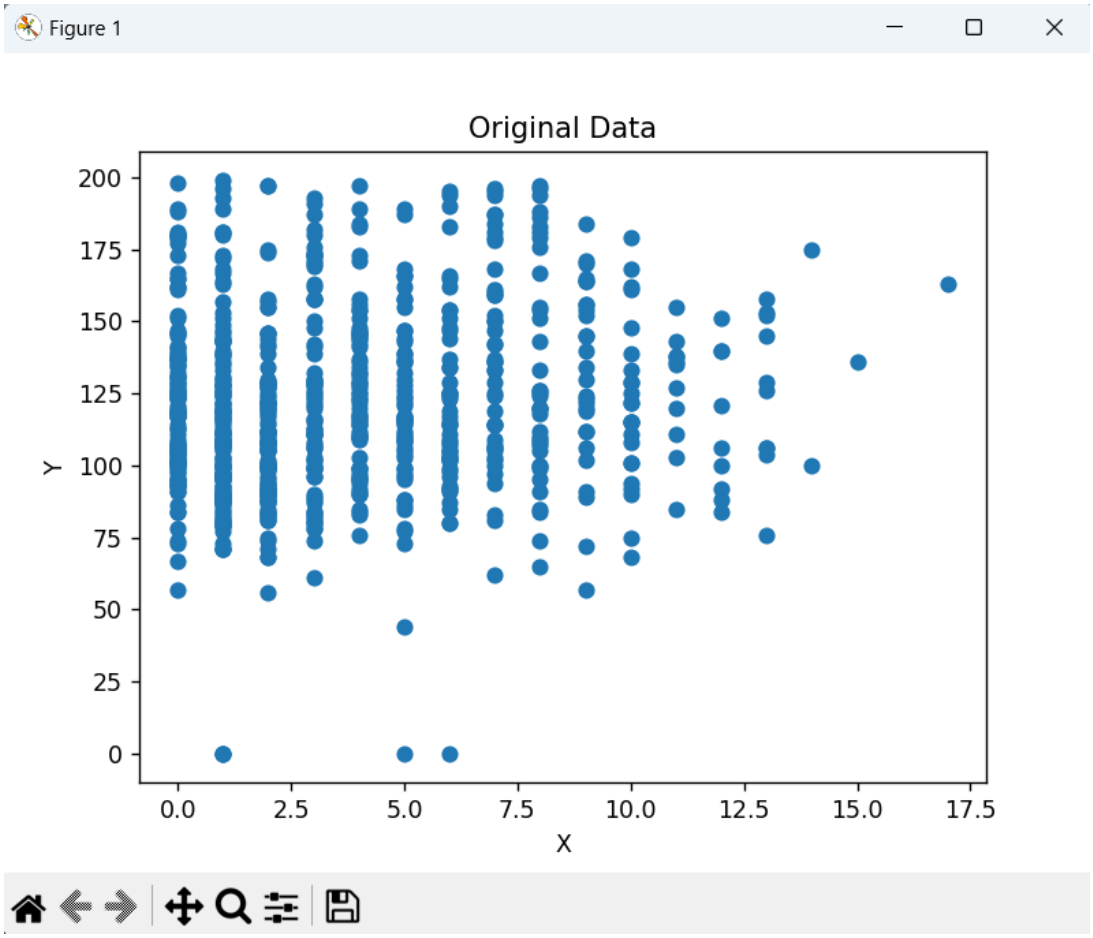
Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# Read data from CSV
```

```
data = pd.read_csv("C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-all-Practicals\\diabetes.csv")
X = data.iloc[:, 0]
Y = data.iloc[:, 1]

# Plot the original data
plt.scatter(X, Y)
plt.title('Original Data')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

# Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)
num = 0
den = 0

for i in range(len(X)):
    num += (X[i] - X_mean) * (Y[i] - Y_mean)
    den += (X[i] - X_mean) ** 2

m = num / den
c = Y_mean - m * X_mean

print(f"Slope (m): {m}, Intercept (c): {c}")

# Making predictions
```

```
Y_pred = m * X + c


# Plotting the original and predicted data
plt.scatter(X, Y, label='Actual Data')
plt.scatter(X, Y_pred, color='red', label='Predicted Data')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red',
label='Regression Line')
plt.title('Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Output:



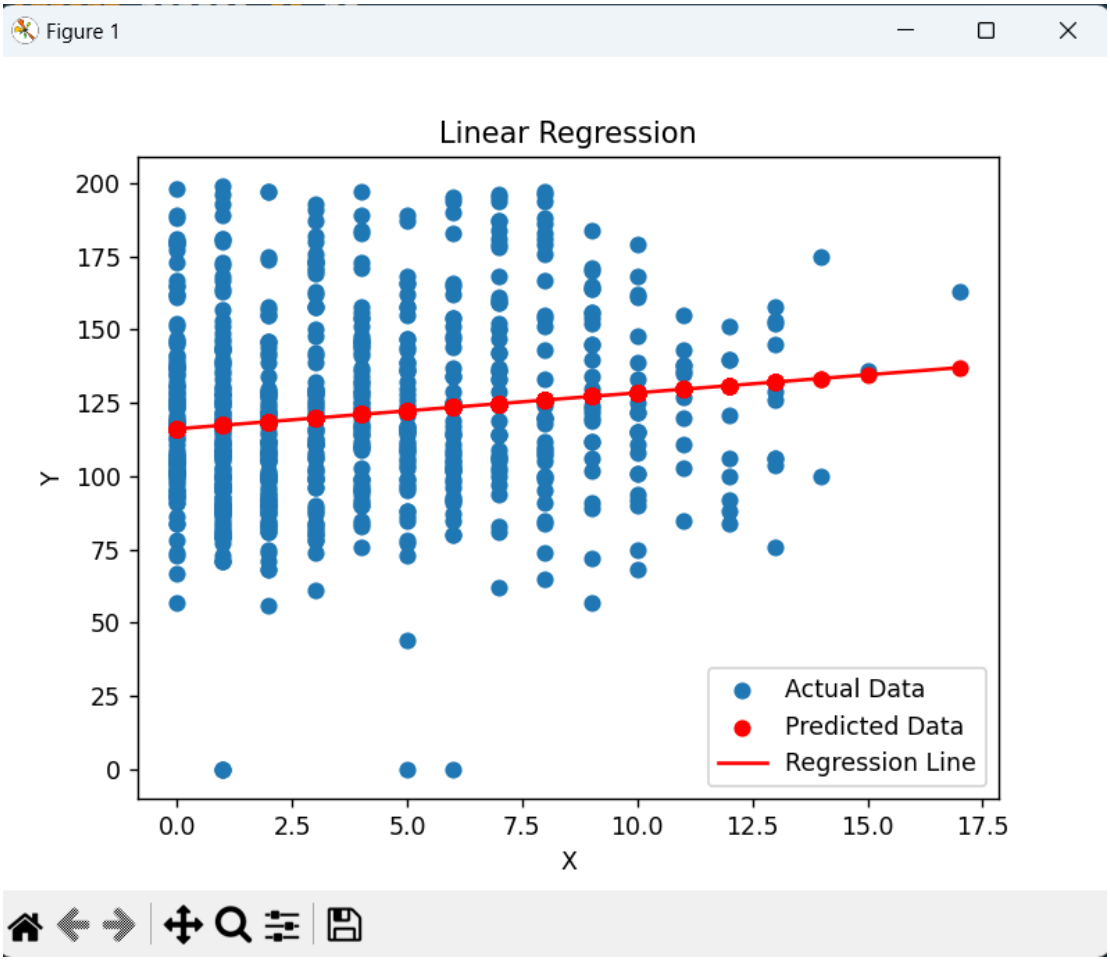


Figure 1
Linear Regression
200
175
150
125
100
75
50
25
0
Y
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
X
Actual Data
Predicted Data
Regression Line

Result Discussion:

In machine learning, least squares regression is a widely used method for modeling the relationship between independent variables and a dependent variable.

