**Ex. No. 1 PERCEPTRON ALGORITHM**

**<https://towardsdatascience.com/perceptron-algorithm-in-python-f3ac89d2e537>**

**Aim** : To implement Perceptron algorithm using python programming.

**Perceptron Algorithm:**

The [Perceptron algorithm](https://en.wikipedia.org/wiki/Perceptron) is a two-class (binary) classification machine learning algorithm.

It is a simplest type of neural network model.It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.

* **Activation** = Weights \* Inputs + Bias

If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.

* **Predict 1**: If Activation > 0.0
* **Predict 0**: If Activation <= 0.0

The Perceptron is a linear classification algorithm. This means that it learns a decision boundary that separates two classes using a line (called a hyperplane) in the feature space. The coefficients of the model are referred to as input weights and are trained using the stochastic gradient descent optimization algorithm.

Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated. The weights of the model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process is repeated for all examples in the training dataset, called an [epoch](https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/). This process of updating the model using examples is then repeated for many epochs.

Model weights are updated with a small proportion of the error each batch, and the proportion is controlled by a hyperparameter called the learning rate, typically set to a small value.

* weights(t + 1) = weights(t) + learning\_rate \* (expected\_i – predicted\_) \* input\_i

Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed. The initial values for the model weights are set to small random values.

**Program** :

from sklearn import datasets

import matplotlib.pyplot as plt

X, y = datasets.make\_blobs(n\_samples=150,n\_features=2,

centers=2,cluster\_std=1.05,

random\_state=2)

#Plotting

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

plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], 'r^')

plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs')

plt.xlabel("feature 1")

plt.ylabel("feature 2")

plt.title('Random Classification Data with 2 classes')

def step\_func(z):

return 1.0 if (z > 0) else 0.0

def perceptron(X, y, lr, epochs):

# X --> Inputs.

# y --> labels/target.

# lr --> learning rate.

# epochs --> Number of iterations.

# m-> number of training examples

# n-> number of features

m, n = X.shape

# Initializing parameters(theta) to zeros.

# +1 in n+1 for the bias term.

theta = np.zeros((n+1,1))

# Empty list to store how many examples were

# misclassified at every iteration.

n\_miss\_list = []

# Training.

for epoch in range(epochs):

# variable to store #misclassified.

n\_miss = 0

# looping for every example.

for idx, x\_i in enumerate(X):

# Insering 1 for bias, X0 = 1.

x\_i = np.insert(x\_i, 0, 1).reshape(-1,1)

# Calculating prediction/hypothesis.

y\_hat = step\_func(np.dot(x\_i.T, theta))

# Updating if the example is misclassified.

if (np.squeeze(y\_hat) - y[idx]) != 0:

theta += lr\*((y[idx] - y\_hat)\*x\_i)

# Incrementing by 1.

n\_miss += 1

# Appending number of misclassified examples

# at every iteration.

n\_miss\_list.append(n\_miss)

return theta, n\_miss\_list

def plot\_decision\_boundary(X, theta):

# X --> Inputs

# theta --> parameters

# The Line is y=mx+c

# So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2

# Solving we find m and c

x1 = [min(X[:,0]), max(X[:,0])]

m = -theta[1]/theta[2]

c = -theta[0]/theta[2]

x2 = m\*x1 + c

# Plotting

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

plt.plot(X[:, 0][y==0], X[:, 1][y==0], "r^")

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs")

plt.xlabel("feature 1")

plt.ylabel("feature 2")

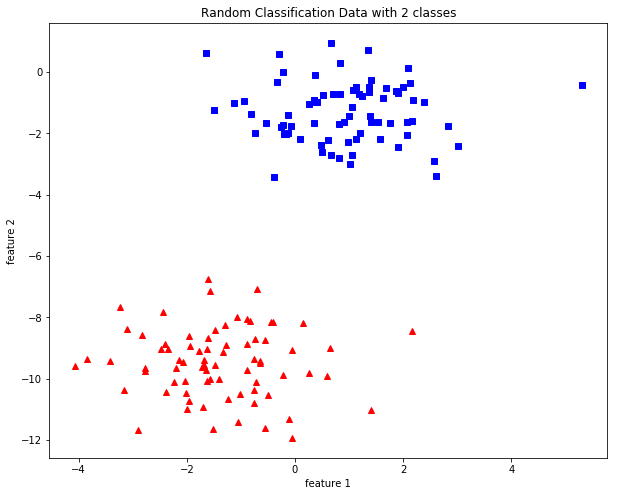
plt.title('Perceptron Algorithm')

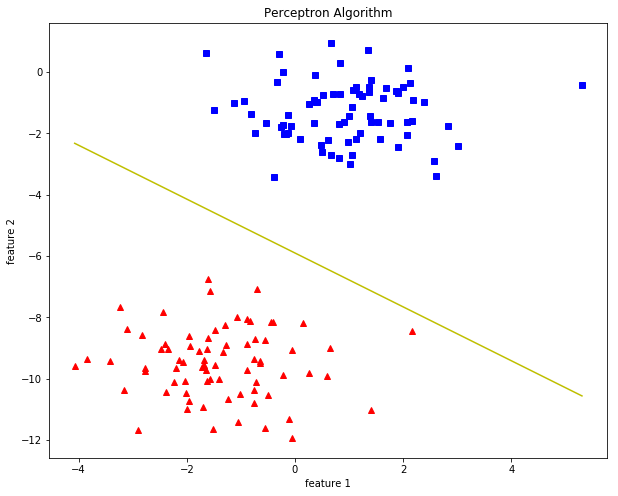
plt.plot(x1, x2, 'y-')

theta, miss\_l = perceptron(X, y, 0.5, 100)

plot\_decision\_boundary(X, theta)

**Output:**

****

****

Result : Thus a python program has been written and executed to implement the

Perceptron algorithm.

