

Contents

1.a.Design a simple machine learning model to train the training instances and test the same.	5
1.b. 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	5
Source with output	5
Data Source TrainingData.csv	6
2.a.Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.	7
b. 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	8
Source and Output	8
3.a 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.	11
Source and Output	11
Data	12
b. Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.	13
4a For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.	17
4 b For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm	20
5 a. 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.	22
5 b Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set	35
6a Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix	39
6b Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.	40
7a Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix	43

- 8a. 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 44
- b. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs 45
- 9A. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. 46
- 9B. ASSUMING A SET OF DOCUMENTS THAT NEED TO BE CLASSIFIED, USE THE NAÏVE BAYESIAN CLASSIFIER MODEL TO PERFORM THIS TASK. BUILT-IN JAVA CLASSES/API CAN BE USED TO WRITE THE PROGRAM. CALCULATE THE ACCURACY, PRECISION, AND RECALL FOR YOUR DATA SET. 48
- 10A . 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. 50

1.a. Design a simple machine learning model to train the training instances and test the same.

```
In [2]: 1 from random import randint
2 TRAIN_SET_LIMIT = 1000
3 TRAIN_SET_COUNT = 100
4
5 TRAIN_INPUT = list()
6 TRAIN_OUTPUT = list()
7 for i in range(TRAIN_SET_COUNT):
8     a = randint(0, TRAIN_SET_LIMIT)
9     b = randint(0, TRAIN_SET_LIMIT)
10    c = randint(0, TRAIN_SET_LIMIT)
11    op = a + (2*b) + (3*c)
12    TRAIN_INPUT.append([a, b, c])
13    TRAIN_OUTPUT.append(op)

In [3]: 1 from sklearn.linear_model import LinearRegression
2
3 predictor = LinearRegression(n_jobs=-1)
4 predictor.fit(X=TRAIN_INPUT, y=TRAIN_OUTPUT)
5

Out[3]: LinearRegression(n_jobs=-1)

In [4]: 1 X_TEST = [[10, 20, 30]]
2 outcome = predictor.predict(X=X_TEST)
3 coefficients = predictor.coef_
4
5 print('Outcome : {}\nCoefficients : {}'.format(outcome, coefficients))

outcome : [140.]
Coefficients : [1. 2. 3.]
```

1.b. 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

Source with output

```
In [8]: 1 #1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training
2 import csv
3 hypo = ['%', '%', '%', '%', '%', '%'];
4
5 with open('C:/2020-21/IDOL/Sem III/Machine Learning/trainingdata.csv') as csv_file:
6     readcsv = csv.reader(csv_file, delimiter=',')
7     print(readcsv)
8
9     data = []
10    print("\nThe given training examples are:")
11    for row in readcsv:
12        print(row)
13        if row[len(row)-1].upper() == "YES":
14            data.append(row)

<_csv.reader object at 0x000001F8D10B88340>

The given training examples are:
['sky', 'airTemp', 'humidity', 'wind', 'water', 'forecast', 'enjoysSport']
['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 [4]: 1 print("\nThe positive examples are:");
2 for x in data:
3     print(x);
4 print("\n");

The positive examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

```
In [5]: 1 TotalExamples = len(data);
2 i=0;
3 j=0;
4 k=0;
5 print("The steps of the Find-s algorithm are :\n",hypo);
6 list = [];
7 p=0;
8 d=len(data[p])-1;
9 for j in range(d):
10     list.append(data[i][j]);
11 hypo=list;
12 i=1;
13 for i in range(TotalExamples):
14     for k in range(d):
15         if hypo[k]!=data[i][k]:
16             hypo[k] = '?';
17             k=k+1;
18         else:
19             hypo[k];
20     print(hypo);
21 i=i+1;

The steps of the Find-s algorithm are :
['%', '%', '%', '%', '%', '%']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
In [6]: 1 print("\nThe maximally specific Find-s hypothesis for the given training examples is :");
2 list=[];
3 for i in range(d):
4     list.append(hypo[i]);
5 print(list);
```

The maximally specific Find-s hypothesis for the given training examples is :
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Data Source TrainingData.csv

	airTem	humidit			forecas	enjoySpo
sky	p	y	wind	water	t	rt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
					Chang	
Rainy	Cold	High	Strong	Warm	e	No
					Chang	
Sunny	Warm	High	Strong	Cool	e	Yes

2.a.Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

- Univariate Selection.

```
In [8]: 1 # Feature Selection with Univariate Statistical Tests
2 from pandas import read_csv
3 from numpy import set_printoptions
4 from sklearn.feature_selection import SelectKBest
5 from sklearn.feature_selection import f_classif
6 # load data
7 filename = 'C:\\2020-21\\IDOL\\Sem III\\Machine Learning\\pima-indians-diabetes.csv'
8 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
9 dataframe = read_csv(filename, names=names)
10 array = dataframe.values
11 X = array[:,0:8]
12 Y = array[:,8]
13 # feature extraction
14 test = SelectKBest(score_func=f_classif, k=4)
15 fit = test.fit(X, Y)
16 # summarize scores
17 set_printoptions(precision=3)
18 print(fit.scores_)
19 features = fit.transform(X)
20 # summarize selected features
21 print(features[0:5,:])

[ 39.67 213.162  3.257  4.304 13.281 71.772 23.871 46.141]
[[ 6.  148.   33.6 50. ]
 [ 1.   85.   26.6 31. ]
 [ 8.  183.   23.3 32. ]
 [ 1.   89.   28.1 21. ]
 [ 0.  137.   43.1 33. ]]
```

- Recursive Feature Elimination.

```
In [9]: 1 from pandas import read_csv
2 from sklearn.feature_selection import RFE
3 from sklearn.linear_model import LogisticRegression
4 # load data
5 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
6 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
7 dataframe = read_csv(url, names=names)
8 array = dataframe.values
9 X = array[:,0:8]
10 Y = array[:,8]
11 # feature extraction
12 model = LogisticRegression(solver='lbfgs')
13 rfe = RFE(model, 3)
14 fit = rfe.fit(X, Y)
15 print("Num Features: %d" % fit.n_features_)
16 print("Selected Features: %s" % fit.support_ import numpy
17 from pandas import read_csv
18 from sklearn.decomposition import PCA
19 # load data
20 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
21 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
22 dataframe = read_csv(url, names=names)
23 array = dataframe.values
24 X = array[:,0:8]
25 Y = array[:,8]
26 # feature extraction
27 pca = PCA(n_components=3)
28 fit = pca.fit(X)
29 # summarize components
30 print("Explained Variance: %s" % fit.explained_variance_ratio_)
31 print(fit.components_)
32 print("Feature Ranking: %s" % fit.ranking_)
```

Num Features: 7
click to scroll output; double click to hide
[True False False False False True True False]
Feature Ranking: [1 2 4 5 6 1 1 3]

- Principle Component Analysis.

```
In [10]: 1 import numpy
2 from pandas import read_csv
3 from sklearn.decomposition import PCA
4 # load data
5 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
6 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
7 dataframe = read_csv(url, names=names)
8 array = dataframe.values
9 X = array[:,0:8]
10 Y = array[:,8]
11 # feature extraction
12 pca = PCA(n_components=3)
13 fit = pca.fit(X)
14 # summarize components
15 print("Explained Variance: %s" % fit.explained_variance_ratio_)
16 print(fit.components_)

Explained Variance: [0.889 0.062 0.026]
[[ -2.022e-03  9.781e-02  1.609e-02  6.076e-02  9.931e-01  1.401e-02
   5.372e-04 -3.565e-03]
 [ -2.265e-02 -9.722e-01 -1.419e-01  5.786e-02  9.463e-02 -4.697e-02
  -8.168e-04 -1.402e-01]
 [ -2.246e-02  1.434e-01 -9.225e-01 -3.070e-01  2.098e-02 -1.324e-01
  -6.400e-04 -1.255e-01]]
```

- Feature Importance.

```
In [11]: 1 from pandas import read_csv
2 from sklearn.ensemble import ExtraTreesClassifier
3 # load data
4 url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv"
5 names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
6 dataframe = read_csv(url, names=names)
7 array = dataframe.values
8 X = array[:,0:8]
9 Y = array[:,8]
10 # feature extraction
11 model = ExtraTreesClassifier(n_estimators=10)
12 model.fit(X, Y)
13 print(model.feature_importances_)

[0.124 0.227 0.094 0.075 0.069 0.148 0.121 0.142]
```

- b. 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

Source and Output

```
In [2]: 1 #2. For a given set of training data examples stored in a .CSV file,
2 #implement and demonstrate the Candidate-Elimination algorithm to
3 #output a description of the set of all hypotheses consistent with the training examples.
4
5 import numpy as np
6 import pandas as pd

In [3]: 1 # Loading Data from a CSV File
2 data = pd.DataFrame(data=pd.read_csv('C:/2020-21/IDOL/Sem III/Machine Learning/trainingdata.csv'))
3 print(data)

      sky  airTemp  humidity    wind  water  forecast enjoySport
0  Sunny     Warm    Normal  Strong  Warm    Same       Yes
1  Sunny     Warm      High  Strong  Warm    Same       Yes
2  Rainy     Cold      High  Strong  Warm  Change       No
3  Sunny     Warm      High  Strong  Cool  Change      Yes
```

```
In [4]: 1 # Separating concept features from Target
2 concepts = np.array(data.iloc[:,0:-1])
3 print(concepts)

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
```

```
In [5]: 1 # Isolating target into a separate DataFrame
2 # copying last column to target array
3 target = np.array(data.iloc[:, -1])
4 print(target)

