1.Aim: Illustrate and Demonstrate the working model and principle of Find-S algorithm.

```
import numpy as np
import pandas as pd  # to read the data in the csv file

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

# Extract attributes and target values
attributes = np.array(data)[:, :-1]  # All rows, all columns except the last
target = np.array(data)[:, -1]  # All rows, only the last column

print("Attributes:\n", attributes)
print("Target:\n", target)

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

# Obtain the final hypothesis
print("\nThe final hypothesis is:", train(attributes, target))
```

2.Aim: Demonstrate the working model and principle of candidate elimination algorithm

```
import numpy as np
import pandas as pd

#to read the data in the csv file
data = pd.read_csv("ENJOYSPORT.csv")
#data = pd.read_csv("Example2.csv")
```

```
print(data,"\n")

concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    #ind=target.tolist().index("Yes")
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in 
range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)


    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', 
'?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

3.Aim: Aim: To construct the Decision tree using the training data sets under supervised learning concept.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn import tree

iris=load_iris()
print(iris.feature_names)
print(iris.target_names)

removed =[0,50,100]
new_target = np.delete(iris.target,removed)
new_data = np.delete(iris.data,removed, axis=0)

clf = tree.DecisionTreeClassifier()
clf=clf.fit(new_data,new_target)
prediction = clf.predict(iris.data[removed])

print("Original Labels",iris.target[removed])
print("Labels Predicted",prediction)
tree.plot_tree(clf)
```

5.Aim: Demonstrate the text classifier using Naïve bayes classifier algorithm.

```
import numpy as np
import pandas as pd

# Loads data
data = pd.read_csv("play_tennis.csv")
# Removes the column "Day" as it is not relevant to the model
data.drop(["day"], axis=1, inplace=True)

def naive_bayes_predict(data, target_name, test_instance):
    valueCounts = data[target_name].value_counts().to_dict()
    pvalue =
(data[data.columns[data.shape[1]1]].value_counts()/data.shape[0]).to_dict()
    pred = {}
    for target, _ in valueCounts.items():
        attribute = 1
        subset = data[data[target_name] == target]
        for attr, value in test_instance.items():
            condProb = subset[subset[attr] == value].shape[0] /
subset.shape[0]
            attribute =attribute* condProb
        prob = pvalue[target] * attribute
        pred[target] = prob
    return pred

    # Creates a test instance
test_instance = {"outlook": "Sunny",
                 "temp": "Cool",
                 "humidity": "High",
                 "wind": "Strong"}
# Performs prediction
prediction = naive_bayes_predict(data.copy(), data.columns[data.shape[1] - 1],
test_instance)
print("Prediction is:",prediction)
```

6.Aim: Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle.

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

data = pd.read_csv("heart.csv")

model = BayesianNetwork([
    ('Age', 'HeartDisease'),
    ('Gender', 'HeartDisease'),
    ('ChestPainType','HeartDisease'),
    ('ExerciseInducedAngina','HeartDisease'),
    ('HeartDisease','RestingECG'),
    ('HeartDisease','Cholesterol')
])

model.fit(data, estimator=MaximumLikelihoodEstimator)
cpd_HeartDisease = MaximumLikelihoodEstimator(model,
data).estimate_cpd('HeartDisease')
inference = VariableElimination(model)

print(inference.query(
    variables=['HeartDisease'],
    evidence={'RestingECG': 1}
    )
```

7.Aim: Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept.

```python
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score

# Generate random data for testing
np.random.seed(0)  # for reproducibility
data = np.random.rand(100, 2)
df = pd.DataFrame(data, columns=['X', 'Y'])

# Save DataFrame to CSV file
df.to_csv('test_data.csv', index=False)

print("CSV file 'test_data.csv' has been generated successfully.")
data = pd.read_csv('test_data.csv')
X = data.values

# K-means clustering
k = KMeans(3)
k.fit(X)
klabel = k.labels_
kcenter = k.cluster_centers_

# EM clustering
em = GaussianMixture(3)
em.fit(X)
elabel = em.predict(X)
ecenter = em.means_

# Compare clustering results
print("K-means labels:")
print(klabel)
print("EM labels:")
print(elabel)

# Visualize clustering results
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=klabel, cmap='viridis')
plt.scatter(kcenter[:, 0], kcenter[:, 1], marker='*', s=300, c='r')
plt.title('K-means Clustering')

plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], c=elabel, cmap='viridis')
plt.scatter(ecenter[:, 0], ecenter[:, 1], marker='*', s=300, c='r')
plt.title('EM Clustering')

plt.show()

