MAJOR PROJECT - I

Ensemble Learning for Regression Applications

Gaurav Mishra (1510110480)
Karan Praharaj (1510110183)

Under the supervision of
Dr. Madan Gopal

SHIV NADAR UNIVERSITY

DEPARTMENT OF ELECTRICAL ENGINEERING

December 2018

Candidate Declaration

We hereby declare that the thesis titled "Ensemble Learning in Regression Applications" submitted towards the partial fulfilment of the requirement for the degree of Bachelor of Technology is a presentation of our original work. Wherever contributions of others and external sources are involved, every effort has been made to indicate this clearly, with due reference to the literature, and acknowledgement of the research and discussions.

The work was done under the guidance of Professor Madan Gopal, at Shiv Nadar University, Greater Noida.

Date : 3rd December, 2018

Gaurav Mishra

Karan Praharaj

Abstract

This project aims to assess the performance and usability of ensemble regression techniques against the performance of other machine learning algorithms by comparing the results using test mean squared error as the metric for comparison. The goal of ensemble regression is to combine several models in order to improve the prediction accuracy on learning problems with a numerical target variable. Many methods for constructing ensembles have been developed. The method used in this project is constructing ensembles that manipulate the training examples to generate multiple hypotheses. For experimental evaluation, we have used a house value prediction problem and a stock index prediction problem. Specifically, the bagging, boosting and random forests ensemble methods were used in addition to the standard machine learning. In terms of test accuracy, it was found that Boosting (AdaBoost) and Bagging perform the best of all the algorithms for the housing problem. We expect that the learning attained from these results will provide a useful platform for our further work on the Stock Index Prediction problem.

INDEX

1. Introduction

Ensemble learning is a process that uses a set of models, each of them obtained by applying a learning process to a given problem. This set of models (ensemble) is integrated to obtain the final prediction.

The process of ensemble learning for regression can be divided into three phases: the generation phase, in which a set of candidate models is induced, the pruning phase, to select a subset of those models and the integration phase, in which the output of the models is combined to generate a prediction. This approach has been the object of a significant amount of research in recent years and good results have been reported. The advantage of ensembles with respect to single models has been reported in terms of increased robustness and accuracy.

A large part of the work done on ensemble learning focuses on classification problems. However, methods that are successful for classification have often been found to be not directly applicable for regression. Thus, although both are related, ensemble learning approaches have been developed somehow independently. Therefore, existing surveys on ensemble methods for classification are not suitable to provide an overview of existing approaches for regression.

This project tests the value in ensemble learning for regression by means of two case studies. What makes the topic of this project even more relevant is the fact that ensemble learning is a widespread focus of research in various communities, including pattern recognition, machine learning, statistics and neural networks.

The case studies that have been picked for this project are the Boston Housing Dataset and the S&P500 Index data from January 1980 to December 1992. ¶

2. Prediction of Median House Values

2.1 Analysis of the Original Data - Boston Housing Dataset

The data used for this project was obtained from the StatLib archive (http://lib.stat.cmu.edu/datasets/boston), and has been used extensively throughout the literature to benchmark algorithms.

2.1.1 Data Description

The Boston Housing data was collected in 1978 by the U.S Census Service concerning housing in the area of Boston, Massachusetts. It was obtained from the StatLib archive (http://lib.stat.cmu.edu/datasets/boston), and has been used extensively throughout the literature to benchmark algorithms.

It has median value of the house along with 13 other parameters that could potentially be related to housing prices. These factors are socio-economic conditions, environmental conditions, educational facilities and some other similar aspects. There are 506 observations in the data for 14 variables including the median price of house in Boston. There are 12 numerical variables in our dataset and 1 categorical variable. The 506 entries represent aggregated data for homes from various suburbs in Boston, Massachusetts. The data was originally published by Harrison, D. and Rubinfeld, D.L. '*Hedonic prices and the demand for clean air'*, J. Environ. Economics & Management, vol.5, 81-102, 1978.

Output Information:

The last column of 'MEDV' denotes median value of owner-occupied homes in the neighbourhood (in thousands of dollars). The variable CAT.MEDV has been formed and is a categorical data. If this is chosen as output variable our problem becomes a classification problem. It is created by simply assigning 0 and 1 to the variable. 0 if MEDV is less than 30 and 1 otherwise. We can use this as a response variable if our job is to classify houses as high or low median value. Since our aim is regression, we shall leave this variable out of further analyses.

The description of the data variables is as follows:

(There are no missing data points but the odd variable names are sometimes confusing so we are going to condense them to representative variable aliases for ease of interpretation.)

Attribute	Representative Variable	Attribute Type	Description
CRIM	x_1	Continuous	per capita crime rate by town
ZN	x_2	Continuous	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	x_3	Continuous	proportion of non-retail business acres per town.
CHAS	x_4	Ordinal	Charles River dummy variable (1 if tract bounds river; 0 otherwise)
NOX	x_5	Continuous	nitric oxides concentration (parts per 10 million)
RM	x_6	Continuous	average number of rooms per dwelling
AGE	x_7	Continuous	proportion of owner-occupied units built prior to 1940
DIS	x_8	Continuous	weighted distances to five Boston employment centers
RAD	x_9	Ordinal	index of accessibility to radial highways
TAX	x_{10}	Continuous	full-value property-tax rate per $10,000
PTRATIO	x_{11}	Continuous	pupil-teacher ratio by town
B	x_{12}	Continuous	$B\text{-}1000(Bk - 0.63)^2$ - where Bk is the proportion of blacks by town
LSTAT	x_{13}	Continuous	% lower status of the population
MEDV	y	Continuous	Median value of owner-occupied homes in $1000's

Table 1 : Feature Description

Number of missing values = zero

The target variable for this data-set is the MEDV (median-value) of houses within a given suburb. After presenting some summary statistics for the MEDV feature we will make some plots to get a sense of the shape of the data.

Statistics	Value
Number of Houses	506
Number of Features	13

Min Price	$5,000
Max Price	$50,000
Mean Price	$22,533
Median Price	$21,200
Standard Deviaton	$9,188

Table 2 : Summary statistics - x_{14}

Feature MEDV seems to be capped at 50.00 (corresponding to a median price of $50,000); Capping is suggested by the fact that the highest median price of exactly $50,000 is reported in 16 cases, while 15 cases have prices between $40,000 and $50,000, with prices rounded to the nearest hundred. Harrison and Rubinfeld do not mention any capping either and therefore we conclude that this is an artificial upper limit of the housing price.

2.1.2 Feature summary - Boston Housing Dataset

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	506	506	506	506	506	506
mean	3.593761	11.363636	11.136779	0.06917	0.554695	6.284634
std dev	8.596783	23.322453	6.860353	0.253994	0.115878	0.702617
min	0.00632	0	0.46	0	0.385	3.561
median	0.25651	0	9.69	0	0.538	6.2085
max	88.9762	100	27.74	1	0.871	8.78

	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
count	506	506	506	506	506	506	506
mean	68.574	3.795	9.549	408.237	18.455	356.674	12.653
std dev	28.148	2.105	8.707	168.537	2.164	91.294	7.141
min	2.9	1.1296	1	187	12.6	0.32	1.73
Median	77.5	3.20745	5	330	19.05	391.44	11.36
max	100	12.1265	24	711	22	396.9	37.97

Table 3,4 : Feature Summary

Statistical quantities such as mean, standard deviation, min, max and median help us infer important characteristics of each predictor. Mean and median show how much skewed our data is for example, for y it can be inferred that it is right skewed as median<mean. min and max help us determine if any error has crept in as range might change if any inadvertent data entry errors have occurred. Count shows us number of missing data points. Standard deviation tells how dispersed data is with respect to mean.

Upon analysis of MEDV, it was found that 4.5% of the data points were outliers (points for which MEDV>37) and the upper third of the points come from a different distribution than the lower two-thirds. Since it is usual to have outliers in the real estate market, we choose to keep these data points in the further analyses as removing them will also result in losing prediction ability of models for tracts with median price above $37,000.

Data Division

Training set and testing set have been formed by randomly splitting the dataset. Training set comprises of 70% of the total data points i.e. 354 data points and testing set comprises of 30% of the total data point i.e. 152 data points. To tune the hyper-parameters, 10-fold cross validation on the training data has been used.

Performance Metric

Throughout the report, Mean Squared Error (MSE) has been used as a performance metric. It is defined as:

Let there be N data points, then

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ is the predicted value and y_i is the actual response.

To tune the hyper-parameters, 10-fold cross validation on the training data has been used and to specify performance of an algorithm, Test MSE has been used. The Test Data set has not been involved in any part of the training or validation.

2.1.4 Feature Selection

Considering the ratio of number of predictors (13) to the number of training examples (506), our intuition was that dropping any feature altogether would not be required. Besides there being a relatively low number of predictors, we see that the information provided by each feature may be important for the user on its own.

However, we decided to inspect the covariance matrix first to examine the possibility of feature reduction. (refer Table 3)

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	y
x1	73.987													
x2	-39.503	528.596												
x3	7.662	-26.980	7.617											
x4	-0.122	-0.239	0.014	0.065										
x5	0.420	-1.368	0.182	0.003	0.013									
x6	-1.325	5.007	-0.547	0.016	-0.025	0.494								
x7	85.405	-368.342	34.720	0.619	2.386	-4.752	792.358							
x8	-6.525	25.511	-2.176	-0.046	-0.174	0.316	-40.152	3.999						
x9	-0.964	-4.075	0.580	0.028	0.002	0.088	0.784	0.009	2.008					
x10	844.822	-1206.308	224.358	-1.523	13.046	-34.583	2402.690	-182.453	-11.756	28404.759				
x11	5.399	-19.112	1.395	-0.067	0.047	-0.541	15.937	-1.165	-0.359	168.153	4.687			
x12	-302.382	366.548	-75.109	1.131	-4.021	8.215	-702.940	53.604	5.266	-6797.911	-35.060	8334.752		
x13	27.986	-67.060	7.576	-0.098	0.489	-3.080	121.078	-6.842	-0.707	654.715	5.783	-238.668	50.995	
y	-30.719	75.520	-8.740	0.409	-0.455	4.493	-97.589	4.861	1.480	-726.256	-10.111	279.990	-48.448	84.587

Matlab: cov()

Table 3 : Covariance matrix

If we choose predictors x10 and x12, the covariance matrix for these two would be

$$S = \begin{bmatrix} 28404.759 & -6797.911 \\ -6797.911 & 8334.752 \end{bmatrix}$$

Finding correlation between the two, $\frac{-6797.911}{\sqrt{28404.759}\sqrt{8334.752}} = -0.44$ (for more correlations refer to Table 4). This means that 44% of total variation in both the predictors is actually the variation in one predictor which is duplicated by similar variation in the other predictor. This means that there is some redundancy in the information conveyed by these two predictors. The information here is in the variability.

This redundancy can be avoided by PCA. The idea of PCA is to find a linear combination of these two predictors which contains most of the information conveyed by these two original predictors. In other words, PCA distributes the variance explained by each predictors into the new predictor (which is a linear combination of the two). Taking these two variables (x10 and x12), the total variability amongst the 506 houses is the sum of variances of these two variables, which is 28404.759+8334.752 = 36739.511. This means that 77% of the total variability is due to x10 (=28404/36739.511) and the other 33% due to x12. PCA finds another variable which can explain most, if not all, of the variability conveyed by these two predictors together.

Using all the predictors, it was found that although PCA explained most of the variance using less predictors, the overall generalisation of the new model, with reduced dimensionality, was not good. The improvement in test error was not very significant. On the other hand, all the predictors practically play an important role in deciding the median price. This is to say that a buyer or seller will be interested in knowing how much each predictor weighs while deciding the price.