The goal of least squares regression is to find the parameters of a linear model that minimize the sum of the squared differences between the observed and predicted values.

Here's a discussion of the results and considerations in least squares regression:

Model Fit:

- The primary result of least squares regression is the set of coefficients that define the linear relationship between the independent and dependent variables. These coefficients represent the slope and intercept of the regression line.
- A well-fitted model should have coefficients that accurately capture the underlying patterns in the data.

Residual Analysis:

- Residuals are the differences between the observed and predicted values. Analyzing residuals is crucial for assessing the goodness of fit. Residual plots can be used to identify patterns, heteroscedasticity, or outliers.
- Ideally, residuals should be randomly distributed around zero, indicating that the model captures the underlying variability in the data.

Practical 4(b)

Aim: For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.

Theory:

Logistic Regression is a statistical method used for binary classification in machine learning. Despite its name, it is used for classification rather than regression tasks. The model is called "logistic" because it uses the logistic function to model a binary dependent variable.

Here's a brief overview of Logistic Regression:

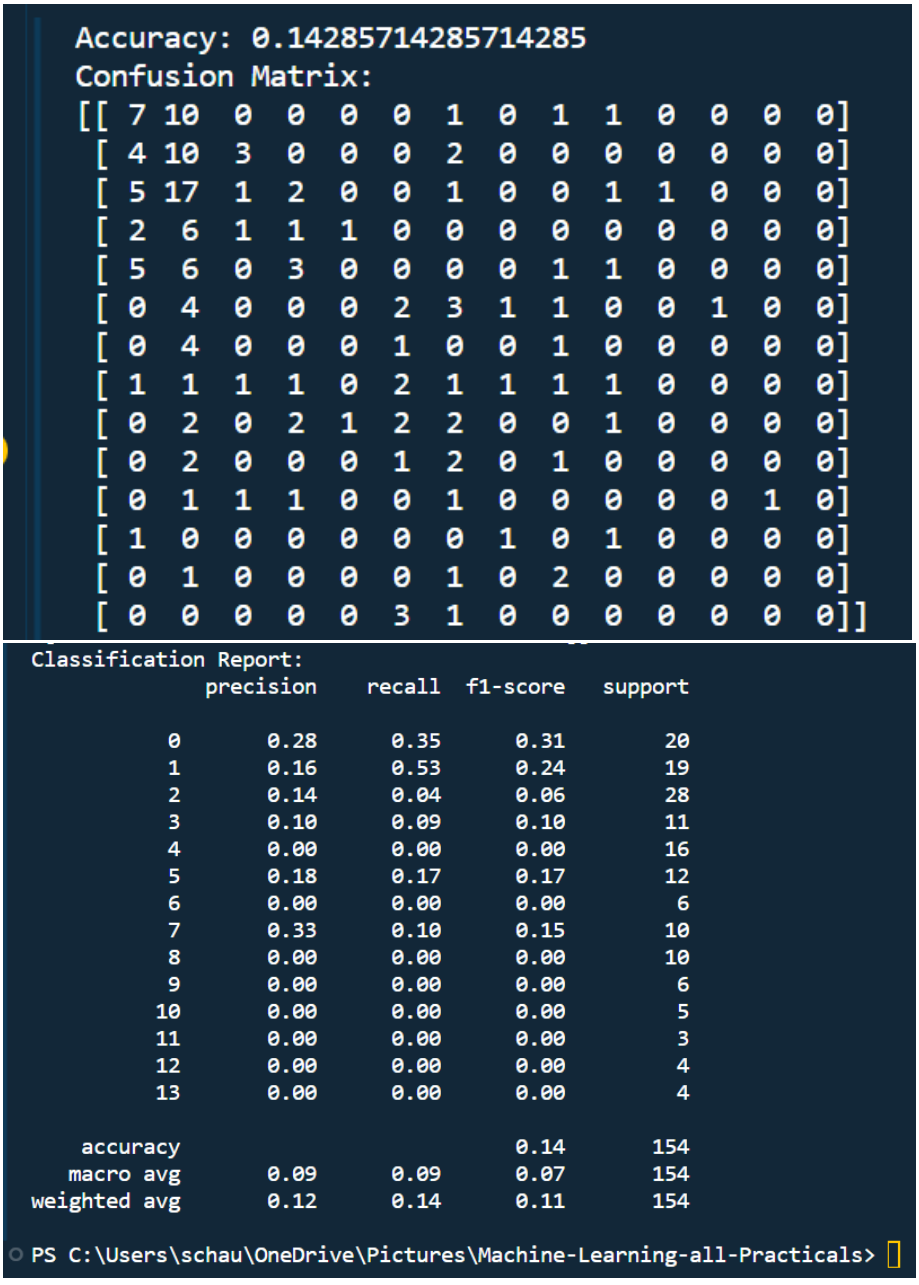
1. Binary Classification:
    - Logistic Regression is primarily used for binary classification problems where the target variable has two possible outcomes (usually 0 or 1, or negative and positive).
2. Logistic Function (Sigmoid):
    - The logistic function, also known as the sigmoid function, is a key component of logistic regression. It transforms any real-valued number into the range between 0 and 1. The formula for the sigmoid function is: $\sigma(z)=11+e-z\sigma(z)=1+e-z1$ where zz is a linear combination of input features and their weights.
3. Hypothesis Function:
    - The logistic regression model's hypothesis function is the sigmoid function applied to the linear combination of input features and their corresponding weights. $h\theta(x)=\sigma(\theta Tx)h\theta(x)=\sigma(\theta Tx)$ Here, $h\theta(x)h\theta(x)$ represents the predicted probability that $y=1y=1$ given the input features xx and model parameters $\theta\theta$.
4. Training Process:
    - During training, the model learns the optimal weights $\theta\theta$ by minimizing a cost function. Commonly used cost functions for logistic regression include the cross-entropy loss.
5. Decision Boundary:
    - The decision boundary is the line (for 2D data) or hyperplane (for more than two features) that separates the instances of one class from another. It is determined by the learned weights.
6. Prediction:
    - To make predictions, the model uses the learned weights to calculate the probability that a given input belongs to the positive class. If this probability is greater than a threshold (commonly 0.5), the model predicts class 1; otherwise, it predicts class 0.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Load the data
file_path = "C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-
all-Practicals\\diabetes.csv"
data = pd.read_csv(file_path)

# Assuming the first column is the target variable (label) and the rest are
features
X = data.iloc[:, 1:]
y = data.iloc[:, 0]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate the model
```

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{classification_rep}")
```

