***INDEX***

|  |  |  |
| --- | --- | --- |
| Sl. No. | Program Name | Page No. |
| 1. | 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. | 02 |
| 2. | Demonstrate the working of the decision tree. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. | 05 |
| 3. | Build an Artificial Neural Network by test the same using appropriate data sets. | 07 |
| 4. | 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. | 13 |
| 5. | Implement Find S Algorithm. | 15 |
| 6. | Apply K-Means clustering algorithm to cluster data stored in a .CSV file. | 17 |
| 7. | Implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. | 20 |
| 8. | Implement the parametric Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs. | 22 |

***1. 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.***

import csv

a = []

print("\n The Given Training Data Set")

with open('enjoysport.csv', 'r') as csvFile:

    reader = csv.reader(csvFile)

    for row in reader:

        a.append (row)

        print(row)

num\_attributes = len(a[0])-1

print("\n The initial value of hypothesis: ")

S = ['0'] \* num\_attributes

G = ['?'] \* num\_attributes

print ("\n The most specific hypothesis S0 : [0,0,0,0,0,0]")

print (" \n The most general hypothesis G0 : [?,?,?,?,?,?]")

for j in range(0,num\_attributes):

    S[j]=a[0][j]

print("\n Candidate Elimination algorithm Hypotheses Version Space Computation\n")

temp=[]

for i in range(0,len(a)):

    if a[i][num\_attributes]=='Yes':

        for j in range(0,num\_attributes):

            if a[i][j]!=S[j]:

                S[j]='?'

        for j in range(0,num\_attributes):

            for k in range(1,len(temp)):

                if temp[k][j]!='?' and temp[k][j]!=S[j]:

                    del temp[k]

        print("----------------------------------------------------------------------------- ")

        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)

        if (len(temp)==0):

            print(" For Training Example No :{0} the hypothesis is G{0} ".format(i+1),G)

        else:

            print(" For  Positive Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)

    if a[i][num\_attributes]=='No':

        for j in range(0,num\_attributes):

            if S[j] != a[i][j] and S[j]!= '?':

                G[j]=S[j]

                temp.append(G)

                G = ['?'] \* num\_attributes

        print("----------------------------------------------------------------------------- ")

        print(" For Training Example No :{0} the hypothesis is S{0} ".format(i+1),S)

        print(" For Training Example No :{0} the hypothesis is G{0}".format(i+1),temp)

**Output**

The Given Training Data Set

['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes']

['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'Yes']

['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'No']

['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'Yes']

The initial value of hypothesis:

The most specific hypothesis S0 : [0,0,0,0,0,0]

The most general hypothesis G0 : [?,?,?,?,?,?]

Candidate Elimination algorithm Hypotheses Version Space Computation

-----------------------------------------------------------------------------

For Training Example No :1 the hypothesis is S1 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

For Training Example No :1 the hypothesis is G1 ['?', '?', '?', '?', '?', '?']

-----------------------------------------------------------------------------

For Training Example No :2 the hypothesis is S2 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No :2 the hypothesis is G2 ['?', '?', '?', '?', '?', '?']

-----------------------------------------------------------------------------

For Training Example No :3 the hypothesis is S3 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

For Training Example No :3 the hypothesis is G3 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

-----------------------------------------------------------------------------

For Training Example No :4 the hypothesis is S4 ['sunny', 'warm', '?', 'strong', '?', '?']

For Positive Training Example No :4 the hypothesis is G4 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

***2. Demonstrate the working of the Decision Tree. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.***

import pandas as pd # Load libraries

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

from sklearn.model\_selection import train\_test\_split # Import train\_test\_split function

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

from sklearn import tree #Import scikit-learn tree module for plotting

import matplotlib.pyplot as plt #Import matplotlib for plotting

# load dataset

pima = pd.read\_csv("diabetes.csv")

pima.columns

#split dataset in features and target variable

feature\_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age', 'Glucose', 'BloodPressure', 'DiabetesPedigreeFunction']

X = pima[feature\_cols] # Features

y = pima.Outcome # Target variable

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1) # 70% training and 30% test

# Create Decision Tree classifer object

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer

clf = clf.fit(X\_train,y\_train)

#Predict the response for test dataset

y\_pred = clf.predict(X\_test)