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	y
x1														
x2	-0.20													
x3	0.32	-0.43												
x4	-0.06	-0.04	0.02											
x5	0.42	-0.51	0.57	0.09										
x6	-0.22	0.31	-0.28	0.09	-0.30									
x7	0.35	-0.57	0.45	0.09	0.73	-0.24								
x8	-0.38	0.55	-0.39	-0.09	-0.75	0.23	-0.71							
x9	-0.08	-0.13	0.15	0.08	0.01	0.09	0.02	0.00						
x10	0.58	-0.31	0.48	-0.04	0.67	-0.29	0.51	-0.54	-0.05					
x11	0.29	-0.38	0.23	-0.12	0.19	-0.36	0.26	-0.27	-0.12	0.46				
x12	-0.39	0.17	-0.30	0.05	-0.38	0.13	-0.27	0.29	0.04	-0.44	-0.18			
x13	0.46	-0.41	0.38	-0.05	0.59	-0.61	0.60	-0.48	-0.07	0.54	0.37	-0.37		
y	-0.39	0.36	-0.34	0.18	-0.43	0.70	-0.38	0.26	0.11	-0.47	-0.51	0.33	-0.74	

Matlab: corr()

Table 4 : Correlation matrix

2.2 Algorithms

2.2.1 Linear Regression

(a) Standard Linear Regression

The figure below shows the graph of Root Mean Square Error vs Number of Epochs (or Iteration) for different values of η (learning rate). In the curve four different values of η, marked by different colours, have been tried (η = 0.001, 0.01, 0.1, 1). It can be seen that the optimisation converges early at around the 40th iteration for η =0.1. We'll choose η as 0.1 for further analyses.

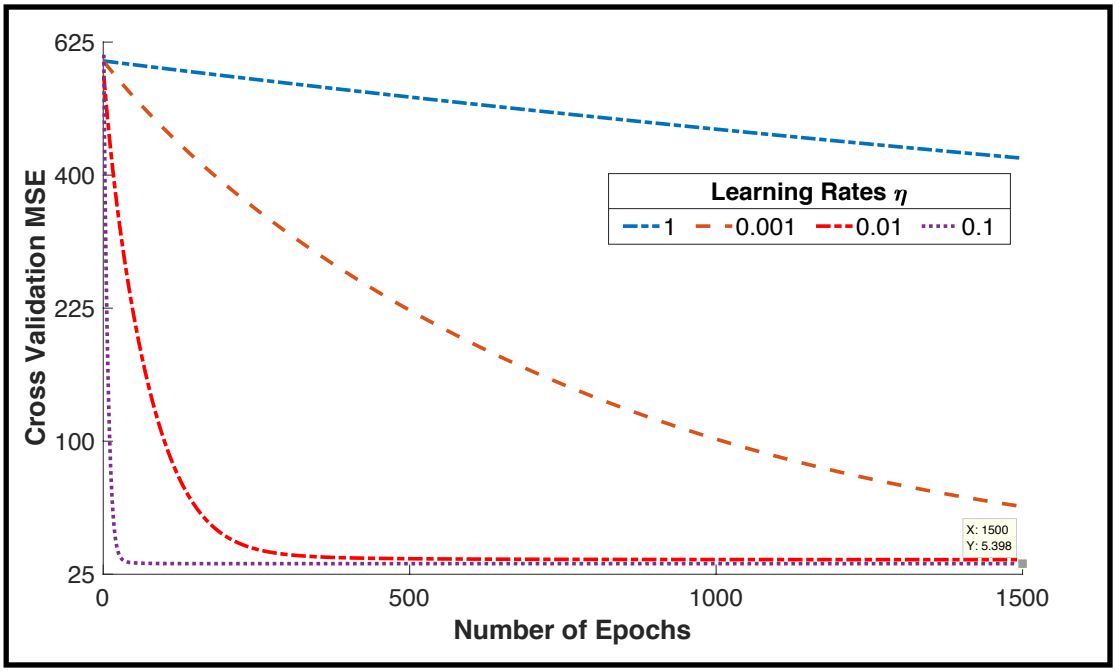


Figure 1 : Cross Validation MSE vs Number of epochs for various learning rates

Note: For learning rates higher than 0.1 the RMSE was behaving erratically and sometimes diverging to infinity, which shows that higher learning rate may not help the optimisation reach the global minimum of the cost function. Lower learning rate takes many epochs to converge which means the converging takes a lot of time.

Result: For η =0.1:
Training MSE = 28.77
Testing MSE = 33.59

(b) Ridge Regression (L2 regularization)

Ridge regression belongs a class of regression tools that use L2 regularization. L2 regularization adds an L2 penalty, which equals the square of the magnitude of coefficients. A tuning parameter (λ) controls the strength of the penalty term. When $\lambda = 0$, ridge regression equals least squares regression. If $\lambda = \infty$, all coefficients are shrunk to zero. The ideal penalty is therefore somewhere in between 0 and ∞.

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables).

Ridge regression places a particular form of constraint on the parameters ($\beta's$): $\hat{\beta}_{ridge}$ is chosen to minimize the penalized sum of squares:

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

which is equivalent to minimization of $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2$ subject to, for some

$c > 0$, $\sum_{j=1}^{p} \beta_j^2 < c$, i.e. constraining the sum of the squared coefficients.

Applying Ridge on Boston Housing Data

The data has been normalised to have mean 0 and unit variance because ridge's objective function penalises the square of weights and if the weights are not normalised the feature with larger values will be penalised more than the one with lower values.

Matlab: ridge()

The new normalised data point is formed using the formula given below:

$$Z_j^{(i)} = \frac{X_j^{(i)} - \bar{X}_j}{\sigma_j}$$

where $\bar{X}_j$ is the mean of jth predictor and σ_j is the standard deviation of that predictor and $X_j^{(i)}$ is the jth feature value of ith data point.

We first try to determine a suitable λ for ridge regression. It can be shown that the error converges for a value of $\log(\lambda)$ below 1. (The data has been normalised)

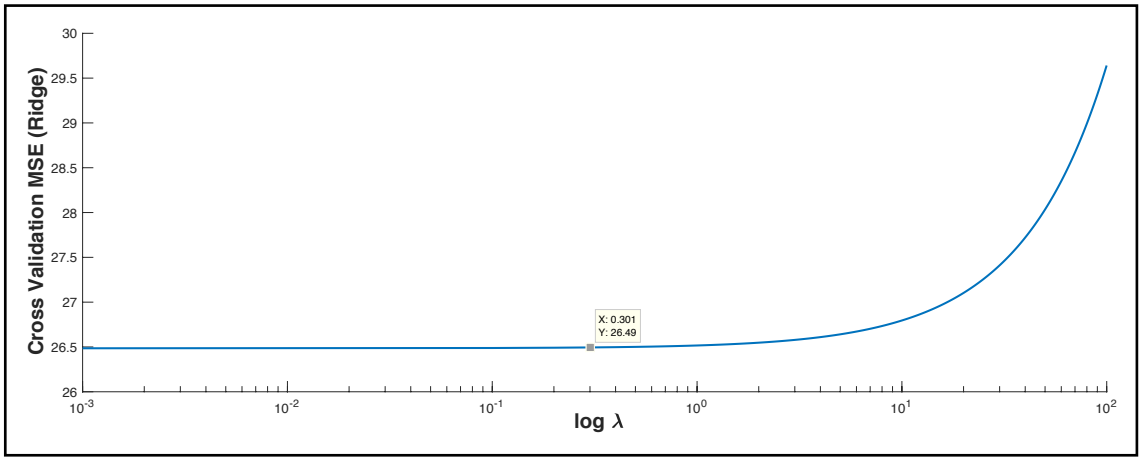


Figure 4 : Cross-validation MSE v/s log(lambda)

Looking at figure 4 on the next leaf, we can say our optimum λ is 0.301. Refer to Table 5 for MSE and **w**'s

(c) LASSO Regression (L1 regularization)

LASSO stands for Least Square Shrinkage and Selection operator.
LASSO is a linear regression technique which provides both variable selection and regularisation. LASSO penalises the L1 norm of the weights of variables. It shrinks the weights of predictors which are not important to the prediction. In unregularised regression, it is often seen that the model 'overfits' the data. This leads to generalisation of model to the unseen data as the model overly relies on some of the predictors which might not have a role in prediction.

Objective:

$$\min_{w_0 \ldots w_n} J = \frac{1}{N} \sum_{i=1}^{N} (w_0 + w_1 x_1^i + \ldots + w_n x_n^{(i)} - y^{(i)})^2 \quad \text{subject to} \quad \sum_{j=1}^{n} \mid w_j \mid \leq t$$

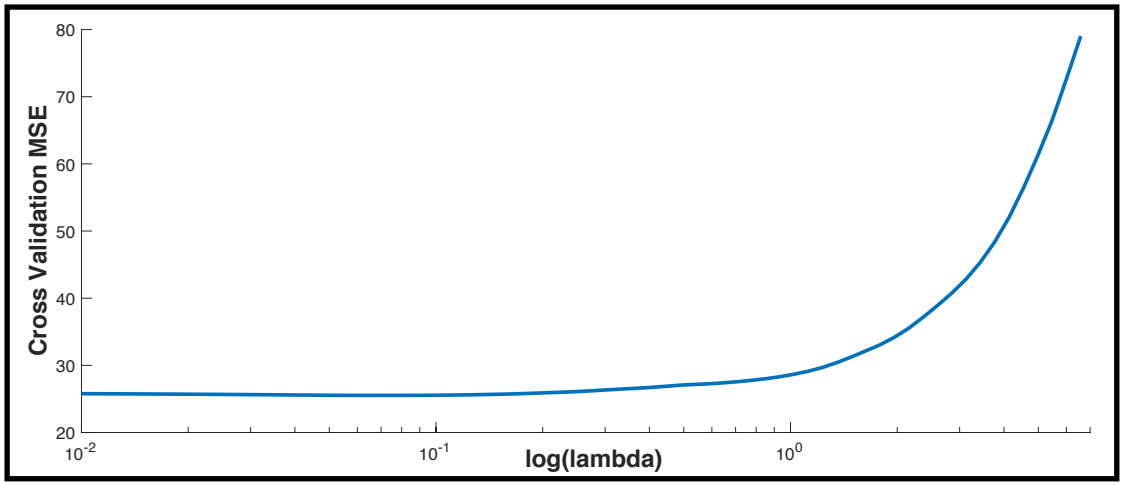
In vector form same can be written as $\min_{\mathbf{w} \in \mathbb{R}^n} \{\frac{1}{N} \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_2^2\}$ **subject to** $\|\mathbf{w}\|_1 \leq t$

This qualifies as a problem of nonlinear programming and can be written in the Lagrangian form:

$$\min_{\mathbf{w} \in \mathbb{R}^n} L = \frac{1}{N} \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 \text{ where } \frac{1}{N} \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|_2^2$$

is the residual sum of squares (RSS) and the remaining term is the norm (penalty) term. [7][10]

Applying LASSO on Boston Housing Data



Matlab: lasso()

Figure 2:

Figure 3 : Cross-validated MSE of Lasso fit

The data has been normalised to have mean 0 and unit variance because Lasso penalises the weights and if the weights are not normalised the feature with larger values will be penalised more than the one with lower values. Same normalisation method has been used as mentioned in ridge (Section 3.1.2)

To tune our hyper parameter λ, we use cross validation (above). As seen in the plot below, $\boldsymbol{\lambda = 0.07}$ **gives the least MSE**, so the same will be chosen for all analyses to follow *(Refer to Table 5 for optimum coefficients' value and MSE)*

A lower value of λ means less regularisation and hence less shrinkage in the value of the coefficients. It can be show that for $\lambda > 1$, 11 of the coefficients shrink to zero. As value of λ decreases our coefficient vector becomes less sparse with increasing degree of freedom of the fit.

