**2018 CSE5CI**

**Computational Intelligence Assignment**

# Problem Description

Develop a fuzzy forecasting system for data analysis using Python. The system performs a forecasting task for power marketing price.

## 1.1 System Inputs and Outputs

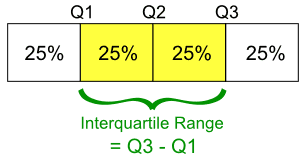
Let the temperature and total demand of electricity at time instant ‘t’ be T(t) and D(t), respectively. The goal of the fuzzy forecasting system is to predict the RRP price by using some historical data as system inputs. The historical data set used for building the fuzzy system at time instant t is composed of a subset of the set M={T(t-2), T(t-1), T(t), D(t-2), D(t-1), D(t)}. The output of your system at time instant t is a forecasting value of the Recommended Retail Price (RRP) of electricity at the next time instant t+1, denoted by P(t+1)

# Outliers Removal

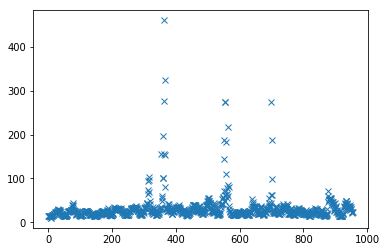
We started by analyzing any presence of the outliers in the training and testing data. An outlier is an input data to an inference system which is at abnormal distance from other input variables in a given population of the data.

## 2.1 Approach taken for outliers’ removal

We eliminated outliers using **interquartile range**. The interquartile range is a measure of variability based on dividing the input set into sets called quartiles. The idea is to divide ordered data set into four equal parts. Thus we have three values between four partitions namely Q1, Q2, and Q3 respectively.



* Q2 is the median of the given input set.
* Q1 and Q3 are the middle values in the first and second half of the input set respectively.
* And, Interquartile range = Q3 - Q1

Then we removed the points which were 1.5\*IQR away from the mean and saved the final training and testing data.

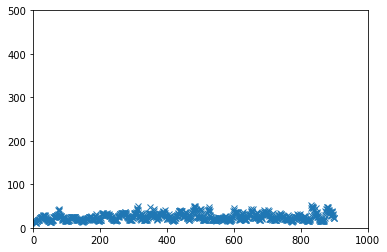


Figure 1Training data with outliers

Figure 2Training data without outliers

# 

Figure Testing data after outlier removal

Figure Testing data with outliers

# Selecting fuzzy subsets and membership functions

# Fuzzy rules from the training data

# Creating Fuzzy inference system

# System performance

## 6.1 Effects of membership function and defuzzification method

# Code for the assignment

======================

In [1]:

#import libraries here

1. import  numpy  as  np
2. import  pandas  as  pd
3. import  matplotlib.pyplot  as  plt
4. import  skfuzzy  as  fuzz  from  skfuzzy
5. import  control  as  ctrl

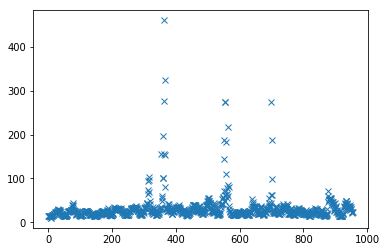
In [2]:

1. fileName = 'Training\_Data.csv'
2. rawTrainingData = pd.read\_csv(fileName).values
3. f = rawTrainingData[: , [6]] f = f.flatten()
4. print(f.shape) plt.plot(f, 'x')

(956,)

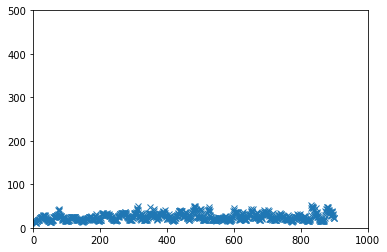
Out[2]:

[<matplotlib.lines.Line2D at 0x15181e4e10>]

In [3]:

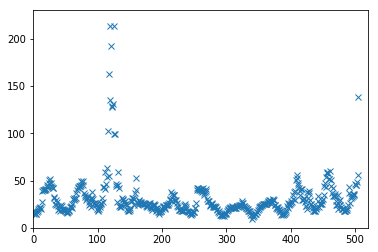
1. Q1 = np.percentile(f, 25);#  the value 25 is fixed  for every problem;
2. Q3 = np.percentile(f, 75);#  the value 25 is fixed  for every problem;
3. r = [Q1 - 1.5 \* (Q3 - Q1), Q3 + 1.5 \* (Q3 - Q1)];
4. pos = np.concatenate((np.where(f > r[1]), np.where(f < r[0])), axis = 1)
5. newData = np.delete(rawTrainingData, pos, axis = 0)
6. g = newData[: , [6]] g = g.flatten()
7. print(g.shape)
8. plt.xlim(0, 1000)
9. # use the same axes setting as the above figure(with three outliers)
10. plt.ylim(0, 500)
11. plt.plot(g, 'x')
12. np.savetxt('ProcessedTrainingData.csv', newData, fmt = '%.2f', delimiter = ',', header = " T(t-2),T(t-1),T(t),D(t-2),D(t-1),D(t),P(t+1)")

(901,)

In [4]:

#remove outliers from the testing data

1. fileName = 'Testing\_Data.csv'
2. rawTestingData = pd.read\_csv(fileName).values
3. f = rawTestingData[: , [6]]
4. f = f.flatten()
5. plt.xlim(0, 520)# use the same axes setting as the above figure(with three outliers) to better reflect the difference
6. plt.ylim(0, 230)
7. plt.plot(f, 'x')
8. Q1 = np.percentile(f, 25);#  the value 25 is fixed  for every problem;
9. Q3 = np.percentile(f, 75);#  the value 75 is fixed  for every problem;
10. r = [Q1 - 1.5 \* (Q3 - Q1), Q3 + 1.5 \* (Q3 - Q1)];
11. pos = np.concatenate((np.where(f > r[1]), np.where(f < r[0])), axis = 1)
12. newTestingData = np.delete(rawTestingData, pos, axis = 0)
13. g = newTestingData[: , [6]]
14. g = g.flatten()

In [5]:

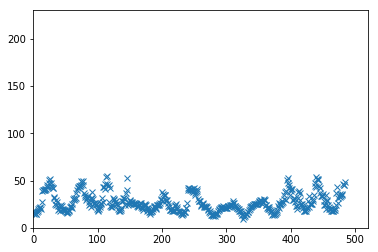
plt.xlim(0, 520) # use the same axes setting as the above figure (with three outliers) to better reflect the difference

plt.ylim(0, 230)

plt.plot(g, 'x')

Out[5]:

[<matplotlib.lines.Line2D at 0x15214d0fd0>]

In [6]:

#save processed testing data

np.savetxt('ProcessedTestingData.csv', newTestingData, fmt='%.2f', delimiter=',', header=" T(t-2),T(t-1),T(t),D(t-2),D(t-1),D(t),P(t+1)")

In [7]:

a = newData[:,[0]] # T(t-2)

a = a.flatten()

b = newData[:,[1]] # T(t-1)

b = b.flatten()

c = newData[:,[2]] #T(t)

c = c.flatten()

d1 = newData[:,[3]] #D(t-2)

d1 = d1.flatten()

d2 = newData[:,[4]] #D(t-1)

d2 = d2.flatten()

d3 = newData[:,[5]] #D(t)

d3 = d3.flatten()

p = newData[:,[6]] # price

p = p.flatten()

v = np.array([a,b,c,d1,d2,d3,p])

CCM=np.corrcoef(v)

plt.matshow(CCM)

groups= ['t-2','t-1','t','d-2','d-1', 'd', 'p']

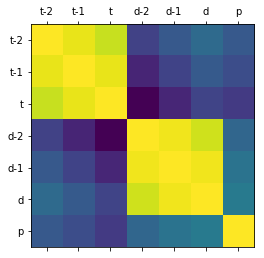
x\_pos = np.arange(len(groups))

plt.xticks(x\_pos,groups)

y\_pos = np.arange(len(groups))

plt.yticks(y\_pos,groups)

plt.show()

In [8]:

temperature = a

plt.hist(temperature)

plt.xlabel('T(t-2)');

In [9]:

temperature = ctrl.Antecedent(np.arange(20,33,1), 'temperature')

t\_cold = fuzz.trimf(temperature.universe, [20, 20, 24])

temperature['cold'] = t\_cold

t\_warm = fuzz.trimf(temperature.universe, [22, 27, 31])

temperature['warm'] = t\_warm

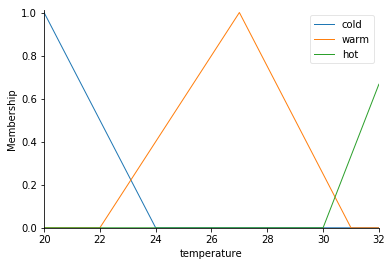
t\_hot = fuzz.trimf(temperature.universe, [30, 33, 33])

temperature['hot'] = t\_hot

temperature.view()

