ML

1)

✅ Super simple Python code (FIND-S with CSV):

python Copy

```python
import csv

def find_s_algorithm(filename):
    with open(filename, 'r') as file:
        data = list(csv.reader(file))
        headers = data[0]
        data = data[1:]

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

    for row in data:
        if row[-1] == 'Yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] == '0':
                    hypothesis[i] = row[i]
                elif hypothesis[i] != row[i]:
                    hypothesis[i] = '?'

    print("The final hypothesis is:", hypothesis)

# Example CSV file name
find_s_algorithm('data.csv')
```

✅ Example `data.csv` file content:

css Copy

```
Sky,Temp,Humidity,Wind,Water,Forecast,EnjoySport
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
```

2)

✅ Example `data.csv` (same structure):

css Copy

```
Sky,Temp,Humidity,Wind,Water,Forecast,EnjoySport
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
```

✅ Candidate-Elimination code (clean and easy to write):

python Copy

```python
import csv

def candidate_elimination(filename):
    with open(filename, 'r') as file:
        data = list(csv.reader(file))
        header = data[0]
        data = data[1:]

    S = ['0'] * (len(data[0]) - 1)
    G = [['?'] * (len(data[0]) - 1)]

    for row in data:
        if row[-1] == 'Yes':
            for i in range(len(S)):
                if S[i] == '0':
                    S[i] = row[i]
                elif S[i] != row[i]:
                    S[i] = '?'
            G = [g for g in G if all(g[i] == '?' or g[i] == S[i] for i in range(len(S)))]
        else:
            new_G = []
            for g in G:
                for i in range(len(g)):
                    if g[i] == '?' and S[i] != row[i]:
                        new_hypothesis = g.copy()
                        new_hypothesis[i] = S[i]
                        new_G.append(new_hypothesis)
            G.extend(new_G)

    print("Final S:", S)
    print("Final G:", G)

candidate_elimination('data.csv')
```

3)

✅ Simple ID3 (Decision Tree) code:

python Copy

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

# Example dataset
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny'],
    'Temp': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak'],
    'Play': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes']
}

df = pd.DataFrame(data)

# Converting categorical data into numbers
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in df.columns:
    df[col] = le.fit_transform(df[col])

X = df[['Outlook', 'Temp', 'Humidity', 'Wind']]
y = df['Play']

# Create decision tree model
model = DecisionTreeClassifier(criterion='entropy')
model.fit(X, y)

# Classifying a new sample: Outlook=Rain, Temp=Cool, Humidity=Normal, Wind=Strong
sample = [[le.transform(['Rain'])[0], le.transform(['Cool'])[0], le.transform(['Normal'])[0], le.transform(['Strong'])[0]]]
prediction = model.predict(sample)

print("Classified as:", le.inverse_transform(prediction)[0])
```

4)

✅ 1. Predicting house prices

Problem:

Given house data with size and price, predict price for a new house.

Dataset example:

Size (sqft)	Price (in lakhs)
1000	50
1500	65
2000	80

Code:

python Copy

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[1000], [1500], [2000]])
y = np.array([50, 65, 80])

model = LinearRegression()
model.fit(X, y)

# Predict price for a 1800 sqft house
price = model.predict([[1800]])
print("Predicted price:", price[0])
```

✅ 2. Predicting student marks based on hours studied

Problem:

Predict marks of a student based on hours studied.

Data:

Hours	Marks
2	50
4	65
6	78

Code:

python Copy

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[2], [4], [6]])
y = np.array([50, 65, 78])

model = LinearRegression()
model.fit(X, y)

# Predict marks for 5 hours of study
marks = model.predict([[5]])
print("Predicted marks:", marks[0])
```

✅ 3. Predicting sales based on advertising budget

Problem:

Predict sales based on advertising spend.

Data:

Ad Spend (in $1000)	Sales (in $1000)
10	25
20	45
30	65

Code:

python Copy

```python
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[10], [20], [30]])
y = np.array([25, 45, 65])

model = LinearRegression()
model.fit(X, y)

# Predict sales for $25,000 ad spend
sales = model.predict([[25]])
print("Predicted sales:", sales[0])
```

5)

✅ 1. Predict if a student will pass or fail based on study hours

Problem:

If a student studies for a certain number of hours, predict whether they will pass (1) or fail (0).

Data:

Hours	Result (Pass=1, Fail=0)
2	0
4	0
6	1
8	1

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[2], [4], [6], [8]])
y = np.array([0, 0, 1, 1])

model = LogisticRegression()
model.fit(X, y)

# Predict for 5 hours
result = model.predict([[5]])
print("Predicted (1=Pass, 0=Fail):", result[0])
```

✅ 2. Predict if a customer will buy based on age

Problem:

Given customer ages, predict whether they will buy (1) or not (0).

