**Computational Intelligence CSE3CI Assignment**[**¶**](http://localhost:8888/notebooks/Google%20Drive/La%20Trobe/2018-s1/CI/Jupyter/Untitled1.ipynb#Computional-Intelligence-CSE3CI-Assignment)

A Fuzzy System for Forecasting Electricity Price

**Group members:**

Mai Hung Vu - 18766139  
Jesse McArdle - 18093396  
Jackson Stoney-Dobell – 18893116

La Trobe University



Contents

[1. Initial code setup and outlier removal 3](#_Toc513726276)

[1.1. Library setup and data loading 3](#_Toc513726277)

[1.2. Outlier detection removal for training data 3](#_Toc513726278)

[1.3. Outlier detection removal for testing data 5](#_Toc513726279)

[2. Correlation analysis and generate new data set 7](#_Toc513726280)

[2.1. Generate a correlation matrix 7](#_Toc513726281)

[2.2. Generate new dataset 7](#_Toc513726282)

[2.3. Dataset analysis 7](#_Toc513726283)

[3. Create Antecedent/Consequent objects hold universe variables 10](#_Toc513726284)

[4. Fuzzy membership functions 10](#_Toc513726285)

[4.1. Create fuzzy membership function 10](#_Toc513726286)

[4.2. Analyze fuzzy records 12](#_Toc513726287)

[5. Generate fuzzy rules 14](#_Toc513726288)

[5.1. Put all fuzzy values of training data together 14](#_Toc513726289)

[5.2. Calculate degree of support for all possible outputs and extract fuzzy rules 14](#_Toc513726290)

[5.3. Plot membership functions 17](#_Toc513726291)

[5.4. Setup the fuzzy rules 19](#_Toc513726292)

[5.5. Creating control system 19](#_Toc513726293)

[6. Analyze system performance 20](#_Toc513726294)

[6.1. Calculate the average relative error for the training data set 20](#_Toc513726295)

[6.2. Visualize system output and target output 20](#_Toc513726296)

[6.3. Calculate the average relative error for the testing data set 21](#_Toc513726297)

[6.4. Visualize system output and target output (testing dataset) 22](#_Toc513726298)

[7. Statistical analysis 22](#_Toc513726299)

[8. Conclusion 30](#_Toc513726300)

[8.1. The quality of original training dataset 30](#_Toc513726301)

[8.2. The ability to improve the system 30](#_Toc513726302)

[8.3. The ability to improve the system 30](#_Toc513726303)

### Initial code setup and outlier removal

#### Library setup and data loading

In [2]:

**import** csv

**import** sys

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** skfuzzy **as** fuzz

**from** skfuzzy **import** control **as** ctrl

**%**matplotlib inline

​

**with** open('2018\_CI\_Assignment\_Training\_Data.csv') **as** csvfile:

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

data = [r **for** r **in** readCSV]

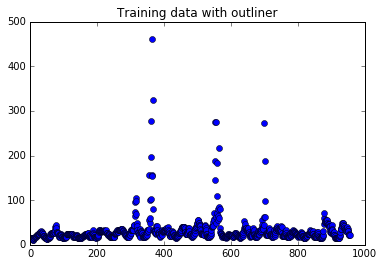
data.pop(0) *#remove header row*

dataArray=np.asanyarray(data,dtype='f') *#save data from training file to array*

plt.plot(dataArray[:,6],'o')

plt.title('Training data with outlier')

plt.show()



#### Outlier detection removal for training data

Q1=np.percentile(dataArray[:,6], 25); *# the value 25 is fixed for every problem;*

Q3=np.percentile(dataArray[:,6], 75); *# the value 75 is fixed for every problem;*

range=[Q1**-**1.5**\***(Q3**-**Q1),Q3**+**1.5**\***(Q3**-**Q1)];

position=np.concatenate((np.where(dataArray[:,6]**>**range[1]),np.where(dataArray[:,6]**<**range[0])),axis=1) *# np.concatenate is used for combining arrays*

p\_new= np.delete(dataArray, position,axis=0) *# np.delete is used for removing some elements in certain positions/places in a list*

​

len(p\_new)

plt.plot(p\_new[:,6],'o')

plt.title('Training data without outlier')

plt.ylim(0, 500)

plt.show()

outlier=[]

p\_new\_max=np.max(p\_new[:,6]) *#get max value of new data set*

count=0

rows=p\_new.shape[0] *# get number of rows in data*

​

**while** count**<**rows:

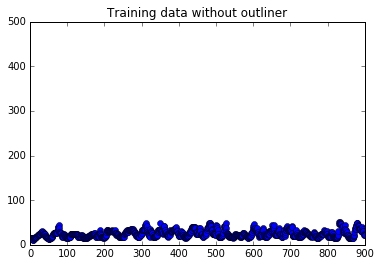
**if**(dataArray[count,6]**>**p\_new\_max): *#create list of outliers*

outlier.append(dataArray[count,6])

count+=1

print("list of outliers:")

print(outlier)



list of outliers:

[64.85, 72.77, 95.09, 103.77, 97.24, 69.41, 155.25, 55.3, 59.61, 101.08, 101.74, 196.34, 276.91, 460.29, 156.48, 153.28, 323.31, 80.21, 52.4, 54.6, 54.69, 52.18, 58.32, 56.47, 71.13, 144.21, 187.0, 274.49, 274.7, 62.48, 71.91, 183.76, 109.23, 64.6, 55.71, 79.26, 216.23, 59.87, 84.36, 80.19, 54.32, 55.05, 273.72, 62.06, 98.49, 188.27, 62.18, 59.81, 72.24, 52.56, 55.95,

54.8, 53.5, 54.07, 55.4]

#### Outlier detection removal for testing data

In [4]:

**with** open('2018\_CI\_Assignment\_Testing\_Data.csv') **as** csvfile:

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

dataTesting = [r **for** r **in** readCSV]

dataTesting.pop(0)

arrayTesting=np.asanyarray(dataTesting,dtype='f')

*#p=array[:,6]*

​

plt.plot(arrayTesting[:,6],'o')

plt.title('Training data with outlier')

plt.show()

​

Q1=np.percentile(arrayTesting[:,6], 25); *# the value 25 is fixed for every problem;*

Q3=np.percentile(arrayTesting[:,6], 75); *# the value 75 is fixed for every problem;*

range=[Q1**-**1.5**\***(Q3**-**Q1),Q3**+**1.5**\***(Q3**-**Q1)];

position=np.concatenate((np.where(arrayTesting[:,6]**>**range[1]),np.where(arrayTesting[:,6]**<**range[0])),axis=1) *# np.concatenate is used for combining arrays*

p\_new\_testing= np.delete(arrayTesting, position,axis=0) *# np.delete is used for removing some elements in certain positions/places in a list*

oo=(p\_new\_testing.shape)

len(p\_new\_testing)

plt.plot(p\_new\_testing[:,6],'o')

plt.title('Training data without outlier')

plt.ylim(0, 500)

plt.show()

outlierTesting=[]

p\_new\_max=np.max(p\_new\_testing[:,6]) *#get max value of new data set*

​

count=0

​

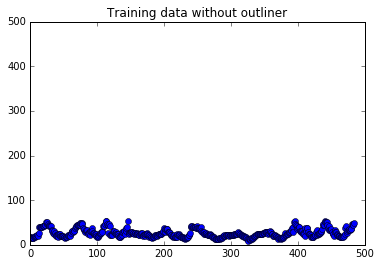
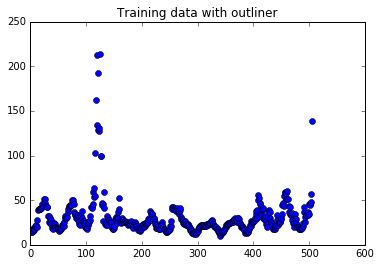
**while** count**<**p\_new\_testing.shape[0]:

**if**(arrayTesting[count,6]**>**p\_new\_max): *#create list of ouliners*

outlierTesting.append(arrayTesting[count,6])

count+=1

print(outlierTesting)



[59.53, 63.49, 102.73, 162.74, 213.1, 134.67, 192.36, 128.53, 127.75, 131.21, 213.88, 99.3, 99.3, 59.32, 56.08, 59.05, 58.03, 58.13, 60.12]

### Correlation analysis and generate new data set

#### Generate a correlation matrix

*#create a input maxtrix for correlation analysis*

inputMatrix=np.column\_stack(p\_new)

*#x1=np.linspace(1, 10, 100)*

​

*#calculate the Correlation Coefficient Matrix*

CCM=np.corrcoef(inputMatrix)

print('Correlation analysis:')

print(CCM)

print()

print(CCM.shape)

​

Correlation analysis:

[[1. 0.97711422 0.94083427 0.43523076 0.49341031 0.54010843

0.49325378]

[0.97711422 1. 0.97678556 0.37103656 0.43915887 0.49715252

0.46219849]

[0.94083427 0.97678556 1. 0.29729517 0.37354106 0.44142235

0.41799109]

[0.43523076 0.37103656 0.29729517 1. 0.98603698 0.9497314

0.52955784]

[0.49341031 0.43915887 0.37354106 0.98603698 1. 0.98603361

0.56492116]

[0.54010843 0.49715252 0.44142235 0.9497314 0.98603361 1.

0.58770781]

[0.49325378 0.46219849 0.41799109 0.52955784 0.56492116 0.58770781

1. ]]

(7, 7)

#### Generate new dataset

The original data has M={T(t-2), T(t-1), T(t), D(t-2), D(t-1), D(t)}. But base on the Correlation analysis we just get M1= { T(t-2),D(t),P(t+1) } as our inputs and output because the relation between T(t-2) and D(t) with P(t+1) are stronger than the other pairs as they are 0.49325378 and 0.58770781 respectively.

In [6]:

*#based on coef analysis the new data set will be (T(t-2),D(t),P(t+1))*

new\_data\_set=p\_new[:, [0,5, 6]] *#extract data from the original columns*

#### Dataset analysis

Display histogram of inputs and output

In [7]:

plt.hist(new\_data\_set[:,0], 3) *# histogram for input1*

plt.title('Histogram of T(t-2)')

plt.ylim(0, 500)

plt.show()

plt.hist(new\_data\_set[:,1], 5) *# histogram for input2*

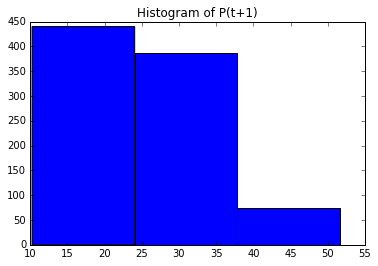
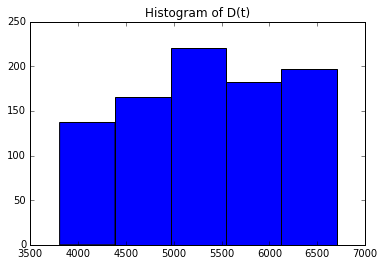
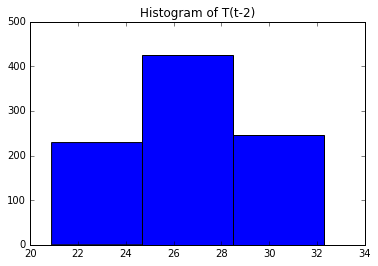
plt.title('Histogram of D(t)')

plt.show()

plt.hist(new\_data\_set[:,2], 3) *# histogram for output*

plt.title('Histogram of P(t+1)')

plt.show()



From the Histograms create linguistic variables and their terms: Temperature T(t-2): low , warm and hot. Demand D(t): very little, little, average, high, very high. PRR P(t+1): low, medium, high

### Create Antecedent/Consequent objects hold universe variables

In this step we also pass the data to each object. The main purpose of this is prepare for the next step because we want to get the fuzzy value corresponding to each value in the training data.

In [8]:

temperature= ctrl.Antecedent(new\_data\_set[:,0],'temperature')

demand=ctrl.Antecedent(new\_data\_set[:,1],'demand')

price=ctrl.Consequent(new\_data\_set[:,2],'price')

### Fuzzy membership functions

#### Create fuzzy membership function

The data from training dataset will be passed through membership functions and the result will be fuzzy values for each record in training data. We save result in variables (A1,A2,A3, B1,B2,B3,B3,B4,B5, C1,C2,C3) corresponding to each linguistic variable.

In [9]:

*#passing 1st column into three function and get the fuzzy values corresponding to each function*

A1=temperature['low']=fuzz.trimf(temperature.universe,[16,16,27])

A2=temperature['warm']=fuzz.trimf(temperature.universe,[16,27,37])

A3=temperature['hot']=fuzz.trimf(temperature.universe,[27,37,37])

*# put them all together*

fuzzyValuesForInput1=np.column\_stack((A1,A2,A3))*# a table of this data will be in APPENDIX A*

plt.hist(fuzzyValuesForInput1, 3) *# histogram for input1*

plt.title('Histogram of fuzzy record for temperature')

print("shape of fuzzyValuesForInput1 ",fuzzyValuesForInput1.shape)

plt.show()

B1=demand['very little']=fuzz.trimf(demand.universe,[3700,3700,4250])

B2=demand['little']=fuzz.trimf(demand.universe,[3700,4250,5250])

B3=demand['average']=fuzz.trimf(demand.universe,[4250,5250,6500])

B4=demand['high']=fuzz.trimf(demand.universe,[5250,6500,6900])

B5=demand['very high']=fuzz.trimf(demand.universe,[6500,6900,6900])

fuzzyValuesForInput2=np.column\_stack((B1,B2,B3,B4,B5))

plt.hist(fuzzyValuesForInput2, 5) *# histogram for input2*

plt.title('Histogram of fuzzy record for demand')

print("shape of fuzzyValuesForInput2 ",fuzzyValuesForInput2.shape)

plt.show()

​

C1=price['low']=fuzz.trimf(price.universe,[0,10,23])

C2=price['medium']=fuzz.trimf(price.universe,[10,23,35])

C3=price['high']=fuzz.trimf(price.universe,[23,35,55])

fuzzyValuesForOutput=np.column\_stack((C1,C2,C3))

plt.hist(fuzzyValuesForOutput, 5) *# histogram for input1*

plt.title('Histogram of fuzzy record for PRR')

print("shape of fuzzyValuesForOutput ",fuzzyValuesForOutput.shape)

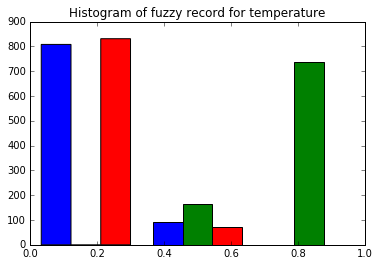
plt.show()

​

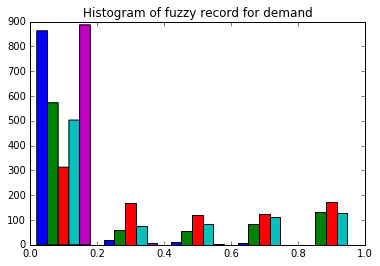
fuzzifiedTrainingDataRecords= np.column\_stack((fuzzyValuesForInput1,fuzzyValuesForInput2,fuzzyValuesForOutput))

#### Analyze fuzzy records

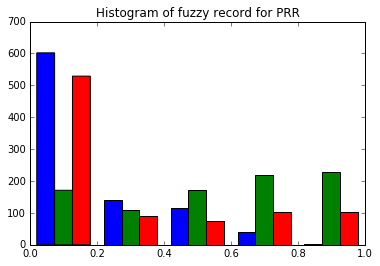
shape of fuzzyValuesForTemperature (901, 3)



shape of fuzzyValuesForDemand (901, 5)



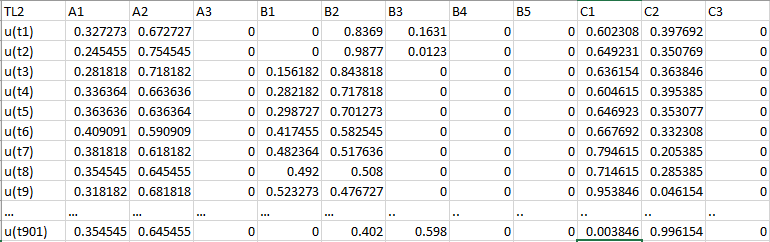
shape of fuzzyValuesForPrice (901, 3)



To explain histograms above let's use the histogram of temperature as an example: the blue columns lie mostly on the left of the histogram, this is because the data has a greater fuzzy value in 1 membership function but also has a smaller fuzzy value or equal to 0 in the other membership function. The same explanations apply for the other histograms. To be more specific, if the temperature is 18 degrees (as an example) then that value would be 0.8 at the membership function “low”, 0.5 at the membership function “medium” and 0 or nearly 0 at the membership function “high”.

### Generate fuzzy rules

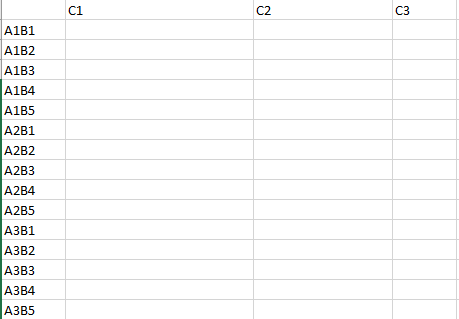
#### Put all fuzzy values of training data together

As the fact that we have 3 linguistic variables for temperature A={A1,A2,A3} and 5 for demand B={B1,B2,B3,B3,B4,B5}. finally, have 3x5= 15 possible fuzzy rules such as: A1B1 A1B2 A1B3 A1B4 A1B5 A2B1 A2B2 A2B3 A2B4 A2B5 A3B1 A3B2 A3B3 A3B4 A3B5. Linguistic variable for output denoted as C={C1,C2,C3} All the degree of support have already calculated based on their membership functions. Now let's put all together and see how the data look like Fuzzified training data: 

#### Calculate degree of support for all possible outputs and extract fuzzy rules

To calculate degree of support for each rule we based on the formula. C:\Users\hungv\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E14A9FF7.tmp

Where μ(Al0 )p, μ(B l1 )p and μ(C l2 )p are the values of membership functions of the linguistic values Al1, Bl0 and Cl2 for the pth record, respectively. • N is the total number of records in training data L2 (the fuzzified records) • l 0 ∈ {1, . . . , c0}, c0 is the number of linguistic values for Temperature • l 1 ∈ {1, . . . , c1}, c1 is the number of linguistic values for Demand 2 ∈ {1, . . . , c2}, c2 is the number of linguistic values for PRR

Outputs of the formula above will be saved in a table as below: 

To calculate the first row, we use a while loop

In [ ]:

count = 0

C1=0

C2=0

C3=0

**while** count**<**901:

C1=C1**+**degreeOfInput1[count,0]**\***degreeOfInput2[count,0]**\***degreeOfOput[count,0] *# change column index number of degreeOfInput1 and degreeOfInput2 to calculate the other rows*

C2=C2**+**degreeOfInput1[count,0]**\***degreeOfInput2[count,0]**\***degreeOfOput[count,1]

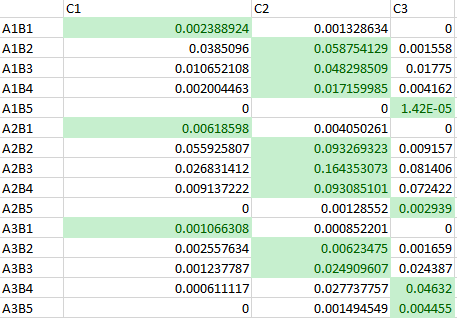
C1=C1**+**degreeOfInput1[count,0]**\***degreeOfInput2[count,0]**\***degreeOfOput[count,2]

count+=1

C1=C1**/**901

C2=C2**/**901

C3=C3**/**901

After calculating each row of the table, the subspace with the highest value of degree of support (highlighted in the table) column will be the output of that rule.

Based on the table above we obtained the following rules:

1. If temperature is LOW and demand is VERY LITTLE then LOW
2. If temperature is LOW and demand is LITTLE then MEDIUM
3. If temperature is LOW and demand is AVERAGE then MEDIUM
4. If temperature is LOW and demand is HIGH then HIGH
5. If temperature is LOW and demand is VERY HIGH then LOW
6. If temperature is WARM and demand is VERY LITTLE then LOW
7. If temperature is WARM and demand is LITTLE then MEDIUM
8. If temperature is WARM and demand is AVERAGE then MEDIUM
9. If temperature is WARM and demand is HIGH then MEDIUM
10. If temperature is WARM and demand is VERY HIGH then HIGH
11. If temperature is HOT and demand is VERY LITTLE then LOW
12. If temperature is HOT and demand is LITTLE then MEDIUM
13. If temperature is HOT and demand is AVERAGE then MEDIUM
14. If temperature is HOT and demand is HIGH then HIGH
15. If temperature is HOT and demand is VERY HIGH then HIGH

#### Plot membership functions

Each membership function is plotted as below:

temperature= ctrl.Antecedent(np.arange(16,37,1),'temperature')

demand=ctrl.Antecedent(np.arange(3700,6900,1),'demand')

price=ctrl.Consequent(np.arange(10,55,1),'price')

​

​

a1=temperature['low']=fuzz.trimf(temperature.universe,[16,16,27])

a2=temperature['warm']=fuzz.trimf(temperature.universe,[16,27,37])

a3=temperature['hot']=fuzz.trimf(temperature.universe,[27,37,37])

​

temperature.view()

​

b1=demand['very little']=fuzz.trimf(demand.universe,[3700,3700,4250])

b2=demand['little']=fuzz.trimf(demand.universe,[3700,4250,5250])

b3=demand['average']=fuzz.trimf(demand.universe,[4250,5250,6500])

b4=demand['high']=fuzz.trimf(demand.universe,[5250,6500,6900])

b5=demand['very high']=fuzz.trimf(demand.universe,[6500,6900,6900])

​

demand.view()

​

c1=price['low']=fuzz.trimf(price.universe,[0,10,23])

c2=price['medium']=fuzz.trimf(price.universe,[10,23,35])

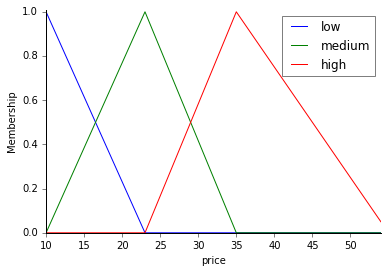
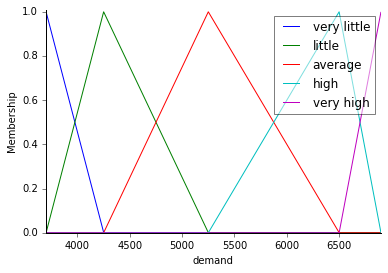
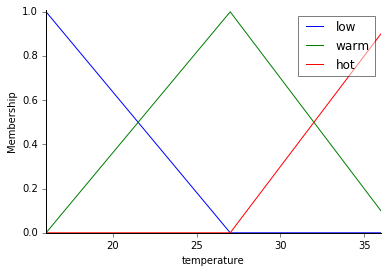
c3=price['high']=fuzz.trimf(price.universe,[23,35,55])

​

price.view()

C:\Users\hungv\AppData\Roaming\Python\Python36\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the figure

"matplotlib is currently using a non-GUI backend, "



#### Setup the fuzzy rules

In [11]:

rule1= ctrl.Rule(temperature['low']**&** demand['very little'],price['low'])

rule2= ctrl.Rule(temperature['low']**&** demand['little'],price['medium'])

rule3= ctrl.Rule(temperature['low']**&** demand['average'],price['medium'])

rule4= ctrl.Rule(temperature['low']**&** demand['high'],price['medium'])

rule5= ctrl.Rule(temperature['low']**&** demand['very high'],price['high'])

rule6= ctrl.Rule(temperature['warm']**&** demand['very little'],price['low'])

rule7= ctrl.Rule(temperature['warm']**&** demand['little'],price['medium'])

rule8= ctrl.Rule(temperature['warm']**&** demand['average'],price['medium'])

rule9= ctrl.Rule(temperature['warm']**&** demand['high'],price['medium'])

rule10= ctrl.Rule(temperature['warm']**&** demand['very high'],price['high'])

rule11= ctrl.Rule(temperature['hot']**&** demand['very little'],price['low'])

rule12= ctrl.Rule(temperature['hot']**&** demand['little'],price['medium'])

rule13= ctrl.Rule(temperature['hot']**&** demand['average'],price['medium'])

rule14= ctrl.Rule(temperature['hot']**&** demand['high'],price['high'])

rule15= ctrl.Rule(temperature['hot']**&** demand['very high'],price['high'])

#### Creating control system

pricing\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3,rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12,rule13, rule14,rule15])

pricing = ctrl.ControlSystemSimulation(pricing\_ctrl)

### Analyze system performance

#### Calculate the average relative error for the training data set

count=0

intput1=new\_data\_set[:,0]

intput2=new\_data\_set[:,1]

targetOutput=np.array(new\_data\_set[:,2])

System\_outputs=np.zeros(901,dtype=np.float64)

**while** count**<**901:

pricing.input['temperature']= intput1[count]

pricing.input['demand']=intput2[count]

pricing.compute()

System\_outputs[count]=pricing.output['price'] *### Save each output in the object 'System\_outputs'*

count+=1

*#%%*

*#print(System\_outputs) # should have three values/elements*

Err=np.sum(np.absolute(targetOutput**-**System\_outputs)**/**np.absolute(targetOutput))**/**901

print('The Average Relative Error Value for traing dataset is', Err)

​

​

plt.plot(targetOutput)

​

plt.plot(System\_outputs)

plt.xlabel('Input')

plt.ylabel('Output')

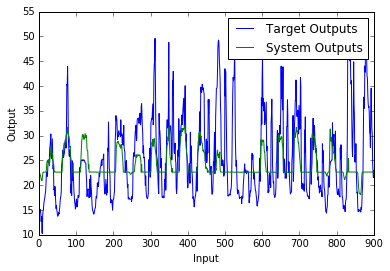
plt.title('')

plt.legend(['Target Outputs', 'System Outputs'])

plt.show()

**The Average Relative Error Value is** 0.20722586411279942

#### Visualize system output and target output



#### Calculate the average relative error for the testing data set

:

*#testing dataset*

count=0

intput1=p\_new\_testing[:,0]

intput2=p\_new\_testing[:,5]

targetOutput=p\_new\_testing[:,6]

​

System\_outputs=np.zeros(485,dtype=np.float64)

**while** count**<**485:

pricing.input['temperature']= intput1[count]

pricing.input['demand']=intput2[count]

pricing.compute()

System\_outputs[count]=pricing.output['price'] *### Save each output in the object 'System\_outputs'*

count+=1

System\_outputs

Err=np.sum(np.absolute(targetOutput**-**System\_outputs)**/**np.absolute(targetOutput))**/**485

print('The Average Relative Error Value for testing dataset is', Err)

plt.cla

plt.clf

plt.figure()

plt.plot(targetOutput)

plt.plot(System\_outputs)

plt.xlabel('Input')

plt.ylabel('Output')

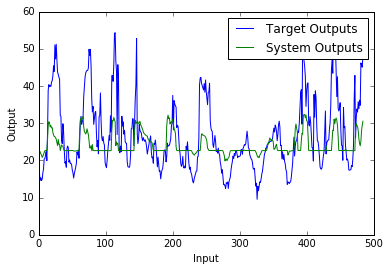
plt.title('')

plt.legend(['Target Outputs', 'System Outputs'])

plt.show()

**The Average Relative Error Value for testing dataset is** 0.2245315505529717

#### Visualize system output and target output (testing dataset)



### Statistical analysis

def workTemp (x):

result = []

for y in x:

if y >= 20.9 and y < 24.7:

result.append('p1')

elif y >= 24.7 and y < 28.64:

result.append('p2')

elif y >= 28.64 and y <= 32.5:

result.append('p3')

else:

result.append(" ")

return result

def workDemand(x):

result = []

for y in x:

if y >= 3500 and y < 4140:

result.append('p1')

elif y >= 4140 and y < 4780:

result.append('p2')

elif y >= 4780 and y < 5420:

result.append('p3')

elif y>= 5420 and y < 6060:

result.append('p4')

elif y >= 6060 and y <= 6700:

result.append('p5')

else:

result.append(" ")

return result

# calling the methods

tempList = new\_data\_set[:,0].tolist()

demandList = new\_data\_set[:,2].tolist()

priceList = new\_data\_set[:,3].tolist()

tempCheck = workTemp(tempList)

demCheck = workDemand(demandList)

# the methods to work out the positions of the values in the graph and record that into the values cell locations

After we have the data we list the 3\*5 possible rules with the temperature represented on the left hand and the demand represented on the right-hand side.