We shall have all the features in the model (for better practical understanding, because all factors in real life have a significant role in deciding what the price should be, our only concern here is not just the response variable(MEDV) but also that how other predictors (some practically obvious predictors such as LSTAT, RM, CRIM etc.) play an important role in making a decision about price) at same MSE (with more degrees of freedom nonetheless, which means we shall use more predictors without compromising MSE and generalisation).

Model complexity is not an issue here (unlike Ensemble techniques wherein more depth (i.e. more overfitting) leads to more space and time complexity).

So we shall go with all the predictors in the model. *(Refer to Table 5 for optimum coefficients' value and MSE)*

IMPORTANT NOTE:

There is evidence of non-linearity between LSTAT and MEDV. We have been trying to fit linear models so far assuming that the predictors vary linearly with output variable. Upon fitting a linear and non-linear curve (5th order polynomial), it becomes evident that there is indeed a non-linearity between LSTAT and MEDV (refer to Figure)

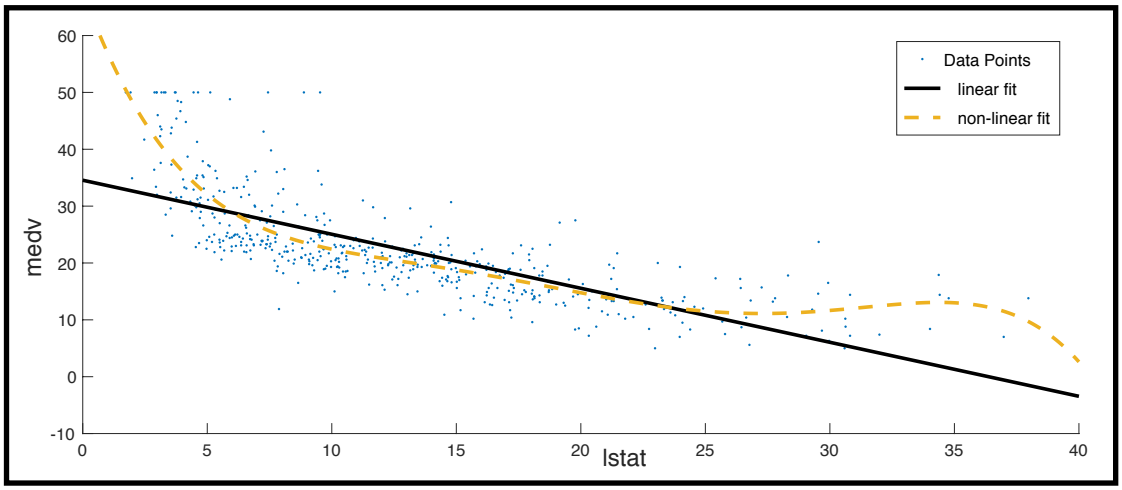


Figure 4 : Evidence of non-linearity between LSTAT and MEDV

To account for this nonlinearity in predictors we shall now not assume a linear model and will try nonlinear regressors such as SVM and Neural Networks.

(d) Comparison of results

The results below have been obtained by shuffling the data randomly and then dividing it into training and testing set with 70% of the data used for training. Since the split is random, the entire process was repeated 10 times and the RMSEs were averaged to give a more generalised value.

Predictors	Standard	Ridge	Lasso
w0	22.818	22.3045	0
w1	-1.285	-0.6411	-0.5463
w2	0.365	0.7264	0.5680
w3	0.446	0.1475	0
w4	0.853	0.9252	0.8660

Predictors	Standard	Ridge	Lasso
w5	-2.899	-1.8355	-1.4142
w6	4.876	2.5698	2.5664
w7	-1.177	0.1026	0
w8	-2.840	-2.6354	-2.2758
w9	0.307	0.2372	0.1804
w10	-0.119	0.1238	0
w11	-2.314	-1.7958	-1.7056
w12	0.961	0.8280	0.7707
w13	0.677	-4.1295	-4.0981

Table 5 : Comparison of coefficient values across regression techniques

Method	Training Set MSE	Test Set MSE
Standard Linear Regression	28.73	33.59
Ridge	23.13	27.04
Lasso	23.52	26.52

Table 6 : Comparison of training and test MSE across regression techniques

(e) Discussion and Inference

- Looking at the test set it can be deduced that the decrease in MSE is due to the fact that the Standard Linear Regression had overfit the data points which resulted in less generalisation to the unseen data points.
- Upon applying the predictors with large coefficient values were penalised so that the prediction is not very largely dependent on strangeness of each of the predictors which might create a problem while predicting unseen data points.
- Since Lasso and Ridge shrink variables, the model fit using these regularised regression techniques shows a decrease in testing error which explains the generalisation done here.
- The methods next employed would be CART Trees and ensemble methods such as bagging, boosting and Random Forest.

2.2.2 CART (Classification and Regression Trees)

Using CART on Boston Housing Data

1) Unpruned Tree, Fully Grown¶

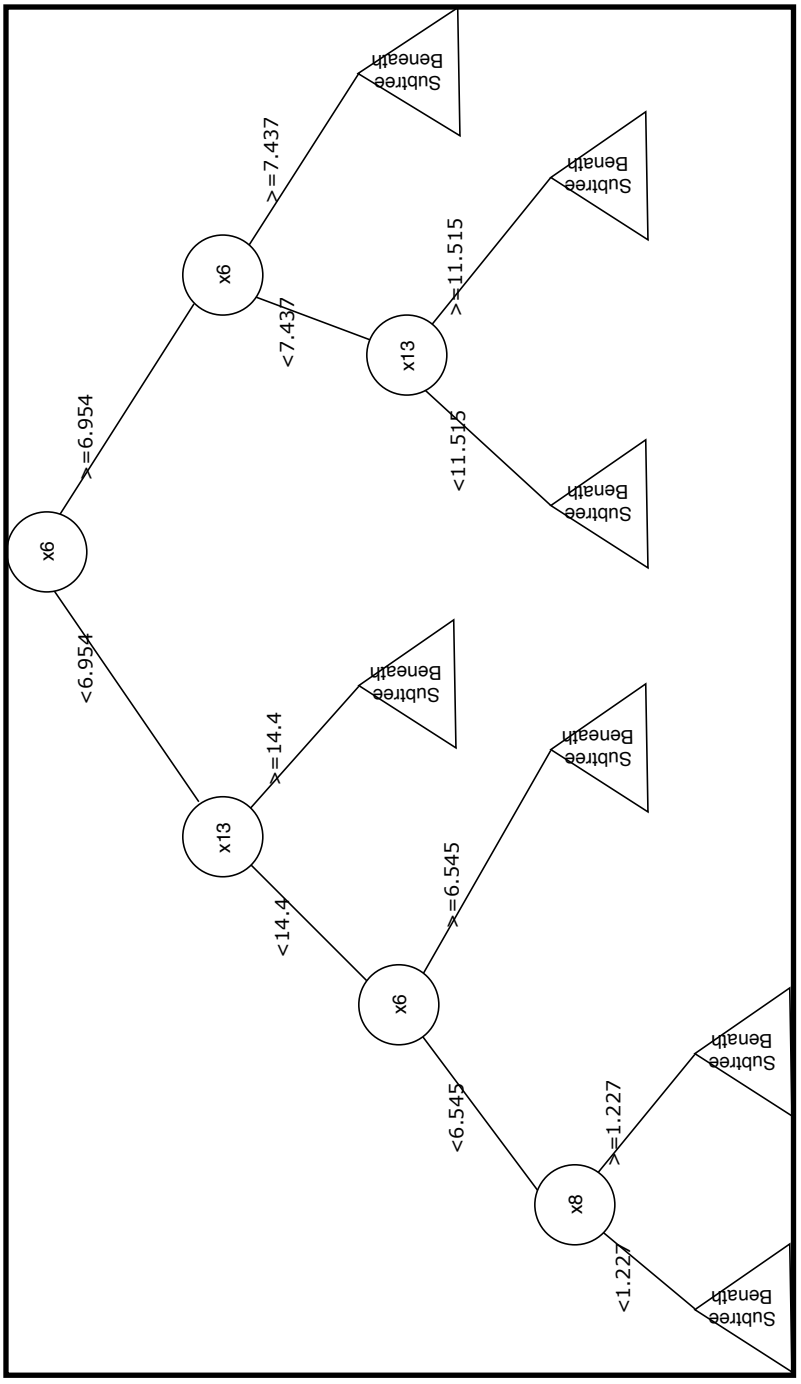


Figure 5 : Visualisation of Unpruned Tree, fully grown

Matlab:
fitrtree()

