VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

NIRANJAN NAGARAJ SAVANUR (1BM19CS104)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning" carried out by **NIRANJAN NAGARAJ SAVANUR(1BM19CS104),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Prof. Saritha A N	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and HOD
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Course Outcome

1.Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

In [1]:

```python
import numpy as np
import pandas as pd
```

In [2]:

```python
data = pd.read_csv("mydata.csv")
print(data,"\n")
```

```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes
```

In [3]:

```python
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)
```

```
 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']
```

In [4]:

```python
def findS(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(d,target))
```

```
 The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

In [1]:

```python
import numpy as np
import pandas as pd
```

In [2]:

```python
print("Enter features separated by space")
features = input().split()
print("Features ", features)
num_samples = int(input("enter number of samples: "))
```

```
Enter features separated by space
Time Weather Temperature Company Humidity Wind
Features  ['Time', 'Weather', 'Temperature', 'Company', 'Humidity', 'Wind']
enter number of samples: 4
```

In [11]:

```python
def findS():
    specific_hypothesis = ["n"]*len(features)
    for a in range(num_samples):
        print("sample", a)

        temp_features = input("Enter features: ").split()
        target = input("Enter outcome: ")
        if target == "Yes":
            for x in range(len(specific_hypothesis)):
                if specific_hypothesis[x] == "n":
                    specific_hypothesis[x] = temp_features[x]
                elif temp_features[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
        print("Specific hypothesis: ", specific_hypothesis)
    return specific_hypothesis
```

In [12]:

```python
print("\n The final hypothesis is:",findS())
```

```
sample 0
Enter features: Morning Sunny Warm Yes Mild Strong
Enter outcome: Yes
Specific hypothesis:  ['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong']
sample 1
Enter features: Evening Rainy Cold No Mild Normal
Enter outcome: No
Specific hypothesis:  ['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong']
sample 2
Enter features: Morning Sunny Moderate Yes Normal Normal
Enter outcome: Yes
Specific hypothesis:  ['Morning', 'Sunny', '?', 'Yes', '?', '?']
sample 3
Enter features: Evening Sunny Cold Yes High Strong
Enter outcome: Yes
Specific hypothesis:  ['?', 'Sunny', '?', 'Yes', '?', '?']

 The final hypothesis is: ['?', 'Sunny', '?', 'Yes', '?', '?']
```

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.

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

In [2]:
```python
data = pd.read_csv('mydata.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']
```

In [5]:
```python
def learn(concepts, target):
    specific_h = ["null"]*len(concepts[0])
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    specific_h = concepts[0].copy()
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
```


```python
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
```


```python
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?']*len(concepts[0])]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

In [6]:
```python
s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

```
Initialization of specific_h and genearal_h

Specific Boundary:  ['null', 'null', 'null', 'null', 'null', 'null']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
```

```
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

In [ ]:

3.Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

In [1]:

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

data = load_iris()
```

In [2]:

```
df = pd.DataFrame(data.data, columns = data.feature_names)
```

In [3]:

```
df.head()
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [4]:

```
df['Species'] = data.target

#replace this with the actual names

target = np.unique(data.target)

target_names = np.unique(data.target_names)

targets = dict(zip(target, target_names))

df['Species'] = df['Species'].replace(targets)
```

In [5]:

```
x = df.drop(columns="Species")
```

In [5]:
```python
x = df.drop(columns="Species")

y = df["Species"]
```

In [6]:
```python
feature_names = x.columns
labels = y.unique()
```

In [7]:
```python
from sklearn.model_selection import train_test_split

X_train, test_x, y_train, test_lab = train_test_split(x,y,test_size = 0.4,random_state = 42)
```

In [10]:
```python
 from sklearn.tree import DecisionTreeClassifier
 clf = DecisionTreeClassifier(random_state = 42, criterion="entropy")
```

In [11]:
```python
clf.fit(X_train, y_train)
```

Out[11]:
```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

In [12]:
```python
test_pred = clf.predict(test_x)
```

In [13]:
```python
from sklearn import metrics