pos11, pos12, pos13, pos14, pos15 = 0, 0, 0, 0, 0

pos21, pos22, pos23, pos24, pos25 = 0, 0, 0, 0, 0

pos31, pos32, pos33, pos34, pos35 = 0, 0, 0, 0, 0

pos11pr, pos12pr, pos13pr, pos14pr, pos15pr = 0, 0, 0, 0, 0

pos21pr, pos22pr, pos23pr, pos24pr, pos25pr = 0, 0, 0, 0, 0

pos31pr, pos32pr, pos33pr, pos34pr, pos35pr = 0, 0, 0, 0, 0

for i in range(len(tempCheck)):

if tempCheck[i] == 'p1' and demCheck[i] == 'p1':

pos11+=1

pos11pr = pos11pr + priceList [i]

elif tempCheck[i] == 'p1' and demCheck[i] == 'p2':

pos12+=1

pos12pr = pos12pr + priceList[i]

elif tempCheck[i] == 'p1' and demCheck[i] == 'p3':

pos13+=1

pos13pr = pos13pr + priceList[i]

elif tempCheck[i] == 'p1' and demCheck[i] == 'p4':

pos14+=1

pos14pr = pos14pr + priceList [i]

elif tempCheck[i] == 'p1' and demCheck[i] == 'p5':

pos15+=1

pos15pr = pos15pr + priceList[i]

elif tempCheck[i] == 'p2' and demCheck[i] == 'p1':

pos21+=1

pos21pr = pos21pr + priceList[i]

elif tempCheck[i] == 'p2' and demCheck[i] == 'p2':

pos22+=1

pos22pr = pos22pr + priceList[i]

elif tempCheck[i] == 'p2' and demCheck[i] == 'p3':

pos23+=1

pos23pr = pos23pr + priceList[i]

elif tempCheck[i] == 'p2' and demCheck[i] == 'p4':

pos24+=1

pos24pr = pos24pr + priceList[i]

elif tempCheck[i] == 'p2' and demCheck[i] == 'p5':

pos25+=1

pos25pr = pos25pr + priceList[i]

elif tempCheck[i] == 'p3' and demCheck[i] == 'p1':

pos31+=1

pos31pr = pos31pr + priceList[i]

elif tempCheck[i] == 'p3' and demCheck[i] == 'p2':

pos32+=1

pos32pr = pos32pr + priceList[i]

elif tempCheck [i] == 'p3' and demCheck[i] == 'p3':

pos33+=1

pos33pr = pos33pr + priceList[i]

elif tempCheck [i] == 'p3' and demCheck[i] == 'p4':

pos34+=1

pos34pr = pos34pr + priceList[i]

elif tempCheck[i] == 'p3' and demCheck[i] == 'p5':

pos35+=1

pos35pr = pos35pr + priceList[i]

After going through the for loop, each value is assigned to a position representing the rule and the total number of values is recorded so we can then print off the average price at that demand and temperature.

print("position 11 ", pos11pr/pos11 )

print("position 12 ", pos12pr/pos12 )

print("position 13 ", pos13pr/pos13 )

print("position 14 ", pos14pr/pos14 )

print("position 15 ", pos15pr/pos15 )

print("position 21 ", pos21pr/pos21 )

print("position 22 ", pos22pr/pos22 )

print("position 23 ", pos23pr/pos23 )

print("position 24 ", pos24pr/pos24 )

print("position 25 ", pos25pr/pos25 )

print("position 31 ", pos31pr/pos31 )

print("position 32 ", pos32pr/pos32 )

print("position 33 ", pos33pr/pos33 )

print("position 34 ", pos34pr/pos34 )

print("position 35 ", pos35pr/pos34 )

which produces this result

position 11 14.485833333333332

position 12 18.028333333333336

position 13 24.275833333333328

position 14 28.014571428571422

position 15 23.08909090909091

position 21 15.166153846153849

position 22 19.077373737373737

position 23 28.269716981132067

position 24 25.3709756097561

position 25 29.96161904761905

position 31 15.821538461538463

position 32 23.195714285714285

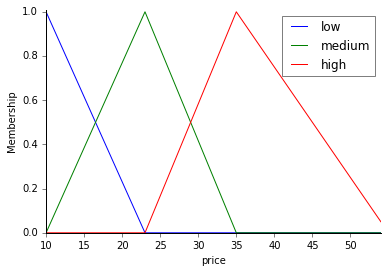
position 33 29.127450980392158

position 34 28.490238095238094

position 35 82.7616666666667

Which is then converted to this table.

|  |  |  |  |
| --- | --- | --- | --- |
| Very high | 23.08909090909091 | 29.96161904761905 | 82.7616666666667 |
| High | 28.014571428571422 | 25.3709756097561 | 28.490238095238094 |
| Average | 24.275833333333328 | 28.269716981132067 | 29.127450980392158 |
| Little | 18.028333333333336 | 19.077373737373737 | 23.195714285714285 |
| Very little | 14.485833333333332 | 15.166153846153849 | 15.821538461538463 |
|  | Low | Warm | hot |



