EE239AS - Project 1 Report

Group Members:
1) Rebecca Correia - 204587944
2) Salil Kanetkar - 704557096
3) Vedant Patil - 104590942

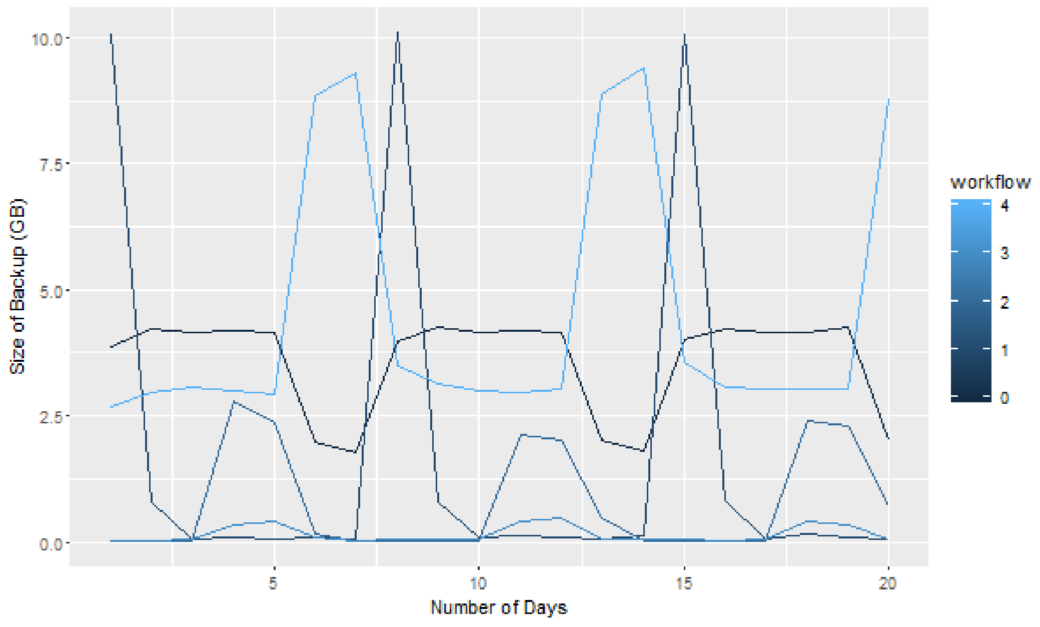
Network Backup Dataset

The given Network Backup Dataset consists of the following parameters:

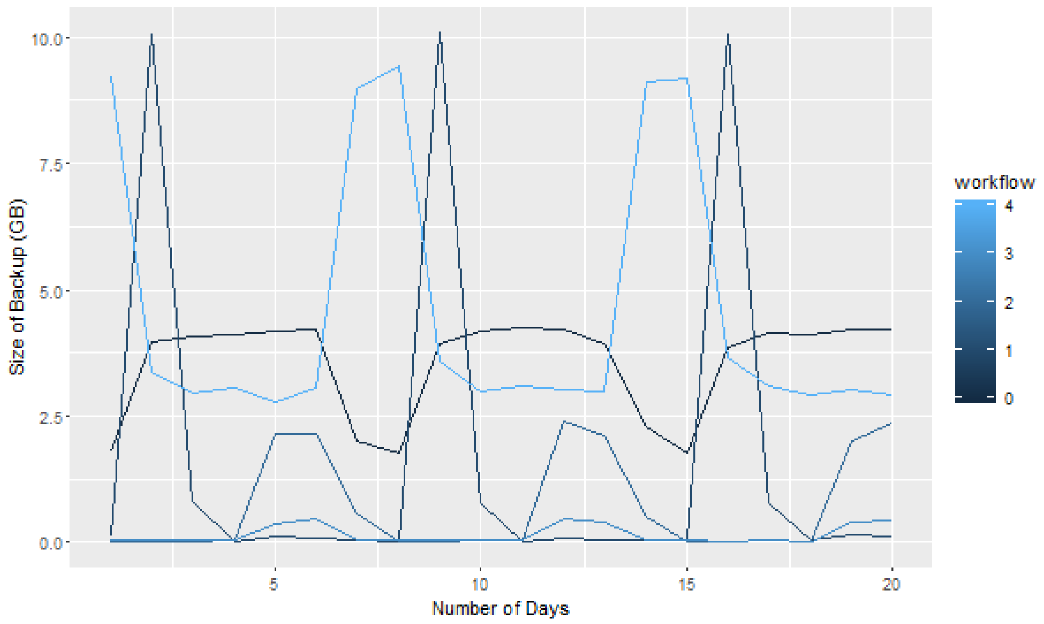
- Week index
- Day of the week – at which the files is backed up starts
- Backup start time-hour of the day – the exact time that the backup process is completed
- Workflow ID
- File name
- Backup size – the size of the file that is backed up in that cycle in GB
- Backup time – the duration of the backup procedure in hour

Problem 1:

In order to get an idea about the relationship between the attributes of the dataset, we first found out a subset of the dataset such that it grouped the total copy sizes for each workflow that existed during the first 20 days. The resulting plot was as follows:



We then applied the same technique to plot for the next 20 days, yielding the following graph:



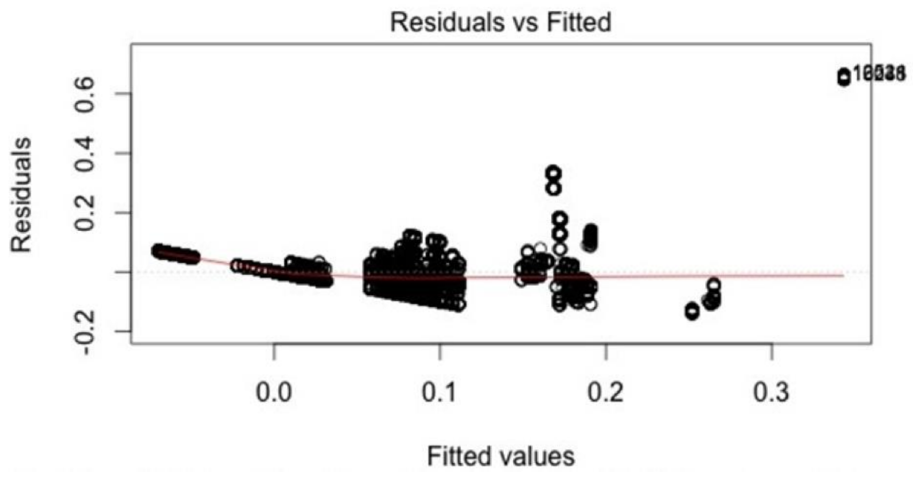
On observing both the graphs, we noticed that there is a unique pattern of ‘Number of Days vs Size of Backup’ followed for each of the 5 workflows in such a way that it repeats itself after a span of roughly 7 days.

<u>Problem 2:</u>

a. <u>Linear Regression Model:</u>

We ran a linear regression model on the dataset to identify which all parameters are important, and we found that ‘Day of the week’, ‘Backup start time-hour of the day’ and ‘Backup Time’ are important. So for all the further models we used these as the dependent variables or the explanatory variables.