Output:



```
Accuracy: 0.14285714285714285
Confusion Matrix:
[[ 7 10  0  0  0  0  1  0  1  1  0  0  0  0]
 [ 4 10  3  0  0  0  2  0  0  0  0  0  0  0]
 [ 5 17  1  2  0  0  1  0  0  1  1  0  0  0]
 [ 2  6  1  1  1  0  0  0  0  0  0  0  0  0]
 [ 5  6  0  3  0  0  0  0  1  1  0  0  0  0]
 [ 0  4  0  0  0  2  3  1  1  0  0  1  0  0]
 [ 0  4  0  0  0  1  0  0  1  0  0  0  0  0]
 [ 1  1  1  1  0  2  1  1  1  1  0  0  0  0]
 [ 0  2  0  2  1  2  2  0  0  1  0  0  0  0]
 [ 0  2  0  0  0  1  2  0  1  0  0  0  0  0]
 [ 0  1  1  1  0  0  1  0  0  0  0  0  1  0]
 [ 1  0  0  0  0  0  0  1  0  1  0  0  0  0]
 [ 0  1  0  0  0  0  1  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  3  1  0  0  0  0  0  0  0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.28      0.35      0.31        20
           1       0.16      0.53      0.24        19
           2       0.14      0.04      0.06        28
           3       0.10      0.09      0.10        11
           4       0.00      0.00      0.00        16
           5       0.18      0.17      0.17        12
           6       0.00      0.00      0.00         6
           7       0.33      0.10      0.15        10
           8       0.00      0.00      0.00        10
           9       0.00      0.00      0.00         6
          10       0.00      0.00      0.00         5
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         4
          13       0.00      0.00      0.00         4

    accuracy                           0.14       154
   macro avg       0.09      0.09      0.07       154
weighted avg       0.12      0.14      0.11       154

PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Logistic Regression is a widely used machine learning algorithm for binary classification problems.

The key idea behind logistic regression is to model the probability that an instance belongs to a particular class. Here's a discussion of the results and considerations in logistic regression:

1. **Model Coefficients:**
    - Logistic regression estimates coefficients for each feature, determining the strength and direction of their influence on the log-odds of the target class. Positive coefficients indicate a positive association with the target class, while negative coefficients suggest a negative association.
2. **Probability Threshold:**
    - Logistic regression outputs probabilities between 0 and 1. A common threshold (e.g., 0.5) is used to classify instances into one of the two classes. Adjusting this threshold can trade off precision and recall, impacting the trade-off between false positives and false negatives.
3. logistic regression is a powerful algorithm for binary classification with interpretable coefficients.

4. Understanding the model's output, evaluating its performance using appropriate metrics, and considering factors like calibration and multicollinearity are essential for a comprehensive discussion of the results.
5. Additionally, regularization and handling imbalanced data may be important considerations based on the characteristics of the dataset.

Practical No: 5(a)

Aim: Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Theory:

The Decision Tree algorithm, based on the ID3 (Iterative Dichotomiser 3) algorithm, is a popular machine learning algorithm used for both classification and regression tasks. The ID3 algorithm was developed by Ross Quinlan and is one of the earliest algorithms used to construct decision trees.

Here are the key concepts associated with the ID3 algorithm and Decision Trees:

1. **Decision Tree Structure:**
   - A Decision Tree is a tree-like model where each internal node represents a decision based on the value of a particular feature, each branch represents the outcome of the decision, and each leaf node represents the final predicted output (class label or regression value).
2. **Entropy and Information Gain:**
   - ID3 uses the concept of entropy to measure the impurity or disorder of a dataset. Entropy is a measure of uncertainty. The algorithm selects the feature that provides the maximum information gain, which is the reduction in entropy after a dataset is split based on that feature.
   - Entropy is given by: H(S)=−∑i=1cpilog⁡2(pi)H(S)=−∑i=1cpilog2(pi) where H(S)H(S) is the entropy of the dataset SS, cc is the number of classes, and pipi is the proportion of instances in class ii.
3. **Recursive Tree Building:**
   - The ID3 algorithm recursively splits the dataset based on the feature that maximizes information gain. This process continues until the tree is deep enough, a stopping criterion is met (e.g., maximum depth reached, minimum samples per leaf), or the dataset is pure (all instances belong to the same class).
4. **Categorical Features:**
   - ID3 is designed for datasets with categorical features. It can handle both discrete and symbolic values.

5. **Handling Overfitting:**
    - Decision Trees have the potential to overfit the training data. Techniques such as pruning (cutting branches of the tree) and setting stopping criteria help mitigate overfitting.
6. **Prediction:**
    - To make predictions for a new instance, it traverses the tree from the root to a leaf, following the decisions based on feature values.

Code:

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.datasets import load_iris



# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target



# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)



# Initialize the Decision Tree Classifier

dt_classifier = DecisionTreeClassifier()



# Train the model
```

```
dt_classifier.fit(X_train, y_train)


# Make predictions
y_pred = dt_classifier.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")


# Example of classifying a new sample
new_sample = [[5.0, 3.5, 1.5, 0.2]]  # Example data for a new sample
predicted_class = dt_classifier.predict(new_sample)
print(f"Predicted Class for New Sample: {predicted_class}")
```

Output:

```
Accuracy: 1.0
Predicted Class for New Sample: [0]
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Results Discussion:

Decision Trees are versatile machine learning algorithms used for both classification and regression tasks.

They work by recursively splitting the data based on feature values, creating a tree-like structure where each internal node represents a decision based on a feature,

Each branch represents the outcome of the decision, and each leaf node represents the final prediction.

Decision Trees offer a transparent and interpretable way to model decision-making processes.

However, care should be taken to address overfitting, and hyperparameter tuning is crucial for optimizing performance.

Ensemble methods and other techniques can be employed to enhance the model's robustness and predictive accuracy.

Practical No 5(b)

Aim: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.

Theory:

1) K-Nearest Neighbors is one of the most basic yet essential classification algorithms in
Machine Learning. It belongs to the supervised learning domain and finds intense application in
pattern recognition, data mining and intrusion detection.

2) It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not
make any underlying assumptions about the distribution of data (as opposed to other algorithms
such as GMM, which assume a Gaussian distribution of the given data).

3) We are given some prior data (also called training data), which classifies coordinates into
groups identified by an attribute.

Advantages:
1) The algorithm is simple and easy to implement.
2) There's no need to build a model, tune several parameters, or make additional assumptions.
3) The algorithm is versatile. It can be used for classification, regression, and search (as we will
see in the next section).

Disadvantages:
1) The algorithm gets significantly slower as the number of examples and/or
predictors/independent variables increase.

Code:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as numpy
from sklearn import datasets
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target
X_train,X_test,y_train,y_test = (train_test_split(iris_data,iris_labels
,test_size=0.3))
classifier = KNeighborsClassifier(n_neighbors=13)
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
print("Accuracy is: ")
print(classification_report(y_test,y_pred))
```

Output:

```
/Users/schau/OneDrive/Pictures/Machine-Learning-all-Practicals/prac5b.py
Accuracy is:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.93      0.93      0.93        15
           2       0.94      0.94      0.94        18

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

After applying K-Means, you would typically analyze the resulting clusters to understand patterns in the data. This might involve examining the characteristics of each cluster, assessing the separation between clusters, and determining if the clusters make sense based on your domain knowledge.

Nearest Neighbors:

- **Purpose:** Nearest Neighbors algorithms, such as k-Nearest Neighbors (k-NN), are used for both classification and regression tasks. They work by finding the k data points in the training set that are closest to a given test data.
-
- After applying a Nearest Neighbors algorithm, you would typically evaluate its performance using metrics such as accuracy (for classification) or mean squared error (for regression). Additionally,
- We might analyze the sensitivity of the algorithm to different values of k and explore the distribution of classes or values in the neighborhood of data points.

If we meant to ask about the combination of K-Means and Nearest Neighbors, it's worth noting that they can be used together in certain scenarios,

such as using K-Means to cluster data and then applying Nearest Neighbors within each cluster.

This can be beneficial in situations where the data has a natural clustering structure, and you want to perform local predictions within each cluster.

Practical No: 6(a)

Aim: Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

Theory:

Distance Method (Euclidean Distance):

1. Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. For a given pair of points (x1,y1,z1,…)(x1,y1 ,z1,…) and (x2,y2,z2,…)(x2,y2,z2,…) in an n-dimensional space, the Euclidean distance dd is calculated as: d=(x2−x1)2+(y2−y1)2+…d=(x2 −x1)2+(y2−y1)2+…
    2.
    -
3. **Prediction:**
    - In the context of machine learning, prediction refers to the process of using a trained model to estimate the output (or class label) for a new set of input features. The prediction is obtained by applying the learned model to the input data.
4. **Test Score:**
    - The test score is a metric that evaluates the performance of a machine learning model on a test dataset. It is often measured using metrics such as accuracy, precision, recall, F1 score, etc. The test score provides an indication of how well the model generalizes to unseen data.
5. **Confusion Matrix:**
    - A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). From these values, various performance metrics like accuracy, precision, recall, and F1 score can be calculated.

Code:

```
 from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

from sklearn.datasets import load_iris


# Load the Iris dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Initialize the k-NN Classifier with Euclidean distance

knn_classifier = KNeighborsClassifier(n_neighbors=3, metric='euclidean')


# Train the model

knn_classifier.fit(X_train, y_train)


# Make predictions

y_pred = knn_classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print results
print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{classification_rep}")

# Example of classifying a new sample
new_sample = [[5.0, 3.5, 1.5, 0.2]]  # Example data for a new sample
predicted_class = knn_classifier.predict(new_sample)
print(f"Predicted Class for New Sample: {predicted_class}")
```

Output:

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Predicted Class for New Sample: [0]
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Euclidean Distance in Machine Learning:

- **Purpose:** Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. In machine learning, it is commonly used as a distance metric in algorithms like k-Nearest Neighbors (k-NN) or hierarchical clustering.
- **Application in k-NN:** In k-NN, for example, when predicting the class or value of a data point, the algorithm looks at the k-nearest neighbors in the feature space, and often Euclidean distance is used to measure this proximity.

Test Scores in Machine Learning:

- **Feature Representation:** Test scores can be used as features in machine learning models. For example, if you have a dataset where each data point represents a student and their features include test scores in various subjects, you can use these scores as input features for predictive models.
- **Predictive Modeling:** You might use machine learning algorithms to predict outcomes such as student performance, graduation likelihood, or other educational indicators based on test scores and potentially other features.

Combining Euclidean Distance and Test Scores:

- **k-NN Example:** Suppose you have a dataset of students with features including test scores in subjects like math, science, and English. If you apply k-NN to predict a student's performance, the algorithm could use the Euclidean distance between the test score vectors of students to identify the k-nearest neighbors.
- **Cluster Analysis:** In another scenario, you might use Euclidean distance within a clustering algorithm (e.g., K-Means) to group students with similar test score patterns into clusters.

Practical 6(b)

Aim: Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

Theory:

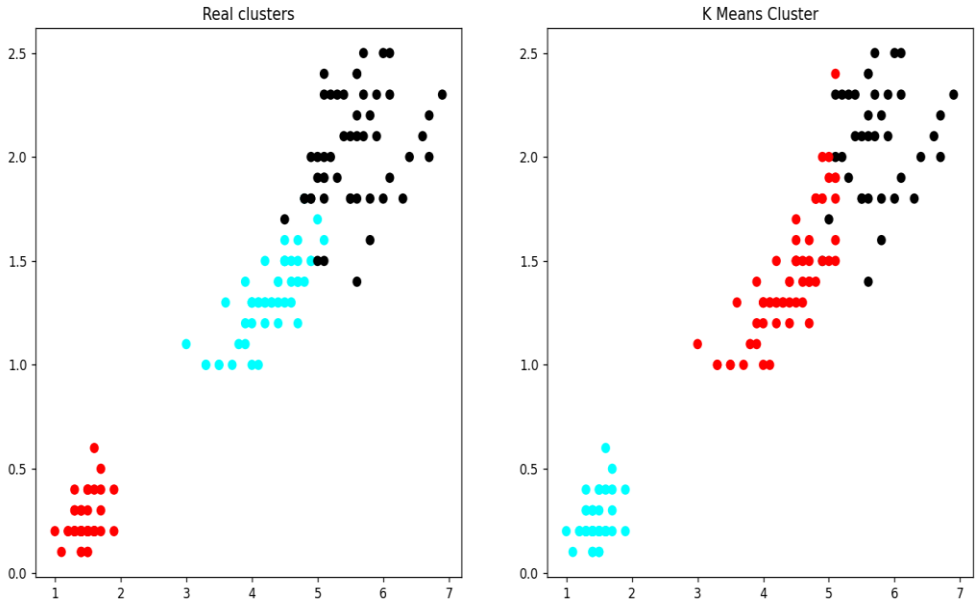
1) K-means clustering is one of the simplest and popular unsupervised machine learning
algorithms.
2) Typically, unsupervised algorithms make inferences from datasets using only input vectors
without referring to known, or labelled, outcomes.
3) The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of
clusters, and repeats the process until it does not find the best clusters. The value of k should be
predetermined in this algorithm.
The k-means clustering algorithm mainly performs two tasks:
□□Determines the best value for K center points or centroids by an iterative process.
□□Assigns each data point to its closest k-center. Those data points which are near to the
particular k-center, create a cluster.
Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

Code:

```
import matplotlib.pyplot as plt

from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()
```

```
X = pd.DataFrame(iris.data,
columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])
y = pd.DataFrame(iris.target, columns=['Targets'])

model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14, 7))

# Real clusters
colormap = np.array(['red','cyan','black'])
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title("Real clusters")