import seaborn as sns

import matplotlib.pyplot as plt

confusion_matrix = metrics.confusion_matrix(test_lab,test_pred)
```

In [14]:
```python
confusion_matrix
```

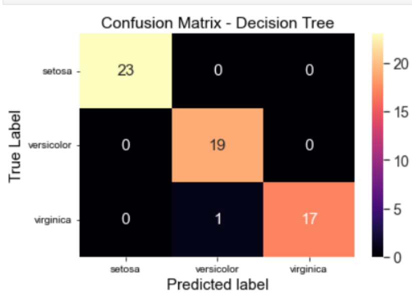
Out[14]:
```
array([[23,  0,  0],
       [ 0, 19,  0],
       [ 0,  1, 17]], dtype=int64)
```

In [15]:
```python
matrix_df = pd.DataFrame(confusion_matrix)
```


In [14]:
```python
confusion_matrix
```

Out[14]:
```
array([[23,  0,  0],
       [ 0, 19,  0],
       [ 0,  1, 17]], dtype=int64)
```

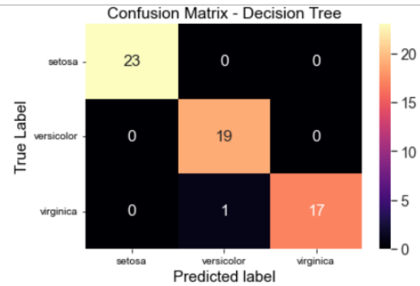

In [15]:
```python
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted label", fontsize =15)
ax.set_xticklabels(['']+labels)
ax.set_ylabel("True Label", fontsize=15)
ax.set_yticklabels(list(labels), rotation = 0)
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

In [16]:
```python
clf.score(test_x,test_lab)
```

Out[16]:
```
0.9833333333333333
```



<Figure size 720x504 with 0 Axes>

In [16]:

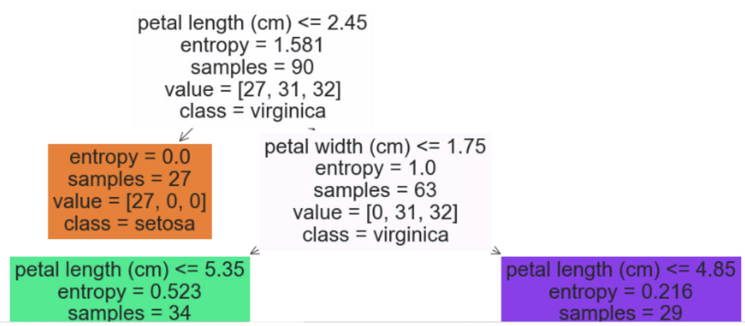
```
clf.score(test_x,test_lab)
```

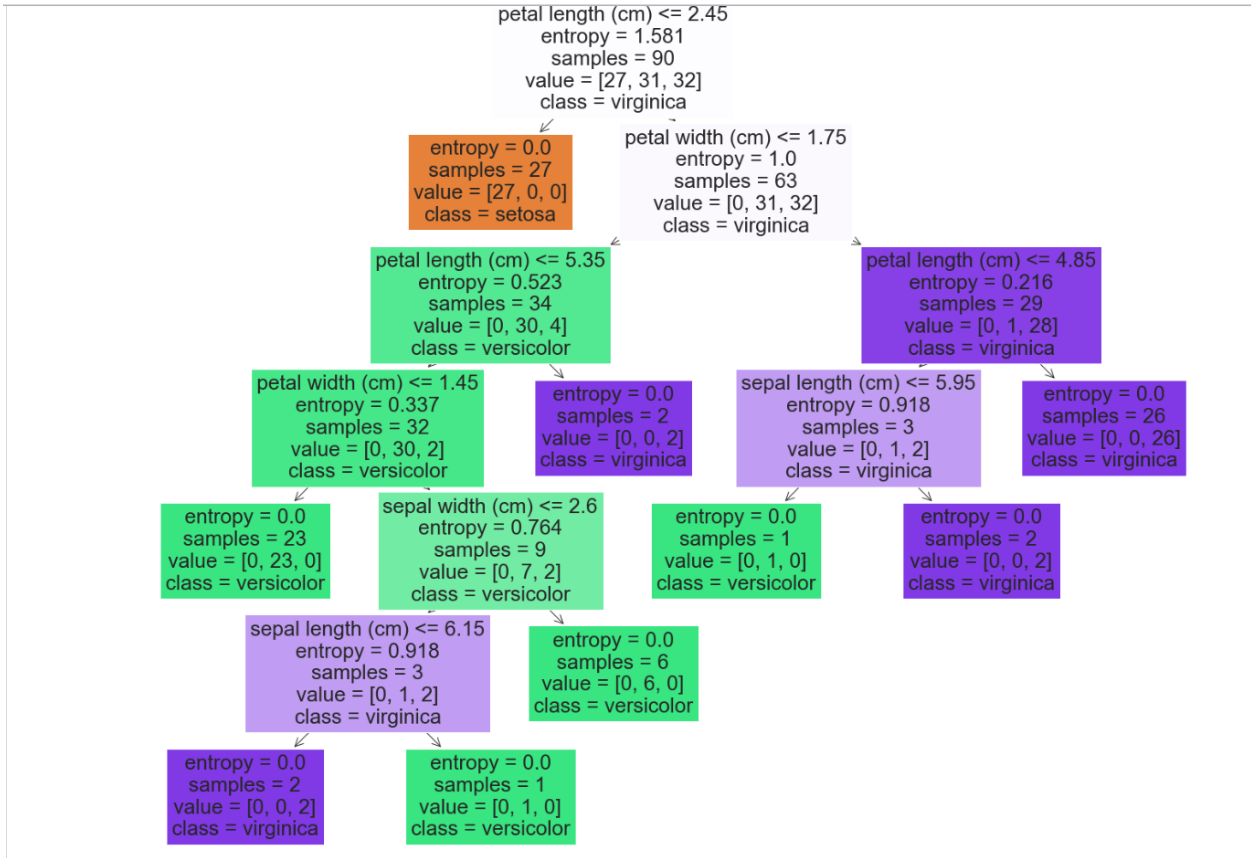
Out[16]:

```
0.9833333333333333
```

In [17]:

```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=data.feature_names,
                   class_names=data.target_names,
                   filled=True)
```





In [1]:

```
import pandas as pd
import math
import numpy as np
```

In [2]:

```
data = pd.read_csv("dataset.csv")
features = [feat for feat in data]
features.remove("answer")
```

In [7]:

```
features
```

Out[7]:

```
['outlook', 'temperature', 'humidity', 'wind']
```

In [4]:

```
data
```

Out[4]:

	outlook	temperature	humidity	wind	answer
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes
5	rain	cool	normal	strong	no
6	overcast	cool	normal	strong	yes
7	sunny	mild	high	weak	no
8	sunny	cool	normal	weak	yes
9	rain	mild	normal	weak	yes
10	sunny	mild	normal	strong	yes
11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes

11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes
13	rain	mild	high	strong	no

In [3]:

```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

In [5]:

```
def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))
```

In [6]:

```
def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain
```

In [8]:

```
def ID3(examples, attrs):
```

In [8]:

```python
def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
```

In [9]:

```python
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
```


```python
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root
```


In [9]:

```python
def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)
```

In [10]:

```python
root = ID3(data, features)
printTree(root)
```

```
outlook
        overcast ->  ['yes']

        rain
                wind
                        strong ->  ['no']

                        weak ->  ['yes']

        sunny
                humidity
                        high ->  ['no']

                        normal ->  ['yes']
```

In [ ]:

4.Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

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

In [2]:
```
data = pd.read_csv('/content/dataset.csv')
data.head()
```

Out[2]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [3]:
```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

In [4]:
```
y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

In [5]:
```
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

```python
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

In [6]:

```python
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
predictions = []
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

In [18]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
X = df[feature_col_names].values
y = df[predicted_class_names].values
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
```

In [19]:

```
df.head()
```

Out[19]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [29]:

```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [30]:

```
metrics.confusion_matrix(ytest,predicted)
```

Out[30]:

```
array([[139,  26],
       [ 33,  56]], dtype=int64)
```

In [28]:

```
print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))
```

In [30]:

```
metrics.confusion_matrix(ytest,predicted)
```

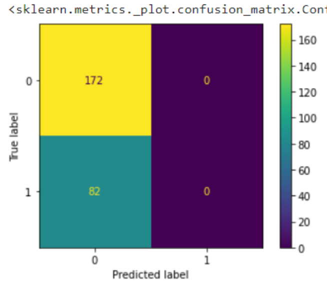
Out[30]:

```
array([[139,  26],
       [ 33,  56]], dtype=int64)
```

In [28]:

```
print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))
```


```
Confusion matrix
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000190E55B3670>
```



In [31]:

```
print(metrics.classification_report(ytest,predicted))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.82       165
           1       0.68      0.63      0.65        89

    accuracy                           0.77       254
   macro avg       0.75      0.74      0.74       254
weighted avg       0.76      0.77      0.77       254
```

In [8]:

```
print("Predicted Value for individual Test Data:", predictTestData)
```

```
Predicted Value for individual Test Data: [1]
```

5.Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

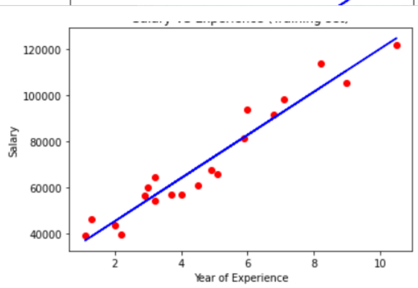
In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

In [4]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[4]:
```
LinearRegression()
```

In [5]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [6]:
```
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```



In [7]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [ ]: