Programming Part A:

(graphs on next page)

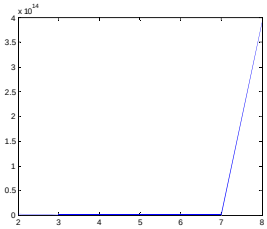
For this programming part I ended up using Regularized Least Squares where I varied lambda between 0, .1, .5 , 10, 500,100, and a 1000 . For the basis, I ended up using a variation of a polynomial basis with some additional terms (explained below). I determined the goodness of these functions by plotting the Total Error function for Regularized least squares as I tried out different variables. The Mathematical step by step logical and formula build up is as follows:

I explain how I chose my functional basis below. But assuming that I had a functional basis I decided that using a regularized least squares would be the best approach. I decided to first implement the equations from 3.1 since that represented the case lambda = 0. After I had that first case down I then would extend the formulas to their more general equation. So for obtaining the weights I first used equation 3.15 and then later changed it to 3.28 in order to incorporate lambda. For the total error function, I first I first used (3.12) to describe the error and then changed it to formula 3.27 when moving to different values of lambda. In my code when changing to the more generalized version, I commented out the original code so you can still see my original implementation. Then as I ran through all possible choices of the missing third value, I would keep track of the error and plot the error.

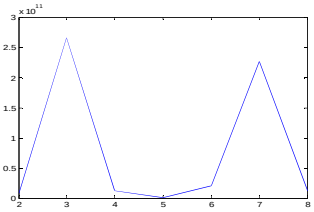
Since we were given that FTP (feature x1) and WE(feature x2) for now are good predictors with one other variable, I assumed that the basis should include those variables somehow. Then since we have to find a third variable I just called it x_3. Since the variables might interact with each other in some way, just using a form of the three variables by themselves might not describe the functional space. I decided that linear combinations of this variable should also be expressed in this basis. So I originally decided to make three of the basis functions : {phi0 = 1, phi1 =x1 phi2 = x2, phi3 = x3}. But when I plotted these errors, the graph produced came out poorly. Therefore I decided to switch the basis to a polynomial format. So then the basis was {phi0 = 1, phi1 = x1, phi2 = x2^2 , phi3 = x3^3}. I tried to also experiment with gaussian and logistic sigmoid functions but was running into problems of the errors being either infinity or 0. Therefore I decided that polynomial basis was the best way to go. Trial's with the lambda varying show that the predictions are fairly consistent and therefore there justifies using a polynomial basis.

The plots of all the trial runs seem to show that variable x5 generated the lowest total error, followed by variable x4. These variables correspond to LIC and GR respectively. It was very beneficial to vary the lambda as the difference in error between LIC and GR became more prominent as lambda increased.

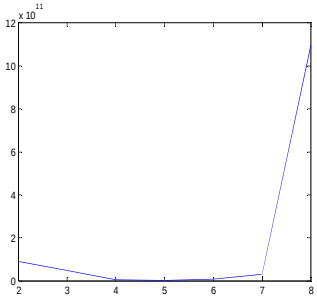
Plots on the next page.



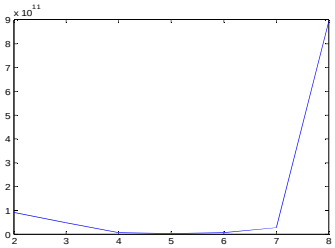
My Naive Radial Basis Attempt?



