**KNS INSTITUTE OF TECHNOLOGY**

**Dept. Of CSE/ISE**

**AI & ML LAB MANUAL (18CSL76)**

****

**MACHINE LEARNING LABORATORY**

[As per Choice Based Credit System (CBCS) scheme]

**Course objectives:**

This course will enable students to

1. Make use of Data sets in implementing the machine learning algorithms

2. Implement the machine learning concepts and algorithms in any suitable language of choice

**Description (If any):**

1. The programs can be implemented in either JAVA or Python.

2. For Problems 1 to 6 and 10, programs are to be developed without using the built-in classes or APIs of Java/Python.

3. Data sets can be taken from standard repositories (https://archive.ics.uci.edu/ml/datasets.html) or constructed by the students.

**Course outcomes:**

The students should be able to:

|  |
| --- |
| **Statement** |
| Understand the implementation procedures for the machine learning algorithms. |
| Design Java/Python programs for various Learning algorithms. |
| Apply appropriate data sets to the Machine Learning algorithms. |
| Identify and apply Machine Learning algorithms to solve real world problems. |

**Conduction of Practical Examination:**

All laboratory experiments are to be included for practical examination.

Students are allowed to pick one experiment from the lot.

Strictly follow the instructions as printed on the cover page of answer script

Marks distribution: Procedure + Conduction + Viva: 20 + 50 +10 (80)

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
|  |  |
| ***Lab Syllabus Programs*** |  |
| 1. *Implement A\* search Algorithm* |  |
| 1. *Implement AO\* search Algorithm* |  |
| 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* |  |
| 1. *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.* |  |
| 1. *Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets* |  |
| 1. *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* |  |
| 1. *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 algorithmsand comment on the quality of clustering. You can add JAVA/Python Libarary classes for ML program* |  |
| 1. *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.* |  |
| 1. *Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.* |  |

**PROGRAM 1**

*Implement A\* search Algorithm*

class Graph:

def \_\_init\_\_(self, adjac\_lis):

self.adjac\_lis = adjac\_lis

def get\_neighbors(self, v):

return self.adjac\_lis[v]

# This is heuristic function which is having equal values for all nodes

def h(self, n):

H = {

'A': 1,

'B': 1,

'C': 1,

'D': 1

}

return H[n]

def a\_star\_algorithm(self, start, stop):

# In this open\_lst is a lisy of nodes which have been visited, but who's

# neighbours haven't all been always inspected, It starts off with the start

#node

# And closed\_lst is a list of nodes which have been visited

# and who's neighbors have been always inspected

open\_lst = set([start])

closed\_lst = set([])

# poo has present distances from start to all other nodes

# the default value is +infinity

poo = {}

poo[start] = 0

# par contains an adjac mapping of all nodes

par = {}

par[start] = start

while len(open\_lst) > 0:

n = None

# it will find a node with the lowest value of f() -

for v in open\_lst:

if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):

n = v;

if n == None:

print('Path does not exist!')

return None

# if the current node is the stop

# then we start again from start

if n == stop:

reconst\_path = []

while par[n] != n:

reconst\_path.append(n)

n = par[n]

reconst\_path.append(start)

reconst\_path.reverse()

print('Path found: {}'.format(reconst\_path))

return reconst\_path

# for all the neighbors of the current node do

for (m, weight) in self.get\_neighbors(n):

# if the current node is not presentin both open\_lst and closed\_lst

# add it to open\_lst and note n as it's par

if m not in open\_lst and m not in closed\_lst:

open\_lst.add(m)

par[m] = n

poo[m] = poo[n] + weight

# otherwise, check if it's quicker to first visit n, then m

# and if it is, update par data and poo data

# and if the node was in the closed\_lst, move it to open\_lst

else:

if poo[m] > poo[n] + weight:

poo[m] = poo[n] + weight

par[m] = n

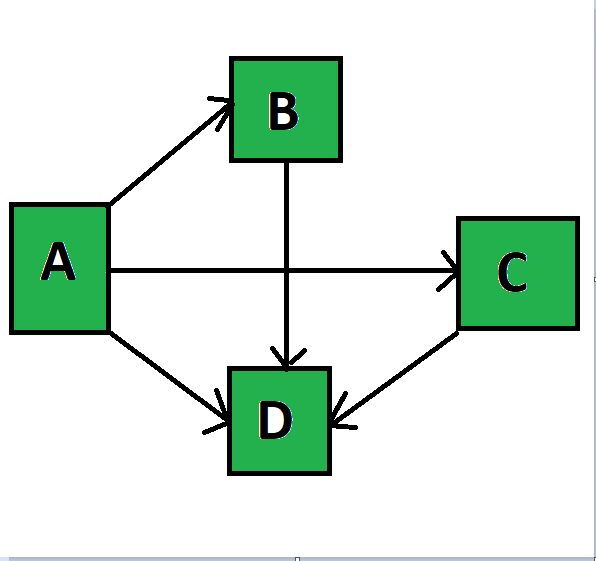
if m in closed\_lst:

closed\_lst.remove(m)

open\_lst.add(m)

# remove n from the open\_lst, and add it to closed\_lst

# because all of his neighbors were inspected

 open\_lst.remove(n)

closed\_lst.add(n)

print('Path does not exist!')

return None

adjac\_lis = {

'A': [('B', 1), ('C', 3), ('D', 7)],

'B': [('D', 5)],

'C': [('D', 12)]

}

graph1 = Graph(adjac\_lis)

graph1.a\_star\_algorithm('A', 'D')

**PROGRAM 3**

**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 \n")

with open('ws.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]\n")

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

# Comparing with First Training Example

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

S[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

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

temp=[]

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

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

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(" 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 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(" 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 Training Example No :4 the hypothesis is G4', [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']])

**PROGRAM 4**

**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.**

import sys

import numpy as np

from numpy import \*

import csv

class Node:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

def read\_data(filename):

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

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

metadata = next(datareader)

traindata=[]

for row in datareader:

traindata.append(row)

return (metadata, traindata)

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col]) # get unique values in particular column

count = np.zeros((items.shape[0], 1), dtype=np.int32) #number of row = number of values

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

#count has the data of number of times each value is present in

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

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)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

#item is the unique value and dict is the data corresponding to it

total\_size = data.shape[0]

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

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

total\_entropy = entropy(data[:, -1])

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

total\_entropy -= entropies[x]

return total\_entropy

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])

return node

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

#size of gains= number of attribute to calculate gain

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

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.item(0).decode("utf-8"))

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value.tobytes().decode("utf-8"))

print\_tree(n, level + 2)

metadata, traindata = read\_data("tennis.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

**tennis.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| outlook | temp | humidity | windy | play |
| sunny | hot | high | Weak | no |
| sunny | hot | high | Strong | no |
| overcast | hot | high | Weak | yes |
| rainy | mild | high | Weak | yes |
| rainy | cool | normal | Weak | yes |
| rainy | cool | normal | Strong | no |
| overcast | cool | normal | Strong | yes |
| sunny | mild | high | Weak | no |
| sunny | cool | normal | Weak | yes |
| rainy | mild | normal | Weak | yes |
| sunny | mild | normal | Strong | yes |
| overcast | mild | high | Strong | yes |
| overcast | hot | normal | Weak | yes |
| rainy | mild | high | Strong | no |

**PROGRAM 5**

**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)

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 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

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))

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

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+ bout

output = sigmoid(outinp)

#Backpropagation

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

bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

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

bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output:\n" ,output)

**Output:**

Input:

[[2. 9.]

[1. 5.]

[3. 6.]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

('Predicted Output:\n', array([[0.89678563],

[0.87179214],

[0.89995692]]))

**PROGRAM 6**

**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**

import numpy as np

import math

import csv

def read\_data(filename):

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

datareader = csv.reader(csvfile)

metadata = next(datareader)

traindata=[]

for row in datareader:

traindata.append(row)

return (metadata, traindata)

def splitDataset(dataset, splitRatio): #splits dataset to training set and test set based on split ratio

trainSize = int(len(dataset) \* splitRatio)

trainSet = []

testset = list(dataset)

i=0

while len(trainSet) < trainSize:

trainSet.append(testset.pop(i))

return [trainSet, testset]

def classify(data,test):

total\_size = data.shape[0]

print("training data size=",total\_size)

print("test data sixe=",test.shape[0])

target=np.unique(data[:,-1])

count = np.zeros((target.shape[0]), dtype=np.int32)

prob = np.zeros((target.shape[0]), dtype=np.float32)

print("target count probability")

for y in range(target.shape[0]):

for x in range(data.shape[0]):

if data[x,data.shape[1]-1] == target[y]:

count[y] += 1

prob[y]=count[y]/total\_size # comptes the probability of target

print(target[y],"\t",count[y],"\t",prob[y])

prob0 = np.zeros((test.shape[1]-1), dtype=np.float32)

prob1 = np.zeros((test.shape[1]-1), dtype=np.float32)

accuracy=0

print("Instance prediction taget")

for t in range(test.shape[0]):

for k in range(test.shape[1]-1): # for each attribute in column

count1=count0=0

for j in range(data.shape[0]):

if test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[0]:

count0+=1

elif test[t,k]== data[j,k] and data[j,data.shape[1]-1]== target[1]:

count1+=1

prob0[k]= count0/count[0] #Find no probability of each attribute

prob1[k]= count1/count[1] #Find yes probability of each attribute

probno=prob[0]

probyes=prob[1]

for i in range(test.shape[1]-1):

probno=probno\*prob0[i]

probyes=probyes\*prob1[i]

if probno>probyes: # prediction

predict='no'

else:

predict='yes'

print(t+1,"\t",predict,"\t ",test[t,test.shape[1]-1])

if predict== test[t,test.shape[1]-1]: # computing accuracy

accuracy+=1

final\_accuracy=(accuracy/test.shape[0])\*100

print("accuracy",final\_accuracy,"%")

return

metadata, traindata = read\_data("tennis.csv")

splitRatio = 0.6

trainingset, testset = splitDataset(traindata, splitRatio)

training=np.array(trainingset)

testing=np.array(testset)

print("------------------Training Data-------------------")

print(trainingset)

print("-------------------Test Data-------------------")

print(testset)

classify(training,testing)

**PROGRAM 7**

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.

from sklearn.cluster import KMeans

#from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.mixture import GaussianMixture

data=pd.read\_csv("clusterdata.csv")

df1=pd.DataFrame(data)

print(df1)

f1 = df1['Distance\_Feature'].values

f2 = df1['Speeding\_Feature'].values

X=np.matrix(list(zip(f1,f2)))

plt.plot(1)

plt.subplot(511)

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title('Dataset')

plt.ylabel('speeding\_feature')

plt.xlabel('distance\_feature')

plt.scatter(f1,f2)

colors = ['b', 'g', 'r']

markers = ['o', 'v', 's']

# create new plot and data for K- means algorithm

plt.plot(2)

ax=plt.subplot(513)

kmeans\_model = KMeans(n\_clusters=3).fit(X)

for i, l in enumerate(kmeans\_model.labels\_):

fig1=plt.plot(f1[i], f2[i], color=colors[l],marker=markers[l])

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title('K- Means')

plt.ylabel('speeding\_feature')

plt.xlabel('distance\_feature')

# create new plot and data for gaussian mixture

plt.plot(3)

plt.subplot(515)

gmm=GaussianMixture(n\_components=3).fit(X)

labels= gmm.predict(X)

for i, l in enumerate(labels):

plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l])

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title('Gaussian Mixture')

plt.ylabel('speeding\_feature')

plt.xlabel('distance\_feature')

plt.show()