NodeID	ParentID	Split predictor	Cut Point	IsBranchNode	Left Child	Right Child	Node Error	Node Mean	NodeSize	NodeID	ParentID	Split predictor	Cut Point	IsBranchNode	Left Child	Right Child	Node Error	Node Mean	NodeSize
0	0	'x6'	6.95	TRUE	2	3	81.66	22.48	354	81	69	'x12'	367.67	TRUE	94	95	3.55	20.61	42
1	1	'x13'	14.40	TRUE	4	5	39.38	19.90	301	82	69	'x1'	0.06	TRUE	96	97	8.34	18.52	15
2	1	'x6'	7.44	TRUE	6	7	69.80	37.12	53	83	70	'x10'	414.00	TRUE	98	99	2.64	24.04	21
3	2	'x6'	6.55	TRUE	8	9	23.45	23.41	179	84	70	'x13'	5.59	TRUE	100	101	2.43	22.64	35
4	2	'x1'	7.46	TRUE	10	11	18.22	14.75	122	85	71	'x6'	6.22	TRUE	102	103	0.92	20.81	15
5	3	'x13'	11.52	TRUE	12	13	37.33	32.45	31	86	71	N/A		FALSE	0	0	0.00	16.10	1
6	3	'x1'	2.74	TRUE	14	15	41.52	43.70	22	87	74	N/A		FALSE	0	0	0.90	19.76	7
7	4	'x8'	1.23	TRUE	16	17	20.32	22.18	137	88	74	N/A		FALSE	0	0	0.91	17.27	3
8	4	'x10'	269.00	TRUE	18	19	12.82	27.40	42	89	76	N/A		FALSE	0	0	0.69	18.04	5
9	5	'x1'	0.18	TRUE	20	21	11.51	16.98	74	90	76	'x3'	1.24	TRUE	104	105	3.06	14.94	38
10	5	'x13'	25.30	TRUE	22	23	9.20	11.33	48	91	78	'x12'	5.17	TRUE	106	107	0.77	13.42	11
11	6	'x3'	7.26	TRUE	24	25	20.78	33.55	29	92	78	N/A		FALSE	0	0	0.19	11.90	3
12	6	N/A		FALSE	0	0	1.96	16.40	2	93	82	N/A		FALSE	0	0	1.88	17.57	3
13	7	'x11'	17.90	TRUE	26	27	19.80	44.73	21	94	82	'x8'	3.13	TRUE	108	109	2.92	20.84	39
14	7	N/A		FALSE	0	0	0.00	21.90	1	95	83	N/A		FALSE	0	0	0.00	11.90	1
15	8	N/A		FALSE	0	0	192.08	40.20	3	96	83	'x8'	1.68	TRUE	110	111	5.58	18.99	14
16	8	'x6'	6.06	TRUE	28	29	9.05	21.78	134	97	84	'x13'	9.27	TRUE	112	113	1.82	24.26	20
17	9	'x11'	17.85	TRUE	30	31	5.24	30.45	13	98	84	N/A		FALSE	0	0	0.00	19.80	1
18	9	'x5'	0.53	TRUE	32	33	10.20	26.03	29	99	85	N/A		FALSE	0	0	0.00	27.00	1
19	10	'x11'	17.20	TRUE	34	35	8.05	19.95	17	100	85	'x12'	155.48	TRUE	114	115	1.92	22.51	34
20	10	'x7'	85.10	TRUE	36	37	9.12	16.09	57	101	86	N/A		FALSE	0	0	0.56	21.33	9
21	11	'x1'	11.45	TRUE	38	39	6.37	12.37	34	102	86	N/A		FALSE	0	0	0.42	20.02	6
22	11	'x6'	4.95	TRUE	40	41	7.00	8.79	14	103	91	N/A		FALSE	0	0	0.00	19.30	1
23	12	'x9'	2.50	TRUE	42	43	11.52	32.96	28	104	91	'x12'	251.77	TRUE	116	117	2.62	14.82	37
24	12	N/A		FALSE	0	0	0.00	50.00	1	105	92	N/A		FALSE	0	0	0.00	11.70	1
25	14	'x8'	2.09	TRUE	44	45	15.53	45.82	17	106	92	'x12'	193.83	TRUE	118	119	0.52	13.59	10
26	14	N/A		FALSE	0	0	11.74	40.13	4	107	95	N/A		FALSE	0	0	0.80	22.83	6
27	17	'x8'	1.35	TRUE	46	47	7.38	20.32	59	108	95	'x11'	18.55	TRUE	120	121	2.45	20.48	33
28	17	'x10'	222.50	TRUE	48	49	7.35	22.93	75	109	97	N/A		FALSE	0	0	4.42	16.63	3
29	18	N/A		FALSE	0	0	3.27	31.70	8	110	97	'x8'	2.46	TRUE	122	123	3.96	19.64	11
30	18	N/A		FALSE	0	0	1.85	28.44	5	111	98	'x3'	5.48	TRUE	124	125	1.10	24.05	19
31	19	'x5'	0.44	TRUE	50	51	8.53	26.96	23	112	98	N/A		FALSE	0	0	0.00	28.10	1
32	19	N/A		FALSE	0	0	0.82	22.50	6	113	101	N/A		FALSE	0	0	0.00	18.60	1
33	20	N/A		FALSE	0	0	5.07	23.35	4	114	101	'x8'	4.87	TRUE	126	127	1.50	22.63	33
34	20	'x1'	0.18	TRUE	52	53	4.31	18.90	13	115	105	N/A		FALSE	0	0	0.27	13.56	7
35	21	'x3'	8.12	TRUE	54	55	9.01	19.26	11	116	105	'x8'	1.38	TRUE	128	129	2.71	15.12	30
36	21	'x10'	688.50	TRUE	56	57	6.17	15.33	46	117	107	N/A		FALSE	0	0	0.04	14.70	2
37	22	'x5'	0.76	TRUE	58	59	2.69	13.74	18	118	107	N/A		FALSE	0	0	0.25	13.31	8
38	22	'x5'	0.68	TRUE	60	61	6.02	10.83	16	119	109	'x6'	5.97	TRUE	130	131	2.97	21.73	10
39	23	N/A		FALSE	0	0	2.99	12.08	4	120	109	'x7'	43.65	TRUE	132	133	1.25	19.94	23
40	23	'x8'	1.60	TRUE	62	63	2.57	7.48	10	121	111	N/A		FALSE	0	0	0.84	21.10	6
41	24	N/A		FALSE	0	0	14.08	29.74	8	122	111	N/A		FALSE	0	0	2.06	17.88	5
42	24	'x3'	5.41	TRUE	64	65	4.66	34.26	20	123	112	'x11'	15.65	TRUE	134	135	0.73	23.53	12
43	26	N/A		FALSE	0	0	0.00	50.00	3	124	112	N/A		FALSE	0	0	0.45	24.96	7
44	26	'x13'	3.15	TRUE	66	67	14.30	44.92	14	125	115	'x5'	0.44	TRUE	136	137	1.25	23.06	23
45	28	N/A		FALSE	0	0	0.00	27.90	1	126	115	'x6'	6.31	TRUE	138	139	0.66	21.63	10
46	28	'x6'	4.26	TRUE	68	69	6.50	20.19	58	127	117	N/A		FALSE	0	0	0.64	12.60	2
47	29	N/A		FALSE	0	0	11.18	31.50	3	128	117	'x7'	94.95	TRUE	140	141	2.37	15.30	28
48	29	'x13'	11.33	TRUE	70	71	4.00	22.58	72	129	120	N/A		FALSE	0	0	0.41	19.83	4
49	32	N/A		FALSE	0	0	3.71	24.69	9	130	120	N/A		FALSE	0	0	0.64	23.00	6
50	32	'x7'	42.25	TRUE	72	73	6.21	28.41	14	131	121	N/A		FALSE	0	0	0.25	21.00	8
51	35	'x13'	18.00	TRUE	74	75	3.08	18.55	12	132	121	'x1'	0.08	TRUE	142	143	0.87	19.37	15
52	35	N/A		FALSE	0	0	0.00	23.10	1	133	124	N/A		FALSE	0	0	0.06	22.25	2
53	36	N/A		FALSE	0	0	3.89	20.66	8	134	124	'x10'	277.00	TRUE	144	145	0.47	23.78	10
54	36	N/A		FALSE	0	0	3.53	15.53	3	135	126	N/A		FALSE	0	0	0.09	21.10	2
55	37	'x9'	5.50	TRUE	76	77	4.73	15.52	45	136	126	'x6'	6.11	TRUE	146	147	0.96	23.25	21
56	37	N/A		FALSE	0	0	0.00	7.00	1	137	127	N/A		FALSE	0	0	0.08	20.73	4
57	38	'x1'	9.95	TRUE	78	79	1.82	13.50	17	138	127	N/A		FALSE	0	0	0.14	22.23	6
58	38	N/A		FALSE	0	0	0.00	17.80	1	139	129	'x1'	1.78	TRUE	148	149	2.09	15.83	12
59	39	N/A		FALSE	0	0	4.64	12.74	7	140	129	'x7'	97.65	TRUE	150	151	2.22	14.90	16
60	39	N/A		FALSE	0	0	2.01	9.33	9	141	133	N/A		FALSE	0	0	0.37	18.68	6
61	41	N/A		FALSE	0	0	0.84	6.30	5	142	133	N/A		FALSE	0	0	0.67	19.83	9
62	41	N/A		FALSE	0	0	1.52	8.66	5	143	135	N/A		FALSE	0	0	0.00	25.00	1
63	43	'x3'	2.32	TRUE	80	81	2.31	35.31	14	144	135	N/A		FALSE	0	0	0.34	23.64	9
64	43	N/A		FALSE	0	0	1.44	31.78	6	145	137	N/A		FALSE	0	0	0.25	24.50	2
65	45	N/A		FALSE	0	0	7.81	42.32	5	146	137	'x6'	6.31	TRUE	152	153	0.85	23.12	19
66	45	N/A		FALSE	0	0	12.06	46.37	9	147	140	N/A		FALSE	0	0	0.74	15.01	7
67	47	N/A		FALSE	0	0	0.00	27.50	1	148	140	N/A		FALSE	0	0	1.75	16.96	5
68	47	'x7'	72.00	TRUE	82	83	5.66	20.06	57	149	141	N/A		FALSE	0	0	1.42	14.17	7
69	49	'x7'	36.90	TRUE	84	85	2.97	23.16	56	150	141	N/A		FALSE	0	0	2.10	15.47	9
70	49	'x6'	6.38	TRUE	86	87	2.16	20.51	16	151	147	N/A		FALSE	0	0	0.46	22.00	4
71	51	N/A		FALSE	0	0	1.30	30.60	6	152	147	'x6'	6.35	TRUE	154	155	0.53	23.41	15
72	51	N/A		FALSE	0	0	3.62	26.78	8	153	153	N/A		FALSE	0	0	0.12	24.27	3
73	52	'x1'	0.17	TRUE	88	89	2.20	19.01	10	154	153	'x10'	258.50	TRUE	156	157	0.41	23.20	12
74	52	N/A		FALSE	0	0	1.10	16.25	2	155	155	N/A		FALSE	0	0	0.06	24.15	2
75	56	'x13'	15.97	TRUE	90	91	3.78	15.30	43	156	155	'x1'	0.04	TRUE	158	159	0.26	23.01	10
76	56	N/A		FALSE	0	0	2.25	20.20	2	157	157			FALSE	0	0	0.00	22.00	1
77	58	'x7'	99.40	TRUE	92	93	1.03	13.09	14	158	157			FALSE	0	0	0.16	23.12	9
78	58	N/A		FALSE	0	0	1.13	15.40	3										
79	64	N/A		FALSE	0	0	0.90	34.02	5										
80	64	N/A		FALSE	0	0	1.64	36.03	9										

Table 7 : Tree Statistics

Number of training data points	354
Number of testing data points	152
Training Error	2.4599
Testing Error	28.3857
Number of Nodes	159
Number of leaves	73
Number of split	72
Average data points per leaf node(in training)	4.84

Table 8 : Unpruned Tree : Description

Discussion:

- The testing error is greater than training error. Clearly the tree has overfit the data and cannot generalise well for the unseen data.
- Number of splits is 72 which is a large number for a smaller training data set of 354 data points.
- The complexity of tree is also more than required with 73 terminal nodes.
- Average data points per leaf node is 4.84 which means the model is trying to fit each and every data point at the cost of testing error.
- We shall try pruning if a decrease in complexity is seen.

2) Pruning the Tree, Fully Grown

We shall use a famous method of pruning called 'The Cost Complexity Pruning' (also called weakest link pruning) which gives an alternative to pessimistic approach(pre-pruning). Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree T$\subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

is minimised. T_0 is the fully grown tree and $|T|$ is the number of terminal nodes in the tree T. R_m is the subset of predictor space corresponding to the mth terminal node and $\hat{y}_{R_m}$ is the predicted response in R_m. α is the pruning parameter which controls the trade-off between the tree's complexity(number of terminal nodes) and its fit to the training data.[9]

Pruning not only lessens the complexity(depth) of tree but also gives a better generalisation. We can start pruning now and shall see for varying values of alpha what the CV MSEs are.

Deciding value of α

α is the pruning parameter and controls the tradeoff between the tree's complexity and its fit to the training data. The optimum value of α is achieved by using cross-validation which minimises MSE of the folds.

The plot below shows how CV MSE varies with increasing values of α, α=0 means there is no pruning or there is no penalty on the increased size of the tree and the tree shall grow to the maximum depth.

MATLAB function used : cvloss()

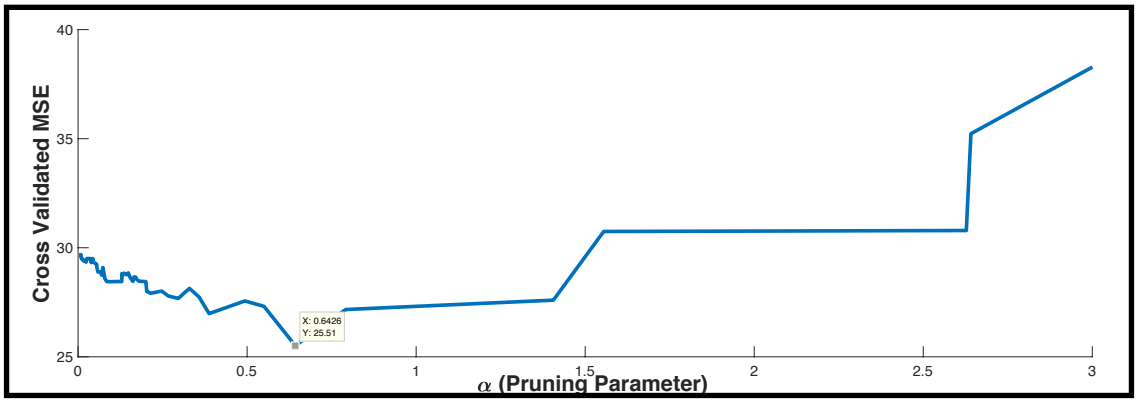


Figure 6: Using cross-validation to get an optimum value of alpha

Using Cross-Validation:

Looking at the curve above(Figure 6) it can be deduced that $\alpha = 0.6426$ gives the least cross validation error. So we choose the same α and then train the model and then test the data. The details at this value of α are:

Training MSE = 10.17
Testing MSE = 27.74
Number of terminal nodes = 10

It can be seen that the complexity of the tree has dropped several times although the decrease in test MSE is not very large.

