B M S COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

Post Box No.: 1908, Bull Temple Road, Bengaluru – 560019

DEPARTMENT OF MACHINE LEARNING

Academic Year: 2022-2023

ADVANCED MACHINE LEARNING(22AM6PCAML)

Lab 3- AML Laboratory Programs

Submitted by

Student Name	Ashish D D
USN	1BM20AI015
Date	24-05-2023
Semester & Section	6th Sem, A
Student Signature	

Valuation Report **(to be filled by the faculty)**

Score:		Faculty In-charge:	**Dr. Seemanthini K Gowda**
Comments:		Faculty Signature: with date	

1. **Construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.**

```
In [ ]: !pip install pgmpy
```

```
In [ ]: import pandas as pd
        from pgmpy.models import BayesianNetwork
        from pgmpy.estimators import MaximumLikelihoodEstimator
        from pgmpy.inference import VariableElimination
```

```
In [ ]: # Load the Heart Disease Data Set (replace with your dataset)
        data = pd.read_csv('heart dis.csv')
```

```
In [43]: print(data)

     Age  Gender         ChestPain BloodPressure Cholesterol ECGResult  \
0  30-39    Male    Typical Angina          High        High    Normal
1  40-49  Female   Atypical Angina        Normal        High  Abnormal
2  50-59    Male  Non-Anginal Pain        Normal      Normal    Normal
3  60-69  Female      Asymptomatic          High      Normal  Abnormal
4  40-49  Female   Atypical Angina        Normal        High    Normal
5  50-59    Male    Typical Angina          High        High  Abnormal
6  60-69    Male  Non-Anginal Pain        Normal      Normal    Normal

   ExerciseAngina  STDepression  Diagnosis
0            True           2.0          1
1           False           0.5          0
2            True           1.0          1
3            True           3.0          1
4           False           0.0          0
5            True           2.5          1
6           False           0.0          0
```

```
In [46]: # Define the variables and their dependencies
         model = BayesianNetwork([
             ('Age', 'ChestPain'),
             ('Gender', 'ChestPain'),
             ('Age', 'BloodPressure'),
             ('Gender', 'BloodPressure'),
             ('Age', 'Cholesterol'),
             ('Gender', 'Cholesterol'),
             ('Age', 'ECGResult'),
             ('ChestPain', 'ECGResult'),
             ('ChestPain', 'ExerciseAngina'),
             ('ECGResult', 'ExerciseAngina'),
             ('ExerciseAngina', 'STDepression'),
             ('STDepression', 'Diagnosis')
         ])

         # Estimate the parameters (probabilities) using Maximum Likelihood Estimation
         model.fit(data, estimator=MaximumLikelihoodEstimator)

         # Perform inference on the model
         infer = VariableElimination(model)

         # Query the model for the posterior probabilities
         query = infer.query(variables=['Diagnosis'], evidence={
             'Age': '50-59',
             'Gender': 'Male',
             'ChestPain': 'Typical Angina',
             'BloodPressure': 'High',
             'Cholesterol': 'High',
             'ECGResult': 'Abnormal',
             'ExerciseAngina': True,
             'STDepression': 2.5
         })

         # Print the posterior probabilities
         print(query)

+--------------+------------------+
| Diagnosis    |   phi(Diagnosis) |
+==============+==================+
| Diagnosis(0) |           0.0000 |
+--------------+------------------+
| Diagnosis(1) |           1.0000 |
+--------------+------------------+
```

Application:

- Medical diagnosis: Bayesian networks are widely used in medical diagnosis systems to model the dependencies between symptoms, diseases, and test results.

- Risk assessment: Bayesian networks can be used to model and assess risks in various domains, such as finance, insurance, and engineering.
- Natural language processing: Bayesian networks have been applied in tasks such as text categorization, sentiment analysis, and information retrieval.

2. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
In [1]: # Importing libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

        # Importing data
        data = pd.read_csv('Breast cancer.csv')
        del data['Unnamed: 32']
```

```
In [2]: X = data.iloc[:, 2:].values
        y = data.iloc[:, 1].values

        # Encoding categorical data
        from sklearn.preprocessing import LabelEncoder
        labelencoder_X_1 = LabelEncoder()
        y = labelencoder_X_1.fit_transform(y)

        # Splitting the dataset into the Training set and Test set
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 0)

        #Feature Scaling
        from sklearn.preprocessing import StandardScaler
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.transform(X_test)
```

```
In [4]: !pip install keras

        Collecting keras
          Downloading keras-2.12.0-py2.py3-none-any.whl (1.7 MB)
             ---------------------------------------- 1.7/1.7 MB 5.8 MB/s eta 0:00:00
        Installing collected packages: keras
        Successfully installed keras-2.12.0
```

```
In [5]: import keras
        from keras.models import Sequential
        from keras.layers import Dense, Dropout
```

```
In [6]: # Initialising the ANN
        classifier = Sequential()
```

```
In [7]:
        classifier.add(Dense(units=16, kernel_initializer='uniform', activation='relu', input_dim=30))
        # Adding dropout to prevent overfitting
        classifier.add(Dropout(rate=0.1))
```

```
In [8]: # Adding the second hidden layer
        classifier.add(Dense(units=16, kernel_initializer='uniform', activation='relu'))

        # Adding dropout to prevent overfitting
        classifier.add(Dropout(rate=0.1))
```

```
In [9]: # Adding the output layer
        classifier.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
```

```
In [10]: # Compiling the ANN
         classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [11]: # Fitting the ANN to the Training set
         classifier.fit(X_train, y_train, batch_size=100, epochs=50)
         # Long scroll ahead but worth
         # The batch size and number of epochs have been set using trial and error. Still looking for more efficient ways. Open to suggest
```

```
Epoch 1/50
6/6 [==============================] - 2s 5ms/step - loss: 0.6926 - accuracy: 0.6094
Epoch 2/50
6/6 [==============================] - 0s 3ms/step - loss: 0.6902 - accuracy: 0.6406
Epoch 3/50
6/6 [==============================] - 0s 3ms/step - loss: 0.6869 - accuracy: 0.6816
Epoch 4/50
6/6 [==============================] - 0s 3ms/step - loss: 0.6817 - accuracy: 0.7637
Epoch 5/50
6/6 [==============================] - 0s 4ms/step - loss: 0.6730 - accuracy: 0.8457
Epoch 6/50
6/6 [==============================] - 0s 4ms/step - loss: 0.6602 - accuracy: 0.9043
Epoch 7/50
6/6 [==============================] - 0s 4ms/step - loss: 0.6409 - accuracy: 0.9199
Epoch 8/50
6/6 [==============================] - 0s 4ms/step - loss: 0.6156 - accuracy: 0.9375
Epoch 9/50
6/6 [==============================] - 0s 4ms/step - loss: 0.5830 - accuracy: 0.9414
Epoch 10/50
6/6 [==============================] - 0s 4ms/step - loss: 0.5418 - accuracy: 0.9414
Epoch 11/50
6/6 [==============================] - 0s 4ms/step - loss: 0.4900 - accuracy: 0.9414
Epoch 12/50
6/6 [==============================] - 0s 4ms/step - loss: 0.4393 - accuracy: 0.9395
Epoch 13/50
6/6 [==============================] - 0s 3ms/step - loss: 0.3886 - accuracy: 0.9336
Epoch 14/50
6/6 [==============================] - 0s 3ms/step - loss: 0.3348 - accuracy: 0.9395
Epoch 15/50
6/6 [==============================] - 0s 3ms/step - loss: 0.2933 - accuracy: 0.9434
Epoch 16/50
6/6 [==============================] - 0s 4ms/step - loss: 0.2570 - accuracy: 0.9453
Epoch 17/50
6/6 [==============================] - 0s 3ms/step - loss: 0.2302 - accuracy: 0.9512
Epoch 18/50
6/6 [==============================] - 0s 3ms/step - loss: 0.2012 - accuracy: 0.9531
Epoch 19/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1856 - accuracy: 0.9609
Epoch 20/50
```

