Statistical Learning: Regression & Classification Assignment 1

STA5077A - Supervised Learning

Lecturers:
Şebnem Er
Stefan Britz

Due: 2019/04/05

Chris Maree - CHRMRX013

Abstract

This paper examines the design, implementation and testing of a number of statistical learning methods to perform regression and classification within two different contexts. In the first section popular regression techniques are applied to build models to predict the median value of a house in Boston suburbs based on 13 explanatory variables. A basic linear model is created and then refined using variable selection methodologies like ridge, lasso and elastic net regression. Lastly, an overgrown then pruned regression tree is implemented. Repeated Cross validation preformed on elastic net regression was found to yield the lowest overall test MSE of 11.3137 when an iterative interaction term selection process was applied. In the second section, a collection of different classification models are created to identify spam email. Logistic regression, discriminant analysis, random forest and boosted trees are all applied to this problem, with the best methods being random forest and boosted trees, with an test detection accuracy of 94.2%. These best models are then refined considering the implications of miss-classification using threshold modification.

Contents

1 Introduction

This report is broken into two main sections that explore regression and classification problems for given problem statements. Each section will begin with breaking down the problem statement, looking at assumptions and specifications and then move onto analyzing the datasets provided. Next, a number of models are created, tested and analyzed to solve the provided problem statements. The best model is discussed and critically analyzed when appropriate. A detailed code appendix can be found at the end of the report in the form of commented and annotated R markdown notebooks. In addition, this source code can be found on Github here.

Regression and classification forms the basis for a large proportion of statistical learning and has real world applications in many facets of daily life. An initial high level overview of regression and classification and the differences between them is valuable in understanding the sections that follow.

Regression problems involve predicting a missing element of a dataset. The output variables take on a continuous value and the goal is to predict a numerical response for an unknown input. An example of this kind of problem could be predicting a student's height based off a number of their known characteristics like age, weight, diet or gender. Regression is also useful in quantifying the relative impact of different characteristics on the desired response variable, such as examining how diet affects the average student height.

Classification problems on the other hand involve determining a class (or category) of an element in a dataset. The output variables take on discrete class labels and the goal is to identify the group membership of an unknown input. A simple example would be classifying a customer as someone who is likely (or not) to default on a loan based on information around their spending habits, credit card balances and history.

2 Question 1: Predicting House Values in Boston Suburbs

The goal of this first question is to derive a number of different regression models to predict the median value of owner occupied homes for suburbs in Boston. To this end, a number of different models were created and tested with model accuracy quantified by means of a test MSE.

2.1 Data Exploration

The dataset provided has 12 predictor variables with one response variable. A total of 506 rows are present in the dataset with each row corresponding to a different suburb in Boston. Not all the data present in the dataset is used in the creation and testing of the model but rather a random subset of 400 rows provided for each student. A full breakdown of all the variables within the dataset can be seen in Table 1 where the class of each variable was extracted using the `class` function applied over all rows within the Boston dataset dataframe with `sapply(Boston,class)`. These variable descriptions were taken from [1]. The column index for each variable was appended to the table for ease of reference during the report and is not actually part of the dataset.

Index	Variable	Class	Description
1	crim	numeric	per capita crime rate by town
2	zn	numeric	proportion of residential land zoned for lots over 25,000 sq.ft.
3	indus	numeric	proportion of non-retail business acres per town.
4	chas	integer	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
5	nox	numeric	nitrogen oxides concentration (parts per 10 million).
6	rm	numeric	average number of rooms per dwelling.
7	age	numeric	proportion of owner-occupied units built prior to 1940.
8	dis	numeric	weighted mean of distances to five Boston employment centres.
9	rad	integer	index of accessibility to radial highways.
10	tax	integer	full-value property-tax rate per $10,000.
11	ptratio	numeric	pupil-teacher ratio by town.
12	lstat	numeric	lower status of the population (percent).
13	medv	numeric	median value of owner-occupied homes in $1000s.

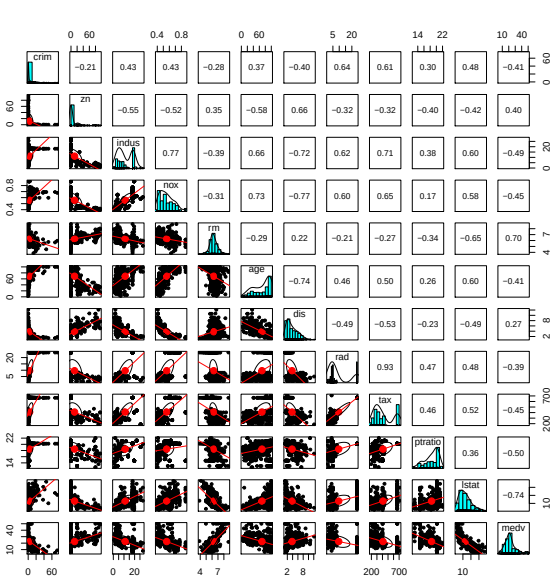
Table 1: Boston housing data variables broken down with classes and descriptions

From Table 1 the predictor variables can be seen as index $1 \rightarrow 12$ and the response variable(`medv`) as index 13. The goal of the regression conducted will be to predict the response variable `medv` with the other 12 predictor variables. The data type of each variable is shown in the class column where numeric is a representation of a double precision floating point number (or double) and integer represents whole numbers. As `R` is a weakly typed language, these types were inferred during the import of the data from the provided CSV file.

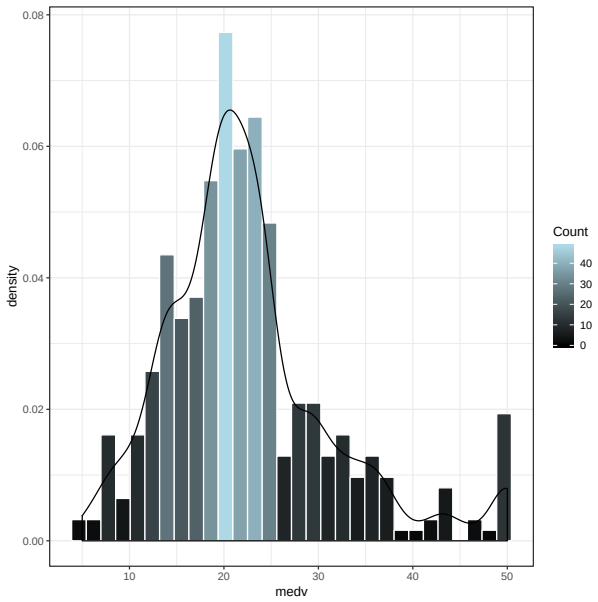
The data provided for this question was read in using the CSV reader in R and then 400 unique samples were taken from the 506 rows in the provided `boston.csv` data set. The provided seed (26 in my case) was used to generate a unique set of results to be used in this question. For the remainder of this question this subset of 400 sampled rows will be treated as if it was the full set of data.

2.1.1 Data Visualization

Before jumping right into modelling it is important to look at the dataset to visually to identify patterns and trends. Figure 1a shows the relationships within the dataset by plotting each variable against each other. Below the diagonal scatter plots are shown with interactions between variables. The main diagonal has histograms for each variable and the Pearson correlation is shown above the main diagonal. We are primarily interested in the extreme bottom and extreme right rows of this plot that show the interaction graphs and Pearson correlation for `medv` onto the predictor variables. A number of key interactions with



(a) Pairs plot of provided Boston Dataset



(b) Histogram of all the median house values

Figure 1: Basic linear model predicted vs actual and variable importance plots

the response variable can be seen immediately from looking at the Person correlations and shape of the associated graphs. For example, a strong relationship is visible between `medv` and `lstat` with a seemingly linear distribution in the plot and a high Pearson correlation of -0.74. Other variables also show strong correlation like `rm` with a very linear pattern show in the plot. These interactions, among others, can be used later when generating the regression models.

Figure 1b shows a histogram of all median values of all the houses within the dataset, providing a visual representation of the distribution. This is valuable as one can see that the general shape is somewhat normal but there are some clear outliers on the extreme right of the dataset that skew the histogram somewhat.

2.2 Quantifying Model Fit and Accuracy

For all subsequent parts of this question the quality of a model fit is quantified by means of a Mean Squared Error(MSE). This quantity represents the average difference between the estimated and actual values for a given set and is defined in Equation 1. A generic implementation of this equation was defined within the R Markdown notebook which can be seen in Appendix 1.3 and is used at all future points to calculate the MSE in a concise, reproducible way. This function is more of a syntactical sugar than anything else.

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

2.3 Question 1.A: Data Pre-processing and Multiple Linear Regression Model

The first part of this question involves preforming some pre-processing on the provided data to generate training and testing sets. After which a basic linear model is generated and analyzed.

2.3.1 Data Splitting

The first step when preforming any kind of regression is to split the provided data into training and test sets whereby the training set is used to build the model and the test set is left until the very end to test the final accuracy of the model. An important detail in using the training and test data is that the test

set not be used at any point during the training of the model as this will bias the test results unfairly towards the test set, thereby giving overly inflated accuracy predictions. The data was split in an 80/20 ratio for training and test data, providing 320 records for training and the remaining 80 for testing.

2.3.2 Basic Multiple Linear Regression Model

To begin exploring the relationship within the dataset, a simple multiple linear regression model was fitted to the training set, regressing `medv` onto all explanatory variables with the `lm` function. This is effectively finding a linear combination of the predictor variables that results in the lowest possible MSE for the given training set. Multiple linear regression creates a model of the form expressed in Equation 2 by finding the regression coefficients β_0 through β_p for p distinct predictors.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon, \tag{2}$$

The coefficients are found through the minimization of the residual sum of squares (RSS) of the model. This process is known as ordinary least squares(OLS). In other words, the β's that yield the lowest prediction error when applied on the training set are used as the final model. The RSS cost function used by OLS that is minimized in the multivariate case is shown in Equation 3.

$$\begin{aligned} RSS &= \sum_{i=1}^{n} (y_i - \hat{y}_0)^2 \\ &= \sum_{i=1}^{n} (y_0 - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \ldots - \hat{\beta}_p x_{ip})^2 \end{aligned} \tag{3}$$

2.3.3 Basic Linear Model Fit Analysis

The initial model generated yielded a training MSE of 21.935 and a test MSE of 25.6233. The model implementation can be found in Appendix 1.3.1. By looking at the `p` values within the `summary` of the model generated, one can identify variables that had a large impact on the response. From this, the key predictor variables can be identified to be `nox`, `rm`, `dis`, `ptratio` and `lstat`. The R-Squared values from the summary represent how much of the training set variation has been expressed through the linear model. This yields a value of 0.6939, indicating that 69.39% of the variability in median value of houses has been expressed.

There are a number of other metrics that can be used to quantify model fit over and above the MSE. One such metric is the *Breusch-Pagan test against heteroskedasticity*[2]. Heteroskedastic data is data with non-constant errors present and this violates the required assumptions for running linear regression. Rather, it is required that the data is Homoskedastic, meaning that the error terms are randomly and normally distributed. This is a requirement as least squares (OLS) regression assumes that all residuals are drawn from a population that has a constant variance (homoskedasticity). This test can be preformed in R using the `bptest()` function as part of the lmtest package. The p-value obtained from the test indicates the heteroskedasticity of the model. Running this on the basic linear model yielded a value of 2.205×10^{-5}, indicating that the dataset is homoskedastic.

Another important test to run on the model is to identify the variable inflation factors to check for *multicollinearity*[3]. This is a measure of correlation within the explanatory variables and checks if any of the variables can be linearly predicted as a function of the other variables within the set. This check can be done using the `vif()` function in R, from the car package. The results from this test can be found in Table 2 wherein the problematic variables ($vif > 5$) have been identified in grey. Exclusion of these variables is left for a later question wherein it is considered.

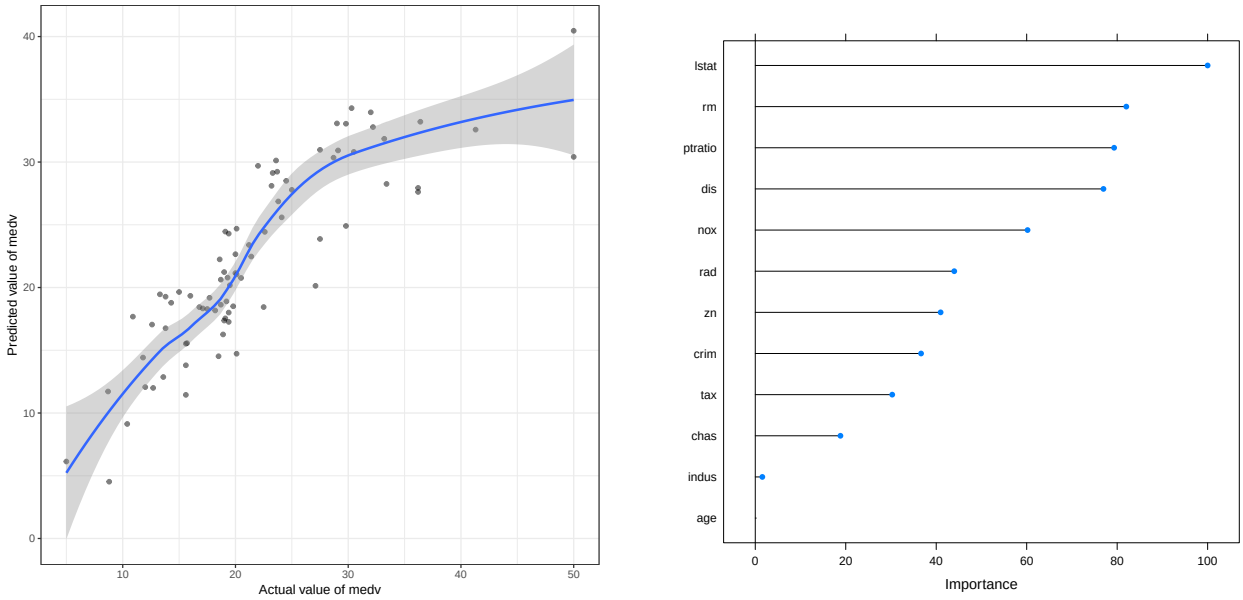
zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	lstat
2.247956	4.0101	1.060881	4.424053	1.928305	3.154613	3.667428	8.186654	10.313219	1.726377	2.86219

Table 2: Basic linear model variance inflation factor for each predictor variable

Another useful plot to quantify the fit of a model is to look at the predicted vs actual values of the model. This kind of plot will show how well the model captured the response variable variation over all actual values of the response. Figure 2a shows this representation. An interesting component of this diagram is the clear non-linearity within predicted vs actual plot showing some element within the data that is missed within the model.

2.3.4 Basic Linear Model Variable Importance

Each variable's relative importance within the model can be seen in Figure 2b which shows the importance of each variable normalized between 0 and 100, with the most relevant variables depicted at 100. This diagram represents the same information as shown in the p-values for the test but provides a nice graphical way of representing it, showing clearly which variables have the largest impact on the response. From this Figure some of the most important variables can be seen to be `lstat`, `ptratio`, `rm` and `dis`.



(a) Predicted vs actual plot

(b) Variable Importance lolipop plot

Figure 2: Basic linear model predicted vs actual and variable importance plots

2.4 Question 1.B: Improved Multiple Linear Regression Model

The linear model previously proposed was improved through a number of different variable selection techniques and the inclusion of interaction terms. The improvement in model performance was quantified through the reduction in training and test MSE. The code implementation for this question can be found in Appendix 1.5. A detailed residual diagnostics was then preformed to analyze model behaviour.

Two broad methods were employed in selecting the interaction terms. Firstly a number of logical interaction terms were selected by analyzing the predictor variables and selecting those that looked like they might have an interaction based off domain knowledge. These interaction terms can be validated through variable selection methods such as lasso wherein irrelevant terms will be removed from the model. The second process for introducing interaction terms is more iterative and involves fitting the model to all possible interaction terms and then using variable selection methodologies like lasso to remove the irrelevant terms. One can then plot the variable interaction terms significance and infer the meaning from

the most significant terms. This process yields a higher accuracy in model performance but sacrifices the model's ability to be interpreted, as higher order interaction terms can not necessarily have their meaning deciphered.

2.4.1 Basic Variable Selection With Lasso

Lasso stands for *least absolute shrinkage and selection operator* and is a method of variable selection and regularization. Lasso acts to enhance model prediction accuracy while removing irrelevant terms through the minimization of mean squared error. This is done in a similar fashion to that of least squares but includes a penalty term for the number of coefficients present within the model. A tuning parameter λ is used to control the degree of influence of the coefficients and can be seen in Equation 4. Lasso aims to minimize the error produced in Equation 4 through the selection of the lasso coefficients $\hat{\beta}_\lambda$. An ideal λ will choose a model that is complex enough to capture the important features in the dataset but not too complex as to result in high model variance. Cross validation can be used in conjunction with Lasso to select the ideal λ values.

$$MSE = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{4}$$

2.4.2 Logical Interaction Terms Selection

Model improvement begins by selecting terms within the data set that are likely to have interactions based on domain knowledge. From analyzing the dataset six interaction terms were selected and analyzed. The rationale for the selection of these interaction terms is first broken down and then their validity is tested using lasso regularization. Clearly it is possible that there are other useful interaction terms that were missed in this selection process. Likewise, it is possible that one of the interaction terms identified in fact provides no value in the model and is removed via a variable selection technique. Each identified interaction term pair is described in detail below.

1. **lstat ↔ rm: lower status of the population & average number of rooms per dwelling**
   There could be an interaction between the economic status of the house inhabitants and the number of rooms of the house. This is to say that people with a high economic status (high net income) that live in a house with many rooms is a very different configuration to people with a low economic status (low net income) with many rooms. For the former (high net income, many rooms) one can logically expect a large, expensive house with lots of guest rooms and amenities that would have a positive effect on the value of the house. Conversely for the case of the latter (low net income, many rooms) one would expect a smaller rooms in a lower valued building that accommodates more people. This setup would result in a lower (or negative) multiplier effect on the value of the house. This interaction term is therefore expected to be negatively correlated with the response.

2. **rm ↔ dis: average number of rooms & mean of distances to five employment centres**
   Along a similar vain of logic as 1, one could expect that the number of rooms within a household would be correlated to the distance to employment centres. This is based off distance being a factor that influences the "impactfulness" of the number of rooms within the household. A house with many rooms close to an employment centre is different to a house with many rooms far from an employment center. The former would potentially house many people seeking cheap rent close to work, while the latter could potentially be the house of a higher economic status family with a large house far from the city. This interaction term is expected to be positively correlated with the response.

3. **dis ↔ rad: mean of distances to five employment centres & index of accessibility to radial highways**
   One could expect that the ease of access to highways would influenced the mean distance to employment centres. Living far from an employment centre could mean that you require higher access to

radial highways to provide the required transport infrastructure to get to work. Conversely, if you are closer to an employment centre you might not be as close to a radial highway. This interaction term is expected to be negatively correlated with the response.

4. **nox ↔ rad: nitrogen oxides concentration & index of accessibility to radial highways**
   The index of accessibility to radial highways could result in higher levels of nitrogen oxide in the environment due to being physically closer to the highways. This interaction could have a negative impact on the value of a house within an area, with households that are close to the highway with high levels of nitrogen oxides being lower valued. This interaction term is expected to be be negatively correlated with the response.

5. **crim ↔ lstat: per capita crime rate by town & lower status of the population**
   The economic status of an area could impact the crime per capita of that area with lower status areas having higher crime rates. This is to say that the impact of a high crime and low status of a population together will have a negative impact on the value of houses within an area. This interaction term is expected to be be negatively correlated.

6. **tax ↔ lstat: full-value property-tax rate & lower status of the population**
   Lastly, the value of a house given by the property-tax and the socio-economic status is expected to interact wherein low status suburbs with high property tax will result in lower house valuation. This interaction term is expected to be be negatively correlated with the response.

Now that the interaction terms have been outlined, a model can be fitted to include these interaction terms and a lasso regularization can then be preformed to select only the most valuable variables. This acts to validate the selection of interaction terms by removing the irrelevant pairs of terms. The model generated using the `glmnet` function that fits generalized linear model via penalized maximum likelihood. Cross validation was then applied to the model using the `cv.glmnet` function to test out a range of λ values, ranging from 1×10^{-5} to 1×10^{10} in with 1000 steps between the top and bottom of the range.

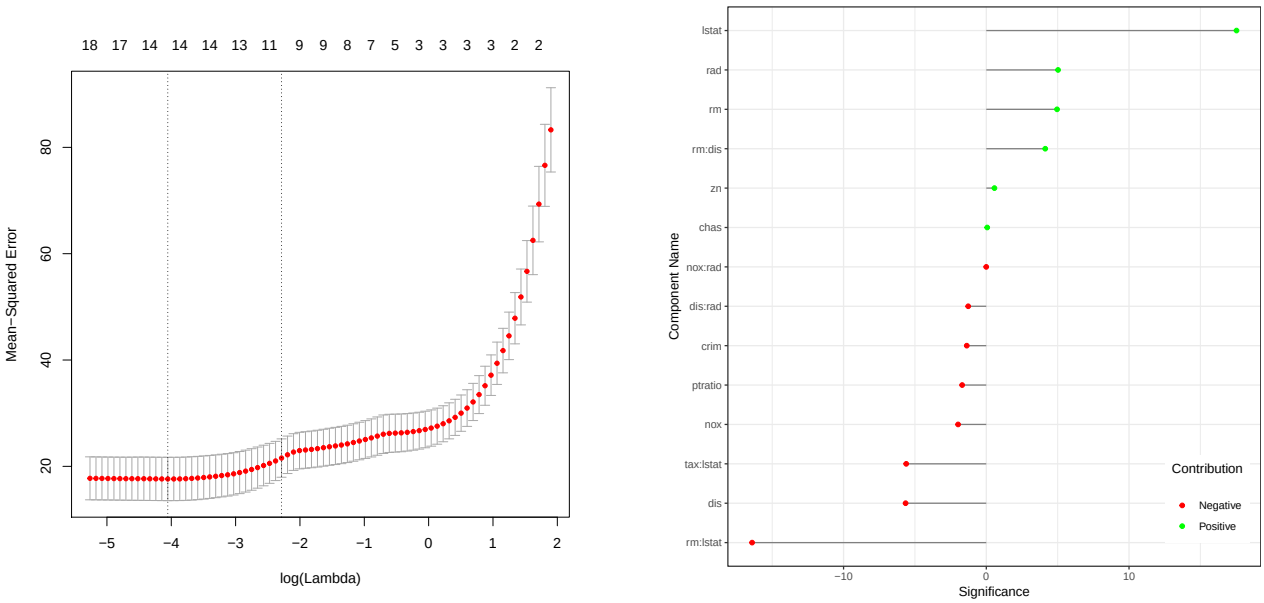
2.4.3 Normalization of Model Parameters and Variable Significance Visualization

In order to use lasso as a variable significance measure all parameters must first be normalized before fitting the model. This process involves standardizing the units between predictor variables. If this is not done the model coefficients can't be compared as a means of quantifying variable significance. To this end the input data to the lasso model was scaled using the `scale` function in `R` and the parameter `standardize=FALSE` was set when running the `glmnet` function. This acts to standardize the variables such that each variable has a mean of 0 and a standard deviation of 1. As a result, these coefficient magnitudes can be used to quantify variable significance without modifying the predicted output magnitude.

As the input variables to lasso have been normalized one can now plot the the magnitude of each input terms coefficients and use this as a method to quantify variable significance. To this end, a function was created called `significancePlot` that takes in the coefficients of the lasso model at a specified λ value, preforms some data processing to manipulate the data into a data frame, removes redundant information like intercepts, orders the rows by significance levels and then plots it using ggplot. The plot output of this function is used in analyzing the interaction terms in the what follows.

2.4.4 Logical Interaction Terms Lasso Performance

Preforming lasso regression with the 6 interaction terms, cross validating the results and using the λ_{min} value in predictions yields a model with a cross validation MSE of 19.20169, a training MSE of 14.62935 and a test MSE of 14.7489. This is a drop of 39% in MSE training error when compared to a lasso preformed on all predictor variables with no interaction terms. The output of the lasso can be seen in Figure 3a. This Figure shows the mean-squared Errors for different λ values and the associated number of variables that would be included in the model for that given λ value. The left dashed line indicates



(a) Lasso output plot

(b) Variable importance lolipop plot for lasso

Figure 3: Logical interaction terms with lasso selection preformed

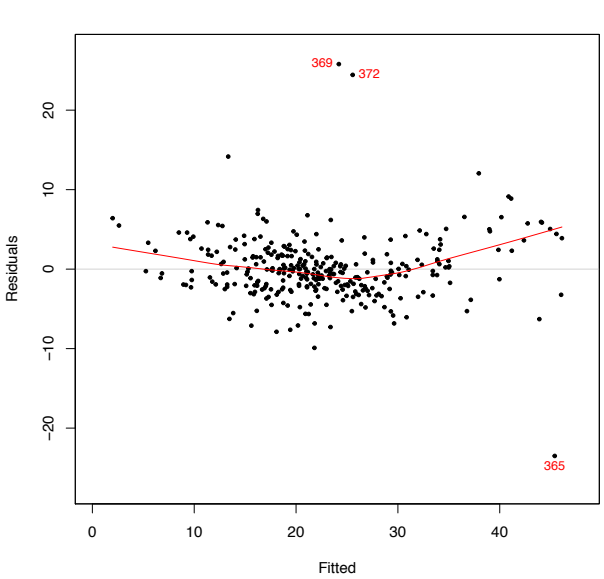
the value λ_{min} that minimizes the out-of-sample loss in cross validation. The right hand dashed line indicates the value λ_{1se} which is the largest λ value within 1 standard error of λ_{min}. From this graph we can see that the ideal model using the interaction terms previously specified should have between 10 and 14 terms within the model.

To identify what terms are deemed important at λ_{min} Figure 3b can be analyzed. This Plot shows the relative importance of different terms within the model and can be used to quantify the value of each parameter included. Terms that have a net positive correlation with the output are shown in green and terms with a negative correlation are shown in red. There are a total of 14 variables shown on the plot out of the possible 18 total (12 predictor and 6 interaction) with 4 variables being removed imidatly by lasso. These removed variables are: `indus`, `age`, `tax` and `crime:lstat`. Interestingly lasso removed tax while keeping it's interaction term with `lstat` in `tax:lstat`. Figure 3b can be used to identify terms and interactions that do not contribute to the overall fit quality and simply result in unnecessary model complexity (over and above those already removed). The most significant interaction terms from those proposed are: `rm:lstat`, `rm:dis` and `tax:lstat`. The other interaction terms did not provide a large contribution to the overall model quality such as `crime:lstat`, `nox:rad` and `dis:rad`. This result validates the hypothesis made for the high significance interaction terms and disproves it for the others.

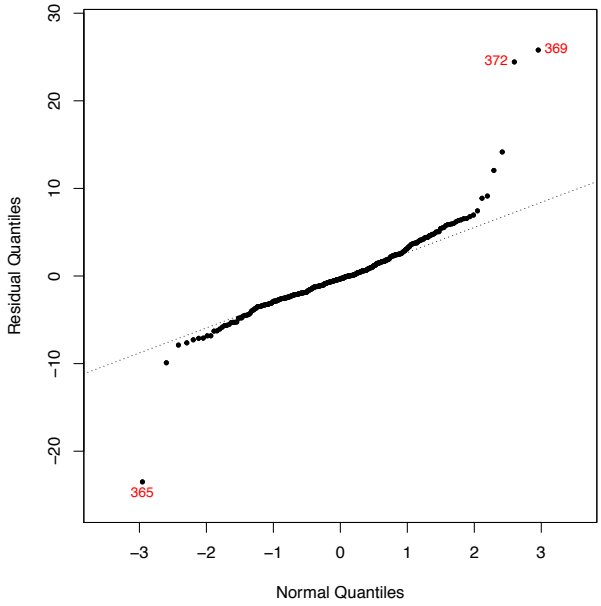
2.4.5 Logical Interaction Terms Residual Diagnostics

A residual diagnosis was preformed on the generated cross validation model. Two key figures from this diagnosis can be seen in Figure 4, primarily the residual vs. fitted plot and residual QQ plot. These figures are used to quantify the quality of the regression model. A residual is the difference between the observed (y) and predicted ($\hat{y}$) value of the dependent variable. The residual vs fitted plot as can be seen in Figure 4a and tests the assumptions of whether the relationship between the variables is linear and the whether there is equal variance along the regression line. These two properties are used to quantify the linearity and homoscedasticity of the model. The residual *quantile-quantile* (QQ) plot in Figure 4b helps determine if the dependent variable is normally distributed by plotting quantiles from the given data set against a known theoretical distribution, such as a normal distribution and can be seen in Figure 4b.

An ideal residual vs fitted plot should not have any defining shape or pattern, be generally symmetrically distributed around the 0 line and have no obvious outliers[4]. Figure 4a embodies much of this



(a) Residual vs fitted plot



(b) Residual QQ plot

Figure 4: Refined logical interaction term lasso output residual diagnosis

description with a somewhat symmetric distribution around the 0 line. This can be seen by looking at the red line drawn through the plot depicting the fit of the residuals vs fitted spread. Ideally this red line should be as close to the 0 line as possible. The red line is quite curved indicating some non-linearity in the residuals showing complexity in the underlying data that was not captured in the model. There are some clear outliers in the residual plot. The top three have been identified and numbered with red letters, such as data points 372, 369 and 365. These points could have had a negative impact on the quality of the model and further refinements of the fit should involve filtering out these outliers before fitting the regression.

Figure 4b also embodies much of the expected trends for a well behaving QQ plot. There are however the same outliers present as shown in Figure 4a that could have negatively effected the quality of the model fit. The large spike in magnitude on the tail end of the residual QQ plot primarily with value 372 and 369 could potentially account for the large spike as seen back in Figure 1b on the tail end of the histogram plot of `medv`.

2.4.6 Iterative Interaction Term Variable Selection

In an effort to further improve this model a more iterative process was employed to find ideal interaction terms that may have been missed by simply looking at the dataset and relying on domain knowledge. This process involves generating a model using all possible interaction terms up to a particular degree then using different variable selection techniques to remove the irrelevant terms and interactions. For the 13 predictor variables, if all possible second order interaction terms are fitted then there will be $\binom{13}{2} = 78$ possible interaction terms fitted and then regularized. In the case of third order interactions $\binom{13}{3} = 286$ total terms will be included. The code for this itterative process can be found in Appendix 5.

Each iterative variable selection process will discussed and then the results will be compared in a concise tabular format. `R` provides a simple way to include all interaction terms within a linear model's creation by defining the `formula` parameter within the model to `medv ~.^2`. This syntax instructs `R` to regress all variables and all possible interaction terms onto `medv`.

A number of different variable selection techniques were applied to the expanded set, including ridge, lasso, elastic net regression as well as step wise model selection by AIC. Ridge regression is similar to

lasso, defined in Section 2.4.1 except it does not do variable elimination when regularizing but rather reduces irrelevant variable coefficients to a lower magnitude. Elastic net acts as a hybrid between ridge and lasso and tries to capture the best elements of each process by imposing both sets of variable penalization. Equation 5 shows the MSE that an elastic net regression tries to minimize. Importantly this expression captures the penalty term for ridge regression ($\sum_{j=1}^{p} \beta_j^2$) and the penalty term for the lasso regression ($\sum_{j=1}^{p} |\beta_j|$). Two tuning parameters are present: λ that controls the significance of the overall penalization and α that makes the penalization move between a lasso regression ($\alpha = 1$) to a ridge regression ($\alpha = 0$).

$$MSE = \sum_{i=1}^{n}(y_i - \hat{y_i}) + \lambda \left((1-\alpha) \sum_{j=1}^{p} \beta_j^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right) \tag{5}$$

In an effort to further the model quality, the caret package was used and repeated cross validation was preformed within each model's selection process for the tuning parameters. $K = 10$ folds with 5 repetitions for each cross validation set was preformed. Repeated cross validation is done by shuffling the data prior to each repetition, which results in a different split of the sample being used for each round of cross validation. This is done to reduce the effect of one variable being dependent on another and to reduce the estimator bias. This process does however result in a rather computationally expensive process as each model fitted requires an additional 10 iterations for each cross validation, multiplied by an additional 5 sets for each repetition. For each regularization technique preformed a range of tuning parameters (λ for ridge and lasso and λ & α for elastic net) were tested and the ideal values selected using cross validation. 150 λ values were tested for lasso and ridge and 5 λ and α values were tested for elastic net.

Lastly, stepped AIC (Akaike information criterion) was also preformed which involves iterating through all possible models, including all possible interaction terms, and then selecting the best possible subset based on the AIC criteria. This process is the least computationally efficient of all the models preformed as it is requires all possible models to be fitted onto the dataset. This process was also done using repeated cross validation from `caret` and resulted in high computation time to find the ideal model.

Each model generated will not be analyzed in detail here but rather the final results for each variable selection method will is shown in Table 3. These results were captured using a data frame in `R` to store the results from each model after generation. Both the training MSE and test MSE were captured for each method to provide relative points of comparison.

model Name	Interaction order	Terms+Interactions	Picked Terms+Interactions	Training MSE	Test MSE
Stepped AIC	3	286	41	7.35243	14.15175
Cross validated lasso	2	78	73	7.721433	11.64391
Cross validated elastic net	2	78	76	8.202387	11.31370
Stepped AIC	2	78	30	8.428501	12.75251
Cross validated elastic net	3	286	135	10.785709	11.31370
Cross validated lasso	3	286	40	11.881763	15.34633
Cross validated ridge	2	78	79	15.999352	17.36434
Stepped AIC	1	13	11	22.651833	19.57705
Cross validated lasso	1	13	12	22.715144	19.45775
Cross validated ridge	1	13	13	23.255275	19.83823

Table 3: Comparison of different iterative interaction term selection methods showing test and training errors. Table is sorted by training MSE. The models discussed are highlighted in grey.

Analyzing these results shows some interesting trends and behaviours in the model selection techniques. All models that included interaction terms were better than models that did not include interaction terms. This shows the *power* that interaction terms can have within models and thus the importance of selecting appropriate interactions.

The model with the lowest training error was Stepped AIC with 3rd order interaction terms. This model included 41 terms, of which 24 were first order interaction terms and 5 were third order interaction terms. The delta between the very low training MSE and the much higher test MSE (especially relative to the rest of the results) suggests that some degree of over fitting has occurred and the complexity in this model makes it a poor candidate.

The second best training MSE, cross validated lasso with second order interaction terms yielded the second lowest test MSE with a value of 11.64391. This model looks like it could be an ideal candidate to select as the best model from the set as it has low training and test MSE's relative to the rest of the results and only requires second order interaction terms. This makes the model simpler and potentially easier to interpret. This lasso model filtered the potential 78 terms and interactions down to 35 terms and interactions when the λ_{1se} was used however this λ value increased the error a substantial amount. To get the best prediction model λ_{min} is used resulting in the model picking 76 terms (as shown in Table 3). While this is still a reduction in terms, this model is very difficult to interpret due to the high number of interaction terms as they are based off nothing more than their significance in the model.

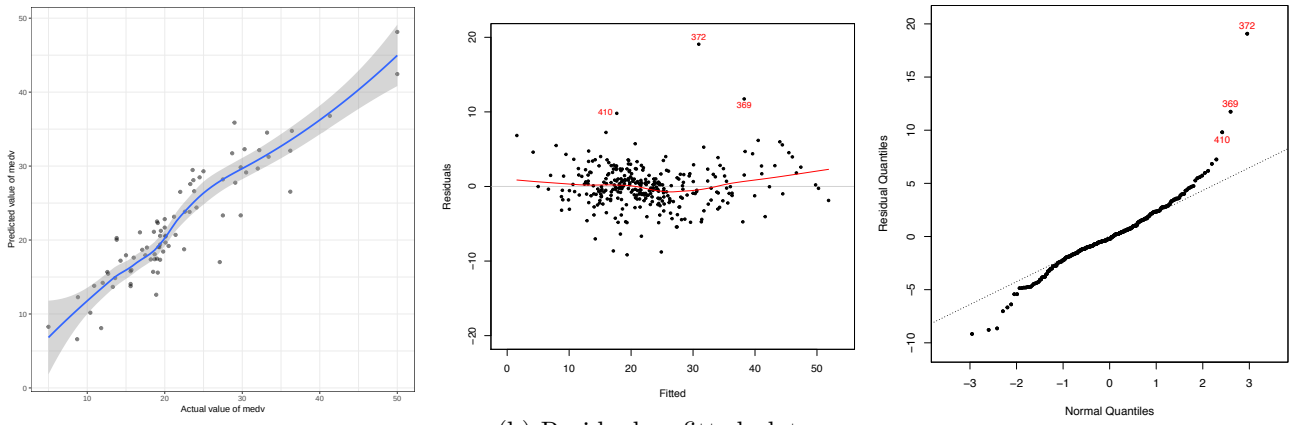
Along a similar vain is the cross validated elastic net model which yielded the lowest overall test error but did include an uninterpretable number of terms when the ideal λ and α values are used. If one wants the highest possible prediction accuracy this would be a good model to use but interpreting 76 terms, of which 63 are interaction terms is not trivial.

The last model of interest is the stepped AIC model that yield a very respectable testing and training error *while* having far fewer selected interaction terms than any other model of that order. This model contained 18 interaction terms and 12 predictor variables and so is still relatively complex but it is far simpler than the other models shown of the same order. This model is a good candidate when wanting high prediction accuracy without requiring an absurd number of interactions.

Ultimately the choice of model between the much simpler previously identified models or these more complex iterative models boils down to what you want from the model. This is a fundamental tradeoff between model interpretability and performance and each context will influence what model is chosen.

2.4.7 Iterative Interaction Term Model Residual Diagnosis

A residual diagnosis will be briefly discussed for the elastic net model with two interaction terms produced which showed a very low training MSE and the lowest testing MSE out of all the models. The key plots of intrest can be seen in Figures 5.



(a) Predicted vs actual plot

(b) Residual vs fitted plot

(c) Residual QQ plot

Figure 5: A collection of plots to visualize elastic net with second order interaction terms performance

To begin with Figure 5a shows the predicted vs actual plot for the elastic net model. When compared

to the equivalent plot generated for the basic linear model in Figure 2a it is clear that the elastic net has captured a lot more of the non-linearity present within the dataset, which is to be expected. The error bands are also much tighter now with the model displaying a higher quality fit to the dataset. Figure 5b shows the residual vs fitted for the refined elastic net model. There is a clear improvement in this plot when compared to Figure 4a which shows the equivalent plot for the lasso model with selected interaction terms. The red line is much closer to the zero line indicated that the residuals are more closely grouped around the zero line. This intern shows that the model has captured a higher degree of complexity. There are clearly still the same outliers present but this is to be expected and they will remain unless some kind of processing is done on the data. Figure 5c shows the residual QQ plot for the refined elastic net model and when compared to Figure 4b shows a tighter, more linear representation of the data indicating a closer fit to the desired normality line. There is still some degree of deviation from the normality line on the extremes with the same outliers effecting the overall plot.

2.5 Question 1.C: Regression Tree

Next, regression trees are used as a mechanism to predict the median house value. Regression trees can be considered as a variant of decision trees but instead of assigning outputs to categorical classes the output approximates a numerical function. Regression trees are built through binary recursive partitioning which iteratively splits the data into partitions. The tree is constructed top down wherein initially all records for the training set are placed within the same partition. Then, the algorithm iteratively splits the data based on minimizing the sum of squared deviations from the mean. This continues until each node becomes a predefined minimum size (stopping criteria) and becomes a terminal node. For the case of this question the regression tree is overgrown by relaxing the stopping criteria resulting in a large, overfit regression tree. This tree is then pruned down to a more appropriate size through the use of cross validation. The code for this question can be found in Appendix 1.6.

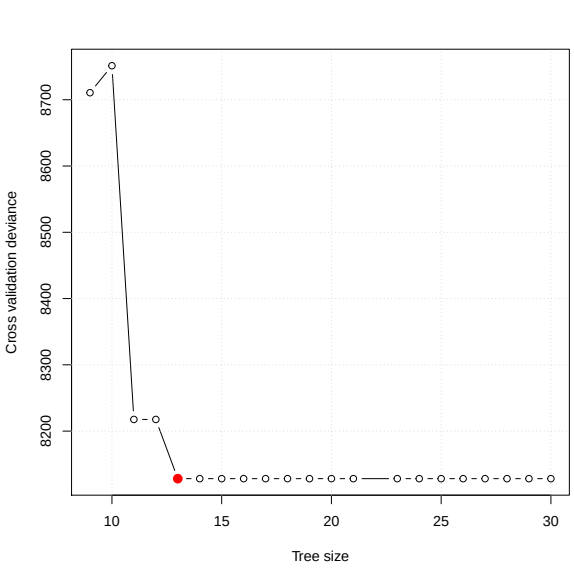
2.5.1 Building Large Regression Tree and Pruning it Back To Ideal Size

The process that was undertaken to grow a large regression tree and then prune it back will now be broken down. This process was initially implemented using the standard `tree` function within `R` and later implemented with the `Rpart` package.

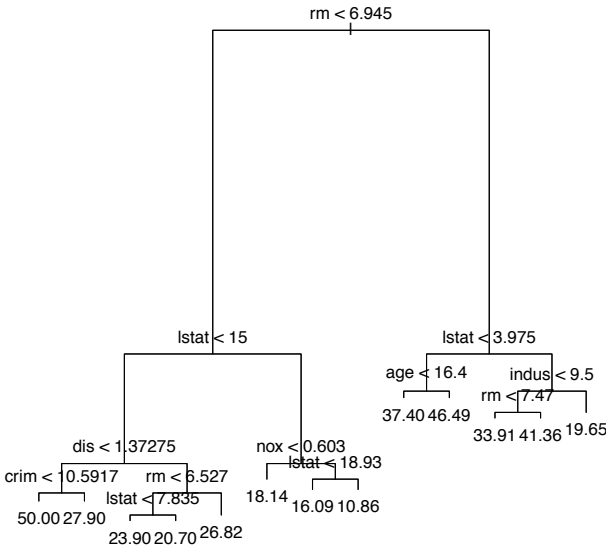
1. A large regression tree was grown, regressing all predictor variables onto `medv`. A custom `tree.control` was added that specified the minimum size that the tree can be to enforce that a large, over fit tree was grown. This is achieved through setting `nobs = 320` (the number of observations in the training set) and `minsize=1` (the smallest allowed node size). For this overfitted tree a MSE of 0.001854 was present for the training set, showing a near perfect prediction for the training data. This is to be expected as the tree is completely over fit to the training data and thus should perfectly predict the all values.

2. Next, cross validation was preformed using the `cv.tree` function, where `K=5` was specified indicating that 5 fold cross validation will be preformed. At this point a plot can be generated to show the cross validation deviance against the model size. The model size that shows the minimum deviance should be selected as the ideal tree size. From Figure 6a this can be seen as a tree size of 13.

3. The full overgrown tree can then be pruned back to the size that minimized the cross validation deviance. This pruned tree can be seen in figure 6b. This final pruned tree yielded a training MSE of 9.20114 which is clearly a large increase from the overfitted model as the tree is now generalizing.



(a) Regression tree cross validation



(b) Pruned regression tree

Figure 6: Regression tree cross validation and pruning results

2.5.2 Regression Tree Testing Results

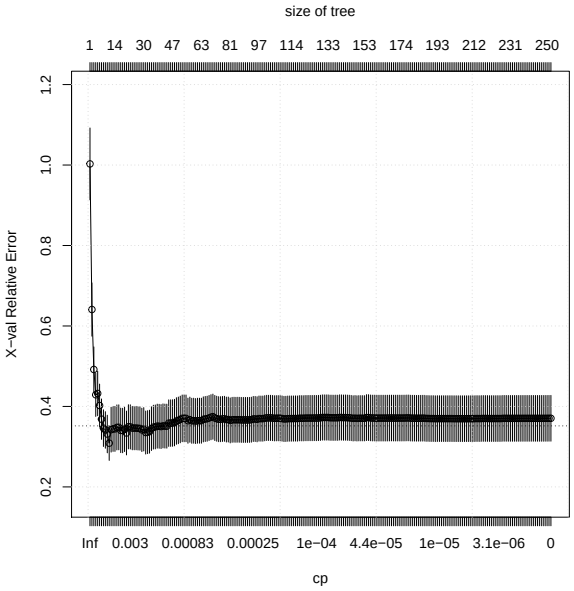
Now that an ideal tree size has been calculated using cross validation the test error can be calculated. The pruned tree yielded a test MSE of 20.83727 which is a fair amount higher than the MSE observed from the training set. This MSE is higher than most of the previously proposed linear regression non-tree based models which is to be expected as a single tree can't capture all the variance, subtleties and interactions within the set. This pruned tree did however preform better than a number of linear regression methods that did not include interaction terms. In fact the tree's test MSE is approximately the same as lasso, ridge, elastic net and stepped AIC when fitted onto the whole model, with no provided interaction terms as can be seen in the bottom four rows in Table 3.

2.5.3 Other Regression Tree Building Approaches and Libraries

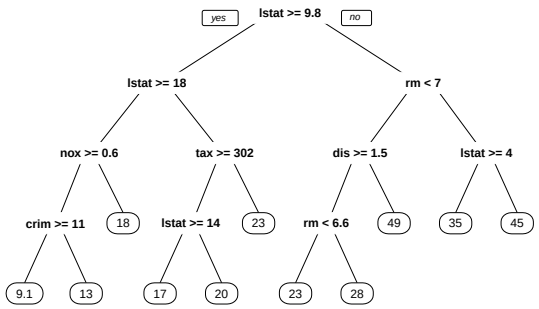
In an effort to improve the previously produced pruned regression tree `rpart` was used as an alternative method to provide a point of comparison. `rpart` was used to generate a regression tree and again cross validation was preformed to select ideal tree size for pruning. The tree size that produced the minimum `xerror` from `rpart`'s cross validation was chosen as the ideal tree size. The cross validation results can be seen in Figure 7a Then, the tree was pruned using the complexity parameter α as a mechanism to specify how the cost of a tree $C(T)$ is penalized by the number of terminal nodes $|T|$. This results in a regularized cost $C_\alpha(T)$ for a given tree $C_\alpha = C(T) + \alpha|T|$. This pruned tree can be seen in figure 7b. This regularization method and tree size selection technique resulted in a similar result with a test MSE of 23.89654 which is in fact higher than the deviance selection process previously outlined.

2.6 Question 1 Further Improvements

From a modeling perspective, there are a number of other methods that could have been applied to further improve the prediction accuracy such as random forests and boosted trees. From the perspective of linear regression within statistical learning an extensive number of models were generated when considering the iterative process undertaken. From a data pre-processing perspective there is a lot of room for improvement with outlier detection, removal and processing which could have improved the overall models performance. Different methods of data normalization should also be explored in future iterations and the impact thereof considered.



(a) Rpart regression tree cross validation



(b) Rpart pruned regression tree

Figure 7: Rpart regression tree cross validation and pruning results

2.7 Question 1 Conclusion

An number of different regression techniques were applied on the provided data set to build a model to predict the median value of houses in Suburbs of Boston. Interaction terms were then included to try and capture all variability within the data set and regularization processes were then applied to filter out irrelevant variables and interaction terms. Lastly, tree based methods were included as a means of preforming regression. These methods yielded an overall inferior result to linear regression models but provide simple and easy to interpret results.

3 Question 2: Classifying Email as Spam or Legitimate

The goal of this question is to explore a number of different classification techniques to identify email as either spam or legitimate. This process is preformed by looking at features in the content of the email and using this to build up classification models to categorize the email.

3.1 Data Exploration

Two csv files were provided for the case of this question, one with training data and a separate data set used exclusively for testing. Both text files were read into `R` using `read.csv`. Each dataset contains 59 columns. The first column is an index and this column was dropped from the dataset in `R` as this should not be part of any model. There are a total of 3601 emails in the training set, of which 2141 are legitimate emails and 1460 are classified as spam. A total of 1000 rows are provided for testing. Each column within the data is broken down in Table 4 below.

Index	**Class**	**Description**
1	integer	Row index. Stripped out for case of modeling
$2 \rightarrow 49$	real [0,100]	Represents a percentage of words in the e-mail that match that particular word
$50 \rightarrow 55$	real [0,100]	Identifies the high frequency occurring characters, representing the percentage of characters in the e-mail that match one a given character.
56	real [1,...]	Average length of uninterrupted sequence of capital letters.
57	integer [1,...]	Longest uninterrupted sequence of capital letters.
58	integer [1,...]	Total number of capital letters.
59	factor [0,1]	Class label. Indicates if the associated email row is spam or legitimate.

Table 4: Email data predictor variables column descriptions

3.2 Quantifying Model Fit, Accuracy and Quality

There are a number of ways to quantify the quality and fit of a classification model. The context that the classification model is trying to predict is important when considering which metrics to use. For the case of this question a number of key metrics will be recorded and used to compare each method. All of these metrics are calculated using a confusion matrix as the starting point. A confusion matrix can be constructed by tabulating the predicted vs actual values for each given model. Table 5 shows the construction of a generic confusion matrix from which all metrics can be calculated.

	Actual: $EMAIL$	**Actual:** $SPAM$
Predicted: $EMAIL$	True Positive(TP)	False Positive(FP)
Predicted: $SPAM$	False Negative(FN)	True Negative(TN)

Table 5: Email confusion matrix construction

To make the visualization and calculation of outputs for each metric easier the `confusionMatrix` function is used from the `caret` package. This is effectively the same as calling `table` on both the predicted and actual results and then calculating the true positive and negative rates. From this, 7 metrics of interest can be identified. Using the `confusionMatrix` function abstracts all this computation away to just calling one function. Results from each classification technique are stored within a `data.frame`, as was done in question 1, to make the comparison of models easier at the end. The list that follows outlines each of the important metrics that is stored for each model generated.

1. **True positive rate:** (Sensitivity) correctly identified emails $TPR = \frac{TP}{TP+FN}$
2. **True negative rate:** (Specificity) correctly identified spam $TNR = \frac{TN}{FP+TN}$
3. **False positive rate:** email is identified as not spam when it is spam $FPR = \frac{FP}{FP+TN} = 1 - TNR$

4. **False negative rate:** email is identified as spam when it is not spam $FNR = \frac{FN}{FN+TP} = 1 - TPR$
5. **Accuracy:** quantifies the overall model quality of fit $ACC = \frac{TP+TN}{TP+TN+FP+FN}$
6. **Kappa:** similar to accuracy, but considers selection that would happen through random predictions $\kappa = \frac{ACC-p_e}{1-p_e}$ where p_e is the expected accuracy from random chance; In this binary case $p_e = 0.5$
7. **F1:** considers both precision (percentage of results which are relevant) and recall (percentage of total relevant results correctly classified) to quantify quality: $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{2TP}{2TP+FP+FN}$

3.3 Introduction of a Validation Set

A large part of this question involves comparing and contrasting different models with each other based off their accuracy in predicting email class. This could be done by generating models on the training data and then picking the models that have the lowest testing classification error. The problem with this approach is that it will indirectly bias the results towards the test data and result in an over inflated final accuracy prediction. It is critical in this case that the data set used to compare and contrast each model is a different to the final testing set. Therefore, a validation dataset is introduced that contains 20% of the 3601 rows of training data and is used to give an estimate of model accuracy as a point of comparing while tuning the model's hyperparameters. In this way the training data is not touched right up until the end of the model selection process, thereby avoiding contamination and bias. An alternative approach could have been to use a cross validation error as the validation set but this does not work for all model types and so it becomes impossible to compare models that can't have cross validation preformed on them.

After each models validation error is calculated the error on the test data is also calculated but this value is not used until Section 3.8 wherein Question 2.e asks for an explicit test of the training data on all generated models. The two sets of model parameters for validation and test sets are stored using two custom functions created called `storeModelValidationPreformance` and `storeModelTestPreformance` which take in the model name and confusion matrix and then add the required values to the data frames which can be retrieved later on.

3.4 Question 2.A Logistic Regression Model

For the first part of the question logistic regression was used to build a model to predict the classification of emails. A number of variable selection techniques were then applied to try and refine the model by removing irrelevant terms. Forward, backward and bothways stepwise selection, followed by lasso are preformed on the logistic model created. The code for this question can be found in Appendix 2.3.

3.4.1 Basic Binary Logistic Regression Model

Logistic regression is a kind of generalized linear model (GLM) procedure that predicts the probability of a categorical outcome. As classifying email as spam or not spam is a binary outcome, logistic regression becomes Binary Logistic Regression (BLR) in the case of this question. Logistic regression does not assume any specific shape or densities for the predictor variables. In the basic binary case the outcome variable can take on one of two states, either 0 or 1. The model is defined mathematically in equation 6 where the independent variables X_i can be binary or continuous. The regression coefficients β_i are found using the maximum likelihood method.

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \sum(\beta_i X_i))}} \tag{6}$$

A simple logistic regression model can be found in `R` using the `glm` function and setting `family = binomial` to indicate that the output is a classification. The model can then be used to generate predictions for a new set using the `predict` function. This function takes in the generated `glm` model, a `newdata` set upon which the predictions are made and a `type` which indicates what kind of response is wanted from the prediction. To work back to a classification of email `type="responce"` is required and provides a probability prediction between 0 and 1 of the class falling in either the spam or not spam category. A

simple logic check can then be preformed to investigate if the magnitude for each prediction is above or below some classification threshold, say 0.5. For example if a prediction is made at 0.7, as this is above 0.5, it would be classified as a spam email. The impact of this classification threshold and how it can be modified will be discussed in section 3.8.2. A `confusionMatrix` can then be generated with the prediction and actual results. This basic binary logistic regression model yielded a validation accuracy of 0.9331.

3.4.2 Binary Logistic Regression Variable Selection Techniques

A number of variable selection techniques were then applied to refine and simplify the logistic regression model. The techniques used include: Forward, backwards, both ways, stepwise selection as well as lasso with normal and repeated cross validation. The basic theory of each variable selection process is first broken down, then a high level overview of its implementation is discussed. After, the results for all the variable selection techniques are compared in a tabular format.

Stepwise selection acts to improve a models accuracy through an automatic procedure wherein each step a variable is considered for addition or subtraction from the set of explanatory variables based on a predefined criteria. The `R` implementation of the stepwise selection is done using the `step` function which iteratively adds or subtracts variables from the model (based on direction) using the `add1` or `drop1` functions. In practice this involves taking the basic linear model defined in Section 3.4.1 and running the `step` function on it with the defined `direction` as either `"backward"`, `"forward"` or `"both"`. The process of finding the predictions is the same as outlined in Section 3.4.1 through the use of the `predict` function. In practice, stepwise selection is computationally inefficient and takes upwards of two minutes to run on the provided dataset.

Lasso with cross validation acts to find the ideal regression coefficients while imposing a penalty on the number of parameters included in the model. It can be preformed in `R` using the `cv.glmnet` function from the glmnet package where `alpha = 1` to indicates lasso and `family = "binomial"` indicates logistic regression. Predicting results is done in a similar way as before except now the minimum λ parameter needs to be specified for the `predict` function as `s = cv.lasso$lambda.min` for a model names `cv.lasso`.

3.4.3 Binary Logistic Regression Variable Selection Testing Results

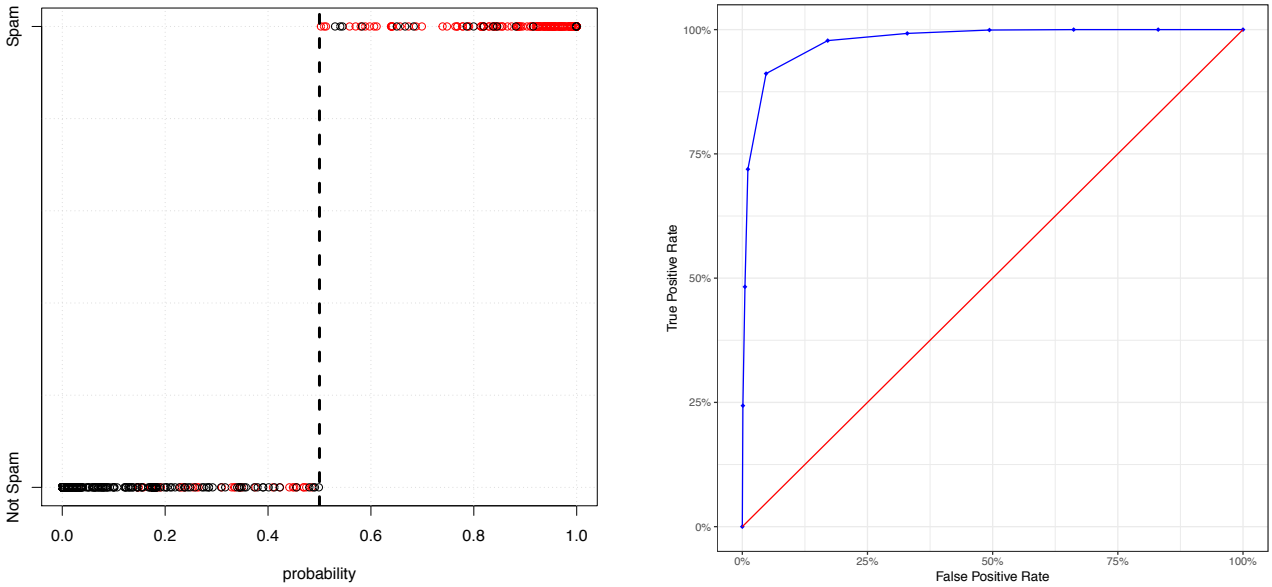
Table 6 shows the validation set results from the different variable selection techniques applied. At this point we can only look at the validation set results as we don't want to contaminate the testing set and bias the output to those results. A number of interesting insights can be drawn from this table in comparing the performance on the validation set of each model. The three models shown in grey all had the exact same performance characteristics and all represented the full model with no variables excluded. This means that the regularization process for GLM backwards and repeated cross validation on GLM yield the exact same model as a full GLM preformed, indicating that all variables are infact relevant to the prediction of spam or non spam emails. The other three models had some degree of regularization that resulted in the removal of variables from the model which resulted in a lower overall prediction performance. GLM backwards and GLM bothways had the same performance characteristics as both stepwise selections selected to include the same set of variables in their models.

Model Name	Accuracy	Kappa	True Positive	True Negative	False Positive	False Negative	F1
GLM	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	0.03908046	0.9403825
GLM Backward	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	0.03908046	0.9403825
Repeated CV GLM	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	0.03908046	0.9403825
GLM Backward	0.9236111	0.8384333	0.9586207	0.8701754	0.1298246	0.04137931	0.9381327
GLM Bothways	0.9236111	0.8384333	0.9586207	0.8701754	0.1298246	0.04137931	0.9381327
CV Lasso GLM	0.9208333	0.8321472	0.9609195	0.8596491	0.1403509	0.03908046	0.9361702

Table 6: Variable selection validation results for logistic regression, ordered by overall accuracy

Two valuable plots were created to visualize the best preforming standard GLM model. Firstly, Figure 8a shows the probability of a response occurring vs the classification response with red representing a spam email and black representing a legitimate email. The classifier in this configuration is setup to define emails with a probability > 0.5 to represent a spam email and those < 0.5 to be legitimate. This threshold is notated with the dashed black line on the figure. Some interesting points on this figure are the miss-classified non spam as spam emails with the black circles amongst the red values in the spam category. Surprisingly the model miss-classified one point which it had assigned a probability of being spam at 1.0 which can be seen at the top right of the figure.

Figure 8b shows the Receiver Operating Characteristics (ROC) curve for the generated model. This relates the false positive (email identified as not spam when it is spam) and the true positive (correctly identified emails) rates together and can be used to quantify model quality to find the ideal ratio between these rates given the context in which the model will be used. This is discussed extensively in Section 3.8.2 where ROC curves are used to find a modefied model given domain context.



(a) Responce vs probility classification plot

(b) GLM ROC plot

Figure 8: Best preforming GLM model visualizations

3.5 Question 2.B: Linear Discriminant Analysis

Next, Linear Discriminant Analysis (LDA) and later Quadratic Discriminant Analysis (QDA) is applied to build a new group of prediction models. For the binary case of classifying emails as either spam or not spam LDA becomes known as Fisher's LDA[5]. LDA models the distribution of each predictor variable X separately for each of the response classes Y and then uses Bayes' Theorem to manipulate and flip these values around to yield estimates for $Pr(Y = k|X = x)$. For the case that these distributions are normal, this model behaves very similarly to Logistic Regression. LDA works by generating a discriminant score for each observation and uses this to classify what class each response variable falls in.

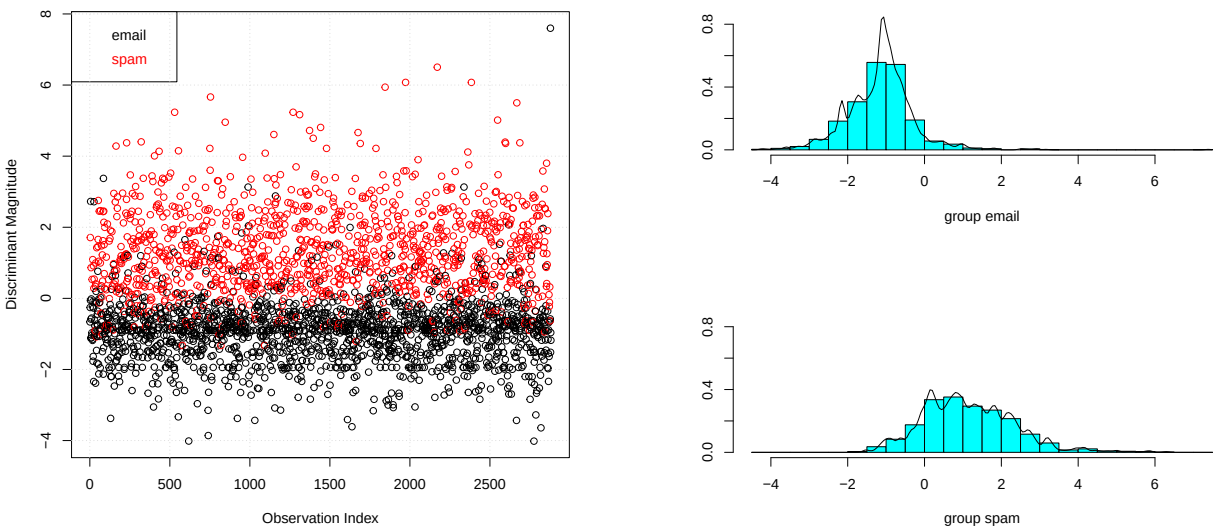
LDA can be simply implemented in `R` using the `lda` function. Predictions are drawn using the `predict` function with `type = "response"` specified. This returns a dataframe with a column `class` that indicates the outcome of the prediction. This means that no classification threshold needs to be implemented as the predict function does this for us with an assumed threshold value of 0.5. The code implementation for this question can be found in Appendix 1.3.

3.5.1 Linear Discriminate Analysis Variable Selection

Now that a generic LDA model has been explored a more refined set of variables was derived using a forward variable selection process with the Wilks' Lambda (Λ) criterion. Wilks' lambda tests how well each level of an independent variable contributes to the overall model. The method scales the significance of variables between 0 and 1 where 0 = total discrimination and 1 = no discrimination. As with other stepped procedures, the ideal model is created by adding the variable to the model, testing it and then removing it. At each step a Λ statistic is generated and at the end significant changes in Λ are measured using an F-test. If this tests yields a value larger than a critical threshold then the variable is kept in the model[6].

The Wilks test is implemented in `R` using the `greedy.wilks` function from the klaR package. The variable selection process selected a total of 39 variables to be included in the simplified LDA model. Once the ideal set of variables has been selected via the Wilks test a standard LDA can be created and then treated as a usual LDA model in generating predictions.

Two figures were created to visualize the refined Wilks' LDA model. Figure 9a shows the discriminate score allocated to each variable based on the discriminant model generated from the Wilks' selected subset. This score value quantifies the notion of separability within the LDA model. The selected coefficients used to generate the model are those that maximize this overall score value. It is a valuable plot as it shows the underlying workings of the discriminate selection process. Black circles above the zero line represent miss classified email and red circles below the zero line indicate miss classified spam. Interestingly the point in the extreme top right is an email that should be considered as spam based off the discriminant magnitude is the same point of interest from Figure 8b. Figure 9b shows the same information as Figure 9a but displayed as two histograms representing the distribution of the discriminates between the two classes.



(a) Discriminant score against variable index

(b) Discriminant distributions

Figure 9: Linear discriminate analysis variable selection visualizations

3.5.2 LDA Model Assumption Validity Testing

Due to the fundamental mechanism that LDA utilizes to classify elements a number of assumptions are required for the model to hold true. Primarily LDA assumes that the data follows a normal distribution, the features are statistically independent and that the covariance matrices for every class are identical. Each of these key properties will be examined in turn as well as the value of the assumptions within the

context of the question.

Assumption: Data Follows a Multivariate Gaussian (Normal) Distribution In the event that data is normally distributed LDA always guarantees a solution that minimizes the expected error. LDA can however be applied to data that does not follow a normal distribution, but then there is no strong guarantee that it will find the minimum expected error.

A number of different tests for multivariate normality testing are available. A common one is to the *Shapiro-Wilk Normality Test*, implemented in `R` using the `shapiro.test` function. This tests the null hypothesis that the samples come from a normal distribution against the alternative hypothesis that the samples do not come from a normal distribution. The provided dataset failed this null hypothesis, indicating that the data is not normal. Another approach was conducted to check for Multivariate Normality using the `MVN` library in `R` that provides a suite of different tests for normality checking. The *Mardia* and *Henze-Zirkler* and tests were both applied and both returned a negative result on testing for Multivariate Normality. A visual `qqplot` was also generated for all the variables to test for multivariate normality. This was not shown here as it had the same outcome as the previous tests showing that the dataset is not not normal. This figure can be found in Appendix 2.3.

However, there is a major problem in using these kinds of tests for normality checking, primarily that the test is testing the null hypothesis *against* the assumption of normality. This becomes a problem as when the sample size is small and large departures from normality are missed and when the sample size is large then even small variance from a normal distribution results in a rejected null hypothesis.

Additionally, it is important to note that there is research to suggest that LDA still achieves good performances despite the violation of the normality assumption (and even the common covariance matrix assumption)[7]. Due to central limit theory the assumption of normality implied in LDA might not actually be an issue at all due to the large sample size in this question. This comes from the fact that when a large number of samples are present the test statistic often approaches a normal distribution. The larger the total data size the more prevalent this effect is. Given that the training set has over 2000 rows and the requirement for the central limit theorem effect is for a sample size of larger than 100 (> 30 for non-skewed data) it is fair to assume that the dataset will be well-approximated by a normal distribution. Due to this reason, the assumption of normality is acceptably justified to hold true despite the tests failing.

Assumption: Features are Statistically Independent Two events are statistically independent if the occurrence of one does not affect the probability of the other occurring. For the given problem set, it is logical to assume that receiving a spam email would be statistically independent of receiving a non-spam email. This assumption was verified using the *Pearson's Chi-Squared Test* implemented in `R` with `chisq.test`. This test yield a `p-value < 2.2e-16` that indicates that the events are statistically independent and so this assumption holds true.

Assumption: Classes have Identical Covariance Matrices Identical covariance matrices is the fundamental property that results in LDA having a linear decision boundary. This can be tested using the *Box's M test* where the null hypothesis for this test is that the observed covariance matrices for the dependent variables are equal across groups. The test was preformed using the `R` function `boxM` and yielded `p-value < 2.2e-16`. From this it can be concluded that the null hypothesis fails and thus there is an issue of heterogeneity of variance for the covariance matrices. However the Box's M test is very sensitive to departures from normality as it assumes that the data is multivariate normally distributed, and as was shown before this assumption does not hold and so the value of this test is questionable[7].

3.5.3 Quadratic Discriminant Analysis

LDA assumes that each class's observations follow multivariate Gaussian distribution and that the covariance of the predictor variables are common for all levels of the response variable. QDA modifies these assumptions by assuming that the observations from each class are drawn from Gaussian distributions but each class has it's own covariance matrix. This means that the predictor variables are not assumed to have common variance across each level in the predictor. The major change here is that the decision boundary that separates predictors no longer represents a linear line and can potentially capture more non-linear variability within the model. The generation and prediction process of QDA is very similar to that done by LDA, utilizing the `qda` and `predict` functions in `R`.

3.5.4 Discriminant Analysis Classification Results

The validation accuracies for LDA and QDA can be seen in Table 7. These results are across the board worse than the results obtained for all logistic regression models. This is most likely due to the fact that the key underlying normality assumptions required to achieve a good LDA and QDA model did not hold true. LDA and QDA are also extremely sensitive to outliers and this could have had a negative effect on the model performance. Ideally LDA should also have roughly equal group sizes to preform well and this is not the case with the data set provided[8]. The LDA model implemented with Wilks' variable selection had the exact same performance characteristics as standard LDA. This is interesting because this model had only 39 predictor variables apposed to the normal LDA that had all 58 predictors. When the Wilks' implementation was run on the test set there was a discrepancy between normal LDA and the Wilks' model. This can be seen later in Section 3.8 where test errors are examined.

Model Name	Accuracy	Kappa	True Positive	True Negative	False Positive	False Negative	F1
LDA	0.8930556	0.7695761	0.9678161	0.7789474	0.22105263	0.03218391	0.9162133
LDA,Wilk	0.8930556	0.7695761	0.9678161	0.7789474	0.22105263	0.03218391	0.9162133
QDA	0.8569444	0.7138889	0.7954023	0.9508772	0.04912281	0.2045977	0.8704403

Table 7: Discriminant analysis classification validation results

One interesting result that can be seen in Table 7 is the true positive and true negative rates for the two methods. The true positive rate in the case of LDA is higher than any of the true positive rates for any logistic regression model previously specified. Similarly, the true negative rate for QDA is higher than any of the other methods seen thus far. This point of intrest will be examined further in section 3.8 when all the methods proposed are compared holistically.

3.6 Question 2.C: Random Forest Classification Model

Next, tree based methods are explored, beginning with random forests. From a high level, random forests build up multiple decision trees with each tree being grown from a random subset of bootstrapped features. This set of trees is then used to preform prediction by aggregating the result from all the trees in the set. In the case of classification the mode is taken as the predicted class and in the case of regression the mean is taken as the predicted value. Random forests are considered as a kind of *ensemble learning* as they consist of multiple learning algorithms (in this case a collection of different trees) to form the overall prediction model with the goal of achieving better predictive performance. Random forests tend to preform well even without hyper-parameter tuning and are simple to apply to both classification and regression problems.

The theory behind restricting the feature set upon which each tree is grown is to reduce the effect of highly correlated bootstrapped samples that is seen in bagging[9]. This effect is even more prevalent when there are a few important predictors within a set of otherwise less relevant predictors. If no restriction is placed on the set of features then each tree is grown with full set of predictors, effectively preforming bagging.

There is no need to do cross-validation when building random forests to get an unbiast estimate of the test error. Rather, as a result of the sampling with replacement that occurs during the bootstrapping process about one-third of variables are left out and not used in the construction of a particular tree, know as the *out of bag* values. These out of bag values can then be used to calculate the the error rate for that particular tree, known as the *out of bag error*.

3.6.1 Random Forest with the H2O Package

The H2O `R` package provides an interface to connect with the H2O framework for running a number of different machine learning algorithms. It was used for running all random forest and boosted trees preformed in this report. No cluster or high performance computation environment was used but rather the algorithms were run on a laptop with a 12 core i7 CPU that ran most tree building experiments in under five minutes. The code implementation for this question can be found in Appendix 1.3.

To begin, a simple random forest model was fitted onto the data set with all parameters left as default. A `seed` was specified along with the other parameters to ensure that the results are reproducible. H2O requires that the input data format is of class `H2OFrame` and the standard `R` dataframes was converted to this using the H2O function `as.h2o()`. After the model was generated predictions were made using the H2O function `h2o.predict` applied on both the validation and test sets. This result was then converted to an `R` dataframe with `as.data.frame` and at this point a normal confusion matrix was generated to extract accuracy expectations for the tree.

Next, `h2o.grid` was used to test a range of different hyper parameters used in growing the random forest. The key hyper parameters experimented with are now broken down in turn. Each hyper parameter is `floor`ed to provide integers to the `h2o.grid`. This is required as you can't have a non-integer value for a discrete quantity like the number of trees to grow. For each hyper parameter 5 steps were included and as there are three tuning hyper parameters this means that a total of $5^3 = 125$ diffrent random forests were grown, from which the best model was selected. The tuning hyper parameters are as follows:

1. `max_depth` restricts the total size that each tree within the random forest can grow to. A range of values between 10 and 50 with 5 discrete steps was used for the hyper parameter selection.
2. `ntrees` defines the total number of trees to grow within the forest and thereby restrict the total size of the forest. A range of values between 50 and 300 with 5 discrete steps was used for this hyper parameter.
3. `mtries` defines the total number of variables randomly sampled as candidates at each split within the creation of each tree within the random forest. Out of all the hyper parameters this one is the most relevant to random forests as it defines the set from which each tree within the forest can be grown. Typically a value of $\approx \sqrt{p}$ is used for p predictors[9]. A range was therefore created for this hyper parameter from $\frac{\sqrt{p}}{2}$ to $3 \times \sqrt{p}$ with 5 discrete steps in the range.

3.6.2 Random Forest Model Classification Results

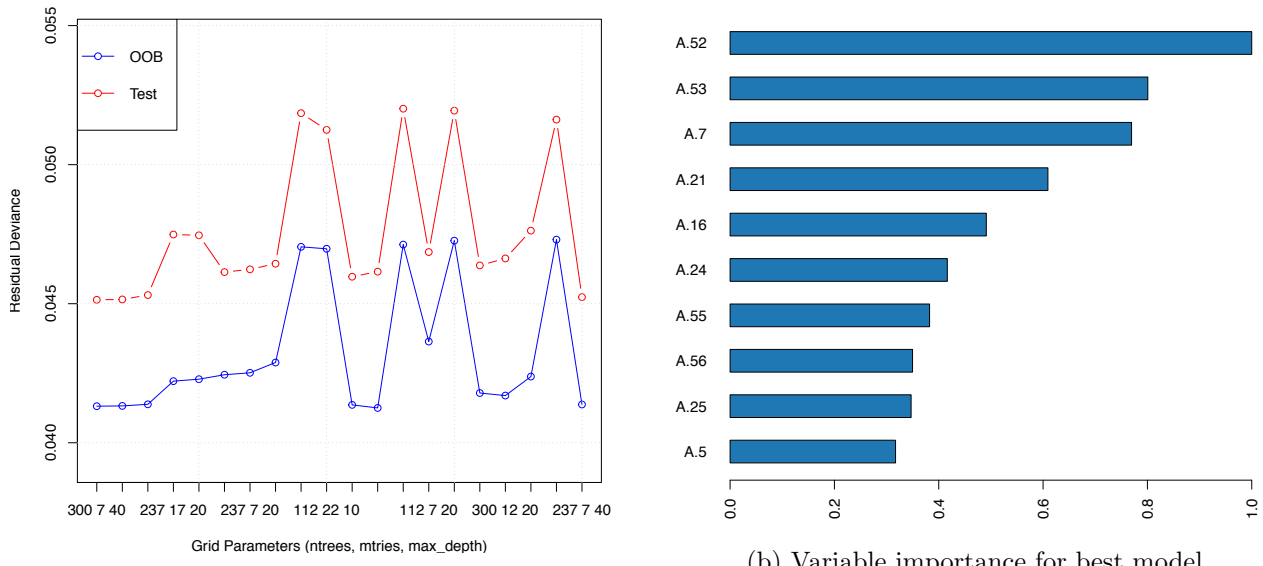
The results from the general random forest with default parameter values and for the best selected hyper parameter optimized random forest can be seen in Table 8. A sizable increase in accuracy is observed with a higher prediction accuracy than any previous model identified by almost 2%. The introduction of the tuning grid resulted in a small improvement of model accuracy over the general default random forest, only accounting for less than half a percent.

Model Name	Accuracy	Kappa	True Positive	True Negative	False Positive	False Negative	F1
Random Forest,Grid2	0.9500000	0.895082	0.9655172	0.9263158	0.07368421	0.03448276	0.9589041
Random Forest,Grid	0.9486111	0.8924939	0.9586207	0.9333333	0.06666667	0.04137931	0.9575201
Random Forest	0.9444444	0.8832827	0.9632184	0.9157895	0.08421053	0.03678161	0.9544419

Table 8: Random forest validation results

The ideal hyper parameters from the `h2o.grid` values specified were for a tree with `max_depth=40`, `ntrees=300` and `mtries=7`. The values for `max_depth` and `mtries` are not at the extreme maximum or minimum of the ranges previously defined but `ntrees` is. This indicates that there are potentially other improvements that were not captured due to the range being too narrow for this `ntrees` hyper parameter. Therefore one final `h2o.grid` was formed with slightly refined hyper parameter ranges with and increase on the upper bound on `ntrees` and the other two hyper parameters having slightly narrow ranges around their identified ideal values. Again another 125 random forests were grown and the results can be seen in Table 8 with the model named `Random Forest,Grid2`. This small refinement yielded a gain of 0.138% over the previous random forest and the new ideal hyper parameters for this refined grid are `max_depth=35`, `ntrees=400` and `mtries=7`. This equates to a net percentage increase of 0.28%, a somewhat negligible amount.

Two interesting figures can be generated from the refined random forest model to compare the out of bag error (OOB) with the test error and to visualise the variable importance for the predictors within the random forest model. These two figures can be seen in Figure 10. Figure 10a shows the top 20 most accurate models generated within the set of 125 different random forests that were built. Along the x axis the different hyper parameters combinations can be observed and the associated residual deviance is visible on the y axis. From this plot the ideal model that generated the lowest possible error can be seen as the point on the extreme left. Figure 13b shows the top 10 most important variables as identified by the randomforest process. From this plot it is clear that `A.52`, `A.53` and `A.7` are very important. These fields identify the high frequency occurring characters within the emails.



(a) Out Of bag error and test error

(b) Variable importance for best model

Figure 10: Random forest plots

3.7 Question 2.D Gradient Boosted Trees Classification Model

The last classification method explored is gradient boosted trees (GBT). Unlike bagging and random forest wherein each tree is built independently on a boostrap data set, boosting grows its trees sequentially where each tree is grown using the information from previous tree grown. Boosting does not use bootstrap sampling but rather each tree is fit on a modified version of the original data set[9]. The boosting approach learns slowly wherein each decision tree is fit to the residual of the model from the previous fit tree rather than the outcome class Y. The generated decision tree is then added to the fitting function to update the residuals used. The improvement in the tree prediction slowly grows as more and more trees are added to update the quality of the fit.

A shrinkage parameter λ is included that slows down the growth speed allowing for different shaped

trees to form. This is called the *learning rate* and is a hyper parameter that can be tuned for the model.

3.7.1 Boosted Trees with the H2O package

Just like the random forest proposed in section 3.6.1 boosted trees can be implemented using the H2O framework. As before a boosted tree is first created with the default parameters. This is done with the `h2o.gbm` function to specify that a gradient boosting machine is to be built. The process of predicting and evaluating the model is the same as the random forest. The code implementation for this question can be found in Appendix 1.3.

Next, like with the random forest, a `h2o.grid` was used to define a number of hyper parameters which are broken down below. This set of hyper parameters results in building a total of $5^3 = 125$ different boosted trees which are compared to identify the ideal values.

- `learn_rate` specifies the shrinkage parameter λ which defines how quickly the trees grow during the sequential boosted tree growing process. This hyper parameter has a range from 0.005 to 0.5 with 5 steps in the range.
- `ntrees` is similar to the random forest but now defines the total number of trees to build during the sequentual process rather than a parallel process. A range between 50 and 250 with 5 steps was specified.
- `max_depth` is exactly the same as in the random forest defining the maximum tree depth for each tree grown. This is the size that each tree can be grown too while itterativly building up the boosted tree. A range of between 10 and 50 was used with 5 steps for this hyper parameter.

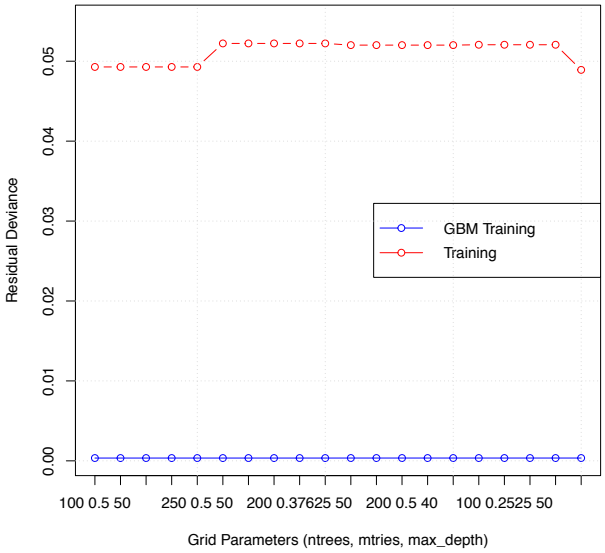
3.7.2 Boosted Trees Classification Results

The classification results for the boosted tree experimentation can be seen in table 9. Like with random forest, one row corresponds to the set of tests with the standard default parameters and another when the `h2o.grid` is included. These results show the highest accuracy out of all models created thus far.

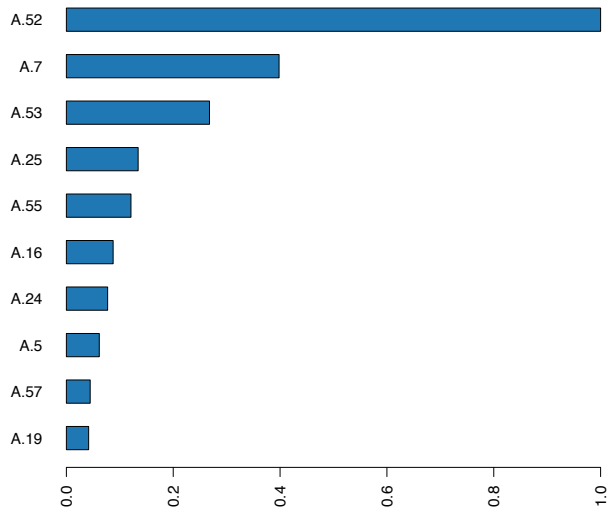
Model Name	Accuracy	Kappa	True Positive	True Negative	False Positive	False Negative	F1
Boosted Tree,Grid	0.9500000	0.895082	0.9655172	0.9263158	0.07368421	0.03448276	0.9589041
Boosted Tree	0.9444444	0.8835658	0.9586207	0.9228070	0.07719298	0.04137931	0.9542334

Table 9: Boosted tree classification validation set results

Two plots were created to visualize the training error vs the validation error and the variable importance for the best created boosted tree model which can be seen in Figures 11. As boosting does not use bootstrapping in each tree creation there is no notion of "out of bag error" for a boosted tree. Figure 11a rather shows the GBM training error on the modified version of the original data set that is used throughout the training process. The magnitude of these errors, as can be seen in the figure, are much lower than the actual testing error and so it appears that some degree of overfitting may have occurred. This could explain the high accuracy seen in Table 9 but this hypothesis can only be fully explored once the testing errors are calculated. Figure 11b shows the variable importance for the boosted tree. This figure indicates that the predictor `A.52` is the most significant for boosting. This was the same term that the random forests model identified as its most important, as shown in Figure 13b. However the second and third most important terms are swapped around for the boosted tree when compared to the random forest. Additionally, the boosted tree shows a much higher relative importance for its top variable, `A.52`, when compared to all other variable. This relative importance is more gradual in the random forest.



(a) boosted tree model error vs training error



(b) Variable importance for best model

Figure 11: Gradient boosted trees plots

3.8 Question 2.E: Different Classification Models Comparison

Now that a number of diffrent classification methods have been examined and refined they can be tested on the until now unused testing data provided. 1000 rows were allocated for this purpose and the results of all previously discussed classification models applied on this testing set can be see in Table 10. From this table it is clear that the tree based methods out preform all other methods, with random forest using the refined grid proving to have the highest accuracy overall.

Model Name	Accuracy	Kappa	True Positive	True Negative	False Positive	False Negative	F1
Random Forest,Grid2	0.942	0.8720364	0.9644513	0.9008499	0.0991501	0.0355487	0.9555896
Boosted Tree,Grid	0.942	0.8713689	0.9706337	0.8895184	0.1104816	0.0293663	0.9558600
Random Forest,Grid	0.940	0.8677954	0.9613601	0.9008499	0.0991501	0.0386399	0.9539877
Random Forest	0.938	0.8628555	0.9644513	0.8895184	0.1104816	0.0355487	0.9526718
Boosted Tree	0.938	0.8630337	0.9629057	0.8923513	0.1076487	0.0370943	0.9525994
GLM	0.934	0.8549508	0.9536321	0.8980170	0.1019830	0.0463679	0.9492308
GLM Forwards	0.934	0.8549508	0.9536321	0.8980170	0.1019830	0.0463679	0.9492308
Repeated CV GLM	0.934	0.8549508	0.9536321	0.8980170	0.1019830	0.0463679	0.9492308
GLM Backward	0.926	0.8375790	0.9459042	0.8895184	0.1104816	0.0540958	0.9429892
GLM Bothways	0.926	0.8375790	0.9459042	0.8895184	0.1104816	0.0540958	0.9429892
CV Lasso GLM	0.926	0.8365241	0.9536321	0.8753541	0.1246459	0.0463679	0.9434251
LDA	0.890	0.7521663	0.9489954	0.7818697	0.2181303	0.0510046	0.9177877
LDA, Wilks	0.886	0.7431542	0.9459042	0.7762040	0.22379603	0.05409583	0.9147982
QDA	0.821	0.6415785	0.7496136	0.9518414	0.0481586	0.2503864	0.8442124

Table 10: All models error statistics on testing set

From Table 10 the best model overall accuracy when applied to the testing set is the random forest with the refined grid. An overall accuracy of 0.942 was achieved, meaning that 94.2% of the time the model will correctly classify emails. If random selection of variables is removed then the κ value shows us that 87.2% of the time the model would correctly classify an email.

Looking at the different miss-classification rates across all models shows an interesting pattern: some models are better at minimizing the false positive rate while others are better at minimizing the false negative rate. For example, boosted trees with a grid show a lower false negative rate than the best overall model (random forest with refined grid) but have a higher false positive rate. This indicates that there is more to picking a classification algorithm than just the overall accuracy and the context of where the algorithm will be used is important to consider based on what kind of accuracy is the most important

to the use case. This tradeoff is now considered in the context of where this classification model would be used in the real world.

3.8.1 Impact of Miss-Classification On User and Use Case Model Optimization

In the context of this problem miss-classification can either be a false positive (where the model identifies a spam email as not spam) or false negative (where an email is identified as spam when it is is not spam). The former (false positive) results in the user having a spam email appear in their inbox while the latter (false negative) results in the user having a legitimate email put into the spam folder. Having a miss-classified spam email appearing in your inbox is really nothing more than an annoyance, you can simply delete this email or move it to the spam directly (assuming you have an antivirus and that you don't get phished from the email). However having a legitimate email end up in your spam folder could result in the you missing an important communication which could have a real world impact and effect. Therefore, the most important metric to consider is to minimize the false negative rate of the classifier as this kind of miss-classification has the largest negative impact on the user.

In order to manipulate the kind of errors that the classification model will minimize it is useful to look at at Receiver Operating Characteristic (ROC) curve that plots the false positive rate against the the true positive rate. A ROC curve for the three best classifiers identified in Table 10 can be seen in Figure 12. From the rational above regarding miss-classification the goal is to reduce the false negative rate, which is defined by $FNR = \frac{FN}{FN+TP} = 1 - TPR$. Therefore, an ideal value from the ROC curve will be one that produces a *high* true positive rate, thereby resulting in a *low* false negative rate.

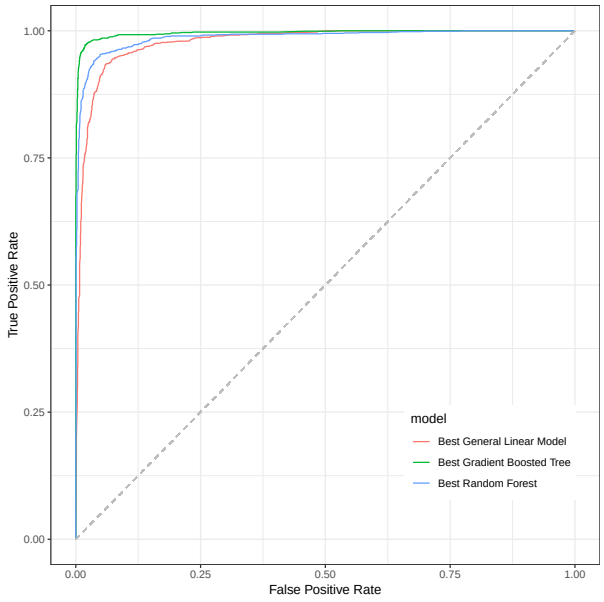


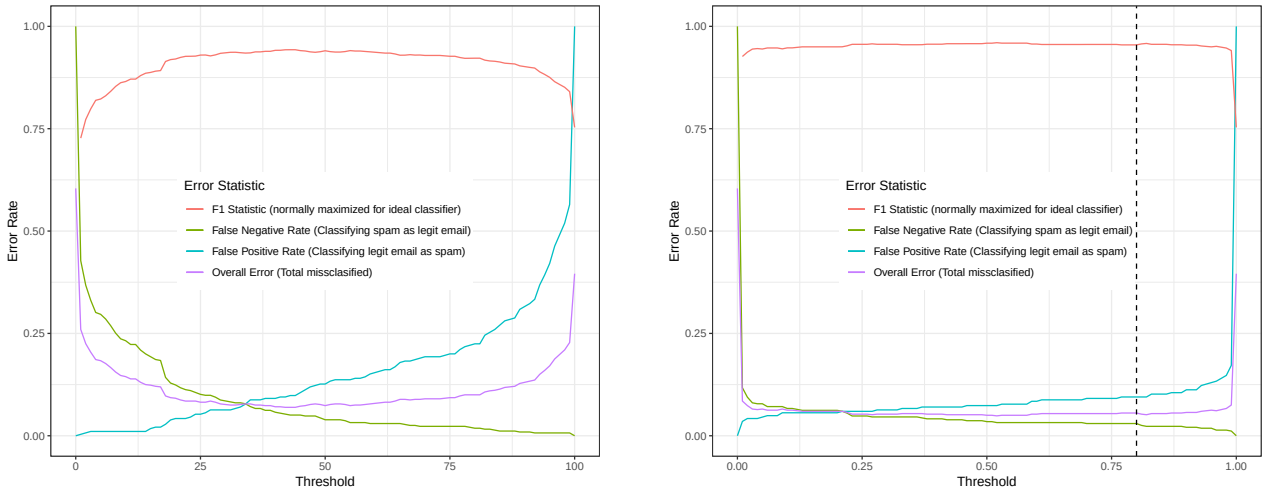
Figure 12: ROC curve for the three best preforming models

3.8.2 Influencing Model False Negative Rate

To influence a model's true positive (and thus false negative) rate the classification threshold that is used to make a class decision is manipulated. For example, logistic regression chooses the class that has the larger probability of occurring. This can be seen in Equation 7 that amounts to assigning the class of the email to spam or not spam if the probability value is larger than ζ. Typically $\zeta = 0.5$ such that `glm.pred[probs > 0.5]<-"spam"`. This is to say that only if the model assigns the probability of an email being spam to a value larger than 0.5 will the email be classified as spam. One therefore influences the ROC curve and directly the false negative rate by changing ζ.

$$
\begin{aligned}
Pr(Spam = Yes|X = x) &> \zeta \\
Pr(Spam = No|X = x) &\leq \zeta
\end{aligned}
\tag{7}
$$

Different contexts require diffrent values of ζ and its value will have a direct impact on the overall performance of the model. Standard MSE calculations, including those done by `h2o.mse`, assume a value of $\zeta = 0.5$. However this might not be the optimal value if the aim is to get the most performant model. Other techniques, such as those used by `h2o.performance`, pick the value of ζ that maximizes the model's F1 statistic. Figures 13 show the relationship between ζ and model error statistics for the basic logistic regression and most optimized boosted tree configuration. These figures were generated on the validation set.



(a) Logistic regression

(b) Best boosted tree with chosen $\zeta = 0.8$ value

Figure 13: Threshold vs statistics plots showing key classification error rates. The F1 statistic is included to quantify the overall model performance

Comparing the shapes of these figures provides insight into how the ζ value can be changed to optimize for a particular error statistic. The goal is to find a value of ζ that greatly reduces the false negative rate (green line) while not increasing the overall Error rate (purple line) or false positive rates too negatively. Interestingly Figure 13a shows a much more curved overall shape in the error rates when compared to Figure 13b which has a much flatter overall performance characteristic for diffrent values of ζ. This means that one can have higher control over the ζ value when using a gradient boosted classifier without effecting the overall model performance too much. This is also very evident when looking at the F1 statistic which depicts a very flat overall shape for the boosted tree, when compared to the logistic regression model.

When a value of $\zeta = 0.8$ is chosen and applied to the best boosted tree model the false negative rate drops to 0.02988 from 0.03448, which equates to a percentage decrease of 13.4% in miss-classified not spam emails as spam. This however increases the false positive rate to 0.09473 from 0.0736 which is amounts to a percentage change of 28.7% increase in false positive miss-classifications. The overall accuracy of the model is not too badly effected, only decreasing to 0.937 from a 0.942, a 0.53% overall percentage change. Ultimately this is not a massive decrease in the overall miss-classification rates but still could yield a better user experience of the classifier. Further refinement of this trade off is required before an ideal implementation is found.

3.9 Question 2 Conclusion

A number of different classification algorithms were explored and implemented to classify email as either spam or not spam. The best possible model was found to be random forest with a refined grid with a

testing accuracy of 94.2%. The impact on the user based on miss-classification rates for different models was then explored wherein optimizations were made to reduce the false negative rate for classifying legitimate emails as spam.

4 Conclusion

This paper examined an extensive number of regression and classification techniques within the provided problem statement context. It aimed to shocase a overview of these techniques with an emphasis on implementation and model refinement in the context of statistical learning.

Question one involved the creation of a regression model to predict the value of houses in Boston suburbs. A number of different models were proposed with first logical interaction terms being included. Next, an iterative interaction term selection process was preformed yielding the best possible training and test MSE. The best model performance wise was a cross validated elastic net with a total 76 terms and interactions with a final test MSE of 11.3137. This iterative method however generates models that are hard to interpret, but if the goal is accurate prediction then it proves to be effective technique in generating powerful models. Tree based methods were also examined but these were found to be inferior to other classical regression techniques as trees can not include interaction terms.

Question two looked at the creation and validation of a number of different classification techniques with the goal of achieving the highest possible overall prediction accuracy. A random forest with a modified grid proved to be the most effective at this task when a held out testing set was used providing a classification accuracy of 94.2%. The implications of miss-classification were then examined within the real world context of where the model would and a modification to the classification threshold was examined.

References

[1] D. Harrison and D. Rubinfeld, "Boston Dataset," pp. 81–102, 1978. [Online]. Available: https://www.cs.toronto.edu/delve/data/boston/bostonDetail.html

[2] R. Williams, "Multicollinearity Multicollinearity Stata Example (See appendices for full example)," University of Notre Dame,, Tech. Rep., 2015. [Online]. Available: https://www3.nd.edu/rwilliam/

[3] ——, "Heteroskedasticity," University of Notre Dame, Tech. Rep., 2015. [Online]. Available: https://www3.nd.edu/rwilliam/

[4] C.-L. Tsai, Z. Cai, and X. Wu, "THE EXAMINATION OF RESIDUAL PLOTS," Tech. Rep., 1998. [Online]. Available: http://www3.stat.sinica.edu.tw/statistica/oldpdf/A8n29.pdf

[5] M. Welling, "Fisher linear discriminant analysis," University of Toronto, Tech. Rep., 2005. [Online]. Available: https://www.ics.uci.edu/welling/teaching/273ASpring09/Fisher-LDA.pdf

[6] C. E. Brown, "Multivariate Analysis of Variance," NCSS, Tech. Rep. January 1987, 2011. [Online]. Available: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Multivariate{_}Analysis{_}of{_}Variance-MANOVA.pdf

[7] M. Friendly and M. Sigal, "Visualizing Tests for Equality of Covariance Matrices," Tech. Rep., 2018. [Online]. Available: https://arxiv.org/pdf/1805.05756.pdf

[8] M. Pohar, M. Blas, and S. Turk, "Comparison of Logistic Regression and Linear Discriminant Analysis: A Simulation Study," Tech. Rep. 1, 2004. [Online]. Available: https://www.stat-d.si/mz/mz1.1/pohar.pdf

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 2013, vol. 103. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-7138-7

Appendix A: Question 1 Code

This appendix contains all the code required for question one. At each point the code has been commented where required and additional explanations are included.

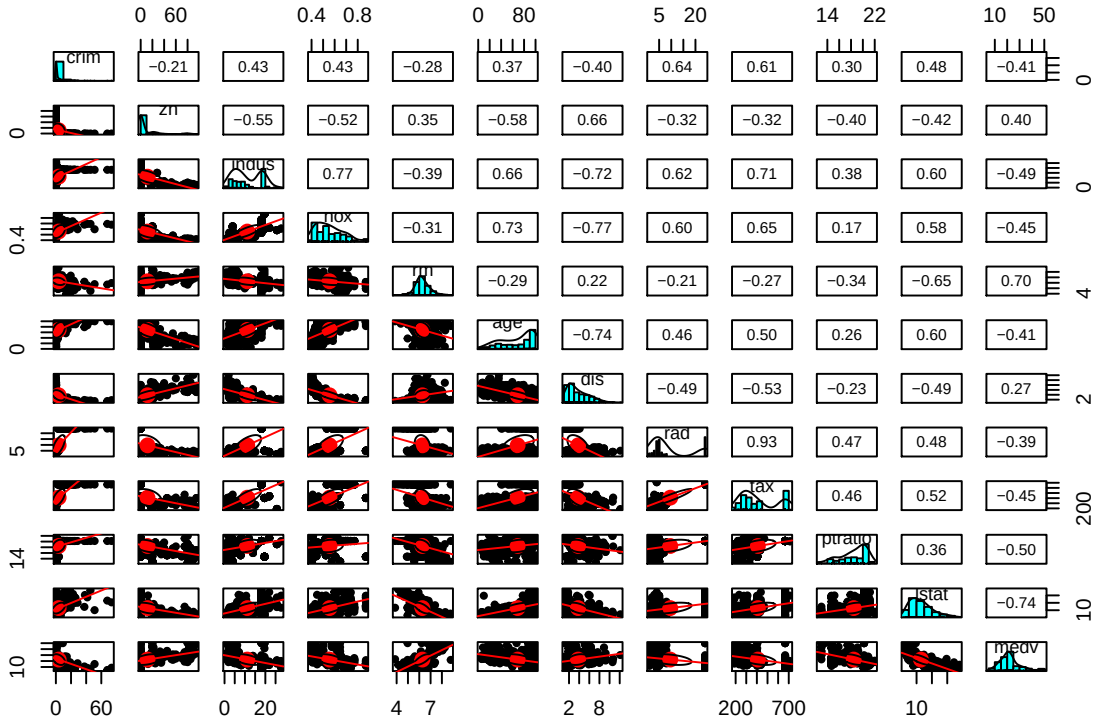
1 Enviroment Setup and Import data files

This notebook is set up to make the results atained as reproducable as posible. To this end, the script from `boston_data_splitter.R` has been included within the notebook such that results can be easily reproduced. This enables all the required information to be pulled directly without having to run a seperate script first. My unique student seed of 26 has been used and is the source of randomess throught the code.

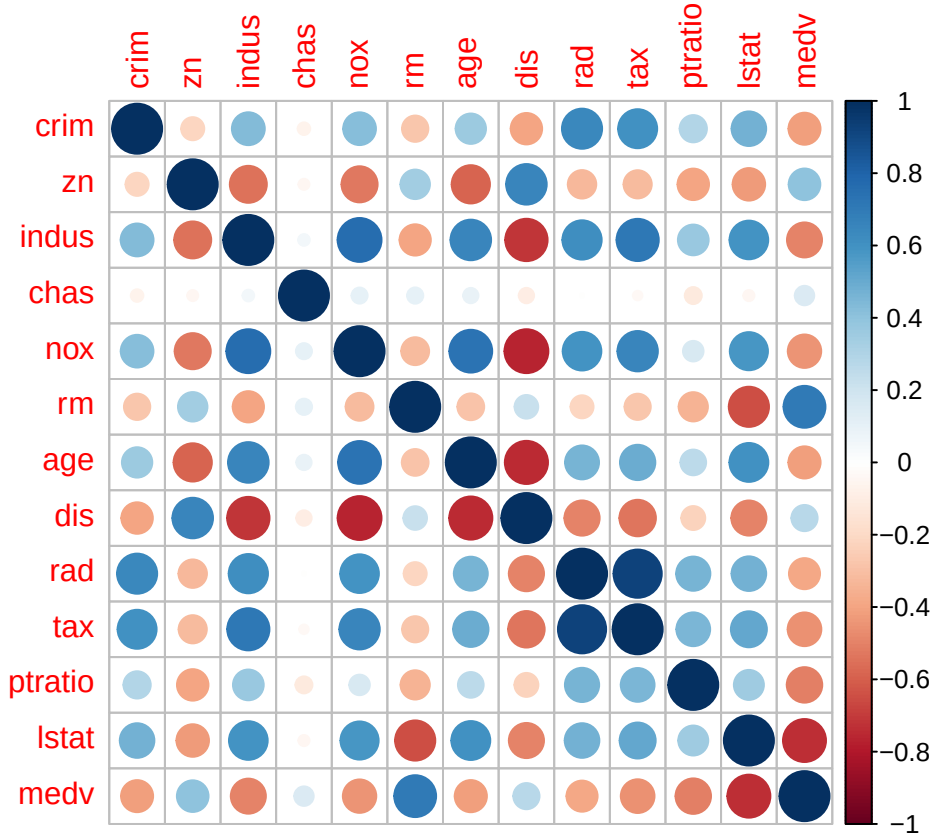
```
#setup work space, install packages and import libs
rm(list=ls())
suppressMessages(library(corrplot))
suppressMessages(library(psych))
suppressMessages(library(MASS))
suppressMessages(library(olsrr))
suppressMessages(library(car))
suppressMessages(library(tree))
suppressMessages(library(leaps))
suppressMessages(library(mlbench))
suppressMessages(library(glmnet))
suppressMessages(library(caret))
suppressMessages(library(lmtest))
suppressMessages(library(plotmo))

#import and sample data, apply seed and sample to get the set of 400 unique data points
fulldata <- read.csv("boston.csv")
set.seed(26)
Boston <- fulldata[sample(1:nrow(fulldata), 400, replace=FALSE), ]
write.csv(Boston, 'my_boston.csv')

#create a dataframe to store all the mean squred error results as this is the
#primarty way of comparing results for this project.
modelPreformance <-data.frame()
```

Basic plotting to visualise information with the data and extraction of key metrics like column names and types

```
pairs.panels(Boston[c(-4)],cex=1,lm=TRUE)
```




```
corrplot(cor(Boston), method = "circle")
```



```
sapply(Boston,class)
```

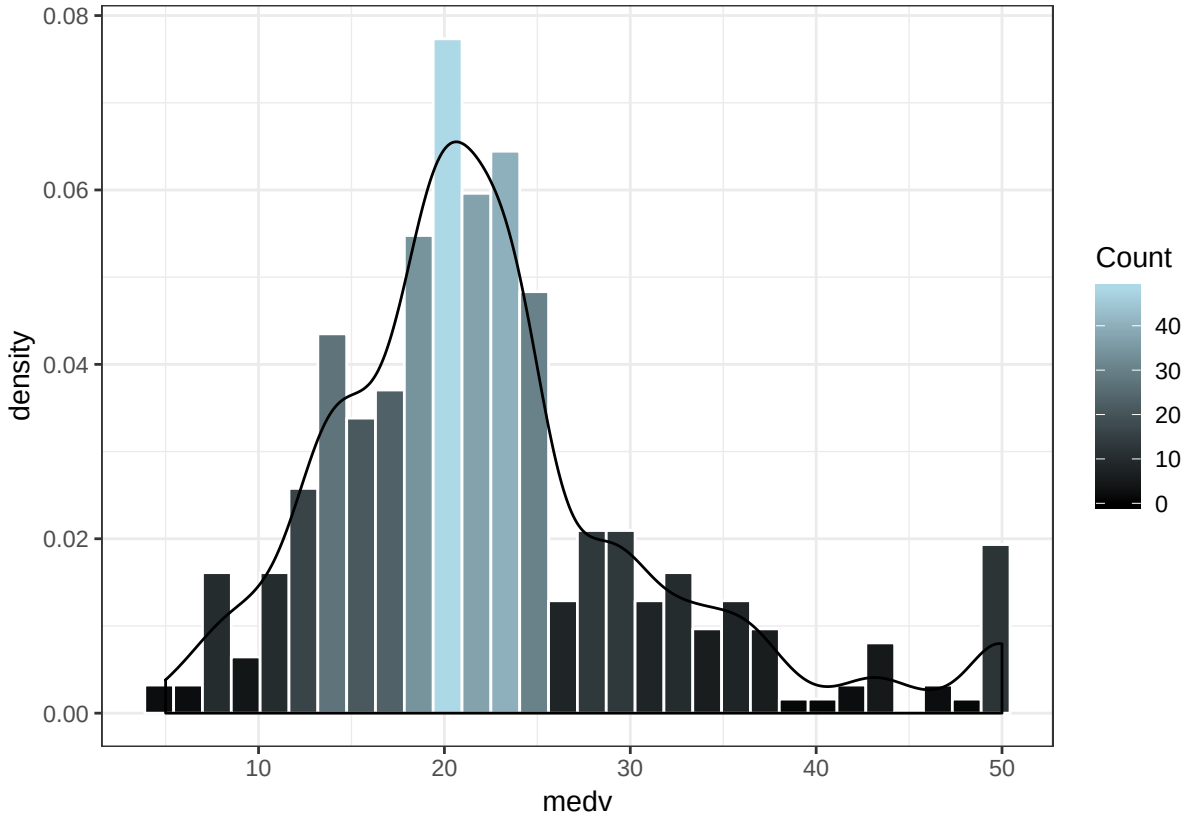
```
##      crim        zn     indus      chas       nox        rm       age
## "numeric" "numeric" "numeric" "integer" "numeric" "numeric" "numeric"
##       dis       rad       tax   ptratio     lstat      medv
## "numeric" "integer" "integer" "numeric" "numeric" "numeric"
```

```
str(Boston)
```

```
## 'data.frame':    400 obs. of  13 variables:
##  $ crim   : num  0.211 2.155 22.051 9.596 2.447 ...
##  $ zn     : num  12.5 0 0 0 0 0 0 0 0 35 ...
##  $ indus  : num  7.87 19.58 18.1 18.1 19.58 ...
##  $ chas   : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ nox    : num  0.524 0.871 0.74 0.693 0.871 ...
##  $ rm     : num  5.63 5.63 5.82 6.4 5.27 ...
##  $ age    : num  100 100 92.4 100 94 76.5 98.9 91.4 95.2 23.3 ...
##  $ dis    : num  6.08 1.52 1.87 1.64 1.74 ...
##  $ rad    : int  5 5 24 24 5 8 24 24 5 1 ...
##  $ tax    : int  311 403 666 666 403 307 666 666 403 304 ...
##  $ ptratio: num  15.2 14.7 20.2 20.2 14.7 17.4 20.2 20.2 14.7 16.9 ...
##  $ lstat  : num  29.9 16.6 22.1 20.3 16.1 ...
##  $ medv   : num  16.5 15.6 10.5 12.1 13.1 25.1 8.5 21.9 22.3 19.4 ...
```

2 Data exploration

```
medvHist <- ggplot(Boston, aes(medv))
medvHist + geom_histogram(aes(y = ..density.., fill = ..count..),
                          colour = 'white', bins = 30) +
  geom_density() +
  scale_fill_gradient("Count", low = "black",
    high = "lightblue") +
  theme(panel.background = element_rect(fill = "gray98"),
        axis.line   = element_line(colour="black"),
        axis.line.x = element_line(colour="gray"),
        axis.line.y = element_line(colour="gray"),
        legend.position=c(0.1,0.5)) +
  theme_bw()
```



3 Useful functions

Before answering the questions there are a number of useful functions to define, such as calculating the MSE for a given model and test set.

```
#calculating MSE for test set
mse <- function(model) {
  model.pred <- predict(model, Boston.test)
  mse_value <- mean((model.pred - Boston.test$medv)^2)
  return(mse_value)
}

#Calculating MSE for a generic input set. Used for getting the training MSE
mse.generic <- function(model, actual) {
  model.pred <- predict(model, actual)
  mse_value <- mean((model.pred - actual$medv)^2)
  return(mse_value)
}
```

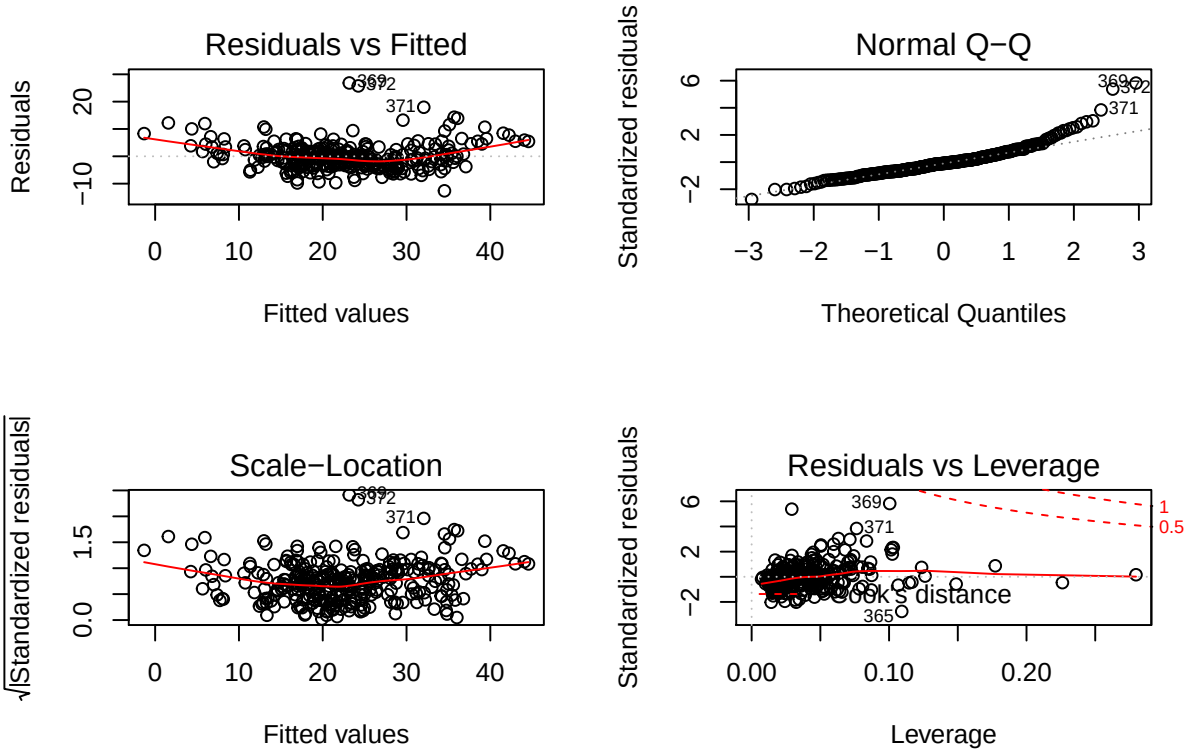
3.1 Question 1.a

Partition the data for training and test sets. Use a seed to make the results reproducable. Set 320 samples for training and 80 for testing. this is 80% for training, 20% for testing

```r
set.seed(26) # need to set the seed again to ensure reproducability
train <- sample(seq_len(nrow(Boston)), size = 320)
Boston.train <- Boston[train, ]
Boston.test <- Boston[-train, ]
```

Then create a multiple linear regression model for the training set to regress `medv` onto all the other explanatory variables. Note that I am using the train function from the caret package. This acts in effect the exact same way as doing somthing like: `lm.fit <- lm(medv ~ ., data = Boston, subset = train)` but it provides a reproduceable syntactical procedure while enabling simple cross validation of the training parameters resulting in more consistant models.

```r
#fit the linear model to the set and plot the output
lm.fit <- lm(medv ~ ., data = Boston, subset = train)
par(mfrow = c(2, 2))
plot(lm.fit)
```




```r
#summary(lm.fit)

model.pred <- predict(lm.fit, Boston.test)
mean((model.pred - Boston.test$medv)^2)
```

```
## [1] 19.59331
```

```r
#use the linear model to predict the output and compare this to the test set
sprintf("Train MSE for basic linear model: %s", mse.generic(lm.fit, Boston.train))
```

```
## [1] "Train MSE for basic linear model: 22.6451926675957"
```

```r
sprintf("Test MSE for basic linear model: %s", mse.generic(lm.fit, Boston.test))
```

```
## [1] "Test MSE for basic linear model: 19.5933110071258"
```

4 Model interms of fit and significance

From the summary we can identify what coefficents have an impact on the model based on each predictors p value. The main variables of significance are: `nox`,`rm`,`dis`,`ptratio` and `lstat`. By looking at the R-squared and adjusted R-squared values we can find how effective the model is. From these we can see that the model expresses 69.39% of the variation in the training set.

4.1 Further model investigation for heteroskedasticity & multicollinearity

```
bptest(lm.fit)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  lm.fit
## BP = 37.902, df = 12, p-value = 0.0001593
```

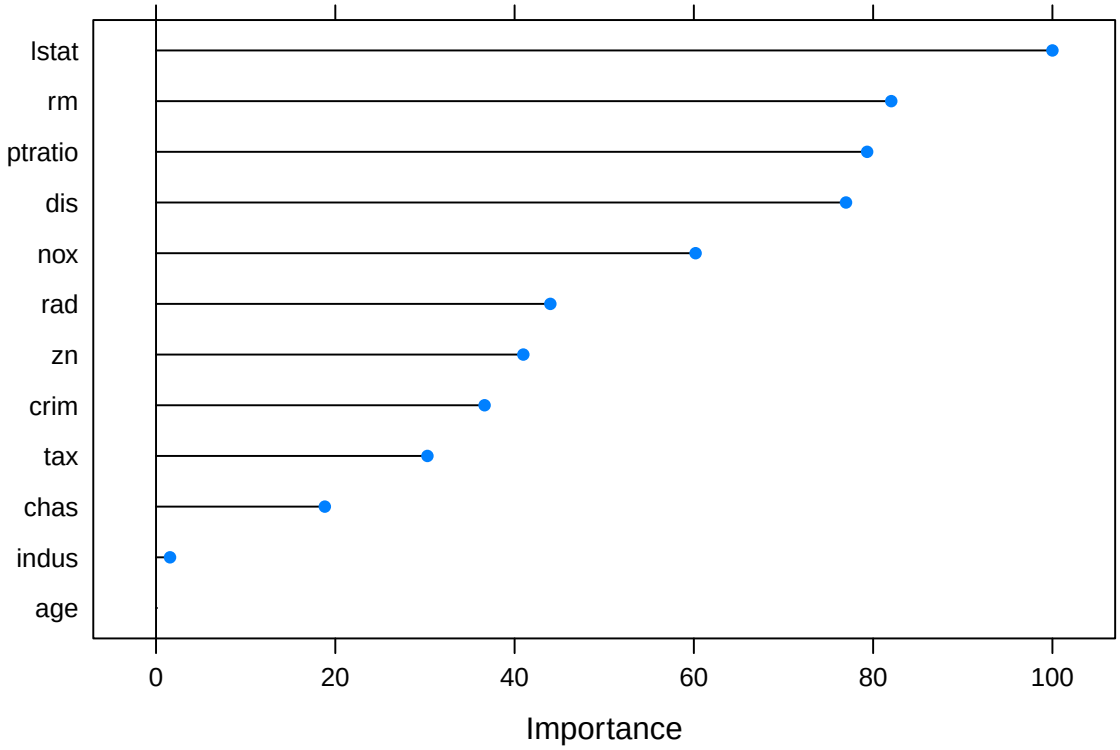
```
vif(lm.fit)
```

```
##      crim        zn     indus      chas       nox        rm       age
##  1.811783  2.265977  4.287403  1.089350  4.248259  1.982035  3.027485
##       dis       rad       tax   ptratio     lstat
##  3.712562 10.161004 12.576020  1.805900  2.792060
```

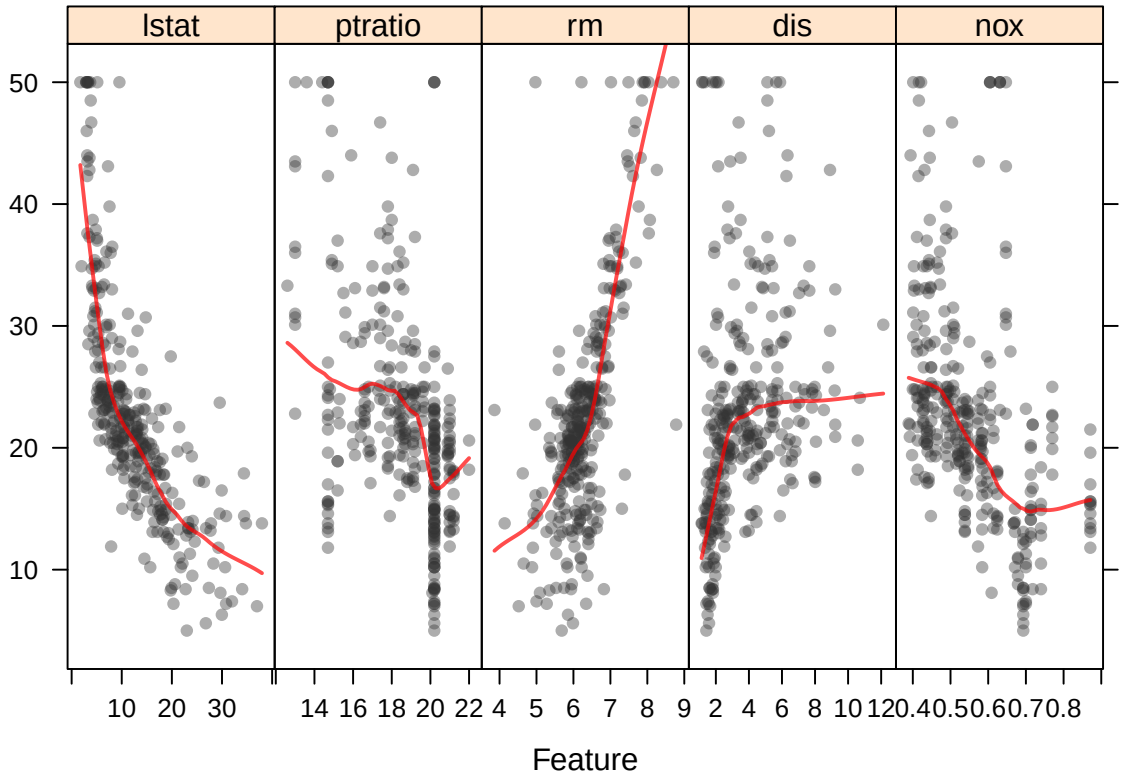
From the studentized Breusch-Pagan test the heteroskedasticity of the model can be investigated. The low p-value of 2.205e-05 suggests that our data is homoskedastic (not heteroskedastic). This is important as it is a requirement for creating linear models. The vif function tells us about the multicollinearity of model model. Values above 5 are considered problimatic and so removing the tax term is advisable.

To quantify the variable significance we can plot the varimportance of the model. we will generate a new Linear model to using repeated cross validation and then plot the variable importance.

```
#Controller to preform repeated cross validation
controller <- trainControl(
  method = "repeatedcv", # repeated cross validation
  number = 10, # k = 10 folds
  repeats = 5, # CV is done 5 times with 5 diffrent sets of k splits
  verboseIter = F)

#create the linear model with no interaction terms
linear <- train(medv~.,
            data = Boston.train,
            method = "lm",
            trControl=controller)
#plot variable importance
par(mfrow = c(2, 2))
plot(varImp(linear, scale=T))
```

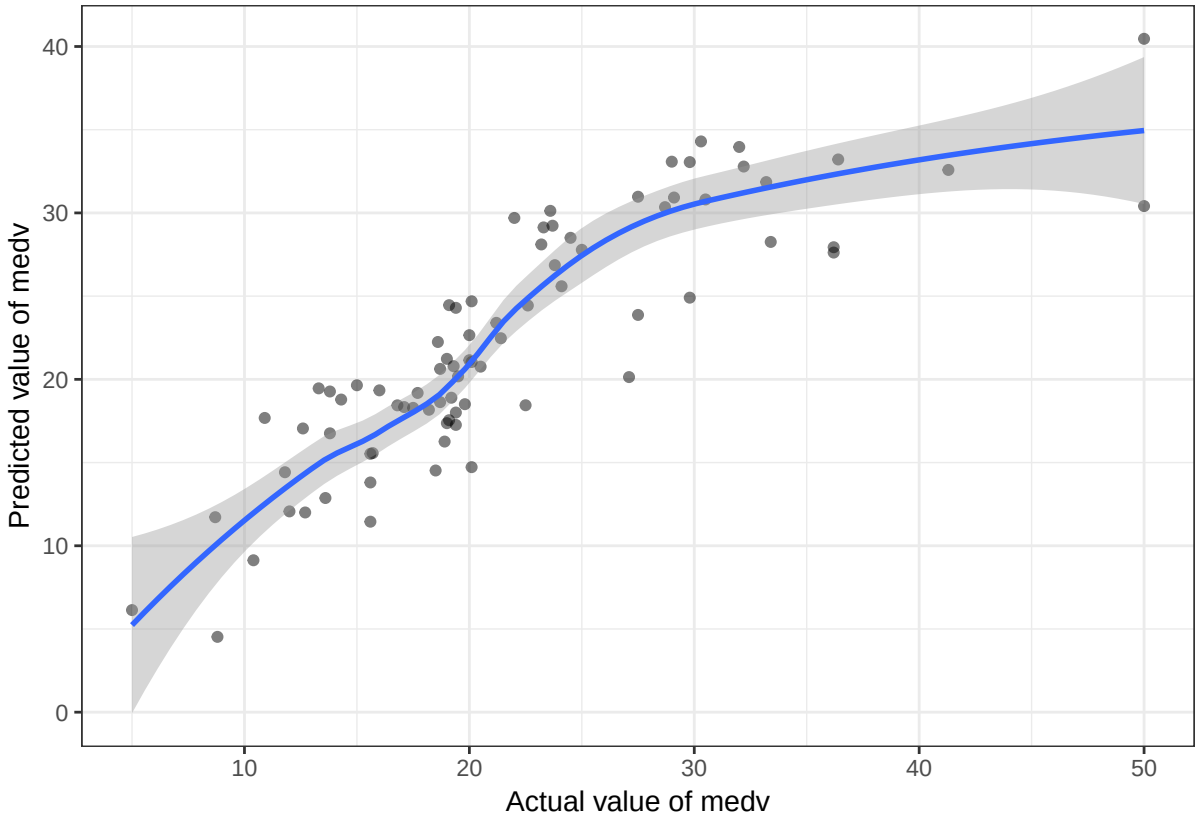



```
# plot the repationship between the medv and the other key variables (top 5)
theme1 <- trellis.par.get()
theme1$plot.symbol$col = rgb(.2, .2, .2, .4)
theme1$plot.symbol$pch = 16
theme1$plot.line$col = rgb(1, 0, 0, .7)
theme1$plot.line$lwd <- 2
trellis.par.set(theme1)
featurePlot(x = Boston.train[, c("lstat", "ptratio", "rm","dis","nox")],
            y = Boston.train$medv,
            plot = "scatter",
            type = c("p", "smooth"),
            span = .5,
            layout = c(5, 1))
```




```
# plot actual vs predicted for the initial basic model
predicted <- predict(linear,Boston.test)
  ggplot(data= Boston.test, aes(medv,predicted)) +
  geom_point(alpha=0.5) +
  stat_smooth() +
  xlab('Actual value of medv') +
  ylab('Predicted value of medv')+
  theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
#use the linear model to predict the output and compare this to the test set
sprintf("Train MSE for validated linear model: %s", mse.generic(lm.fit, Boston.train))
```

```
## [1] "Train MSE for validated linear model: 22.6451926675957"
```

```
sprintf("Test MSE for Cross validated linear model: %s", mse.generic(lm.fit, Boston.train))
```

```
## [1] "Test MSE for Cross validated linear model: 22.6451926675957"
```

5 Question 1.b

The basic linear model derived in question 1.a will now be improved. The quality of this improvement will be quantified by the MSE on the test data. Two main process will be implemented to improve the model. The first involves analizing the previous linear model to identify the important variables and potentual interaction terms based off this exploration will then be included in the model. Variable selection methods such as Lasso will be preformed on the model. This model will be tested and a breif investigation into residual diagnosis will then be preformed.

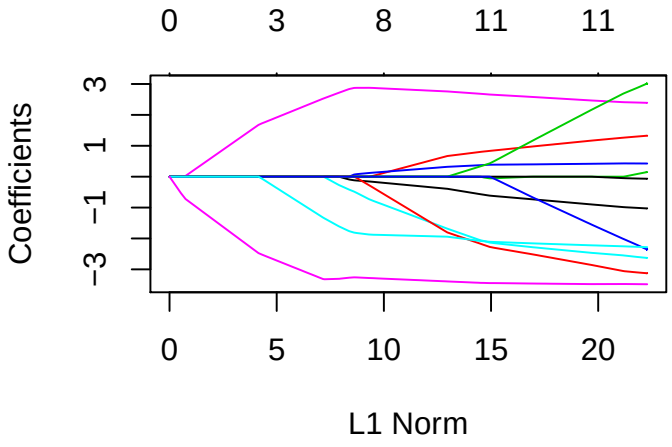
Can define a generic function to plot the variable significance. This is used later on when plotting

```
significancePlot <- function(lassocoef) {
#next we need to conver this to a dataframe so we can do proccessing on it
lassocoef.df <- as.data.frame(as.matrix(lassocoef))
#rename the variables so they are accessable
df.processed <- data.frame(Name = rownames(lassocoef.df), Significance = lassocoef.df$`1`)
#remove the intercept term
df.processed <- df.processed[-c(df.processed$Name == '(Intercept)'),]
```

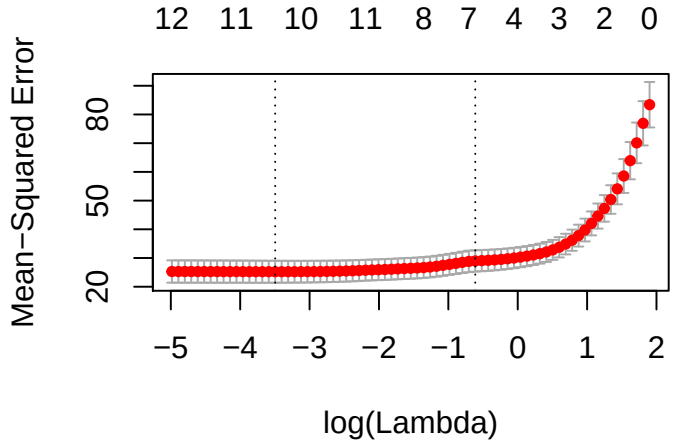
```
#Print the terms
#print(df.processed)

print(colSums(df.processed!=0))
#order by value of significance
df.processed$Name <- factor(df.processed$Name,
                            levels = df.processed$Name[order(df.processed$Significance)])
#plot the variable significance. Note that the colour is defined by the sign on significance. Only non-
ggplot(df.processed[which(df.processed$Significance != 0),],
       aes(x=Significance,
           y=Name,
           color = Significance > 0)) +
  labs(x = "Significance",
       y = "Component Name",
       color = "Contribution\n") +
  scale_color_manual(labels = c("Negative", "Positive"),
                     values = c("red", "green")) +
       geom_segment(aes(x = 0,
                        y = Name,
                        xend = Significance,
                        yend = Name),
                    color = "grey50") +
       geom_point() +
  theme_bw() +
  theme(legend.position=c(0.9,0.12))
}
```

First a simple lasso is preformed as a point of comparison for when interaction terms are included.

```
#create a matrix & vector to store the input parameters
x <- model.matrix(medv ~ . , Boston)[, -1]
x <- scale(x)
y <- Boston$medv

#store all lambda values to test against
grid <- 10^seq(10, -5, length = 1000)

#run the lasso. alpha = 1 for lasso
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid, standardize=FALSE)
plot(lasso.mod)
```




```
#preform the cross validation
set.seed(42)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```
#we can now select the lowest lambda and use this to predict the output
bestlambda <- cv.out$lambda.min

#cross validated error
sprintf("Train cross validation error MSE for ideal Lasso model: %s", min(cv.out$cvm))
```

```
## [1] "Train cross validation error MSE for ideal Lasso model: 25.2200768683371"
```

```
#training error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[train, ])
MSE <- mean((lasso.pred - y[train])^2)
sprintf("Train MSE for ideal Lasso model: %s", MSE)
```

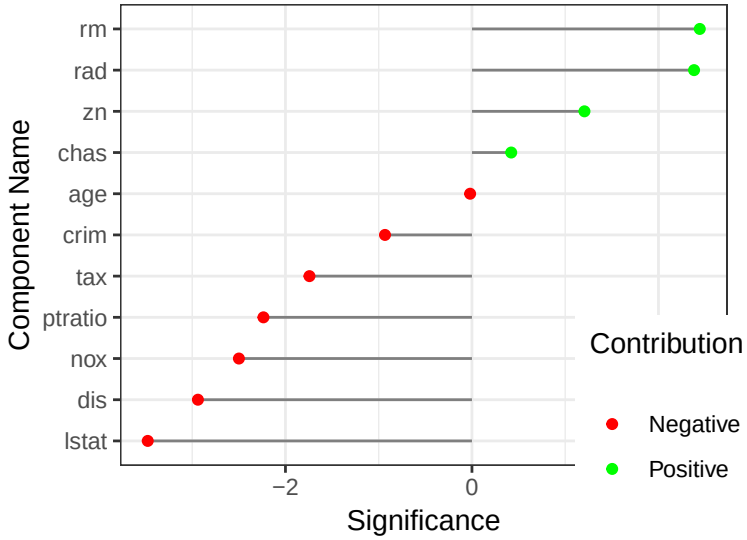
```
## [1] "Train MSE for ideal Lasso model: 22.7009069095856"
```

```
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[-train, ])
MSE <- mean((lasso.pred - y[-train])^2)
sprintf("Test MSE for ideal Lasso model: %s", MSE)
```

```
## [1] "Test MSE for ideal Lasso model: 19.4635184689217"
```

```
significancePlot(coef(cv.out, s = "lambda.min"))
```

```
##          Name Significance
##            12           11
```

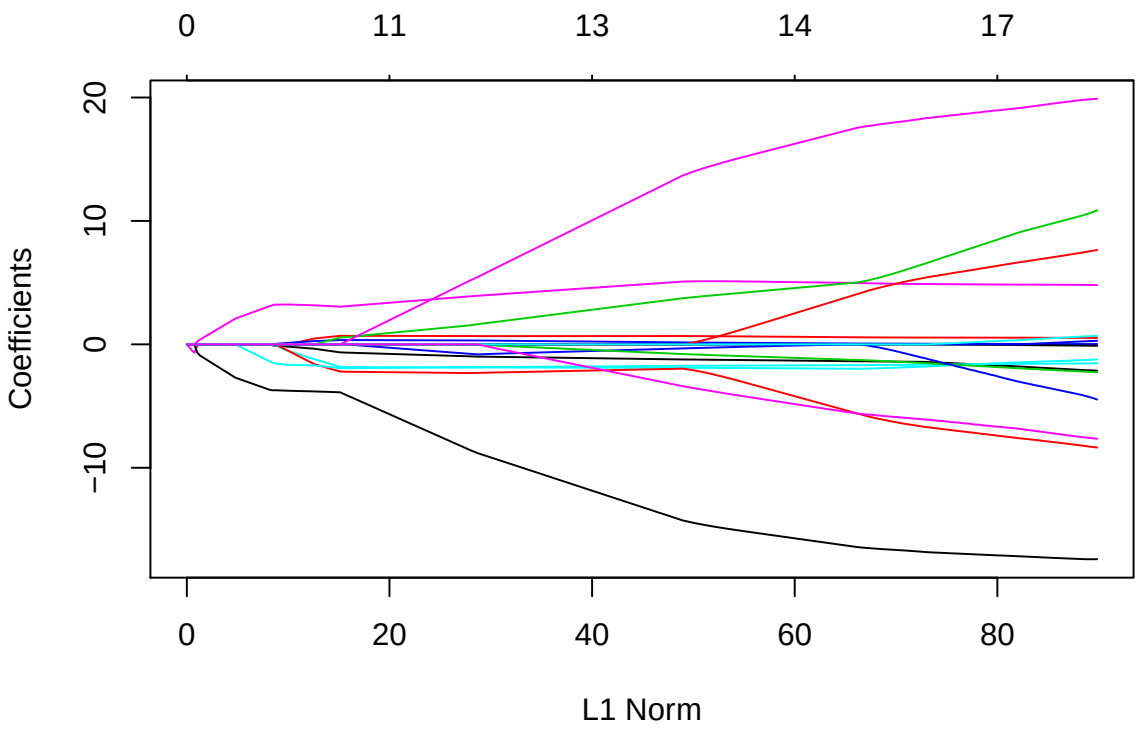


Next introduce the interaction terms to try and improve the model. Lasso is used to remove the irrelevent terms

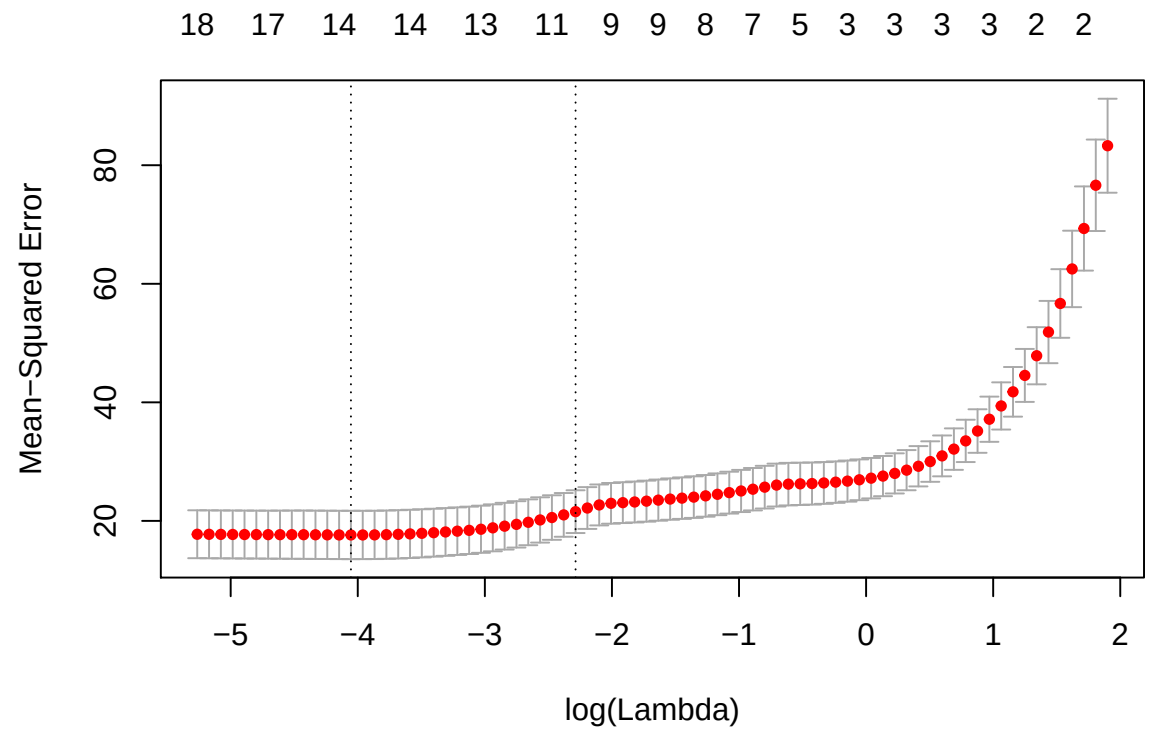
The following posible interaction terms will be examined:

1. rm <-> dis: average number of rooms per dwelling & weighted mean of distances to five Boston employment centres
2. dis <-> rad: weighted mean of distances to five Boston employment centres & index of accessibility to radial highways
3. tax <-> lstat: full-value property-tax rate & lower status of the population (percent).
4. nox <-> rad: nitrogen oxides concentration & index of accessibility to radial highways
5. crim <-> lstat: per capita crime rate by town & lower status of the population (percent).
6. lstat <-> rm: lower status of the population (percent) & average number of rooms per dwelling

```
x <- model.matrix(medv ~ . + rm:lstat +rm:dis + dis:rad + nox:rad + crim:lstat +tax:lstat,
                  Boston)[, -1]
x <- scale(x)
y <- Boston$medv

#store all lambda values to test against
grid <- 10^seq(10, -5, length = 1000)

#run the lasso. alpha = 1 for lasso
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid, standardize=FALSE)
plot(lasso.mod)
```




```
#preform the cross validation
set.seed(42)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```
#we can now select the lowest lambda and use this to predict the output
bestlambda <- cv.out$lambda.min
```

```
#cross validated error
sprintf("Train cross validation error for selected interaction term Masso model: %s", min(cv.out$cvm))
```

```
## [1] "Train cross validation error for selected interaction term Masso model: 17.6249416408094"
```

```
#training error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[train, ])
MSE <- mean((lasso.pred - y[train])^2)
sprintf("Train MSE for selected interaction terms Lasso model: %s", MSE)
```

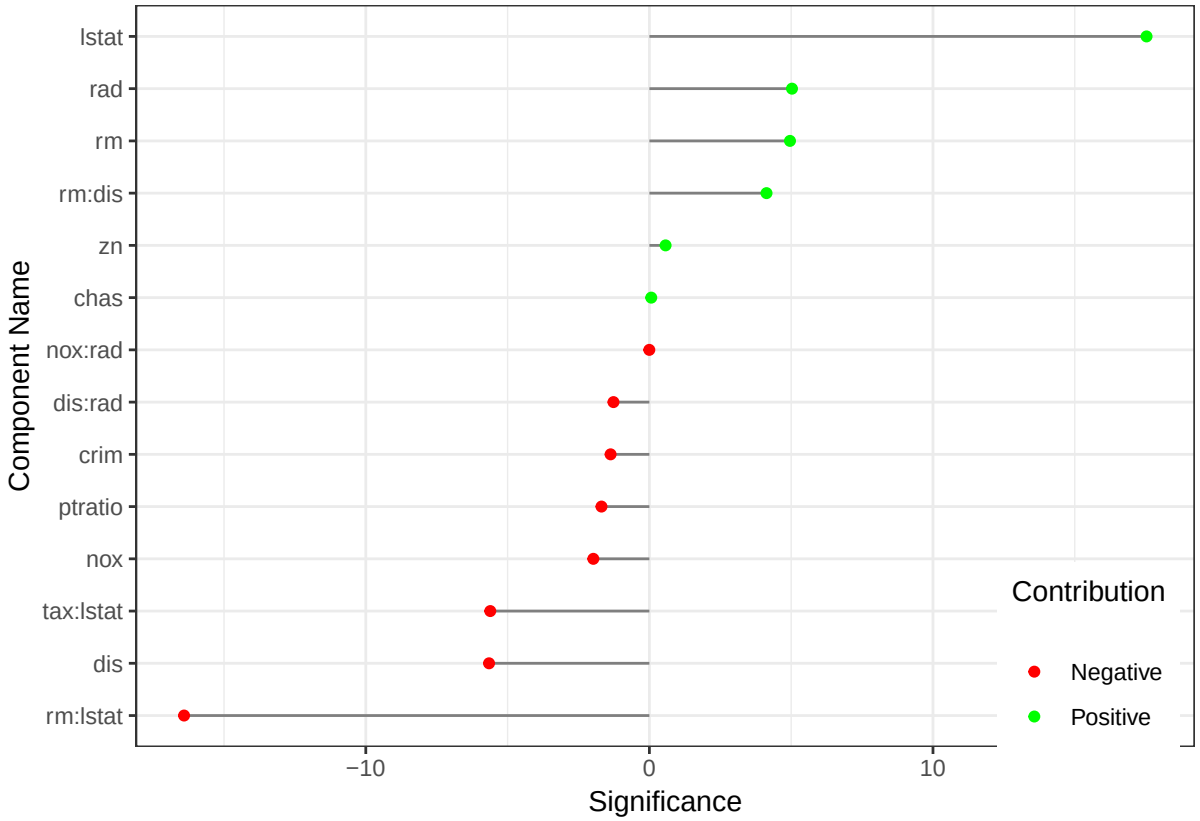
```
## [1] "Train MSE for selected interaction terms Lasso model: 14.7489554901412"
```

```
#testing error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[-train, ])
MSE <- mean((lasso.pred - y[-train])^2)
sprintf("Test MSE for selected interaction terms Lasso model: %s", MSE)
```

```
## [1] "Test MSE for selected interaction terms Lasso model: 14.6293555725652"
```

```
significancePlot(coef(cv.out, s = "lambda.min"))
```

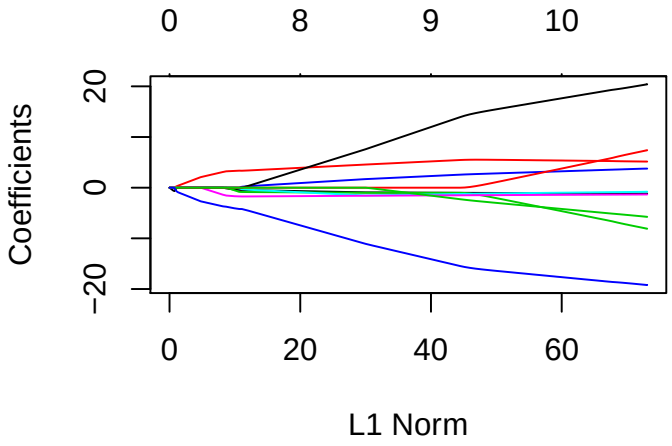
```
##          Name Significance
##            18           14
```



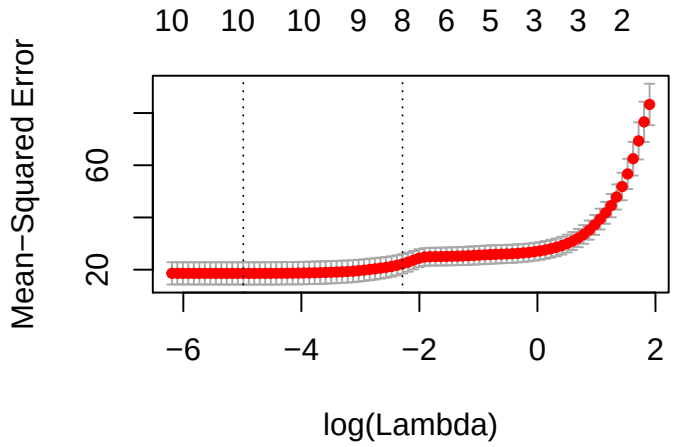
We can take the previous method of preforming the lasso and rebuild with fewer variables

```
x <- model.matrix(medv ~ . + rm*dis + tax*lstat + rm*lstat -age -chas -zn -nox -indus,
                  Boston)[, -1]
x <- scale(x)
y <- Boston$medv
```

```r
#store all lambda values to test against
grid <- 10^seq(10, -5, length = 1000)

#run the lasso. alpha = 1 for lasso
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid, standardize=FALSE)
plot(lasso.mod)
```




```r
#preform the cross validation
set.seed(42)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```r
#we can now select the lowest lambda and use this to predict the output
bestlambda <- cv.out$lambda.min

#cross validation testing error
sprintf("Cross validation testing error for refined model %s", min(cv.out$cvm))
```

```
## [1] "Cross validation testing error for refined model 18.5784688587374"
```

```r
#training error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[train, ])
MSE <- mean((lasso.pred - y[train])^2)
sprintf("Train MSE for lasso with interaction terms and remove irrelevent terms: %s", MSE)
```

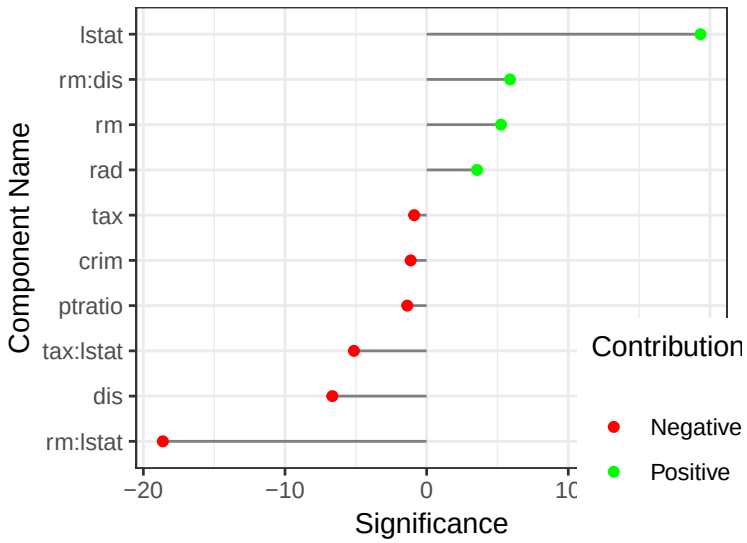
```
## [1] "Train MSE for lasso with interaction terms and remove irrelevent terms: 16.4771801289492"
```

```
#test mse
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[-train, ])
MSE <- mean((lasso.pred - y[-train])^2)
sprintf("Test MSE for lasso with interaction terms and remove irrelevent terms: %s", MSE)
```

```
## [1] "Test MSE for lasso with interaction terms and remove irrelevent terms: 15.736727793155"
```

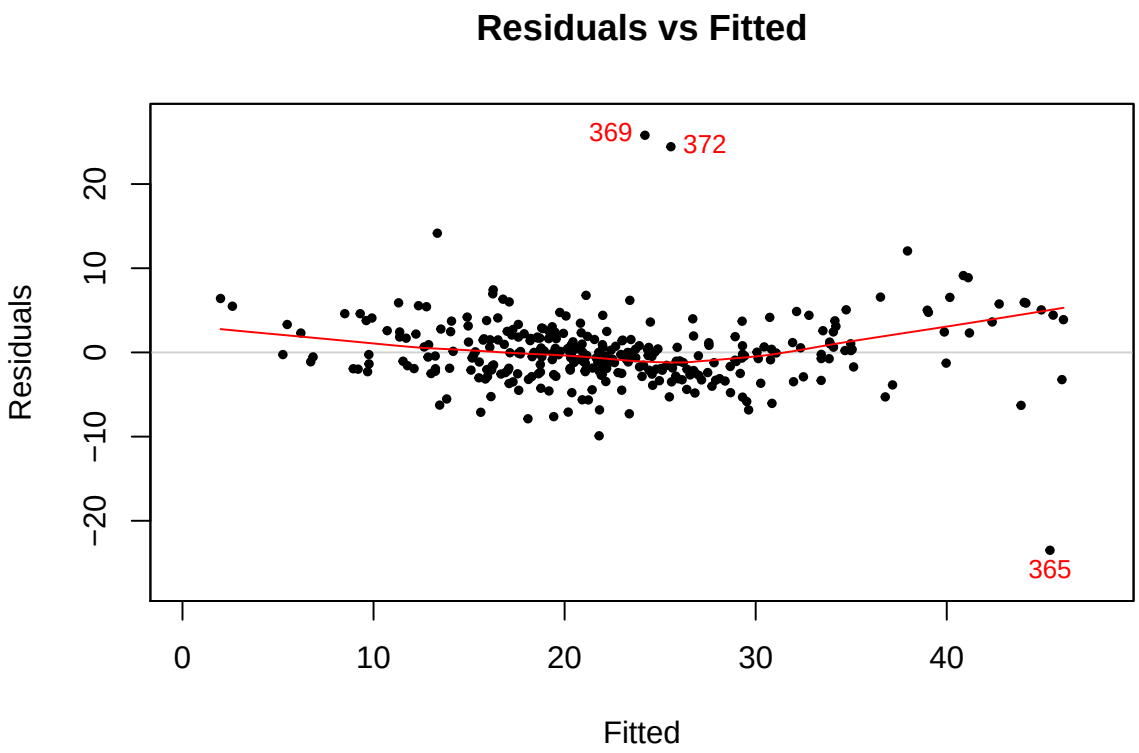
```
significancePlot(coef(cv.out, s = "lambda.min"))
```

```
##          Name Significance
##            10           10
```

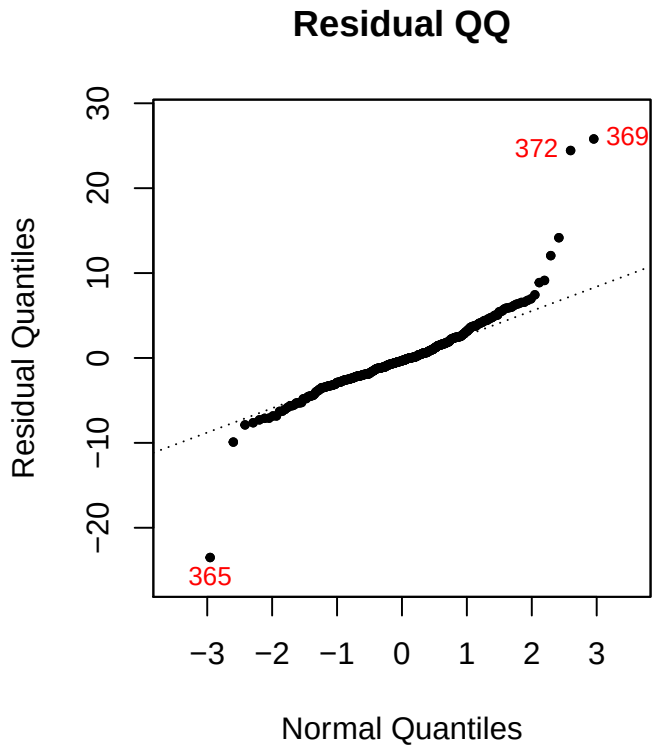


A breif residual diagnosis is now done on this refined model.

```
plotres(lasso.mod, which = 3)
```




```
plotres(lasso.mod, which = 4)
```

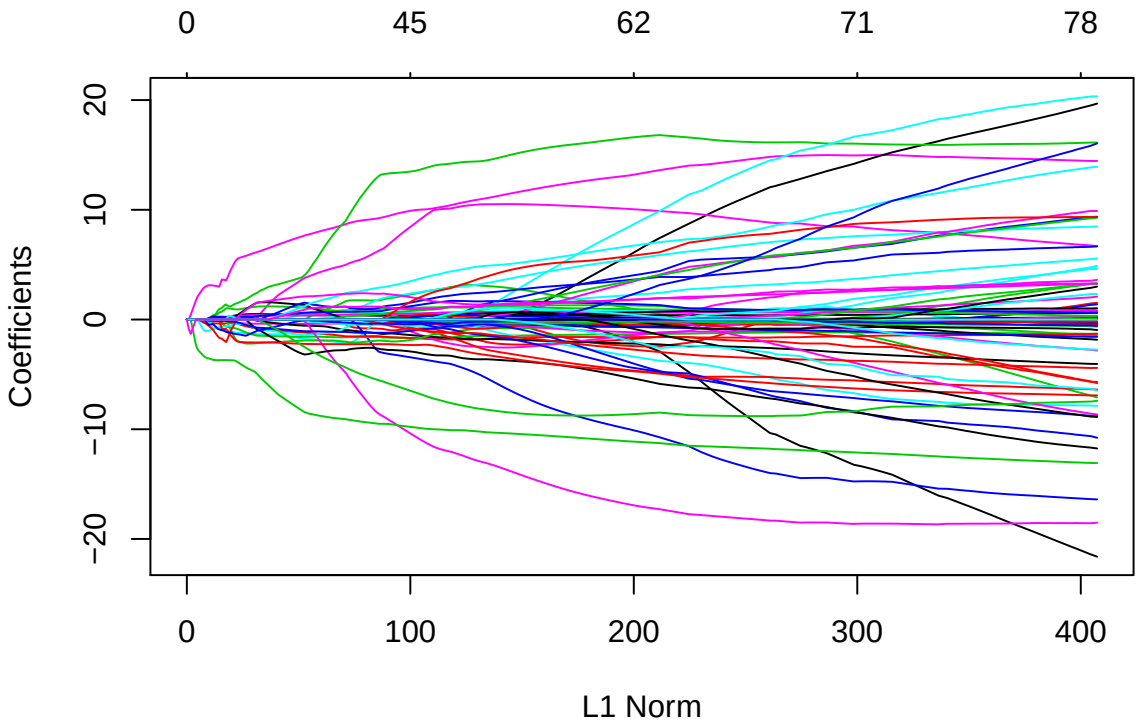


After this, a more itterative process will be preformed to try and derive the best posible model interms of lowest MSE resulting in highest prediction accuracy. A number of new methods will be introduced in this process, including Ridge regression, Lasso regression and Elastic net regression. Tree based methods are used

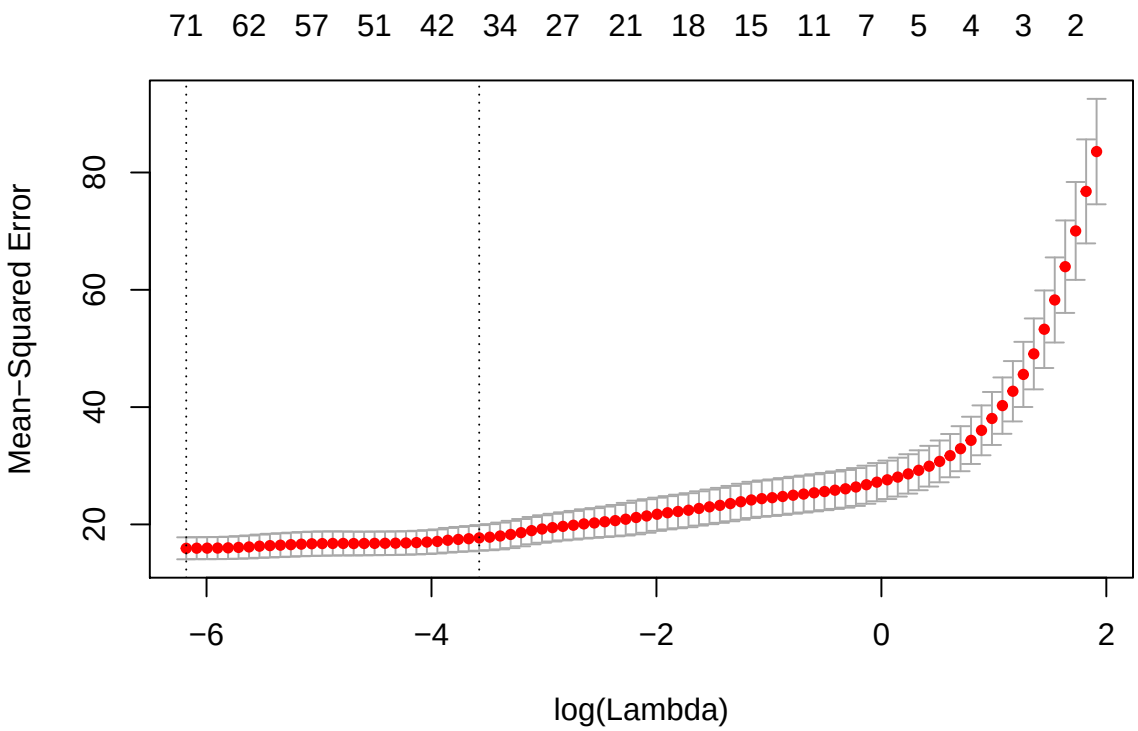
in the question 1.c

We begin by applying lasso over all terms and letting it select the best interaction terms

```
x <- model.matrix(medv ~ .^2, Boston)[, -1]
x <- scale(x)
y <- Boston$medv

#store all lambda values to test against
grid <- 10^seq(10, -5, length = 1000)

#run the lasso. alpha = 1 for lasso
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, lambda = grid, standardize=FALSE)
plot(lasso.mod)
```




```
#preform the cross validation
set.seed(26)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```




```
#we can now select the lowest lambda and use this to predict the output
bestlambda <- cv.out$lambda.1se

#cross validation testing error
sprintf("Cross validation testing error for refined model %s", min(cv.out$cvm))
```

```
## [1] "Cross validation testing error for refined model 15.9295090486571"
```

```
#train error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[train, ])
MSE <- mean((lasso.pred - y[train])^2)
sprintf("Train MSE for ideal Lasso model: %s", MSE)
```

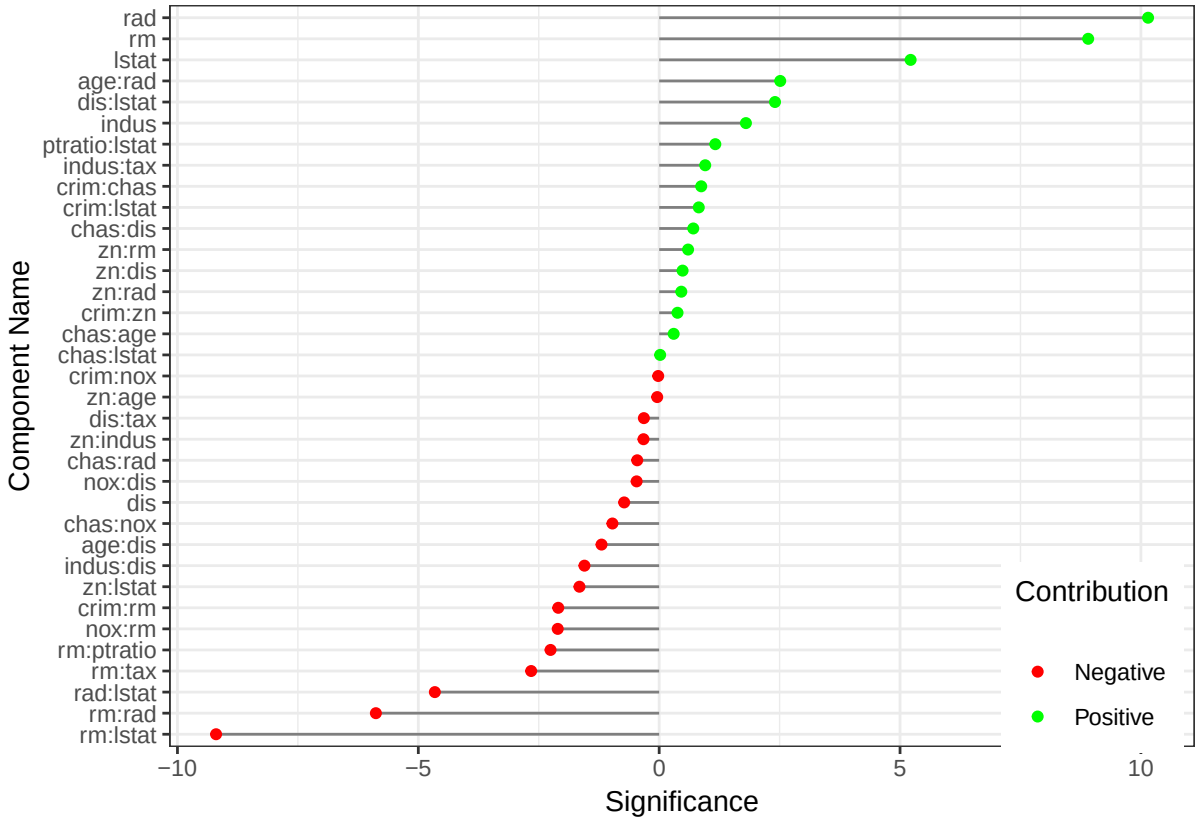
```
## [1] "Train MSE for ideal Lasso model: 11.0862189979485"
```

```
#test error
lasso.pred <- predict(lasso.mod, s = bestlambda, newx = x[-train, ])
MSE <- mean((lasso.pred - y[-train])^2)
sprintf("Test MSE for ideal Lasso model: %s", MSE)
```

```
## [1] "Test MSE for ideal Lasso model: 14.4212350066642"
```

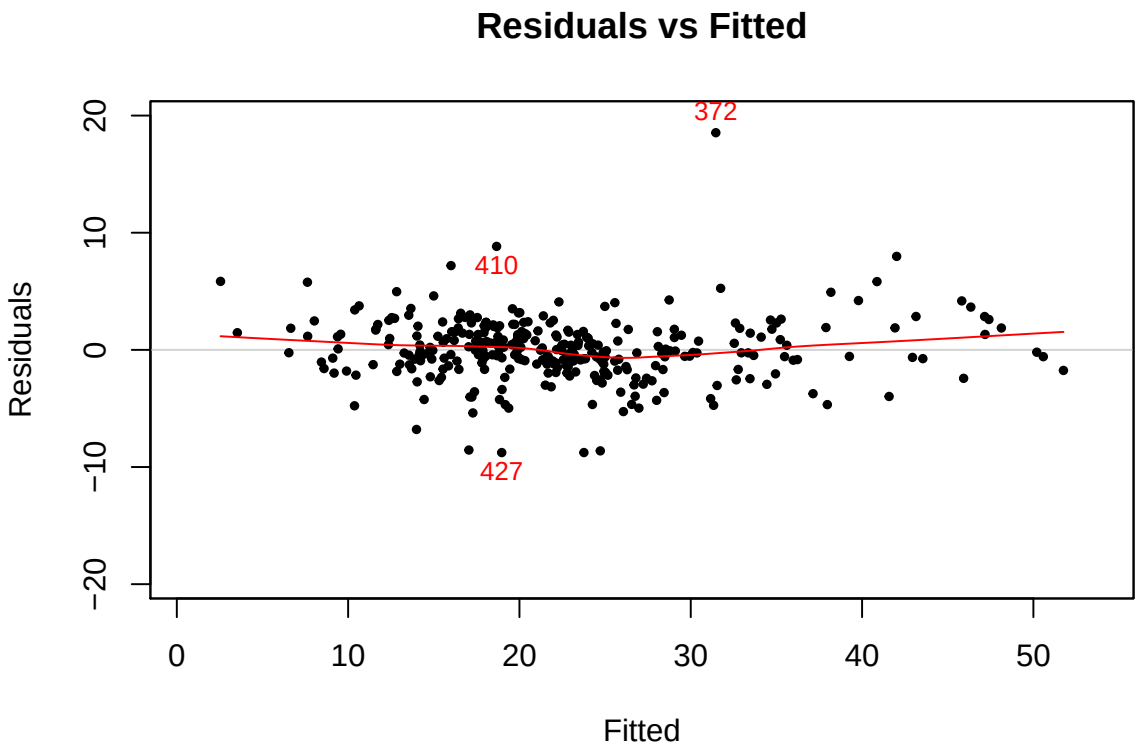
```
significancePlot(coef(cv.out, s = "lambda.1se"))
```

```
##         Name Significance
##           78           35
```

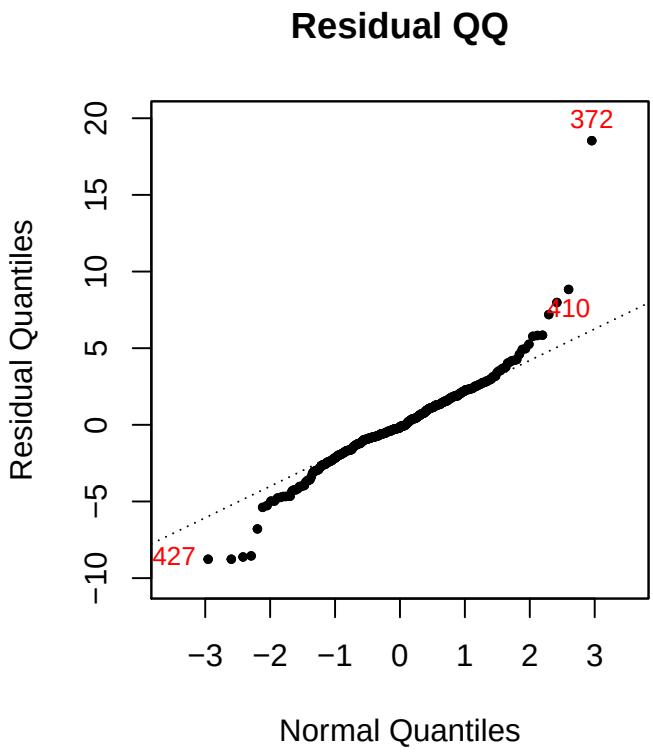


Plot the Residual vs fitted and QQ plots for the lasso model

```
plotres(lasso.mod, which = 3)
```



```r
plotres(lasso.mod, which = 4)
```



5.1 Cross validation controller

Each model constructed will use repeated Cross validation to select the hyper marameters. To this end, a controller is used to preform repeated cross validation on each of the models that follow. The data sample is shuffled prior to each repetition, which results in a different split of the sample being used for each round of cross validation. This is done to reduce the effect of one variable being dependent on another and to reduce the estimator bias.

```r
# this set of functions come from the caret package and make the process
# of generating models easier, more reproducable and presentable
controller <- trainControl(
  method = "repeatedcv", # repeated cross validation
  number = 10, # k = 10 folds
  repeats = 5, # CV is done 5 times with 5 diffrent sets of k splits
  verboseIter = F
)
```

5.2 Ridge Regression

First, a Ridge regression model is fitted. Second order interaction terms are included in the model to allow for higher quality model fits. The theory and justification for using the ridge regression process can be found in the report. The ridge regression model aims to minimize the following expression.

$$\sum_{i=1}^{n}(y_i - \hat{y}_i) + \lambda \sum_{j}^{m} \beta_j^2$$

```
ridge <- train(medv ~ .,
               data = Boston,
               subset = train,
               method = 'glmnet',
               tuneGrid = expand.grid(
                 alpha = 0, # alpha = 0 for ridge regression
                 #generate a series of 20 lambdas from 0.0001 up to 1
                 lambda = seq(0.4, 0.8,length = 150)),
               #use the controller to do cross validation for each lambda selection step
               trControl = controller)

mse.value <- mse(ridge)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated ridge",
                                     orderOfInteractionTerms=1,
                                     mse.train=mse.generic(ridge,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(ridge$finalModel,
                                                    s = ridge$bestTune$lambda)!=0))))
sprintf("Test MSE for Ridge regression with first order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Ridge regression with first order interaction terms: 19.8382281223397"
```

```
ridge2 <- train(medv ~ .^2,
               data = Boston,
               subset = train,
               method = 'glmnet',
               tuneGrid= expand.grid(
                 alpha = 0, # alpha = 0 for ridge regression
                 #generate a series of 20 lambdas from 0.0001 up to 1
                 lambda = seq(0.4, 0.8,length = 150)),
               trControl = controller)

# Print the MSE
mse.value <- mse(ridge2)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated ridge",
                                     orderOfInteractionTerms=2,
                                     mse.train=mse.generic(ridge2,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(ridge2$finalModel,
                                                    s = ridge2$bestTune$lambda)!=0))))
sprintf("Test MSE for Ridge regression with second order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Ridge regression with second order interaction terms: 17.3643357985172"
```

5.3 Lasso Regression

Next, a Lasso is used to try and furter reduce the model error. Lasso is similar to Ridge except it is able to compleatly remove interaction terms that are not required by the model. Lasso aims to minamize the following expression:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i) + \lambda \sum_{j}^{m} |\beta_j|$$

```
lasso <- train(medv ~ .,
               data=Boston,
               subset = train,
               method = 'glmnet',
               tuneGrid= expand.grid(
                 alpha = 1,
                 lambda = seq(0.00001, 0.1,length = 150)),
               trControl = controller)

mse.value <- mse(lasso)
sprintf("Test MSE for Lasso regression with first First interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Lasso regression with first First interaction terms: 19.4540023312528"
```

```
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated lasso",
                                     orderOfInteractionTerms=1,
                                     mse.train=mse.generic(lasso,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(lasso$finalModel,
                                                    s = lasso$bestTune$lambda)!=0))))
#Second order
lasso2 <- train(medv ~ .^2,
               data=Boston,
               subset = train,
               method = 'glmnet',
               tuneGrid= expand.grid(
                 alpha = 1,
                 lambda = seq(0.00001, 0.1,length = 150)),
               trControl = controller)

mse.value <- mse(lasso2)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated lasso",
                                     orderOfInteractionTerms=2,
                                     mse.train=mse.generic(lasso2,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric
                                     (colSums(coef(lasso2$finalModel,
                                                   s = lasso2$bestTune$lambda)!=0))))
sprintf("Test MSE for Lasso regression with second order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Lasso regression with second order interaction terms: 11.6439085318765"
```

```
#third order
lasso3 <- train(medv ~ .^3,
               data=Boston,
               subset = train,
               method = 'glmnet',
               tuneGrid= expand.grid(
```

```
                  alpha = 1,
                  lambda = seq(0.00001, 0.1,length = 150)),
                trControl = controller)

mse.value <- mse(lasso3)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated lasso",
                                     orderOfInteractionTerms=3,
                                     mse.train=mse.generic(lasso3,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(lasso3$finalModel,
                                                    s = lasso3$bestTune$lambda)!=0))))
sprintf("Test MSE for Lasso regression with third order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Lasso regression with third order interaction terms: 15.3630782022589"
```

5.4 Elastic Net regression

Lastly, an Elastic net is fitted onto the data set. Elastic net uses a combination of Ridge and Lasso regression such that it has two tuning parameters λ and α. The closer to 1 α is the more the Elastic net acts like a Lasso and the closer to 0 it gets the more it acts like a ridge. In this way the elastic net is able to identify the optimum combination of ridge and lasso to fit a given model. The elastic net aims to minimize:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i) + \lambda\left((1-\alpha)\sum_{j}^{m}\beta_j^2 + \alpha\sum_{j}^{m}|\beta_j|\right)$$

```
elasticNet <- train(medv ~.,
                data=Boston.train,
                method = 'glmnet',
                tuneGrid= expand.grid(
                  alpha = seq(0, 1,length = 5),
                  lambda = seq(0, 1,length = 5)),
                trControl = controller)

mse.value <- mse(elasticNet)

modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated elastic net",
                                     orderOfInteractionTerms=1,
                                     mse.train=mse.generic(elasticNet,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(elasticNet$finalModel,
                                                    s = elasticNet$bestTune$lambda)!=0))))
sprintf("Test MSE for Elastic net regression with first order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Elastic net regression with first order interaction terms: 19.5539825192154"
```

```
# second order elastic net
elasticNet2 <- train(medv ~.^2,
                data=Boston.train,
                method = 'glmnet',
```

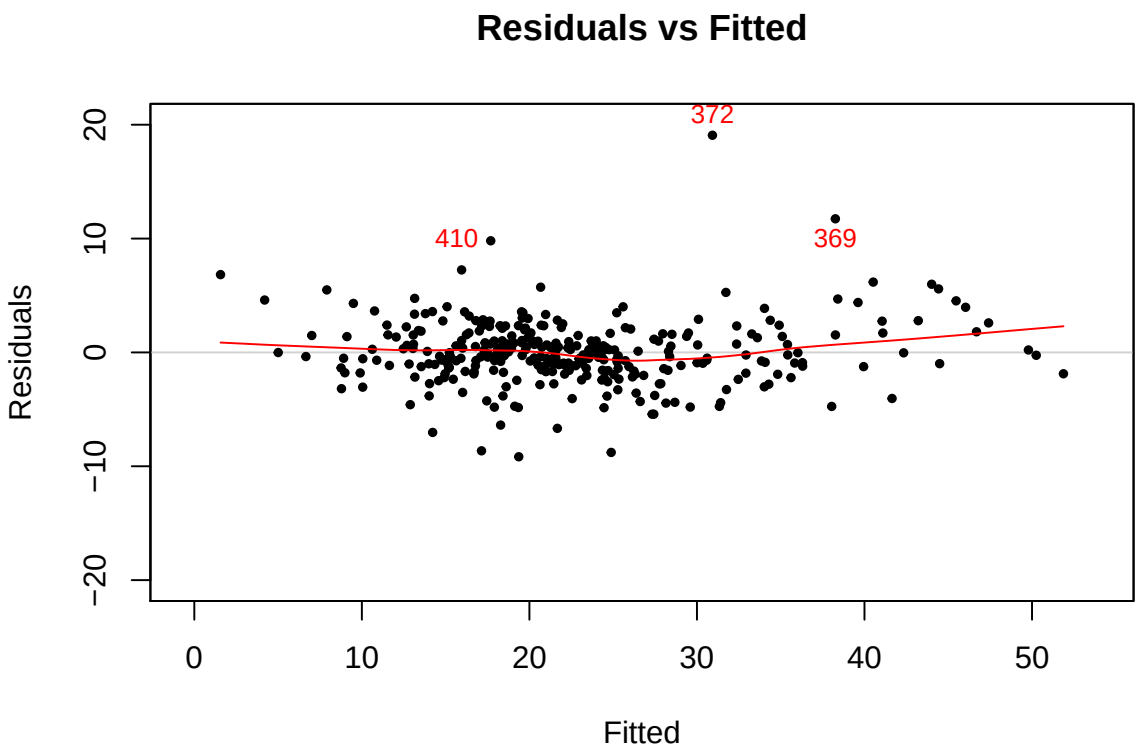
```
                tuneGrid= expand.grid(
                  alpha = seq(0, 0.1,length = 5),
                  lambda = seq(0, 0.2,length = 5)),
                trControl = controller)

mse.value <- mse(elasticNet2)

modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated elastic net",
                                     orderOfInteractionTerms=2,
                                     mse.train=mse.generic(elasticNet2,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(elasticNet2$finalModel,
                                                    s = elasticNet2$bestTune$lambda)!=0))))
sprintf("Test MSE for Elastic net regression with second order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Elastic net regression with second order interaction terms: 11.3137028528586"
```

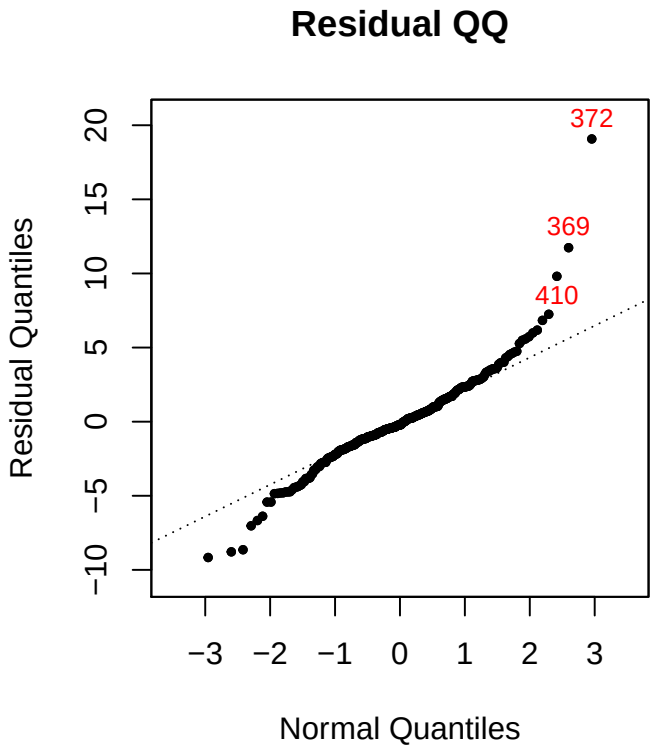
```
#Third order elastic net
elasticNet3 <- train(medv ~.^3,
                data=Boston.train,
                method = 'glmnet',
                tuneGrid= expand.grid(
                  alpha = seq(0, 0.1,length = 5),
                  lambda = seq(0, 0.2,length = 5)),
                trControl = controller)

mse.value <- mse(elasticNet2)

modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Cross validated elastic net",
                                     orderOfInteractionTerms=3,
                                     mse.train=mse.generic(elasticNet3,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       colSums(coef(elasticNet3$finalModel,
                                                    s = elasticNet3$bestTune$lambda)!=0))))
sprintf("Test MSE for Elastic net regression with third order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Elastic net regression with third order interaction terms: 11.3137028528586"
```

```
plotres(elasticNet2, which = 3)
```




```
plotres(elasticNet2, which = 4)
```



5.5 Stepped AIC

we can look at including all posible interaction terms within the linear model and then identifying the one with the lowest error through a reccursive stepped AIC process

```
lm.fit <- lm(medv ~ ., data = Boston, subset = train)

stepped <- train(medv ~.,
                 data=Boston.train,
                 method = 'glmStepAIC',
                 scope = list(
                    upper=medv~.,
                    lower = medv ~ 1),
                 trControl = controller,
                 trace=FALSE)

mse.value <- mse(stepped)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Stepped",
                                     orderOfInteractionTerms=1,
                                     mse.train=mse.generic(stepped,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       length(coef(stepped$finalModel)))
                                     ))
sprintf("Test MSE for Stepped model with first order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Stepped model with first order interaction terms: 19.5770482601268"
```

```
# second order stepped
stepped2 <- train(medv ~.,
                 data=Boston.train,
                 method = 'glmStepAIC',
                 scope = list(
                    upper=medv~.^2,
                    lower = medv ~ 1),
                 trControl = controller,
                 trace=FALSE)

mse.value <- mse(stepped2)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Stepped",
                                     orderOfInteractionTerms=2,
                                     mse.train=mse.generic(stepped2,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(
                                       length(coef(stepped2$finalModel)))
                                     ))
sprintf("Test MSE for Stepped model with second order interaction terms: %s", mse.value)
```

```
## [1] "Test MSE for Stepped model with second order interaction terms: 12.7525115451988"
```

```
stepped3 <- step(lm.fit,
                  scope = list(
                    upper=medv~.^3,
                    lower = medv ~ 1),
```

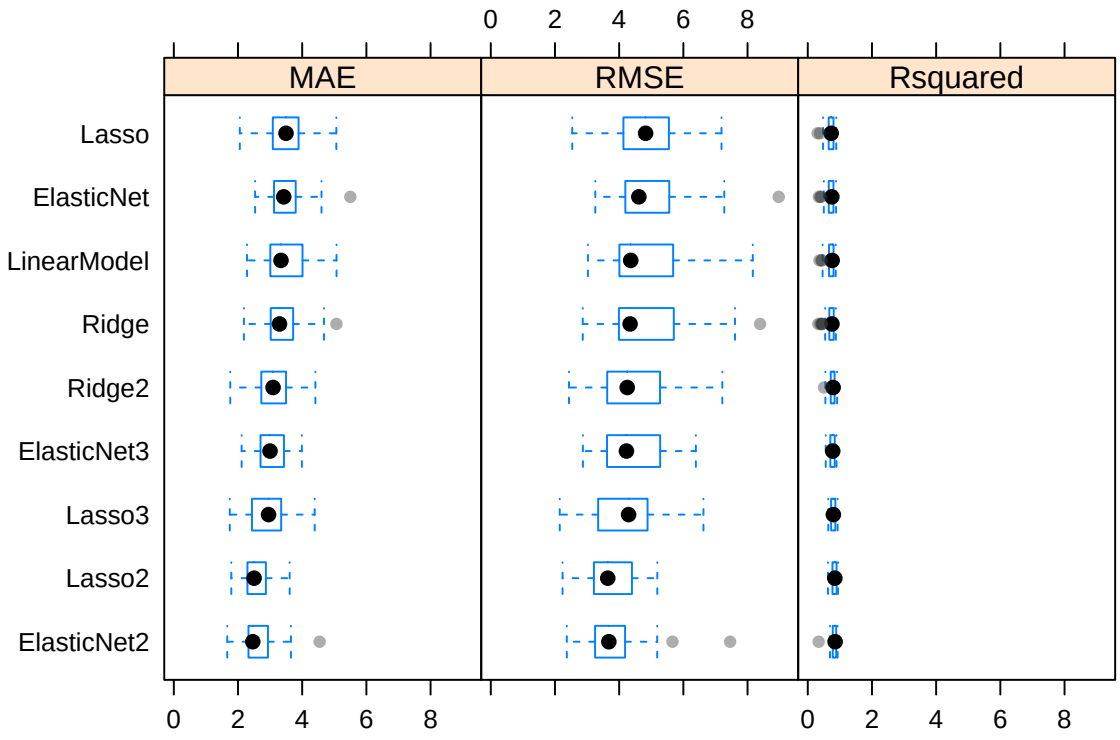
```
                direction= "both",
                steps = 1000,
                trace=FALSE)

mse.value <- mse(stepped3)
modelPreformance <- rbind(modelPreformance,
                          data.frame(modelName="Stepped",
                                     orderOfInteractionTerms=3,
                                     mse.train=mse.generic(stepped3,Boston.train),
                                     mse.test=mse.value,
                                     interactionTerms=as.numeric(length(coef(stepped3)))
                                    ))
sprintf("Test MSE for Stepped model with third order interaction terms %s", mse.value)
```

```
## [1] "Test MSE for Stepped model with third order interaction terms 14.1517460979673"
```

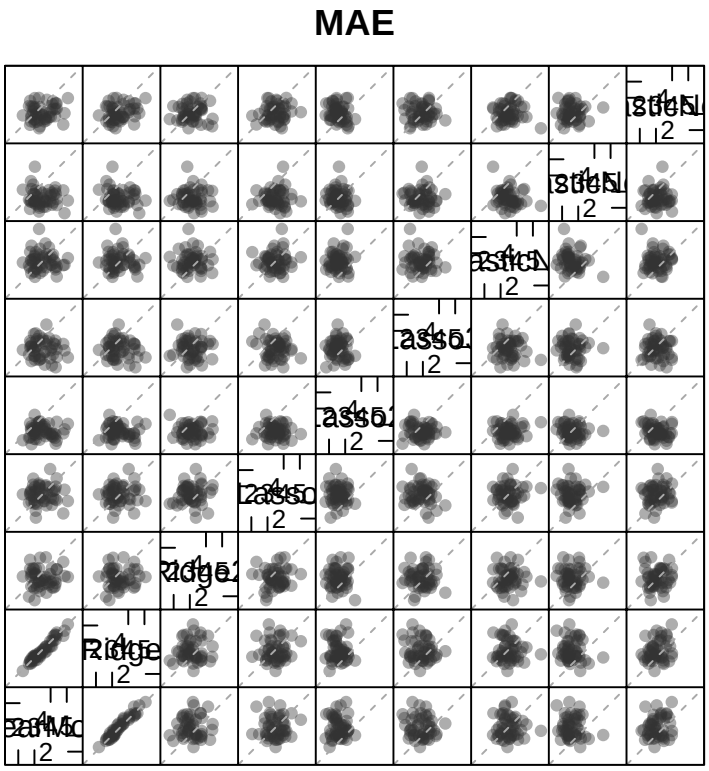
5.6 Model comparison

The last thing to do is to compare the diffrent models and look for the one that produced the best fit overall.

```
model_list <- list(LinearModel = linear, Ridge = ridge, Ridge2 = ridge2, Lasso = lasso,  Lasso2 = lasso
res <- resamples(model_list)
#summary(res)
bwplot(res)
```




```
splom(res)
```



Scatter Plot Matrix

```
modelPreformance[order(modelPreformance$mse.train),]
```

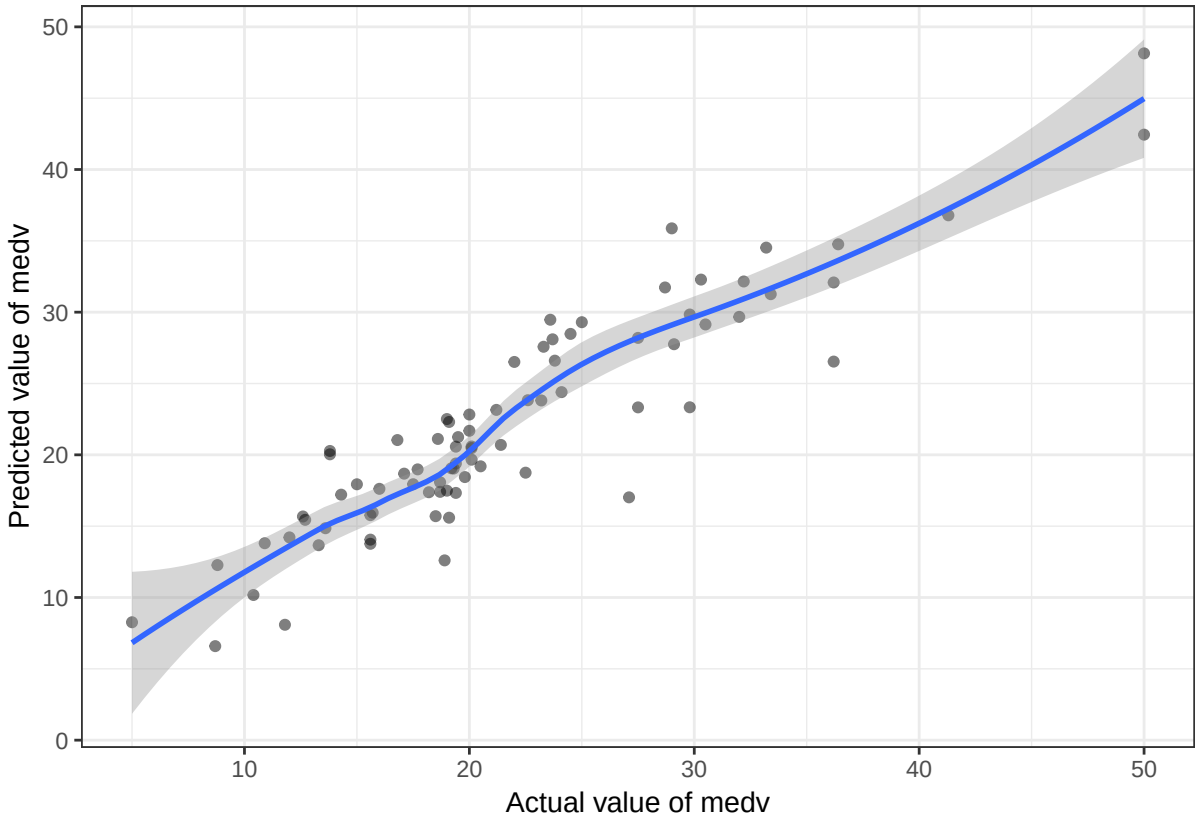
```
##                       modelName orderOfInteractionTerms mse.train mse.test
## 11                      Stepped                       3  7.352430 14.15175
## 4         Cross validated lasso                       2  7.721433 11.64391
## 7   Cross validated elastic net                       2  8.202387 11.31370
## 10                      Stepped                       2  8.428501 12.75251
## 8   Cross validated elastic net                       3 10.785709 11.31370
## 5         Cross validated lasso                       3 11.920960 15.36308
## 2         Cross validated ridge                       2 15.999352 17.36434
## 6   Cross validated elastic net                       1 22.648507 19.55398
## 9                       Stepped                       1 22.651833 19.57705
## 3         Cross validated lasso                       1 22.722959 19.45400
## 1         Cross validated ridge                       1 23.255275 19.83823
##    interactionTerms
## 11               41
## 4                73
## 7                76
## 10               30
## 8               135
## 5                40
## 2                79
## 6                13
## 9                11
## 3                12
## 1                13
```

Now that we have identified the best posible model we can plot the test results and compare the diffrence

between that the inclusion of interaction terms creates.

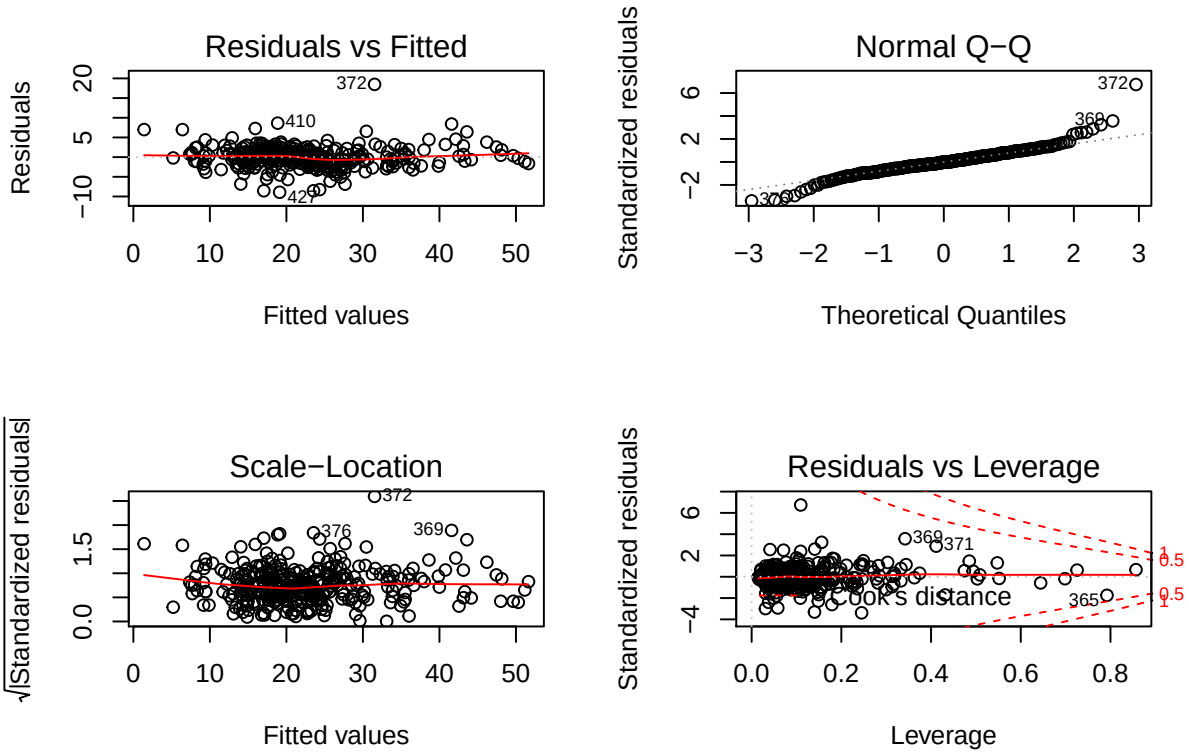
```
  pred <- predict(elasticNet2,Boston.test)
  ggplot(data= Boston.test, aes(medv,pred)) +
  geom_point(alpha=0.5) +
  stat_smooth() +
  xlab('Actual value of medv') +
  ylab('Predicted value of medv')+
  theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

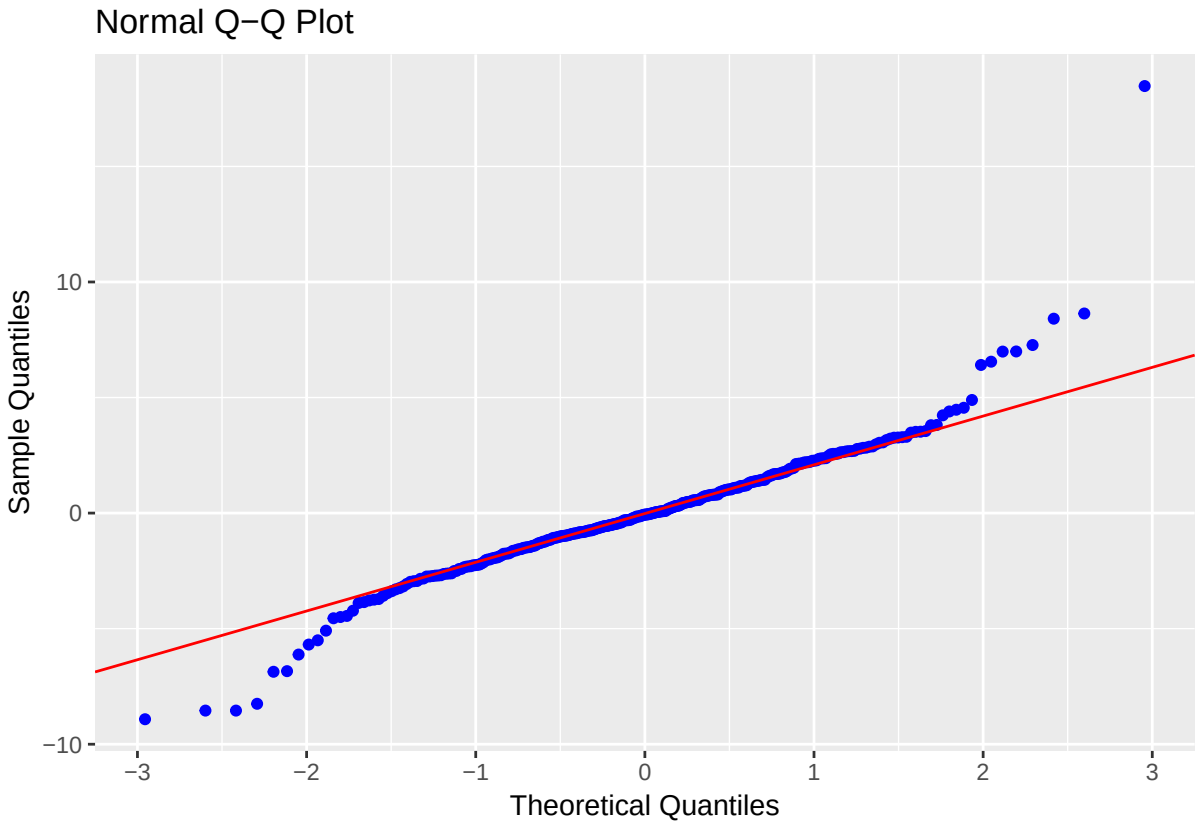


Now that we have the ideal model we can look into preforming residual diagnostics of the final model.

```
par(mfrow = c(2, 2))
plot(stepped3)
```




```
#summary(stepped3)

#residual diagnosis
ols_plot_resid_qq(stepped3)
```



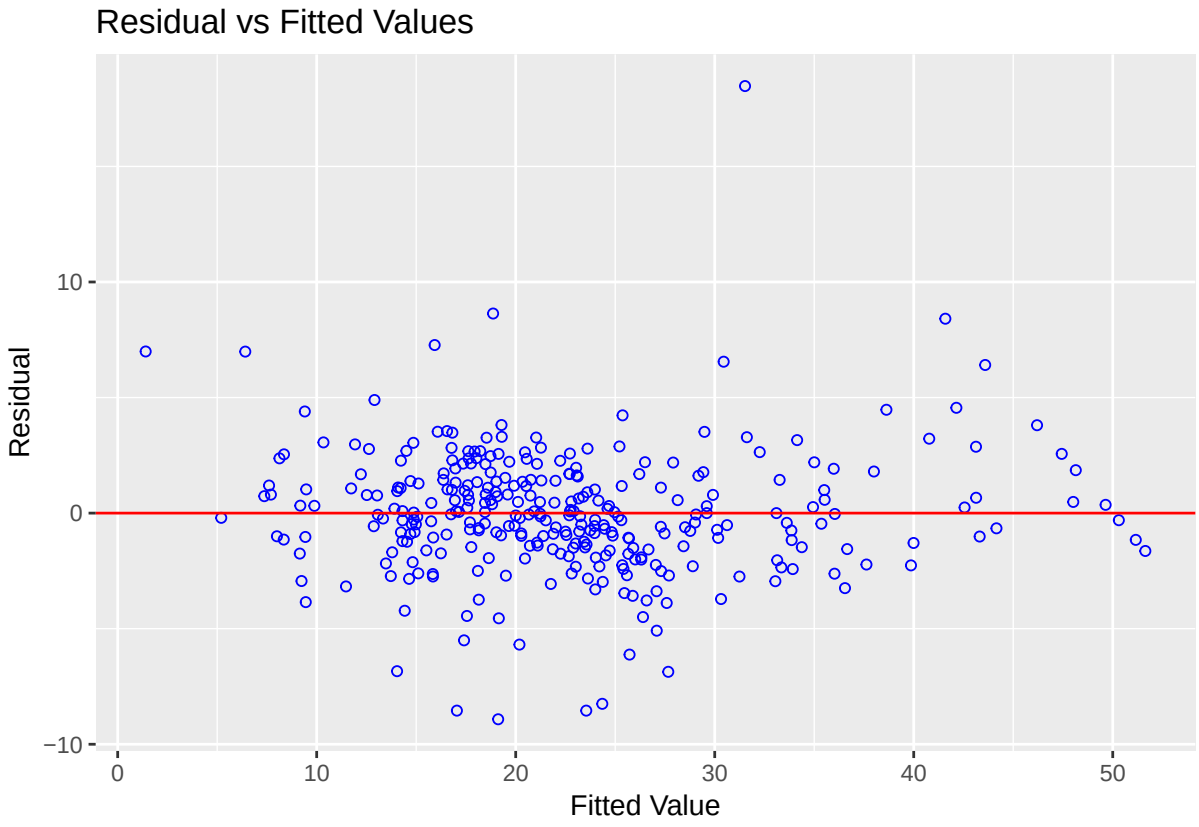

```
ols_test_normality(stepped3)
```

```
## -----------------------------------------------
##        Test             Statistic       pvalue
## -----------------------------------------------
## Shapiro-Wilk              0.9246         0.0000
## Kolmogorov-Smirnov        0.0603         0.1951
## Cramer-von Mises         22.3019         0.0000
## Anderson-Darling          3.3475         0.0000
## -----------------------------------------------
```

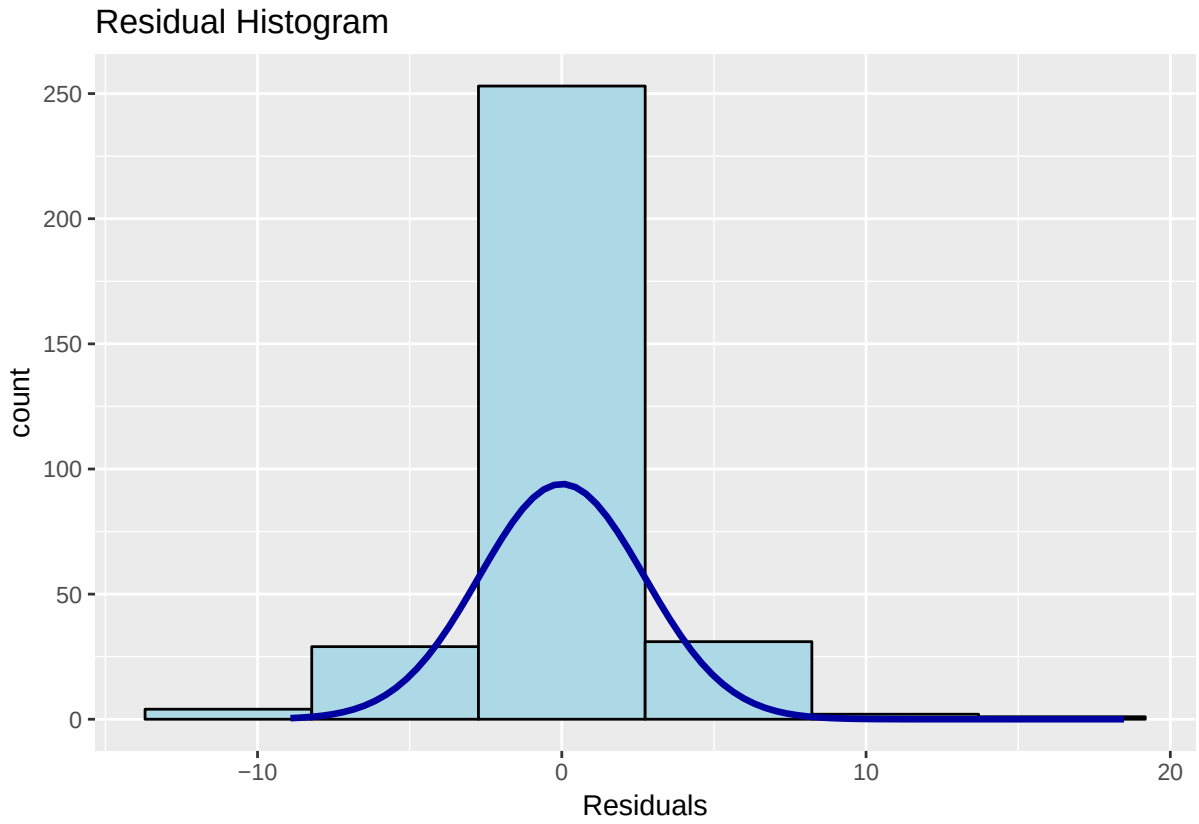
```
ols_test_correlation(stepped3)
```

```
## [1] 0.9584107
```

```
ols_plot_resid_fit(stepped3)
```




```
ols_plot_resid_hist(stepped3)
```



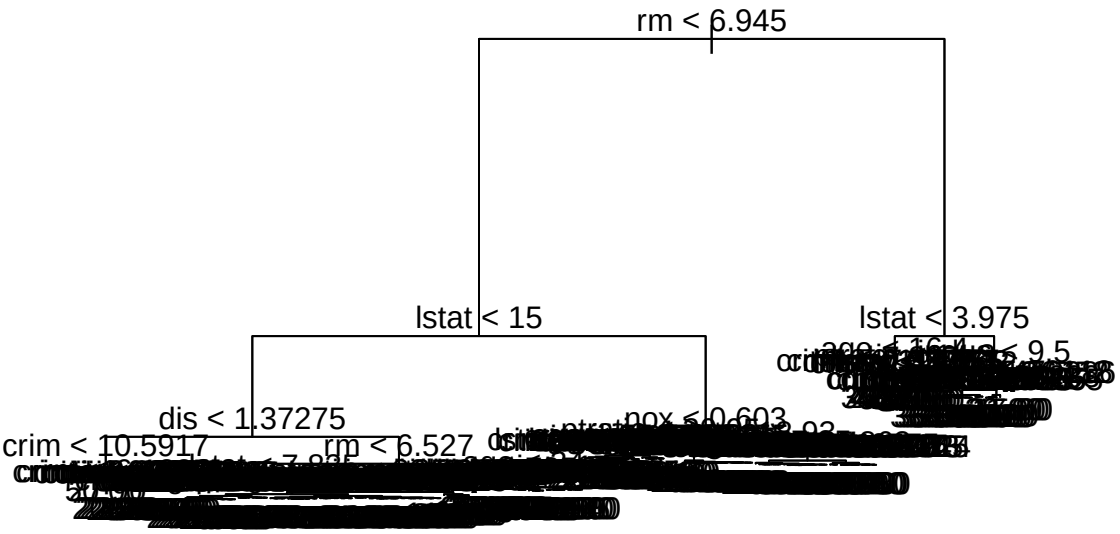

```
ols_test_breusch_pagan(stepped3)
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                Data
##  -------------------------------
##  Response : medv
##  Variables: fitted values of medv
##
##         Test Summary
##  ----------------------------
##  DF            =    1
##  Chi2          =    3.132442
##  Prob > Chi2   =    0.0767487
```

6 Question 1.c

Next, the use of a regression tree to model is explored. First, a large regression tree is fitted to the data such that it perfectly predicts the test cases. After this, the tree is pruned down through a process of cross validation. Note that I did not use the train method from the caret packet here becuse it obverscates some of the steps in generating the tree using `rpart` that were undesirable.

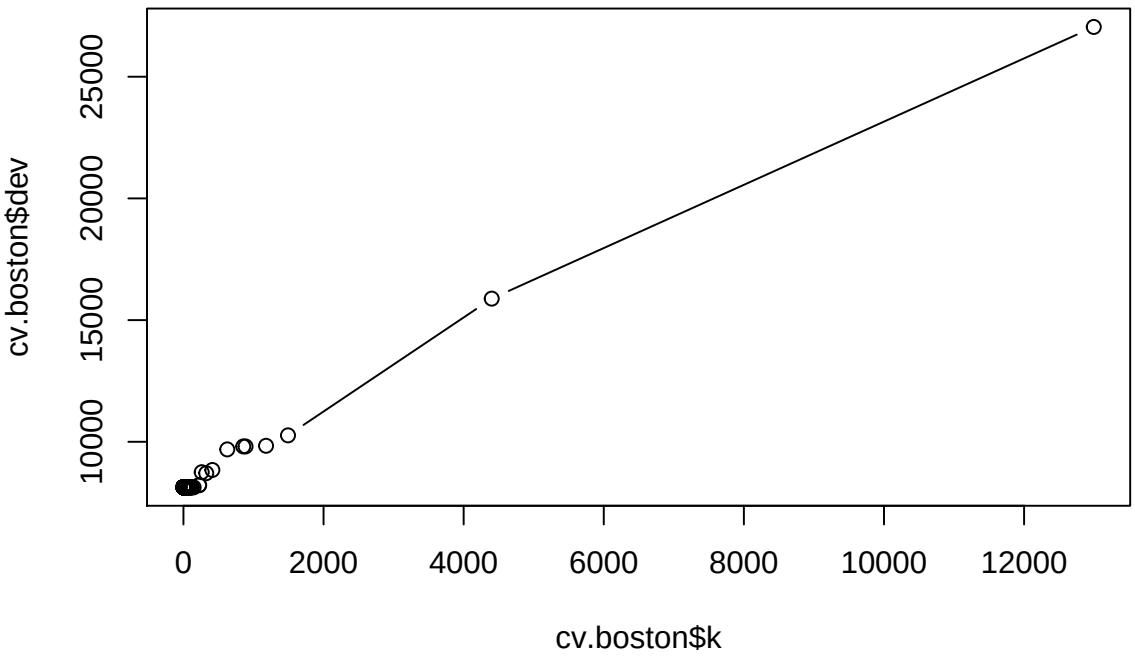
```r
#first we build the full tree on the whole set.
#tree.control is used to ensure we build a large tree that is overfitted
tree.boston <- tree(medv ~ .,
                    data=Boston,
                    control = tree.control(nobs = 320,
                     mindev = 0,
                     minsize = 1),
                    subset = train)
plot(tree.boston)
text(tree.boston, pretty = 0)
```



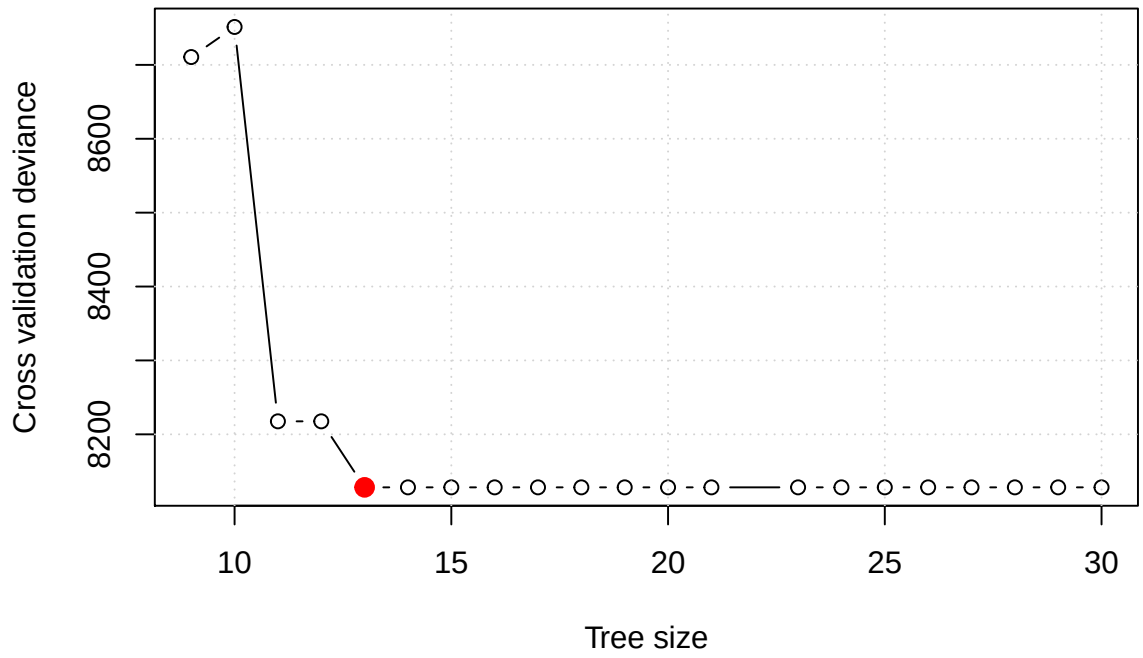

```r
#summary(tree.boston)

#we can look at the training and test error for the full tree
yhat_train <- predict(tree.boston, newdata = Boston.train)
MSE <- mean((yhat_train - Boston.train$medv)^2)
sprintf("Train MSE for full tree: %s", MSE)
```

```
## [1] "Train MSE for full tree: 0.00185416666666668"
```

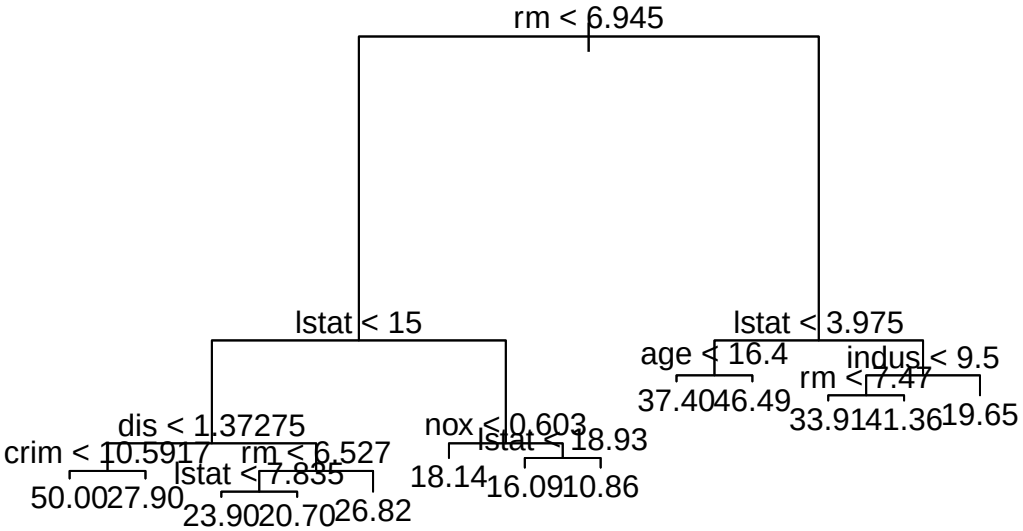
```r
#Next we preform cross validation on the tree to identify what the ideal size is
set.seed(26) #set seed before doing cross validation to make it reproducable
cv.boston <- cv.tree(tree.boston, FUN=prune.tree, K=5)


plot(cv.boston$k, cv.boston$dev, type = "b")
```




```
#zoom in on the smaller side of the spectrum to clearly see the turning point
plot(cv.boston$size[160:180],
     cv.boston$dev[160:180],
     type = "b",
     xlab="Tree size",
     ylab= "Cross validation deviance")
grid()


#the ideal tree size is the one that minimises the cross-validated error
tree.min <- which.min(rev(cv.boston$dev))
points(tree.min, cv.boston$dev[tree.min], col = "red", cex = 2, pch = 20)
```




```
#Next we can prune down this tree. This acts to act the tree back from an over fit model
prune.boston <- prune.tree(tree.boston, best = tree.min)

plot(prune.boston)
text(prune.boston, pretty = 0)
```




```
#summary(prune.boston)

#calculate the pruned training error. This will have been introduced as the tree
#no longer fits the training data with 100% accuracy
#i.e the tree is now no longer overfit.
yhat_train <- predict(prune.boston, newdata = Boston.train)
MSE <- mean((yhat_train - Boston.train$medv)^2)
sprintf("Train MSE for pruned tree: %s", MSE)
```

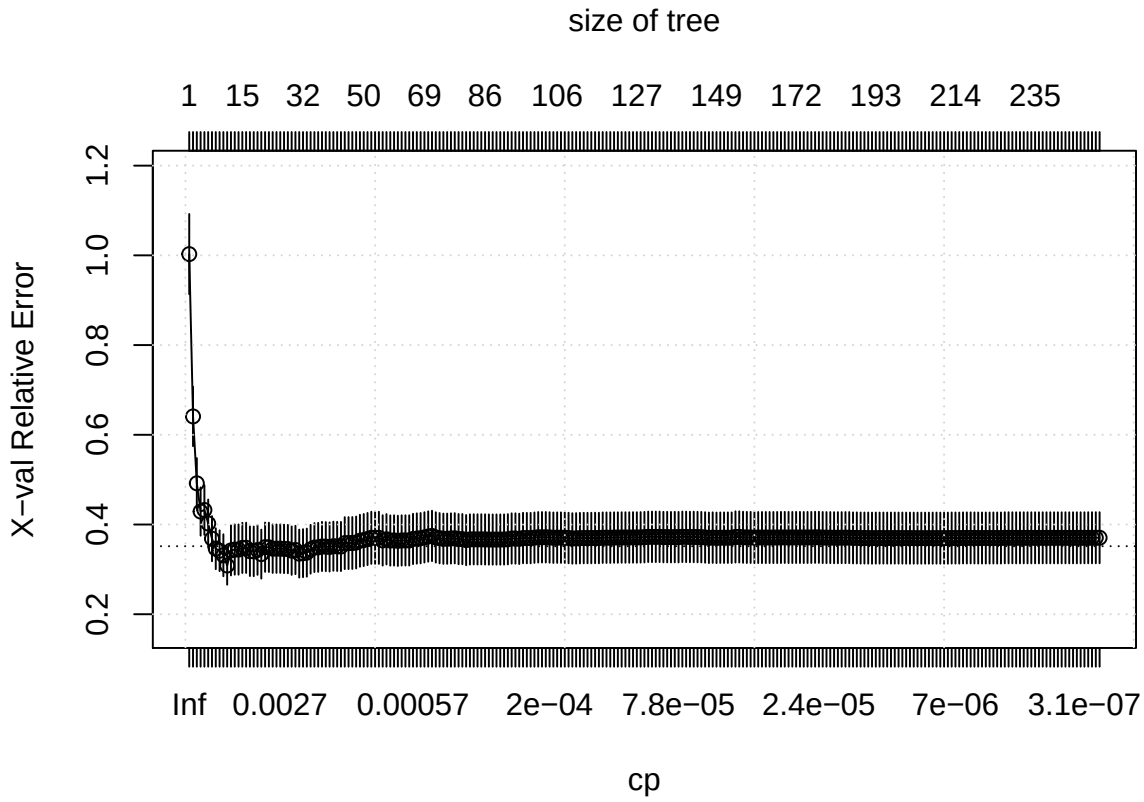
```
## [1] "Train MSE for pruned tree: 9.20114209656504"
```

```
#lastly we can calculate the pruned tree test error
yhat.prune <- predict(prune.boston, newdata = Boston.test)
MSE <- mean((yhat.prune - Boston.test$medv)^2)
sprintf("Test MSE for pruned tree: %s", MSE)
```

```
## [1] "Test MSE for pruned tree: 20.8372742848094"
```

6.1 Implement using the rpart package

```
# Classification Tree with rpart
library(rpart)
library(rpart.plot)

# grow tree
set.seed(26)
fit <- rpart(medv ~.,
             method="poisson",
             data=Boston,
             control = rpart.control(
              minsplit = 2,
              cp = 0,
              mindev = 0,
              minbucket = 1,
              minsize = 2,
              maxdepth = 30,
              xval= 5 #number of cross validations
              ),
              subset = train)
plotcp(fit)
grid()
```



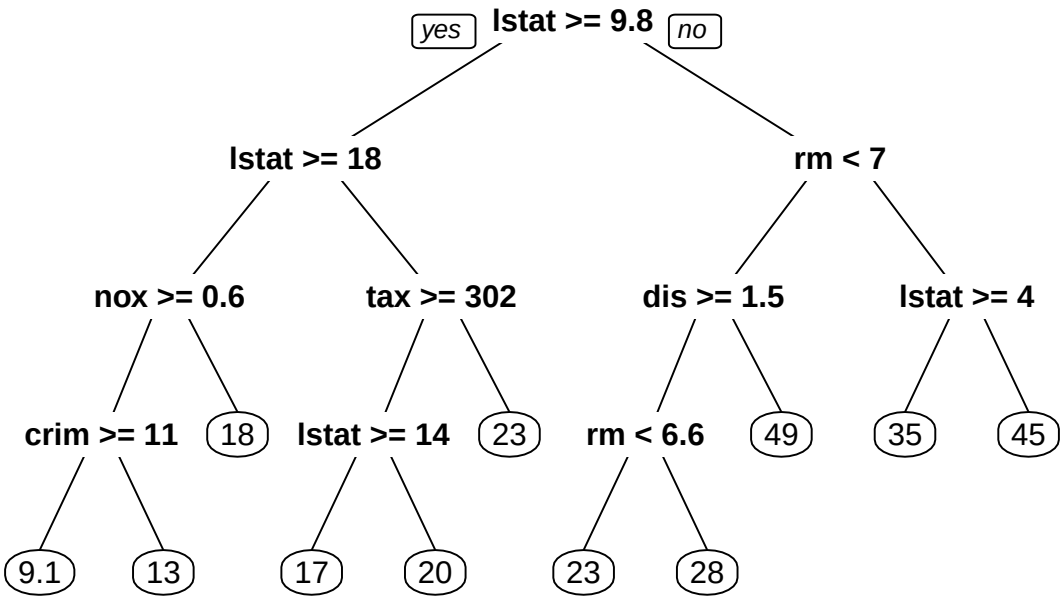

```
#training mse
yhat <- predict(fit, newdata = Boston.train)
MSE <- mean((yhat - Boston.train$medv)^2)
sprintf("Training MSE for full tree: %s", MSE)
```

```
## [1] "Training MSE for full tree: 0.0930158276741652"
```

```
#test mse
yhat <- predict(fit, newdata = Boston.test)
MSE <- mean((yhat - Boston.test$medv)^2)
sprintf("Test MSE for full tree: %s", MSE)
```

```
## [1] "Test MSE for full tree: 22.8276485237778"
```

```
# prune the tree
pfit<- prune(fit, cp=fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])

prp(pfit) # display the new tree
```




```
#training error
yhat <- predict(pfit, newdata = Boston.train)
MSE <- mean((yhat - Boston.train$medv)^2)
sprintf("Training MSE for pruned tree: %s", MSE)
```

```
## [1] "Training MSE for pruned tree: 10.773387975008"
```

```
#test error
yhat <- predict(pfit, newdata = Boston.test)
MSE <- mean((yhat - Boston.test$medv)^2)
sprintf("Test MSE for pruned tree: %s", MSE)
```

```
## [1] "Test MSE for pruned tree: 23.8965403193335"
```

Appendix B: Question 2 Code

This appendix contains all the code to answer question 2 of the assignment. It should be considered along with the report wherein the analisis can be found.

For each model created a confusion is generated to compute all relevent metrics like Accuracy, Sensitivity, Specificity, Kappa and the F1 Statistic. These values are stored within a dataframe called `modelPreformance` and are then used later in question 2.e to evaluate and compare the models generated. As each model is generated the associated confusion matrix is printed at the end of the computation.

```
# Setup work space, install packages and import libs
rm(list = ls())
suppressMessages(library(caret))
suppressMessages(library(ROCR))
suppressMessages(library(blorr))
suppressMessages(library(magrittr))
suppressMessages(library(glmnet))
suppressMessages(library(MASS))
suppressMessages(library(caret))
suppressMessages(library(h2o))
suppressMessages(library(tictoc))
suppressMessages(library(heplots))
suppressMessages(library(knitr))
suppressMessages(library(klaR))
options(warn = -1)

# Extract the testing and training data from the provided csv files. These
# should be stored in the same root directory as this notebook.
Spam <- data.frame()

Spam.test <- read.csv("spam_test.csv")

full.data <- read.csv("spam_data.csv")
# we further split the training set up into a training and validation set so that
# hyperparameters can be chosen and models compared without biasing the results
# into the training set. For this, 90% of the data is used in the training set
# and 20% of the rows are left as the validation set.

set.seed(42)
#~80% of the set
train <- sample(seq_len(nrow(full.data)), size = ceiling(dim(full.data)[1]*0.8))
Spam.train <- full.data[train, ] #2881 observations used for training
Spam.validation <- full.data[-train, ] #720 observations left for validation

# Remove the index column from the data
Spam.train <- Spam.train[, -1]
Spam.test <- Spam.test[, -1]
Spam.validation <- Spam.validation[, -1]

train <- seq(1, nrow(Spam.train))

# Create a dataframe to store the results modelPreformance. This is populated
# Later after each model is fit
```

```
modelValidationPreformance <- data.frame()
modelTestPreformance <- data.frame()
```

First, a generic function needs to be created to store the preformance metrics for models. This function takes in `confusionMatrix` generated by `caret` and stores the relevent information to be used later on.

```
storeModelValidationPreformance <- function(modelName, confusionMatrix){
  #takes the incoming confusion matrix and adds it to the dataframe. note the
  #use of the <<- syntax notating that the function can modify the dataframe
  #modelValidationPreformance which is out of the scope of this function.
  modelValidationPreformance <<- rbind(
    modelValidationPreformance,
    data.frame(
      modelName = modelName,
      Accuracy = confusionMatrix$overall["Accuracy"],
      Kappa = confusionMatrix$overall["Kappa"],
      TruePositiveRate = confusionMatrix$byClass["Sensitivity"],
      TrueNegativeRate = confusionMatrix$byClass["Specificity"],
      FalsePositiveRate = 1 - confusionMatrix$byClass["Specificity"],
      FlaseNegativeRate = 1 - confusionMatrix$byClass["Sensitivity"],
      F1 = confusionMatrix$byClass["F1"]
    )
  )
}

storeModelTestPreformance <- function(modelName, confusionMatrix){
  modelTestPreformance <<- rbind(
    modelTestPreformance,
    data.frame(
      modelName = modelName,
      Accuracy = confusionMatrix$overall["Accuracy"],
      Kappa = confusionMatrix$overall["Kappa"],
      TruePositiveRate = confusionMatrix$byClass["Sensitivity"],
      TrueNegativeRate = confusionMatrix$byClass["Specificity"],
      FalsePositiveRate = 1 - confusionMatrix$byClass["Specificity"],
      FlaseNegativeRate = 1 - confusionMatrix$byClass["Sensitivity"],
      F1 = confusionMatrix$byClass["F1"]
    )
  )
}
```

Question 1.a: Logistic Regression Model

First we use standard logistic regression to fit a model onto the data set. To make the visulization of outputs easier the `confusionMatrix` function is used from the `caret` package. This is effectivly the same as calling `table` on both the predicted and actual results and then calculating the true positive and negative rates. The key terms from the confusion matrix of intrest are:

1. *Accuracy:* Quantifies the overall model quality of fit
2. *Kappa:* similar to Accuracy score, but it takes into account the accuracy that would have happened anyway through random predictions.
3. *Sensitivity:* true positive rate (correctly identified emails)

4. *Specificity:* true negative rate (correctly identified spam)
5. *F1:* = cosniders both the precision and recall to quantify quality such that$(2 \times Precision \times Recall)/(Precision + Recall)$

General Linear Model

First step is to fit a general linear model, with famialy binomial, to indicate the fit is creating a standard logistic regression model.

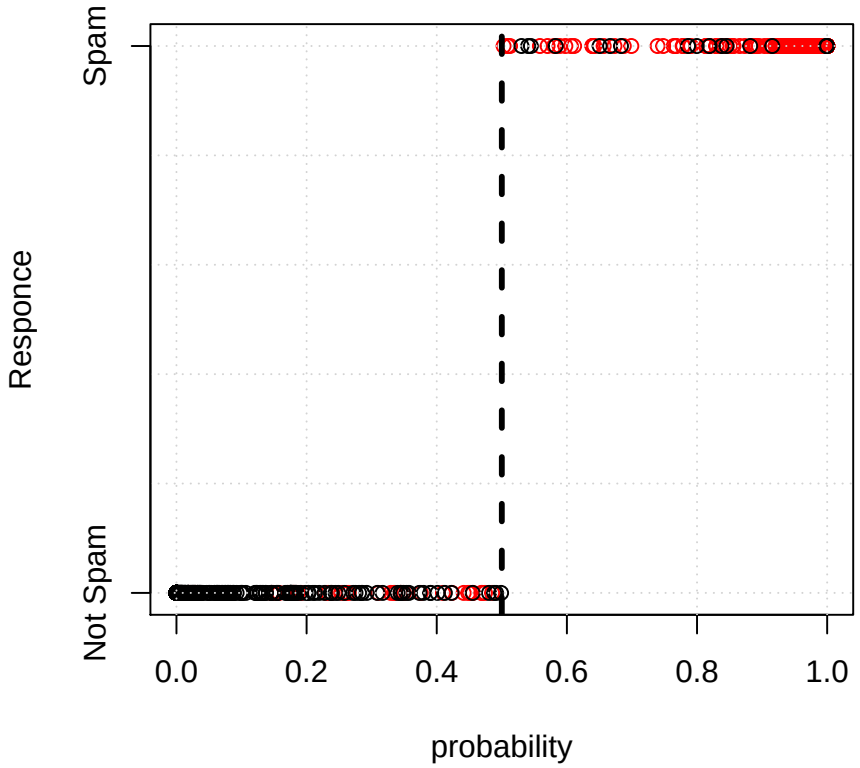
```
glm.fit <- glm(spam ~ .,
               data = Spam.train,
               family = binomial)

# train predictions
probs <- predict(glm.fit, newdata = Spam.train[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
confusionMatrix(factor(glm.pred), Spam.train$spam, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email  1629  114
##      spam     77 1061
##
##                Accuracy : 0.9337
##                  95% CI : (0.924, 0.9425)
##     No Information Rate : 0.5922
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8621
##  Mcnemar's Test P-Value : 0.009191
##
##             Sensitivity : 0.9549
##             Specificity : 0.9030
##          Pos Pred Value : 0.9346
##          Neg Pred Value : 0.9323
##               Precision : 0.9346
##                  Recall : 0.9549
##                      F1 : 0.9446
##              Prevalence : 0.5922
##          Detection Rate : 0.5654
##    Detection Prevalence : 0.6050
##       Balanced Accuracy : 0.9289
##
##        'Positive' Class : email
##
```

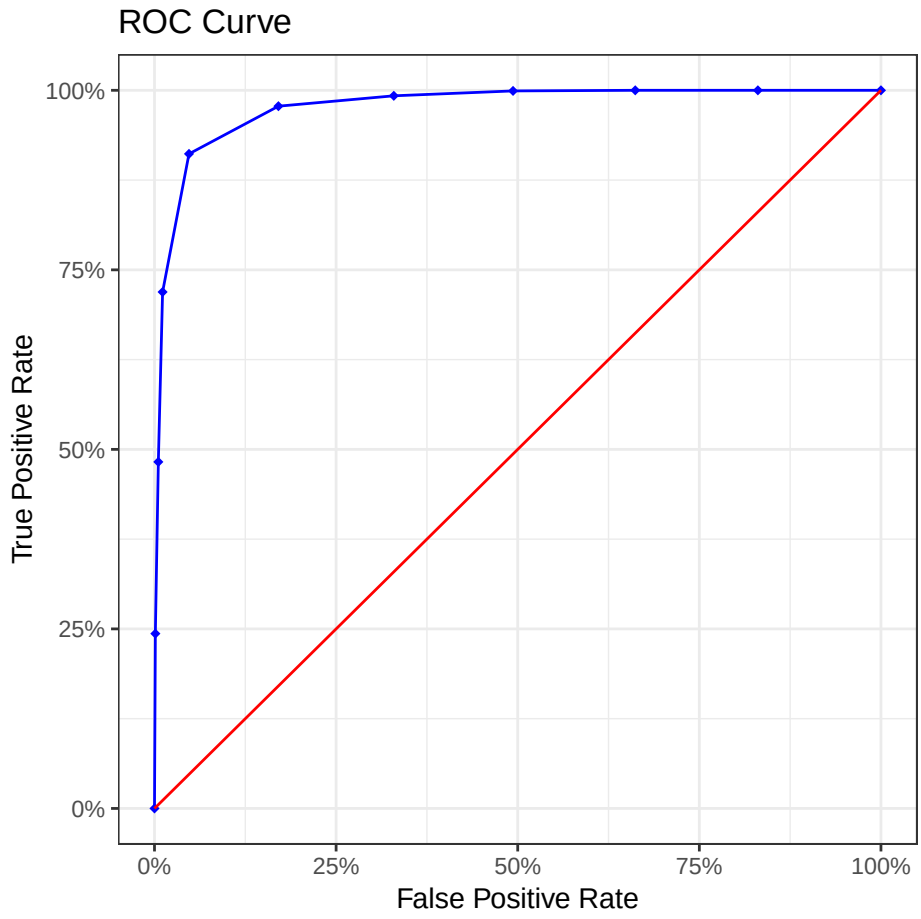
```
# validation predictions
probs <- predict(glm.fit, newdata = Spam.validation[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
(cm <- confusionMatrix(factor(glm.pred), Spam.validation$spam, mode = "everything"))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   418   36
##      spam     17  249
##
##                Accuracy : 0.9264
##                  95% CI : (0.9048, 0.9444)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8443
##  Mcnemar's Test P-Value : 0.01342
##
##             Sensitivity : 0.9609
##             Specificity : 0.8737
##          Pos Pred Value : 0.9207
##          Neg Pred Value : 0.9361
##               Precision : 0.9207
##                  Recall : 0.9609
##                      F1 : 0.9404
##              Prevalence : 0.6042
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.6306
##       Balanced Accuracy : 0.9173
##
##        'Positive' Class : email
##
```

```
storeModelValidationPreformance("GLM", cm)

plot(probs, as.factor(glm.pred),
     col=(Spam.validation$spam=="spam")+9, #colour 10 is red, colour 11 is green
     type="p",
     xlab="probability",
     ylab="Responce",
     yaxt="n")
grid()
abline(v=c(0.5), lty=2, lwd=3)
axis(2, at=1:2, labels=c('Not Spam','Spam'))
```




```
# Roc
glm.fit %>%
  blr_gains_table() %>%
  blr_roc_curve(xaxis_title = "False Positive Rate",
                yaxis = "True Positive Rate") +
  theme_bw()
```




```
# test predictions
probs <- predict(glm.fit, newdata = Spam.test[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
(cm <- confusionMatrix(factor(glm.pred), Spam.test$spam, mode = "everything"))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   617   36
##      spam     30  317
##
##                Accuracy : 0.934
##                  95% CI : (0.9168, 0.9486)
##     No Information Rate : 0.647
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.855
##  Mcnemar's Test P-Value : 0.5383
##
##             Sensitivity : 0.9536
##             Specificity : 0.8980
##          Pos Pred Value : 0.9449
```

```
##          Neg Pred Value : 0.9135
##               Precision : 0.9449
##                  Recall : 0.9536
##                      F1 : 0.9492
##              Prevalence : 0.6470
##          Detection Rate : 0.6170
##    Detection Prevalence : 0.6530
##       Balanced Accuracy : 0.9258
##
##        'Positive' Class : email
##
```

```
storeModelTestPreformance("GLM", cm)
```

Forward, backward and bothways stepwise selection

```
#***backwards***#
suppressMessages(backwards <- stepAIC(glm.fit,
                  direction = "backward",
                  trace=FALSE))

#predict validation results
probs <- predict(backwards, newdata = Spam.validation[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store validation results
cm <- confusionMatrix(factor(glm.pred), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("GLM Backward", cm)

#test results (for question 2.e)
probs <- predict(backwards, newdata = Spam.test[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store test results
cm <- confusionMatrix(factor(glm.pred), Spam.test$spam, mode = "everything")
storeModelTestPreformance("GLM Backward", cm)


#***forwards***#
forwards <- step(glm.fit,
                  direction = "forward",
                  trace=FALSE)

#predict validation results
probs <- predict(forwards, newdata = Spam.validation[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store validation results
cm <- confusionMatrix(factor(glm.pred), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("GLM Forwards", cm)

#test results (for question 2.e)
probs <- predict(forwards, newdata = Spam.test[-58], type = "response")
```

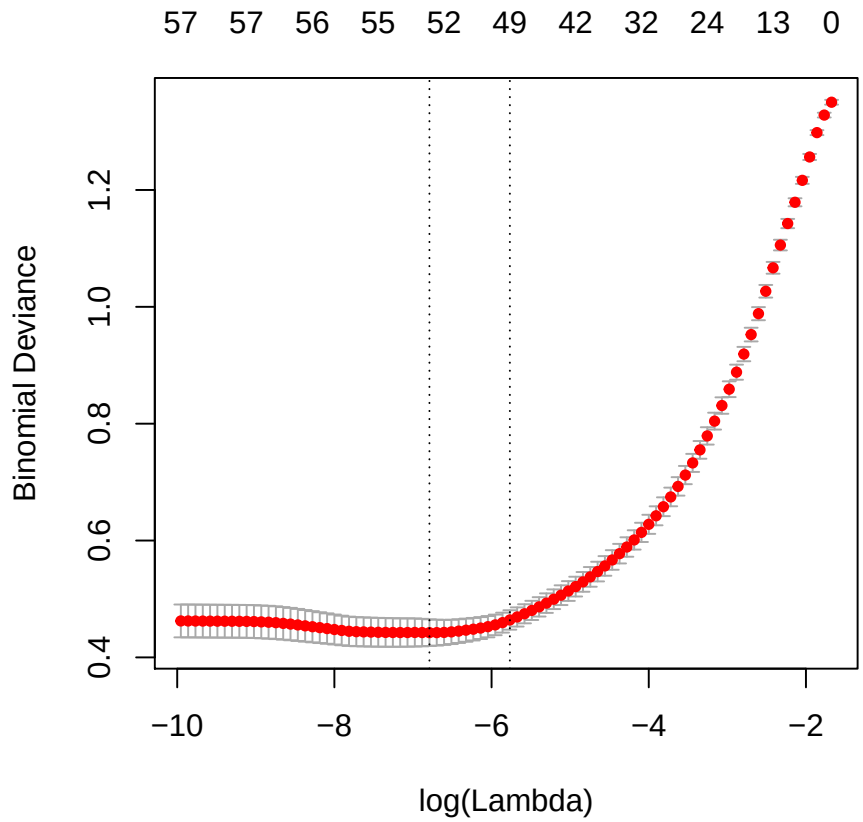
```r
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store test results
cm <- confusionMatrix(factor(glm.pred), Spam.test$spam, mode = "everything")
storeModelTestPreformance("GLM Forwards", cm)

#***bothways****#
bothways <- step(glm.fit,
                 direction = "both",
                 trace=FALSE)
#predict validation results
probs <- predict(bothways, newdata = Spam.validation[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store validation results
cm <- confusionMatrix(factor(glm.pred), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("GLM Bothways", cm)

#test results (for question 2.e)
probs <- predict(bothways, newdata = Spam.test[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
#store test results
cm <- confusionMatrix(factor(glm.pred), Spam.test$spam, mode = "everything")
storeModelTestPreformance("GLM Bothways", cm)
```

Cross Validation with Lasso on General Linear Model

Next, we can try using diffrent variable selection methods, such as using cross validation with lasso to select the best posible model. This is has the effect of removing irrelevent variables from the model and in the process refines the output. The output of the Lasso is also plotted to show the effect of increasing lambda with the output.

```r
#generate lasso model
x <- model.matrix(spam ~ ., data = Spam.train)
cv.lasso <- cv.glmnet(x, Spam.train[, 58], alpha = 1, family = "binomial")

#validation results
newx <- model.matrix(spam ~ ., data = Spam.validation)
cv.probs <- predict(cv.lasso, newx = newx, type = "response", s = cv.lasso$lambda.min)
cv.pred <- rep("email", dim(cv.probs)[1])
cv.pred[cv.probs > 0.5] <- "spam"
cm <- confusionMatrix(factor(cv.pred), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("CV Lasso GLM", cm)

#test results
newx <- model.matrix(spam ~ ., data = Spam.test)
cv.probs <- predict(cv.lasso, newx = newx, type = "response", s = cv.lasso$lambda.min)
cv.pred <- rep("email", dim(cv.probs)[1])
cv.pred[cv.probs > 0.5] <- "spam"
cm <- confusionMatrix(factor(cv.pred), Spam.test$spam, mode = "everything")
storeModelTestPreformance("CV Lasso GLM", cm)
```

```r
plot(cv.lasso)
```




```r
cm
```

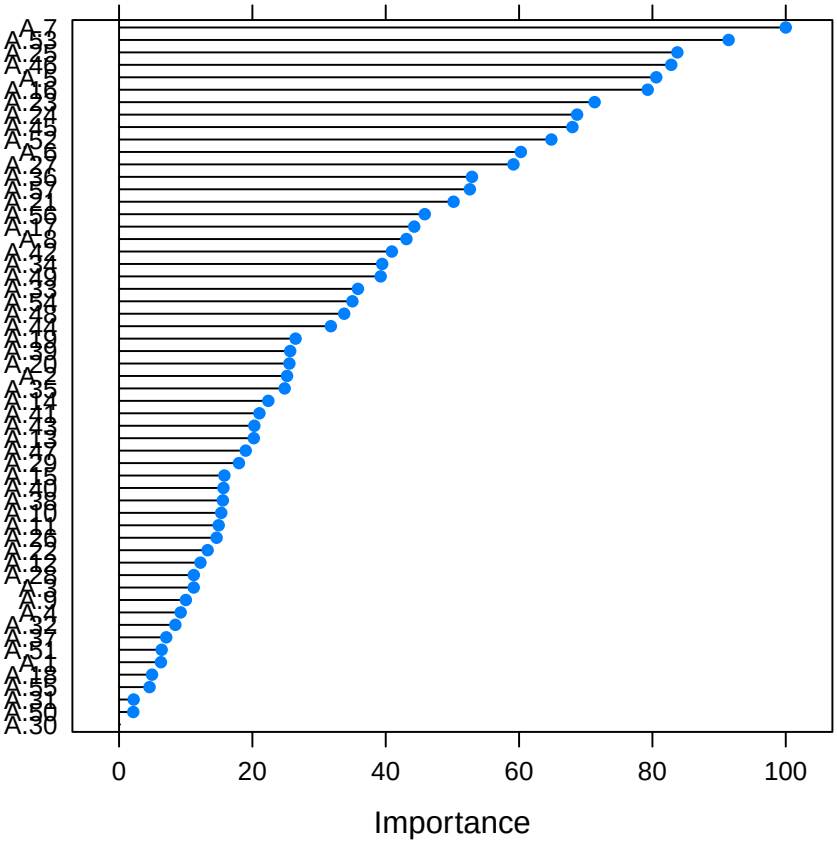
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   617   44
##      spam     30  309
##
##                Accuracy : 0.926
##                  95% CI : (0.908, 0.9415)
##     No Information Rate : 0.647
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8365
##  Mcnemar's Test P-Value : 0.1307
##
##             Sensitivity : 0.9536
##             Specificity : 0.8754
##          Pos Pred Value : 0.9334
##          Neg Pred Value : 0.9115
##               Precision : 0.9334
##                  Recall : 0.9536
##                      F1 : 0.9434
##              Prevalence : 0.6470
##          Detection Rate : 0.6170
```

```
##    Detection Prevalence : 0.6610
##       Balanced Accuracy : 0.9145
##
##        'Positive' Class : email
##
```

Repeated cross validation with linear model

Last thing to try in variable selection is to employ the `caret` package and try doing some kind of cross validated lasso using on the glm method. We can also look at variable importance using this method.

