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.**

Program :

```
import csv

file='training.csv'
with open(file,'r') as f:
   data=list(csv.reader(f))
   headers=data[0]
   concepts=[row[:-1] for row in data[1:]]
   target=[row[-1] for row in data[1:]]

hypothesis=['?' for _ in range(len(concepts[0]))]

for i, val in enumerate(concepts):
   if target[i]=='Yes':
      if hypothesis[0]=='?':
         hypothesis=val.copy()
      else:
         for j in range(len(hypothesis)):
            if hypothesis[j]!=val[j]:
               hypothesis[j]='?'

print('Final Hypothesis:',hypothesis)
```

Output :

```
Final Hypothesis: ['Sunny', 'Warm', 'High', 'Strong', 'Cold', 'Change']
```

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

Program :

```
import pandas as pd

data = pd.read_csv("CE_DS1.csv")

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

S = ['Ø'] * len(X[0])
G = ['?'] * len(X[0])


for i in range(len(X)):
   if y[i] == 'Yes':      # Positive example
      for j in range(len(S)):
         if S[j] == 'Ø':
            S[j] = X[i][j]
         elif S[j] != X[i][j]:
            S[j] = '?'
   else:                  # Negative example
      for j in range(len(G)):
         if G[j] == '?' and S[j] != X[i][j]:
            G[j] = S[j]


print("Specific Boundary (S):", S)
print("General Boundary (G):", G)
```

Output :

Specific Boundary (S): ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General Boundary (G): ['Sunny', 'Warm', '?', '?', '?', 'Same']

3. Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Program :

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

data = pd.read_csv("DATA.csv")

kmeans = KMeans(n_clusters=2, random_state=0)
kmeans_labels = kmeans.fit_predict(data)

em = GaussianMixture(n_components=2, random_state=0)
em_labels = em.fit_predict(data)

kmeans_score = silhouette_score(data, kmeans_labels)
em_score = silhouette_score(data, em_labels)

print("k-Means Silhouette Score:", kmeans_score)
print("EM Silhouette Score:", em_score)

if em_score > kmeans_score:
   print("EM algorithm produces better clustering.")
else:
   print("k-Means produces comparable clustering.")
```

Output :

k-Means Silhouette Score: 0.7676359143284108
EM Silhouette Score: 0.7676359143284108
k-Means produces comparable clustering.

4. Analyse discretization by considering data as ages and find the bin values.

Program :

```
import pandas as pd

ages = [12, 15, 18, 22, 25, 30, 34, 40, 45, 50, 55, 60, 65, 70]
df = pd.DataFrame({'Age': ages})

print("Original Age Data:")
print(df)


num_bins = 4
df['EW_Bin'] = pd.cut(df['Age'], bins=num_bins)
print("\nEqual-Width Binning:")
print(df[['Age', 'EW_Bin']])


df['EF_Bin'] = pd.qcut(df['Age'], q=4)
print("\nEqual-Frequency Binning:")
print(df[['Age', 'EF_Bin']])


bins = [0, 12, 19, 59, 100]
labels = ['Child', 'Teen', 'Adult', 'Senior']
df['Custom_Bin'] = pd.cut(df['Age'], bins=bins, labels=labels)