['Yes' 'Yes' 'No' 'Yes']
```

```
In [6]: 1 def learn(concepts, target):
2 ...
3     ...
4     learn() function implements the learning method of the Candidate elimination algorithm.
5     Arguments:
6         concepts - a data frame with all the features
7         target - a data frame with corresponding output values
8     ...
9
10    # Initialise S0 with the first instance from concepts
11    # .copy() makes sure a new list is created instead of just pointing to the same memory location
12    specific_h = concepts[0].copy()
13    print("\nInitialization of specific_h and general_h")
14    print(specific_h)
15    #h="#"
16    #print(h)
17
18    general_h = [[ "?" for i in range(len(specific_h))] for i in range(len(specific_h))]
19    print(general_h)
20    # The learning iterations
21    for i, h in enumerate(concepts):
22
23        # Checking if the hypothesis has a positive target
24        if target[i] == "Yes":
25            for x in range(len(specific_h)):
26
27                # Change values in S & G only if values change
```

```
27
28                # Change values in S & G only if values change
29                if h[x] != specific_h[x]:
30                    specific_h[x] = '?'
31                    general_h[x][x] = '?'
32
33        # Checking if the hypothesis has a positive target
34        if target[i] == "No":
35            for x in range(len(specific_h)):
36                # For negative hypothesis change values only in G
37                if h[x] != specific_h[x]:
38                    general_h[x][x] = specific_h[x]
39                else:
40                    general_h[x][x] = '?'
41
42        print("\nSteps of Candidate Elimination Algorithm",i+1)
43        print(specific_h)
44        print(general_h)
45
46    # find indices where we have empty rows, meaning those that are unchanged
47    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
48    for i in indices:
49        # remove those rows from general_h
50        general_h.remove(['?', '?', '?', '?', '?', '?'])
51
52    # Return final values
53    return specific_h, general_h
```

```
In [7]: 1 s_final, g_final = learn(concepts, target)
2 print("\nFinal Specific_h:", s_final, sep="\n")
3 print("\nFinal General_h:", g_final, sep="\n")
```

```
Initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[[ '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 1
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[[ '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[[ '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
[[ 'Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 4
['Sunny' 'Warm' '?' 'Strong' '?' '?']
[[ 'Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

3.a 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.

Source and Output

```
In [18]: 1 #write a program to implement the naive Bayesian classifier for a sample training
2 #data set stored as a .csv file. Compute the accuracy of the classifier, considering
3 #few test data sets.
4 # import necessary libarities
5 import pandas as pd
6 from sklearn import tree
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.naive_bayes import GaussianNB
```

```
In [19]: 1 # Load data from CSV
2 data = pd.read_csv('C:/2020-21/IDOL/Sem III/tennisdata.csv')
3 print("The first 5 values of data is :\n",data.head())

The first 5 values of data is :
   Outlook Temperature Humidity Windy PlayTennis
0    Sunny           Hot    High  False     No
1    Sunny           Hot    High  True      No
2  Overcast          Hot    High  False    Yes
3    Rainy           Mild   High  False    Yes
4    Rainy          Cool  Normal  False    Yes
```

```
In [20]: 1 # obtain Train data and Train output
2 X = data.iloc[:, :-1]
3 print("\nThe First 5 values of train data is\n",X.head())
```

```
The First 5 values of train data is
   Outlook Temperature Humidity Windy
0    Sunny           Hot    High  False
1    Sunny           Hot    High  True
2  Overcast          Hot    High  False
3    Rainy           Mild   High  False
4    Rainy          Cool  Normal  False
```

```
In [21]: 1 y = data.iloc[:, -1]
2 print("\nThe first 5 values of Train output is\n",y.head())
```

```
The first 5 values of Train output is
0    No
1    No
2   Yes
3   Yes
4   Yes
Name: PlayTennis, dtype: object
```

```
In [22]: 1 # Convert them in numbers
2 le_outlook = LabelEncoder()
3 X.outlook = le_outlook.fit_transform(X.outlook)
4
5 le_Temperature = LabelEncoder()
6 X.Temperature = le_Temperature.fit_transform(X.Temperature)
7
8 le_Humidity = LabelEncoder()
9 X.Humidity = le_Humidity.fit_transform(X.Humidity)
10
11 le_Windy = LabelEncoder()
12 X.Windy = le_Windy.fit_transform(X.Windy)
13
14 print("\nNow the Train data is :\n",X.head())
```

```
Now the Train data is :
   Outlook Temperature Humidity Windy
0        2           1       0     0
1        2           1       0     1
2        0           1       0     0
3        1           2       0     0
4        1           0       1     0
```

```
In [23]: 1 le_PlayTennis = LabelEncoder()
2 y = le_PlayTennis.fit_transform(y)
3 print("\nNow the Train output is\n",y)

Now the Train output is
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

In [24]: 1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)
3
4 classifier = GaussianNB()
5 classifier.fit(X_train,y_train)
6
7 from sklearn.metrics import accuracy_score
8 print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))

Accuracy is: 1.0
```

Data

Outloo k	Temperatur e	Humidit y	Windy FALS	PlayTenn is
Sunny	Hot	High	E	No
Sunny	Hot	High	TRUE	No
Overca st	Hot	High	FALS	
Rainy	Mild	High	E	Yes
Rainy	Cool	High	FALS	
Rainy	Cool	Normal	E	Yes
Overca st	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALS	
Sunny	Cool	Normal	E	Yes
Rainy	Mild	Normal	E	Yes
Sunny	Mild	Normal	TRUE	Yes
Overca st	Mild	High	TRUE	Yes
Overca st	Hot	Normal	E	Yes
Rainy	Mild	High	TRUE	No

b. Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

```
In [1]: 1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline

In [2]: 1 path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DA0101EN/automobileEDA.csv'
2 df = pd.read_csv(path)
3 df.head()

Out[2]:
symboling normalized-losses make aspiration num-of-doors body-style drive-wheels engine-location wheel-base length ... compression-ratio horsepower peak-rpm city-mpg highway-mpg
0 3 122 alfa-romero std two convertible rwd front 88.6 0.811148 ... 9.0 111.0 5000.0 21 27 13
1 3 122 alfa-romero std two convertible rwd front 88.6 0.811148 ... 9.0 111.0 5000.0 21 27 16
2 1 122 alfa-romero std two hatchback rwd front 94.5 0.822681 ... 9.0 154.0 5000.0 19 26 16
3 2 164 audi std four sedan fwd front 99.8 0.848630 ... 10.0 102.0 5500.0 24 30 13
4 2 164 audi std four sedan 4wd front 99.4 0.848630 ... 8.0 115.0 5500.0 18 22 17

5 rows × 29 columns
```

```
In [3]: 1 df.dtypes
2

Out[3]:
symboling          int64
normalized-losses    int64
make                object
aspiration          object
num-of-doors        object
body-style          object
drive-wheels        object
engine-location     object
wheel-base          float64
length              float64
width               float64
height              float64
curb-weight         int64
engine-type         object
num-of-cylinders   object
engine-size         int64
fuel-system         object
bore                float64
stroke              float64
compression-ratio   float64
horsepower          float64
peak-rpm             float64
city-mpg            int64
highway-mpg          int64
price               float64
city-L/100km        float64
horsepower-binned   object
diesel              int64
gas                 int64
dtype: object
```

```
In [4]: 1 df.describe()
Out[4]:
   symboling  normalized-
   losses  wheel-
   base  length  width  height  curb-weight  engine-
   size  bore  stroke  compression-
   ratio  horsepower
count    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000    201.000000
mean     0.840796    122.000000    98.797015    0.837102    0.915126    53.766667    2555.666667    126.875622    3.330692    3.256904    10.164279    103.405534
std      1.254802    31.996250    6.066366    0.059213    0.029187    2.447822    517.296727    41.546834    0.268072    0.319256    4.004965    37.365700
min     -2.000000    65.000000    86.600000    0.678039    0.837500    47.800000    1488.000000    61.000000    2.540000    2.070000    7.000000    48.000000
25%     0.000000    101.000000    94.500000    0.801538    0.890278    52.000000    2169.000000    98.000000    3.150000    3.110000    8.600000    70.000000
50%     1.000000    122.000000    97.000000    0.832292    0.909722    54.100000    2414.000000    120.000000    3.310000    3.290000    9.000000    72.000000
75%     2.000000    137.000000    102.400000    0.881788    0.925000    55.500000    2926.000000    141.000000    3.580000    3.410000    9.000000    72.000000
max     3.000000    256.000000    120.900000    1.000000    1.000000    59.800000    4066.000000    326.000000    3.940000    4.170000    2.000000    201.000000
```

```
In [5]: 1 df.dtypes
2 for x in df:
3     if df[x].dtypes == "int64":
4         df[x] = df[x].astype(float)
5     print (df[x].dtypes)
6
float64
float64
float64
float64
float64
float64
float64
float64

In [6]: 1 df = df.select_dtypes(exclude=['object'])
2 df=df.fillna(df.mean())
3 X = df.drop('price',axis=1)
4 y = df['price']

In [7]: 1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

In [8]: 1 from sklearn.ensemble import RandomForestRegressor
2 regressor = RandomForestRegressor(n_estimators = 1000, random_state = 42)
3 regressor.fit(X_train, y_train)

Out[8]: RandomForestRegressor(n_estimators=1000, random_state=42)
```

```
In [10]: 1 y_pred = regressor.predict(X_test)
2 df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
3 df
4
Out[10]:
   Actual    Predicted
 18  6295.0  5828.668000
 170 10698.0 10492.447583
 107 13860.0 21763.165000
 98 13499.0 15317.683000
 177 15750.0 16334.467000
 ...
 30  6855.0  6115.846000
 160  8238.0  8129.287000
 40 12945.0 11438.811000
 56  8845.0 10125.190667
 131 15510.0 13600.018833

61 rows x 2 columns
```

```
In [11]: 1 from sklearn import metrics
2 print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
3 print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
4 print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

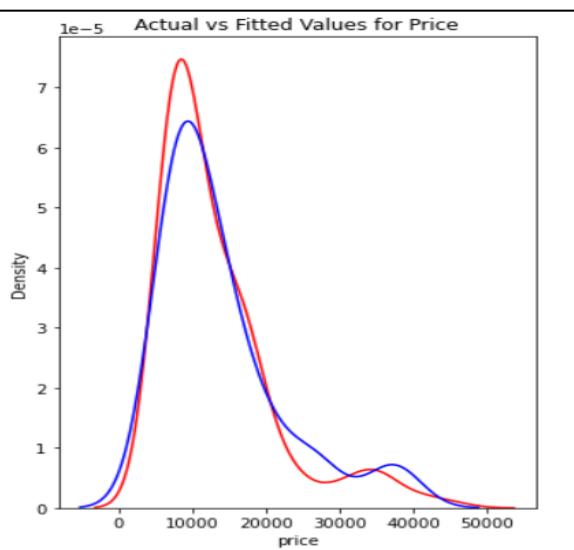
Mean Absolute Error: 1993.2901175839186
Mean Squared Error: 9668487.223350348
Root Mean Squared Error: 3109.4191134921566

In [12]: 1 # Calculate the absolute errors
2 errors = abs(y_pred - y_test)
3 # Print out the mean absolute error (mae)
4 print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')
5
6 # Calculate mean absolute percentage error (MAPE)
7 mape = 100 * (errors / y_test)
8 # Calculate and display accuracy
9 accuracy = 100 - np.mean(mape)
10 print('Accuracy:', round(accuracy, 2), '%.')

Mean Absolute Error: 1993.29 degrees.
Accuracy: 87.87 %.
```

```
In [13]: 1 import seaborn as sns
2 plt.figure(figsize=(5, 7))
3
4
5 ax = sns.distplot(y, hist=False, color="r", label="Actual Value")
6 sns.distplot(y_pred, hist=False, color="b", label="Fitted Values" , ax=ax)
7
8
9 plt.title('Actual vs Fitted Values for Price')
10
11
12 plt.show()
13 plt.close()

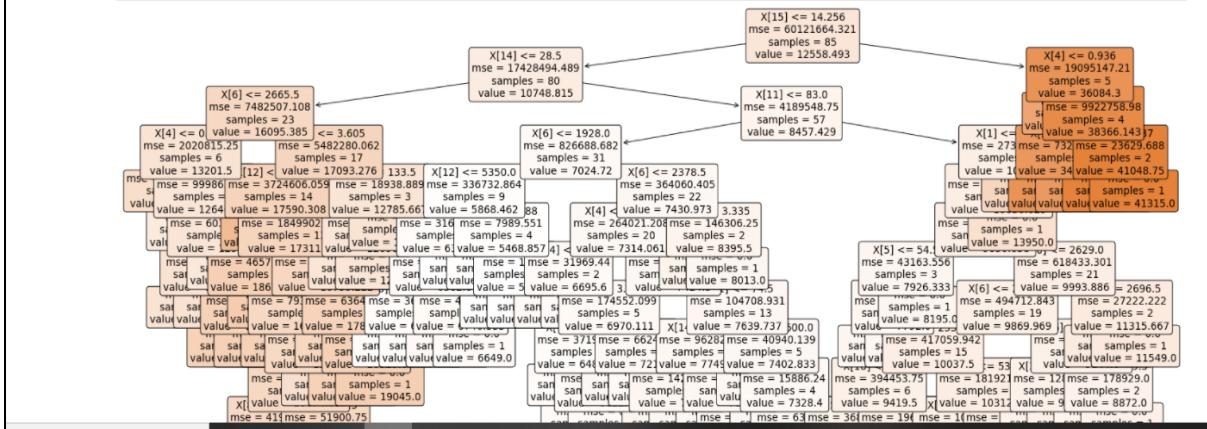
C:\Users\hina_\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function; it will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar functionality) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\hina_\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function; it will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar functionality) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



```

In [15]: 1 # Import tools needed for visualization
2 from sklearn.tree import export_graphviz
3 import pydot
4 # Pull out one tree from the forest
5 Tree = regressor.estimators_[5]
6 # Import tools needed for visualization
7 from sklearn.tree import export_graphviz
8 import pydot
9 # Pull out one tree from the forest
10 Tree = regressor.estimators_[5]
11 # Export the image to a dot file
12 from sklearn import tree
13 plt.figure(figsize=(25,15))
14 tree.plot_tree(Tree,filled=True,
15                 rounded=True,
16                 fontsize=14);

```



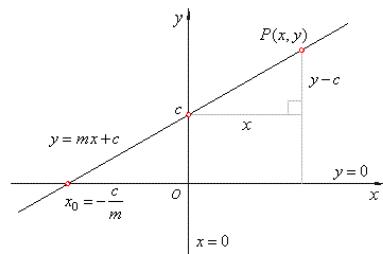
4a For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables. In the case of one independent variable it is called simple linear regression. For more than one independent variable, the process is called multiple linear regression. We will be dealing with simple linear regression in this tutorial.

Let X be the independent variable and Y be the dependent variable. We will define a linear relationship between these two variables as follows:

$$Y = mX + c$$



This is the equation for a line that you studied in high school. m is the slope of the line and c is the y intercept. Today we will use this equation to train our model with a given dataset and predict the value of Y for any given value of X .

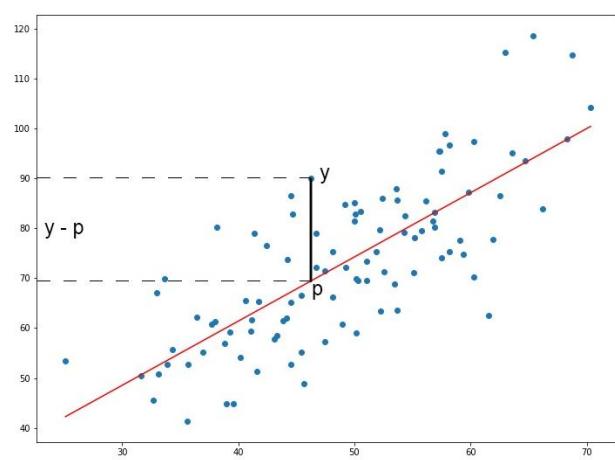
Our challenge today is to determine the value of m and c , that gives the minimum error for the given dataset. We will be doing this by using the **Least Squares** method.

Finding the Error

So to minimize the error we need a way to calculate the error in the first place. A **loss function** in machine learning is simply a measure of how different the predicted value is from the actual value.

Today we will be using the **Quadratic Loss Function** to calculate the loss or error in our model. It can be defined as:

$$L(x) = \sum_{i=1}^n (y_i - p_i)^2$$



We are squaring it because, for the points below the regression line $\mathbf{y} - \mathbf{p}$ will be negative and we don't want negative values in our total error.

Least Squares method

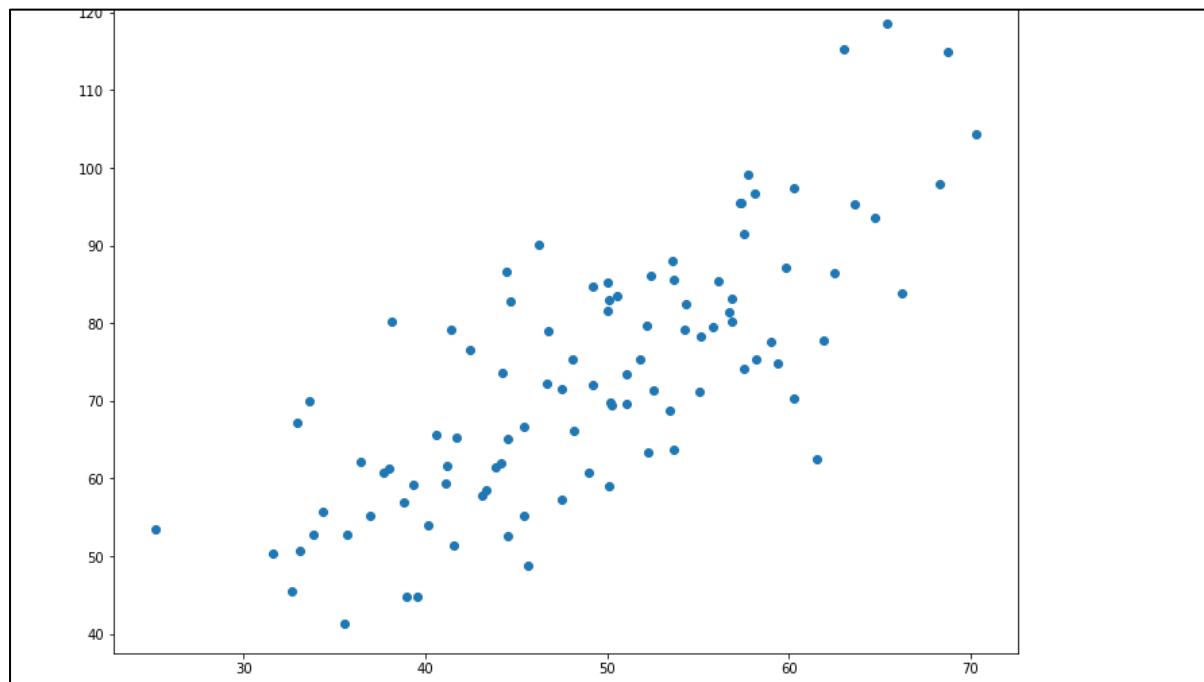
Now that we have determined the loss function, the only thing left to do is minimize it. This is done by finding the partial derivative of L equating it to 0 and then finding an expression for \mathbf{m} and \mathbf{c} . After we do the math, we are left with these equations:

$$m = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$
$$c = \bar{y} - m\bar{x}$$

Here \bar{x} is the mean of all the values in the input \mathbf{X} and \bar{y} is the mean of all the values in the desired output \mathbf{Y} . This is the Least Squares method. Now we will implement this in python and make predictions.

