**Task1:-**

1. 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.

# Mounting google colab with google drive

from google.colab import drive

drive.mount('/content/gdrive')

# importing the dataset from google drive

import pandas as pd

sports = pd.read\_csv("/content/drive/MyDrive/MLLab/enjoysport.csv", sep="\t",header=None)

print(sports)

sports.count()

a=[]

#To read the contents of a csv file

with open("/content/drive/MyDrive/MLLab/enjoysport.csv") as csvfile:

    for row in csv.reader(csvfile):

        a.append(row)

print(a)

# To display the total number of records in a file

print("\n The total number of training instances are : ",len(a))

# the number of attributes (Except the target variable)

num\_attribute = len(a[0])-1

# To display the initial hypothesis

print("\n The initial hypothesis is : ")

hypothesis = ['pi']\*num\_attribute

print(hypothesis)

# To display the specific hypothesis at each and every iteration

for i in range(0, len(a)):

    if a[i][num\_attribute] == 'yes':

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

            if hypothesis[j] == 'pi' or hypothesis[j] == a[i][j]:

                hypothesis[j] = a[i][j]

            else:

                hypothesis[j] = '?'

    print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)

print("\n The Maximally specific hypothesis for the training instance is ")

print(hypothesis)

**Output:-**

The hypothesis for the training instance 1 is:

['pi', 'pi', 'pi', 'pi', 'pi', 'pi']

The hypothesis for the training instance 2 is:

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

The hypothesis for the training instance 3 is:

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

The hypothesis for the training instance 4 is:

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

The hypothesis for the training instance 5 is:

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

The Maximally specific hypothesis for the training instance is

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

Task2:

**Task2:-** 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.

**Aim: -** To find specific and generalized hypothesis items using Candidate Elimination Algorithm

Program:-

# Mounting google colab with google drive

from google.colab import drive

drive.mount('/content/gdrive')

# send the data set into a data frame

import pandas as pd

import numpy as np

import csv

data = pd.DataFrame(data=pd.read\_csv('/content/gdrive/MyDrive/MLLab/enjoysport.csv'))

# storing and displaying the all the attributes into an array called as concept expect the target variable

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

print(concepts)

# storing the target variableinto an array called as target expect the target variable

target = np.array(data.iloc[:,-1])

print(target)

|  |
| --- |
| Output for the above sample code  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]  ['yes' 'yes' 'no' 'yes'] |

def learn(concepts, target):

    specific\_h = concepts[0].copy()

    print("initialization 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] ='?'

                print(specific\_h)

        print(specific\_h)

        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(" steps 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("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

**Output:-**

initialization of specific\_h and general\_h

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

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

**Final Specific:**

[‘Sunny' 'warm' '?' 'strong' '?' '?']

**Final General\_h:**

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

**Task3:-**

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

from google.colab import drive

drive.mount('/content/gdrive')

import math

import csv

import pandas as pd

import numpy as np

# load dataset

data = pd.read\_csv("/content/gdrive/MyDrive/MLLab/ID3dataset.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

    def \_\_init\_\_(self):

        self.children = []

        self.value = ""

        self.isLeaf = False

        self.pred = ""

def entropy(examples):

    pos = 0.0

    neg = 0.0

    for \_, row in examples.iterrows():

        if row["answer"] == "yes":

            pos += 1

        else:

            neg += 1

    if pos == 0.0 or neg == 0.0:

        return 0.0

    else:

        p = pos / (pos + neg)

        n = neg / (pos + neg)

        return -(p \* math.log(p, 2) + n \* math.log(n, 2))

def info\_gain(examples, attr):

    uniq = np.unique(examples[attr])

    #print ("\n",uniq)

    gain = entropy(examples)

    #print ("\n",gain)

    for u in uniq:

        subdata = examples[examples[attr] == u]

        #print ("\n",subdata)

        sub\_e = entropy(subdata)

        gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

        #print ("\n",gain)

    return gain

def ID3(examples, attrs):

    root = Node()

    max\_gain = 0

    max\_feat = ""

    for feature in attrs:

        #print ("\n",examples)

        gain = info\_gain(examples, feature)

        if gain > max\_gain:

            max\_gain = gain

            max\_feat = feature

    root.value = max\_feat

    #print ("\nMax feature attr",max\_feat)

    uniq = np.unique(examples[max\_feat])

    #print ("\n",uniq)

    for u in uniq:

        #print ("\n",u)

        subdata = examples[examples[max\_feat] == u]

        #print ("\n",subdata)

        if entropy(subdata) == 0.0:

            newNode = Node()

            newNode.isLeaf = True

            newNode.value = u

            newNode.pred = np.unique(subdata["answer"])

            root.children.append(newNode)

        else:

            dummyNode = Node()

            dummyNode.value = u

            new\_attrs = attrs.copy()

            new\_attrs.remove(max\_feat)

            child = ID3(subdata, new\_attrs)

            dummyNode.children.append(child)

            root.children.append(dummyNode)

    return root

def printTree(root: Node, depth=0):

    for i in range(depth):

        print("\t", end="")

    print(root.value, end="")

    if root.isLeaf:

        print(" -> ", root.pred)

    print()

    for child in root.children:

        printTree(child, depth + 1)

root = ID3(data, features)

printTree(root)

**Output:-**

outlook

overcast -> ['yes']

rain

wind

strong -> ['no']

weak -> ['yes']

sunny

humidity

high -> ['no']

normal -> ['yes']

**Task4:**

Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

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

#Sigmoid Function

def sigmoid (x):

    return 1/(1+np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

    return x\*(1-x)

#Variable Initialization

epoch=10000 #Setting training iterations

lr=0.1  #Setting Learning Rate

inputlayer\_neurons=2   #no.of features in dataset

hidden\_layers\_neurons=3  # no.of neurons in hidden layer

output\_neurons=1         #no.of neurons in output layer

#Weight and Bias Initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hidden\_layers\_neurons))

bh=np.random.uniform(size=(1,hidden\_layers\_neurons))

wout=np.random.uniform(size=(hidden\_layers\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dimensions x\*y

for i in range(epoch):

#Forward Propagation

    hinp1=np.dot(X,wh)

    hinp=hinp1+bh

    hlayer\_act=sigmoid(hinp)

    outinp1=np.dot(hlayer\_act,wout)

    outinp=outinp1+bout

    output=sigmoid(outinp)

#Backward Propagation

EO=y-output  #finding the error output

outgrad=derivatives\_sigmoid(output)

d\_output=EO\*outgrad

EH=d\_output.dot(wout.T)

#How much hidden layer wts contributed to error

hiddengrad=derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer=EH\*hiddengrad

#dot product of next layer error and current layerop

wout+=hlayer\_act.T.dot(d\_output)\*lr

wh+=X.T.dot(d\_hiddenlayer)\*lr

print("Input: \n"+str(X))

print("Actual Output: \n" +str(y))

print("Predicted Output: \n",output)

**Output:-**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.77104055]

[0.76699202]

[0.77066072]]

Task6:-

Assuming a set of documents that need to be classified, use the naÃ¯ve 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

"""

import pandas as pd

msg=pd.read\_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.message

y=msg.labelnum

print(X)

print(y)

#splitting the dataset into train and test data

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

xtrain,xtest,ytrain,ytest=train\_test\_split(X,y)

print ('\n the total number of Training Data :',ytrain.shape)

print ('\n the total number of Test Data :',ytest.shape)

#output of the words or Tokens in the text documents

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

xtrain\_dtm = count\_vect.fit\_transform(xtrain)

xtest\_dtm=count\_vect.transform(xtest)

print('\n The words or Tokens in the text documents \n')

print(count\_vect.get\_feature\_names())

df=pd.DataFrame(xtrain\_dtm.toarray(),columns=count\_vect.get\_feature\_names())

# Training Naive Bayes (NB) classifier on training data.

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain\_dtm,ytrain)

predicted = clf.predict(xtest\_dtm)

#printing accuracy, Confusion matrix, Precision and Recall

from sklearn import metrics

print('\n Accuracy of the classifier is',metrics.accuracy\_score(ytest,predicted))

print('\n Confusion matrix')

print(metrics.confusion\_matrix(ytest,predicted))

print('\n The value of Precision', metrics.precision\_score(ytest,predicted))

print('\n The value of Recall', metrics.recall\_score(ytest,predicted))

Taks7:-

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. You can use Java/Python ML library classes/API