**Ex. No. 2 BACKPROPAGATION NEURAL NETWORK**

**Aim** : To implement backpropagation neural network model using Python Programming.

**Program :**

import numpy as np

# X = (hours sleeping, hours studying), y = score on test

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

print(X)

print(y)

# scale units

X = X/np.amax(X, axis=0) # maximum of X array

y = y/100 # max test score is 100

print(X)

print(y)

class Neural\_Network(object):

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

#parameters

self.inputSize = 2

self.outputSize = 1

self.hiddenSize = 3

#weights

self.W1 = np.random.randn(self.inputSize, self.hiddenSize) # (3x2) weight matrix from input to hidden layer

self.W2 = np.random.randn(self.hiddenSize, self.outputSize) # (3x1) weight matrix from hidden to output layer

def forward(self, X):

#forward propagation through our network

self.z = np.dot(X, self.W1) # dot product of X (input) and first set of 3x2 weights

self.z2 = self.sigmoid(self.z) # activation function

self.z3 = np.dot(self.z2, self.W2) # dot product of hidden layer (z2) and second set of 3x1 weights

o = self.sigmoid(self.z3) # final activation function

return o

def sigmoid(self, s):

# activation function

return 1/(1+np.exp(-s))

def sigmoidPrime(self, s):

#derivative of sigmoid

return s \* (1 - s)

def backward(self, X, y, o):

# backward propgate through the network

self.o\_error = y - o # error in output

self.o\_delta = self.o\_error\*self.sigmoidPrime(o) # applying derivative of sigmoid to error

self.z2\_error = self.o\_delta.dot(self.W2.T) # z2 error: how much our hidden layer weights contributed to output error

self.z2\_delta = self.z2\_error\*self.sigmoidPrime(self.z2) # applying derivative of sigmoid to z2 error

self.W1 += X.T.dot(self.z2\_delta) # adjusting first set (input --> hidden) weights

self.W2 += self.z2.T.dot(self.o\_delta) # adjusting second set (hidden --> output) weights

def train(self, X, y):

o = self.forward(X)

self.backward(X, y, o)

NN = Neural\_Network()

for i in range(1000): # trains the NN 1,000 times

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

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

print ("Predicted Output: \n" + str(NN.forward(X)))

print ("Loss: \n" + str(np.mean(np.square(y - NN.forward(X))))) # mean sum squared loss

print ("\n")

NN.train(X, y)

**Input/Output :**

**Input Data in First epoch:**

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

Predicted Output:

[[ 0.64293334]

[ 0.64778178]

[ 0.63319242]]

Loss:

0.0625842148581

**Input in second epoch:**

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

Predicted Output:

[[ 0.66685628]

[ 0.67120326]

[ 0.65596939]]

Loss:

0.0514987603276

………

……….

……………

**Input after 1000 epochs**

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

**Predicted Output after 1000 epochs**

[[ 0.90015511]

[ 0.8766163 ]

[ 0.89265864]]

Loss:

0.000225663124995

**Result :** Thus a python program has been written and executed to implement

backpropagation neural network model.

**Ex. No. 3 RADIAL BASIS FUNCTION NEURAL NETWORK**

**Aim** : To implement Radial Basis Function Neural Network model using Python Programming.

**Program:**

from scipy import \*

from scipy.linalg import norm, pinv

from matplotlib import pyplot as plt

class RBF:

def\_\_init\_\_(self, indim, numCenters, outdim):

self.indim = indim

self.outdim = outdim

self.numCenters = numCenters

#Here we initialize the center corresponding to the hidden neuron of RBF

self.centers = [random.uniform(-1, 1, indim) for i in range(numCenters)]

#Here we are two important parameters that define the RBF network..

#The first parameter represents β, the second represents the connection weight

self.beta = 8

self.W = random.random((self.numCenters, self.outdim))

def \_basisfunc(self, c, d):

assert len(d) == self.indim

return exp(-self.beta \* norm(c-d)\*\*2)

def \_calcAct(self, X):

# calculate activations of RBFs

G = zeros((X.shape[0], self.numCenters), float)

for ci, c in enumerate(self.centers):

for xi, x in enumerate(X):

G[xi,ci] = self.\_basisfunc(c, x)

return G

#Pass x and y values ​​for training

def train(self, X, Y):

""" X: matrix of dimensions n x indim

y: column vector of dimension n x 1 """

# choose random center vectors from training set

rnd\_idx = random.permutation(X.shape[0])[:self.numCenters]

self.centers = [X[i,:] for i in rnd\_idx]

#The center of operation is: [76 21 58 61 2 1 64 77 34 33]

print ("center", self.centers)

# calculate activations of RBFs

G = self.\_calcAct(X)

print (G)

# calculate output weights (pseudoinverse)

self.W = dot(pinv(G), Y)

def test(self, X):

""" X: matrix of dimensions n x indim """

G = self.\_calcAct(X)

Y = dot(G, self.W)

return Y

if \_\_name\_\_ == '\_\_main\_\_':

n = 100

x = mgrid[-1:1:complex(0,n)].reshape(n, 1)

#print(x)

# set y and add random noise

y = sin(3\*(x+0.5)\*\*3 - 1)

y += random.normal(0, 0.1, y.shape)

# rbf regression

rbf = RBF(1, 10, 1)

rbf.train(x, y)

z = rbf.test(x)

# plot original data

plt.figure(figsize=(12, 8))

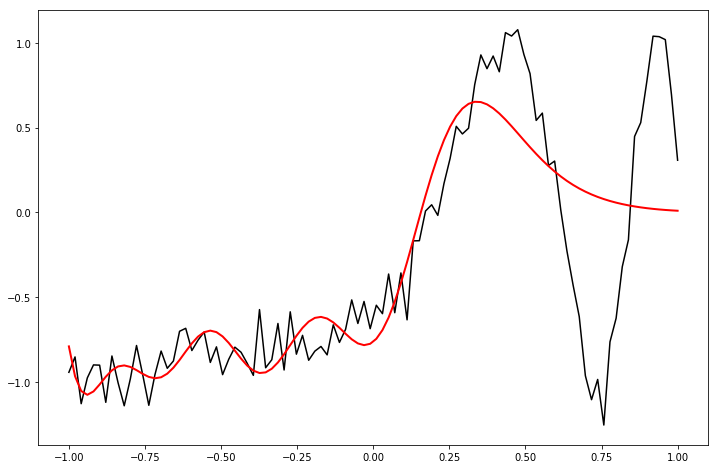
plt.plot(x, y, 'k-')

# plot learned model

plt.plot(x, z, 'r-', linewidth=2)

plt.show()

**Output:**



**Result :** Thus a python program has been written and executed to implement

Radial Basis function neural network model.

**Ex. No. 4 SUPPORT VECTOR MACHINE**

**Aim** : To implement Character Recognition with Support Vector Machine using Python Programming.

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

from subprocess import check\_output

#print(check\_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.

