VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

**Machine Learning Lab**

Submitted by

**Supriya M Lingdal (1BM20CS415)**

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

April-2022 to July-202

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled **Machine Learning Lab** carried out by **Supriya M Lingdal (1BM20CS415),** who is bonafide student of B. M. S. College of Engineering. It is in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic year 2021-2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning Lab** work prescribed for the said degree.

Dr. Jyothi S Nayak

Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Program1**

**Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Code:**

import pandas as pd

import numpy as np

#to read the data in the csv file

data = pd.read\_csv("Desktop/data.csv")

print(data,"\n")

#making an array of all the attributes

d = np.array(data)[:,:-1]

print("\n The attributes are: ",d)

#segragating the target that has positive and negative examples target = np.array(data)[:,-1]

print("\n The target is: ",target)

#training function to implement find-s algorithm

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:

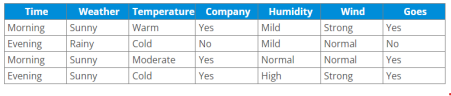
pass

return specific\_hypothesis

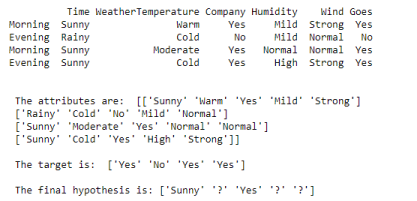
#obtaining the final hypothesis

print("\n The final hypothesis is:",train(d,target))

**DataSet:**

****

**Output:**

****

**Program 2**

**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.**

**Code:**

import numpy as np

import pandas as pd

data = pd.read\_csv('Desktop/data2.csv')

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

print("\nInstances are:\n",concepts)

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

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and

genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

general\_h = [["?" for i in range(len(specific\_h))]

for i in range(len(specific\_h))]

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":

print("Instance is Positive ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

specific\_h[x] ='?'

general\_h[x][x] ='?'

if target[i] == "no":

print("Instance is Negative ")

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("Specific Boundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

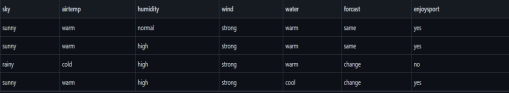
for i in indices:

general\_h.remove(['?', '?', '?', '?', '?', '?']) return specific\_h, general\_h

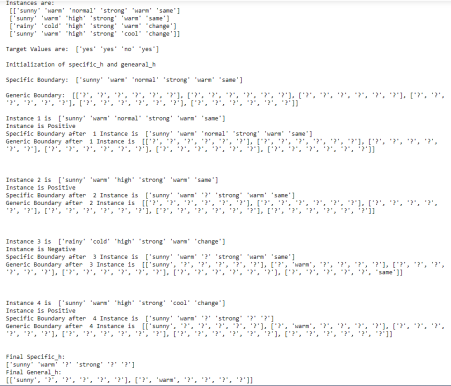
s\_final, g\_final = learn(concepts, target)

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

**Dataset:**

****

**Output:**

****

**Program 3**

**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 pandas as pd

import math

import numpy as np

data = pd.read\_csv("Desktop/data3.csv")

features = [f for f 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])

gain = entropy(examples)

for u in uniq:

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

sub\_e = entropy(subdata)

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

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

uniq = np.unique(examples[max\_feat]) for u in uniq:

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

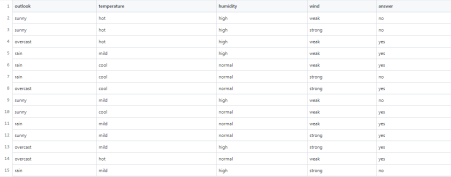
for child in root.children:

printTree(child, depth + 1)

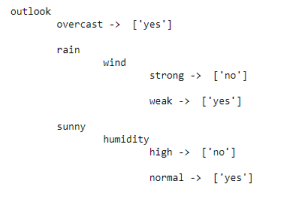
root = ID3(data, features)

printTree(root)

**Dataset:**

****

**Output:**

****

**Program4**

**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 pandas as pd

data = pd.read\_csv('Desktop/dat4.csv.csv')

data.head()

y = list(data['PlayTennis'].values)

X = data.iloc[:,1:].values

print(f'Target Values: {y}')

print(f'Features: \n{X}')

y\_train = y[:8]

y\_val = y[8:]

X\_train = X[:8]

X\_val = X[8:]

print(f"Number of instances in training set: {len(X\_train)}")

print(f"Number of instances in testing set: {len(X\_val)}")

class NaiveBayesClassifier:

def \_\_init\_\_(self, X, y):

self.X, self.y = X, y

self.N = len(self.X)

self.dim = len(self.X[0])

self.attrs = [[] for \_ in range(self.dim)]

self.output\_dom = {}

self.data = []

for i in range(len(self.X)):

for j in range(self.dim):

if not self.X[i][j] in self.attrs[j]:

self.attrs[j].append(self.X[i][j])

if not self.y[i] in self.output\_dom.keys():

self.output\_dom[self.y[i]] = 1

else:

self.output\_dom[self.y[i]] += 1

self.data.append([self.X[i], self.y[i]])

def classify(self, entry):

solve = None

max\_arg = -1

for y in self.output\_dom.keys():

prob = self.output\_dom[y]/self.N

for i in range(self.dim):

cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y] n = len(cases)

prob \*= n/self.N

if prob > max\_arg:

max\_arg = prob

solve = y

return solve

nbc = NaiveBayesClassifier(X\_train, y\_train)

total\_cases = len(y\_val)

good = 0

bad = 0

predictions = []

for i in range(total\_cases):

predict = nbc.classify(X\_val[i])

predictions.append(predict)

if y\_val[i] == predict:

good += 1

else:

bad += 1

print('Predicted values:', predictions)

print('Actual values:', y\_val)

print()

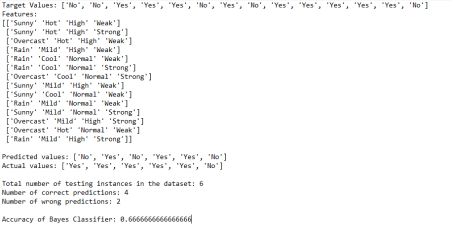
print('Total number of testing instances in the dataset:', total\_cases) print('Number of correct predictions:', good)

print('Number of wrong predictions:', bad)

print()

print('Accuracy of Bayes Classifier:', good/total\_cases)

**Output:**

****

**Program 5**

**Write a program to construct a Bayesian network considering training data. Use this**

**model to make predictions.**

**Program**

!pip install pgmpy

# Starting with defining the network structure

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

#Define a Structure with nodes and edges

cancer\_model = BayesianModel([('Pollution', 'Cancer'),

('Smoker', 'Cancer'),

('Cancer', 'Xray'),

('Cancer', 'Dyspnoea')])

print('Bayesian network nodes:')

print('\t', cancer\_model.nodes())

print('Bayesian network edges:')

print('\t', cancer\_model.edges())

#Creation of Conditional Probability Table

cpd\_poll = TabularCPD(variable='Pollution', variable\_card=2,

values=[[0.9], [0.1]])

cpd\_smoke = TabularCPD(variable='Smoker', variable\_card=2,

values=[[0.3], [0.7]])

cpd\_cancer = TabularCPD(variable='Cancer', variable\_card=2,

values=[[0.03, 0.05, 0.001, 0.02],

[0.97, 0.95, 0.999, 0.98]],

evidence=['Smoker', 'Pollution'],

evidence\_card=[2, 2])

cpd\_xray = TabularCPD(variable='Xray', variable\_card=2,

values=[[0.9, 0.2], [0.1, 0.8]],

evidence=['Cancer'], evidence\_card=[2])

cpd\_dysp = TabularCPD(variable='Dyspnoea', variable\_card=2,

values=[[0.65, 0.3], [0.35, 0.7]],

evidence=['Cancer'], evidence\_card=[2])

# Associating the parameters with the model structure.

cancer\_model.add\_cpds(cpd\_poll, cpd\_smoke, cpd\_cancer, cpd\_xray, cpd\_dysp)

print('Model generated bt adding conditional probability distribution(cpds)')

# Checking if the cpds are valid for the model.

print('Checking for Correctness of model:', end='')

print(cancer\_model.check\_model())

'''print('All local dependencies are as follows')

cancer\_model.get\_independencies()

'''

print('Displaying CPDs')

print(cancer\_model.get\_cpds('Pollution'))

print(cancer\_model.get\_cpds('Smoker'))

print(cancer\_model.get\_cpds('Cancer'))

print(cancer\_model.get\_cpds('Xray'))

print(cancer\_model.get\_cpds('Dyspnoea'))

#Inferencing with Bayesian Network

#Computing the probability of Cancer given smoke

cancer\_infer = VariableElimination(cancer\_model)

print('\nInferencing with Bayesian Network')

print('\nProbability of Cancer given Smoker')

q = cancer\_infer.query(variables=['Cancer'], evidence={'Smoker': 1})

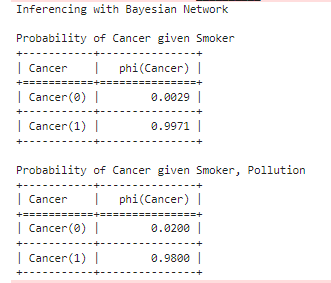
print(q)

print('\nProbability of Cancer given Smoker, Pollution')

q = cancer\_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})

print(q)

