Exp 1

```
import pandas as pd
import numpy as np
data = pd.read_csv("animal.csv")
print(data,"n")
d = np.array(data)[:,:-1]
print("\n The attributes are:\n ",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)
def train(c,t):
   for i, val in enumerate(t):
      if val == "yes":
         specific_hypothesis = c[i].copy()
         break
      for i, val in enumerate(c):
      if t[i] == "yes":
         for x in range(len(specific_hypothesis)):
            if val[x] != specific_hypothesis[x]:
               specific_hypothesis[x] = '?'
            else:
               pass
       return specific_hypothesis
 print("\n The final hypothesis is:",train(d,target))
```

Exp 2

```
import numpy as np
import pandas as pd     (separate)

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

print(data)       (separate)

concepts = np.array(data.iloc[:,0:-1])
print(concepts)      (separate)

target = np.array(data.iloc[:,-1])
print(target)    (separate)



def learn(concepts, target):

   '''
   learn() function implements the learning method of the Candidate
elimination algorithm.
   Arguments:
      concepts - a data frame with all the features
      target - a data frame with corresponding output values
   '''

   specific_h = concepts[0].copy()
   print("\nInitialization of specific_h and general_h")
   print(specific_h)

   general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
   print(general_h)

 for i, h in enumerate(concepts):

       if target[i] == "Yes":
```

```
        for x in range(len(specific_h)):
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x][x] = '?'

 if target[i] == "No":
        for x in range(len(specific_h)):
            if h[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = '?'

      print("\nSteps of Candidate Elimination Algorithm",i+1)
      print(specific_h)
      print(general_h)

indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
   for i in indices:

general_h.remove(['?', '?', '?', '?', '?', '?'])

 return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("\nFinal Specific_h:", s_final, sep="\n")
print("\nFinal General_h:", g_final, sep="\n")
```

Exp 3

```
import numpy as np
def find_entropy(df):
  Class = df.keys()[-1]
  values = df[Class].unique()
  entropy = 0
  for value in values:
    prob = df[Class].value_counts()[value]/len(df[Class])
    entropy += -prob * np.log2(prob)
  return np.float(entropy)
def find_entropy_attribute(df, attribute):
  Class = df.keys()[-1]
  target_values = df[Class].unique()
  attribute_values = df[attribute].unique()
  avg_entropy = 0
  for value in attribute_values:
    entropy = 0
    for value1 in target_values:
      num = len(df[attribute][df[attribute] == value][df[Class] == value1])
      den = len(df[attribute][df[attribute] == value])
      prob = num/den
      entropy += -prob * np.log2(prob + 0.000001)
    avg_entropy += (den/len(df))*entropy
  return np.float(avg_entropy) (separate)
def find_winner(df):
```

```
  IG = []


    IG.append(find_entropy(df) - find_entropy_attribute(df, key))
  return df.keys()[:-1][np.argmax(IG)]  (separate)
def get_subtable(df, attribute, value):
  return df[df[attribute] == value].reset_index(drop = True)  (separate)
def buildtree(df, tree = None):
  node = find_winner(df)
  attvalue = np.unique(df[node])
  Class = df.keys()[-1]
  if tree is None:
    tree = {}
    tree[node] = {}
  for value in attvalue:
    subtable = get_subtable(df,node,value)
    (lvalue, counts = np.unique(subtable[Class], return_counts = True)
    if len(counts) == 1:
      tree[node][value] = Clvalue[0]
    else:
      tree[node][value] = buildtree(subtable)
  return tree (separate)
import pandas as pd
df = pd.read_csv('PlayTennis.csv') (separate)
tree = build tree(df) (separate)
```

```
import pprint
pprint.pprint(tree)  (separate)
test={'Outlook':'Sunny','Temperature':'Hot','Humidity':'High','Wind':'Weak'}
def func(test, tree, default=None):
    attribute = next(iter(tree))
    print(attribute)
    if test[attribute] in tree[attribute].keys():
        print(tree[attribute].keys())
        print(test[attribute])
        result = tree[attribute][test[attribute]]
        if isinstance(result, dict):
            return func(test, result)
        else:
            return result
    else:
        return default
ans = func(test, tree)
print(ans)
```

Exp 4

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100  (separate)
def sigmoid (x):
 return 1/(1 + np.exp(-x))  (separate)
def derivatives_sigmoid(x):
 return x * (1 - x)  (separate)
epoch=1 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer  (separate)
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))  (separate)
for i in range(epoch):
hinp1=np.dot(X,wh)
 hinp=hinp1 + bh
 hlayer_act = sigmoid(hinp)
 outinp1=np.
(hlayer_act,wout)
```

```
 outinp= outinp1+bout
 output = sigmoid(outinp) (separate)
EO = y-output
 outgrad = derivatives_sigmoid(output)
 d_output = EO * outgrad
 EH = d_output.dot(wout.T)
 hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
 d_hiddenlayer = EH * hiddengrad

 wout += hlayer_act.T.dot(d_output) *lr # dotproduct of nextlayererror and currentlayerop
 wh += X.T.dot(d_hiddenlayer) *lr

 print("Input: \n" + str(X))
 print("Actual Output: \n" + str(y))
 print("Predicted Output: \n" ,output)
```

exp 5

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
df = pd.read_csv("diabetes.csv")
feature_col_names = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin',
                'BMI', 'DiabetesPedigreeFunction', 'Age']
predicted_class_names = ['Outcome']
X = df[feature_col_names].values # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n Accuracy of the classifier
is',metrics.accuracy_score(ytest,predicted))
print('\n The value of Precision', metrics.precision_score(ytest,predicted))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
print("Predicted Value for individual Test Data:", predictTestData)
```

exp 7

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt    (separate)
names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class'] (separate)
dataset = pd.read_csv("8-dataset.csv", names=names)  (separate)
X = dataset.iloc[:, :-1]  (separate)
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2} (separate)
y = [label[c] for c in dataset.iloc[:, -1]]  (separate)
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])    (separate)
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])  (separate)
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
```

```
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y,
model.labels_))
```
(separate)

```
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)

y_cluster_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])


print('The accuracy score of EM: ',metrics.accuracy_score(y,
y_cluster_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y,
y_cluster_gmm))
```

exp 8

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

dataset = pd.read_csv("8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)
classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label',
'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
   print ('%-25s %-25s' % (label, ypred[i]), end="")
```

```
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

exp 9

```
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]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    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
data = pd.read_csv('9-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
#preparing and add 1 in bill
```

```
mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
ypred = localWeightRegression(X,mtip,0.5)
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();
```