Driver\_ID Distance\_Feature Speeding\_Feature

0 3423311935 71.24 28

1 3423313212 52.53 25

2 3423313724 64.54 27

3 3423311373 55.69 22

4 3423310999 54.58 25

5 3423313857 41.91 10

6 3423312432 58.64 20

7 3423311434 52.02 8

8 3423311328 31.25 34

9 3423312488 44.31 19

10 3423311254 49.35 40

11 3423312943 58.07 45

12 3423312536 44.22 22

13 3423311542 55.73 19

14 3423312176 46.63 43

15 3423314176 52.97 32

16 3423314202 46.25 35

17 3423311346 51.55 27

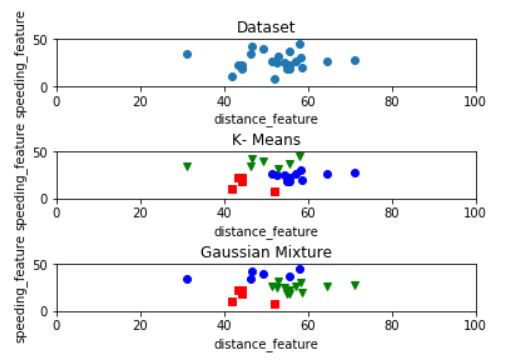
18 3423310666 57.05 26

19 3423313527 58.45 30

20 3423312182 43.42 23

21 3423313590 55.68 37

22 3423312268 55.15 18

****

**PROGRAM 8**

**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()

iris\_data=iris.data

iris\_labels=iris.target

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.30)

classifier=KNeighborsClassifier(n\_neighbors=5)

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

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

print('Confusion matrix is as follows')

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

print('Accuracy Matrics')

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

Output:

Confusion matrix is as follows

[[15 0 0]

[ 0 15 1]

[ 0 0 14]]

Accuracy Matrics

precision recall f1-score support

0 1.00 1.00 1.00 15

1 1.00 0.94 0.97 16

2 0.93 1.00 0.97 14

avg / total 0.98 0.98 0.98 45

**Program 09**

**Implement the non-parametric Locally Weighted Regression algorithm in order**

**to fit data points. Select appropriate data set for your experiment and draw**

**graphs.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# load data points

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

ColA = np.array(data.total\_bill)

ColB = np.array(data.tip)

#preparing and add 1 in bill

mColA = np.mat(ColA) # mat treats array as matrix

mColB = np.mat(ColB)

m= np.shape(mColB)[1]

#print(m)

one = np.ones((1,m) ,dtype=int)

#print(one)

#Horizontal Stacking

X= np.hstack((one.T,mColA.T))

#print(X.shape)

tem=np.hstack((mColA.T))

#print(tem)

#Gaussiaon Kernel

def kernel(point,xmat, k):

m,n = np.shape(xmat)

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

#print(weights)

for j in range(m):

diff = point - X[j]

weights[j,j] = np.exp(diff\*diff.T/(-2.0\*k\*\*2))

return weights

#Define Local Weights

def localWeight(point,xmat,ymat,k):

wt = kernel(point,xmat,k)

W = (X.T\*(wt\*X)).I\*(X.T\*(wt\*ymat.T))

return W

#Final prediction value

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

#prediction value

ypred = localWeightRegression(X,mColB,2)

#Plot Regression Lines

Xvalues=X.copy()

print(Xvalues)

Xvalues.sort(axis=0)

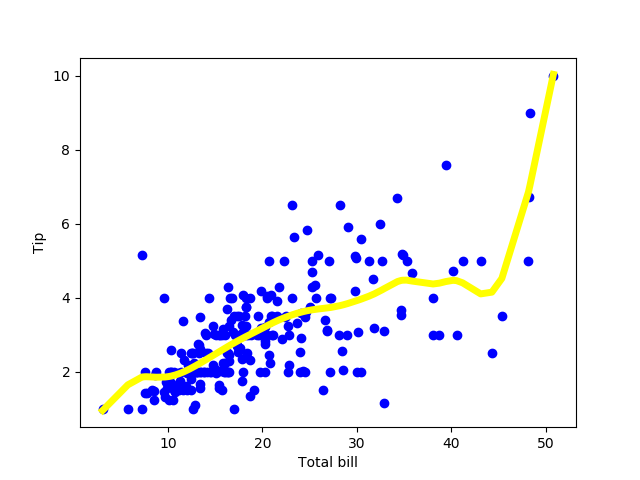
plt.scatter(ColA, ColB, color='blue')

plt.plot(Xvalues[:,1], ypred[X[:,1].argsort(0)], color='yellow', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();



**Program 02**

**IMPLEMENT AO\* SEARCH ALGORITHM**