# Standard scientific Python imports

import matplotlib.pyplot as plt

import numpy as np

%matplotlib inline

# Import datasets, classifiers and performance metrics

from sklearn import datasets, svm, metrics

# load the digits dataset

digits = datasets.load\_digits()

print('Digits dataset keys \n{}'.format(digits.keys()))

print('dataset target name: \n{}'.format(digits.target\_names))

print('shape of datset: {} \nand target: {}'.format(digits.data.shape, digits.target.shape))

print('shape of the images: {}'.format(digits.images.shape))

#plot the data, which is just the images flattened into a 1-d array

for i in range(0,4):

plt.subplot(2, 4,i + 1)

plt.axis('off')

imside = int(np.sqrt(digits.data[i].shape[0]))

im1 = np.reshape(digits.data[i],(imside,imside))

plt.imshow(im1, cmap=plt.cm.gray\_r, interpolation='nearest')

plt.title('Training: {}'.format(digits.target[i]))

plt.show()

#the images are also included in the dataset as digits.images

for i in range(0,4):

plt.subplot(2, 4,i + 1)

plt.axis('off')

plt.imshow(digits.images[i], cmap=plt.cm.gray\_r, interpolation='nearest')

plt.title('Training: {}'.format(digits.target[i]))

plt.show()

#from here we will be using the images. First we should flatten the images

n\_samples = len(digits.images)

data\_images = digits.images.reshape((n\_samples, -1))

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data\_images,digits.target)

print('Training data and target sizes: \n{}, {}'.format(X\_train.shape,y\_train.shape))

print('Test data and target sizes: \n{}, {}'.format(X\_test.shape,y\_test.shape))

# Create a classifier: a support vector classifier

classifier = svm.SVC(gamma=0.001)

#fit to the trainin data

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

SVC(gamma=0.001)

# now to Now predict the value of the digit on the test data

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

print("Classification report for classifier %s:\n%s\n"

% (classifier, metrics.classification\_report(y\_test, y\_pred)))

print("Confusion matrix:\n%s" % metrics.confusion\_matrix(y\_test, y\_pred))

**Output:**

Digits dataset keys

dict\_keys(['data', 'target', 'frame', 'feature\_names', 'target\_names', 'images', 'DESCR'])

dataset target name:

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

shape of datset: (1797, 64)

and target: (1797,)

shape of the images: (1797, 8, 8)

****

Training data and target sizes:

(1347, 64), (1347,)

Test data and target sizes:

(450, 64), (450,)

Confusion matrix:

[[40 0 0 0 0 0 0 0 0 0]

[ 0 46 0 0 0 0 0 0 0 0]

[ 0 0 53 0 0 0 0 0 0 0]

[ 0 0 0 49 0 0 0 0 0 0]

[ 0 0 0 0 48 0 0 0 0 0]

[ 0 0 0 0 0 45 0 0 0 0]

[ 0 0 0 0 0 0 47 0 0 0]

[ 0 0 0 0 0 0 0 38 0 0]

[ 0 0 0 0 0 0 0 0 38 0]

[ 0 0 0 1 0 0 0 0 1 44]]

Classification report for classifier SVC(gamma=0.001):

precision recall f1-score support

0 1.00 1.00 1.00 40

1 1.00 1.00 1.00 46

2 1.00 1.00 1.00 53

3 0.98 1.00 0.99 49

4 1.00 1.00 1.00 48

5 1.00 1.00 1.00 45

6 1.00 1.00 1.00 47

7 1.00 1.00 1.00 38

8 0.97 1.00 0.99 38

9 1.00 0.96 0.98 46

accuracy 1.00 450

macro avg 1.00 1.00 1.00 450

weighted avg 1.00 1.00 1.00 450

**Result :** Thus a python program has been written and executed to implement

Character Recognition with Support Vector Machine.

**Ex. No. 5 SELF-ORGANIZING MAPS**

**Aim** : To implement Self-Organizing Map using Python Programming.

**Program:**

import math

class SOM :

# Function here computes the winning vector

# by Euclidean distance

def winner( self, weights, sample ) :

D0 = 0

D1 = 0

for i in range( len( sample ) ) :

D0 = D0 + math.pow( ( sample[i] - weights[0][i] ), 2 )

D1 = D1 + math.pow( ( sample[i] - weights[1][i] ), 2 )

if D0 > D1 :

return 0

else :

return 1

# Function here updates the winning vector

def update( self, weights, sample, J, alpha ) :

for i in range( len ( weights ) ) :

weights[J][i] = weights[J][i] + alpha \* ( sample[i] - weights[J][i] )

return weights

# Driver code

def main() :

# Training Examples ( m, n )

T = [ [ 1, 1, 0, 0 ], [ 0, 0, 0, 1 ], [ 1, 0, 0, 0 ], [ 0, 0, 1, 1 ] ]

m, n = len( T ), len( T[0] )

# weight initialization ( n, C )

weights = [ [ 0.2, 0.6, 0.5, 0.9 ], [ 0.8, 0.4, 0.7, 0.3 ] ]

# training

ob = SOM()

epochs = 3

alpha = 0.5

for i in range( epochs ) :

for j in range( m ) :

# training sample

sample = T[j]

# Compute winner vector

J = ob.winner( weights, sample )

# Update winning vector

weights = ob.update( weights, sample, J, alpha )

# classify test sample

s = [ 1, 1, 0, 1 ]

J = ob.winner( weights, s )

print( "Test Sample s belongs to Cluster : ", J )

print( "Trained weights : ", weights )

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

Test Sample s belongs to Cluster : 1

Trained weights : [[0.6000000000000001, 0.8, 0.5, 0.9], [0.3333984375, 0.0666015625, 0.7, 0.3]]

**Result :** Thus a python program has been written and executed to implement

Self Organizing Map.

|  |  |
| --- | --- |
| **Ex. No. 6** | **Fuzzy Set Operations** |
|  |  |

**Aim:**

To write a MATLAB program to find algebraic sum, algebraic subtraction, algebraic product, bounded sum, bounded subtraction and bounded product of two fuzzy sets.