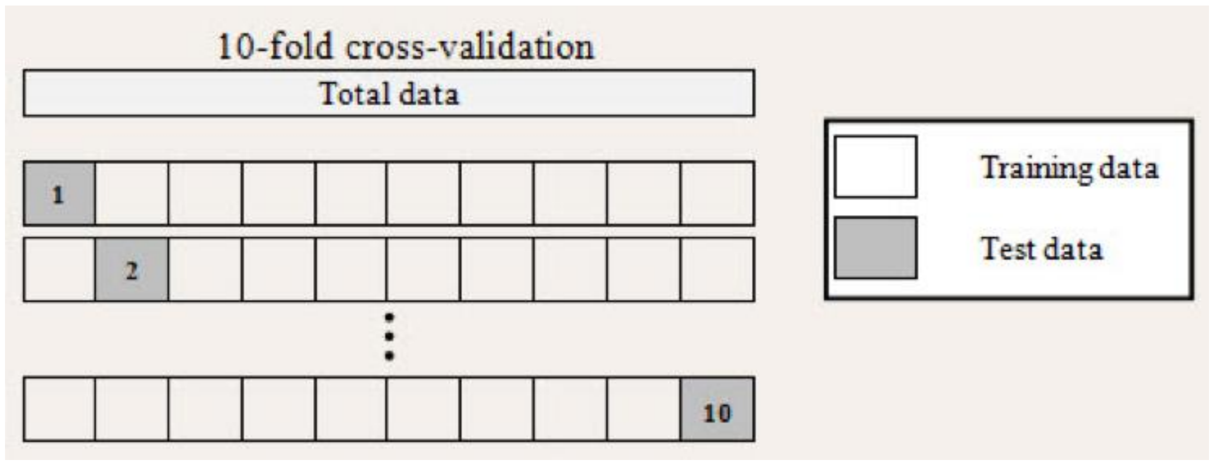
Now we trained a model using Ordinary Least Squares as the penalty function and obtained the following results-



```
Coefficients:
                                Estimate Std. Error t value Pr(>|t|)    
(Intercept)                    -7.07e-02   1.88e-03  -37.58   <2e-16 ***
Day.of.WeekMonday               8.22e-02   2.06e-03   39.91   <2e-16 ***
Day.of.WeekSaturday             7.03e-02   2.08e-03   33.79   <2e-16 ***
Day.of.WeekSunday               7.13e-02   2.08e-03   34.19   <2e-16 ***
Day.of.WeekThursday             4.77e-02   2.06e-03   23.18   <2e-16 ***
Day.of.WeekTuesday              1.89e-03   2.06e-03    0.92     0.36    
Day.of.WeekWednesday            5.51e-02   2.10e-03   26.30   <2e-16 ***
Backup.Start.Time...Hour.of.Day 9.33e-04   7.96e-05   11.72   <2e-16 ***
Backup.Time..hour.              8.00e-02   6.16e-04  129.76   <2e-16 ***
```

RMSE Value: 0.7404

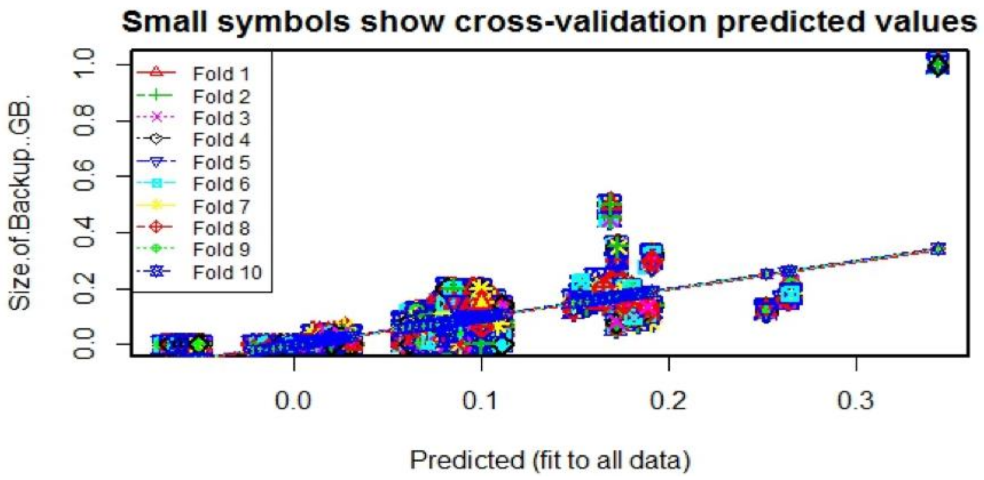
Next we moved on to perform Cross Validation on the data. We used 10-fold cross validation. This means that the data would be split into 10 equal parts. For every iteration, 90% of the data would be used for training and the rest 10% would be used for testing the data. The training and testing data would change for every iteration.



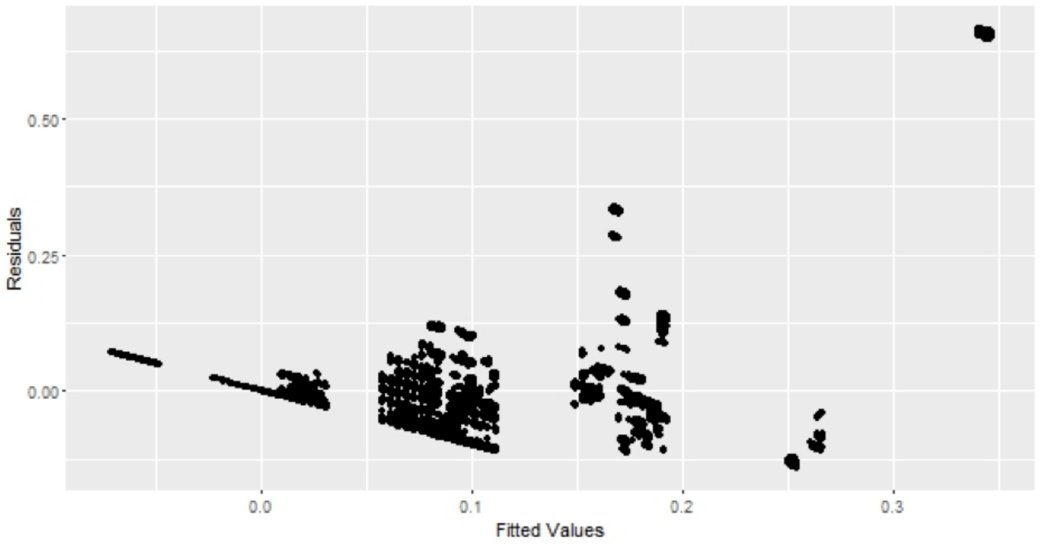
To sample the data, we used to sample function. But before doing this, we randomized the data in jumbled order so that the samples we take would represent every kind of feature and observation equally.

We got the following results after cross-validation:

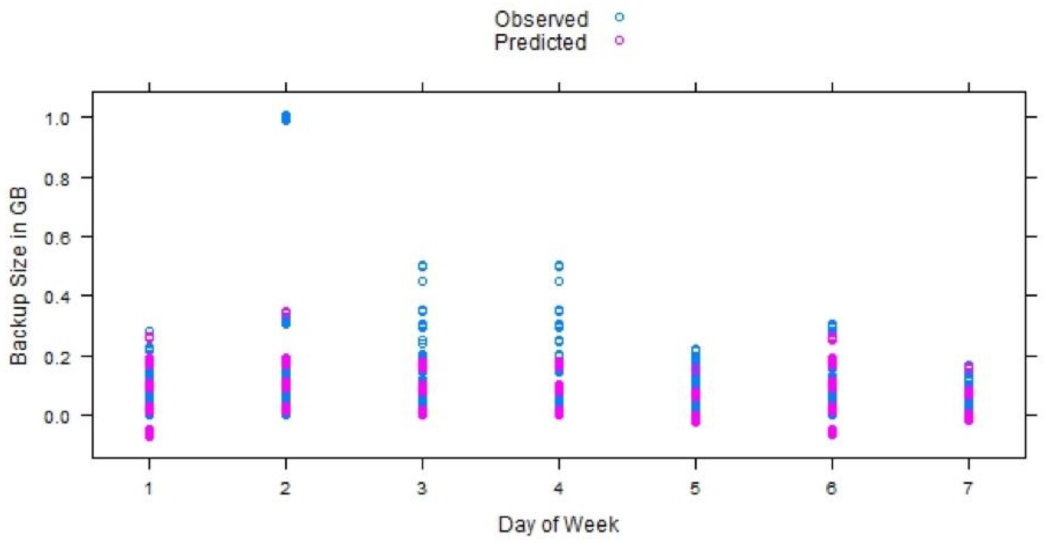
The **RMSE (0.7409)** that we obtained was similar to the RMSE that we got without performing cross-validation.
We also tried cross validating using the in-built function of cv.lm() in 'DAAG' package of R. The **RMSE** that we obtained using this was **0.7407**.