print("\nCustom Binning:")
print(df[['Age', 'Custom_Bin']])
```

Output :

Original Age Data:

	Age
0	12
1	15
2	18
3	22
4	25
5	30
6	34
7	40
8	45
9	50
10	55
11	60
12	65
13	70

Equal-Width Binning:

	Age	EW_Bin
0	12	(11.942, 26.5]
1	15	(11.942, 26.5]
2	18	(11.942, 26.5]
3	22	(11.942, 26.5]
4	25	(11.942, 26.5]
5	30	(26.5, 41.0]
6	34	(26.5, 41.0]
7	40	(26.5, 41.0]
8	45	(41.0, 55.5]
9	50	(41.0, 55.5]
10	55	(41.0, 55.5]
11	60	(55.5, 70.0]
12	65	(55.5, 70.0]
13	70	(55.5, 70.0]

Equal-Frequency Binning:

	Age	EF_Bin
0	12	(11.999, 22.75]
1	15	(11.999, 22.75]
2	18	(11.999, 22.75]
3	22	(11.999, 22.75]
4	25	(22.75, 37.0]
5	30	(22.75, 37.0]
6	34	(22.75, 37.0]
7	40	(37.0, 53.75]
8	45	(37.0, 53.75]
9	50	(37.0, 53.75]
10	55	(53.75, 70.0]
11	60	(53.75, 70.0]
12	65	(53.75, 70.0]
13	70	(53.75, 70.0]

Custom Binning:

	Age	Custom_Bin
0	12	Child
1	15	Teen
2	18	Teen
3	22	Adult
4	25	Adult
5	30	Adult
6	34	Adult
7	40	Adult
8	45	Adult
9	50	Adult
10	55	Adult
11	60	Senior
12	65	Senior
13	70	Senior

5. Write a machine learning program to print a confusion matrix.

Program :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

df = pd.read_csv("spam.csv", encoding="latin-1")[['v1', 'v2']]
df.columns = ['label', 'message']

df['label_num'] = df['label'].map({'ham': 0, 'spam': 1})

X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label_num'],
test_size=0.2, random_state=42)

vectorizer = CountVectorizer()
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_vectors, y_train)

y_pred = model.predict(X_test_vectors)

print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n",classification_report(y_test, y_pred))
```

Output :

Confusion Matrix:
[[964 1]
[24 126]]

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       965
           1       0.99      0.84      0.91       150

    accuracy                           0.98      1115
   macro avg       0.98      0.92      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

6. Write a program to implement Bayes classifier by considering input as fruit and calculate the entropy and gini.

Program :

```
import math
from collections import Counter

data = [
   ("Red", "Big", "Apple"),
   ("Red", "Small", "Apple"),
   ("Yellow", "Big", "Banana"),
   ("Yellow", "Small", "Banana"),
   ("Green", "Big", "Apple")
]

def entropy(data):
   labels = Counter(row[2] for row in data)
   total = len(data)
   return -sum((c/total) * math.log2(c/total) for c in labels.values())

def gini(data):
   labels = Counter(row[2] for row in data)
   total = len(data)
   return 1 - sum((c/total) ** 2 for c in labels.values())

def bayes_predict(test):
   classes = Counter(row[2] for row in data)
   probs = {}

   for c in classes:
      probs[c] = classes[c] / len(data)
      for i in range(len(test)):
         match = sum(1 for row in data if row[i] == test[i] and row[2] == c)
         probs[c] *= (match + 1) / (classes[c] + 2)
   return max(probs, key=probs.get)

test = ("Yellow", "Big")
print("Predicted Fruit:", bayes_predict(test))
```

```
print("Entropy:", entropy(data))
print("Gini Index:", gini(data))
```

Output :

```
Predicted Fruit: Banana
Entropy: 0.9709505944546686
Gini Index: 0.48
```

7. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print correct predictions. Java/Python ML library classes can be used for this problem.

Program :

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

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

pred = model.predict(X_test)

print("Correct Predictions:")
for i in range(len(pred)):
   if pred[i] == y_test[i]:
      print("Actual:", y_test[i], "Predicted:", pred[i])
```

Output :