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDisease = pd.read\_csv('heart.csv')

heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')

print(heartDisease.head())

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

model= BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')

q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')

q2=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)

**Task8:**

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

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)

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

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

model = KMeans(n\_clusters=3)

model.fit(X)

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

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')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# 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')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('The accuracy score of K-Mean: ',sm.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean: ',sm.confusion\_matrix(y, model.labels\_))

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)

#xs.sample(5)

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_gmm = gmm.predict(xs)

#y\_cluster\_gmm

plt.subplot(2, 2, 3)

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

plt.title('GMM Classification')

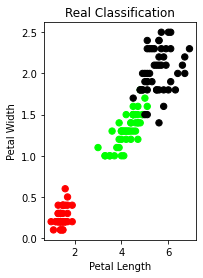
plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy\_score(y, y\_gmm))

print('The Confusion matrix of EM: ',sm.confusion\_matrix(y, y\_gmm))

**Output:** KMeans(algorithm='auto', copy\_x=True, init='k-means++', max\_iter=300,n\_clusters=3, n\_init=10, n\_jobs=None, precompute\_distances='auto',random\_state=None, tol=0.0001, verbose=0)

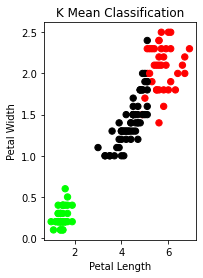


The accuracy score of K-Mean: 0.09333333333333334

The Confusion matrixof K-Mean: [[ 0 50 0]

[ 2 0 48]

[36 0 14]]

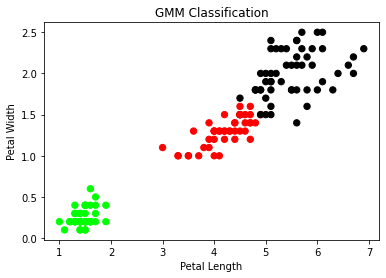


The accuracy score of EM: 0.3333333333333333

The Confusion matrix of EM: [[ 0 50 0]

[45 0 5]

[ 0 0 50]]



Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

iris=datasets.load\_iris()

x = iris.data

y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

#To Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train, y\_train)

#To make predictions on our test data

y\_pred=classifier.predict(x\_test)

print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

print(classification\_report(y\_test,y\_pred))

**Output:-**

Confusion Matrix

[[19 0 0]

[ 0 11 0]

[ 0 3 12]]

Accuracy Metrics

precision recall f1-score support

0 1.00 1.00 1.00 19

1 0.79 1.00 0.88 11

2 1.00 0.80 0.89 15

accuracy 0.93 45

macro avg 0.93 0.93 0.92 45

weighted avg 0.95 0.93 0.93 45

Task10:

from numpy import \*

from os import listdir

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

import numpy.linalg as np

from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):

m,n = np1.shape(xmat)

weights = np1.mat(np1.eye((m)))

for j in range(m):

diff = point - X[j]

weights[j,j] = np1.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 = np1.shape(xmat)

ypred = np1.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('tips.csv')

bill = np1.array(data.total\_bill)

tip = np1.array(data.tip)

#preparing and add 1 in bill

mbill = np1.mat(bill)

mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form

m= np1.shape(mbill)[1]

# print(m) 244 data is stored in m

one = np1.mat(np1.ones(m))

X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE

#print(X)

#set k here

ypred = localWeightRegression(X,mtip,0.3)

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();

Task10a

import numpy as np

from bokeh.plotting import figure, show, output\_notebook

from bokeh.layouts import gridplot

from bokeh.io import push\_notebook

def local\_regression(x0, X, Y, tau):# add bias term

x0 = np.r\_[1, x0] # Add one to avoid the loss in information

X = np.c\_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

# predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial\_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau))

# Weight or Radial Kernal Bias Function

n = 1000

# generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set ( 10 Samples) X :\n",X[1:10])

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])

# jitter X

X += np.random.normal(scale=.1, size=n)

print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)

print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot\_lwr(tau):

# prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plot = figure(plot\_width=400, plot\_height=400)

plot.title.text='tau=%g' % tau

plot.scatter(X, Y, alpha=.3)

plot.line(domain, prediction, line\_width=2, color='red')

return plot

show(gridplot([

[plot\_lwr(10.), plot\_lwr(1.)],

[plot\_lwr(0.1), plot\_lwr(0.01)]]))

**Additional Task1:-**

Implement the random forest algorithm on salaries dataset to predict the salary based on position level of an employee and depict along with a plot diagram.

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

data = pd.read\_csv('Salaries.csv')

print(data)

x = data.iloc[:, 1:2].values

print(x)

y = data.iloc[:, 2].values

# Fitting Random Forest Regression to the dataset

# import the regressor

from sklearn.ensemble import RandomForestRegressor

# create regressor object

regressor = RandomForestRegressor(n\_estimators = 100, random\_state = 0)

# fit the regressor with x and y data

regressor.fit(x, y)

# create regressor object

regressor = RandomForestRegressor(n\_estimators = 100, random\_state = 0)

  # fit the regressor with x and y data

regressor.fit(x, y)

#Predicting a new result

Y\_pred = regressor.predict(np.array([6.5]).reshape(1, 1))  # test the output by changing values

# Visualising the Random Forest Regression results

# arange for creating a range of values

# from min value of x to max

# value of x with a difference of 0.01

# between two consecutive values

X\_grid = np.arange(min(x), max(x), 0.01)

# reshape for reshaping the data into a len(X\_grid)\*1 array,

# i.e. to make a column out of the X\_grid value

X\_grid = X\_grid.reshape((len(X\_grid), 1))

# Scatter plot for original data

plt.scatter(x, y, color = 'blue')

# plot predicted data

plt.plot(X\_grid, regressor.predict(X\_grid),

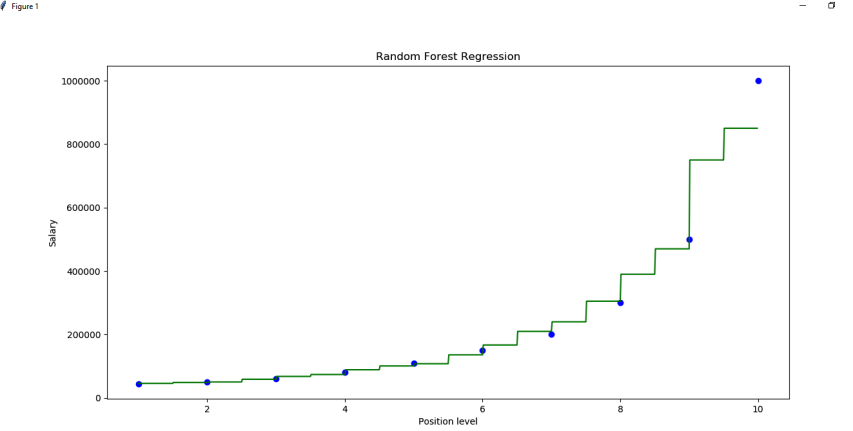
color = 'green')

plt.title('Random Forest Regression')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()



**Additional Task2:-**

Implement the SVM algorithm using machine learning and find the best fit predicted new values on cancer dataset.

# importing scikit learn with make\_blobs

from sklearn.datasets.samples\_generator import make\_blobs

# creating datasets X containing n\_samples

# Y containing two classes

X, Y = make\_blobs(n\_samples=500, centers=2,

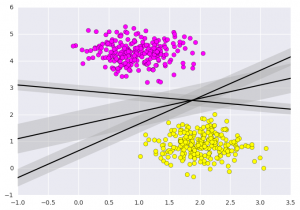
random\_state=0, cluster\_std=0.40)

import matplotlib.pyplot as plt

# plotting scatters

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');

plt.show()



# importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# reading csv file and extracting class column to y.

x = pd.read\_csv("C:\...\cancer.csv")

a = np.array(x)

y  = a[:,30] # classes having 0 and 1

# extracting two features

x = np.column\_stack((x.malignant,x.benign))

# 569 samples and 2 features

x.shape

print (x),(y)

# import support vector classifier

# "Support Vector Classifier"

from sklearn.svm import SVC

clf = SVC(kernel='linear')

# fitting x samples and y classes

clf.fit(x, y)

clf.predict([[120, 990]])

clf.predict([[85, 550]])

array([ 0.])

array([ 1.])