```
In [4]: # Making imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
```

```
In [5]: # Preprocessing Input data
data = pd.read_csv('C:/Hina/Machine Learning/Data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
```



```
In [6]: # Building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)

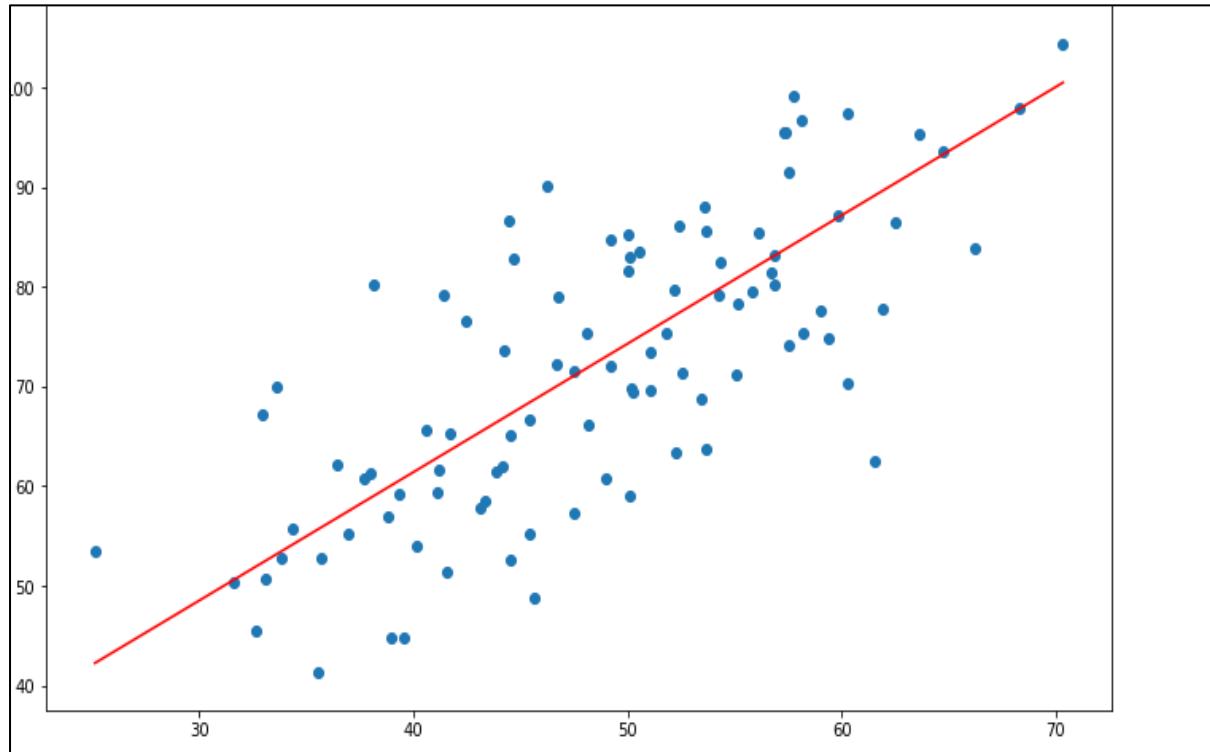
num = 0
den = 0
for i in range(len(X)):
    num += (X[i] - X_mean)*(Y[i] - Y_mean)
    den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean

print (m, c)
```

1.287357370010931 9.908606190326509

```
In [7]: # Making predictions
Y_pred = m*X + c

plt.scatter(X, Y) # actual
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') # predicted
plt.show()
```



4 b For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm

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

In [2]: dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Classification/master/DMVWrittenTests.csv')
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values
dataset.head(5)

Out[2]:
   DMV_Test_1  DMV_Test_2  Results
0    34.623660    78.024693      0
1    30.286711    43.894998      0
2    35.847409    72.902198      0
3    60.182599    86.308552      1
4    79.032736    75.344376      1
```

```
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 = 0.25, random_state = 0)

In [4]: from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

In [5]: from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

Out[5]: LogisticRegression()

In [6]: y_pred = classifier.predict(X_test)
y_pred

Out[6]: array([1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0], dtype=int64)
```

```
In [7]: from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm

Accuracy :  0.88

Out[7]: array([[11,  0],
       [ 3, 11]], dtype=int64)

In [8]: df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})

Out[8]:
```

	Real Values	Predicted Values
0	1	1
1	0	0
2	0	0
3	0	0
4	1	1
5	1	1
6	1	0
7	1	1
8	0	0
9	1	1
10	0	0
11	0	0
12	0	0
13	1	1
14	1	0
15	1	1
16	0	0
17	1	1

18	1	0
19	1	1
20	0	0
21	0	0
22	1	1
23	1	1
24	0	0

```
In [9]: from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression')
plt.xlabel('DMV_Test_1')
plt.ylabel('DMV_Test_2')
plt.legend()
plt.show()
```

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
 c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.



5 a. 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 [7]: 1 import pandas as pd
2 from pandas import DataFrame
3 df =pd.read_csv('C:\\2020-21\\IDOL\\Sem III\\Machine Learning\\PlayTennis.csv')
4 df.head(15)

Out[7]: Unnamed: 0 PlayTennis Outlook Temperature Humidity Wind
0 0 No Sunny Hot High Weak
1 1 No Sunny Hot High Strong
2 2 Yes Overcast Hot High Weak
3 3 Yes Rain Mild High Weak
4 4 Yes Rain Cool Normal Weak
5 5 No Rain Cool Normal Strong
6 6 Yes Overcast Cool Normal Strong
7 7 No Sunny Mild High Weak
8 8 Yes Sunny Cool Normal Weak
9 9 Yes Rain Mild Normal Weak
10 10 Yes Sunny Mild Normal Strong
11 11 Yes Overcast Mild High Strong
12 12 Yes Overcast Hot Normal Weak
13 13 No Rain Mild High Strong

In [9]: 1 #df_tennis.columns[0]
2 df.keys()[0]

Out[9]: 'Unnamed: 0'
```

```
In [11]: 1 #Function to calculate the entropy of probability of observations
2 # -p*log2*p
3
4 def entropy(probs):
5     import math
6     return sum( [-prob*math.log(prob, 2) for prob in probs] )
7
8 #Function to calculate the entropy of the given Data Sets/List with respect to target attributes
9 def entropy_of_list(a_list):
10    #print("A-list",a_list)
11    from collections import Counter
12    cnt = Counter(x for x in a_list) # Counter calculates the proportion of class
13    # print("\nClasses:",cnt)
14    #print("No and Yes Classes:",a_list.name,cnt)
15    num_instances = len(a_list)*1.0 # = 14
16    print("\n Number of Instances of the Current Sub Class is {0}: ".format(num_instances ))
17    probs = [x / num_instances for x in cnt.values()] # x means no of YES/NO
18    print("\n Classes:",min(cnt),max(cnt))
19    print(" \n Probabilities of Class {0} is {1}: ".format(min(cnt),min(probs)))
20    print(" \n Probabilities of Class {0} is {1}: ".format(max(cnt),max(probs)))
21    return entropy(probs) # Call Entropy :
22
23 # The initial entropy of the YES/NO attribute for our dataset.
24 print("\n INPUT DATA SET FOR ENTROPY CALCULATION:\n", df['PlayTennis'])
25
26 total_entropy = entropy_of_list(df['PlayTennis'])
27
28 print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

```

INPUT DATA SET FOR ENTROPY CALCULATION:
0      No
1      Yes
2      Yes
3      Yes
4      Yes
5      No
6      Yes
7      No
8      Yes
9      Yes
10     Yes
11     Yes
12     Yes
13     No
Name: PlayTennis, dtype: object

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Total Entropy of PlayTennis Data Set: 0.9402859586706309

```

```

In [15]: 1 def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
2     print("Information Gain Calculation of ",split_attribute_name)
3     ...
4     Takes a DataFrame of attributes, and quantifies the entropy of a target
5     attribute after performing a split along the values of another attribute.
6     ...
7     # Split Data by Possible Vals of Attribute:
8     df_split = df.groupby(split_attribute_name)
9     # for name,group in df_split:
10    #     print("Name:\n",name)
11    #     print("group:\n",group)
12
13    # Calculate Entropy for Target Attribute, as well as
14    # Proportion of Obs in Each Data-Split
15    nobs = len(df.index) * 1.0
16    # print("NOBS",nobs)
17    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs] })[target_attribute_name]
18    #print([target_attribute_name])
19    #print("Entropy List ",entropy_of_list)
20    #print("DFAGGENT",df_agg_ent)
21    df_agg_ent.columns = ['Entropy', 'PropObservations']
22    #if trace: # helps understand what ffn is doing:
23    #    print(df_agg_ent)
24
25    # calculate Information Gain:
26    new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
27    old_entropy = entropy_of_list(df[target_attribute_name])
28    return old_entropy - new_entropy
29
30
31 print('Info-gain for outlook is :'+str( information_gain(df, 'outlook', 'PlayTennis')),"\n")
32 print('\n Info-gain for Humidity is: ' + str( information_gain(df, 'Humidity', 'PlayTennis')), "\n")
33 print('\n Info-gain for Wind is:' + str( information_gain(df, 'Wind', 'PlayTennis'))," \n")
34 print('\n Info-gain for Temperature is! '+ str( information_gain(df, 'Temperature', 'PlayTennis'))," \n")

```

```

Information Gain Calculation of Outlook

Number of Instances of the Current Sub Class is 4.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.4:

Probabilities of Class Yes is 0.6:

Number of Instances of the Current Sub Class is 5.0:

```

```

In [16]: 1 def id3(df, target_attribute_name, attribute_names, default_class=None):
2     ## Tally target attribute:
3     from collections import Counter
4     cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO
5
6     ## First check: Is this split of the dataset homogeneous?
7     if len(cnt) == 1:
8         return next(iter(cnt)) # next input data set, or raises StopIteration when EOF is hit.
9
10    ## Second check: Is this split of the dataset empty?
11    # if yes, return a default value
12    elif df.empty or (not attribute_names):
13        return default_class # Return None for Empty Data set
14
15    ## Otherwise: This dataset is ready to be devied up!
16    else:
17        # Get Default Value for next recursive call of this function:
18        default_class = max(cnt.keys()) #No of YES and NO Class
19
20        # Compute the Information Gain of the attributes:
21        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names] #
22        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
23
24        # Choose Best Attribute to split on:
25        best_attr = attribute_names[index_of_max]
26
27        # Create an empty tree, to be populated in a moment
28        tree = {best_attr:{} } # Initiate the tree with best attribute as a node
29        remaining_attribute_names = [i for i in attribute_names if i != best_attr]
30
31        # Split dataset
32        # On each split, recursively call this algorithm.
33        # populate the empty tree with subtrees, which
34        # are the result of the recursive call
35        for attr_val, data_subset in df.groupby(best_attr):
36            subtree = id3(data_subset,
37                           target_attribute_name,
38                           remaining_attribute_names,
39                           default_class)
39
40            tree[best_attr][attr_val] = subtree
40
return tree

```

```

In [17]: 1 # Get Predictor Names (all but 'class')
2 attribute_names = list(df.columns)
3 print("List of Attributes:", attribute_names)
4 attribute_names.remove('PlayTennis') #Remove the class attribute
5 print("Predicting Attributes:", attribute_names)

```

List of Attributes: ['Unnamed: 0', 'PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Unnamed: 0', 'Outlook', 'Temperature', 'Humidity', 'Wind']

```

In [18]: 1 # Run Algorithm:
2 from pprint import pprint
3 tree = id3(df,'PlayTennis',attribute_names)
4 print("\n\nThe Resultant Decision Tree is :\n")
5 #print(tree)
6 pprint(tree)
7 attribute = next(iter(tree))
8 print("Best Attribute : \n",attribute)
9 print("Tree Keys:\n",tree[attribute].keys())

```

```

INFORMATION GAIN Calculation of Unnamed: 0

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

```

```
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
classes: Yes Yes
```

```
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: No No  
Probabilities of Class No is 1.0:  
Probabilities of Class No is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes
```

```
Probabilities of Class No is 1.0:  
Probabilities of Class No is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: No No  
Probabilities of Class No is 1.0:
```

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

```
Probabilities of Class No is 0.35714285714285715:  
Probabilities of Class Yes is 0.6428571428571429:  
Information Gain Calculation of Outlook  
Number of Instances of the Current Sub Class is 4.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 5.0:  
Classes: No Yes
```

```
Probabilities of Class No is 0.4:  
Probabilities of Class Yes is 0.6:  
Number of Instances of the Current Sub Class is 5.0:  
Classes: No Yes  
Probabilities of Class No is 0.4:  
Probabilities of Class Yes is 0.6:  
Number of Instances of the Current Sub Class is 14.0:  
Classes: No Yes  
Probabilities of Class No is 0.35714285714285715:
```

```
Probabilities of Class Yes is 0.6428571428571429:  
Information Gain Calculation of Temperature  
Number of Instances of the Current Sub Class is 4.0:  
Classes: No Yes  
Probabilities of Class No is 0.25:  
Probabilities of Class Yes is 0.75:  
Number of Instances of the Current Sub Class is 4.0:  
Classes: No Yes  
Probabilities of Class No is 0.5:
```

```
Probabilities of Class Yes is 0.5:  
Number of Instances of the Current Sub Class is 6.0:  
Classes: No Yes  
Probabilities of Class No is 0.3333333333333333:  
Probabilities of Class Yes is 0.6666666666666666:  
Number of Instances of the Current Sub Class is 14.0:  
Classes: No Yes  
Probabilities of Class No is 0.35714285714285715:  
Probabilities of Class Yes is 0.6428571428571429:  
Information Gain Calculation of Humidity
```

```
Number of Instances of the Current Sub Class is 7.0:  
Classes: No Yes  
Probabilities of Class No is 0.42857142857142855:  
Probabilities of Class Yes is 0.5714285714285714:  
Number of Instances of the Current Sub Class is 7.0:  
Classes: No Yes  
Probabilities of Class No is 0.14285714285714285:  
Probabilities of Class Yes is 0.8571428571428571:  
Number of Instances of the Current Sub Class is 14.0:
```

```
Number of Instances of the Current Sub Class is 14.0:  
Classes: No Yes  
Probabilities of Class No is 0.35714285714285715:  
Probabilities of Class Yes is 0.6428571428571429:  
Information Gain Calculation of Wind  
Number of Instances of the Current Sub Class is 6.0:  
Classes: No Yes  
Probabilities of Class No is 0.5:  
Probabilities of Class Yes is 0.5:  
Number of Instances of the Current Sub Class is 8.0:
```

```
Classes: No Yes
```

```
Probabilities of Class No is 0.25:
```

```
Probabilities of Class Yes is 0.75:
```

```
Number of Instances of the Current Sub Class is 14.0:
```

```
Classes: No Yes
```

```
Probabilities of Class No is 0.35714285714285715:
```

```
Probabilities of Class Yes is 0.6428571428571429:
```

```
The Resultant Decision Tree is :
```

```
{'Unnamed: 0': {0: 'No',
 1: 'No',
 2: 'Yes',
 3: 'Yes',
 4: 'Yes',
 5: 'No',
 6: 'Yes',
 7: 'No',
 8: 'Yes',
 9: 'Yes',
 10: 'Yes',
 11: 'Yes',
 12: 'Yes',
 13: 'No'}}
```

```
Best Attribute :
```

```
Unnamed: 0
```

```
Tree Keys:
```

```
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
```

```
In [19]: 1 def classify(instance, tree, default=None): # Instance of Play Tennis with Predicted
2
3     #print("Instance:",instance)
4     attribute = next(iter(tree)) # outlook/Humidity/Wind
5     print("Key:",tree.keys()) # [outlook,Humidity,Wind ]
6     print("Attribute:",attribute) # [Key /Attribute Both are same ]
7
8     # print("Insance of Attribute : ",instance[attribute],attribute)
9     if instance[attribute] in tree[attribute].keys(): # Value of the attributs in set of Tree keys
10         result = tree[attribute][instance[attribute]]
11         print("Instance Attribute:",instance[attribute],"TreeKeys :",tree[attribute].keys())
12         if isinstance(result, dict): # this is a tree, delve deeper
13             return classify(instance, result)
14         else:
15             return result # this is a label
16     else:
17         return default
```

```
In [21]: 1 df['predicted'] = df.apply(classify, axis=1, args=(tree,'No') )
2     # classify func allows for a default arg: when tree doesn't have answer for a particular
3     # comibation of attribute-values, we can use 'no' as the default guess
4
5     print(df['predicted'])
6
7     print('\n Accuracy is:\n' + str( sum(df['PlayTennis']==df['predicted']) / (1.0*len(df.index)) ))
8
9
10 df[['PlayTennis', 'predicted']]
```

```
Instance Attribute: 0 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 1 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 2 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 3 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 4 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 5 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 6 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 7 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 8 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 9 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 10 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 11 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
Key: dict_keys(['Unnamed: 0'])
Attribute: Unnamed: 0
Instance Attribute: 12 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
```

```

Instance Attribute: 13 TreeKeys : dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
0      No
1      No
2      Yes
3      Yes
4      Yes
5      No
6      Yes
7      No
8      Yes
9      Yes
10     Yes
11     Yes
12     Yes
13     No
Name: predicted, dtype: object

Accuracy is:
1.0

Out[21]:

```

	Play Tennis	predicted
0	No	No
1	No	No
2	Yes	Yes
3	Yes	Yes
4	Yes	Yes
5	No	No
6	Yes	Yes
7	No	No
8	Yes	Yes
9	Yes	Yes
10	Yes	Yes
11	Yes	Yes
12	Yes	Yes
13	No	No

```

In [22]: 1 training_data = df.iloc[1:-4] # all but last four instances
2 test_data = df.iloc[-4:] # just the last four
3 train_tree = id3(training_data, 'PlayTennis', attribute_names)
4
5 test_data['predicted2'] = test_data.apply(
6                         classify,
7                         axis=1,
8                         args=(train_tree,'Yes')) # <---- train_data tree
9
10
11 print ('\n\n Accuracy is : ' + str( sum(test_data['PlayTennis']==test_data['predicted2']) / (1.0*len(test_data.index)) ))

```

```

Information Gain Calculation of  Unnamed: 0

Number of Instances of the Current Sub Class is 1.0:
Classes: No No

Probabilities of Class No is 1.0:
Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:
Classes: Yes Yes

Probabilities of Class Yes is 1.0:
Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

```

```
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: No No  
Probabilities of Class No is 1.0:
```

```
Probabilities of Class No is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: No No  
Probabilities of Class No is 1.0:  
Probabilities of Class No is 1.0:
```

```
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 1.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 9.0:
```

```
Classes: No Yes  
Probabilities of Class No is 0.333333333333333:  
Probabilities of Class Yes is 0.666666666666666:  
Information Gain Calculation of Outlook  
Number of Instances of the Current Sub Class is 2.0:  
Classes: Yes Yes  
Probabilities of Class Yes is 1.0:  
Probabilities of Class Yes is 1.0:  
Number of Instances of the Current Sub Class is 4.0:  
Classes: No Yes
```

```
Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 3.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:

Number of Instances of the Current Sub Class is 9.0:

Classes: No Yes
```

```
Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:
Information Gain Calculation of Temperature

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 2.0:

Classes: No Yes

Probabilities of Class No is 0.5:
```

```
Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 3.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:

Number of Instances of the Current Sub Class is 9.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:
```

Probabilities of Class Yes is 0.6666666666666666:
Information Gain Calculation of Humidity