```
Epoch 21/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1550 - accuracy: 0.9707
Epoch 22/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1427 - accuracy: 0.9688
Epoch 23/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1312 - accuracy: 0.9648
Epoch 24/50
6/6 [==============================] - 0s 4ms/step - loss: 0.1272 - accuracy: 0.9668
Epoch 25/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1243 - accuracy: 0.9707
Epoch 26/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1194 - accuracy: 0.9668
Epoch 27/50
6/6 [==============================] - 0s 4ms/step - loss: 0.1103 - accuracy: 0.9727
Epoch 28/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1024 - accuracy: 0.9746
Epoch 29/50
6/6 [==============================] - 0s 3ms/step - loss: 0.1007 - accuracy: 0.9766
Epoch 30/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0965 - accuracy: 0.9766
Epoch 31/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0945 - accuracy: 0.9785
Epoch 32/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0914 - accuracy: 0.9785
Epoch 33/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0878 - accuracy: 0.9805
Epoch 34/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0894 - accuracy: 0.9727
Epoch 35/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0835 - accuracy: 0.9785
Epoch 36/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0802 - accuracy: 0.9785
Epoch 37/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0780 - accuracy: 0.9844
Epoch 38/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0788 - accuracy: 0.9844
Epoch 39/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0790 - accuracy: 0.9785
Epoch 40/50
```

```
Epoch 41/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0733 - accuracy: 0.9805
Epoch 42/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0712 - accuracy: 0.9805
Epoch 43/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0775 - accuracy: 0.9805
Epoch 44/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0705 - accuracy: 0.9824
Epoch 45/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0684 - accuracy: 0.9824
Epoch 46/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0722 - accuracy: 0.9805
Epoch 47/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0691 - accuracy: 0.9824
Epoch 48/50
6/6 [==============================] - 0s 3ms/step - loss: 0.0669 - accuracy: 0.9824
Epoch 49/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0650 - accuracy: 0.9805
Epoch 50/50
6/6 [==============================] - 0s 4ms/step - loss: 0.0650 - accuracy: 0.9844
```

Out[11]: <keras.callbacks.History at 0x120f3816590>

In [12]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

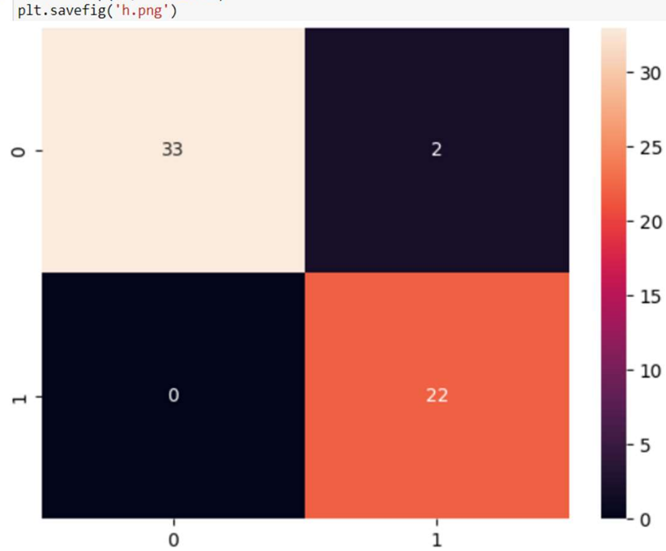
```
2/2 [==============================] - 0s 22ms/step
```

In [13]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [14]:
```
print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/57)*100))
```