**Output**

****

**Program6**

**Apply k-Means algorithm to cluster a set of data stored in a .CSV file.**

**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')

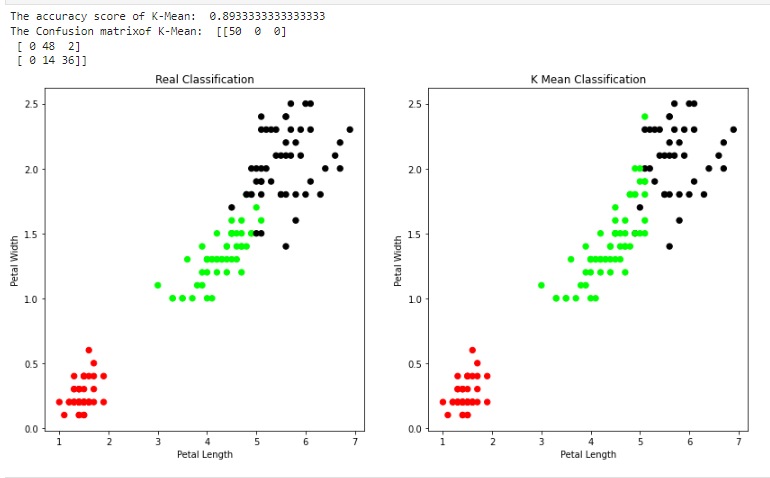
plt.xlabel('Petal Length')

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\_)

**Output:**

****

**Program7**

**Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means**

**algorithm and EM algorithm.**

**Program:**

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')

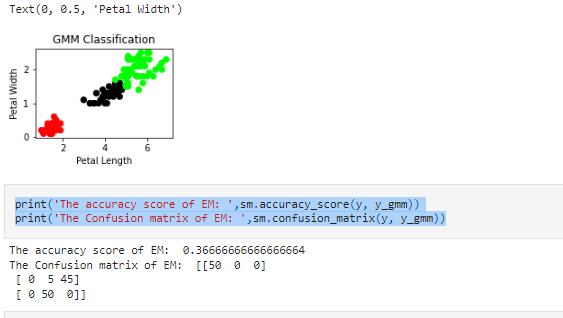
plt.xlabel('Petal Length')

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:**

****

**Program 8**

**Write a program to implement k-Nearest Neighbour algorithm to classify the iris dataset.**

**Print both correct and wrong predictions.**

**Program:**

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('target')

print(Y)

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

classier = KNeighborsClassifier(n\_neighbors=5)

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

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

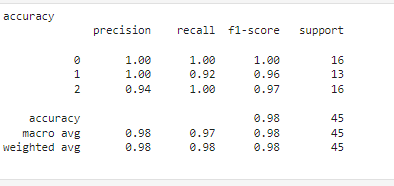
print('confusion matrix')

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

print('accuracy')

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

**Output:**



**Program 9**

**Implement the 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 csv

import matplotlib.pyplot as plt

dataset = pd.read\_csv("Desktop/data9.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

dataset.head()

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_stfrom sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=Fy\_pred = regressor.predict(X\_test)

pd.DataFrame(data={'Actuals': y\_test, 'Predictions': y\_pred})

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue')

plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

plt.figure(figsize=(4,3))

plt.scatter(y\_test,y\_pred)

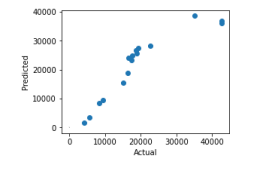
plt.plot([0,50],[0,50],'--k')

plt.xlabel('Actual')

plt.ylabel('Predicted')

plt.show()

**Output:**

****

**Program 10**

**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:**

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,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='blue')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show()

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')

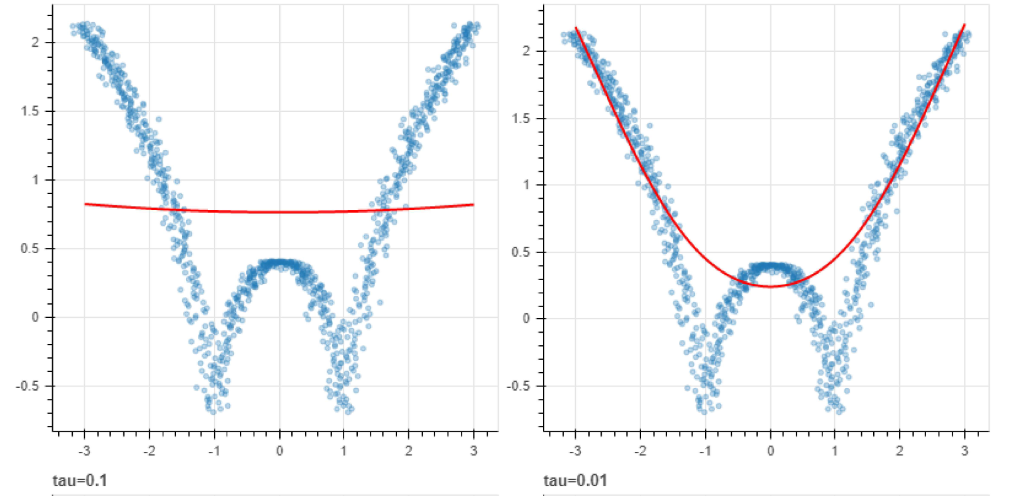
return plot

show(gridplot([

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

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

**Output:**

****