/Users/rpsr15/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:418: UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the figure

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

In [10]:

demand = d3

plt.hist(demand)

plt.xlabel('D(t)');

In [11]:

demand = ctrl.Antecedent(np.arange(3500,7000,5), 'demand')

d\_verylittle = fuzz.trimf(demand.universe, [3500, 3500, 4250])

demand['very little'] = d\_verylittle

d\_little = fuzz.trimf(demand.universe, [3600, 4250, 5100])

demand['little'] = d\_little

d\_middle = fuzz.trimf(demand.universe, [4500, 5400, 6500])

demand['middle'] = d\_middle

d\_high = fuzz.trimf(demand.universe, [5500, 6200, 7000])

demand['high'] = d\_high

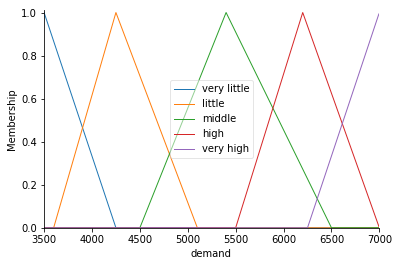
d\_veryhigh = fuzz.trimf(demand.universe, [6250, 7000, 7000])

demand['very high'] = d\_veryhigh

demand.view()

/Users/rpsr15/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:418: UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the figure

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

In [12]:

prices = p

plt.hist(prices)

plt.xlabel('P(t+1)');

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

p\_low= fuzz.trimf(prices.universe, [10,10, 22])

prices['low'] = p\_low

p\_medium = fuzz.trimf(prices.universe, [14, 15,40])

prices['medium'] = p\_medium

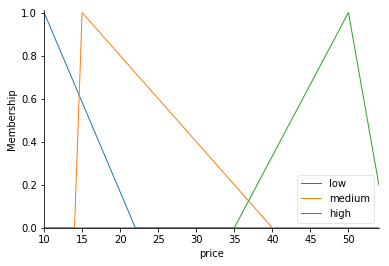
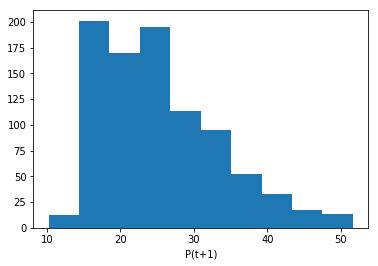
p\_high = fuzz.trimf(prices.universe, [35, 50, 55])

prices['high'] = p\_high

prices.view()

/Users/rpsr15/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:418: UserWarning: matplotlib is currently using a non-GUI backend, so cannot show the figure

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

In [13]:

**#method to generate rule and the degree for each rule, only the rule with max degree would be selected**

def getRule(temp, dmd, prc):

**#calculate membeship for temperature**

m\_t\_cold = fuzz.interp\_membership(temperature.universe,t\_cold, temp)

m\_t\_warm = fuzz.interp\_membership(temperature.universe,t\_warm, temp)

m\_t\_hot = fuzz.interp\_membership(temperature.universe,t\_hot, temp)

temp\_t = [m\_t\_cold, m\_t\_warm, m\_t\_hot]

max\_t = max(temp\_t)

temp\_t = temp\_t.index(max\_t)

**#calculate membership for demand**

m\_d\_vl = fuzz.interp\_membership(demand.universe,d\_verylittle, dmd)

m\_d\_l = fuzz.interp\_membership(demand.universe,d\_little, dmd)

m\_d\_m = fuzz.interp\_membership(demand.universe,d\_middle, dmd)

m\_d\_h = fuzz.interp\_membership(demand.universe,d\_high, dmd)

m\_d\_vh = fuzz.interp\_membership(demand.universe,d\_veryhigh, dmd)

temp\_d = [m\_d\_vl, m\_d\_l, m\_d\_m, m\_d\_h, m\_d\_vh]

max\_d = max(temp\_d)

temp\_d = temp\_d.index(max\_d)

#calculate for price

m\_p\_l = fuzz.interp\_membership(prices.universe,p\_low, prc)

m\_p\_m = fuzz.interp\_membership(prices.universe,p\_medium, prc)

m\_p\_h = fuzz.interp\_membership(prices.universe,p\_high, prc)

temp\_p = [m\_p\_l, m\_p\_m, m\_p\_h]

max\_p = max(temp\_p)

temp\_p = temp\_p.index(max\_p)

degreeRule = max\_t \* max\_d \* max\_p

rule\_string = "{}{}{}".format(temp\_t, temp\_d, temp\_p)

tup1 = (rule\_string, degreeRule)

return tup1

In [14]:

rulebase = {"000":1.0}

for i in range(a.size):

rule\_result = getRule(a[i], d3[i], p[i])

if rule\_result[0] in rulebase:

if rule\_result[1] > rulebase[rule\_result[0]]:

rulebase[rule\_result[0]] = rule\_result[1] \* rulebase[rule\_result[0]]

#print("upgrading",(rule\_result[1] \* rulebase[rule\_result[0]]))

else:

rulebase[rule\_result[0]] = rule\_result[1]

In [15]:

**#creating rule base using algorithm based on paper published by lin wang, mentioned in references**

final\_rules = {"00":(0, 1.0)}

for rule\_s, rule\_de in rulebase.items():

in\_val = rule\_s[:2]

out\_val = rule\_s[2]

dd = round(rule\_de,6)

print(in\_val, out\_val, dd)

#if alrready there check for degree of rule

if in\_val in final\_rules:

if dd > final\_rules[in\_val][1]:

final\_rules[in\_val] = (out\_val, dd)

#else add to final rules

else:

#temp\_st =

final\_rules[in\_val] = (out\_val, dd)

00 0 1.0

11 1 0.0

11 0 5e-06

01 0 3.2e-05

01 1 0.0

12 1 0.0

13 1 0.0

23 1 0.0

23 2 0.0

24 2 0.000342

14 2 0.022461

24 1 0.037957

22 1 0.0

13 2 0.0

14 1 0.094978

12 2 0.0

03 1 0.0

03 2 0.060499

02 1 0.0

02 2 0.06468

21 1 0.000503

22 2 0.005958

10 1 0.000548

10 0 0.227959

In [16]:

print(final\_rules)

{'00': (0, 1.0), '11': ('0', 5e-06), '01': ('0', 3.2e-05), '12': ('1', 0.0), '13': ('1', 0.0), '23': ('1', 0.0), '24': ('1', 0.037957), '14': ('1', 0.094978), '22': ('2', 0.005958), '03': ('2', 0.060499), '02': ('2', 0.06468), '21': ('1', 0.000503), '10': ('0', 0.227959)}

In [17]:

#3 linguistic variables for temperature and 5 liguistic variables for demand

#so it can be represented on a 2d matrix with 5\*3 = 15 rules

#rules generated from algorithm above based on research paper(mentioned in references)

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

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

rule3 = ctrl.Rule(temperature['cold'] & demand['middle'], prices['high'])

rule4 = ctrl.Rule(temperature['cold'] & demand['high'], prices['high'])

rule5 = ctrl.Rule(temperature['cold'] & demand['very high'], prices['high'])

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

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

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

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

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

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

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

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

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

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

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

price\_simulation = ctrl.ControlSystemSimulation(prices\_ctrl)

In [18]:

#read processed training and testing data

finalTrainingData = pd.read\_csv("ProcessedTrainingData.csv").values

finalTrainingData = finalTrainingData[:,(0,5,6)]

finalTestingData = pd.read\_csv("ProcessedTestingData.csv").values

finalTestingData = finalTestingData[:,(0,5,6)]

In [21]:

System\_outputs=np.zeros(finalTrainingData.shape[0],dtype=np.float64)

i = 0

sum = 0

for t\_val in finalTrainingData:

price\_simulation.input['temperature'] = t\_val[0]

price\_simulation.input['demand'] = t\_val[1]

price\_simulation.compute()

sim\_out = price\_simulation.output['price']

diff = abs(t\_val[2] - (sim\_out))

div = diff/t\_val[2]

sum += div

System\_outputs[i]=sim\_out

i += 1

print("Average Relative Error for Training Data: %.2f%%" %((sum/finalTrainingData.shape[0])\*100))

**Average Relative Error for Training Data: 20.36%**

In [22]:

System\_outputs=np.zeros(finalTestingData.shape[0],dtype=np.float64)

i = 0

sum = 0

for t\_val in finalTestingData:

price\_simulation.input['temperature'] = t\_val[0]

price\_simulation.input['demand'] = t\_val[1]

price\_simulation.compute()

sim\_out = price\_simulation.output['price']

diff = abs(t\_val[2] - (sim\_out))

div = diff/t\_val[2]

sum += div

System\_outputs[i]=sim\_out

i += 1

print("Average Relative Error for Testing Data: %.2f%%" %((sum/finalTestingData.shape[0])\*100))

**Average Relative Error for Testing Data: 19.96%**

# References

* Generating Fuzzy rules by learning from examples by li-xin wang, Jerry M. Mendel