```
Our accuracy is 96.49122807017544%
```

In [15]:
```
sns.heatmap(cm,annot=True)
plt.savefig('h.png')
```



Application:

- Neural networks: Backpropagation is the primary learning algorithm used to train artificial neural networks for various tasks, including pattern recognition, image classification, and speech recognition.
- Function approximation: Backpropagation can be used to approximate complex functions by training neural networks with appropriate architectures and activation functions.
- Time series prediction: Backpropagation-based neural networks can be applied to predict future values in time series data, such as stock market prices or weather forecasting.

3. **Demonstrate the working of EM algorithm to cluster a set of data stored in a .CSV file.**

```
In [3]: import numpy as np
        import pandas as pd
        from scipy.stats import multivariate_normal
```

```
In [9]: class GMM:
            def __init__(self, n_clusters, max_iterations=100):
                self.n_clusters = n_clusters
                self.max_iterations = max_iterations

                self.weights = None
                self.means = None
                self.covariances = None

            def fit(self, X):
                n_samples, n_features = X.shape

                # Initialize parameters randomly
                self.weights = np.ones(self.n_clusters) / self.n_clusters
                self.means = np.random.rand(self.n_clusters, n_features)
                self.covariances = np.array([np.eye(n_features)] * self.n_clusters)

                # EM algorithm
                for _ in range(self.max_iterations):
                    # E-step: Calculate responsibilities
                    responsibilities = self._expectation(X)

                    # M-step: Update parameters
                    self._maximization(X, responsibilities)

            def predict(self, X):
                # E-step: Calculate responsibilities
                responsibilities = self._expectation(X)

                # Assign samples to clusters based on responsibilities
                labels = np.argmax(responsibilities, axis=1)

                return labels
```

```
    def _expectation(self, X):
        n_samples = X.shape[0]

        # Calculate probabilities using the current parameters
        probabilities = np.zeros((n_samples, self.n_clusters))
        for k in range(self.n_clusters):
            probabilities[:, k] = self.weights[k] * multivariate_normal.pdf(X, self.means[k], self.covariances[k])

        # Calculate responsibilities using Bayes' rule
        responsibilities = probabilities / np.sum(probabilities, axis=1, keepdims=True)

        return responsibilities

    def _maximization(self, X, responsibilities):
        n_samples = X.shape[0]

        # Update weights
        self.weights = np.mean(responsibilities, axis=0)

        # Update means
        for k in range(self.n_clusters):
            weight_k = self.weights[k]
            mean_k = np.sum(responsibilities[:, k].reshape(-1, 1) * X, axis=0) / (n_samples * weight_k)
            self.means[k] = mean_k

        # Update covariances
        for k in range(self.n_clusters):
            weight_k = self.weights[k]
            diff = X - self.means[k]
            covariance_k = np.dot((responsibilities[:, k].reshape(-1, 1) * diff).T, diff) / (n_samples * weight_k)
            self.covariances[k] = covariance_k

# Load the dataset from CSV file
data = pd.read_csv('Em algo.csv')
X = data.values

# Create and fit the GMM model
n_clusters = 3
gmm = GMM(n_clusters)
gmm.fit(X)
```

```
from sklearn.preprocessing import LabelEncoder

# label_encoder = LabelEncoder()
# X_encoded = label_encoder.fit_transform(X_categorical)

# Make predictions
labels = gmm.predict(X)
print("Cluster labels:", labels)
```

```
Cluster labels: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
 1 1 1 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1
 2 1 1 1 1 1 1 0 1 1 1 1 2 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2
 2 2 2 1 1 2 2 1 1 2 2 1 2 1 0 2 2 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2
 2 1 1 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2
 1 2 2 1 2 1 2 1 2 2 2 2 1 0 2 2 2 2 1 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2
 1 0 2 1 1 2 2 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2
 2 2 2 2 2 2 2 1 2 2 2 2 0 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 2 1 1 2 2 1 1 1 1 0 1 1 2 1 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 1 1 2 2 2
 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 1 2 2 1
 2 0 2 1 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2
 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2
 1 1 2 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 1 1 2
 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 2 0
 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 1
 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 2 0 2 2 2 1 2 1 2 2 2 2 2 2
 1 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 2 1 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 1 1 0 2 0 1
 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 0 2 2 2 2 2 2 2 2 1 2 1 1 1 2 2 1 1 1 1 2 2
 1 2 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 1 1
 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 2 1
 2]
```

Application:

- Clustering: The EM algorithm is used in clustering algorithms like Gaussian Mixture Models (GMM) to estimate the parameters of each cluster and assign data points to the most likely cluster.
- Image segmentation: EM algorithm can be used to segment images by modeling the distribution of pixel intensities in different regions.
- Missing data imputation: EM algorithm can be applied to impute missing values in datasets by estimating the missing values based on observed data.

4. Demonstrate Pre processing (Data Cleaning, Integration and Transformation) activity on suitable data: For example: Identify and Delete Rows that Contain Duplicate Data by considering an appropriate dataset. Identify and Delete Columns That Contain a Single Value by considering an appropriate dataset.

```python
In [1]: import pandas as pd

        # Load the dataset into a pandas DataFrame
        data1= pd.read_csv('Emp info.csv')
        print(data1)

            Name  Age                Email         Phone Country
        0   John   25     john@example.com  123-456-7890     USA
        1  Emily   30    emily@example.com  987-654-3210      UK
        2   John   25     john@example.com  123-456-7890     USA
        3  Sarah   35    sarah@example.com  555-555-5555  Canada
        4  Peter   28    peter@example.com  111-222-3333     USA
        5  Emily   30    emily@example.com  987-654-3210      UK
        6   John   25  john.doe@example.com  123-456-7890     USA
        7  Sarah   35    sarah@example.com  555-555-5555  Canada

In [2]: # Identify and delete duplicate rows based on all columns
        data = data1.drop_duplicates()

        # Print the updated dataset after removing duplicates
        print(data)

            Name  Age                Email         Phone Country
        0   John   25     john@example.com  123-456-7890     USA
        1  Emily   30    emily@example.com  987-654-3210      UK
        3  Sarah   35    sarah@example.com  555-555-5555  Canada
        4  Peter   28    peter@example.com  111-222-3333     USA
        6   John   25  john.doe@example.com  123-456-7890     USA

In [3]: # # Identify and delete columns that contain a single value
        # Assuming your dataset is stored in a DataFrame called 'data'
        data.drop(data.columns[-1], axis=1, inplace=True)
        print(data)

In [4]: data2 = pd.read_csv('Employee Salary.csv')
        print(data2)

            Name  Employee_id
        0   John            1
        1  Emily            2
        2   John            3
        3  Sarah            4
        4  Peter            5
        5  Emily            6
        6   John            7
        7  Sarah            8

In [12]: #merging the datasets

         merged_data = pd.merge(data1 ,data2, on = 'Employee_id')
         print(merged_data)

             Employee_id  Experience_Years  Age  Gender  Salary
         0             1                 5   28  Female   30000
         1             2                 1   21    Male   60000
         2             3                 3   23  Female   40000
         3             4                 2   22    Male   10000
         4             5                 1   17    Male   20000
         5             6                25   62    Male   40000
         6             7                19   54  Female   60000
         7             8                 2   21  Female   40000
         8             9                10   36  Female   30000
         9            10                10   36  Female   50000
         10           11                 4   26  Female   20000
         11           12                 6   29    Male   50000
         12           13                14   39    Male   10000
         13           14                11   40    Male   60000
```

Applications:

- Data cleaning: Pre-processing techniques like handling missing values, removing outliers, and correcting inconsistent data are used to improve the quality of data before further analysis.
- Feature scaling and normalization: Pre-processing can involve scaling or normalizing features to ensure that they have similar ranges or distributions, which can improve the performance of certain algorithms like SVM or K-means.
- Dimensionality reduction: Techniques like Principal Component Analysis (PCA) or Feature Selection can be used to reduce the number of features while preserving relevant information and reducing computational complexity.

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

In [1]:
```python
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('trainingdata.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```python
import random
import csv
```

In [3]:
```python
def g_0(n):
    return ("?",)*n