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.2:

Probabilities of Class Yes is 0.8:

5 b Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set

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

In [3]: col=['sepal_length','sepal_width','petal_length','petal_width','type']
iris=pd.read_csv("C:\Hina\Machine Learning\iris.xlsx",names=col)

In [4]: print("First five rows")
print(iris.head())
print("*****")
print("columns",iris.columns)
print("*****")
print("shape:",iris.shape)
print("*****")
print("Size:",iris.size)
print("*****")
print("no of samples available for each type")
print(iris['type'].value_counts())
print("*****")
print(iris.describe())
print("*****")
```

```
First five rows
   sepal_length  sepal_width  petal_length  petal_width      type
0          5.1         3.5         1.4        0.2  Iris-setosa
1          4.9         3.0         1.4        0.2  Iris-setosa
2          4.7         3.2         1.3        0.2  Iris-setosa
3          4.6         3.1         1.5        0.2  Iris-setosa
4          5.0         3.6         1.4        0.2  Iris-setosa
*****
columns Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'type'], dtype='object')
*****
shape: (150, 5)
*****
Size: 750
*****
no of samples available for each type
Iris-setosa    50
Iris-versicolor  50
Iris-virginica   50
Name: type, dtype: int64
*****
   sepal_length  sepal_width  petal_length  petal_width
count    150.000000    150.000000    150.000000    150.000000
mean     5.843333     3.054000     3.758667     1.198667
std      0.828066     0.433594     1.764420     0.763161
min      4.300000     2.000000     1.000000     0.100000
25%     5.100000     2.800000
50%     5.800000     3.000000
75%     6.400000     3.300000
```

```
In [6]: def euclidianDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        distance += np.square(data1[x] - data2[x])

    return np.sqrt(distance)
def knn(trainingSet, testInstance, k):

    distances = {}
    sort = {}
    length = testInstance.shape[1]
    print(length)

    # Calculating euclidean distance between each row of training data and test data
    for x in range(len(trainingSet)):

        dist = euclidianDistance(testInstance, trainingSet.iloc[x], length)
        distances[x] = dist[0]
```

```

# Sorting them on the basis of distance
sorted_d = sorted(distances.items(), key=operator.itemgetter(1)) #by using it we store indices also
sorted_d1 = sorted(distances.items())
print(sorted_d[5:])
print(sorted_d1[5:])

neighbors = []

# Extracting top k neighbors
for x in range(k):
    neighbors.append(sorted_d[x][0])
counts = {"Iris-setosa":0,"Iris-versicolor":0,"Iris-virginica":0}

# Calculating the most freq class in the neighbors
for x in range(len(neighbors)):
    response = trainingSet.iloc[neighbors[x]][-1]

    if response in counts:
        counts[response] += 1
    else:
        counts[response] = 1

```

```

print(counts)
sortedVotes = sorted(counts.items(), key=operator.itemgetter(1), reverse=True)
print(sortedVotes)
return(sortedVotes[0][0], neighbors)

In [8]: testSet = [[1.4, 3.6, 3.4, 1.2]]
test = pd.DataFrame(testSet)
result,neigh = knn(iris, test, 4)#here we gave k=4
print("And the flower is:",result)
print("the neighbors are:",neigh)

4
[(57, 3.706750598570128), (8, 3.8065732621348567), (42, 3.817066936798463), (93, 3.8340579025361627), (38, 3.8431757701151272)]
[(0, 4.32434966208793), (1, 4.196427051671457), (2, 4.057092555020159), (3, 3.8858718455450894), (4, 4.237924020083418)]
{'Iris-setosa': 2, 'Iris-versicolor': 2, 'Iris-virginica': 0}
[('Iris-setosa', 2), ('Iris-versicolor', 2), ('Iris-virginica', 0)]
And the flower is: Iris-setosa
the neighbors are: [57, 8, 42, 93]

```

```

In [11]: from sklearn.neighbors import KNeighborsClassifier
x=iris.iloc[:,4:] #all parameters
y=iris["type"] #class Labels

In [12]: neigh=KNeighborsClassifier(n_neighbors=4)
neigh.fit(iris.iloc[:,4:],iris["type"])

Out[12]: KNeighborsClassifier(n_neighbors=4)

In [13]: testSet = [[1.4, 3.6, 3.4, 1.2]]
test = pd.DataFrame(testSet)
print(test)
print("predicted:",neigh.predict(test))
print("neighbors",neigh.kneighbors(test))

      0   1   2   3
0  1.4  3.6  3.4  1.2
predicted: ['Iris-setosa']
neighbors (array([[3.7067506 , 3.80657326, 3.81706694, 3.8340579 ]]), array([[57,  8, 42, 93]]), dtype=int64)

```

```

In [15]: import pandas as pd
import numpy as np
import math
import operator
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score , precision_score , roc_auc_score ,roc_curve
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore") #to remove unwanted warnings

In [16]: x=iris.iloc[:,3:]#features
y=iris.iloc[:,4:]#Class Labels
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
#test_size determines the percentage of test data you want here
#train=80% and test=20% data is randomly split

```

```
In [17]: cv_scores = []
neighbors = list(np.arange(3,50,2))
for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors = n,algorithm = 'brute')

    cross_val = cross_val_score(knn,x_train,y_train,cv = 5 , scoring = 'accuracy')
    cv_scores.append(cross_val.mean())

error = [1-x for x in cv_scores]
optimal_n = neighbors[ error.index(min(error)) ]
knn_optimal = KNeighborsClassifier(n_neighbors = optimal_n,algorithm = 'brute')
knn_optimal.fit(x_train,y_train)
pred = knn_optimal.predict(x_test)
acc = accuracy_score(y_test,pred)*100
print("The accuracy for optimal k = {0} using brute is {1}".format(optimal_n,acc))

The accuracy for optimal k = 7 using brute is 96.66666666666667
```

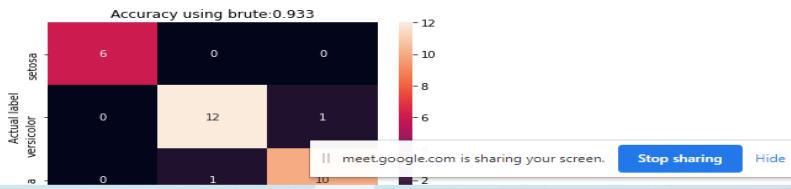
```
In [18]: print("classification_report using brute force")
print(classification_report(y_test,pred))

classification_report using brute force
precision    recall   f1-score   support
Iris-setosa      1.00      1.00      1.00       6
Iris-versicolor  1.00      0.92      0.96      13
Iris-virginica   0.92      1.00      0.96      11

accuracy         0.97      0.97      0.97      30
macro avg        0.97      0.97      0.97      30
weighted avg     0.97      0.97      0.97      30
```

```
In [19]: clf = SVC(kernel = 'linear').fit(x_train,y_train)
clf.predict(x_test)
# Creates a confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Transform to df for easier plotting
cm_df = pd.DataFrame(cm,
                      index = ['setosa','versicolor','virginica'],
                      columns = ['setosa','versicolor','virginica'])

sns.heatmap(cm_df, annot=True)
plt.title('Accuracy using brute:{0:.3f}'.format(accuracy_score(y_test, y_pred)))
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```



```
In [20]: for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors = n,algorithm = 'kd_tree')

    cross_val = cross_val_score(knn,x_train,y_train,cv = 5 , scoring = 'accuracy')
    cv_scores.append(cross_val.mean())

error = [1-x for x in cv_scores]
optimal_n = neighbors[ error.index(min(error)) ]
knn_optimal = KNeighborsClassifier(n_neighbors = optimal_n,algorithm = 'kd_tree')
knn_optimal.fit(x_train,y_train)
pred = knn_optimal.predict(x_test)
acc = accuracy_score(y_test,pred)*100
print("The accuracy for optimal k = {0} using kd-tree is {1}".format(optimal_n,acc))

The accuracy for optimal k = 7 using kd-tree is 96.66666666666667

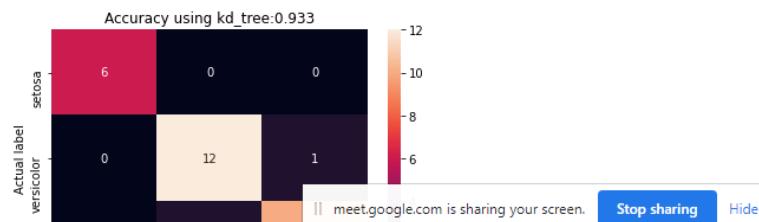
In [21]: print("classification_report using kd-tree")
print(classification_report(y_test,pred))

classification_report using kd-tree
precision    recall   f1-score   support
Iris-setosa      1.00      1.00      1.00       6
Iris-versicolor  1.00      0.92      0.96      13
Iris-virginica   0.92      1.00      0.96      11

accuracy         0.97      0.97      0.97      30
macro avg        0.97      0.97      0.97      30
weighted avg     0.97      0.97      0.97      30
```

```
In [22]: clf = SVC(kernel = 'linear').fit(x_train,y_train)
clf.predict(x_train)
y_pred = clf.predict(x_test)
# Creates a confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Transform to df for easier plotting
cm_df = pd.DataFrame(cm,
                      index = ['setosa','versicolor','virginica'],
                      columns = ['setosa','versicolor','virginica'])

sns.heatmap(cm_df, annot=True)
plt.title('Accuracy using kd_tree:{0:.3f}'.format(accuracy_score(y_test, y_pred)))
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```



6a Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix

```
In [1]: #Import required Libraries
#Import required Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
#Load the dataset
url = "https://raw.githubusercontent.com/SharmaNatasha/Machine-Learning-using-Python/master/Datasets/IRIS.csv"
df = pd.read_csv(url)
#quick Look into the data
df.head(5)
#Separate data and label
x = df.iloc[:,1:4]
y = df.iloc[:,4]
#Prepare data for classification process
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
In [2]: #Create a model
KNN_Classifier = KNeighborsClassifier(n_neighbors = 6, p = 2, metric='minkowski')