**Algorithm:**

1. Read the values of the two fuzzy sets.
2. Perform the algebraic sum operation by,

A + B= (a + b) – (a \* b)

1. Perform the algebraic subtraction operation by,

A – B = (a + b`) where b`= 1- b

1. Perform the algebraic product operation by,

A \* B = (a \* b)

1. Perform the bounded sum operation by,

A ⊕B = min [1, (a + b)]

1. Perform bounded subtraction operation by,

A ⊝B= max [0, (a - b)]

1. Perform bounded product operation by,

A ⊙B = max [0, (a + b - 1)]

1. Display the results

**Program:**

a= input(‘Enter the fuzzy set a’ )

b= input(‘Enter the fuzzy set b’)

c= a + b

d= a \* b

as= c – d

e= 1 – b

ad= a + e

f= a – b

bs= min (1, c)

bd= max (0, f)

g= c – 1

bp= max (0,g)

disp(‘The algebraic sum’)

disp(as)

disp(‘The algebraic difference’)

disp(ad)

disp(‘The algebraic product’)

disp(d)

disp(‘The bounded sum’)

disp(bs)

disp(‘The bounded difference’)

disp (bd)

disp(‘The bounded product’)

disp(bp)

**Output:**

Enter fuzzy set a [1 0.5]

Enter fuzzy set b [0.4 0.2]

The algebraic sum

[1.0000 0.6000 ]

The algebraic difference

[1 0.9000]

The algebraic product

[0.4000 0.1000]

The bounded sum

[1.0000 0.7000]

The bounded difference

[0.6000 0.3000]

The bounded product

[0.4000 0]

**Result:**

Thus, a MATLAB program to perform simple fuzzy set operations has been executed and successfully verified.

|  |  |
| --- | --- |
|  | **Fuzzy Membership Functions** |
| **Ex. No: 7** |  |

**Aim:**

To write a program in MATLAB to plot triangular, trapezoidal and bell shaped membership functions.

**Algorithm:**

* 1. Set the limits of x axis.
  2. Calculate y using trimf() function with three parameters for triangular membership function.
  3. Calculate y using trapmf() function with four parameters for trapezoidal membership function.
  4. Calculate y using gbellmf() function with three parameters for bell shaped membership function.
  5. Plot x and y values.

**Program:**

%Triangular membership function

x=(0.0:1.0:10.0)’;

y1= trimf(x, [1 3 5]);

subplot(311 )

plot(x,[y1]);

%Trapezoidal membership function

x=(0.0:1.0:10.0)’;

y1= trapmf(x, [1 3 5 7]);

subplot(312)

plot(x, [y1] );

%Bell-shaped membership function

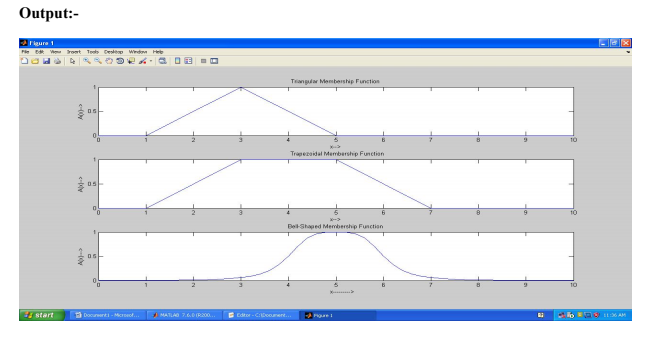
x=(0.0:0.2:10.0);

y1=gbellmf (x,[3 57]);

subplot(313)

plot(x, [y1]);

**Sample Input and Output:**



**Result:**

Thus, the MATLAB program for plotting membership functions has been executed successfully and the output is verified.

|  |  |
| --- | --- |
|  | **Fuzzy Inference System** |
| **Ex. No: 8** |  |

**Aim :**

To implement a Fuzzy Inference System (FIS) for which the inputs, output and rules are given as below.

**Inputs:** Temperature and Cloud Cover

**Temperature:** {Freeze, Cool, Warm and Hot}

**Cloud Cover:** {Sunny, Partly Cloud and Overcast}

**Output:** Speed

**Speed :** {Fast and Slow}

**Rules:**

1. If cloud cover is Sunny and temperature is warm, then drive Fast

Sunny (Cover) and Warm (Temp) -> Fast (Speed)

1. If cloud cover is cloudy and temperature is cool, then drive Slow

Cloudy (Cover) and Cool (Temp) -> Slow (Speed)

**Procedure**

1. Go to command window in Matlab and type fuzzy.
2. Now, new Fuzzy Logic Designer window will be opened.
3. Input / Output Variable
   1. Go to Edit Window and click Add variable.
   2. As per our requirements create two input variables, Temperature and Cloud Cover.
   3. Create one output variable, Speed.
4. Temperature:
   1. Double click the Temperature input variable in Fuzzy Logic Designer window.
   2. New window will be opened and remove all the Membership Functions.
   3. Now, Go to Edit and Click Add MFs and select the 4 Parameters for Temperature Class.
   4. Change the following fields as mentioned data in the given below table.

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs : Temperature 🡪 Freezing, Cool, Warm and Hot | | | |
| MF1:  Range : [0 110]  Name : Freezing  Type :trapmf  Parameter [0 0 30 50] | MF2:  Range : [0 110]  Name : Cool  Type :trimf  Parameter [30 50 70] | MF3:  Range : [0 110]  Name : Warm  Type :trimf  Parameter [50 70 90] | MF4:  Range : [0 110]  Name : Hot  Type :trapmf  Parameter [70 90 110 110] |

1. Similarly, add the datas to the Cloud Cover variables and Speed variables.
2. Cloud Cover:

|  |  |  |
| --- | --- | --- |
| Inputs : Cloud Cover 🡪 Sunny, Partly Cloud and Overcast | | |
| MF1:  Range : [0 100]  Name : Sunny  Type :trapmf  Parameter [0 0 20 40] | MF2:  Range : [0 100]  Name : Partly Cloud  Type :trimf  Parameter [20 50 80] | MF3:  Range : [0 100]  Name : Overcast  Type :trapmf  Parameter [60 80 100] |

1. Speed:

|  |  |
| --- | --- |
| Output : Speed 🡪 Slow and Fast | |
| MF1:  Range : [0 100]  Name : Slow  Type :trapmf  Parameter [0 0 25 75] | MF2:  Range : [0 100]  Name : Fast  Type :trapmf  Parameter [25 75 100 100] |

1. Goto Rules: Edit 🡪 Rules
2. Add the Rules

Rule-1 : Sunny (Cover) and Warm (Temp) -> Fast (Speed)