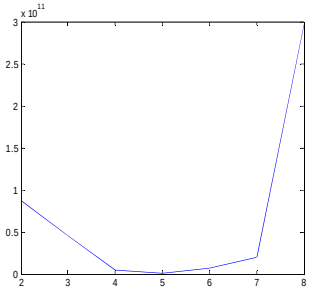
My Polynomial Basis Attempt (lambda = 0)



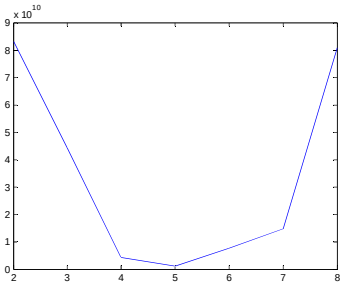
Polynomial Basis attempt (lambda = 1)



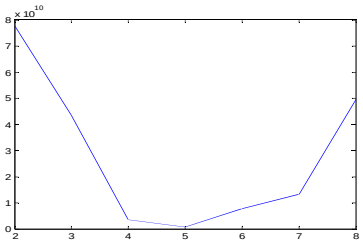
Lambda = 10



Lambda = 100



Lambda = 500



Lambda = 1000

Programming Part B: Nearest Neighbor.

(I) Method for changing the question marks when the features are categories.
Justification for the Mean/Median Imputation. What follows is my reason and justification for what to change the question marks to. In general I used a cut command to see the general distribution of categories, as well as the long grep piped with the cut command to see the distribution for positive and negative labeled samples. The only real different cases were features A9 and A10 where the instances with positive and negative labels had different letter appear the most times. In these cases the question marks should be changed based on the label classification. (At the end of the write up I have attached the output from my shell) I decided to impute based on just the training since I felt the training set is more important in the k nearest neighbor algorithm

1. Feature A1:

Label all question marks to 'b' since no matter the positive or negative classification, 'b' appeared the most times.

2. Feature A4:
   Label all question marks to 'u' since no matter the positive or negative classification 'u' appears the most times
3. Feature A5:

Label question marks to 'g' for the same reasoning as above.

4. Feature A6:

Label question marks to 'c' for the same reasoning as above.

5. Feature A7:
   Label question marks to 'v' for the same reasoning as above.
6. Feature A9:
   'f' occurred more frequently in negative samples and 't' occurred more frequently in positive samples. So any question marks in for this feature in instances with a negative label should be changed to 'f' while positive instances should be changed to 't'.

7. Feature A10:

't' occurred more frequently in negative samples and 'f' occurred more frequently in positive samples. So any question marks in for this feature in instances with a negative label should be changed to 't' while positive instances should be changed to 'f'.

8. Feature A12: Change question marks to f
9. Feature A13: Change question marks to g
10. Field A16: No question marks present

(II) Method for changing Question marks for when the features are continuous variables.
For real value features I replaced the missing values with the label conditioned mean. This meant counting how many instances of each label there were, summing their values and dividing by the amount. I assumed that it was better to replace question marks based on the training data set since for both data sets during the processing since I assumed that it had a higher significance since we are using that as our data base.

(III) Z Scaling: for real values I used the z scaling as described in the homework handout.

For Z scaling the mean and standard deviation are calculated with for each data set separately.

(IV) I wrote the programs in python and named them knn.py and process.py They were written in python 2.7 (tried it on csa2 and it ran) I implemented it based of an algorithm I found from http://www.math.le.ac.uk/people/ag153/homepage/KNN/OliverKNN_Talk.pdf

(V) In order to create a program that ran on the BU Computing enviroment, I made a shell script called process. This shell script took the parameters and passes them into a python program called process.py and runs the program. So the command "./process crx.data.training crx.data.testing" can be used to run the python program that will then produce crx.training processed and crx.testing.processed. For the man/median imputation, I decided to only use the training data. I did a similar analysis on the testing set and changed them seperately.
My Method that the python program takes the file names, and imports them as csv.

Table of Results for K Nearest Neighbors:

For at least 2 different values of k. I wrote it into the knn.py file a smalle script that checks the accuracy in a similar format to the pearl script. The results seem to make sense because the more neighbors you take, the more opportunities there are for an individual point to be mislabeled.

K Value	Lenses Results	CRX results
1	83.00%	100%?
2	67.00%	89.49%
3	83.00%	89.84
10	50.00%	85.00%
100	NA	78.00%
300	NA	69.00%