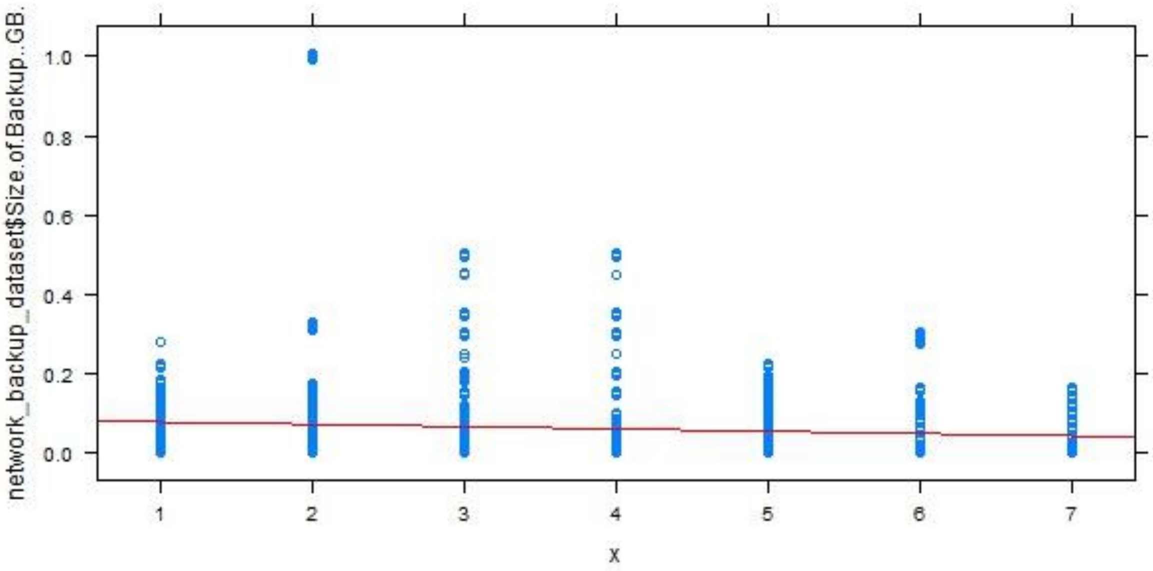


The graph below is of the residuals vs Fitted values for the model:



The two plots below are those of observed and predicted values. The red line in the second plot is the fitted linear regression for the data.





b. <u>Random Forests:</u>

To train our random forest regression model we used the randomForest() function in the ‘randomForest’ package of R. The function has different parameters which we can use to tune the model and optimize the output. The parameters that we used for tuning are the ‘No. of Trees’ and ‘Node Size’. As the name suggests, ‘No of Trees’ implies the maximum number of trees that the model will train. The parameter ‘Node Size’ represents the maximum number of observations that can be fit at a particular node. This parameter indirectly determines the depth of the tree and is inversely proportional to it. So as the node size decreases, the depth of the tree increases.

We trained around 300 models by incrementing the number of trees in every iteration and decreasing the node size thus increasing the depth of the tree. Following are our results:

Sr. No	**RMSE**	**Node Size**	**No. of Trees**
1	0.09013327	4567	24
2	0.08345971	4407	32
3	0.08522949	4227	41
4	0.08094212	4047	50
5	0.07996574	3867	59
6	0.07814296	3687	68
7	0.07902206	3487	78
8	0.08002904	3307	87
9	0.07777699	3147	95
10	0.0756081	2887	108
11	0.0741249	2667	119
12	0.07071039	2447	130
13	0.06807292	2287	138
14	0.06784398	2107	147
15	0.07121831	1887	158
16	0.06670809	1727	166
17	0.06486669	1507	177
18	0.06076341	1247	190
19	0.06020055	1067	199
20	0.05645044	847	210
21	0.05291683	607	222

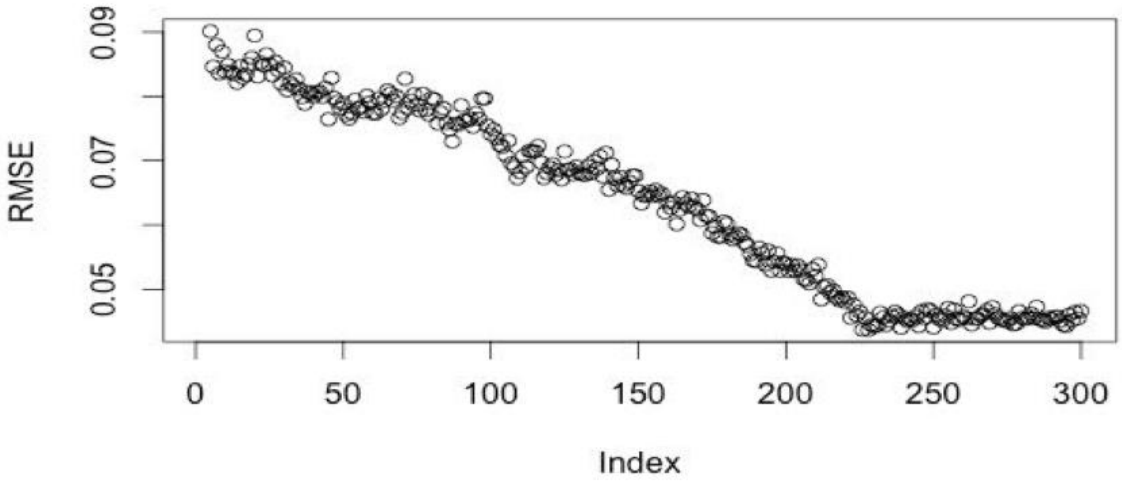
22	0.05315402	487	228
23	0.04879361	327	236
24	0.04653843	167	244
25*	***0.0437***	***147***	***245***
26	0.04558845	127	246
27	0.04445245	87	248
28	0.04435979	47	250
29	0.04636911	27	251
39	0.04564339	7	252

We observe from the above table that beyond a certain point the RMSE becomes constant after a certain point. The best **RMSE** of **0.0437** was obtained when No of trees were 245 and the node size was 147.