# K Means Cluster
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_],
s=40)
plt.title("K Means Cluster")

plt.show()
```

output:



Result Discussion:

Clustering is a type of unsupervised learning that involves grouping similar data points together.

K-means is a popular clustering algorithm that partitions a dataset into K clusters, where each data point belongs to the cluster with the nearest mean.

1. We visualize the synthetic data to see the clusters.
2. We apply the K-means algorithm to cluster the data into four clusters.
3. We visualize the clustered data along with the cluster centers.

The red 'x' markers represent the cluster centers, and each data point is colored according to the cluster it belongs to. The K-means algorithm iteratively assigns data points to the nearest cluster center and updates the cluster centers until convergence.

Keep in mind that the number of clusters, K, is a hyperparameter that you need to choose based on your understanding of the data and the problem at hand.

In this example, K was set to 4 since we generated the data with 4 centers. However, in real-world scenarios, determining the optimal value of K can be a more involved process, and various techniques

such as the elbow method or silhouette analysis can be employed to assist in this decision.

Practical No 7(a)

Aim: Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix.

Theory:

Hierarchical clustering is primarily used for unsupervised learning and clustering, rather than classification. However, you can use the clusters formed by hierarchical clustering as features for a separate classification model. In this case, we can use the KMeans clusters obtained earlier as features for a classification model.

we're using logistic regression for classification, and KMeans clusters as additional features. The accuracy and confusion matrix are calculated based on the predictions made by the logistic regression model on the test set. Adjust the code as needed based on your specific requirements and dataset.

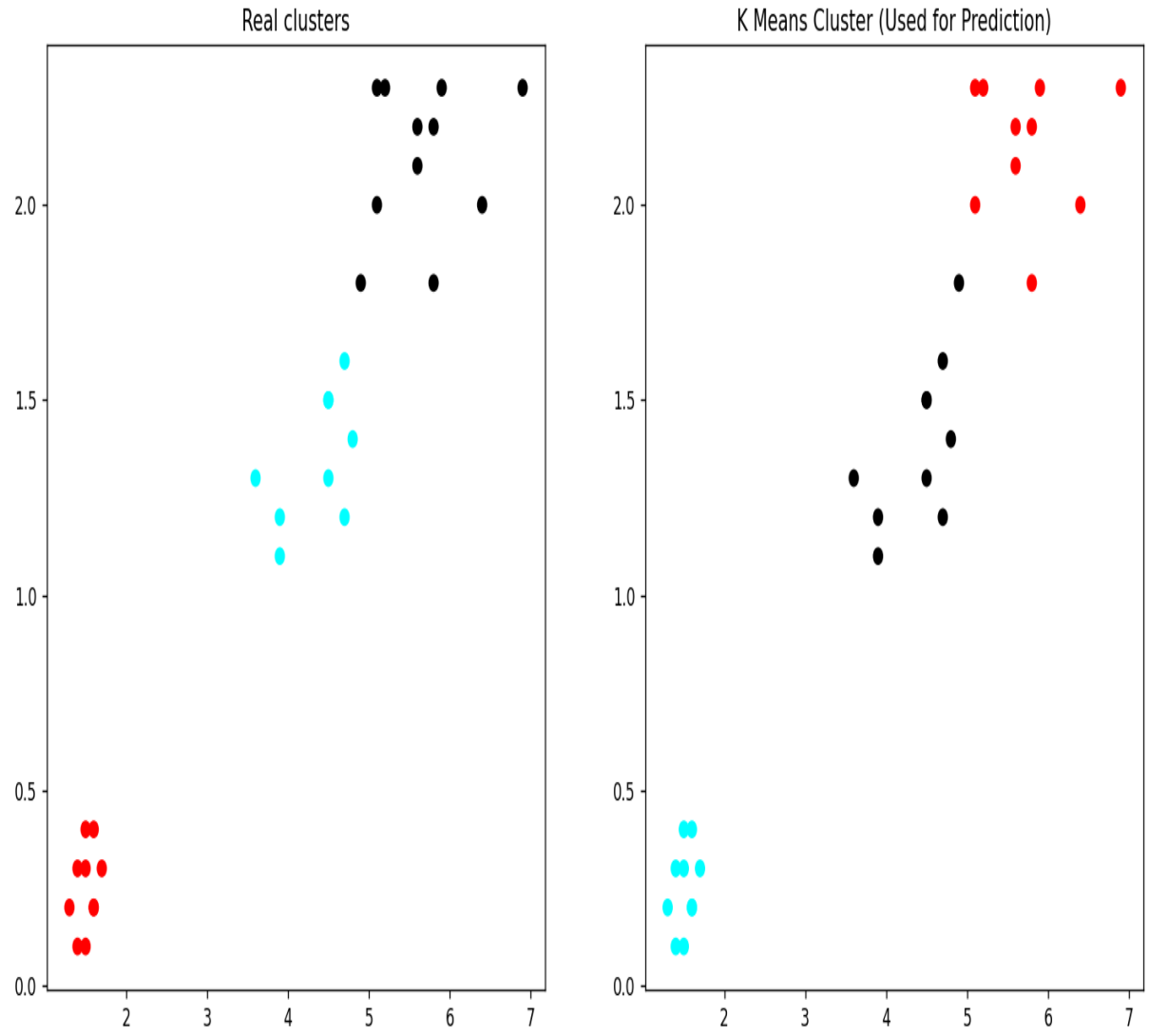
Code:

```
import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix

import pandas as pd

import numpy as np


# Load the Iris dataset

iris = datasets.load_iris()
```

```
X = pd.DataFrame(iris.data,
columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])

y = pd.DataFrame(iris.target, columns=['Targets'])


# Apply KMeans clustering

kmeans_model = KMeans(n_clusters=3)

kmeans_model.fit(X)

X['KMeans_Clusters'] = kmeans_model.labels_


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# Train a logistic regression classifier

classifier = LogisticRegression()

classifier.fit(X_train, y_train.values.ravel())


# Make predictions on the test set

y_pred = classifier.predict(X_test)


# Calculate the accuracy

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
```

```
# Generate and display the confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(conf_matrix)


# Hierarchical clustering doesn't have a direct predict method, so we use the KMeans clusters for prediction.

# The actual hierarchical clustering steps are not performed for prediction in this example.


# Display the scatter plot with KMeans clusters

plt.figure(figsize=(14, 7))

# Real clusters

colormap = np.array(['red','cyan','black'])

plt.subplot(1, 2, 1)

plt.scatter(X_test['Petal_Length'], X_test['Petal_Width'],
c=colormap[y_test['Targets']], s=40)

plt.title("Real clusters")

# K Means Cluster

plt.subplot(1, 2, 2)

plt.scatter(X_test['Petal_Length'], X_test['Petal_Width'],
c=colormap[X_test['KMeans_Clusters']], s=40)

plt.title("K Means Cluster (Used for Prediction)")


