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)