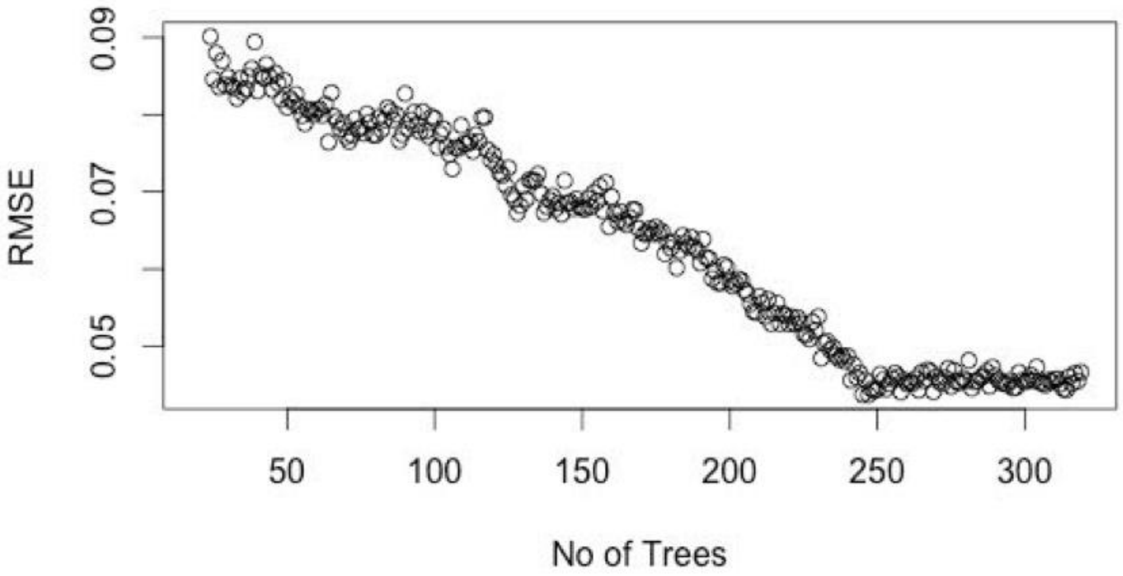
Comparison with Linear Regression:

As compared to Linear Regression, Random Forest gave a lot better performance and an RMSE almost 0.03 lower than it.

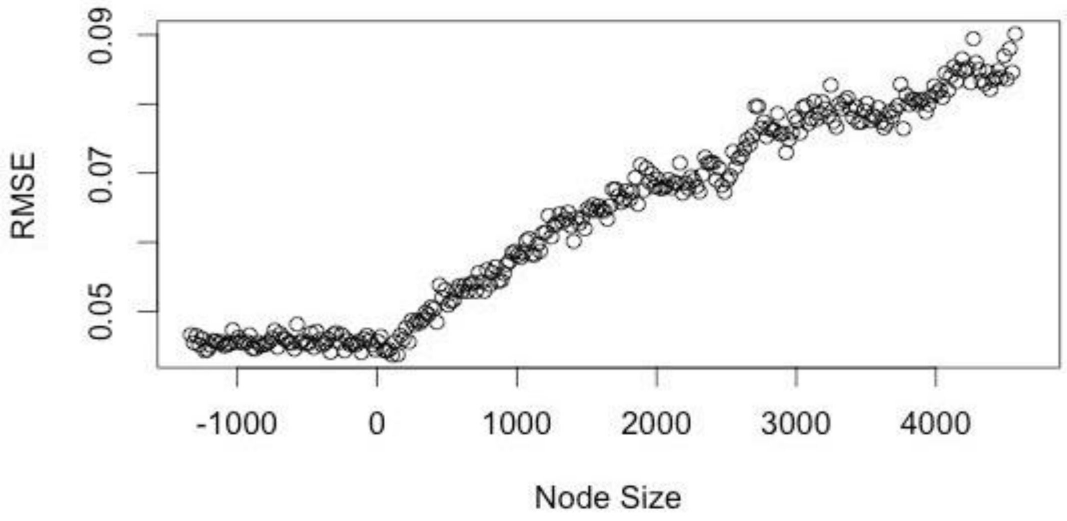
The graph below shows the decreasing RMSE values over the 300 random forest models that we trained. After a certain point the RMSE doesn't seem to decrease any further and remains constant.



The above fact is confirmed with the plot below. As the number of trees increase, the RMSE decreases.



The graph for RMSE Values v/s the Node Size is inversely proportional to the above plot. As the node size increases, the RMSE increases which also means that the depth of the tree increases.



The random forest model seems to fit the data pretty well and this has resulted in a low RMSE value. The regression curve is much smoother and better.

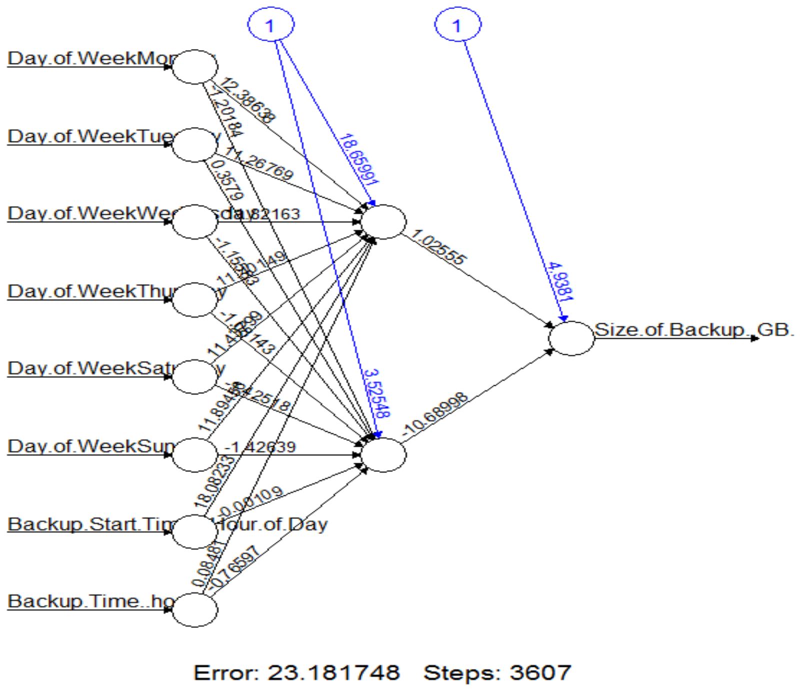
c. <u>Neural Networks:</u>

Further, we used the neural network regression model in order to fit the data that has been provided to us. In doing so, we used the 'neuralnet' package that is available in R. Moreover, we passed the following parameters to the 'neuralnet' function that was being used to fit the model:

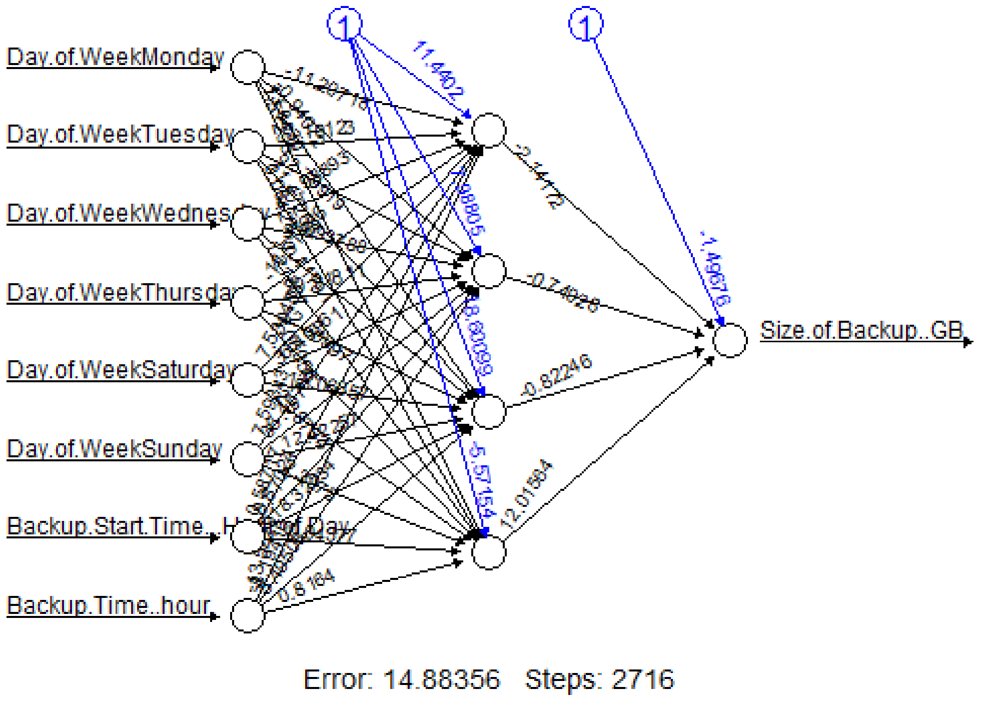
- <u>Function:</u> We used the 'Size of Backup' as the target variable, whereas 'Day of the week', 'Backup start time' and 'Backup time' as the features. However, 'Day of the Week' being categorical data, we converted it into its equivalent quantitative value prior to the fitting
- <u>Number of hidden neurons and layers:</u> We tested our neural network model by varying the number of hidden neurons and layers, as discussed furthermore in this section.
- <u>Threshold:</u> We used a threshold of 0.1 as a stopping criteria for the partial derivatives of the error function.

Out of the above three major parameters, we noticed that changing the values of the number of hidden neurons that were to be present in each hidden layer resulted into the most amount of change in the RMSE values.

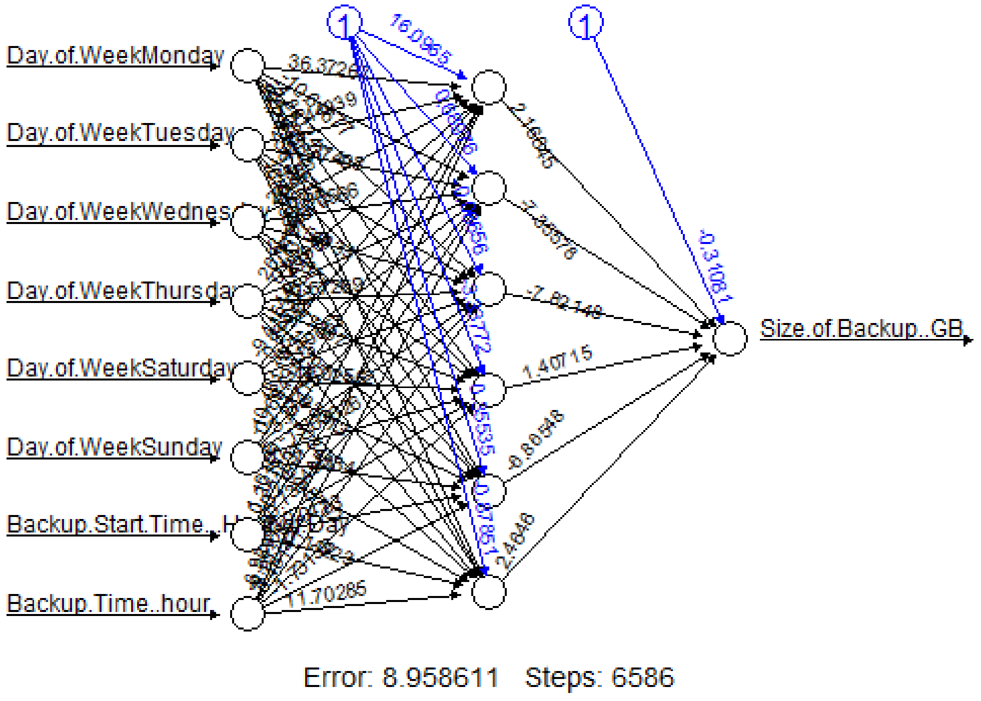
We first started off our experiment considering only one hidden layer consisting of two hidden neurons. This setting resulted into an **RMSE** of **0.0499**.



We then decided to further increase the number of hidden neurons to 4 neurons, keeping the number of layers constant i.e. 1. It was found that the **RMSE** value dropped to **0.400**, which indicated some improvement in our model.



On increasing the number of hidden neurons furthermore to a total of 6 neurons per hidden layer, there was an improvement in the performance yet again giving us an **RMSE** of **0.0313**.



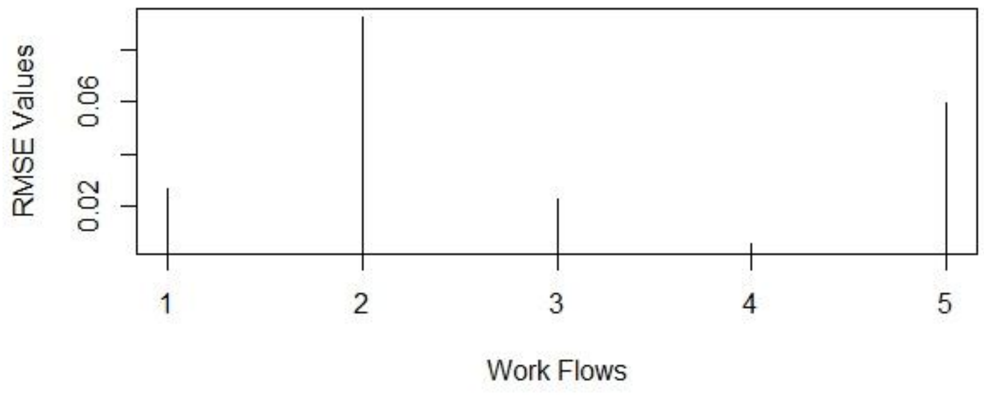
According to the rule-of thumb, the number of hidden neurons in each hidden layer should be approximately two-thirds of the input layer units plus the output layer units in order to achieve the most optimal fit. The results that were obtained by us exactly complied to this rule, whereby having 6 hidden neurons in the hidden layer gave us the lowest **RMSE** i.e. **0.0313**.

It is also worth noting that increasing the number of hidden layers did not affect the performance in terms of the RMSE by a huge factor. As a result, we decided to use only a single hidden layer for all our experiments on the neural network model.

Problem 3:

Piece Wise Linear Regression:

Piece Wise Linear Regression is useful when the data is not continuous. For discrete or non-continuous data, piecewise often fits better than the normal linear regression. For our dataset we performed piecewise linear regression on the basis of Workflow ID's. In our dataset, we have 5 unique WorkFlowID's ranging from work_flow_0 to work_flow_4. Since the data set is pretty discreet in many senses, piece-wise linear regression actually helped it fit better.
We obtained an **RMSE** value of **0.0411** which was even better as compared to Random Forest where the best **RMSE** that we obtained was around **0.043**. A plot of the different RMSE's for the different work-flows is as below-



The **RMSE** of **0.0411** is the mean of all the 5 RMSE's shown in the graph above. So we can conclude that the fit definitely improved.