Data:

Age	Buy (1/0)
22	0
25	0
47	1
52	1

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[22], [25], [47], [52]])
y = np.array([0, 0, 1, 1])

model = LogisticRegression()
model.fit(X, y)

# Predict for age 30
buy = model.predict([[30]])
print("Will buy (1=yes, 0=no):", buy[0])
```

✅ 3. Predict admission based on test scores

Problem:

Predict if a student will get admitted (1) or not (0) based on test scores.

Data:

Test Score	Admission (1/0)
50	0
60	0
85	1
95	1

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[50], [60], [85], [95]])
y = np.array([0, 0, 1, 1])

model = LogisticRegression()
model.fit(X, y)

# Predict admission for score 70
admit = model.predict([[70]])
print("Admitted (1=yes, 0=no):", admit[0])
```

---

6)

✅ 1. Predict if an email is spam or not

Problem:

Based on word frequency in the email, predict spam (1) or not spam (0).

Data:

Word Count (common spam words)	Spam (1/0)
2	0
5	1
7	1
1	0

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[2], [5], [7], [1]])
y = np.array([0, 1, 1, 0])

model = LogisticRegression()
model.fit(X, y)

# Predict for 4 spam words in email
prediction = model.predict([[4]])
print("Spam (1) or Not Spam (0):", prediction[0])
```

✅ 2. Predict whether a transaction is fraudulent

Problem:

Based on transaction amount, predict if fraud (1) or not (0).

Data:

Amount (in $)	Fraud (1/0)
20	0
500	1
1000	1
50	0

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[20], [500], [1000], [50]])
y = np.array([0, 1, 1, 0])

model = LogisticRegression()
model.fit(X, y)

# Predict for a $300 transaction
fraud = model.predict([[300]])
print("Fraud (1) or Not (0):", fraud[0])
```

✅ 3. Predict if a patient has diabetes

Problem:

Based on glucose level, predict diabetes (1) or no diabetes (0).

Data:

Glucose Level	Diabetes (1/0)
80	0
150	1
180	1
90	0

Code:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[80], [150], [180], [90]])
y = np.array([0, 1, 1, 0])

model = LogisticRegression()
model.fit(X, y)

# Predict for glucose level 140
result = model.predict([[140]])
print("Diabetes (1) or Not (0):", result[0])
```

7)

✅ Code for Bias, Variance (conceptual), Remove Duplicates, and Cross Validation:

python Copy

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Sample dataset with duplicates
data = {
    'X': [1, 2, 2, 3, 4, 4, 5],
    'Y': [1.2, 1.9, 1.9, 3.2, 4.1, 4.1, 5.0]
}
df = pd.DataFrame(data)
print("Original Data:\n", df)

# Remove duplicates
df = df.drop_duplicates()
print("\nAfter removing duplicates:\n", df)

X = df[['X']]
y = df['Y']

# Fit linear regression
model = LinearRegression()
model.fit(X, y)

# Conceptual Bias and Variance calculation
predictions = model.predict(X)
bias = np.mean((y - predictions)**2)
variance = np.var(predictions)
print("\nBias (Mean Squared Error):", bias)
print("Variance of predictions:", variance)

# Cross validation
scores = cross_val_score(model, X, y, cv=3, scoring='r2')
print("\nCross Validation R2 scores:", scores)
print("Mean CV Score:", scores.mean())
```

8)

✅ Code for One-hot Encoding:

python Copy

```python
import pandas as pd

# Sample data
data = {
    'Color': ['Red', 'Blue', 'Green', 'Blue', 'Red']
}

df = pd.DataFrame(data)
print("Original Data:\n", df)

# Apply one-hot encoding
encoded_df = pd.get_dummies(df)
print("\nOne-hot Encoded Data:\n", encoded_df)
```

✅ Output Example:

mathematica Copy

```
Original Data:
   Color
0    Red
1   Blue
2  Green
3   Blue
4    Red

One-hot Encoded Data:
   Color_Blue  Color_Green  Color_Red
0           0            0          1
1           1            0          0
2           0            1          0
3           1            0          0
4           0            0          1
```

9)

