Week-1 Find-S

```
import io
import csv
import pandas as pd
num_attributes=6
a=[]
print("The given Dataset is")
with open("C:\\Users\\vshiv\\OneDrive\\Desktop\\MLDS CSV FILES\\Climate1.csv",'r') as csvfile:
 reader=csv.reader(csvfile)
 for row in reader:
  a.append(row)
  print(row)
print("The initial hypothesis is")
hypothesis=['0']*num_attributes
print(hypothesis)
for j in range(0,num_attributes):
 hypothesis[j]=a[1][j]

print(hypothesis)
#Find the Maximum specific Hypothesis
print("Find S: Finding maximal Specific Hypothesis")
for i in range(1,len(a)):
  if a[i][num_attributes]=='Yes' or a[i][num_attributes]== 'yes':
   for j in range(0,num_attributes):
     if a[i][j]!=hypothesis[j]:
       hypothesis[j]='?'
     else :
       hypothesis[j]=a[i][j]
  print("For Training instance No: {0} the hypothesis is ". format(i), hypothesis)
print("The hypothesis is:")
```

print(hypothesis)

O/P:

The given Dataset is

['sky', 'Airtemp', 'Humidity', 'Wind', 'Water ', 'Forecast ', 'EnjoySport']

['Sunny', 'Warm ', 'Normal ', 'Strong ', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm ', 'High', 'Strong ', 'Warm', 'Same', 'Yes']

['Rainy ', 'Cold', 'High', 'Strong ', 'Warm', 'change', 'No']

['Sunny', 'Warm ', 'High', 'Strong ', 'Cool', 'change', 'Yes']

The initial hypothesis is

['0', '0', '0', '0', '0', '0']

['Sunny', 'Warm ', 'Normal ', 'Strong ', 'Warm', 'Same']

Find S: Finding maximal Specific Hypothesis

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

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

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

For Training instance No: 4 the hypothesis is ['Sunny', 'Warm ', '?', 'Strong ', '?', '?']

The hypothesis is:

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

Week-5:

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

import pandas as pd

import numpy as np

iris=datasets.load_iris()

print("Iris Data set loaded...")

x_train, x_test, y_train, y_test= train_test_split(iris.data,iris.target,test_size=0.2)

print("Dataset is split into Training and Testing...")

print("Size of training data and it's label",x_train.shape,y_train.shape)

print("Size of Test data and it's label",x_test.shape,y_test.shape)
```

```
for i in range(len(iris.target_names)):
 print("Label",i,"-",str(iris.target_names[i]))
classifier=KNeighborsClassifier(n_neighbors=1)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
print("Result of Classification using K-NN with K=1")
for r in range(0,len(x_test)):
 print("Sample:",str(x_test[r]),"Actual-label:",str(y_test[r]),"PredictedLabel:",str(y_pred[r]))
print("Classification Accuracy:",classifier.score(x_test,y_test))
from sklearn.metrics import classification_report, confusion_matrix
print("Confusion Matrix")
print(confusion_matrix(y_test,y_pred))
print("Accuracy Matrix")
print(classification_report(y_test,y_pred))
```

O/P:

```
Iris Data set loaded...
Dataset is split into Training and Testing...
Size of training data and it's label (120, 4) (120,)
Size of Test data and it's label (30, 4) (30,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Result of Classification using K-NN with K=1
Sample: [6.  2.2 4.  1. ] Actual-label: 1 PredictedLabel: 1
Sample: [6.5 3.  5.8 2.2] Actual-label: 2 PredictedLabel: 2
Sample: [5.2 4.1 1.5 0.1] Actual-label: 0 PredictedLabel: 0
Sample: [5.  3.5 1.3 0.3] Actual-label: 0 PredictedLabel: 0
Sample: [4.9 2.4 3.3 1. ] Actual-label: 1 PredictedLabel: 1
Sample: [7.7 3.8 6.7 2.2] Actual-label: 2 PredictedLabel: 2
Sample: [6.1 3.  4.9 1.8] Actual-label: 2 PredictedLabel: 2
```

Sample: [6.4 3.1 5.5 1.8] Actual-label: 2 PredictedLabel: 2

Sample: [6.4 3.2 5.3 2.3] Actual-label: 2 PredictedLabel: 2

Sample: [6.2 2.9 4.3 1.3] Actual-label: 1 PredictedLabel: 1

Sample: [4.9 3. 1.4 0.2] Actual-label: 0 PredictedLabel: 0

Sample: [6.5 2.8 4.6 1.5] Actual-label: 1 PredictedLabel: 1

Sample: [4.4 3.2 1.3 0.2] Actual-label: 0 PredictedLabel: 0

Sample: [4.8 3.4 1.9 0.2] Actual-label: 0 PredictedLabel: 0

Sample: [6.3 2.9 5.6 1.8] Actual-label: 2 PredictedLabel: 2

Sample: [5.7 2.8 4.1 1.3] Actual-label: 1 PredictedLabel: 1

Sample: [5.8 2.7 3.9 1.2] Actual-label: 1 PredictedLabel: 1

Sample: [5.7 3. 4.2 1.2] Actual-label: 1 PredictedLabel: 1

Sample: [5.9 3. 4.2 1.5] Actual-label: 1 PredictedLabel: 1

Sample: [6.3 2.3 4.4 1.3] Actual-label: 1 PredictedLabel: 1

Sample: [6. 2.9 4.5 1.5] Actual-label: 1 PredictedLabel: 1

Sample: [4.4 2.9 1.4 0.2] Actual-label: 0 PredictedLabel: 0

Sample: [6.1 2.6 5.6 1.4] Actual-label: 2 PredictedLabel: 2

Sample: [7.6 3. 6.6 2.1] Actual-label: 2 PredictedLabel: 2

Sample: [6. 2.7 5.1 1.6] Actual-label: 1 PredictedLabel: 2

Sample: [5.1 3.8 1.9 0.4] Actual-label: 0 PredictedLabel: 0

Sample: [5.5 2.3 4. 1.3] Actual-label: 1 PredictedLabel: 1

Sample: [5.8 2.6 4. 1.2] Actual-label: 1 PredictedLabel: 1

Sample: [5. 3.2 1.2 0.2] Actual-label: 0 PredictedLabel: 0

Sample: [6.4 2.7 5.3 1.9] Actual-label: 2 PredictedLabel: 2

Classification Accuracy: 0.9666666666666667

Confusion Matrix

[[8 0 0]

 [0 12 1]

 [0 0 9]]

Accuracy Matrix

            precision   recall f1-score   support

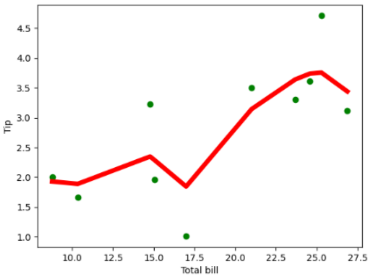
```
        0       1.00      1.00      1.00         8
        1       1.00      0.92      0.96        13
        2       0.90      1.00      0.95         9

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

Week-3:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point, xmat, k):
   m,n = np.shape(xmat)
   weights = np.mat(np.eye((m)))
   for j in range(m):
     diff = point - X[j]
     print("Point",point)
     print("Diff",diff)
     weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
     print("Weights",weights)
   return weights
def localWeight(point, xmat, ymat, k):
  wei = kernel(point,xmat,k)
  W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
  print("W",W)
  return W
def localWeightRegression(xmat, ymat, k):
  m,n = np.shape(xmat)
  ypred = np.zeros(m)
  for i in range(m):
```

```
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
  return ypred
# load data points
data = pd.read_csv("E:\\3-2\\MLDS\\Datasets\\week3.csv")
bill = np.array(data.total_bill)
tip = np.array(data.tip)
#preparing and add 1 in bill
mbill = np.mat(bill)
print("MBILL",mbill)
mtip = np.mat(tip)
print("Mtip",mtip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
print("X",X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

O/P:



MBILL [[16.99 10.34 21.01 23.68 24.59 25.29 8.77 26.88 15.04 14.78]]

Mtip [[1.01 1.66 3.5 3.31 3.61 4.71 2.   3.12 1.96 3.23]]

```
X [[ 1.   16.99]
 [ 1.   10.34]
 [ 1.   21.01]
 [ 1.   23.68]
 [ 1.   24.59]
 [ 1.   25.29]
 [ 1.    8.77]
 [ 1.   26.88]
 [ 1.   15.04]
 [ 1.   14.78]]
Point [[ 1.   16.99]]
Diff [[0. 0.]]
```

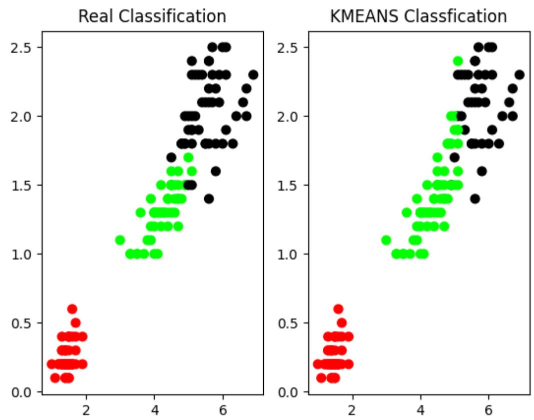
Week-4:

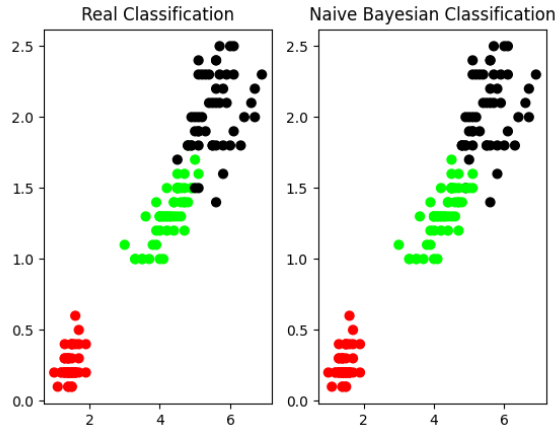
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris =datasets.load_iris()
X=pd.DataFrame(iris.data)
print(X.shape)
```

```
X.columns=['Sepal_Length','Sepal_Width', 'Petal_length', 'Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['target']
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
plt.subplot(1,2,1)
plt.scatter(X.Sepal_Length,X.Sepal_Width,c=colormap[y.target],s=40)
plt.title('Sepal')
plt.subplot(1,2,2)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)
plt.title('Petal')
plt.show()
model=KMeans(n_clusters=3)
model.fit(X)
print(model.labels_)
plt.subplot(1,2,1)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)
plt.title('Real Classification')
plt.subplot(1,2,2)
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[model.labels_],s=40)
plt.title( 'KMEANS Classfication')
plt.show()
print(sm.accuracy_score(y,model.labels_))
print(sm.confusion_matrix(y,model.labels_))

from sklearn.naive_bayes import GaussianNB
clf=GaussianNB()
clf.fit(X,y)
y_cluster_gmm=clf.predict(X)
print(y_cluster_gmm)
plt.subplot(1,2,1)
```

```
plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y.target],s=40)

plt.title('Real Classification')

plt.subplot(1,2,2)

plt.scatter(X.Petal_length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)

plt.title("Naive Bayesian Classification")

plt.show()

print(sm.accuracy_score(y,y_cluster_gmm))

print(sm.confusion_matrix(y,y_cluster_gmm))

#print(confusion_matrix)
```

O/P:





[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2
 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
0.96
[[50  0  0]
 [ 0 47  3]
```

