1. **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.**

import pandas as pd

data = pd.DataFrame({

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes']

})

hypothesis = ['0'] \* (len(data.columns) - 1)

# Apply the FIND-S algorithm

for index, row in data.iterrows():

if row['PlayTennis'] == 'Yes':

for i in range(len(hypothesis)):

if hypothesis[i] == '0':

hypothesis[i] = row.iloc[i]

elif hypothesis[i] != row.iloc[i]:

hypothesis[i] = '?'

print("Most specific hypothesis found by FIND-S algorithm:")

print(hypothesis)

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

import pandas as pd

data = pd.read\_csv("E:/Machine learning/training\_data.csv")

attributes = list(data.columns)[:-1]

examples = data.values

S = ['0'] \* len(attributes)

G = [['?'] \* len(attributes)]

for example in examples:

if example[-1] == 'Yes':

for i in range(len(S)):

if S[i] == '0':

S[i] = example[i]

elif S[i] != example[i]:

S[i] = '?'

G = [g for g in G if all(g[i] == '?' or g[i] == example[i] for i in range(len(g)))]

else:

new\_G = []

for g in G:

for i in range(len(g)):

if g[i] == '?':

new\_h = g.copy()

new\_h[i] = example[i]

if all(new\_h[j] == '?' or new\_h[j] == S[j] for j in range(len(new\_h))):

new\_G.append(new\_h)

G = new\_G

print("Most specific hypothesis (S):", S)

print("Most general hypotheses (G):", G)

1. **Write a program for Implementation of K-Nearest Neighbors (K-NN) in Python**

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

import matplotlib.pyplot as plt

import seaborn as sns

iris = load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.25, random\_state = 0)

model = KNeighborsClassifier(n\_neighbors = 5)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix: ", confusion)

print("Accuracy:" ,accuracy)

plt.figure(figsize=(10,6))

sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',

xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.ylabel("Actual")

plt.xlabel("Confusion Matrix")

plt.show()

1. **Write a program to implement Naïve Bayes algorithm in python**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

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)

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

print("\nAccuracy:", accuracy)

plt.figure(figsize=(10, 7))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues",

xticklabels=iris.target\_names, yticklabels=iris.target\_names)

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

1. **Write a program to implement Logistic Regression (LR) algorithm in python**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

X = X[y != 2]

y = y[y != 2]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LogisticRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

print("\nAccuracy:", accuracy)

plt.figure(figsize=(10, 7))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues",

xticklabels=iris.target\_names[:2], yticklabels=iris.target\_names[:2])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

1. **Write a program to implement Linear Regression (LR) algorithm in python**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

from sklearn.datasets import load\_diabetes

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R^2 Score:", r2)

plt.figure(figsize=(10, 6))

plt.scatter(y\_test, y\_pred, color='blue')

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', lw=2)

plt.xlabel('Actual')

plt.ylabel('Predicted')

plt.title('Actual vs Predicted')

plt.show()

1. **Polynomial Regression In Python**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_diabetes

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

diabetes = load\_diabetes()

X = diabetes.data

y = diabetes.target

X = X[:, 2].reshape(-1, 1) s

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

poly = PolynomialFeatures(degree=3)

X\_poly\_train = poly.fit\_transform(X\_train)

X\_poly\_test = poly.transform(X\_test)

model = LinearRegression()

model.fit(X\_poly\_train, y\_train)

y\_pred = model.predict(X\_poly\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse)

print("R^2 Score:", r2)

plt.figure(figsize=(10, 6))

plt.scatter(X, y, color='blue', label='Data')

X\_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)

X\_range\_poly = poly.transform(X\_range)

y\_range\_poly = model.predict(X\_range\_poly)

plt.plot(X\_range, y\_range\_poly, color='red', linewidth=2, label='Polynomial fit')

plt.xlabel('Feature')

plt.ylabel('Target')

plt.title('Polynomial Regression')

plt.legend()

plt.show()

1. **House Prediction**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

# Load the dataset

data = pd.read\_csv('E:/Machine learning/houseprice.csv')

# Define features and target

X = data[['bedrooms', 'square\_feet', 'year\_built']]

y = data['price']

# Preprocess data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict and evaluate

y\_pred = model.predict(X\_test)

mse = ((y\_test - y\_pred) \*\* 2).mean()

print(f"Mean Squared Error: {mse}")

# Show predictions for test set

results = pd.DataFrame({'Actual': y\_test.values, 'Predicted': y\_pred})

print(results)