In [3]: #Train the model
KNN_Classifier.fit(x_train, y_train)
#Let's predict the classes for test data
pred_test = KNN_Classifier.predict(x_test)
```

```
In [4]: import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
#Load the dataset
url = "https://raw.githubusercontent.com/SharmaNatasha/Machine-Learning-using-Python/master/Datasets/IRIS.csv"
df = pd.read_csv(url)
#quick Look into the data
df.head(5)
#Separate data and label
x = df.iloc[:,1:4].values
#Creating the kmeans classifier
KMeans_Cluster = KMeans(n_clusters = 3)
y_class = KMeans_Cluster.fit_predict(x)
```

```
In [5]: import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as pyplot
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
corpus = [
    'the brown fox jumped over the brown dog',
    'the quick brown fox',
    'the brown brown dog',
    'the fox ate the dog'
]
query = ["brown"]
X = vectorizer.fit_transform(corpus)
Y = vectorizer.transform(query)

In [6]: cosine_similarity(Y, X.toarray())
Out[6]: array([[0.54267123, 0.44181486, 0.84003859, 0. ]])
```

6b Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

```
In [1]: # read in the iris data
        from sklearn.datasets import load_iris
        iris = load_iris()

        # create X (features) and y (response)
        X = iris.data
        y = iris.target

In [2]: # import the class
        from sklearn.linear_model import LogisticRegression

        # instantiate the model (using the default parameters)
        logreg = LogisticRegression()

        # fit the model with data
        logreg.fit(X, y)

        # predict the response values for the observations in X
        logreg.predict(X)
```

```
In [3]: # store the predicted response values
y_pred = logreg.predict(X)

# check how many predictions were generated
len(y_pred)
```

Out[3]: 150

```
In [4]: # compute classification accuracy for the logistic regression model
from sklearn import metrics

print(metrics.accuracy_score(y, y_pred))

0.9733333333333334
```

```
In [5]: from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
y_pred = knn.predict(X)
print(metrics.accuracy_score(y, y_pred))

0.9666666666666667
```

```
In [6]: knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
y_pred = knn.predict(X)
print(metrics.accuracy_score(y, y_pred))

1.0

In [7]: # print the shapes of X and y
# X is our features matrix with 150 x 4 dimension
print(X.shape)
# y is our response vector with 150 x 1 dimension
print(y.shape)

(150, 4)
(150,)
```

```
In [9]: # STEP 1: split X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)

In [10]: # print the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)

(90, 4)
(60, 4)

In [11]: # print the shapes of the new y objects
print(y_train.shape)
print(y_test.shape)

(90,)
(60,)

Out[11]: (90,)
```

```
In [12]: # STEP 2: train the model on the training set
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

Out[12]: LogisticRegression()

In [13]: # STEP 3: make predictions on the testing set
y_pred = logreg.predict(X_test)

# compare actual response values (y_test) with predicted response values (y_pred)
print(metrics.accuracy_score(y_test, y_pred))

0.9666666666666667
```

```
In [14]: knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))

0.9666666666666667

In [15]: knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))

0.9666666666666667

In [16]: # try K=1 through K=25 and record testing accuracy
k_range = range(1, 26)

# We can create Python dictionary using [] or dict()
scores = []
```

```
In [17]: # import Matplotlib (scientific plotting Library)
import matplotlib.pyplot as plt

# allow plots to appear within the notebook
%matplotlib inline

# plot the relationship between K and testing accuracy
# plt.plot(x_axis, y_axis)
plt.plot(k_range, scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Testing Accuracy')
```



```
In [18]: # instantiate the model with the best known parameters
knn = KNeighborsClassifier(n_neighbors=11)

# train the model with X and y (not X_train and y_train)
knn.fit(X, y)

# make a prediction for an out-of-sample observation
knn.predict([3, 5, 4, 2])

-----
ValueError                                Traceback (most recent call last)
<ipython-input-18-9d289456709c> in <module>
      6
      7 # make a prediction for an out-of-sample observation
----> 8 knn.predict([3, 5, 4, 2])

~/anaconda3/lib/site-packages\sklearn\neighbors\_classification.py in predict(self, X)
    171         Class labels for each data sample.
    172         """
--> 173         X = check_array(X, accept_sparse='csr')
    174
    175         neigh_dist, neigh_ind = self.kneighbors(X)

~/anaconda3/lib/site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
```

```
ValueError: Expected 2D array, got 1D array instead:  
array=[3 5 4 2].  
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) if it contains a  
single sample.
```

7a Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

```
In [1]: #Importing required Libraries
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import matplotlib.pyplot as plt

#Getting the data ready
data = load_iris()
df = data.data
#Selecting certain features based on which clustering is done
df = df[:,1:3]
```

```
In [2]: #Creating the model
agg_clustering = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage = 'ward')
#predicting the labels
labels = agg_clustering.fit_predict(df)
#Plotting the results
plt.figure(figsize = (8,5))
plt.scatter(df[labels == 0 , 0] , df[labels == 0 , 1] , c = 'red')
plt.scatter(df[labels == 1 , 0] , df[labels == 1 , 1] , c = 'blue')
plt.scatter(df[labels == 2 , 0] , df[labels == 2 , 1] , c = 'green')
plt.show()
```

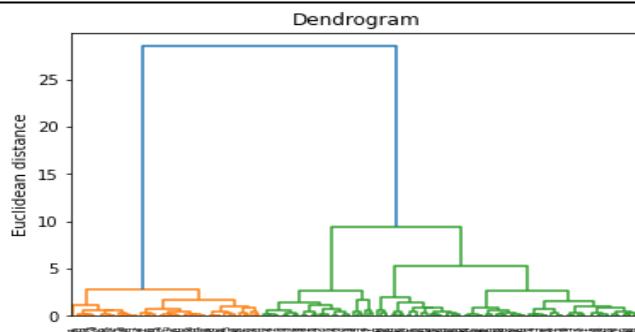


```
In [3]: #Importing Libraries
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram , linkage

#Getting the data ready
data = load_iris()
df = data.data
#Selecting certain features based on which clustering is done
df = df[:,1:3]

#Linkage Matrix
Z = linkage(df, method = 'ward')

#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
```



8a. 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

```
In [1]: import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

In [2]: # Define Parameter Enum values
# Age
ageEnum = ('SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4)
# Gender
genderEnum = ('Male':0, 'Female':1)
# FamilyHistory
familyHistoryEnum = ('Yes':0, 'No':1)
# Diet(Caloric Intake)
dietEnum = ('High':0, 'Medium':1, 'Low':2)
# Lifestyle
lifeStyleEnum = ('Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3)---  
# Cholesterol
cholesterolEnum = ('High':0, 'Borderline':1, 'Normal':2)
# HeartDisease
heartDiseaseEnum = ('Yes':0, 'No':1)
#heart_disease.data.csv

In [3]: with open('HeartDisease.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:
        data.append([ageEnum[x[0]], genderEnum[x[1]], familyHistoryEnum[x[2]], dietEnum[x[3]], lifeStyleEnum[x[4]], cholesterolEnum[x[5]], heartDiseaseEnum[x[6]]])
    data = np.array(data)
N = len(data)
```

```
In [4]: # Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:,4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:,5])

In [5]: # Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.Multinomial(1, p_heartdisease)
heartdisease.observe(data[:,6]) || meet.google.com is sharing a window.
```

```
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.Multinomial([age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:,6])
p_heartdisease.update()

# Sample Test with hardcoded values
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary,
#print(bp.nodes.Multinomial([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'],
#                           lifeStyleEnum['Sedetary']])))

In [*]: # Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.Multinomial([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter Diet: ' + str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))])
    print("Probability(HeartDisease) = " + str(res))
    #print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1"))
```

Enter Age: ('SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4)
 Enter Gender: {'Male': 0, 'Female': 1}
 Enter FamilyHistory: {'Yes': 0, 'No': 1}
 Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}
 Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
 Enter Cholesterol: {'High': 0, 'Borderline': 1, 'Normal': 2}
 Probability(HeartDisease) = 0.5
 Enter for Continue:0, Exit :1

b. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

jupyter PRAC 8B Last Checkpoint: 6 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

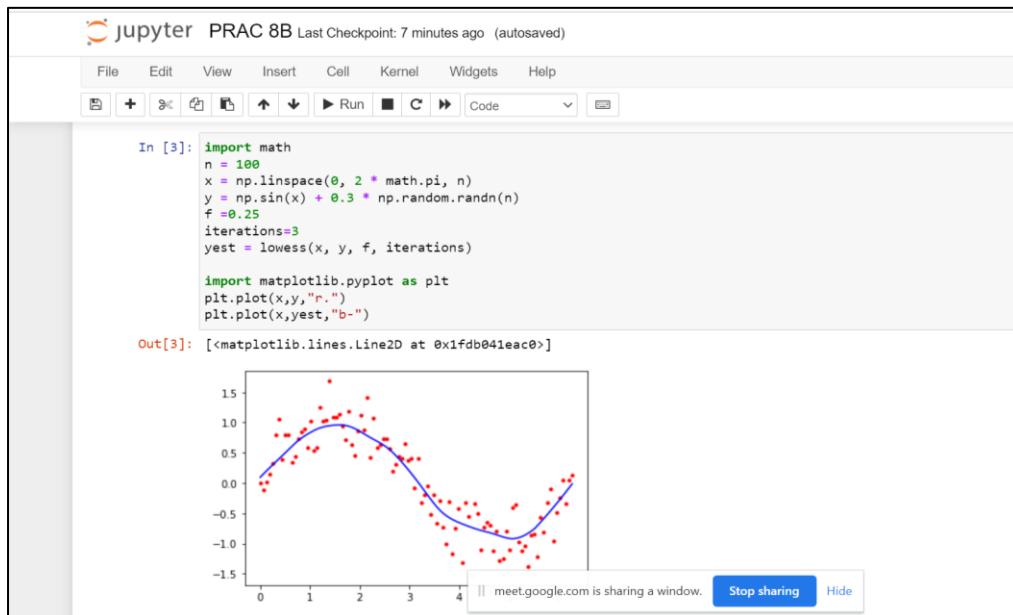
In [1]: `from math import ceil
import numpy as np
from scipy import linalg`

In [2]: `def lowess(x, y, f, iterations):
 n = len(x)
 r = int(ceil(f * n))
 h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
 w = np.clip(np.abs([x[:, None] - x[None, :]]) / h, 0.0, 1.0)
 w = (1 - w ** 3) ** 3
 yest = np.zeros(n)
 delta = np.ones(n)
 for iteration in range(iterations):
 for i in range(n):
 weights = delta * w[:, i]
 b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
 A = np.array([[np.sum(weights), np.sum(weights * x)], [np.sum(weights * x), np.sum(weights * x * x)]])
 beta = linalg.solve(A, b)
 yest[i] = beta[0] + beta[1] * x[i]

 residuals = y - yest
 s = np.median(np.abs(residuals))
 delta = np.clip(residuals / (6.0 * s), -1, 1)
 delta = (1 - delta ** 2) ** 2

 return yest`

|| meet.google.com is sharing a window. Stop sharing Hide



9A. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```

jupyter PRAC 9A Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help
Trusted Python

In [1]: import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)      # X = (hours sleeping, hours studying)
y = np.array([92, [86], [89]], dtype=float)                # y = score on test

# scale units
X = X/np.amax(X, axis=0)                                # maximum of X array
y = y/100                                                 # max test score is 100