```
Correct Predictions:
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 2 Predicted: 2
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0
```

Actual: 2 Predicted: 2
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 2 Predicted: 2
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 2 Predicted: 2
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 0 Predicted: 0
Actual: 1 Predicted: 1
Actual: 1 Predicted: 1
Actual: 2 Predicted: 2
Actual: 1 Predicted: 1
Actual: 0 Predicted: 0

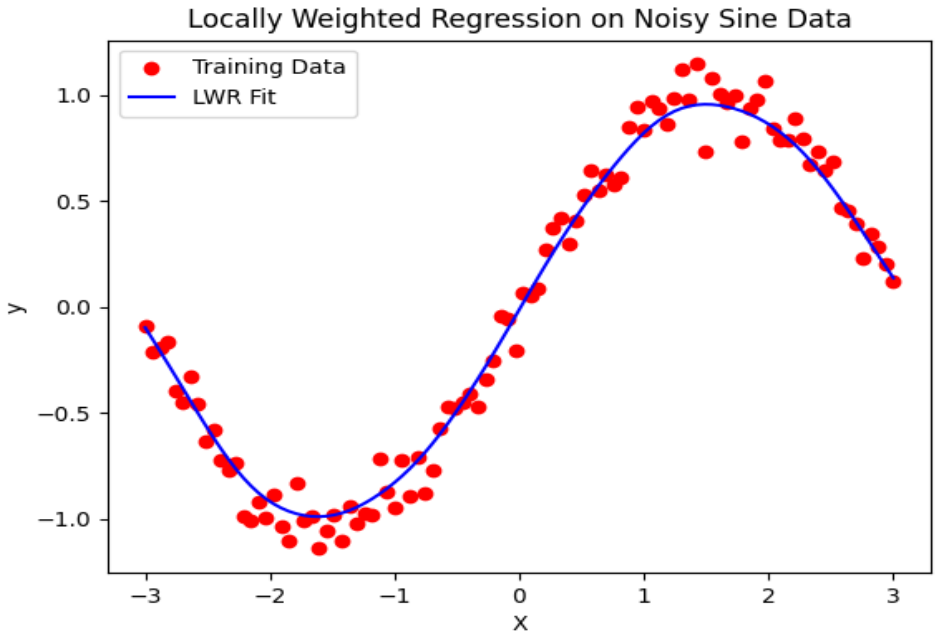
- **0 → Iris-setosa**
- **1 → Iris-versicolor**
- **2 → Iris-virginica**

8. Implement the non-parametric Locally Weighted Regression algorithm to fit data points.

Program :

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

def lwr(X, y, xq, tau):
    m = X.shape[0]
    W = np.eye(m)

    for i in range(m):
        d = X[i] - xq
        W[i, i] = np.exp(-(d @ d) / (2 * tau**2))

    theta = np.linalg.pinv(X.T @ W @ X) @ X.T @ W @ y
    return xq @ theta


# Data
X = np.linspace(-3, 3, 100).reshape(-1, 1)
y = np.sin(X) + np.random.normal(0, 0.1, (100, 1))

X = np.c_[np.ones(100), X]
X_test = np.c_[np.ones(100), np.linspace(-3, 3, 100).reshape(-1, 1)]

# Prediction
tau = 0.3
y_pred = np.array([lwr(X, y, x, tau) for x in X_test]).ravel()

# Plot
plt.scatter(X[:, 1], y, color='red', label="Data")
plt.plot(X_test[:, 1], y_pred, color='blue', label="LWR Fit")
plt.legend()
plt.show()
```

Output :


Locally Weighted Regression on Noisy Sine Data
Training Data
LWR Fit
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
X

9. Write a program to implement Multiple Regression algorithm to print correct predictions. Python ML library classes can be used for this problem.

Program :

```
from sklearn.linear_model import LinearRegression
import pandas as pd

data = pd.DataFrame({
   'Age': [45, 50, 60, 35, 40],
   'BP': [120, 130, 140, 110, 115],
   'Chol': [200, 220, 250, 180, 190],
   'Risk': [1, 1, 1, 0, 0]
})

X = data[['Age', 'BP', 'Chol']]
y = data['Risk']

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

prediction = model.predict([[55, 135, 230]])

print("Predicted Value:", prediction)

if prediction[0] >= 0.5:
   print("Prediction: Disease Present")
else:
   print("Prediction: No Disease")
```

Output :

```
Predicted Value: [1.5]
Prediction: Disease Present
```

10. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Program :

```
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
import pandas as pd

data = pd.DataFrame({
   'Age': ['old', 'old', 'young', 'young'],
   'BP': ['high', 'normal', 'high', 'normal'],
   'Chol': ['high', 'high', 'normal', 'normal'],
   'HeartDisease': ['yes', 'yes', 'no', 'no']
})

model = DiscreteBayesianNetwork([
   ('Age', 'HeartDisease'),
   ('BP', 'HeartDisease'),
   ('Chol', 'HeartDisease')
])

model.fit(data, estimator=MaximumLikelihoodEstimator)

inference = VariableElimination(model)

result = inference.query(
   variables=['HeartDisease'],
   evidence={'Age': 'old', 'BP': 'high', 'Chol': 'high'}
)
print(result)
```

Output :

HeartDisease	phi(HeartDisease)
HeartDisease(no)	0.0000
HeartDisease(yes)	1.0000