plt.show()
```

Output:


Real clusters
K Means Cluster (Used for Prediction)
0.0
0.5
1.0
1.5
2.0
2
3
4
5
6
7

Result Discussion:

1. **Confusion Matrix:**
    - A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.
    - It is particularly useful in binary classification problems, where there are two classes, often referred to as positive and negative.
    - The confusion matrix is a 2x2 table with four entries:
        - True Positive (TP): The model correctly predicted instances of the positive class.
        - True Negative (TN): The model correctly predicted instances of the negative class.
        - False Positive (FP): The model incorrectly predicted instances as the positive class when they are actually negative (Type I error).
        - False Negative (FN): The model incorrectly predicted instances as the negative class when they are actually positive (Type II error).

    The confusion matrix is useful for understanding where a model is making errors and can be used to calculate various performance metrics like precision, recall, accuracy, and F1 score.

2. **Test Score:**
    - The test score is a general term used to describe the overall performance of a machine learning model on a test dataset.
    - The test score could be a measure of accuracy, precision, recall, F1 score, or any other metric that is chosen based on the specific goals of the machine learning task.
    - Accuracy is a commonly used test score, and it is the ratio of correctly predicted instances to the total instances in the test set. It is calculated using the formula:

      Accuracy=Number of Correct PredictionsTotal Number of Predicti onsAccuracy=Total Number of PredictionsNumber of Correct Pred ictions

    - While accuracy is a straightforward metric, it may not be sufficient in cases where class distribution is imbalanced. In such cases, precision, recall, or F1 score might provide a more informative evaluation.

Practical 7(b)

Aim: Implement hierachical clustering using Dendogram.

Theory:

Hierarchical clustering:

1) It is where you build a cluster tree (a dendrogram) to represent data, where each group (or

"node") links to two or more successor groups. The groups are nested and organized as a

tree, which ideally ends up as a meaningful classification scheme.

2) Each node in the cluster tree contains a group of similar data; Nodes group on the graph

next to other, similar nodes. Clusters at one level join with clusters in the next level up,

using a degree of similarity; The process carries on until all nodes are in the tree, which

gives a visual snapshot of the data contained in the whole set. The total number of

clusters is not predetermined before you start the tree creation.

3) A dendrogram is a type of tree diagram showing hierarchical clustering - relationships

between similar sets of data. They are frequently used in biology to show clustering between

genes or samples, but they can represent any type of grouped data.

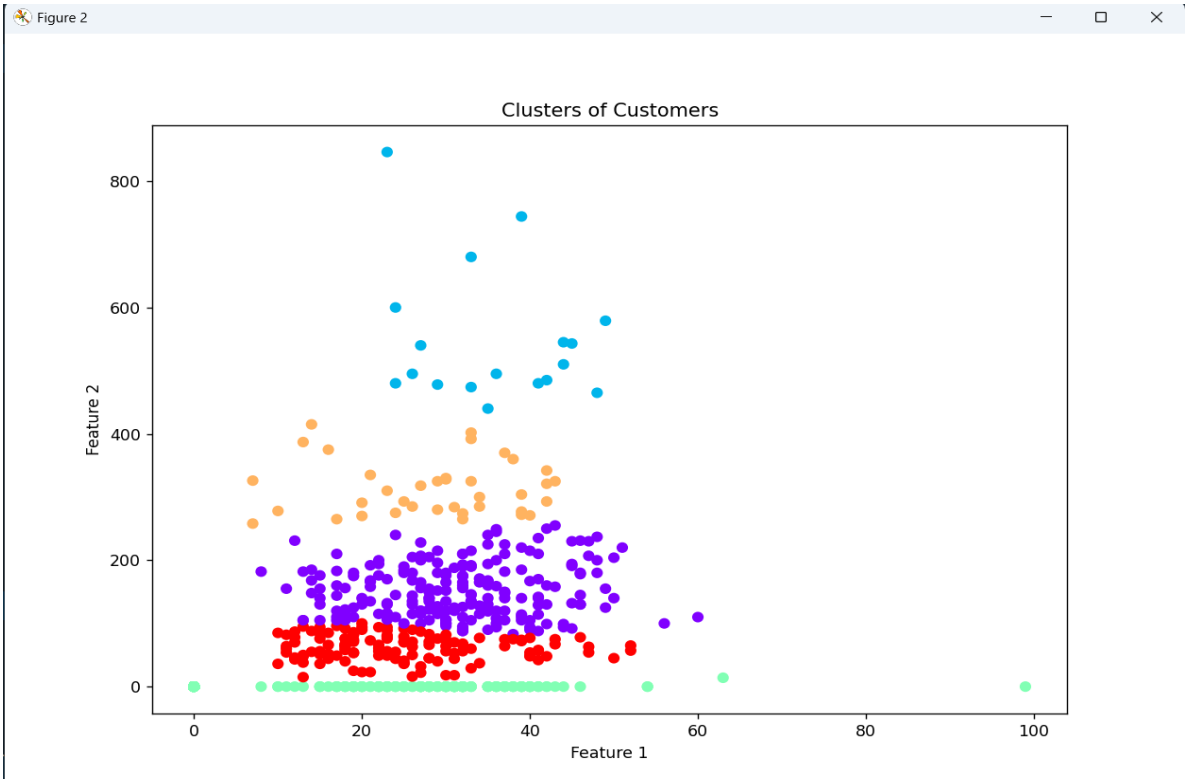
Code:

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import scipy.cluster.hierarchy as shc
```

```
from sklearn.cluster import AgglomerativeClustering


# Load the data
customer_data = pd.read_csv("C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-all-Practicals\\diabetes.csv")


# Display the shape and the first few rows of the data
print(customer_data.shape)
print(customer_data.head())


# Select relevant columns for clustering
data = customer_data.iloc[:, 3:5].values


# Plot dendrogram
plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))
# Perform hierarchical clustering
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster_labels = cluster.fit_predict(data)
# Visualize the clusters
plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='rainbow')
plt.title("Clusters of Customers")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Output:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1



Result Discussion:

Hierarchical clustering is another popular technique in unsupervised machine learning that aims to build a hierarchy of clusters.

The key idea is to create a tree of clusters, known as a dendrogram, where the leaves represent individual data points and the root represents a single cluster containing all data points.

There are two main types of hierarchical clustering:

1. **Agglomerative Hierarchical Clustering:**
    - Start with each data point as a single cluster.
    - Iteratively merge the closest pairs of clusters until only one cluster remains.
2. **Divisive Hierarchical Clustering:**
    - Start with all data points in a single cluster.
    - Iteratively split the cluster into smaller clusters until each cluster contains only one data point.

We use the `linkage` function from to perform agglomerative hierarchical clustering with 'single' linkage.

We visualize the dendrogram, which shows the merging of clusters over the course of the algorithm. The y-axis represents the distance between clusters.

Keep in mind that the choice of linkage method (e.g., 'single', 'complete', 'average') can impact the results, and you may need to choose the method based on the characteristics of your data.

After creating the dendrogram, you can decide how many clusters you want by cutting the dendrogram at a certain height. The vertical lines in the dendrogram indicate the clusters formed by cutting the tree at a specific height.

Hierarchical clustering has the advantage of providing a visual representation of the relationships between data points at different levels of granularity. However, it can be computationally expensive for large datasets.

Practical 8(a)

Aim: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Theory:

1) Bayesian networks are a type of Probabilistic Graphical Model that can be used to build

models from data and/or expert opinion.

2) They can be used for a wide range of tasks including prediction, anomaly detection,

diagnostics, automated insight, reasoning, time series prediction and decision making

under uncertainty.

3) Bayesian networks are probabilistic because they are built from probability distributions

and also use the laws of probability for prediction and anomaly detection, for reasoning and

diagnostics, decision making under uncertainty and time series prediction.

Code:

```
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination


data = pd.read_csv("C:\\Users\\schau\\OneDrive\\Pictures\\Machine-Learning-all-Practicals\\ds4.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)
```

```
model = BayesianNetwork([
    ('age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholestrol'),
    ('Lifestyle', 'diet'),
    ('cholestrol', 'heartdisease'),
    ('diet', 'cholestrol')
])

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)

HeartDisease_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2,
Youth:3, Teen:4')
print('For Gender enter Male:0, Female:1')
print('For Family History enter Yes:1, No:0')
print('For Diet enter High:0, Medium:1')
print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')
print('for Cholesterol enter High:0, BorderLine:1, Normal:2')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age': int(input('Enter Age: ')),
    'Gender': int(input('Enter Gender: ')),
    'Family': int(input('Enter Family History: ')),
    'diet': int(input('Enter Diet: ')),
```

```
    'Lifestyle': int(input('Enter Lifestyle: ')),
    'cholestrol': int(input('Enter Cholestrol: '))
})

print(q)
```

Output:

```
schau/OneDrive/Pictures/Machine-Learning-all-Practicals/prac8a-Bayesian.py
    age  Gender  Family  diet  Lifestyle  cholestrol  heartdisease
0     0       0       1     1          3           0             1
1     0       1       1     1          3           0             1
2     1       0       0     0          2           1             1
3     4       0       1     1          3           2             0
4     3       1       1     0          0           2             0
5     2       0       1     1          1           0             1
6     4       0       1     0          2           0             1
7     0       0       1     1          3           0             1
8     3       1       1     0          0           2             0
9     1       1       0     0          0           2             1
10    4       1       0     1          2           0             1
11    4       0       1     1          3           2             0
12    2       1       0     0          0           0             0
13    2       0       1     1          1           0             1
14    3       1       1     0          0           1             0
15    0       0       1     0          0           2             1
16    1       1       0     1          2           1             1
17    3       1       1     1          0           1             0
18    4       0       1     1          3           2             0
For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4
For Gender enter Male:0, Female:1
For Family History enter Yes:1, No:0
For Diet enter High:0, Medium:1
for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3
for Cholesterol enter High:0, BorderLine:1, Normal:2
Enter Age: 3
Enter Gender: 0
Enter Family History: 1
Enter Diet: 1
Enter Lifestyle: 2
Enter Cholestrol: 2
+-----------------+----------------------+
| heartdisease    |   phi(heartdisease) |
+=================+======================+
| heartdisease(0) |               0.8333 |
+-----------------+----------------------+
| heartdisease(1) |               0.1667 |
+-----------------+----------------------+
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

A Bayesian network, also known as a Bayesian belief network or probabilistic graphical model, is a graphical representation of probabilistic relationships among a set of variables.

It is named after the Bayesian probability theory, which deals with updating beliefs or probabilities based on new evidence.

Bayesian networks are widely used in machine learning and artificial intelligence for modeling uncertainty and making predictions under uncertainty.