```

```

In [2]: class Neural_Network(object):
    def __init__(self):
        # Parameters
        self.inputSize = 2
        self.outputSize = 1
        self.hiddenSize = 3
        # Weights
        self.W1 = np.random.randn(self.inputSize, self.hiddenSize)      # (3x2) weight matrix from input to hidden Layer
        self.W2 = np.random.randn(self.hiddenSize, self.outputSize)     # (3x1) weight matrix from hidden to output Layer

    def forward(self, X):
        #forward propagation through our network
        self.z = np.dot(X, self.W1)                                     # dot product of X (input) and first set of 3x2 weights
        self.z2 = self.sigmoid(self.z)                                 # activation function
        self.z3 = np.dot(self.z2, self.W2)                             # dot product of hidden Layer (z2) and second set of 3x1 weights
        o = self.sigmoid(self.z3) || meet.google.com is sharing a window.
        return o

```

```

jupyter PRAC 9A Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help
Logout Trusted Python 3

def sigmoid(self, s):
    return 1/(1+np.exp(-s))      # activation function

def sigmoidPrime(self, s):
    return s * (1 - s)          # derivative of sigmoid

def backward(self, X, y, o):
    # backward propagate through the network
    self.o_error = y - o        # error in output
    self.o_delta = self.o_error*self.sigmoidPrime(o) # applying derivative of sigmoid to
    self.z2_error = self.o_delta.dot(self.W2.T)      # z2 error: how much our hidden Layer weights contributed to output error
    self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error
    self.W1 += X.T.dot(self.z2_delta)                 # adjusting first set (input --> hidden) weights
    self.W2 += self.z2.T.dot(self.o_delta)             # adjusting second set (hidden --> output) weights

def train (self, X, y):
    o = self.forward(X)
    self.backward(X, y, o)

In [3]: NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
    print ("Input: " + str(X))
    print ("Actual Output: " + str(y))
    print ("Predicted Output: " + str(NN.forward(X)))
    print ("Loss: " + str(np.mean(np.square(y - NN.forward(X))))) # mean sum squared loss
    NN.train(X, y)

```

jupyter PRAC 9A Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

```
print ("\nInput: " + str(X))
print ("\nActual Output: " + str(y))
print ("\nPredicted Output: " + str(NN.forward(X)))
print ("\nLoss: " + str(np.mean(np.square(y - NN.forward(X)))))      # mean sum squared Loss
NN.train(X, y)
```

Input:
[[0.66666667 1.
 [0.33333333 0.55555556
 [1. 0.66666667]]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]]

Predicted Output:
[[0.43002716]
 [0.48592398]
 [0.37913027]]]

Loss:
0.23574876335276804

jupyter PRAC 9A Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

```
print ("\nInput: " + str(X))
print ("\nActual Output: " + str(y))
print ("\nPredicted Output: " + str(NN.forward(X)))
print ("\nLoss: " + str(np.mean(np.square(y - NN.forward(X)))))      # mean sum squared Loss
NN.train(X, y)
```

Loss:
0.0898885991386246

Input:
[[0.66666667 1.
 [0.33333333 0.55555556
 [1. 0.66666667]]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]]

Predicted Output:
[[0.60822351]
 [0.60179461]
 [0.64604075]]]

jupyter PRAC 9A Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Code

```
print ("\nInput: " + str(X))
print ("\nActual Output: " + str(y))
print ("\nPredicted Output: " + str(NN.forward(X)))
print ("\nLoss: " + str(np.mean(np.square(y - NN.forward(X)))))      # mean sum squared Loss
NN.train(X, y)
```

0.02506664843147398

Input:
[[0.66666667 1.
 [0.33333333 0.55555556
 [1. 0.66666667]]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]]

Predicted Output:
[[0.73664301]
 [0.7263411]
 [0.79781391]]]

Loss:
0.01999425391900239

9B. ASSUMING A SET OF DOCUMENTS THAT NEED TO BE CLASSIFIED, USE THE NAÏVE BAYESIAN CLASSIFIER MODEL TO PERFORM THIS TASK. BUILT-IN JAVA CLASSES/API CAN BE USED TO WRITE THE PROGRAM. CALCULATE THE ACCURACY, PRECISION, AND RECALL FOR YOUR DATA SET.

In [1]:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np
```

In [2]:

```
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
```

In [4]:

```
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)
print("\n".join(twenty_train.data[0].split("\n")))
print(twenty_train.target[0])
```

2257
1502
['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.christian']
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton
Organization: The City University
Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to convert tif/jpg/tga files into La:
do the same, converting to HPGL (|| meet.google.com is sharing a window. Stop sharing Hide

Please email any response.
Is this the correct group?
Thanks in advance. Michael.
--
Michael Collier (Programmer)
Email: M.P.Collier@uk.ac.city
Tel: 071 477-8000 x3769
Fax: 071 477-8565
The Computer Unit,
The City University,
London,
EC1V 0HB.

1

In []:

In [5]:

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
```

In [6]:

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
```

Out[6]: (2257, 35788) || meet.google.com is sharing a window. Stop sharing Hide

The screenshot shows a Jupyter Notebook interface running on localhost:8891/notebooks/PRAC%209B.ipynb. The notebook has two cells:

In [7]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
```

In [8]:

```
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_names))
print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

Output of In [8]:

	precision	recall	f1-score	support
alt.atheism	0.97	0.60	0.74	319
comp.graphics	0.96	0.89	0.92	389
sci.med	0.97	0.81	0.88	396
soc.religion.christian	0.65	0.99	0.78	398
accuracy			0.83	1502
macro avg	0.89	0.82	0.83	1502
weighted avg	0.88	0.83	0.84	1502

confusion matrix is

```
[[192  2  6 119]
 [ 2 347  4 36]
 [ 2 11 322 61]
 [ 2  2  1 393]]
```

Bottom right corner of the window shows a sharing message: "meet.google.com is sharing a window. Stop sharing Hide".

10A . 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.

```

jupyter PRAC 10A Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help
In [1]: import numpy as np
import math
import csv

In [2]: def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)

    return (metadata, traindata)

In [3]: class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""

    def __str__(self):
        return self.attribute

In [4]: def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype='|S32')
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
    if delete:
        dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict

```

```

jupyter PRAC 10A Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help
In [4]: def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1

    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype='|S32')
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
    if delete:
        dict[items[x]] = np.delete(dict[items[x]], col, 1)

    return items, dict

In [5]: def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0
    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

```

In [5]:

```
def entropy(S):
    items = np.unique(S)

    if items.size == 1:
        return 0

    counts = np.zeros((items.shape[0], 1))
    sums = 0

    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)

    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
```

In [6]:

```
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)

    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))

    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)

    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)

    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]

    return total_entropy / iv
```

```
total_size = data.shape[0]
entropies = np.zeros((items.shape[0], 1))
intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):
    ratio = dict[items[x]].shape[0]/(total_size * 1.0)
    entropies[x] = ratio * entropy(dict[items[x]][:, -1])
    intrinsic[x] = ratio * math.log(ratio, 2)

total_entropy = entropy(data[:, -1])
iv = -1 * sum(intrinsic)

for x in range(entropies.shape[0]):
    total_entropy -= entropies[x]

return total_entropy / iv
```

In [7]:

```
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node

    gains = np.zeros((data.shape[1] + 1, 1))

    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)

    split = np.argmax(gains)

    node = Node(metadata[split])
    metadata = np.delete(metadata, split)

    return node
```

```
gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):
    gains[col] = gain_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split])
metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):
    child = create_node(dict[items[x]], metadata)
    node.children.append((items[x], child))

return node

In [8]: def empty(size):
    s = ""
    for x in range(size):
        s += "    "
    return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1),
              print_tree(n, level + 2) || meet.google.com is sharing a window.
```

```
return s

def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)

In [9]: metadata, traindata = read_data("tennisdata.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)

Outlook
Overcast
b'Yes'
Rainy
Windy
b'FALSE'
b'Yes'
b'TRUE'
b'No'
Sunny
Humidity
b'High'
b'No'
b'Normal'
b'Yes'
```