def s_0(n):
    return ('0',)*n
```

In [4]:
```python
def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

l1 = [1, 2, 3]
l2 = [3, 4, 5]

list(zip(l1, l2))
```

Out[4]: `[(1, 3), (2, 4), (3, 5)]`

In [5]:
```python
# min_generalizations
def fulfills(example, hypothesis):
    ### the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != '0' else x[i]
    return [tuple(h_new)]
```

In [6]:
```python
min_generalizations(h=('0', '0'  , 'sunny'),
                    x=('rainy', 'windy', 'cloudy'))
```

Out[6]: `[('rainy', 'windy', '?')]`

In [7]:
```python
def min_specializations(h, domains, x):
    results = []
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results
```

In [8]:
```python
min_specializations(h=('?', 'x',),
                    domains=[['a', 'b', 'c'], ['x', 'y']],
                    x=('b', 'x'))
```

Out[8]: `[('a', 'x'), ('c', 'x'), ('?', '0')]`

```
In [11]: with open('trainingdata.csv')  as csvFile:
                 examples = [tuple(line) for line in csv.reader(csvFile)]



         examples
```

```
Out[11]: [('sky', 'airTemp', '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')]
```

```
In [12]: def get_domains(examples):
             d = [set() for i in examples[0]]
             for x in examples:
                 for i, xi in enumerate(x):
                     d[i].add(xi)
             return [list(sorted(x)) for x in d]

         get_domains(examples)
```

```
Out[12]: [['Rainy', 'Sunny', 'sky'],
          ['Cold', 'Warm', 'airTemp'],
          ['High', 'Normal', 'humidity'],
          ['Strong', 'wind'],
          ['Cool', 'Warm', 'water'],
          ['Change', 'Same', 'forecast'],
          ['No', 'Yes', 'enjoySport']]
```

```
In [13]: def candidate_elimination(examples):
             domains = get_domains(examples)[:-1]

             G = set([g_0(len(domains))])
             S = set([s_0(len(domains))])
             i=0
             print("\n G[{0}]:".format(i),G)
             print("\n S[{0}]:".format(i),S)
             for xcx in examples:
```

```
                 if cx=='Y': # x is positive example
                     G = {g for g in G if fulfills(x, g)}
                     S = generalize_S(x, G, S)
                 else: # x is negative example
                     S = {s for s in S if not fulfills(x, s)}
                     G = specialize_G(x, domains, G, S)
                 print("\n G[{0}]:".format(i),G)
                 print("\n S[{0}]:".format(i),S)
             return
```

```
In [14]: def generalize_S(x, G, S):
             S_prev = list(S)
             for s in S_prev:
                 if s not in S:
                     continue
                 if not fulfills(x, s):
                     S.remove(s)
                     Splus = min_generalizations(s, x)
                     ## keep only generalizations that have a counterpart in G
                     S.update([h for h in Splus if any([more_general(g,h)
                                                        for g in G])])
                     ## remove hypotheses less specific than any other in S
                     S.difference_update([h for h in S if
                                          any([more_general(h, h1)
                                               for h1 in S if h != h1])])
             return S
```

```
In [15]: def specialize_G(x, domains, G, S):
             G_prev = list(G)
             for g in G_prev:
                 if g not in G:
                     continue
                 if fulfills(x, g):
                     G.remove(g)
                     Gminus = min_specializations(g, domains, x)
                     ## keep only specializations that have a conuterpart in S
                     G.update([h for h in Gminus if any([more_general(h, s)
                                                         for s in S])])
                     ## remove hypotheses less general than any other in G
```

```
In [16]: candidate_elimination(examples)
```

```
G[0]: {('?', '?', '?', '?', '?', '?')}

S[0]: {('0', '0', '0', '0', '0', '0')}

G[1]: {('?', '?', '?', '?', '?', 'Same'), ('?', '?', 'Normal', '?', '?', '?'), ('?', 'Cold', '?', '?', '?', '?'), ('?', 'War
m', '?', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', '?'), ('?', '?', '?', '?', '?', 'Change'), ('?', '?', 'High', '?', '?',
'?'), ('Rainy', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', 'Warm', '?'), ('Sunny', '?', '?', '?', '?', '?'), ('?', '?',
'?', '?', 'Cool', '?')}