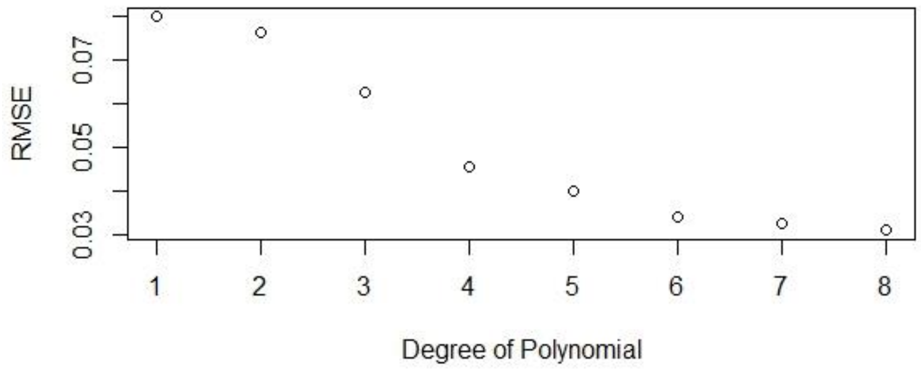
Polynomial Regression:

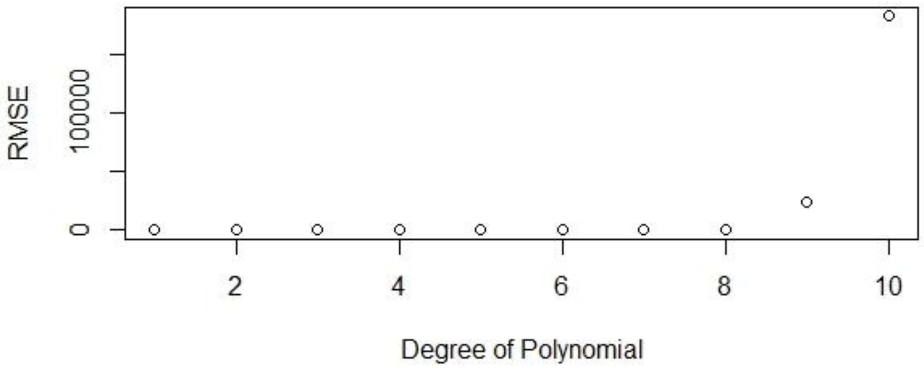
All the regression models that we had fit till now were linear in nature. But we can increase the degree of each dependent variable and try fitting the model with different degrees of polynomials.

We used the 'mpoly' function in R to execute this. This function uses binomial expansion of the variables. We started from degree 1 and incremented it till degree 10. A graphical result of our implementation is given below. As the degree went on increasing the RMSE reduced till a certain level. Beyond that certain point the RMSE increased and shot to extremely high values. The **threshold** for us was around **Degree = 5**.

Degree	1	2	3	4	5	6	7	8	9	10
RMSE	0.0079	0.0763	0.0626	0.0456	0.0399	0.0342	0.0326	0.0311	2.92e+05	1.54e+04

Since the values shot very high for Degree = 9 and Degree = 10, we have plotted two different graphs. The first graph only shows the RMSE for Degree 1 to 8 and the next graph shows all the 10 RMSE's.





Benefits of Cross Validation:

Cross Validation certainly helps fit the data better because it leaves some data every time and then tests that model on the other data. This prevents the model from overfitting and reduces the RMSE for the testing data too. So Cross validation helps reduce the complexity of the model by preventing overfitting.

Boston Housing Dataset:

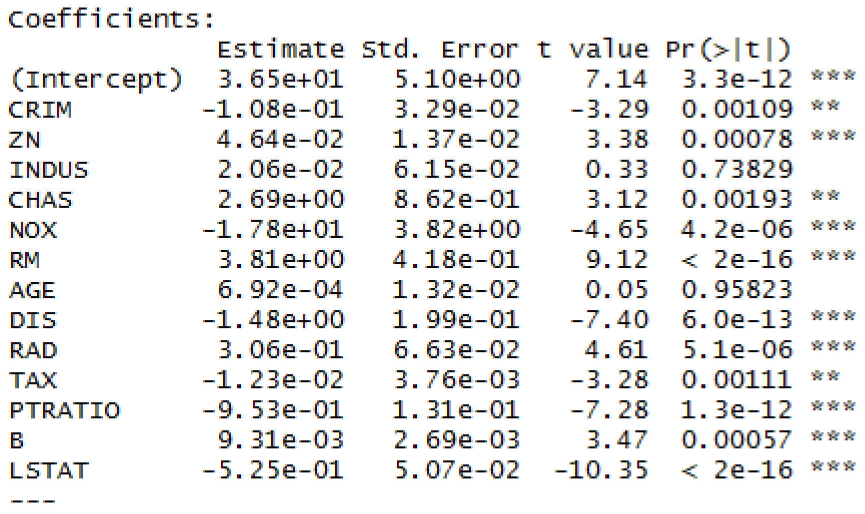
This dataset concerns housing values in the suburbs of the greater Boston area and is taken from the StatLib library which is maintained at Carnegie Mellon University. There are around 500 data points with the following features:

• CRIM: per capita crime rate by town
• ZN: proportion of residential land zoned for lots over 25,000 sq. ft.
• INDUS: proportion of non-retail business acres per town
• CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
• NOX: nitric oxides concentration (parts per 10 million)
• RM: average number of rooms per dwelling
• AGE: proportion of owner-occupied units built prior to 1940
• DIS: weighted distances to five Boston employment centers
• RAD: index of accessibility to radial highways
• TAX: full-value property-tax rate per $10,000
• PTRATIO: pupil-teacher ratio by town
• B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
• LSTAT: % lower status of the population
• MEDV: Median value of owner-occupied homes in $1000's

Problem 4:

Ordinary Least Squares Regression:

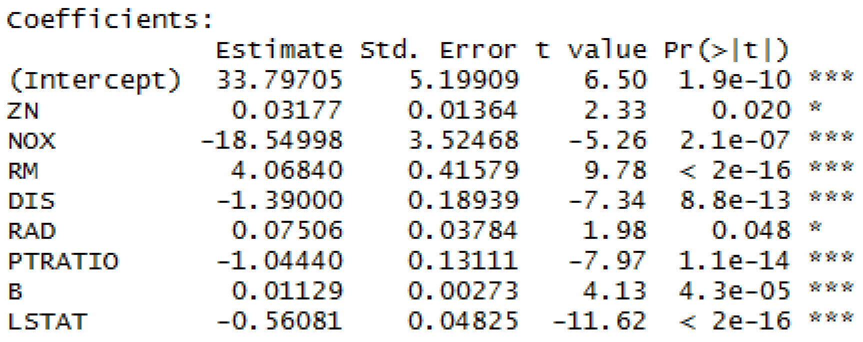
Initially to start off, we trained a linear regression model with all the variables except MEDV as the dependent variables. This mainly done to understand the important variables. Below is the summary of the model obtained. We took the parameters with *** for the next model as they turned out to be the significant variables.



```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.65e+01   5.10e+00    7.14  3.3e-12 ***
CRIM        -1.08e-01   3.29e-02   -3.29  0.00109 ** 
ZN           4.64e-02   1.37e-02    3.38  0.00078 ***
INDUS        2.06e-02   6.15e-02    0.33  0.73829    
CHAS         2.69e+00   8.62e-01    3.12  0.00193 ** 
NOX         -1.78e+01   3.82e+00   -4.65  4.2e-06 ***
RM           3.81e+00   4.18e-01    9.12  < 2e-16 ***
AGE          6.92e-04   1.32e-02    0.05  0.95823    
DIS         -1.48e+00   1.99e-01   -7.40  6.0e-13 ***
RAD          3.06e-01   6.63e-02    4.61  5.1e-06 ***
TAX         -1.23e-02   3.76e-03   -3.28  0.00111 ** 
PTRATIO     -9.53e-01   1.31e-01   -7.28  1.3e-12 ***
B            9.31e-03   2.69e-03    3.47  0.00057 ***
LSTAT       -5.25e-01   5.07e-02  -10.35  < 2e-16 ***
---
```