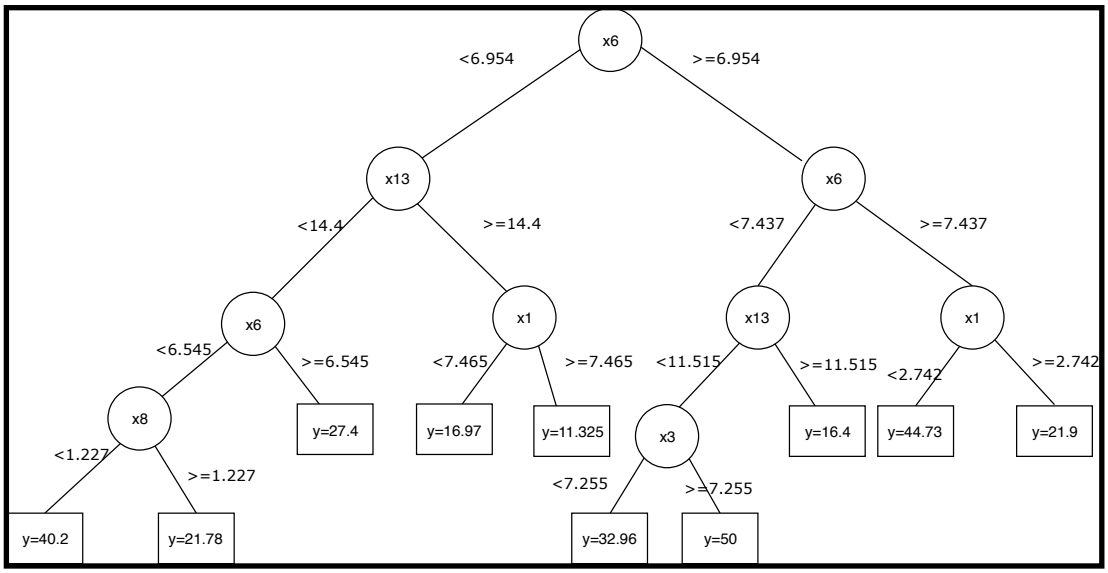


Figure 7: Final Pruned Tree

2.2.3 Ensemble Learning Algorithms

(a) Bagging

In bagging, each tree is trained on a different resampled-bootstrapped data set. Number of observation per leaf (leaf size) is limited so as to achieve this motive of growing shallow trees. Optimal leaf size makes sure that we do not try to fit each and every data point, therefore, the number of observation per leaf node is set to a minimum value. This means that no learner would have number of observations per leaf node lower than that, which prevents overfitting (resulting in higher variance per learner, but low bias per learner and the fitting would converge if number of points per leaf decreases below the Optimal Leaf Size.

In general, for regression, the optimal leaf size (minimum number of data points per leaf) is 5 (refer to Figure 9) and all of the features are used to train each bootstrapped model in the ensemble.

Out-of-bag Error Estimation:

When deciding if a sample should be included in the bootstrapped training sample, there is a 66% probability that the sample will be selected and be a part of that bootstrapped model. Now while calculating prediction of a sample we only consider those individual learners which have not used that sample to train their learner (or which were not in the bag for the training), this is called out-of-bag error estimation. It is close to cross validation if number of learners is large as it gets the prediction from only those learners who have never seen that data sample. This metric has been used in Bagging and Random forests.

Using Bagging on Boston Housing Data

Growing an ensemble with 150 trees, the graph of OOBError vs Number of trees was plotted to find the optimum number of trees to be used in the ensemble.

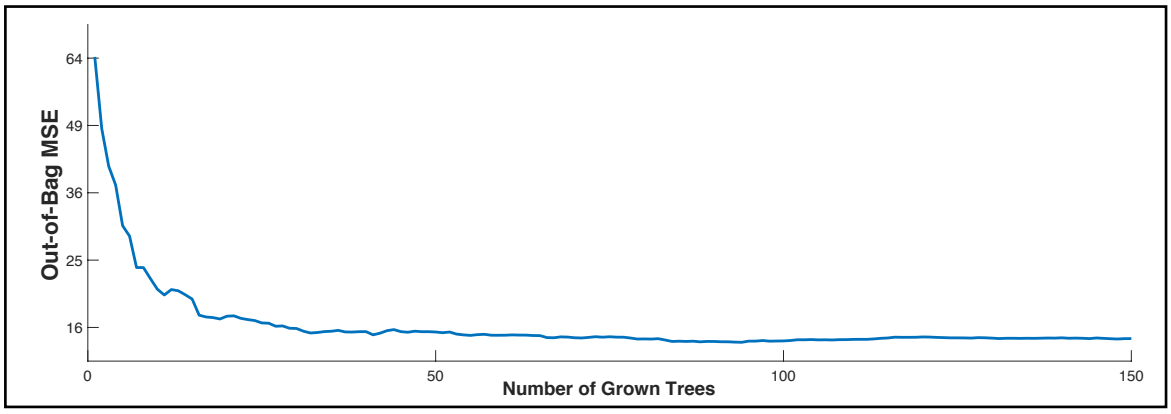
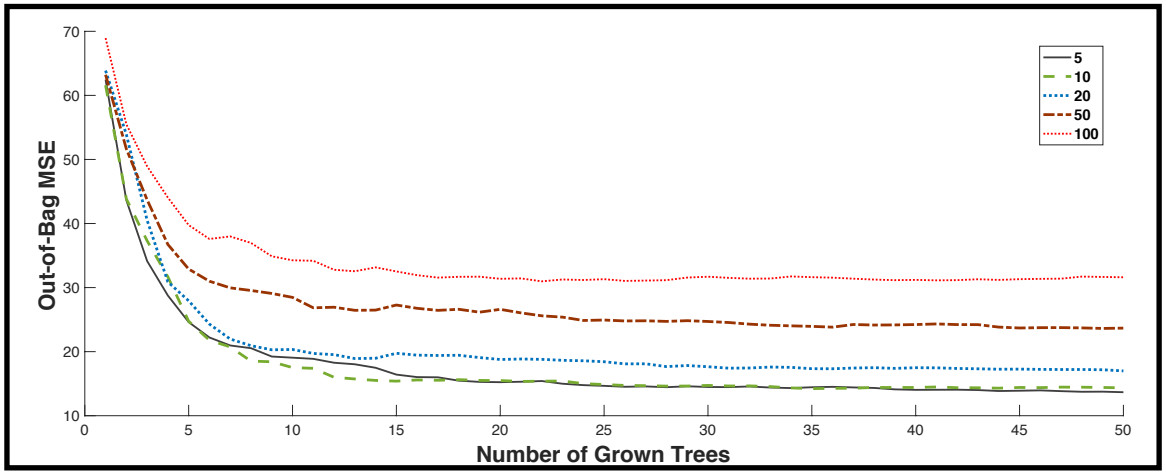


Figure 8 : Deciding number of learners in the Ensemble - OOBMSE vs No. of grown trees(learners)

The error is minimum for an ensemble of 143 trees (OOBError=13.8309) Using this ensemble to test its performance on the shuffled test set.

Test MSE (bagging) = 12.2843

A hyper-parameter which needed to be tuned was leaf-size (minimum number of data points per leaf) of each individual tree in the ensemble, this has been decided by looking at the Out-of-Bag Error (OOBEroor) by varying the leaf size and then training 100 trees for each leaf value. We saw that the OOBError was least when the leaf size was limited to 5 (Refer to the Figure 9)



Matlab:
treebagger()

Figure 9 : Deciding number of leaf nodes - OOBMSE vs No. of grown trees

MATLAB function used : oobError()

The black trace corresponds to 5 leaf size and can be seen that keeping the leaf size limited to 5 would give the least mean squared OOB Error.

It is worth noting that each predictive learner is very simple (less depth, low bias, high variance) but when the predictions are bagged together the problem of high variance is dealt with as all the bagged predictions are averaged. So, instead of relying on a single learner which has high variance, we take predictions from all high-variance-low-bias learners and then average it out. Averaging the values reduces the variance. In ensemble techniques, it is fairly easy to achieve the correct bias-variance trade-off.

(b) Random Forests

Bagging is a particular case of Random Forest. As opposed to Bagging, in Random Forest we do not use all the predictors while deciding for a split but instead we use a random predictor from a subset of predictors. For random forest, the predictors considered at each split are one-third of the total predictors.[3]

The least OOBError is when around 84 trees are used in the ensemble which is around 14.1300 So, 100 trees will be grown in the ensemble. The generalisation error converges as number of trees increases.

Upon running the trained model on 100 test sets, all randomly shuffled, the mean MSE was:

Test MSE (Random Forest) = 12.5549

(c) Boosting (AdaBoost)

In Ada Boosting, the process of growing each weak learner is sequential and a new learner is learnt only when the previous learner has been trained. It changes as the performance of prior learners on that training example changes. [5] [6]

As in bagging, the first machine is trained on examples picked with replacement (of size N) from the original training set. We then pass all the training patterns through this first machine and note which ones are most in error. For regression machines, those patterns whose predicted values differ most from their observed values are defined to be "most" in error. For those patterns most in error, their sampling probabilities are adjusted so that they are more likely to be picked as members of the training set for the second machine. Therefore, as we proceed in constructing machines, patterns that are difficult are more likely to appear in the training sets.

In Boosting, while giving probabilities to each training example three types of loss functions can be considered:

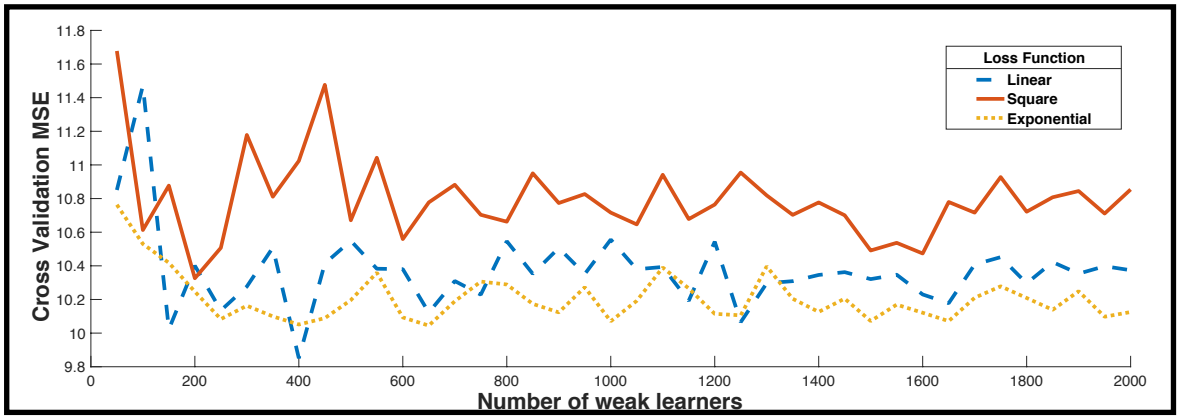
Let D = $\sup|\hat{y}_i - y_i|$ for i=1,.....,N (where N is number of training points)

1) Linear Loss Function $\dfrac{|\hat{y}_i - y_i|}{D}$

2) Square Loss Function $\dfrac{|\hat{y}_i - y_i|^2}{D^2}$

3) Exponential Loss Function $1 - \exp(-\dfrac{|\hat{y}_i - y_i|}{D})$

The figure below (Figure 10) plots cross validated vs Number of weak learners for all three loss functions.