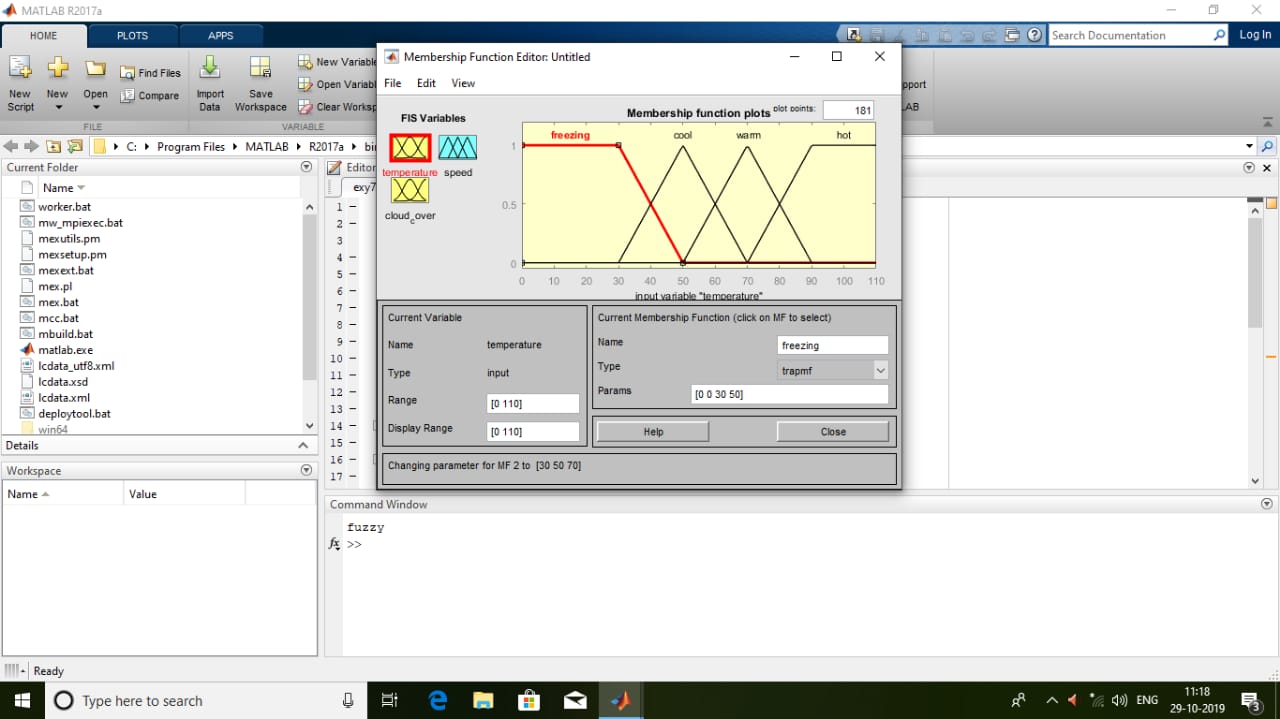
Rule-2 : Cloudy (Cover) and Cool (Temp) -> Slow (Speed)