✅ Program to implement Categorical Encoding:

python Copy

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Sample data
data = {
    'Fruit': ['Apple', 'Banana', 'Mango', 'Banana', 'Apple']
}
df = pd.DataFrame(data)
print("Original Data:\n", df)

# Label Encoding
le = LabelEncoder()
df['Fruit_Label'] = le.fit_transform(df['Fruit'])
print("\nLabel Encoded Data:\n", df)

# One-hot Encoding
one_hot_encoded = pd.get_dummies(df['Fruit'])
print("\nOne-hot Encoded Data:\n", one_hot_encoded)
```

✅ Output Example:

yaml Copy

```
Original Data:
    Fruit
0  Apple
1 Banana
2  Mango
3 Banana
4  Apple

Label Encoded Data:
    Fruit  Fruit_Label
0  Apple            0
1 Banana            1
2  Mango            2
3 Banana            1
4  Apple            0

One-hot Encoded Data:
   Apple  Banana  Mango
0      1       0      0
1      0       1      0
2      0       0      1
3      0       1      0
4      1       0      0
```

10)

✅ Code to Build an ANN with Backpropagation:

python Copy

```python
import numpy as np

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Input dataset (XOR problem)
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])

# Output dataset
y = np.array([[0],
              [1],
              [1],
              [0]])

# Initialize weights randomly
input_neurons = 2
hidden_neurons = 2
output_neurons = 1

np.random.seed(1)
weights_input_hidden = np.random.uniform(size=(input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_neurons, output_neurons))

# Learning rate
lr = 0.5

# Train the network
for epoch in range(10000):
    # Forward pass
    hidden_layer_input = np.dot(X, weights_input_hidden)
    hidden_layer_output = sigmoid(hidden_layer_input)

    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
    output = sigmoid(output_layer_input)

    # Backpropagation
    error = y - output
    d_output = error * sigmoid_derivative(output)

    error_hidden = d_output.dot(weights_hidden_output.T)
    d_hidden = error_hidden * sigmoid_derivative(hidden_layer_output)

    # Update weights
    weights_hidden_output += hidden_layer_output.T.dot(d_output) * lr
    weights_input_hidden += X.T.dot(d_hidden) * lr

# Test
print("Trained output:\n", output.round())
```

✅ Output (approx.)

lua Copy

```lua
Trained output:
 [[0.]
 [1.]
 [1.]
 [0.]]
```

11)

✅ Program to implement k-NN on Iris dataset:

python Copy

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Create and train the k-NN model (k=3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Predict on test data
y_pred = knn.predict(X_test)

# Print correct and wrong predictions
print("Correct predictions:")
for actual, predicted in zip(y_test, y_pred):
    if actual == predicted:
        print(f"Actual: {actual}, Predicted: {predicted}")

print("\nWrong predictions:")
for actual, predicted in zip(y_test, y_pred):
    if actual != predicted:
        print(f"Actual: {actual}, Predicted: {predicted}")
```

✅ Output Example:

yaml Copy

```yaml
Correct predictions:
Actual: 0, Predicted: 0
Actual: 1, Predicted: 1
Actual: 2, Predicted: 2
...

Wrong predictions:
Actual: 2, Predicted: 1
```

12)

✅ Program for Locally Weighted Regression (LWR):

python Copy

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

# Sample dataset (simple linear pattern with slight noise)
X = np.linspace(1, 10, 10)
y = 2 * X + np.random.randn(10)

# Add bias term
X_mat = np.c_[np.ones(X.shape[0]), X]

# Locally Weighted Regression Function
def lwlr(test_point, X, y, tau=0.5):
    m = X.shape[0]
    W = np.eye(m)
    for i in range(m):
        diff = test_point - X[i]
        W[i, i] = np.exp(-(diff @ diff.T) / (2 * tau ** 2))
    theta = np.linalg.pinv(X.T @ W @ X) @ (X.T @ W @ y)
    return test_point @ theta

# Predict points
pred_y = []
for point in X_mat:
    pred = lwlr(point, X_mat, y)
    pred_y.append(pred)

# Plotting
plt.scatter(X, y, color='blue', label="Original Data")
plt.plot(X, pred_y, color='red', label="LWR Fit")
plt.xlabel('X')
plt.ylabel('y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.show()
```

✅ How this works:

- We manually compute local weights based on each test point's distance from all training points.
- `tau` is the bandwidth (smaller `tau` means more weight to nearby points).
- Predictions are made for each point using weighted least squares.