Python(sklearn):
AdaBoostregressor()

Figure 10 : Deciding number of estimators for different loss functions

The minimum CV MSE was given when number of estimator = 400 and the loss function used in linear

Testing MSE (AdaBoost R) = 10.12

Algorithm	Testing MSE
Normal Tree (Unpruned)	28.3857
Normal Tree (Pruned, regularised)	27.7400
Bagging	12.2843
Random Forest	12.5549
Ada Boost R	10.1200

Table 9 : Comparison of CART and Ensemble Methods

2.2.4 Support Vector Regression (SVR)

The dual formula of SVR is the following:

Minimise

$$L(\alpha) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) + \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i^* - \alpha_i)$$

such that

$$\sum(\alpha_n - \alpha_n^*) = 0 \qquad \forall n : 0 \leq \alpha_n, \alpha_n^* \leq C$$

where ϵ is the 'soft-margin' margin and C is the penalty imposed on observations that lie beyond the ϵ-margin. It prevents overfitting. This value determines the trade-off between the flatness of f(x) and the amount upto
which deviations beyond ϵ are tolerated.

If the data points are not linearly separable, kernels are used to transform the data points into higher dimension wherein they become linearly separable.

Three types of kernels are most commonly used:

Linear Kernel :	$\phi(x_j, x_k) = x_j' x_k$	
Gaussian Kernel:	$\phi(x_j, x_k) = e^{-\gamma\|\|x_j - x_k\|\|}$	
Polynomial Kernel:	$\phi(x_j, x_k) = (1 + x_j' x_k)^p$	where p is the degree.

Applying SVR on Boston Housing Dataset

SVR was applied on the dataset and only Gaussian kernel was tried to account for non-linearity in the predictors. The data has been normalised for convergence, using same normalisation technique mention in Section 3.1.2.

10-fold cross validation was applied to tune the following parameters:
1) C
2) ϵ
3) γ in Gaussian Kernel

Results for SVR
1) Gaussian kernel gave the best CV MSE with C = 100, $\epsilon = 0.1$ and $\gamma = 1$.
2) The Test MSE is as follows:
   Gaussian Kernel(at C = 100, ϵ = 0.1 and γ = 1) = 11.9

3) As initially stated in Linear Regression section, it is clear now that there was some nonlinearity in the data points. Using kernels, which transformed these non linear data points to linearly separable data points, we achieved a significantly lesser MSE.

2.2.5 Neural Network Regressor

After SVM, Neural Network was tried to compensate for non-linearity found in the variables. Gradient Descent Back-propagation was used to train our Neural network. A single hidden layer with different hidden layer size was tried (number of neurons) with learning rate(η) as 0.0001 and with initial weights set as 0.01.

10-fold Cross Validation on training dataset was used to decide number of neurons in the hidden layers.
The graph below demonstrates the validation process:

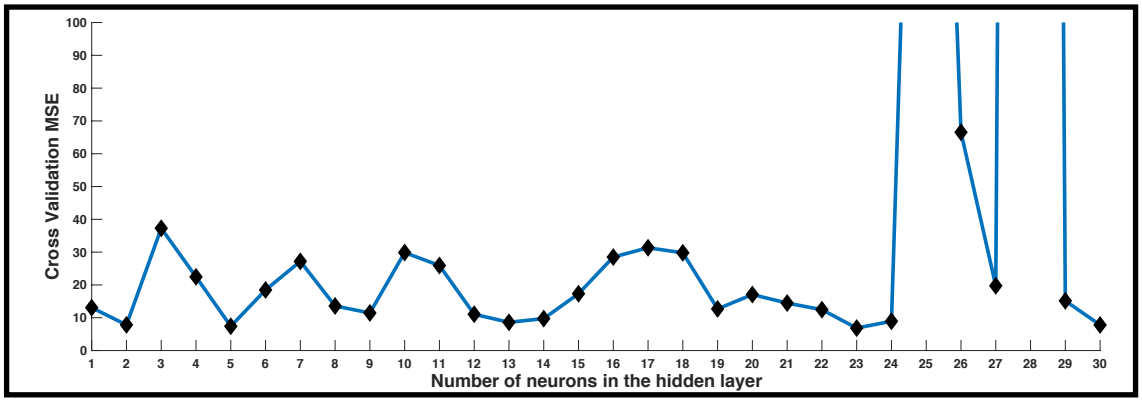
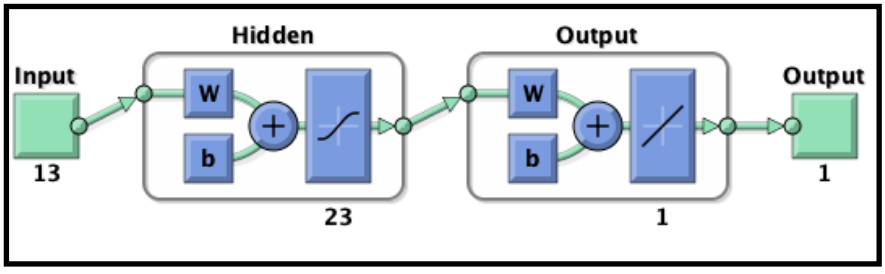


Figure 11 : Deciding number of neurons in the hidden layers

The lowest cross validation MSE was at number of neurons = 23.

The figure below shows the chosen architecture:



Matlab:
feedforwardnet()

Figure 12: Architecture of chosen model

The activation function used in the hidden layer tan sigmoid:

$\sigma_{tansig}(a) = \dfrac{1 - e^{-a}}{1 + e^{-a}}$, where a is the activation value

whereas identity or linear activation function has been used in the output layer

$$\sigma_{identity}(a) = a$$

The figure below gives the training and CV error for number of neurons=23

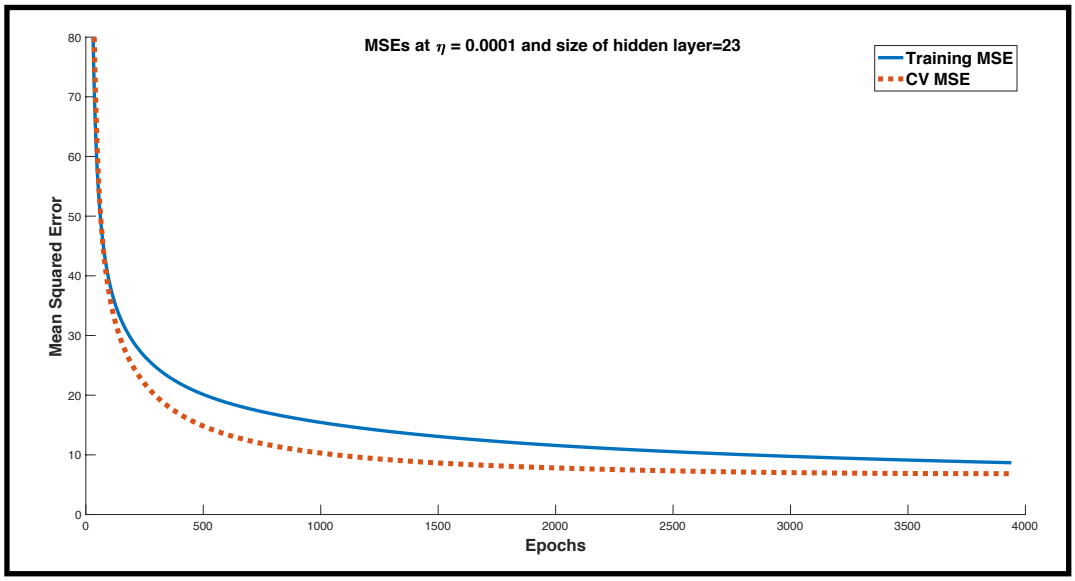


Figure 13 : Training and Validation with size of hidden layer=23

The trained model was then tested on the test set(not used during training or validation) and the results are as follows:

Training MSE = 8.6881
CV MSE = 6.8472
Testing MSE = 14.1300

A different learning rate was tried but the model was not converging at other learning rate. An additional hidden layer was also added but the performance did not improve.

NN and SVR both give better results than linear regression method but still give worse result than tree-based methods. This explains that the non-linearity in a few variables led the linear models to perform badly but there is another factor which leads the non-linear models to perform worse than tree-based methods. This factor could perhaps be ***outliers*** *in the dataset (since it is natural to have outliers in real*

estate market due to luxury houses, bad shape foreclosure properties, etc.) and as tree-based methods are robust to outliers, we got better results in those.

2.2.6 k-Nearest Neighbours

Parametric methods used so far have a disadvantage, they assume a model for f(x) and then tune the parameters on that assumed framework. If the assumed model is not the same as the real model, we end up getting a poor performance. In contrast, non-parametric approaches such as k-NN do not explicitly assume a parametric form for the target function f(X) and in turn provide a more flexible approach.

General approach of KNN regressor is as follows:

1) For a value of k and a prediction data point x_0
2) KNN identifies the training data points N_0 closest* to the prediction data point x_0
3) Estimate $\hat{y}$ (prediction of that particular data point) using average of all responses in N_0 i.e.

$$\hat{y} = \frac{1}{k} \sum_{x_i \in N_0} y_i$$

where y_i is the response of each nearest data point

---

* To come to a decision about how close a point is to other data point (in higher dimension), a number of distance metrics have been taken into account. Some of them are:

1) Minkowski Distance = $\left(\sum |x_1 - x_2|^p \right)^{\frac{1}{p}}$ where x_1 and x_2 are two data points
   a. If p=1, it is called Manhattan Distance
   b. If p=2, it is called Euclidean Distance

2) There are other types of distance metrics as well like Chebyshev, Mahalanobis, etc.

Applying KNN on Boston Housing Data

The parameters which are needed to be tuned is basically k. We have chosen Euclidean Distance as our distance metric. The data has been normalised using same normalisation technique as mentioned in Section 3.1.2 .
To decide this parameter, 10-fold cross validation has been used.

Python(sklearn): knnregressor()

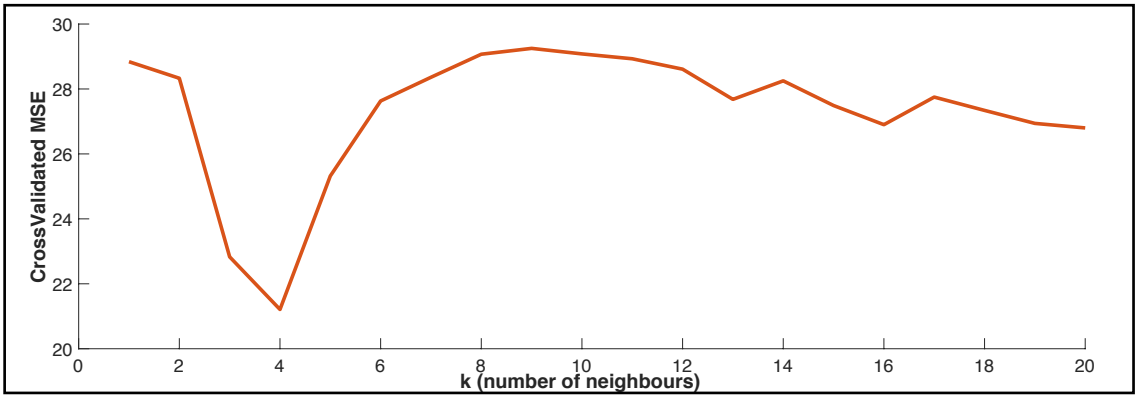


Figure 14 : Deciding number of neighbours

Results for KNN:

1. The minimum CV MSE occurred at k=4 and when using Euclidean distance as the distance metric.
2. Then entire data set was randomly split into testing and training set with 354 training points and 152 testing points.
3. The Testing MSE at k=4 was 23.23.
4. While choosing k in k-nn, it is, in general, a case of bias-variance trade off.
5. A small value of k provides the most flexible fit with low bias but high variance.
6. A large value of k provides a smoother and least flexible fit with high bias but low variance.
7. The above two points can be validated by the fact that in (5) the prediction depends only on smaller number of neighbours so *varying* even one of the neighbours will have a huge impact on predicted output. The case is opposite with larger k where prediction will depend on a lot of observations so by a *variation* in one of the neighbours would not affect the predicted output.