Then using the membership functions from the table above. We compare each average price from the table above and which fuzzy name holds the highest member value for that average price is result of the two fuzzy values in the table. Example. A hot temp and a very little demand correspond to a low price. Using this method again for each cell produces the following rules.

rule1= ctrl.Rule(temperature['low']& demand['very little'],price['low'])

rule2= ctrl.Rule(temperature['low']& demand['little'],price['medium'])

rule3= ctrl.Rule(temperature['low']& demand['average'],price['medium'])

rule4= ctrl.Rule(temperature['low']& demand['high'],price['medium'])

rule5= ctrl.Rule(temperature['low']& demand['very high'],price['medium'])

rule6= ctrl.Rule(temperature['warm']& demand['very little'],price['low'])

rule7= ctrl.Rule(temperature['warm']& demand['little'],price['medium'])

rule8= ctrl.Rule(temperature['warm']& demand['average'],price['medium'])

rule9= ctrl.Rule(temperature['warm']& demand['high'],price['medium'])

rule10= ctrl.Rule(temperature['warm']& demand['very high'],price['medium'])

rule11= ctrl.Rule(temperature['hot']& demand['very little'],price['low'])

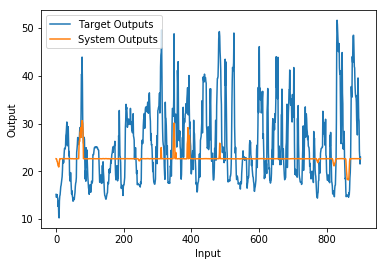
rule12= ctrl.Rule(temperature['hot']& demand['little'],price['medium'])

rule13= ctrl.Rule(temperature['hot']& demand['average'],price['medium'])

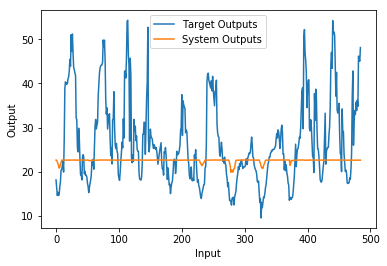
rule14= ctrl.Rule(temperature['hot']& demand['high'],price['medium'])

rule15= ctrl.Rule(temperature['hot']& demand['very high'],price['high'])

Applying these rules to the system produces the following average relative error of 0.23222134924027654 for the training data set.



Applying these rules to the system produces the following average relative error of 0.25104389154549134 for the testing data set.



# apply # apply on training set

input1= new\_data\_set[:,0]

input2= new\_data\_set[:,2]

targetOutput=np.array(new\_data\_set[:,3])

count = 0

System\_outputs=np.zeros(len(tempList),dtype=np.float64)

while count<len(g4):

pricing.input['temperature']= input1[count]

pricing.input['demand']=input2[count]

pricing.compute()

System\_outputs[count]=pricing.output['price'] ### Save each output in the object 'System\_outputs'

count+=1

#print(System\_outputs) # should have three values/elements

Err=np.sum(np.absolute(targetOutput-System\_outputs)/np.absolute(targetOutput))/901

print('The Average Relative Error Value for traing dataset is', Err)

plt.cla

plt.clf

plt.figure()

plt.plot(targetOutput)

plt.plot(System\_outputs)

plt.xlabel('Input')

plt.ylabel('Output')

plt.title('')

plt.legend(['Target Outputs', 'System Outputs'])

plt.show()

# apply on the testing set

count=0

intput1=testingArray[:,0]

intput2=testingArray[:,5]

targetOutput=testingArray[:,6]

System\_outputs=np.zeros(len(g2),dtype=np.float64)

while count<len(g2):

pricing.input['temperature']= intput1[count]

pricing.input['demand']=intput2[count]

pricing.compute()

System\_outputs[count]=pricing.output['price'] ### Save each output in the object 'System\_outputs'

count+=1

System\_outputs

Err=np.sum(np.absolute(targetOutput-System\_outputs)/np.absolute(targetOutput))/485

print('The Average Relative Error Value for testing dataset is', Err)

plt.cla

plt.clf

plt.figure()

plt.plot(targetOutput)

plt.plot(System\_outputs)

plt.xlabel('Input')

plt.ylabel('Output')

plt.title('')

plt.legend(['Target Outputs', 'System Outputs'])

plt.show()

### Conclusion

#### The quality of original training dataset

When we have a closer look at target output, we noticed that there are a huge number of records. When given the same or similar inputs the outputs produced are completely different. From these records we can conclude that the target output has a strong relation to other input variable(s) which is not included in the training and testing dataset. Therefore, the average relative error roughly of 0.2 is an ideal value given the current dataset.

#### The ability to improve the system

Following the approach that we implemented above, there are 2 possible solutions to improve the quality of the system:

1. Finding additional input(s) which has strong correlation coefficient to the output.
2. Adding more data sample to the training dataset.

#### The ability to improve the system

#### Looking at the Average Relative Error Value for both the testing data set and the training data set we notice it is higher than the results used in the method before, so one can deduce from that while statistical analysis is an effective for finding making predictions data mining is more accurate.