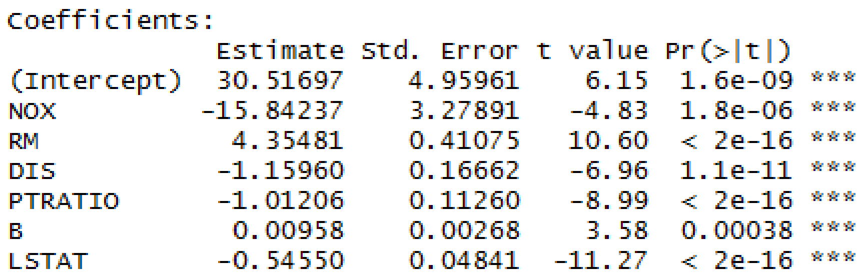
After this we trained our linear regression model with CRIM, ZN, NOX, RM, DIS, RAD, PTRATIO, B and LSTAT as the predictor variables. The summary of this model is shown below. We again filtered out more variables from this and found out that only these six variables as the most significant ones:

NOX, RM, DIS, PTRATIO, B and LSTAT.



```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  33.79705    5.19909    6.50  1.9e-10 ***
ZN            0.03177    0.01364    2.33    0.020 *  
NOX         -18.54998    3.52468   -5.26  2.1e-07 ***
RM            4.06840    0.41579    9.78  < 2e-16 ***
DIS          -1.39000    0.18939   -7.34  8.8e-13 ***
RAD           0.07506    0.03784    1.98    0.048 *  
PTRATIO      -1.04440    0.13111   -7.97  1.1e-14 ***
B             0.01129    0.00273    4.13  4.3e-05 ***
LSTAT        -0.56081    0.04825  -11.62  < 2e-16 ***
```

After finalizing the predictor variables we trained our final model with list mentioned above as the predictor variables. The RMSE that we obtained for the model was 4.901. A summary of the model is shown below-



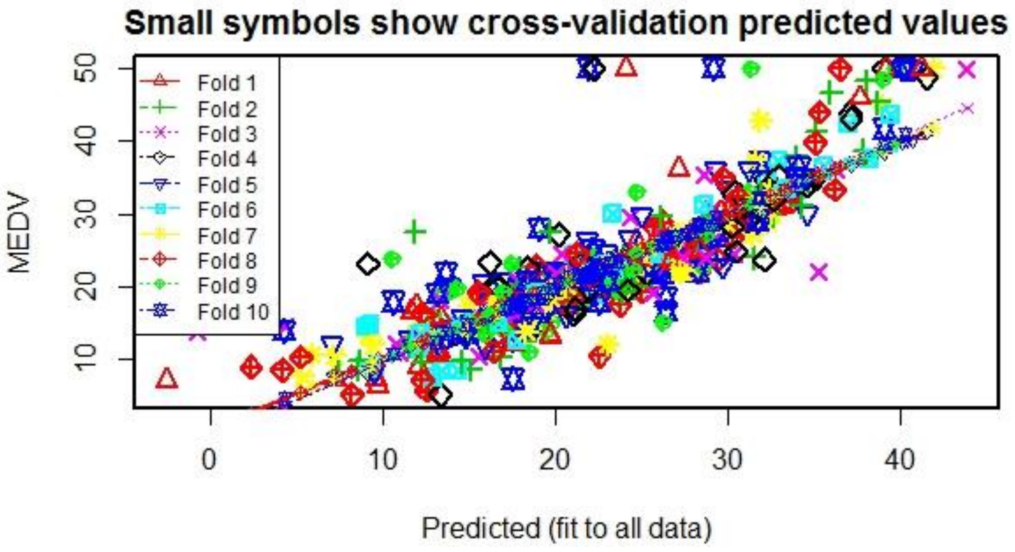
```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  30.51697    4.95961    6.15  1.6e-09 ***
NOX         -15.84237    3.27891   -4.83  1.8e-06 ***
RM            4.35481    0.41075   10.60  < 2e-16 ***
DIS          -1.15960    0.16662   -6.96  1.1e-11 ***
PTRATIO      -1.01206    0.11260   -8.99  < 2e-16 ***
B             0.00958    0.00268    3.58  0.00038 ***
LSTAT        -0.54550    0.04841  -11.27  < 2e-16 ***
```

Variance Inflation Factors: Since all the VIF's are below 4, the explanatory variables chosen would be ideal.

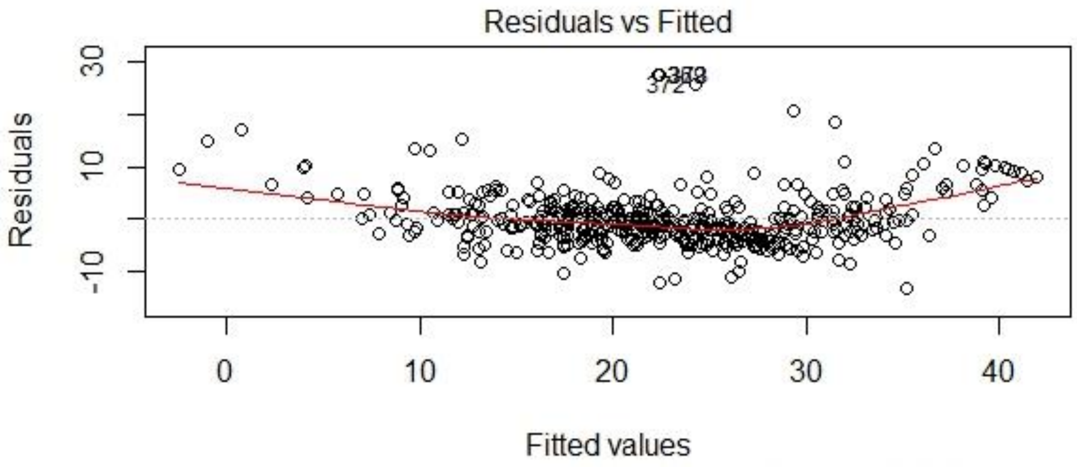
NOX	RM	DIS	PTRATIO	B	LSTAT
2.99	1.73	2.55	1.23	1.24	2.48

Now we tried implementing 10-fold cross validation on the data set.

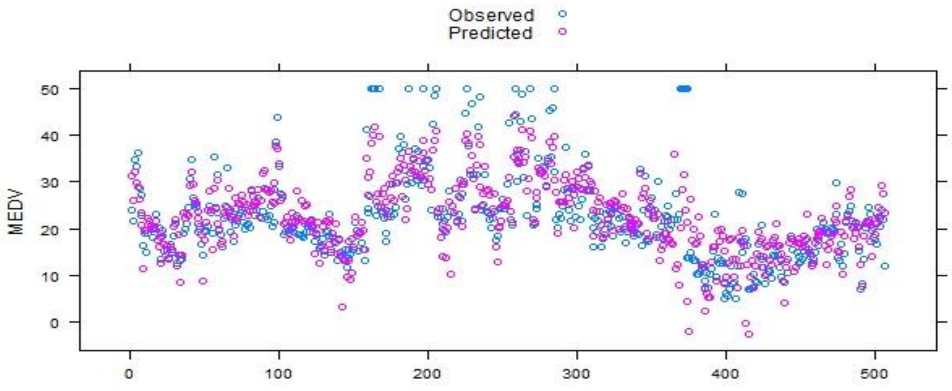
The **RMSE** value that we obtained using Cross Validation on the data-set was around **5.01**. This CV was done using the function coded by us. We also decided to experiment with the inbuilt cv.lm() function of 'DAAG' package. The **RMSE** that we got was **5.02**. Below is the plot for the same-