2.2.7 K-means Clustering (Unsupervised Learning)

We will apply k-means clustering to Boston Housing data set to see if some natural clusters show up. We shall begin with k(number of clusters) = 2.(The data has been normalised)

Lets analyse what the mean value of each parameter in each cluster is:

Variable Name	Actual Predictor Name	Cluster 1	Cluster 2	Comment
x1	CRIM	-0.40	0.49	Per capita crime rate is less in C1

Matlab:
kmeans()

Variable Name	Actual Predictor Name	Cluster 1	Cluster 2	Comment
x2	ZN	0.39	-0.48	More proportion of residential land in C1
x3	INDUS	-0.58	0.71	Less non-retail business in C1
x4	CHAS	0.04	-0.05	—
x5	NOX	-0.68	0.83	Less Nitric Oxide concentration in C1
x6	RM	0.38	-0.46	More average room in C1
x7	AGE	-0.60	0.74	Less number of old houses in C1
x8	DIS	0.57	-0.70	Distance is more to employment centres in C1
x9	RAD	0.09	-0.12	Radial highway is more accessible to tracts in C1
x10	TAX	-0.67	0.82	—
x11	PTRATIO	-0.43	0.53	Less pupil to teacher ratio in C1
x12	B	0.36	-0.44	—
x13	LSTAT	-0.59	0.73	More lower status in C2
y	MEDV	0.49	-0.60	Costlier houses in C1

Table 9: Cluster analysis

Discussion:

Looking at the remarks above, it can be seen that tracts in cluster 1 seem to be well-to-do. Tracts in cluster 1 can be seen with less crime rate, larger residential plots, less nitric oxide concentration, larger homes (in terms of room number), new houses, less distance to radial highways, more teachers per pupil, less percentage of lower standard population whereas tracts in cluster 2 indicate less well-to-do neighbourhoods. We can see that clustering has resulted in natural and intuitive segregation of Boston into two, areas one with higher standard of living and the other with lesser standard of living.

2.3 Comparison of Results of Applied Algorithms

Algorithm	Testing MSE
Normal Tree (Unpruned)	28.3857
Normal Tree (Pruned)	27.7400
Bagging	12.2843
Random Forest	12.5549
Ada Boost R	10.1200
Standard Linear Regression	33.5924
Ridge	27.0618
Lasso	26.5348
KNN (Euclidean distance)	23.2300
SVR(Gaussian Kernel)	11.9000
Neural Network Regressor	14.1300

Table 10: Comparison of all algorithms applied

2.4 Feature Selection

To study feature importance, three methods have been considered.

Out-of-bag, predictor importance estimates by permutation measure how influential the predictor variables in the model are at predicting the response. The influence of a predictor increases with the value of this measure.

If a predictor is influential in prediction, then permuting its values should affect the model error. If a predictor is not influential, then permuting its values should have little to no effect on the model error. [7]

Another method of estimating the feature importance is via Lasso, as it shrinks only those predictors which have least role in predicting the response variable.In the end of the process, the one with most shrunken value, is the least important predictor and the one with the largest absolute value is the most important predictor.

Feature importance using CART involves finding those predictors which result in lesser error after the splitting is done based on that chosen predictor. According to Lasso x6 and x13 are most important features and x3, x7, x10 are least important feature (refer to Table 5). According to Random Forest, x6 and x13 are most important and x3, x4 and x2 are least important (using Matlab oobPermuteVarImportance function). CART, as well, gives out x6 and x13 as most important feature (refer to Table 7)

2.5 Discussion and Inference

1) Clearly, the ensemble and decision tree method perform better than any of the linear regression methods.

2) The result above shows that the best result is given by Boosting. Boosting grows learners sequentially, these learners are called weak or base learners, which are dependent on the performance of previous learner. Every time a new is trained keeping in consideration the previous model which helps the ensemble improve the performance.

3) AdaBoost.R performs the best because the trained models concentrate more on the examples which had more loss in the previous learners.

4) Bagging performs better than Normal CART as it an ensemble method which helps the overall model have low variance (due to shallow learners) as well as low bias (due to averaging of predictions made by each learner)

5) The results in Table 10 corroborate with [6]

6) Random forest, as the name suggests, brings in randomness while selecting which feature to be as the split feature. Chances are, in bagging, since all the features are considered for all the split every time each learner would select the same feature (as it will always result in lower MSE). This problem is tackled by Random Forest as it will only randomly select from a smaller number of features to do the split, thereby avoiding selecting the same best feature in every learner.

7) Boosting performs better because it learns sequentially and each time the learner (weak) improves a bit based on the error of the last learner on each data point.

8) The ensemble methods use Bootstrap Resampling method to train their learners.

9) All hyper-parameters have been decided using 10 fold cross-validation.

10) In SVR, Gaussian Kernel gave best results which proved that there was non linearity in the variables.

11) Neural Network was tried which gave better results than the linear techniques.

12) From table 10, it can be concluded that Ensemble Methods outperform all other methods except SVR.

13) Clustering resulted in two clusters, with each cluster signifying the status of the neighbourhood. We did not go to further analysis by increasing clusters because deduction of information would have become difficult.

14) Boston Housing Dataset has been used extensively throughout the literature to benchmark algorithms. [4][8]

3. Stock Index Prediction

3.1 Analysis of the Data - S&P 500 data

The data used for this project was obtained from http://www.ibiblio.org/pub/archives/misc.invest/historical-data/index/ stocks/sp500/

3.1.1 Data Description (Original Data)

The original dataset has 679 samples, 21 features. The description of the original features can be found below.

1. S&P High (S&P_HIGH)	2. S&P Low (S&P_LOW)
3. NYSE Advancing Issues (NYSE_ADV_ISS)	4. NYSE Declining Issues (NYSE_DECL_ISS)
5. OTC Advancing Issues (OTC_ADV_ISS)	6. OTC Declining Issues (OTC_DECL_ISS)
7. NYSE New Highs (NYSE_NEW_HIGHS)	8. NYSE New Lows (NYSE_NEW_LOWS)
9. OTC New Highs (OTC_NEW_HIGHS)	10. OTC New Lows (OTC_NEW_LOWS)
11. NYSE Total Volume (NYSE_TOT_VOL)	12. NYSE Advancing Volume (NYSE_ADV_VOL)
13. NYSE Declining Volume (NYSE_DECL_VOL)	14. OTC Total Volume (OTC_TOT_VOL)
15. OTC Advancing Volume(OTC_ADV_VOL)	16. OTC Declining Volume (OTC_DECL_VOL)
17. S&P Earnings (S&P_EARNINGS)	18. Short-term Interest Rates in the Three- Month Treasury Bond Yield (3MOBILLS)
19. Long-term Interest Rates in the Thirty year Treasury Bond Yield (LNGBONDS)	20. Gold (GOLD)
21. S&P Close (S&P_CLOSE)	

The dataset has these weekly information from January 1980 to December 1992. In the original data, no missing values were found.

3.2 Extracting New Dataset From The Old Data

From the 21 features mentioned above, 5 features were derived by Rao and Rao (1996) which have some relevance to the prediction of S&P500 index.

The following table shows these features:

Feature	Description
S&P_CLOS E	Weekly S&P Closing Price
NYSE Adv/ Decl	The ratio of the number of advancing issues and declining issues in the week for the stocks in the New York Stock Exchange (NYSE) ; breadth indicator for the stock market.
NYSE NewH/NewL	The ratio of the number of new highs and new lows achieved in the week for NYSE market; indicating strength of an uptrend or downtrend.
3MOBILLS	Short-term interest rates in the Three-Months Treasury Bill Yield.
30YRBOND S	Long-term interest rates in the 30-year Treasury Bond Yield.

First, lets go through a few terms which have been used to form these new features:

Block Average

The idea of block average is to capture the information conveyed by the data points adjacent to it. In this case, five unit block average has been taken. The formula to calculate the same is :

$$BA(t) = \frac{x(t-2) + x(t-1) + x(t) + x(t+1) + x(t+2)}{5}$$

Rate of Change

The rate of change is calculated using the formula:

$$ROC_n = \frac{x(t) - BA(t-n)}{x(t) + BA(t+n)}$$

where BA(t-n) is the five unit block average of adjacent values centered around the value n periods ago.

We prepared the new data matrix in the following way:

1) In the original dataset, there are three "0" entries under "NYSE_DECL_ISS". One of the features derived (unnormalised) from the given dataset is a ratio, such that the values in "NYSE_DECL_ISS" go in as the denominator. This means that we had to deal with three indeterminate values in x2. We have done this by simply replacing the 0s with 1s.

2) The zero values in question have been recorded against the following dates:
   - 1/7/83
   - 6/3/83
   - 2/22/91

3) The original dataset was observed to have a very large range. Therefore the variables have been normalised before incorporation into the new data matrix.

4) We have used the following function for normalisation:

$$New\,Value = \frac{(Old\,value - Mean)}{(Maximum\,Range)}$$

5) To predict the weekly percentage change in S&P 500 index, we use the following target :

$$Target = 100 \times \frac{(S\,\&\,P\,three\,weeks\,ahead) - (S\,\&\,P\,this\,week)]}{(S\,\&\,P\,this\,week)}$$

   This gives us values that vary from -24.339 to 18.132

6) The target value has been scaled such that the value varies from 0 to 1.

$$Scaled\,target = \frac{(Target + 24.339)}{50}$$

7) The ROC(3) for the first five features has been included in this data matrix.

Feature Description of New Dataset

Variable	Feature	Description
x1	S&P_CLOSE	Weekly S&P Closing Price (normalized)
x2	NYSE Adv/ Decl	The ratio of the number of advancing issues and declining issues in the week for the stocks in the New York Stock Exchange (NYSE) ; breadth indicator for the stock market. (normalized)

x3	NYSE NewH/ NewL	The ratio of the number of new highs and new lows achieved in the week for NYSE market; indicating strength of an uptrend or downtrend. (normalized)
x4	3MOBILLS	Short-term interest rates in the Three-Months Treasury Bill Yield. (normalized)
x5	30YRBONDS	Long-term interest rates in the 30-year Treasury Bond Yield. (normalized)
x6	ROC3_SPC	Rate-of-change of feature x1
x7	ROC3_A/D	Rate-of-change of feature x2
x8	ROC3_H/L	Rate-of-change of feature x3
x9	ROC3_3Mo	Rate-of-change of feature x4
x10	ROC3_30YRB	Rate-of-change of feature x5
y	Weekly_P/ C_Change	Weekly Percentage Change in S&P 500 index (scaled)

In further analyses, unless stated otherwise, this new dataset will be used.