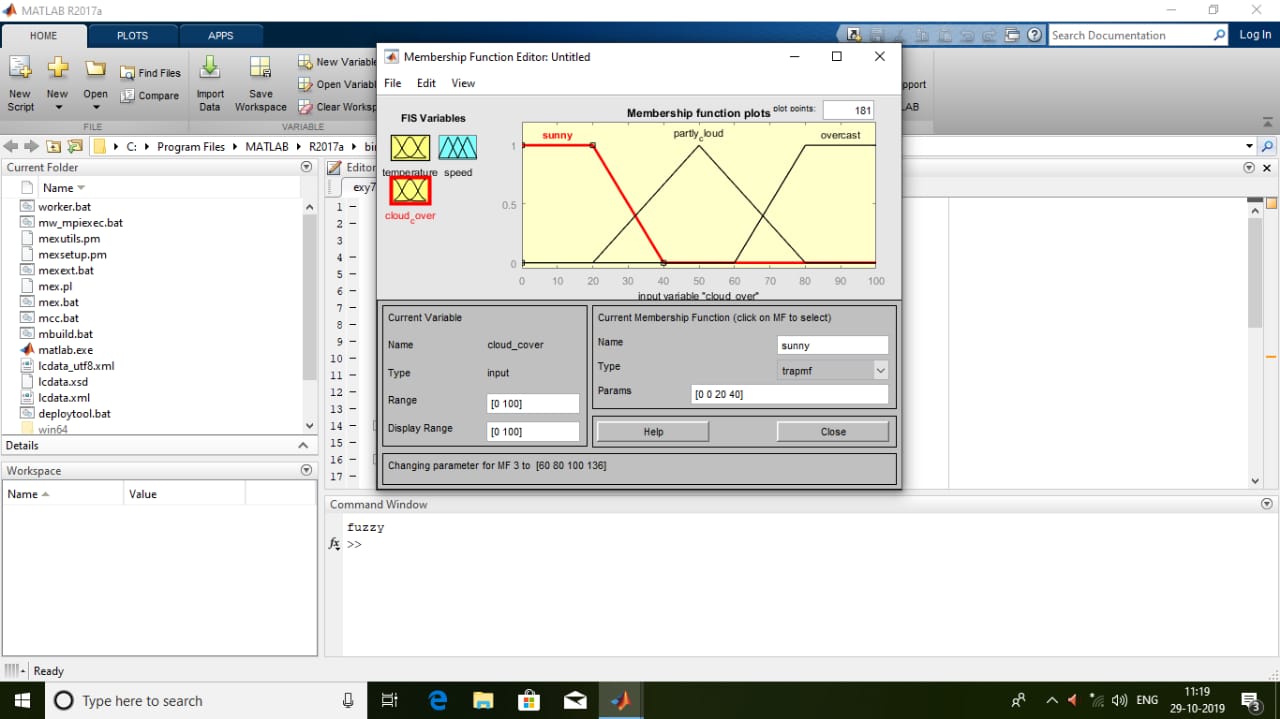
10. Go to view 🡪 Rules

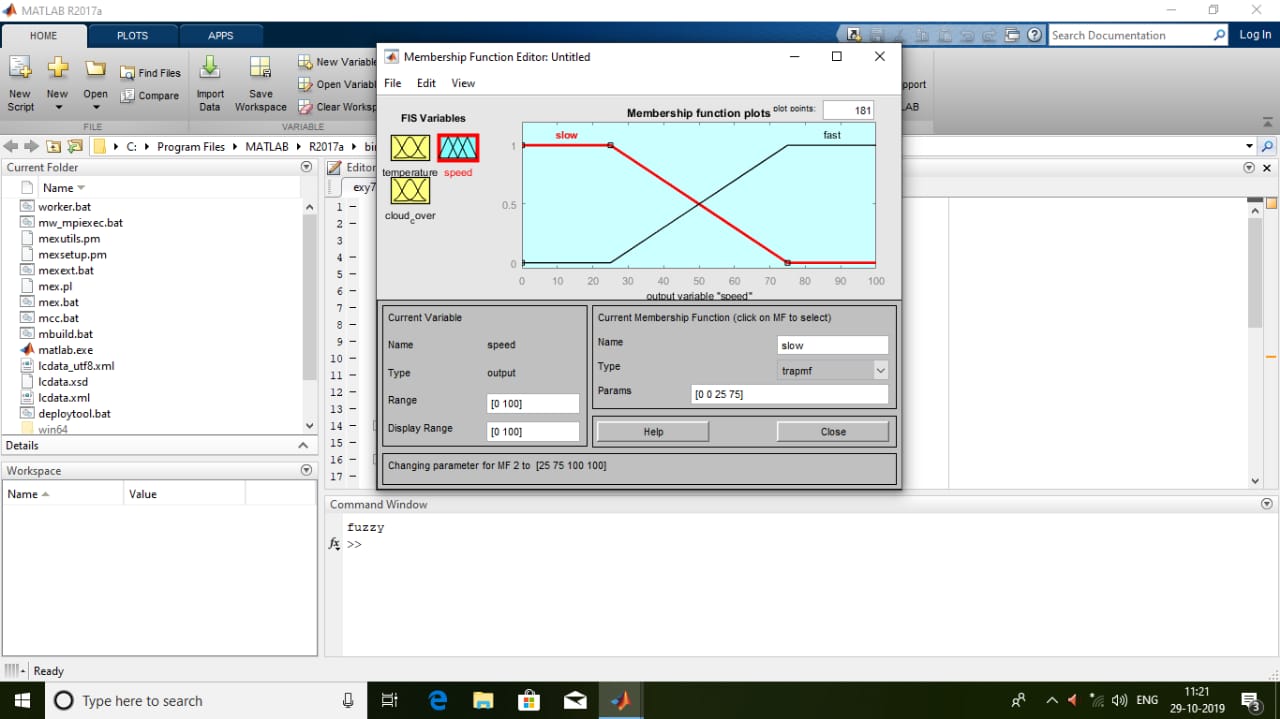
11. Exit.