S[1]: {('0', '0', '0', '0', '0', '0')}

G[2]: {('?', 'airTemp', 'Normal', '?', '?', '?'), ('Sunny', '?', '?', '?', 'water', '?'), ('?', 'airTemp', '?', '?', 'Warm',
'?'), ('?', '?', '?', '?', 'water', 'Same'), ('sky', '?', '?', 'Strong', '?', '?'), ('?', 'Warm', '?', 'wind', '?', '?'), ('?',
'?', 'Normal', 'wind', '?', '?'), ('?', '?', '?', 'Strong', 'water', '?'), ('?', '?', 'humidity', '?', 'Warm', '?'), ('?', '?',
'humidity', 'Strong', '?', '?'), ('?', '?', '?', 'wind', '?', 'Same'), ('Sunny', 'airTemp', '?', '?', '?', '?'), ('?', 'Warm',
'?', '?', '?', 'forecast'), ('sky', '?', '?', '?', '?', 'Same'), ('?', 'Warm', 'humidity', '?', '?', '?'), ('Sunny', '?', '?',
'wind', '?', '?'), ('?', '?', '?', 'wind', 'Warm', '?'), ('Sunny', '?', '?', '?', '?', 'forecast'), ('?', '?', 'Normal', '?',
'water', '?'), ('sky', '?', '?', '?', 'Warm', '?'), ('sky', '?', 'Normal', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', 'fore
cast'), ('Rainy', '?', '?', '?', '?', '?'), ('sky', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', 'Warm', 'forecast'),
('?', 'airTemp', '?', '?', '?', 'Same'), ('?', '?', 'Normal', '?', '?', 'forecast'), ('?', 'Cold', '?', '?', '?', '?'), ('Sunn
y', '?', 'humidity', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Change'), ('?', 'Warm', '?', '?', 'water', '?'), ('?', '?', 'Hi
gh', '?', '?', '?'), ('?', '?', 'humidity', '?', '?', 'Same'), ('?', '?', '?', '?', 'Cool', '?'), ('?', 'airTemp', '?', 'Stron
g', '?', '?')}

S[2]: {('0', '0', '0', '0', '0', '0')}

G[3]: {('?', 'airTemp', 'Normal', '?', '?', '?'), ('?', 'airTemp', 'High', '?', '?', '?'), ('Sunny', '?', '?', '?', 'water',
'?'), ('?', 'airTemp', '?', '?', 'Warm', '?'), ('?', '?', '?', '?', 'water', 'Same'), ('sky', '?', '?', 'Strong', '?', '?'),
('?', 'Warm', '?', 'wind', '?', '?'), ('?', '?', 'Normal', 'wind', '?', '?'), ('?', '?', '?', 'Strong', 'water', '?'), ('?',
'?', 'humidity', '?', 'Warm', '?'), ('?', '?', 'humidity', 'Strong', '?', '?'), ('?', '?', '?', 'wind', '?', 'Same'), ('Sunny',
'airTemp', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', 'forecast'), ('sky', '?', '?', '?', '?', 'Same'), ('?', 'Warm', 'h
umidity', '?', '?', '?'), ('Sunny', '?', '?', 'wind', '?', '?'), ('?', '?', '?', 'wind', 'Warm', '?'), ('?', '?', 'High', '?',
'water', '?'), ('Sunny', '?', '?', '?', '?', 'forecast'), ('?', '?', 'Normal', '?', 'water', '?'), ('sky', '?', '?', '?', 'War
m', '?'), ('sky', '?', 'Normal', '?', '?', '?'), ('?', '?', '?', 'Strong', '?', 'forecast'), ('?', '?', 'High', 'wind', '?',
'?'), ('Rainy', '?', '?', '?', '?', '?'), ('sky', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', 'Warm', 'forecast'), ('?',
'airTemp', '?', '?', '?', 'Same'), ('?', '?', 'Normal', '?', '?', 'forecast'), ('sky', '?', 'High', '?', '?', '?'), ('?', 'Col
d', '?', '?', '?', '?'), ('Sunny', '?', 'humidity', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Change'), ('?', 'Warm', '?',
```

Applications:

- Concept learning: The Candidate-Elimination algorithm is used to learn concepts from labeled data and update the hypothesis space iteratively.
- Inductive logic programming: The algorithm can be used to learn logical rules or hypotheses from examples and background knowledge.

6. **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 and show the output for test cases. Develop an interactive program by Comparing the result by implementing LIST THEN ELIMINATE algorithm.**

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [41]: import csv
         num_attributes = 6
         a = []
         print("\n The Given Training Data Set \n")
         file = 'ws.csv'

         with open(file, 'r') as csvfile:
             reader = csv.reader(csvfile)
             for row in reader:
                 a.append (row)
                 print(row)

         type(reader)
```

```
 The Given Training Data Set

['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']
```

```
Out[41]: _csv.reader
```

```
In [42]: a[0][:-1]
```

```
Out[42]: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

