S-ALGORIYHM

data = [

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

]

num\_attributes = len(data[0]) - 1

hypothesis = ['ϕ'] \* num\_attributes

for instance in data:

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

for i in range(num\_attributes):

if hypothesis[i] == 'ϕ':

hypothesis[i] = instance[i]

elif hypothesis[i] != instance[i]:

hypothesis[i] = '?'

print("The most specific hypothesis is:", hypothesis)

1)linear regression

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

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

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, 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("Intercept:", model.intercept\_[0])

print("Slope:", model.coef\_[0][0])

print("Mean Squared Error:", mse)

print("R² Score:", r2)

plt.scatter(X\_test, y\_test, color='blue', label='Actual data')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Regression line')

plt.title('Linear Regression: Actual vs Predicted')

plt.xlabel('X')

2)logistic regression

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report

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

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = (4 + 3 \* X + np.random.randn(100, 1) > 8).astype(int).ravel()

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

model = LogisticRegression()

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

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

y\_prob = model.predict\_proba(X\_test)[:, 1]

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

plt.scatter(X\_test, y\_test, color='blue', label='Actual class')

plt.scatter(X\_test, y\_prob, color='red', label='Predicted probability')

plt.title('Logistic Regression: Probability vs Input')

plt.xlabel('X')

plt.ylabel('Probability of Class 1')

plt.legend()

plt.grid(True)

plt.show()

3)Polynomial Regression

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

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

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

np.random.seed(42)

X = 2 \* np.random.rand(100, 1)

y = 5 + 2 \* X + X\*\*2 + np.random.randn(100, 1)

poly = PolynomialFeatures(degree=2)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.2, 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("R² Score:", r2)

X\_plot = np.linspace(0, 2, 100).reshape(100, 1)

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

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

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

plt.plot(X\_plot, y\_plot, color='red', linewidth=2, label='Polynomial regression curve')

plt.title('Polynomial Regression (Degree 2)')

plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.grid(True)

plt.show()

4)KNN

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, classification\_report

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)

k = 3

model = KNeighborsClassifier(n\_neighbors=k)

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

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

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

5)Naive Bayes

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

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)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

6)EM (Expectation Maximization)

import numpy as np

from scipy.stats import norm

data = np.hstack([np.random.normal(0, 1, 100), np.random.normal(5, 1, 100)])

k = 2

means = np.array([0, 5], dtype=float)

variances = np.array([1, 1], dtype=float)

weights = np.array([0.5, 0.5])

for \_ in range(10):

resp = np.array([weights[i] \* norm.pdf(data, means[i], np.sqrt(variances[i])) for i in range(k)])

resp /= resp.sum(axis=0)

for i in range(k):

weights[i] = resp[i].mean()

means[i] = (resp[i] @ data) / resp[i].sum()

variances[i] = (resp[i] @ (data - means[i])\*\*2) / resp[i].sum()

print("Means:", means)

print("Variances:", variances)

print("Weights:", weights)

7)candidate elimination

import csv

def more\_general(h1, h2):

return all(x == '?' or x == y for x, y in zip(h1, h2))

def consistent(hypothesis, example):

return all(h == '?' or h == e for h, e in zip(hypothesis, example))

def candidate\_elimination(examples):

attributes = len(examples[0]) - 1

S = ['Ø'] \* attributes

G = [['?'] \* attributes]

for example in examples:

inputs, output = example[:-1], example[-1]

if output == 'Yes':

G = [g for g in G if consistent(g, inputs)]

for i in range(attributes):

if S[i] == 'Ø':

S[i] = inputs[i]

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

S[i] = '?'

else:

new\_G = []

for g in G:

if consistent(g, inputs):

for i in range(attributes):

if g[i] == '?':

for value in set(e[i] for e in examples if e[-1] == 'Yes'):

if value != inputs[i]:

new\_h = g.copy()

new\_h[i] = value

if more\_general(new\_h, S):

new\_G.append(new\_h)

else:

new\_G.append(g)

G = new\_G

final\_G = []

for g in G:

if not any(more\_general(other, g) and other != g for other in G):

final\_G.append(g)

return S, final\_G

examples = [

['Big', 'Red', 'Circle', 'No'],

['Small', 'Red', 'Triangle', 'No'],

['Big', 'Red', 'Circle', 'Yes'],

['Small', 'Red', 'Circle', 'No'],

['Small', 'Blue', 'Circle', 'Yes'],

]

S, G = candidate\_elimination(examples)

print("Final Specific Hypothesis (S):", S)

print("Final General Hypotheses (G):")

for g in G:

print(g)

8)Perseptron Algorithm

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data[:, :2]

y = iris.target

y = np.where(y == 0, 1, -1)

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

scaler = StandardScaler()

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

X\_test = scaler.transform(X\_test)

perceptron = Perceptron(max\_iter=1000, eta0=0.01, random\_state=42)

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

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

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

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

1Q)

# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LinearRegression

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

# a) Read the house dataset

data = pd.read\_csv('house\_data.csv') # <-- Replace with your CSV file

# b) Print the first five rows

print("First five rows of the dataset:")

print(data.head())

# c) Perform basic statistical computations

print("\nBasic statistics of the dataset:")

print(data.describe())

# Show distribution of the target column (price)

plt.figure(figsize=(8, 5))

sns.histplot(data['price'], kde=True)

plt.title('Distribution of House Prices')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.show()

# d) Print columns and data types

print("\nColumns and data types:")

print(data.dtypes)

# e) Detect and handle null values using mode

print("\nNull values before handling:")

print(data.isnull().sum())

for col in data.columns:

if data[col].isnull().sum() > 0:

data[col].fillna(data[col].mode()[0], inplace=True)

print("\nNull values after handling:")

print(data.isnull().sum())

# f) Explore the dataset using a heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(data.corr(), annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap")

plt.show()

# g) Split the data into training and testing sets

# Separate target and features

X = data.drop(['price'], axis=1)

y = data['price']

# One-hot encode categorical features if any

X = pd.get\_dummies(X)

# Split data

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

# h) Predict the price of a house using Linear Regression

model = LinearRegression()

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

# Predict and evaluate

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

print("\nModel Evaluation:")

print(f"Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred):,.2f}")

print(f"R^2 Score: {r2\_score(y\_test, y\_pred):.2f}")

# Predict for one sample house (first in test set)

sample\_house = X\_test.iloc[0].values.reshape(1, -1)

predicted\_price = model.predict(sample\_house)

print(f"\nPredicted price for the sample house: ${predicted\_price[0]:,.2f}")

1Q b)

import time

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

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)

models = {

"Decision Tree": DecisionTreeClassifier(),

"Logistic Regression": LogisticRegression(max\_iter=200),

"K-Nearest Neighbors": KNeighborsClassifier()

}

results = []

for name, model in models.items():

start\_train = time.time()

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

train\_time = time.time() - start\_train

start\_pred = time.time()

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

pred\_time = time.time() - start\_pred

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

total\_time = train\_time + pred\_time

results.append((name, accuracy, round(train\_time, 4), round(pred\_time, 4), round(total\_time, 4)))

results\_df = pd.DataFrame(results, columns=["Model", "Accuracy", "Train Time (s)", "Predict Time (s)", "Total Time (s)"])

print("\nComparison of Classifiers on Iris Dataset:")

print(results\_df.sort\_values(by="Accuracy", ascending=False))