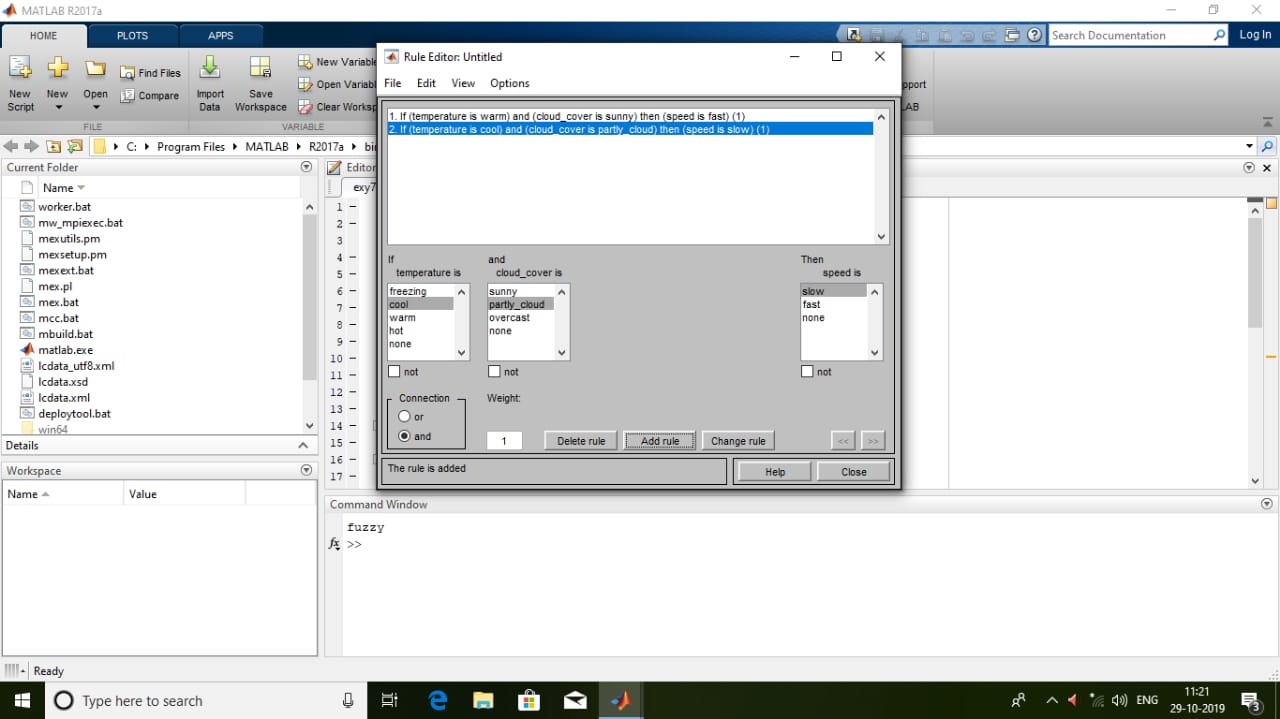
**Sample Input and Output:**

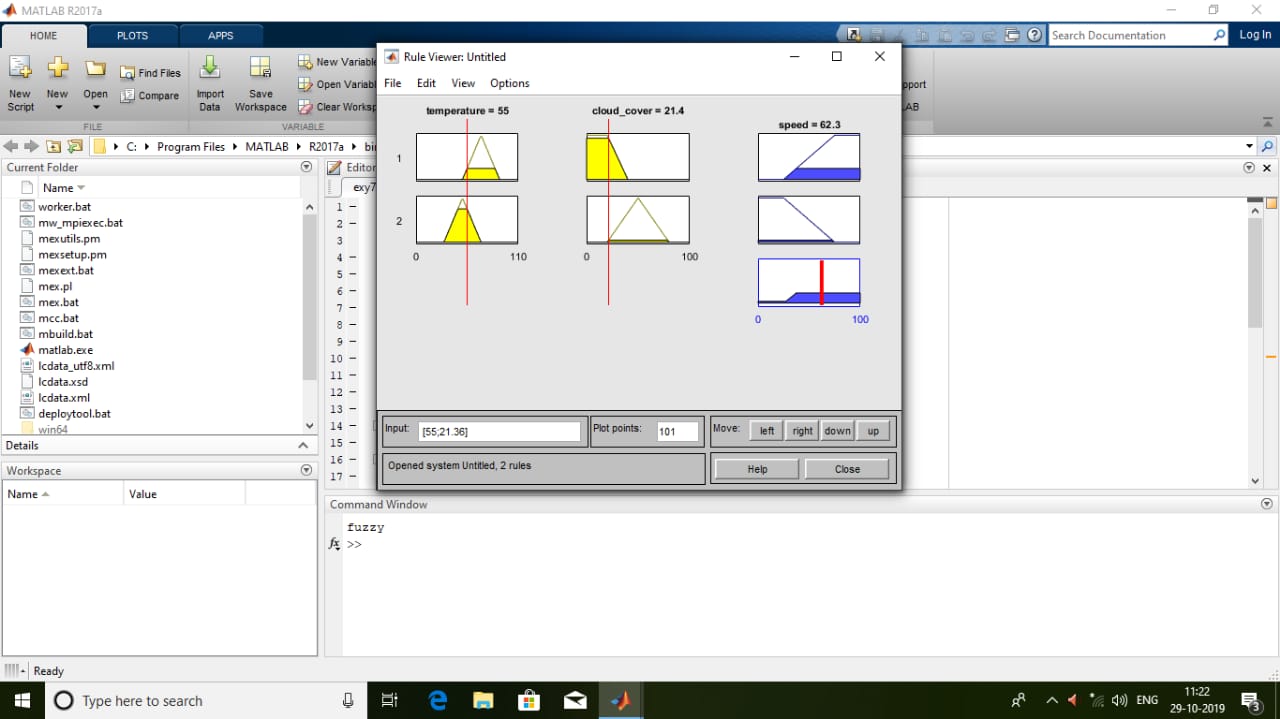
**Membership functions for Temperature variable**

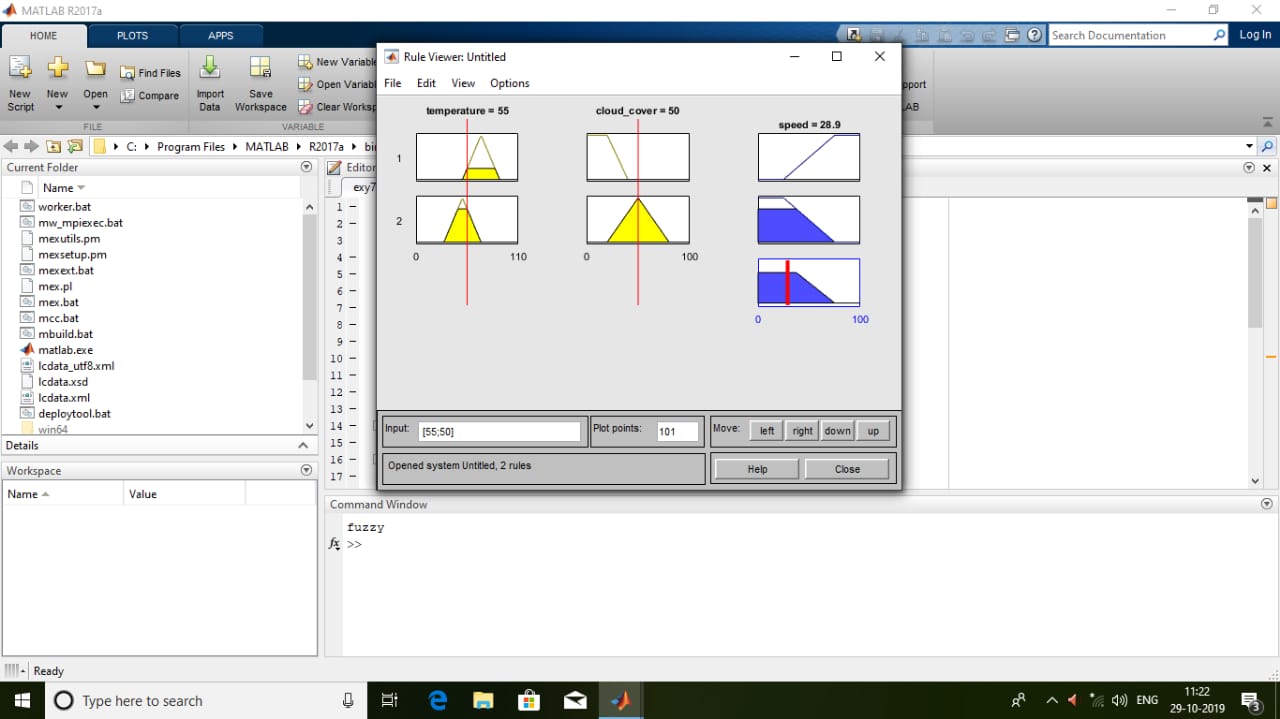












**Result:**

Thus a Fuzzy Inference System is implemented for temperature, cloud cover and speed using the given rules.

**Ex. No. 9 Defuzzification Methods**

|  |  |
| --- | --- |
|  |  |

**Aim:**

To write a program in Python to Implement defuzzification methods using fuzzy toolbox for trapezoidal membership function.

**Algorithm:**

1**.** Import numpy, matplotlib and skfuzzy packages.

2. Generate trapezoidal membership function on range [0, 1] using trapmf function with

four parameters.

