# ML

1.

import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

# Load the California Housing Dataset

data = fetch\_california\_housing(as\_frame=True)

df = data.frame

# Set up the grid layout

n\_columns = 3

n\_rows = (len(df.select\_dtypes(include=['float64', 'int64']).columns) \* 2 + n\_columns - 1) // n\_columns

fig, axes = plt.subplots(n\_rows, n\_columns, figsize=(15, 10))

axes = axes.flatten()

# Create plots

columns = df.select\_dtypes(include=['float64', 'int64']).columns

for i, column in enumerate(columns):

# Histogram

ax = axes[i]

df[column].hist(bins=30, edgecolor='black', ax=ax)

ax.set(title=f"Histogram of {column}", xlabel=column, ylabel="Frequency")

# Box Plot

ax = axes[len(columns) + i]

df.boxplot(column=column, grid=False, ax=ax)

ax.set(title=f"Box Plot of {column}")

# Adjust layout and display

plt.tight\_layout()

plt.savefig("combined\_plots.png")

plt.show()

2

import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

# Load the dataset

df = fetch\_california\_housing(as\_frame=True).frame

# Set up the grid layout for plots (2 rows, 1 column)

fig, axes = plt.subplots(2, 1, figsize=(12, 12))

# Heatmap of correlation matrix (Top plot)

sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm", cbar=True, ax=axes[0])

axes[0].set\_title("Correlation Matrix Heatmap")

# Pair plot for selected features (Bottom plot)

sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'AveOccup', 'MedHouseVal']], diag\_kind="kde")

plt.subplots\_adjust(hspace=0.4) # Adjust space between subplots

# Show the combined figure

plt.show()

3

import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

# Standardize the features (important for PCA)

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df)

# Apply PCA to reduce to 2 dimensions

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(scaled\_data)

# Create a DataFrame for the 2 principal components

pca\_df = pd.DataFrame(pca\_result, columns=['PC1', 'PC2'])

# Visualize the result

plt.figure(figsize=(8, 6))

plt.scatter(pca\_df['PC1'], pca\_df['PC2'], c=iris.target, cmap='viridis')

plt.title("PCA of Iris Dataset (2 components)")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.colorbar(label='Target')

plt.show()

4

import pandas as pd

# Load the dataset

df = pd.read\_csv('training\_data.csv')

# Assume the last column is the class (target variable)

X = df.iloc[:, :-1] # Features (all columns except the last)

y = df.iloc[:, -1] # Class (the last column)

# Find-S algorithm

def find\_s\_algorithm(X, y):

# Initialize the hypothesis to the most general hypothesis (all attributes can be anything)

hypothesis = ['?' for \_ in range(X.shape[1])]

# Loop through all examples in the dataset

for i in range(len(X)):

if y[i] == 'Yes': # If the example is a positive example

for j in range(len(X.columns)):

# If the hypothesis is still general or the feature matches the example, keep it

if hypothesis[j] == '?' or hypothesis[j] == X.iloc[i, j]:

hypothesis[j] = X.iloc[i, j]

# If the feature doesn't match, make it specific to the example

else:

hypothesis[j] = '?'

return hypothesis

# Get the most specific hypothesis

hypothesis = find\_s\_algorithm(X, y)

# Output the hypothesis

print("Hypothesis consistent with the positive examples:", hypothesis)

5

Python Code:

import matplotlib

matplotlib.use('TkAgg') # Use the TkAgg backend for stable display

import numpy as np

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

# Step 1: Generate 100 random values of x in the range [0, 1]

np.random.seed(42) # For reproducibility

x\_values = np.random.rand(100, 1) # 100 random values in the range [0,1]

# Step 2: Label the first 50 points as Class1 and the rest as Class2

y\_labels = np.array(['Class1' if x <= 0.5 else 'Class2' for x in x\_values.flatten()])

# Split into training and testing sets

X\_train = x\_values[:50] # First 50 points

y\_train = y\_labels[:50] # First 50 labels

X\_test = x\_values[50:] # Remaining 50 points

y\_test = y\_labels[50:] # Remaining 50 labels

# Step 3: Classify using KNN for different k values

k\_values = [1, 2, 3, 4, 5, 20, 30]

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

for i, k in enumerate(k\_values, 1):

# Initialize the k-NN classifier with the current k value

knn = KNeighborsClassifier(n\_neighbors=k)

# Fit the model on the training data

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

# Predict the labels for the test set

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

# Plot the decision boundary and the points

plt.subplot(3, 3, i)

plt.scatter(X\_test, y\_test, color='blue', label='True Label')

plt.scatter(X\_test, y\_pred, color='red', marker='x', label='Predicted Label')

plt.title(f"KNN with k={k}")

plt.xlabel("X value")

plt.ylabel("Class Label")

plt.legend(loc='best')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Step 4: Evaluate classification accuracy for each k value

for k in k\_values:

knn = KNeighborsClassifier(n\_neighbors=k)

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

accuracy = knn.score(X\_test, y\_test)

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

6

import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend for interactive plotting

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

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

# Load the dataset and select a feature

data = fetch\_california\_housing(as\_frame=True)

df = data.frame

X = df['MedInc'].values.reshape(-1, 1)

y = df['MedHouseVal'].values

# Split into training and test sets

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

# Locally Weighted Regression (LWR)

def locally\_weighted\_regression(X\_train, y\_train, X\_test, tau=0.1):

predictions = []

for x in X\_test:

weights = np.exp(-np.sum((X\_train - x) \*\* 2, axis=1) / (2 \* tau \*\* 2))

X\_train\_b = np.c\_[np.ones((X\_train.shape[0], 1)), X\_train]

# Solve the weighted least squares problem using np.linalg.lstsq for efficiency

theta, \_, \_, \_ = np.linalg.lstsq(X\_train\_b \* weights[:, np.newaxis], y\_train \* weights, rcond=None)

# Make prediction for the current test point

X\_test\_b = np.c\_[1, x] # Add bias term

predictions.append(X\_test\_b @ theta)

return np.array(predictions)

# Predict values for the test set

y\_pred = locally\_weighted\_regression(X\_train, y\_train, X\_test, tau=0.1)

# Plot results

plt.scatter(X\_test, y\_test, color='blue', label='True values')

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

plt.xlabel('Median Income')

plt.ylabel('Median House Value')

plt.title('Locally Weighted Regression (LWR)')

plt.legend()

plt.grid(True)

# Show plot (interactive window)

plt.show()

# Evaluate performance

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

print(f"Mean Squared Error: {mse:.4f}")

7

import numpy as np

import pandas as pd

import matplotlib

matplotlib.use('TkAgg') # Use TkAgg backend

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

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

from sklearn.datasets import fetch\_california\_housing

# Load California Housing dataset for Linear Regression

data = fetch\_california\_housing(as\_frame=True)

X = data.data[['AveRooms']]

y = data.target

# Split dataset

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

# Linear Regression

linear\_reg = LinearRegression()

linear\_reg.fit(X\_train, y\_train)

y\_pred = linear\_reg.predict(X\_test)

# Polynomial Regression

poly = PolynomialFeatures(degree=3)

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

poly\_reg = LinearRegression()

poly\_reg.fit(X\_poly, y\_train)

y\_pred\_poly = poly\_reg.predict(poly.fit\_transform(X\_test))

# Plotting results

plt.subplot(1, 2, 1)

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

plt.plot(X\_test, y\_pred, color='red')

plt.title('Linear Regression')

plt.subplot(1, 2, 2)

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

plt.plot(X\_test, y\_pred\_poly, color='green')

plt.title('Polynomial Regression')

plt.tight\_layout()

plt.show()

# Output MSE

print(f"Linear Regression - MSE: {mean\_squared\_error(y\_test, y\_pred):.4f}")

print(f"Polynomial Regression - MSE: {mean\_squared\_error(y\_test, y\_pred\_poly):.4f}")

8

Python Code:

# Import necessary libraries

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

# Load the Breast Cancer dataset

data = load\_breast\_cancer()

X = data.data

y = data.target

# Split the data into training and testing sets

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

# Create a DecisionTreeClassifier instance and train it

clf = DecisionTreeClassifier(random\_state=42)

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

# Predict the test set results

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

# Evaluate the classifier performance

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

print(f'Accuracy on Test Set: {accuracy:.4f}')

# Classify a new sample (randomly selected from the test set for demonstration)

new\_sample = X\_test[0].reshape(1, -1) # Take the first sample from the test set

predicted\_class = clf.predict(new\_sample)

# Output the predicted class (0: malignant, 1: benign)

print(f'Predicted Class for New Sample: {"Benign" if predicted\_class == 1 else "Malignant"}')

9

import numpy as np

from scipy.io import loadmat

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.metrics import accuracy\_score

# Load the olivettifaces.mat file (ensure it's in the same directory or update the path)

data = loadmat('olivettifaces.mat')

# Inspect the keys in the dataset

print("Keys in the dataset:", data.keys())

# Use 'faces' as the feature matrix

X = data['faces'] # Features (faces), this is the matrix of images

# Assuming labels are the index of faces (0-40 for 40 individuals, 10 images per individual)

y = np.repeat(np.arange(40), 10) # 40 classes (individuals), 10 images per class

# Split the data into training and test sets

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

for correct shape

# Create and train the Naive Bayes classifier

model = GaussianNB()

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

# Make predictions

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

# Calculate accuracy

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

print(f"Accuracy: {accuracy:.4f}")

10

import matplotlib

matplotlib.use('TkAgg') # Use the TkAgg backend for interactive GUI rendering

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import load\_breast\_cancer

from sklearn.preprocessing import StandardScaler

# Load the breast cancer dataset

data = load\_breast\_cancer()

X = data.data

y = data.target

# Standardize the data

scaler = StandardScaler()

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

# Apply KMeans clustering

kmeans = KMeans(n\_clusters=2, random\_state=42)

y\_kmeans = kmeans.fit\_predict(X\_scaled)

# Visualize the clustering result

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

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=y\_kmeans, cmap='viridis', edgecolors='k')

plt.title('K-Means Clustering (2D) on Wisconsin Breast Cancer Data')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.colorbar(label='Cluster')

# Show the plot interactively using TkAgg

plt.show()

# Optionally, print cluster centers

print("Cluster centers:\n", kmeans.cluster\_centers\_)