Graphical Structure:

- A Bayesian network consists of a directed acyclic graph (DAG) where nodes represent variables, and edges represent probabilistic dependencies between the variables.
- Each node in the graph corresponds to a random variable, and the edges between nodes represent probabilistic dependencies.

Nodes and Conditional Probability Tables (CPTs):

- Nodes in a Bayesian network are associated with probability distributions. Each node's distribution is defined by a Conditional Probability Table (CPT).
- The CPT for a node describes the probability of that node given its parents in the graph. It encapsulates the conditional dependencies between variables.

Conditional Independence:

- One of the key advantages of Bayesian networks is their ability to model and exploit conditional independence relationships between variables. If a node is conditionally independent of its non-descendants given its parents, it can be separated from the rest of the graph.

Practical 8(b)

Aim: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Theory:

Locally Weighted Regression (LWR) is a non-parametric algorithm used for regression tasks where the assumption is that the data can be locally approximated by fitting a regression model to nearby data points. The weights assigned to data points are higher for points closer to the query point and decrease as the distance increases.

Below is a simple implementation of Locally Weighted Regression in Python using NumPy.

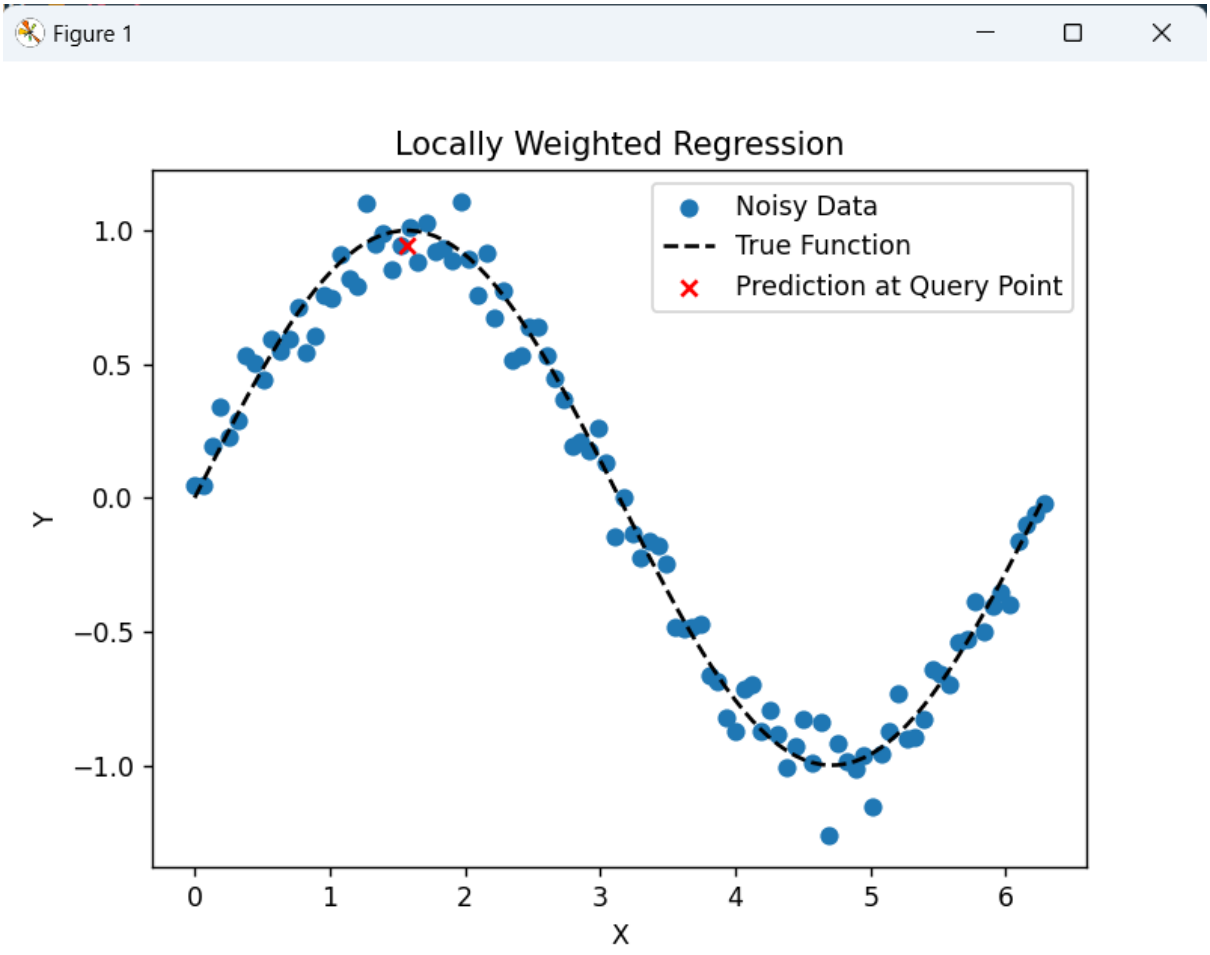
Code:

```
import numpy as np

import matplotlib.pyplot as plt



def locally_weighted_regression(x, y, query_point, tau):

    # Add a bias term to x

    X = np.vstack((np.ones_like(x), x)).T



    # Calculate weights based on the distance between query_point and each data point

    weights = np.exp(-(x - query_point)**2 / (2 * tau**2))



    # Calculate the weighted least squares solution
```

```
    W = np.diag(weights)
    theta = np.linalg.inv(X.T @ W @ X) @ X.T @ W @ y

    # Predict the value at the query point
    query_point_vector = np.array([1, query_point])
    prediction = query_point_vector @ theta

    return prediction

# Generate synthetic data
np.random.seed(42)
x = np.linspace(0, 2 * np.pi, 100)
y_true = np.sin(x)
y_noisy = y_true + np.random.normal(0, 0.1, size=len(x))

# Set the query point and bandwidth parameter (tau)
query_point = np.pi / 2
tau = 0.1

# Perform locally weighted regression
prediction = locally_weighted_regression(x, y_noisy, query_point, tau)

# Plot the results
```

```
plt.scatter(x, y_noisy, label='Noisy Data')

plt.plot(x, y_true, label='True Function', linestyle='--', color='black')

plt.scatter(query_point, prediction, color='red', marker='x', label='Prediction at
Query Point')


plt.title('Locally Weighted Regression')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()
```

Output:



Result Discussion:

Locally Weighted Regression (LWR) is a non-parametric regression algorithm that is used for modeling the relationship between a dependent variable and one or more independent variables.

The key idea behind LWR is to give different weights to different data points based on their proximity to the point where the prediction is being made.

Algorithm Steps:

1. **Input:**
   - Training dataset with pairs (X, y), where X is the input feature vector, and y is the corresponding output.
2. **Initialization:**
   - Choose a query point or a set of query points where predictions need to be made.
3. **Weight Calculation:**
   - Assign weights to the training examples based on their distance from the query point. Commonly used weighting functions include Gaussian or triangular kernels.
   - The weight assigned to each training example decreases as its distance from the query point increases.
4. **Local Model Fitting:**
   - Fit a weighted linear regression model (or any other regression model) using the training examples and their weights in the vicinity of the query point.
   - The model is fitted such that points closer to the query point have a higher influence on the model.
5. **Prediction:**
   - Use the locally fitted model to make predictions for the query point.