3.3 Algorithms

3.3.1 Linear Regression

Data Division

Training set, validation and testing set have been formed by splitting the newly formed dataset. Training set comprises of 95% of the total data points i.e. 647 data points and testing set comprises of 5% of the total data point i.e. 32 data points. The dataset has not been split randomly because it is time series data and we would not like our testing set to have entries from the relative past of a particular data point that has already been included in the training set.

The figure below shows the graph of Root Mean Square Error vs Number of Epochs (or Iteration) for different values of η (learning rate). In the curve four different values of η, marked by different colours, have been tried (η = 0.001, 0.01, 0.1, 1). It can be seen that the optimisation converges early and comfortably before the 100th iteration for η=1. **We choose η as 1 for further analyses.**

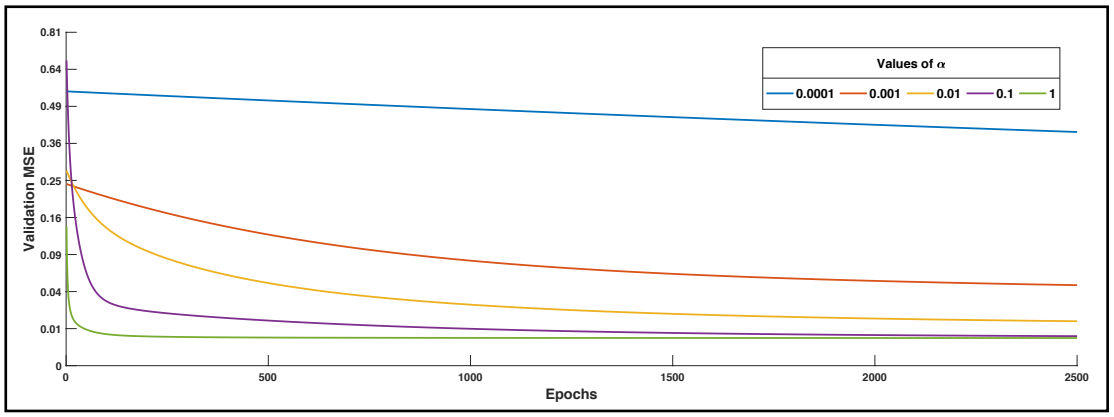


Figure 15: Validation MSE v/s No. Of epochs for various learning rates

It is observed that the training error exceeds the test error for this particular percentage split (95%-5%). We conclude that this occurs because of the relatively greater instability in the stock market over the time period of the training data points. There are two sudden and significant spikes in the time plot of the training set whereas on the other hand the testing set appears to be a lot more stable (i.e. - greater stationarity with less significant and less frequent shocks in the time plot.) This is the reason for a lower mean squared error over the final 32 data points of the time series.

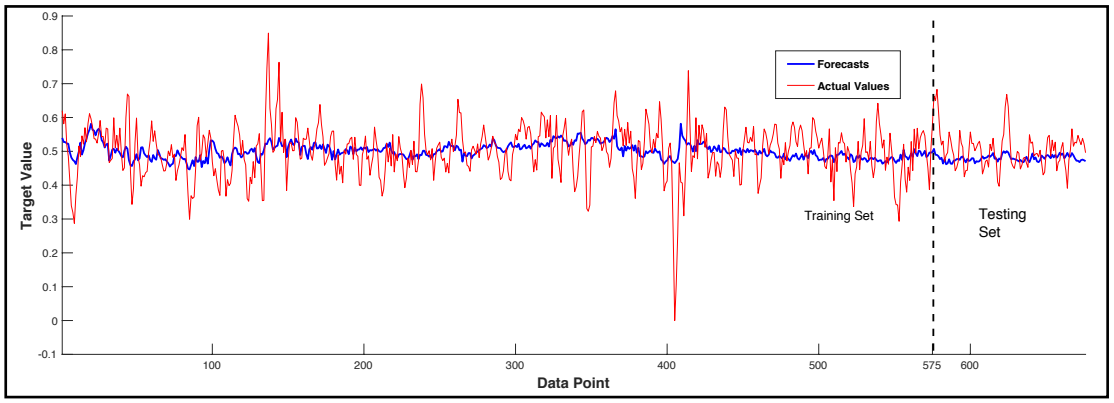


Figure 16: Time plot for actual and forecasted values

On changing the percentage split to 60%-40% for training and testing sets respectively, we observed that the testing error was now greater than the training error. The reason for this was that the spike near the 400th data point (refer to figure above) was included in the test set this time around, therefore resulting in a greater MSE. Normally, it is seen that the training error is greater than the testing error. This happens because in most of the cases, the variance of unseen data is more than the data we had trained on. The data we try to test on is different from the data on which we trained our model. In this case, as evident from the time plot,

the test set (the first case) has very less variance while the training set has much more variance with multiple big spikes in both the directions. In simpler terms, the test data patterns are way easy for the model to predict.

Result

The model with learning rate = 1 and no. of epochs = 5000, gave the following results:

Training MSE : 0.0056

Testing MSE : 0.0038

3.3.2 Neural Network Regressor

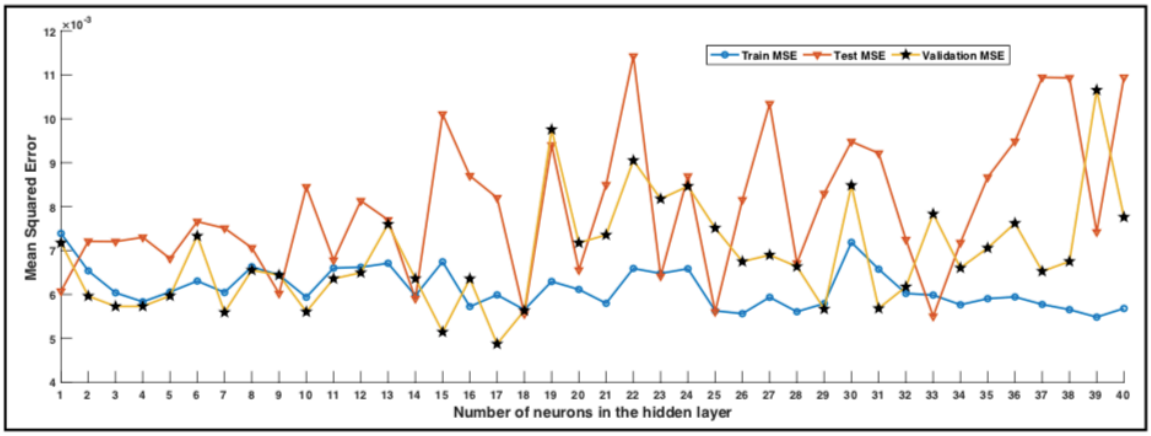
Back propagation neural network with following specifications (the architecture and parameters) was applied on the the dataset.

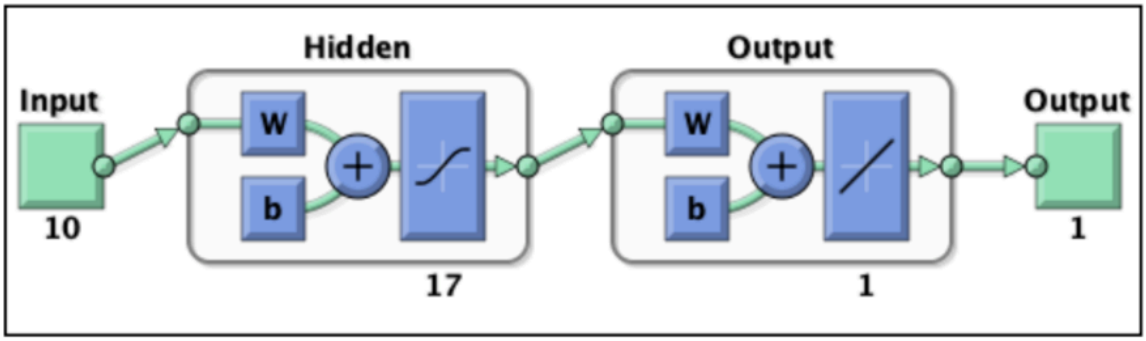
Entire data set was divided into three periods, the first one for training, the next one for validation and the last one for testing. The data has not been randomly divided so as to make sure that the test set does not predict values of the past. The real performance of the trained model can be judged on the basis of how well it forecasts which can be made sure by using those data points as testing tuples which are temporally later than the training or validation data points.

Number of hidden layers	1
Number of neurons in the hidden layer	1-40
Algorithm used	Gradient Descent Backpropagation
Activation function in hidden layer	Tan sigmoid
Activation function in output layer	Linear
Learning rate	0.1
Training Data points	Tuples 1 to 578, 578 tuples
Validation Data points	Tuples 579 to 647, 69 tuples
Testing Data points	Tuples 648 to 679, 32 tuples

Method used for tuning parameters	Using Validation set
Maximum Number of epochs	3000

Number of neurons in the hidden layer was decided using validation. Following plot shows the testing, training and validation MSE. It can be seen that minimum validation error is when number of hidden neurons is 17. We shall go with the same model. The architecture is drawn as well.





MATLAB functions:
train()
traingd feedforwardnet()

Result
The model with hidden units=17, gave the following results:

Training MSE : 0.00599
Testing MSE : 0.00180
Just as linear regression, the same error occurs here too and is rectified upon choosing more training points, with Train MSE = 0.0059 and Test MSE as 0.0065.

Discussion

Neural network regressor was observed to perform extremely well on the data. A different learning rate was tried but the model was not converging at other learning rate. An additional hidden layer was also added but the performance did not improve.

3.4 Future Work

Further analyses into Time Series will be done. Non-linear auto regressive models, which take into consideration the trends of previous outputs, will be tried. The advantage of this model would be that as we input the lag (previous outputs) into the model, trend will be analysed rather than the average of the adjacent values, which might result in better prediction. Then Ensemble methods would be tried on the dataset. The algorithms will also be applied again after the dataset has been put through Rough Set Data Analysis to reject superfluous features.

REFERENCES

[1] http://lib.stat.cmu.edu/datasets/boston

[2] D. Harrison and D.L. Rubinfeld 'Hedonic prices and the demand for clean air', *Journal Environ. Economics & Management*, vol.5, pp. 81-102, 1978.

[3] Leo Breiman, 'Random forests', *Machine Learning(Springer)*, vol. 45, pp. 5–32, 2001.

[4] N. Meinshausen. 'Quantile Regression Forests', *Journal of Machine Learning Research*, vol. 7, pp. 983–999, 2006.

[5] Y. Freund & R.E. Schapire. 'A decision-theoretic generalization of on-line learning and an application to boosting', *Journal of Computer and System Sciences* , vol. 55, pp. 119–139, 1997.

[6] H. Drucker, 'Improving regressors using boosting techniques', *14th International Conference on Machine Learning*, pp. 107–115, 1997.

[7] T. Hastie, R. Tibshirani, & J. Friedman, 'The Elements of Statistical Learning', *Springer*, Second Edition, 2001.

[8] L. Han, M. Embrechts, 'Random Forests Feature Selection with K-PLS: Detecting Ischemia from Magnetocardiograms', *European Symposium on Artificial Neural Networks*, pp. 26–28, 2006

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, 'An Introduction to Statistical Learning', *Springer New York*, vol. 103, 2013.

[10] Robert Tibshirani, 'Regression Shrinkage and Selection via the Lasso', *Journal of the Royal Statistical Society B,* Vol.58, Issue 1, pp. 267-288, 1996.

[11] Liu, C.D., Wang, J.H., Xiao, D. and Liang, *'Forecasting S&P 500 Stock Index Using Statistical Learning Models'. Open Journal of Statistics*, 6, 1067-1075, 2016.