# Silhouette Score for K-means
ksil = silhouette_score(X, klabel)
print(f"K-means Silhouette Score: {ksil}")

# Silhouette Score for EM (GMM)
esil = silhouette_score(X, elabel)
print(f"EM (GMM) Silhouette Score: {esil}")
```

The silhouette score is a metric used to evaluate the quality of a clustering algorithm's results. It provides an indication of how well-separated and distinct the clusters are. This metric combines two aspects of cluster quality: cohesion (how close the points in a cluster are to each other) and separation (how far apart the clusters are from each other).
8.Aim: Demonstrate and analyse the results of classification based on KNN Algorithm

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 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=0.3,
random_state=42)

k = 3
knn = KNeighborsClassifier(n_neighbors=k)
# Train the KNN classifier
knn.fit(X_train, y_train)

# Make predictions on the testing set
predictions = knn.predict(X_test)
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
# Print correct and wrong predictions

correct_predictions = []
wrong_predictions = []
for i in range(len(predictions)):
    if predictions[i] == y_test[i]:
       correct_predictions.append((X_test[i], y_test[i], predictions[i]))
    else:
        wrong_predictions.append((X_test[i], y_test[i], predictions[i]))

print("\nCorrect Predictions:")
for prediction in correct_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])

print("\nWrong Predictions:")
for prediction in wrong_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])

# Confusion Matrix

conf_matrix = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:")
print(conf_matrix)
```

A confusion matrix is used in classification problems to evaluate the performance of a classification algorithm. It provides a detailed breakdown of how the algorithm's predictions compare to the actual labels

9.Aim: Understand and analyse the concept of Regression algorithm techniques.

```
import numpy as np
import matplotlib.pyplot as plt

def locally_weighted_regression(x_query, X_train, y_train, tau=0.1):
    m = X_train.shape[0]
    weights = np.exp(-np.sum((X_train - x_query) ** 2, axis=1) / (2 * tau *
tau))
```

```
    W = np.diag(weights)
    theta = np.linalg.inv(X_train.T @ W @ X_train) @ (X_train.T @ W @ y_train)
    prediction = x_query @ theta
    return prediction
# Generate synthetic data for demonstration

np.random.seed(0)
X_train = np.linspace(0, 10, 50)
y_train = np.sin(X_train) + np.random.normal(0, 0.1, X_train.shape[0])

# Query points
X_query = np.linspace(0, 10, 100)

tau = 0.5

predictions = []
for xq in X_query:
    x_query = np.array([1, xq])  # Adding bias term
    prediction = locally_weighted_regression(x_query,
np.c_[np.ones(X_train.shape[0]), X_train], y_train, tau)
    predictions.append(prediction)


plt.figure(figsize=(10, 6))
plt.scatter(X_train, y_train, color='blue', label='Training data')
plt.plot(X_query, predictions, color='red', label='Locally Weighted
Regression')
plt.xlabel('X')
plt.ylabel('Y'
plt.title('Locally Weighted Regression')
plt.legend()
plt.grid(True)
plt.show()
```

10.Aim: Implement and demonstrate classification algorithm using Support vector machine Algorithm.

```
# Import necessary libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

```python
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
svm_classifier.fit(X_train, y_train)

# Calculate accuracy
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Aim: Demonstrate and analyse the results of classification based on KNN Algorithm

```python


import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

predictions = knn.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
```

```
print("Accuracy:", accuracy)

correct_predictions = []
wrong_predictions = []
for i in range(len(predictions)):
    if predictions[i] == y_test[i]:
        correct_predictions.append((X_test[i], y_test[i], predictions[i]))
    else:
        wrong_predictions.append((X_test[i], y_test[i], predictions[i]))

print("\nCorrect Predictions:")
for prediction in correct_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])

print("\nWrong Predictions:")
for prediction in wrong_predictions:
    print("Input:", prediction[0], "Actual Class:",
iris.target_names[prediction[1]], "Predicted Class:",
iris.target_names[prediction[2]])

conf_matrix = confusion_matrix(y_test, predictions)
print("\nConfusion Matrix:")
print(conf_matrix)
```