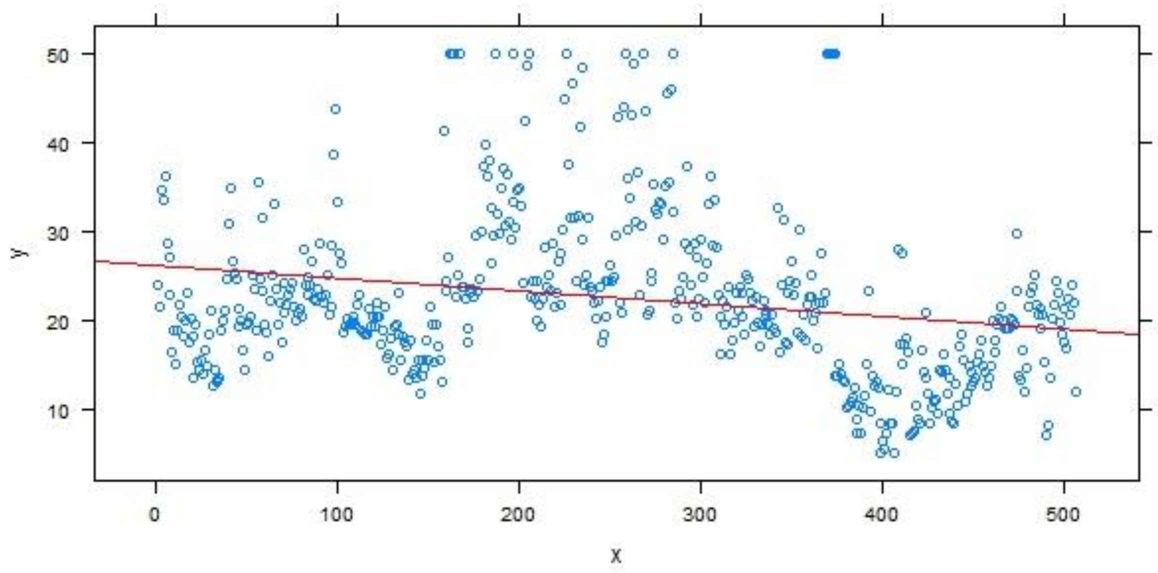
The Residuals v/s Fitted Values plot is as shown below:



The Observed Values v/s Fitted Values plot is as shown below:

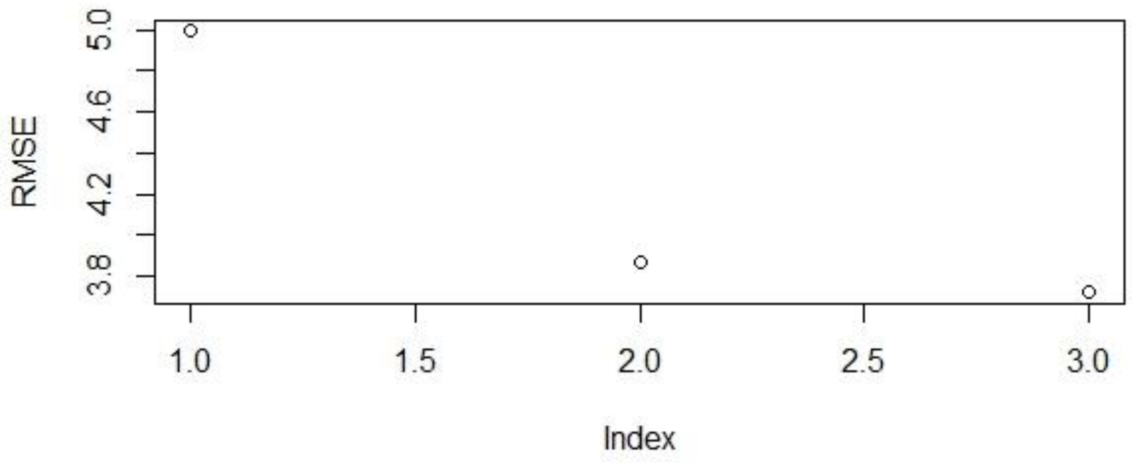


The plot below shows the linear regression line ('red') for the Boston housing dataset. The blue scattered points are the observed data points.



Polynomial Regression:

As was the case with problem 3 of the project, we did a polynomial regression on the data set with cross validation. We started with Degree=1 and trained different models till Degree=10. As expected, the RMSE went down as the degree of the polynomial expression increased, but then started shooting up drastically. The graph below shows RMSE only for degrees 1,2 and 3. For Degree = 3 we observed our lowest RMSE and then it started shooting up.



The plot below is for all the degrees. Please make note of the scale of both the plots. The error shoots up to a very high value for a Degree = 10. We got the lowest **RMSE** of **3.74** at **Degree = 3**.

Degree	1	2	3	4	5	6	7	8	9
RMSE	5.03	3.968	3.74	20.8	1.531e+05	6.561e+04	7.220e+04	4.552e+05	2.92e+08

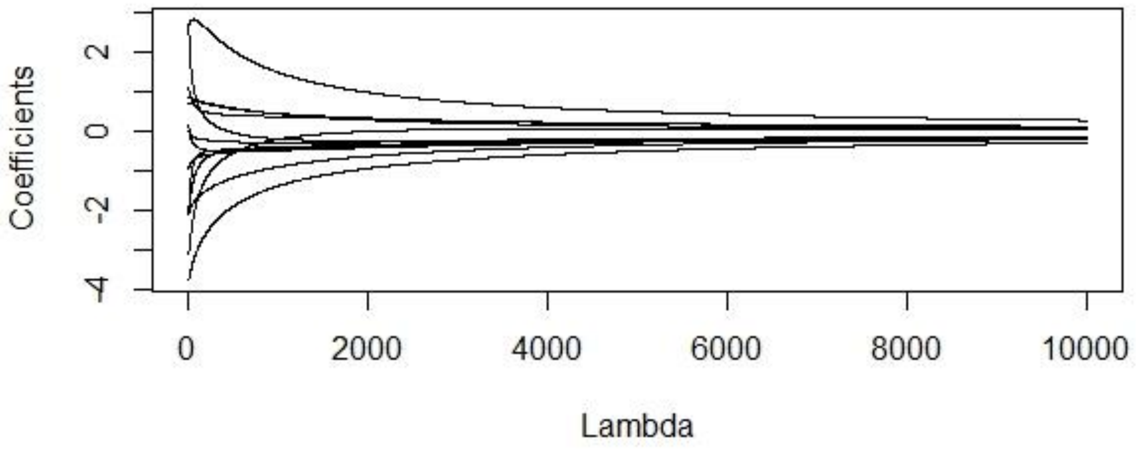
Problem 5:

Regularization:
Regularization is used to prevent overfitting of models by imposing a penalty on the fit. We implemented two techniques for this.

Ridge Regression:
We implemented Ridge Regression with ‘glmnet’ function in the package ‘glmnet’ in R. We tuned the parameter alpha but no considerable change was observed in the value of RMSE. This means that the penalty that we were imposing on the model did not have much effect. The model wasn’t really over fitting the data.

Alpha Value	**RMSE**
0.1	4.991
0.01	4.99
0.001	5.01

We also implemented Ridge with ‘lm.ridge’ function in the ‘MASS’ package of R. We varied the value of lambda over a large range.



LASSO Regression:

Similarly we implemented LASSO Regression with the lars() function in the “lars” package of R. Both the regularizations were done with cross validation. The **RMSE** we got with LASSO Regression was about **4.987**.