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

plt.figure(figsize=(10,8))

tree.plot\_tree(clf, feature\_names=feature\_cols, class\_names=['0','1'], label=all, filled=True, proportion=True, rounded=True, fontsize=3)

print('Number of correct predictions are')

diff = list(y\_pred - y\_test)

print(diff.count(0)," out of ",len(diff))

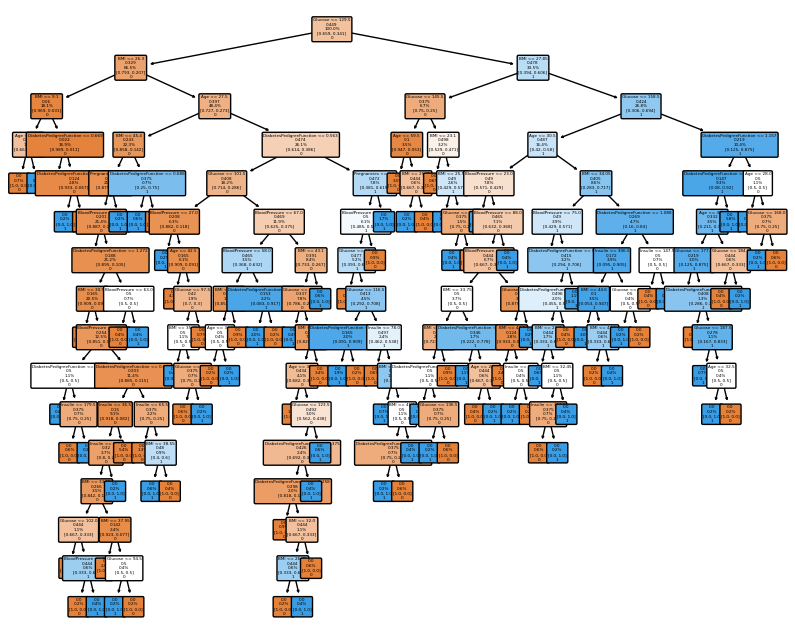
**Output**

Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],

dtype='object')

Accuracy: 0.683982683982684



***3. Build an Artificial Neural Network by testing the same using appropriate data sets.***

import numpy as np

import pandas as pd

from matplotlib import pyplot as plt

data = pd.read\_csv('digit-recognizer/train.csv')

data = np.array(data)

m, n = data.shape

np.random.shuffle(data) # shuffle before splitting into dev and training sets

data\_dev = data[0:1000].T

Y\_dev = data\_dev[0]

X\_dev = data\_dev[1:n]

X\_dev = X\_dev / 255.

data\_train = data[1000:m].T

Y\_train = data\_train[0]

X\_train = data\_train[1:n]

X\_train = X\_train / 255.

\_,m\_train = X\_train.shape

Y\_train

def init\_params():

    W1 = np.random.rand(10, 784) - 0.5

    b1 = np.random.rand(10, 1) - 0.5

    W2 = np.random.rand(10, 10) - 0.5

    b2 = np.random.rand(10, 1) - 0.5

    return W1, b1, W2, b2

def ReLU(Z):

    return np.maximum(Z, 0)

def softmax(Z):

    A = np.exp(Z) / sum(np.exp(Z))

    return A

def forward\_prop(W1, b1, W2, b2, X):

    Z1 = W1.dot(X) + b1

    A1 = ReLU(Z1)

    Z2 = W2.dot(A1) + b2

    A2 = softmax(Z2)

    return Z1, A1, Z2, A2

def ReLU\_deriv(Z):

    return Z > 0

def one\_hot(Y):

    one\_hot\_Y = np.zeros((Y.size, Y.max() + 1))

    one\_hot\_Y[np.arange(Y.size), Y] = 1

    one\_hot\_Y = one\_hot\_Y.T

    return one\_hot\_Y

def backward\_prop(Z1, A1, Z2, A2, W1, W2, X, Y):

    one\_hot\_Y = one\_hot(Y)

    dZ2 = A2 - one\_hot\_Y

    dW2 = 1 / m \* dZ2.dot(A1.T)

    db2 = 1 / m \* np.sum(dZ2)

    dZ1 = W2.T.dot(dZ2) \* ReLU\_deriv(Z1)

    dW1 = 1 / m \* dZ1.dot(X.T)

    db1 = 1 / m \* np.sum(dZ1)

    return dW1, db1, dW2, db2

def update\_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha):

    W1 = W1 - alpha \* dW1

    b1 = b1 - alpha \* db1

    W2 = W2 - alpha \* dW2

    b2 = b2 - alpha \* db2

    return W1, b1, W2, b2

def get\_predictions(A2):

    return np.argmax(A2, 0)

def get\_accuracy(predictions, Y):

    print(predictions, Y)

    return np.sum(predictions == Y) / Y.size

def gradient\_descent(X, Y, alpha, iterations):

    W1, b1, W2, b2 = init\_params()

    for i in range(iterations):

        Z1, A1, Z2, A2 = forward\_prop(W1, b1, W2, b2, X)

        dW1, db1, dW2, db2 = backward\_prop(Z1, A1, Z2, A2, W1, W2, X, Y)

        W1, b1, W2, b2 = update\_params(W1, b1, W2, b2, dW1, db1, dW2, db2, alpha)

        if i % 10 == 0:

            print("Iteration: ", i)

            predictions = get\_predictions(A2)

            print(get\_accuracy(predictions, Y))

    return W1, b1, W2, b2

W1, b1, W2, b2 = gradient\_descent(X\_train, Y\_train, 0.10, 500)

def make\_predictions(X, W1, b1, W2, b2):

    \_, \_, \_, A2 = forward\_prop(W1, b1, W2, b2, X)

    predictions = get\_predictions(A2)

    return predictions

def test\_prediction(index, W1, b1, W2, b2):

    current\_image = X\_train[:, index, None]

    prediction = make\_predictions(X\_train[:, index, None], W1, b1, W2, b2)

    label = Y\_train[index]

    print("Prediction: ", prediction)

    print("Label: ", label)

    current\_image = current\_image.reshape((28, 28)) \* 255

    plt.gray()

    plt.imshow(current\_image, interpolation='nearest')

    plt.show()

test\_prediction(0, W1, b1, W2, b2)

test\_prediction(1, W1, b1, W2, b2)

test\_prediction(2, W1, b1, W2, b2)

test\_prediction(3, W1, b1, W2, b2)

dev\_predictions = make\_predictions(X\_dev, W1, b1, W2, b2)

get\_accuracy(dev\_predictions, Y\_dev)

**Output**

array([6, 4, 7, ..., 4, 1, 0], dtype=int64)

Output exceeds the [size limit](command:workbench.action.openSettings?%5B%22notebook.output.textLineLimit%22%5D). Open the full output data [in a text editor](command:workbench.action.openLargeOutput?c972c805-afa0-44ef-9dde-0342e7295bf9)

Iteration: 0

[6 2 6 ... 6 6 6] [6 4 7 ... 4 1 0]

0.09565853658536586

Iteration: 10

[4 8 9 ... 8 9 1] [6 4 7 ... 4 1 0]

0.1646829268292683

Iteration: 20

[4 8 9 ... 8 6 0] [6 4 7 ... 4 1 0]

0.2276829268292683

Iteration: 30

[4 8 9 ... 8 6 0] [6 4 7 ... 4 1 0]

0.2939512195121951

Iteration: 40

[4 4 9 ... 8 6 0] [6 4 7 ... 4 1 0]

0.3631951219512195

Iteration: 50

[4 4 3 ... 8 6 0] [6 4 7 ... 4 1 0]

0.4305121951219512

Iteration: 60

[6 4 3 ... 7 6 0] [6 4 7 ... 4 1 0]

0.48236585365853657

Iteration: 70

[6 4 3 ... 9 6 0] [6 4 7 ... 4 1 0]

0.5249512195121951

Iteration: 80

...

0.8472439024390244

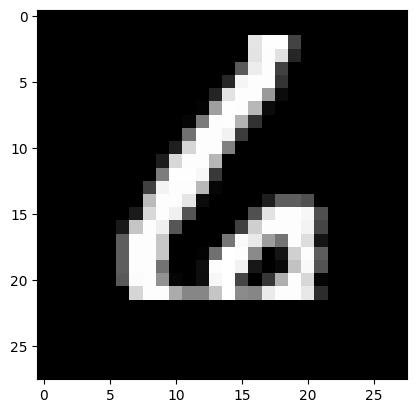
Iteration: 490

[6 4 2 ... 9 8 0] [6 4 7 ... 4 1 0]

0.8479756097560975

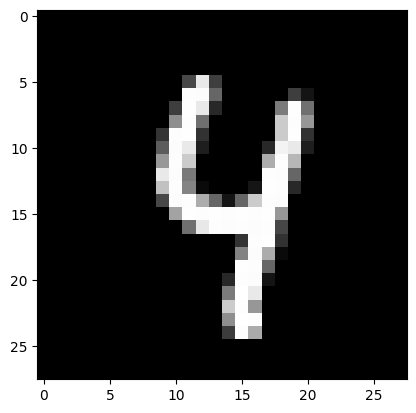
Prediction: [6]

Label: 6



Prediction: [4]

Label: 4



[8 2 8 6 3 9 5 3 5 6 1 4 6 0 9 0 7 5 1 1 9 3 3 3 1 2 1 4 2 7 4 8 9 4 6 7 1

7 9 7 5 1 1 3 9 1 1 9 1 5 2 4 8 4 3 6 8 6 6 9 8 9 7 6 9 1 8 1 6 4 4 6 5 3

6 5 1 3 2 5 9 1 9 7 6 3 4 3 4 3 5 6 2 2 2 6 9 6 5 5 8 1 5 7 0 9 6 5 0 2 2

3 5 5 0 8 6 7 2 2 1 9 8 9 9 2 1 2 8 4 8 6 0 8 8 0 5 7 7 4 9 3 5 1 9 6 2 1

9 9 5 4 1 1 2 0 2 7 3 1 6 5 7 3 6 4 0 2 4 1 3 1 5 9 0 6 9 9 8 7 8 6 7 3 1

2 2 2 4 5 8 4 3 1 0 2 9 1 0 4 4 2 0 8 4 8 3 5 0 9 2 2 0 1 9 7 2 6 2 0 7 1

8 6 8 1 9 8 9 4 9 5 1 1 4 5 4 9 2 1 3 0 8 9 2 6 8 3 8 1 4 0 2 4 0 8 3 7 8

3 6 2 8 3 1 8 7 9 4 7 7 4 8 9 3 0 7 4 0 5 4 3 7 2 6 1 2 0 5 0 6 9 0 6 6 1

1 8 7 3 4 1 3 4 6 5 2 0 3 6 4 7 1 6 4 9 5 7 8 8 6 1 2 2 4 4 9 2 3 6 6 0 6

9 1 8 2 2 8 9 9 5 7 8 2 9 9 7 6 0 7 1 6 6 0 6 6 7 6 2 6 3 8 2 6 0 3 8 0 1

2 0 3 0 4 1 0 7 1 2 4 6 3 4 8 3 7 2 8 4 8 6 2 1 0 3 2 9 6 0 7 6 6 0 4 9 3

6 1 8 4 5 6 0 9 4 0 7 6 7 5 9 4 7 6 9 2 2 5 5 6 2 6 4 7 5 2 6 3 3 4 3 3 4

5 2 8 6 2 7 7 1 2 7 7 1 7 3 3 0 1 1 0 8 7 4 7 8 9 5 1 5 2 6 9 8 7 7 3 1 8

4 6 0 3 4 2 5 2 7 6 5 1 6 2 5 8 7 4 1 3 1 7 3 9 4 0 8 9 5 8 1 8 8 3 3 1 9

3 8 2 7 4 8 8 5 6 3 2 4 8 3 7 0 0 5 6 7 1 0 8 2 8 5 8 8 2 4 3 1 7 5 2 6 1

6 7 8 5 9 6 3 6 2 7 2 1 0 4 1 3 6 0 6 1 0 7 7 6 9 0 8 2 0 4 3 6 7 3 1 4 0

9 6 3 7 1 1 3 5 8 2 6 8 8 4 2 8 7 2 8 0 9 9 4 7 7 6 5 1 1 8 2 3 6 1 3 8 6

4 1 7 8 1 2 2 7 3 4 2 3 5 6 9 4 6 0 1 7 1 1 2 6 4 7 1 7 7 8 8 9 8 3 0 4 8

8 3 4 4 4 1 7 2 6 2 1 9 7 4 7 7 4 5 2 4 1 7 1 5 0 2 8 2 4 5 8 9 4 0 0 1 8

7 7 9 7 1 7 7 2 9 4 4 7 2 0 9 1 6 6 2 0 2 2 4 8 1 8 2 4 0 5 5 3 6 8 9 0 6

1 2 6 1 8 5 9 0 7 6 4 0 6 2 4 3 1 5 0 9 7 7 3 6 4 8 5 2 8 7 4 8 3 9 9 1 6

1 2 1 5 7 7 7 8 3 9 4 7 0 5 4 9 6 4 2 1 0 3 9 7 9 9 4 4 9 6 4 6 4 5 4 6 2

3 5 9 6 8 4 9 0 9 6 1 3 8 0 0 7 4 1 5 5 2 1 0 7 5 0 8 5 0 6 7 1 0 3 8 6 7

0 5 7 7 5 5 5 4 0 2 8 6 1 9 8 4 3 9 4 6 3 0 5 2 2 0 6 9 1 8 4 3 8 4 8 1 7

2 3 7 3 1 8 1 6 5 3 4 0 9 8 7 3 6 6 8 9 3 1 5 3 1 6 7 9 2 6 1 0 0 3 9 1 8

...

2 3 7 3 1 3 1 6 5 3 4 0 9 8 7 5 6 6 2 9 5 1 5 3 1 3 7 9 2 6 1 0 0 3 7 1 8

3 6 0 6 0 9 9 5 9 1 4 8 7 8 1 4 7 0 3 7 4 2 1 1 6 4 7 4 5 1 6 7 0 5 1 5 1

6 1 1 0 9 8 8 5 6 6 2 7 7 5 1 0 2 0 8 7 0 4 4 6 1 8 1 7 3 1 7 2 1 6 0 9 7

1]

0.843

***4. 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.***

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB

X, y = load\_iris(return\_X\_y=True)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

X

y

model = GaussianNB()

y\_pred = model.fit(X\_train, y\_train).predict(X\_test)

y\_pred

y\_test

diff = y\_pred - y\_test

diff

print('Number of incorrect classifications are: ',len(y\_pred) - list(diff).count(0))

**Output**

array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

[5.4, 3.9, 1.7, 0.4],

[4.6, 3.4, 1.4, 0.3],

[5. , 3.4, 1.5, 0.2],

[4.4, 2.9, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.1],

[5.4, 3.7, 1.5, 0.2],

[4.8, 3.4, 1.6, 0.2],

[4.8, 3. , 1.4, 0.1],

[4.3, 3. , 1.1, 0.1],

[5.8, 4. , 1.2, 0.2],

[5.7, 4.4, 1.5, 0.4],

...

[6.7, 3. , 5.2, 2.3],

[6.3, 2.5, 5. , 1.9],

[6.5, 3. , 5.2, 2. ],

[6.2, 3.4, 5.4, 2.3],

[5.9, 3. , 5.1, 1.8]])

array([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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,

0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,

0])

array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,

0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,

0])

array([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])

Number of incorrect classifications are: 0

***5. Implement Find S Algorithm.***

import csv

def read\_csv(filename):

    with open(filename, 'r') as csvfile:

        datareader = csv.reader(csvfile, delimiter=',')

        traindata = []

        for row in datareader:

            traindata.append(row)

        print(traindata)

    return traindata

attributes = ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']

print("Attributes: ",attributes)

num\_attributes = len(attributes)

dataset = read\_csv("EnjoySport.csv")

dataset

len(dataset)

hypothesis = ['0']\*len(attributes)

hypothesis

print("Initial Hypothesis: ",hypothesis)

for j in range(0, num\_attributes):

    hypothesis[j] = dataset[0][j]

print("Find S: Finding a Maximally Specific Hypothesis")

for i in range(len(dataset)):

    if dataset[i][num\_attributes] == "Yes":

        for j in range(num\_attributes):

            if dataset[i][j] != hypothesis[j]:

                hypothesis[j] = "?"

    else:

        continue

i

print("Hypothesis")

print(hypothesis)

**Output**

Attributes: ['Sky', 'Temp', 'Humidity', 'Wind', 'Water', 'Forecast']

[['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']]

Initial Hypothesis: ['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

Hypothesis

['sunny', 'warm', '?', 'strong', '?', '?']

***6. Apply K-Means Clustering algorithm to cluster data stored in a .CSV file.***

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

# import some data to play with

iris = datasets.load\_iris()

# print("\n IRIS DATA :",iris.data);

# print("\n IRIS FEATURES :\n",iris.feature\_names)

# print("\n IRIS TARGET :\n",iris.target)

# print("\n IRIS TARGET NAMES:\n",iris.target\_names)

# Store the inputs as a Pandas Dataframe and set the column names

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Set the size of the plot

plt.figure(figsize=(14,7))

# Create a colormap

colormap = np.array(['red', 'lime', 'black'])

# Plot Sepal

plt.subplot(1, 2, 1)

plt.scatter(X.Sepal\_Length,X.Sepal\_Width, c=colormap[y.Targets], s=40)

plt.title('Sepal')

plt.subplot(1, 2, 2)

plt.scatter(X.Petal\_Length,X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Petal')

# K Means Cluster

model = KMeans(n\_clusters=3, n\_init='auto')

model.fit(X)

model.labels\_

# View the results

# Set the size of the plot

plt.figure(figsize=(14,7))

# Create a colormap

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications

plt.subplot(1, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Classification')

# Plot the Models Classifications

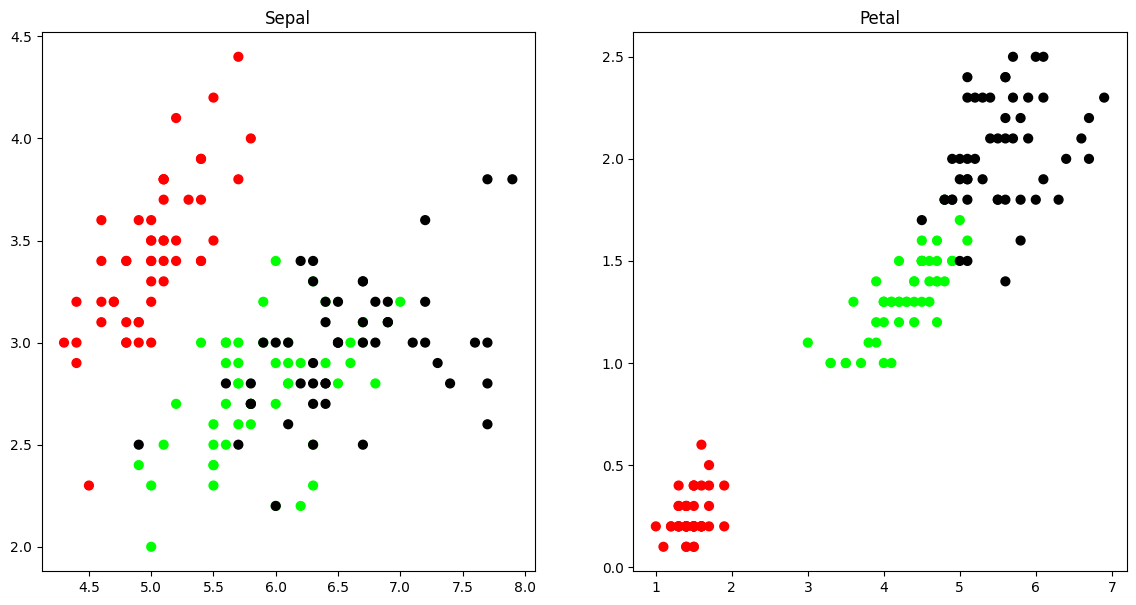
plt.subplot(1, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K Mean Classification')

**Output**

Before K Means Clustering



After K Means Clustering

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

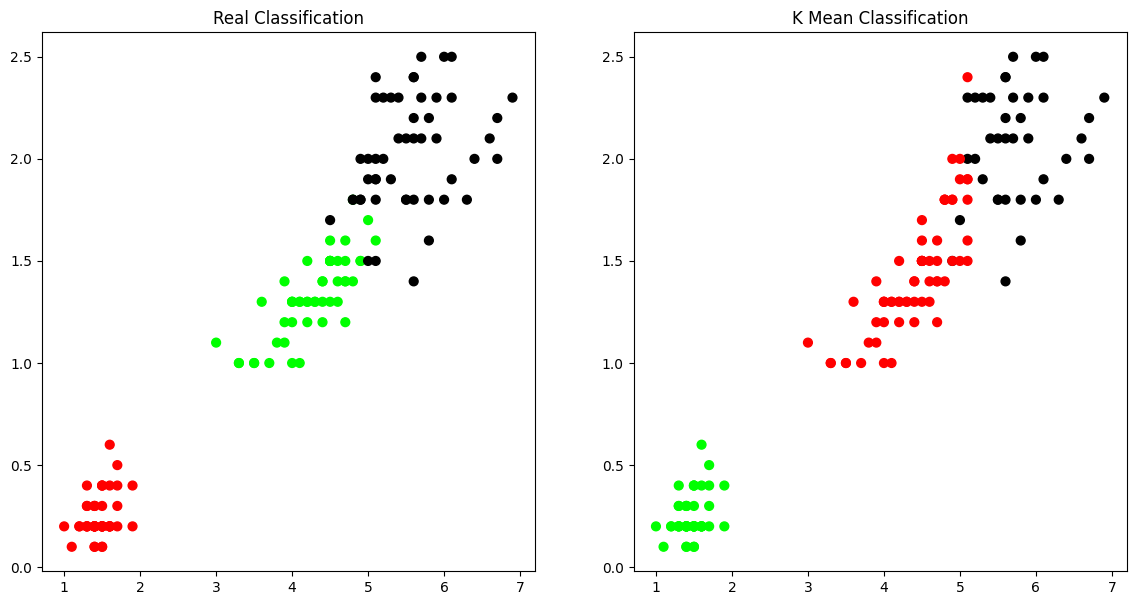
1, 1, 1, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,

2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2,

2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])



***7. Implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.***

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

import numpy as np

df = pd.read\_csv('iris.csv')

df.head()

df['variety'].unique()

x = df[['sepal.length','sepal.width','petal.length','petal.width']]

y = df['variety'].replace({'Setosa':1, 'Versicolor':2, 'Virginica':3 })

train\_x, test\_x , train\_y, test\_y= train\_test\_split(x,y,test\_size=0.1,random\_state=3, shuffle=True)

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(train\_x,train\_y)

pred\_y = knn.predict(test\_x)

print('The prediction is',pred\_y)

print('Actual expected results are:', list(test\_y))

print('Number of incorrect classifications are')

diff = np.array(pred\_y) - np.array(test\_y)

print(len(diff) - list(diff).count(0))

**Output**

array(['Setosa', 'Versicolor', 'Virginica'], dtype=object)

The prediction is [1 1 1 1 1 3 2 1 3 2 2 1 2 2 3]

Actual expected results are: [1, 1, 1, 1, 1, 3, 2, 1, 3, 2, 2, 1, 2, 2, 3]

Number of incorrect classifications are

0

***8. Implement the parametric Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs***

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn import linear\_model

df = pd.read\_csv('iris.csv')

newdf = df[['petal.length','petal.width']]

train\_x, test\_x , train\_y, test\_y= train\_test\_split(newdf['petal.length'],newdf['petal.width'],test\_size=0.1,random\_state=3, shuffle=True)

train\_x = np.array(train\_x).reshape(-1,1)

test\_x = np.array(test\_x).reshape(-1,1)

train\_y = np.array(train\_y).reshape(-1,1)

test\_y = np.array(test\_y).reshape(-1,1)

lr = linear\_model.LinearRegression()

lr.fit(train\_x,train\_y)

import matplotlib.pyplot as plt

plt.scatter(test\_x,test\_y)

y\_predict = lr.predict(test\_x)

plt.scatter(test\_x,y\_predict,color = 'red')

plt.xlabel("petal length")

plt.ylabel("petal width")

from sklearn.metrics import mean\_squared\_error

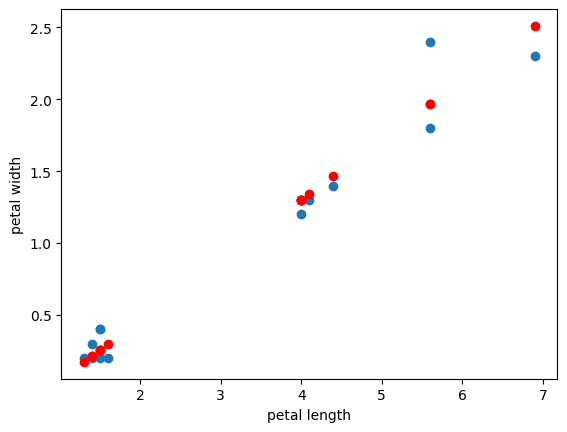
sd = (10/4)\*\*0.5

se = sd/(5\*\*0.5)

print('Standard Error ',se )

print('Mean Square Error ', mean\_squared\_error(test\_y,y\_predict))

**Output**



Standard Error 0.7071067811865476

Mean Square Error 0.022523547640619812