Week-2

```
import pandas as pd
df=pd.read_csv("C:\\Users\\vshiv\\OneDrive\\Desktop\\MLDS CSV FILES\\lab2dataset.csv")
print(df)
def entropy(probs):
 import math
 return sum(-prob*math.log(prob,2) for prob in probs)
def entropy_of_list(a_list):
 from collections import Counter
 cnt = Counter (x for x in a_list)
 num_instances =len(a_list)
 probs=[x/num_instances for x in cnt.values()]
 return entropy(probs)
total_entropy= entropy_of_list(df['PlayTime'])
print(total_entropy)
def information_gain(df,split_attribute_name, target_attribute_name, trace=0):
  df_split =df.groupby(split_attribute_name)
  for name,group in df_split:
    nobs=len(df.index)*1.0
    df_agg_ent=df_split.agg({target_attribute_name: [entropy_of_list,lambda x: len(x)/nobs]
})[target_attribute_name]
    avg_info=sum(df_agg_ent['entropy_of_list'] * df_agg_ent['<lambda_0>'])
    old_entropy=entropy_of_list(df[target_attribute_name])
```

```
   return old_entropy-avg_info
def id3DT(df, target_attribute_name, attribute_names, default_class=None):
  from collections import Counter
  cnt = Counter(x for x in df[target_attribute_name])
  if len(cnt)==1:
     return next(iter(cnt))
  elif df.empty or (not attribute_names):
     return default_class
  else:
     default_class =max(cnt.keys())
     gainz=[information_gain(df,attr, target_attribute_name) for attr in attribute_names]
     index_of_max=gainz.index(max(gainz))
     best_attr=attribute_names[index_of_max]
     tree={best_attr:{}}
     remaining_attributes_names=[i for i in attribute_names if i != best_attr]
     for attr_val, data_subset in df.groupby(best_attr):
      subtree=id3DT(data_subset,target_attribute_name,remaining_attributes_names,default_class)
      tree[best_attr][attr_val]=subtree
  return tree
attribute_names=list(df.columns)
attribute_names.remove('PlayTime')
from pprint import pprint
tree= id3DT(df,'PlayTime',attribute_names)
print("The Resultant Decision Tree is ")
pprint(tree)
attribute=next(iter(tree))
print("Best Attribute: \n", attribute)
print("Tree Keys\n ", tree[attribute].keys())
def classify(instance, tree, default=None):
  attribute=next(iter(tree))
  print("Key:",tree.keys())
```

```
  print("Attribute",attribute)
  if instance[attribute] in tree[attribute].keys():
    result=tree[attribute][instance[attribute]]
    print("Instance Attribute:",instance[attribute], "TreeKeys:",tree[attribute].keys())
    if isinstance(result,dict):
      return classify(instance,result)
    else:
      return result
  else:
    return default
tree1={'Outlook':['Rainy','Sunny'],'Temperature':['Mild','Hot'],'Humidity':['Normal','High'],'Windy':['Weak','Strong']}
df2=pd.DataFrame(tree1)
df2['Predicted']=df2.apply(classify,axis=1, args=(tree,'No'))
print(df2)
```

```
Outlook Temperature Humidity    Wind PlayTime
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  Overcast       Mild     High  Strong      Yes
11  Overcast        Hot  Normal    Weak      Yes
12     Rain       Mild     High  Strong       No
0.961236604722876
```

The Resultant Decision Tree is

```
{'Outlook': {'Overcast': 'Yes',
        'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
        'Sunny': {'Temperature': {'Cool': 'Yes',
                          'Hot': 'No',
                          'Mild': 'No'}}}}
```

```
Best Attribute:
 Outlook
Tree Keys
  dict_keys(['Overcast', 'Rain', 'Sunny'])
Key: dict_keys(['Outlook'])
Attribute Outlook
Key: dict_keys(['Outlook'])
Attribute Outlook
Instance Attribute: Sunny TreeKeys: dict_keys(['Overcast', 'Rain', 'Sunny'])
Key: dict_keys(['Temperature'])
Attribute Temperature
Instance Attribute: Hot TreeKeys: dict_keys(['Cool', 'Hot', 'Mild'])
  Outlook Temperature Humidity   Windy Predicted
0   Rainy        Mild   Normal    Weak        No
1   Sunny         Hot     High  Strong        No
```