Practical 9(a)

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Theory:

An Artificial Neural Network (ANN) with the Backpropagation algorithm involves several key steps. Below is a high-level overview of the process:

1. **Initialization:**
   - Initialize the weights and biases of the neural network randomly. These weights and biases are the parameters that the network will learn during training.
2. **Forward Pass:**
   - Perform a forward pass to compute the predicted output of the neural network for a given input.
   - Apply the activation function at each neuron in the hidden layers and output layer to introduce non-linearity.
   - The output of the network is compared to the true target values using a loss function, which measures the difference between the predicted and actual values.
3. **Backward Pass (Backpropagation):**
   - Calculate the gradient of the loss with respect to the weights and biases using the chain rule of calculus.
   - Update the weights and biases in the opposite direction of the gradient to minimize the loss.
   - The learning rate determines the step size in the weight and bias updates.
4. **Repeat:**
   - Repeat steps 2 and 3 for multiple epochs (iterations over the entire dataset).
   - The network adjusts its weights and biases iteratively to reduce the overall error on the training data.
5. **Training Termination:**
   - Training can be terminated after a fixed number of epochs, or when the performance on a validation set stops improving.
6. **Testing and Inference:**
   - Once trained, the network can be used for making predictions on new, unseen data by performing a forward pass.

Code:

```
import numpy as np


class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights and biases
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_output = np.zeros((1, output_size))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def forward(self, X):
        # Forward pass
        self.hidden_input = np.dot(X, self.weights_input_hidden) +
self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) +
self.bias_output
        self.final_output = self.sigmoid(self.final_input)
        return self.final_output
```

```
    def backward(self, X, y, output):
        # Backpropagation
        error = y - output
        output_delta = error * self.sigmoid_derivative(output)

        error_hidden = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = error_hidden * self.sigmoid_derivative(self.hidden_output)

        # Update weights and biases
        self.weights_hidden_output += self.hidden_output.T.dot(output_delta)
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True)
        self.weights_input_hidden += X.T.dot(hidden_delta)
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True)

    def train(self, X, y, epochs):
        for epoch in range(epochs):
            # Forward and backward pass for each training example
            for i in range(len(X)):
                input_data = X[i:i+1]
                target = y[i:i+1]

                output = self.forward(input_data)
                self.backward(input_data, target, output)

                # Print loss every 100 epochs
                if epoch % 100 == 0:
                    loss = np.mean(np.square(target - output))
```

```
 print(f'Epoch {epoch}, Loss: {loss}')


# Example usage with a simple dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input
y = np.array([[0], [1], [1], [0]])  # Output
# Create a neural network with 2 input neurons, 2 hidden neurons, and 1 output neuron
nn = NeuralNetwork(input_size=2, hidden_size=2, output_size=1)
# Train the neural network for 1000 epochs
nn.train(X, y, epochs=1000)
# Test the trained network
test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
predictions = nn.forward(test_data)
print("Predictions:")
print(predictions)
```

Output:

```
Epoch 300, Loss: 0.25387474062646714
Epoch 300, Loss: 0.2811894637048328
Epoch 400, Loss: 0.2543177785294074
Epoch 400, Loss: 0.2803313648748311
Epoch 400, Loss: 0.25542478718931194
Epoch 400, Loss: 0.28106510761346365
Epoch 500, Loss: 0.2533377501411814
Epoch 500, Loss: 0.26984083809712817
Epoch 500, Loss: 0.26218476964683873
Epoch 500, Loss: 0.2813720677078733
Epoch 600, Loss: 0.24359887108329456
Epoch 600, Loss: 0.21337502546650863
Epoch 600, Loss: 0.3031574455397565
Epoch 600, Loss: 0.2719309067868525
Epoch 700, Loss: 0.15737686809626716
Epoch 700, Loss: 0.06518996242071368
Epoch 700, Loss: 0.4360945322406773
Epoch 700, Loss: 0.1803522979781236
Epoch 800, Loss: 0.09542730604816785
Epoch 800, Loss: 0.02179487959080782
Epoch 800, Loss: 0.4378398385369377
Epoch 800, Loss: 0.16606919986730617
Epoch 900, Loss: 0.044127975930109
Epoch 900, Loss: 0.013792956059770186
Epoch 900, Loss: 0.350286470857974
Epoch 900, Loss: 0.22351388381293508
Predictions:
[[0.14779534]
 [0.90186536]
 [0.45286046]
 [0.47197332]]
PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Artificial Neural Network (ANN)

- An artificial neural network is a computational model composed of interconnected nodes, also known as neurons or perceptrons. It is inspired by the structure and functioning of biological neural networks.
- ANNs are organized into layers, including an input layer, one or more hidden layers, and an output layer.

2. **Layers:**
   - **Input Layer:** Takes in the features or input data.
   - **Hidden Layers:** Intermediate layers that process information and capture patterns.
   - **Output Layer:** Produces the final output or prediction.
3. **Weights and Biases:**
   - Connections between neurons have associated weights, and each neuron has a bias.
   - During training, these weights and biases are adjusted to optimize the network's performance on a specific task.

Backpropagation:

- Backpropagation is a supervised learning algorithm used to train artificial neural networks.
- It involves iteratively adjusting the weights and biases by propagating errors backward through the network.

Process:

- **Forward Pass:** Input data is fed forward through the network to produce a predicted output.
- **Calculate Error:** The difference between the predicted output and the actual output is calculated using a loss or cost function.
- **Backward Pass:** The error is propagated backward through the network.
- **Gradient Descent:** The weights and biases are adjusted in the opposite direction of the gradient of the error, reducing the error and improving the mo

Practical 9(b)

Aim: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.

Theory:

The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem, with the "naive" assumption that features are conditionally independent given the class label. This classifier is often used for text classification tasks, such as spam detection or sentiment analysis.

Let's go through a basic example using Python and the popular scikit-learn library. For simplicity, let's assume we have a set of text documents and corresponding labels (classes) that we want to classify.

Code:

```
# Import necessary libraries
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report


# Sample documents and labels
documents = [
    "This is a positive document.",
    "Negative sentiment is observed here.",
    "I feel positive about this.",
    "I do not like this at all.",
    "This is a negative example.",
]
```

```
labels = ["Positive", "Negative", "Positive", "Negative", "Negative"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(documents, labels,
test_size=0.2, random_state=42)

# Convert text data to a matrix of token counts
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# Train a Naive Bayes classifier
clf = MultinomialNB()
clf.fit(X_train_counts, y_train)

# Make predictions on the test set
predictions = clf.predict(X_test_counts)

# Evaluate the classifier
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("Classification Report:\n", classification_report(y_test, predictions))
```

Output:

```
/Users/schau/OneDrive/Pictures/Machine-Learning-all-Practicals/pract9b.py
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

    Negative       1.00      1.00      1.00         1

    accuracy                           1.00         1
   macro avg       1.00      1.00      1.00         1
weighted avg       1.00      1.00      1.00         1

PS C:\Users\schau\OneDrive\Pictures\Machine-Learning-all-Practicals>
```

Result Discussion:

Naive Bayes Classifier:

1. **Prior probabilities:**
   - Calculate the prior probability of each class P(Ci)P(Ci). This is often estimated by the frequency of each class in the training set.
2. **Likelihood:**
   - For each feature xixi, calculate the likelihood P(xi|C)P(xi|C) of that feature occurring given each class CC. This is often estimated by counting the occurrences of xixi within each class.
3. **Posterior probabilities:**
   - Use Bayes' theorem to calculate the posterior probability of each class given the features: P(C|x1,x2,...,xn)P(C|x1,x2,...,xn).
   - Since we assume feature independence, the likelihood term becomes the product of individual likelihoods: P(x1|C)·P(x2|C)·...·P(xn|C)P(x1|C)·P(x2|C)·...·P(xn|C).
4. **Decision rule:**
   - Assign the class label with the highest posterior probability as the predicted class.

Example:

Let's say you have two classes C1C1 and C2C2, and two features x1x1 and x2x2. Given an observation with values x1=ax1=a and x2=bx2=b, you calculate the posterior probabilities for both classes and predict the class with the higher probability.