```
In [43]: print("\n The initial value of hypothesis: ")
         hypothesis = ['0'] * num_attributes
         print(hypothesis)
```

```
for j in range(0,num_attributes):
    hypothesis[j] = a[0][j]

hypothesis
```

```
 The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
```

Out[43]:
```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

In [44]:
```
hypothesis == a[0][:-1]
```

Out[44]:
```
True
```

In [45]:
```
print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(0,len(a)):
    if a[i][num_attributes]=='Yes':
        for j in range(0,num_attributes):
            print(a[i][j], end=' ')
            if a[i][j]!=hypothesis[j]:
                hypothesis[j]='?'
            else :
                hypothesis[j]= a[i][j]
    print("\n\nFor Training instance No:{} the hypothesis is ".format(i), hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

```
 Find S: Finding a Maximally Specific Hypothesis

Sunny Warm Normal Strong Warm Same

For Training instance No:0 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
Sunny Warm High Strong Warm Same

For Training instance No:1 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']


For Training instance No:2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
Sunny Warm High Strong Cool Change

For Training instance No:3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally Specific Hypothesis for a given Training Examples :

['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

In [18]:
```
import pandas as pd

def list_then_eliminate(training_data):
    positive_examples = training_data[training_data['Play'] == 'No']

    if positive_examples.empty:
        return None

    hypothesis = positive_examples.iloc[0, :-1]

    for _, example in training_data.iterrows():
        if example['Play'] == 'No':
            for attr in hypothesis.keys():
                if example[attr] != hypothesis[attr]:
                    hypothesis[attr] = '?'

    return hypothesis

# Load the training data
training_data = pd.read_csv('dataset 1.csv')
print(training_data)
```

```
    Outlook Temperature Humidity  Windy Play
0     Sunny         Hot     High  False   No
1  Overcast         Hot     High  False  Yes
2      Rain        Mild     High  False  Yes
3      Rain        Cool   Normal  False  Yes
4     Sunny        Mild   Normal   True  Yes
```

```
In [14]: # Run the LIST THEN ELIMINATE algorithm
hypothesis_space = list_then_eliminate(training_data)

# Output the hypothesis space
print("Hypothesis space:")
print(hypothesis_space)
```

```
Hypothesis space:
Outlook        Sunny
Temperature      Hot
Humidity        High
Windy          False
Name: 0, dtype: object
```

Application:

- Concept learning: The FIND-S algorithm is used to learn the most specific hypothesis from labeled examples in the form of attribute-value pairs.
- Rule induction: FIND-S can be applied to induce classification rules from data, where each rule represents a specific condition that leads to a particular class label.

****