3. Defuzzify trapezoidal membership function by using fuzzy toolbox functions centroid, bisector, mom, Som and lom.

4. Plot the defuzzified values using matplotlib.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

import skfuzzy as fuzz

# Generate trapezoidal membership function on range [0, 1]

x = np.arange(0, 5.05, 0.1)

mfx = fuzz.trapmf(x, [2, 2.5, 3, 4.5])

# Defuzzify this membership function five ways

defuzz\_centroid = fuzz.defuzz(x, mfx, 'centroid')

defuzz\_bisector = fuzz.defuzz(x, mfx, 'bisector')

deuzz\_mom = fuzz.defuzz(x, mfx, 'mom')

defuzz\_som = fuzz.defuzz(x, mfx, 'som')

defuzz\_lom = fuzz.defuzz(x, mfx, 'lom')

# Collect info for vertical lines

labels = ['centroid', 'bisector', 'mean of maximum', 'min of maximum',

'max of maximum']

xvals = [defuzz\_centroid,

defuzz\_bisector,

defuzz\_mom,

defuzz\_som,

defuzz\_lom]

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

ymax = [fuzz.interp\_membership(x, mfx, i) for i in xvals]

# Display and compare defuzzification results against membership function

plt.figure(figsize=(8, 5))

plt.plot(x, mfx, 'k')

for xv, y, label, color in zip(xvals, ymax, labels, colors):

plt.vlines(xv, 0, y, label=label, color=color)

plt.ylabel('Fuzzy membership')

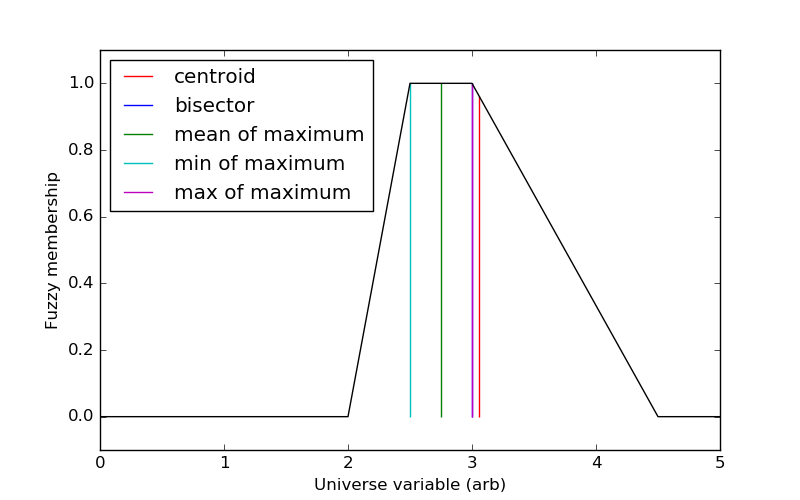
plt.xlabel('Universe variable (arb)')

plt.ylim(-0.1, 1.1)

plt.legend(loc=2)

plt.show()

**Sample Input and Output:**



**Result:**

Thus, the Python program for implementing defuzzification methods has been executed successfully and the output is verified.

Ex. No : 10 **THE TIPPING PROBLEM**

**Aim:** To develop a fuzzy controller for the Tipping problem using python programming.

The 'tipping problem' is commonly used to illustrate the power of fuzzy logic principles to generate complex behavior from a compact, intuitive set of expert rules.

The Tipping Problem

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

A fuzzy control system is created to model how you might choose to tip

at a restaurant. When tipping, the service and food quality are considered and rated between 0 and 10. A tip of between 0 and 25% is suggested.

Problem Formulation:

\* Antecednets (Inputs)

- `service`

\* Universe (ie, crisp value range): How good was the service of the wait

staff, on a scale of 0 to 10?

\* Fuzzy set (ie, fuzzy value range): poor, acceptable, amazing

- `food quality`

\* Universe: How tasty was the food, on a scale of 0 to 10?

\* Fuzzy set: bad, decent, great

\* Consequents (Outputs)

- `tip`

\* Universe: How much should we tip, on a scale of 0% to 25%

\* Fuzzy set: low, medium, high

\* Rules

- IF the \*service\* was good \*or\* the \*food quality\* was good,

THEN the tip will be high.

- IF the \*service\* was average, THEN the tip will be medium.

- IF the \*service\* was poor \*and\* the \*food quality\* was poor

THEN the tip will be low.

\* Usage

- If I tell this controller that I rated:

\* the service as 9.8, and

\* the quality as 6.5,

- it would recommend I leave:

\* a 20.2% tip.

Creating the Tipping Controller Using the skfuzzy control API

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

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership

# functions

quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')

service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')

tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-membership function population is possible with .automf(3, 5, or 7)

quality.automf(3)

service.automf(3)

# Custom membership functions can be built interactively with a familiar,

# Pythonic API

tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

# You can see how these look with .view()

quality['average'].view()

service.view()

tip.view()

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])

rule2 = ctrl.Rule(service['average'], tip['medium'])

rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()

tipping\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API

# Note: if you like passing many inputs all at once, use .inputs(dict\_of\_data)

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

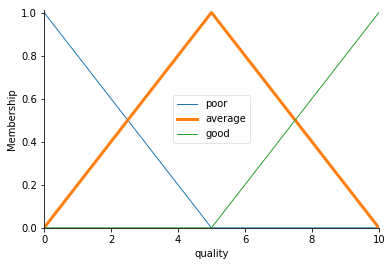
# Crunch the numbers

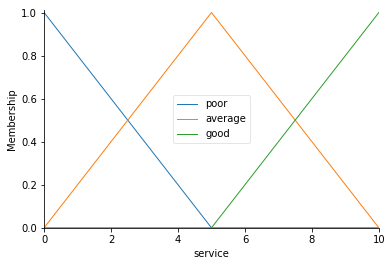
tipping.compute()

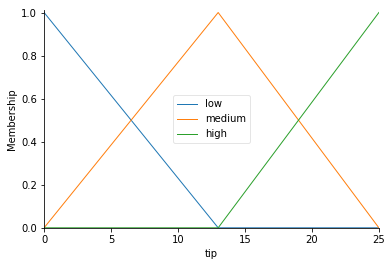
print(tipping.output['tip'])

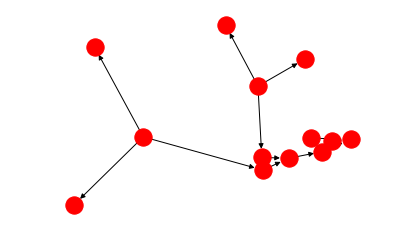
tip.view(sim=tipping)

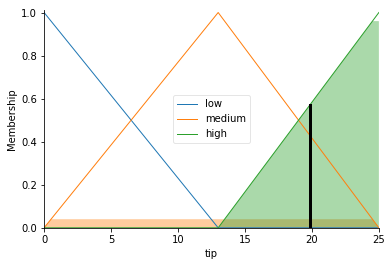
**OUTPUT:**

****









The resulting suggested tip is 20.24%.

**Result:**

Thus, the Python program for developing a fuzzy controller for the Tipping problem has been executed successfully and the output is verified.