```
controller <- trainControl(
  method = "repeatedcv", # repeated cross validation
  number = 10, # k = 10 folds
  repeats = 5, # CV is done 5 times with 5 diffrent sets of k splits
  verboseIter = F
)

linear <- train(spam ~ .,
  data = Spam.train,
  method = "glm",
  trControl = controller
)

# validation results
probs <- predict(linear$finalModel, newdata = Spam.validation, type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
cm <- confusionMatrix(factor(glm.pred), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Repeated CV GLM",cm)

plot(varImp(linear, scale = T))
```




```
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   418   36
##      spam     17  249
##
##                Accuracy : 0.9264
##                  95% CI : (0.9048, 0.9444)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8443
##  Mcnemar's Test P-Value : 0.01342
##
##             Sensitivity : 0.9609
##             Specificity : 0.8737
##          Pos Pred Value : 0.9207
##          Neg Pred Value : 0.9361
##               Precision : 0.9207
##                  Recall : 0.9609
##                      F1 : 0.9404
##              Prevalence : 0.6042
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.6306
```

```
##       Balanced Accuracy : 0.9173
##
##        'Positive' Class : email
##
```

```
# Testing results
probs <- predict(linear$finalModel, newdata = Spam.test, type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.5] <- "spam"
cm <- confusionMatrix(factor(glm.pred), Spam.test$spam, mode = "everything")
storeModelTestPreformance("Repeated CV GLM",cm)
```

Comparing diffrent variable selection techniques.

```
rownames(modelValidationPreformance) <- c() # clean up the row data from the model preformance
modelValidationPreformance[order(modelValidationPreformance$Kappa, decreasing = TRUE),]
```

```
##         modelName  Accuracy     Kappa TruePositiveRate TrueNegativeRate
## 1             GLM 0.9263889 0.8443084        0.9609195        0.8736842
## 3    GLM Forwards 0.9263889 0.8443084        0.9609195        0.8736842
## 6 Repeated CV GLM 0.9263889 0.8443084        0.9609195        0.8736842
## 2    GLM Backward 0.9236111 0.8384333        0.9586207        0.8701754
## 4    GLM Bothways 0.9236111 0.8384333        0.9586207        0.8701754
## 5    CV Lasso GLM 0.9208333 0.8321472        0.9609195        0.8596491
##   FalsePositiveRate FlaseNegativeRate        F1
## 1         0.1263158        0.03908046 0.9403825
## 3         0.1263158        0.03908046 0.9403825
## 6         0.1263158        0.03908046 0.9403825
## 2         0.1298246        0.04137931 0.9381327
## 4         0.1298246        0.04137931 0.9381327
## 5         0.1403509        0.03908046 0.9361702
```

```
#Looking at number of variables included in the models:

coefs <- coef(cv.lasso, cv.lasso$lambda.min)
print(colSums(coefs!=0))
```

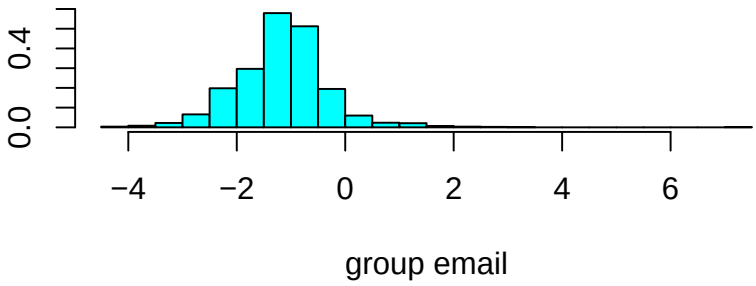
```
##  1
## 54
```

Question 2.b: Discriminant Analysis

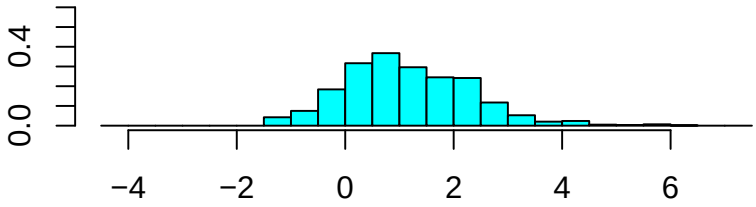
Next discriminant analysis is applied to try and get a more refined model. Both linear discriminant analysis and quadratic discriminant analysis are applied and the same confusion matrix is generated.

Linear discriminat analysis

```
lda.fit <- lda(spam ~ ., data = Spam.train)
plot(lda.fit)
```



group email



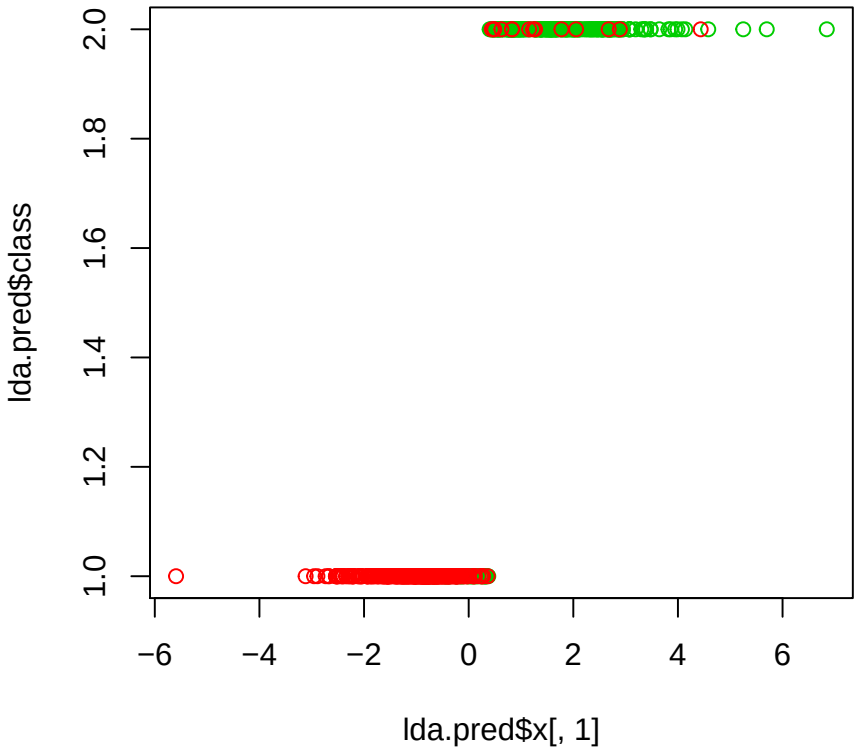
group spam

```r
#prediction results
lda.pred <- predict(lda.fit, newdata = Spam.validation, type = "response")
cm <- confusionMatrix(factor(lda.pred$class), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("LDA",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   421   63
##      spam     14  222
##
##                Accuracy : 0.8931
##                  95% CI : (0.8682, 0.9147)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7696
##  Mcnemar's Test P-Value : 4.498e-08
##
##             Sensitivity : 0.9678
##             Specificity : 0.7789
##          Pos Pred Value : 0.8698
##          Neg Pred Value : 0.9407
##               Precision : 0.8698
##                  Recall : 0.9678
##                      F1 : 0.9162
##              Prevalence : 0.6042
##          Detection Rate : 0.5847
```

```
##    Detection Prevalence : 0.6722
##       Balanced Accuracy : 0.8734
##
##        'Positive' Class : email
##
```

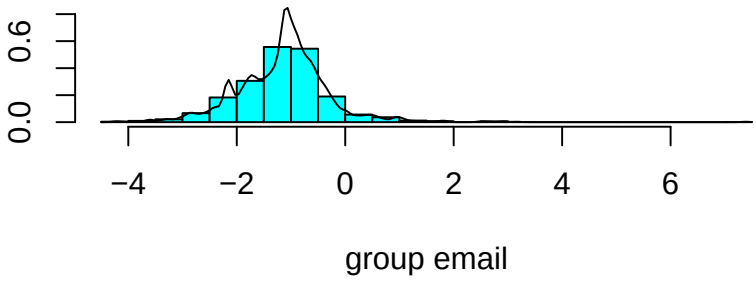
```r
plot(lda.pred$x[,1], lda.pred$class, col=(Spam.validation$spam=="spam")+10, type="p")
```



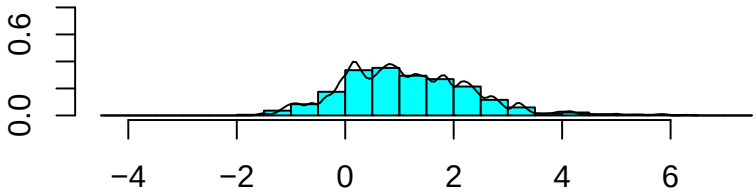

```r
#test results
lda.pred <- predict(lda.fit, newdata = Spam.test, type = "response")
cm <- confusionMatrix(factor(lda.pred$class), Spam.test$spam, mode = "everything")
storeModelTestPreformance("LDA",cm)
```

LDA stepwise selection. for this the greedy.wilks function is used to preform variable selection.

```r
#select the relevent variables
lda.wilks <- greedy.wilks(spam ~ .,data=Spam.train, niveau = 0.05)

#build the model on the new variables
lda.fit <- lda(lda.wilks$formula, data=Spam.train, method="moment")

# Step2) Discriminant Score Estimates for Each Observation ####
pred <- predict(lda.fit)

# Plot the discriminant scores
plot(lda.fit, dimen = 1, type="both")
```

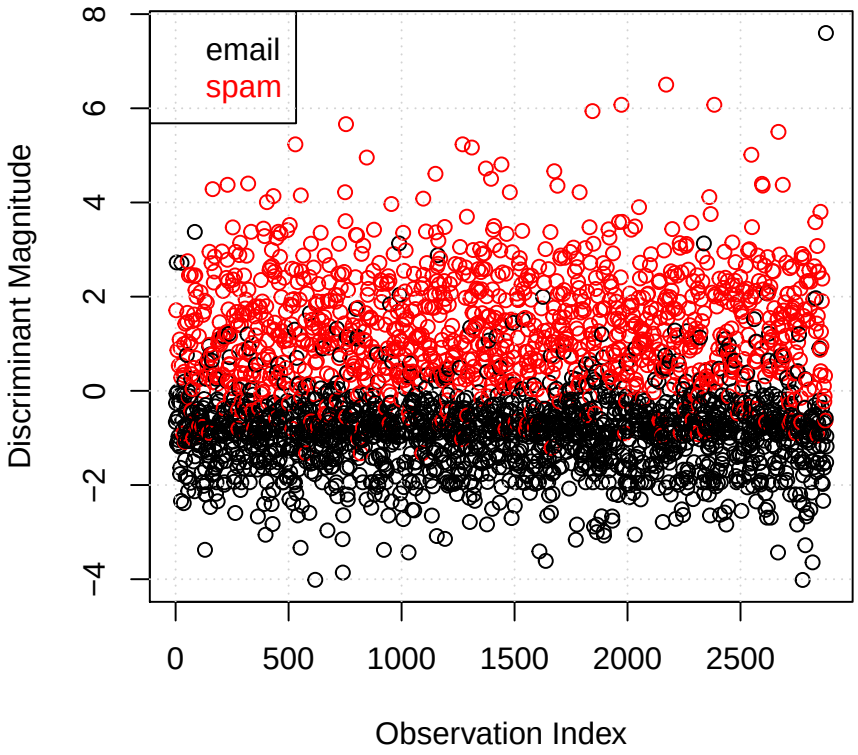


group email



group spam

```
plot(pred$x[,1],
     xlab="Observation Index",
     ylab="Discriminant Magnitude",
     col=factor(Spam.train$spam))
legend("topleft",
       legend=levels(factor(Spam.train$spam)),
       text.col=seq_along(levels(factor(Spam.train$spam))))
grid()
```




```
#prediction results
lda.pred <- predict(lda.fit, newdata = Spam.validation, type = "response")
cm <- confusionMatrix(factor(lda.pred$class), Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("LDA, wilks",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   421   63
##      spam     14  222
##
##                Accuracy : 0.8931
##                  95% CI : (0.8682, 0.9147)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7696
##  Mcnemar's Test P-Value : 4.498e-08
##
##             Sensitivity : 0.9678
##             Specificity : 0.7789
##          Pos Pred Value : 0.8698
##          Neg Pred Value : 0.9407
##               Precision : 0.8698
##                  Recall : 0.9678
##                      F1 : 0.9162
##              Prevalence : 0.6042
##          Detection Rate : 0.5847
```

```
##    Detection Prevalence : 0.6722
##       Balanced Accuracy : 0.8734
##
##        'Positive' Class : email
##
```

```
#test results
lda.pred <- predict(lda.fit, newdata = Spam.test, type = "response")
cm <- confusionMatrix(factor(lda.pred$class), Spam.test$spam, mode = "everything")
storeModelTestPreformance("LDA, wilks",cm)
```

Quadratic discriminant analysis

```
qda.fit <- qda(spam ~ ., data = Spam.train)

#validation preformance
probs <- predict(qda.fit, newdata = Spam.validation, type = "response")
cm <- confusionMatrix(probs$class, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("QDA",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   346   14
##      spam     89  271
##
##                Accuracy : 0.8569
##                  95% CI : (0.8292, 0.8817)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7139
##  Mcnemar's Test P-Value : 3.067e-13
##
##             Sensitivity : 0.7954
##             Specificity : 0.9509
##          Pos Pred Value : 0.9611
##          Neg Pred Value : 0.7528
##               Precision : 0.9611
##                  Recall : 0.7954
##                      F1 : 0.8704
##              Prevalence : 0.6042
##          Detection Rate : 0.4806
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8731
##
##        'Positive' Class : email
##
```

```
#validation preformance
probs <- predict(qda.fit, newdata = Spam.test, type = "response")
cm <- confusionMatrix(probs$class, Spam.test$spam, mode = "everything")
```

```
storeModelTestPreformance("QDA",cm)
```

Assumption Testing

LDA assumes normal distributed data, features that are statistically independent, and identical covariance matrices for every class. These assumptions are now tested. When these assumptions hold, then LDA approximates the Bayes classifier very closely and the discriminant function produces a linear decision boundary. QDA assumes that observation of each class are drawn from a normal distribution (same as LDA). It also assumes that each class has its own covariance matrix (different from LDA). Assumptions:

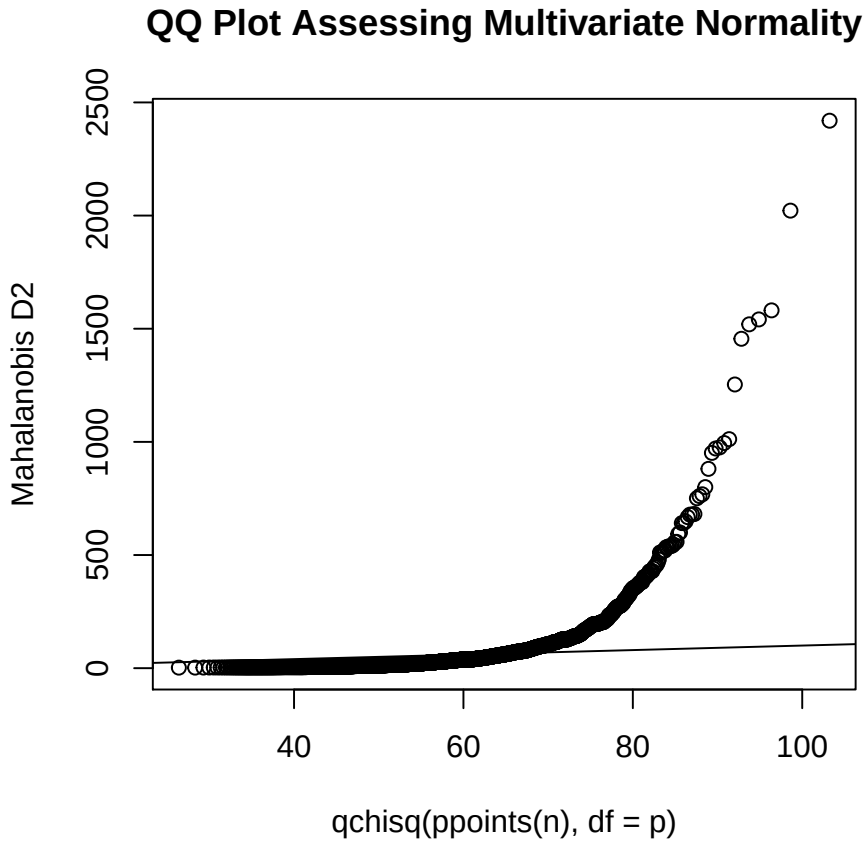
```
### LDA Assumptions.
#1: normality testing. Doesent really make sence with such a large data set but
#will do it in any case. using the shapiro.test and Multivariate Normality will
#be tested using MVN.
test <- shapiro.test(Spam.train[,1])
results <- rep(0,dim(Spam.train)[2] - 1)

for (i in 1:length(results)){
  results[i] <- shapiro.test(Spam.train[,i])
}

library(MVN)
```

```
## sROC 0.1-2 loaded
```

```
result <- mvn(data = Spam.validation[-58])
result$multivariateNormality
```

```
##              Test        Statistic p value Result
## 1 Mardia Skewness 760217.352342856       0     NO
## 2 Mardia Kurtosis 1036.09498051966       0     NO
## 3             MVN             <NA>    <NA>     NO
```

```
# Graphical Assessment of Multivariate Normality
x <- as.matrix(Spam.train[-58])
center <- colMeans(x)
n <- nrow(x);
p <- ncol(x);
cov <- cov(x);
d <- mahalanobis(x,center,cov) # distances
qqplot(qchisq(ppoints(n),df=p),d,
       main="QQ Plot Assessing Multivariate Normality",
       ylab="Mahalanobis D2")
abline(a=0,b=1)
```




```
#2: statistically independence
(chiSqaured <- chisq.test(x = Spam.train[-58],
                          y = Spam.train[58]))
```

```
##
##  Pearson's Chi-squared test
##
## data:  Spam.train[-58]
## X-squared = 2960100, df = 161280, p-value < 2.2e-16
```

```
#3: identical covariance matrices for every class
(res <- boxM(Spam.train[,-58], Spam.train$spam))
```

```
##
##  Box's M-test for Homogeneity of Covariance Matrices
##
## data:  Spam.train[, -58]
## Chi-Sq (approx.) = 152430, df = 1653, p-value < 2.2e-16
```

Question 2.c

Random forrest train a bunch of tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data. Values are then taken as an average from the generated set of trees. First a generic random forrest is created and then the `h2o.grid` function will be used to try various combinations of the different possible model parameters.

```r
suppressMessages(h2o.init(max_mem_size = "10g"))


invisible(
  rf.spam <- h2o.randomForest(
    training_frame = as.h2o(Spam.train),
    y = 58,
    seed = 1
  )
)

#validation results
rf.probs <- h2o.predict(
  object = rf.spam,
  newdata = as.h2o(Spam.validation)
)
```

```r
# we need to do a bit of manipulation to get the h2o dataframe out of it's given
#format to put it into a data frame that the confusionMatrix function from caret
#can understand alternativly, one can use# the confusion matrix function from
#h2o but this formats data diffrently so for the sake of consistancy I will
#format it this way. The h2o function would be:
#h2o.confusionMatrix(rf.spam, as.h2o(Spam.test))
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Random Forest",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   419   24
##      spam     16  261
##
##                Accuracy : 0.9444
##                  95% CI : (0.9251, 0.96)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8833
##  Mcnemar's Test P-Value : 0.2684
##
##             Sensitivity : 0.9632
##             Specificity : 0.9158
##          Pos Pred Value : 0.9458
##          Neg Pred Value : 0.9422
##               Precision : 0.9458
##                  Recall : 0.9632
##                      F1 : 0.9544
##              Prevalence : 0.6042
##          Detection Rate : 0.5819
##    Detection Prevalence : 0.6153
##       Balanced Accuracy : 0.9395
```

```
##
##        'Positive' Class : email
##
```

```
#testing results
rf.probs <- h2o.predict(
  object = rf.spam,
  newdata = as.h2o(Spam.test)
)
```

```
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.test$spam, mode = "everything")
storeModelTestPreformance("Random Forest",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   624   39
##      spam     23  314
##
##                Accuracy : 0.938
##                  95% CI : (0.9212, 0.9521)
##     No Information Rate : 0.647
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8629
##  Mcnemar's Test P-Value : 0.05678
##
##             Sensitivity : 0.9645
##             Specificity : 0.8895
##          Pos Pred Value : 0.9412
##          Neg Pred Value : 0.9318
##               Precision : 0.9412
##                  Recall : 0.9645
##                      F1 : 0.9527
##              Prevalence : 0.6470
##          Detection Rate : 0.6240
##    Detection Prevalence : 0.6630
##       Balanced Accuracy : 0.9270
##
##        'Positive' Class : email
##
```

Next, the h2o.grid function is impelemented. This is done to build up a number of diffrent models over a range of diffrent hyper parameters

```
#this will generate 125 diffrent models: for each  mtries, each ntrees and
#each max_dept.
#Number of variables randomly sampled as candidates at each split.
mtries <- floor(seq(floor(sqrt(length(Spam.train))/2),
              floor(sqrt(length(Spam.train))*3),
              length.out = 5))
ntrees = floor(seq(50, 300,length = 5)) #Number of trees
max_depth = seq(10, 50,length = 5) #Maximum tree depth
```

```r
hyper_params <- list(mtries = mtries, ntrees = ntrees, max_depth = max_depth)

rf.grid <- h2o.grid(algorithm = "randomForest",
                    hyper_params = hyper_params,
                    x = 1:57, y = 58,
                    training_frame = as.h2o(Spam.train),
                    #validation_frame = datTest_h2o,
                    nfolds = 0,
                    seed = 1)


rf.best <- h2o.getModel(rf.grid@model_ids[[1]])

#Validation results
suppressMessages(
  rf.probs <- h2o.predict(
    object = rf.best,
    newdata = as.h2o(Spam.validation)
  )
)
```

```r
#calculate the results and store
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Random Forest,Grid",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   417   19
##      spam     18  266
##
##                Accuracy : 0.9486
##                  95% CI : (0.9299, 0.9636)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8925
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9586
##             Specificity : 0.9333
##          Pos Pred Value : 0.9564
##          Neg Pred Value : 0.9366
##               Precision : 0.9564
##                  Recall : 0.9586
##                      F1 : 0.9575
##              Prevalence : 0.6042
##          Detection Rate : 0.5792
##    Detection Prevalence : 0.6056
##       Balanced Accuracy : 0.9460
##
```

```
##         'Positive' Class : email
##
```

```
#Testing results
suppressMessages(
  rf.probs <- h2o.predict(
    object = rf.best,
    newdata = as.h2o(Spam.test)
  )
)
```

```
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.test$spam, mode = "everything")
storeModelTestPreformance("Random Forest,Grid",cm)
```

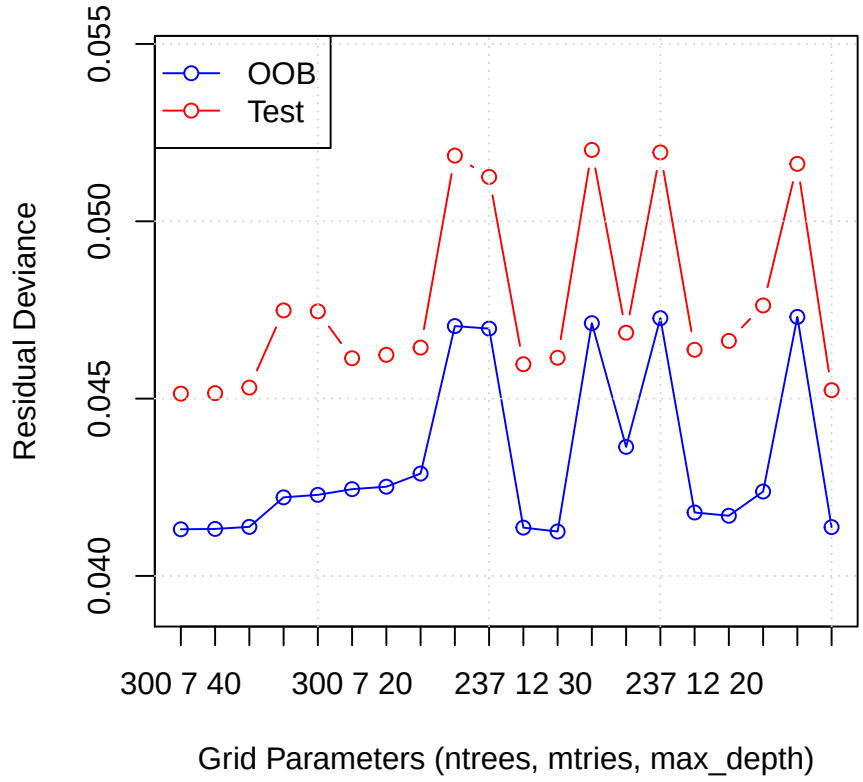
generate a plot to show the OOB error

```
models=list()
n <- 20 #the number of models to plot

for(i in 1:n){
  models[i] = h2o.getModel(rf.grid@model_ids[[i]])
}

# capture the requried x lables
xlabel = NULL
for(i in 1:n){
xlabel[i] = paste(models[[i]]@allparameters$ntrees,
                  models[[i]]@allparameters$mtries,
                  models[[i]]@allparameters$max_depth)
}

#capture the mse to quantify the OOB error
res_devOOB = NULL
for(i in 1:n){
res_devOOB[i] = h2o.mse(models[[i]])
}

#calculate the test set mse
res_devTest = NULL
for (i in 1:n) {
  res_devTest[i] <- h2o.mse(h2o.performance(h2o.getModel(rf.grid@model_ids[[i]]),
                                            newdata = as.h2o(Spam.test)))
}
```

Plot of training errors for different models

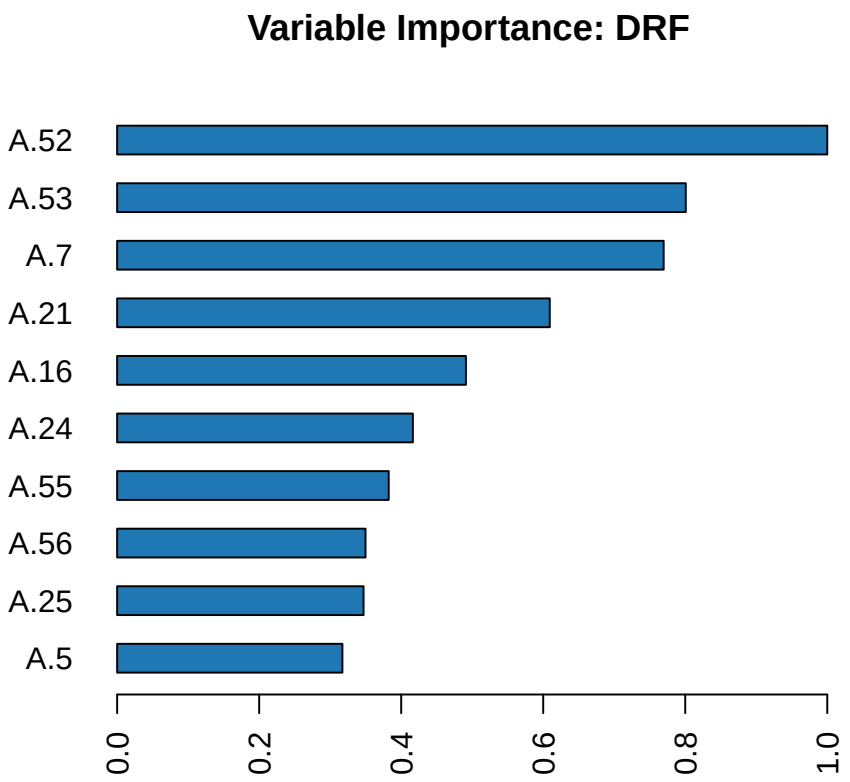
```
#trying to see if the out of bag error and the test errors give similar results
plot(res_devTest[1:n],
     xaxt = "n",
     ylab='Residual Deviance',
     xlab='Grid Parameters (ntrees, mtries, max_depth)',
     ylim=c(min(res_devOOB,res_devTest)*0.95,
            max(res_devOOB,res_devTest)*1.05),
```

```
     type = "b",
     col ="red")
axis(1,
     at=1:n,
     labels=xlabel[1:n])
lines(res_devOOB[1:n],
      type = "o",
      col ="blue")
legend("topleft",
       c("OOB","Test"),
       pch = 21,
       pt.bg = "white",
       lty = 1,
       col = c("blue", "red"))
grid(col = "lightgray", lty = "dotted")
```



Variable importance plot

```
h2o.varimp_plot(rf.best)
```



Now that we know what the ideal values are for the random forest we can try and refine the parameters even further by tuning the range of the hyper parameters to prevent saturation.

```
#this will generate 125 diffrent models: for each  mtries, each ntrees and
#each max_dept.
#Number of variables randomly sampled as candidates at each split.
mtries <- floor(seq(floor(sqrt(length(Spam.train))/2),
              floor(sqrt(length(Spam.train))*2),
              length.out = 5))
ntrees = floor(seq(250, 500,length = 5)) #Number of trees
max_depth = floor(seq(20, 50,length = 5)) #Maximum tree depth

hyper_params <- list(mtries = mtries, ntrees = ntrees, max_depth = max_depth)

rf.grid2 <- h2o.grid(algorithm = "randomForest",
                     hyper_params = hyper_params,
                     x = 1:57, y = 58,
                     training_frame = as.h2o(Spam.train),
                     #validation_frame = datTest_h2o,
                     nfolds = 0,
                     seed = 1)

rf.best2 <- h2o.getModel(rf.grid2@model_ids[[1]])

#Validation results
rf.probs <- h2o.predict(
    object = rf.best2,
    newdata = as.h2o(Spam.validation)
  )
```

```
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Random Forest,Grid2",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   419   21
##      spam     16  264
##
##                Accuracy : 0.9486
##                  95% CI : (0.9299, 0.9636)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8922
##  Mcnemar's Test P-Value : 0.5108
##
##             Sensitivity : 0.9632
##             Specificity : 0.9263
##          Pos Pred Value : 0.9523
##          Neg Pred Value : 0.9429
##               Precision : 0.9523
##                  Recall : 0.9632
##                      F1 : 0.9577
##              Prevalence : 0.6042
##          Detection Rate : 0.5819
##    Detection Prevalence : 0.6111
##       Balanced Accuracy : 0.9448
##
##        'Positive' Class : email
##
```

```
#Testing results
rf.probs <- h2o.predict(
    object = rf.best2,
    newdata = as.h2o(Spam.test)
)
```

```
rf.probs <- as.data.frame(rf.probs)
cm <- confusionMatrix(rf.probs$predict, Spam.test$spam, mode = "everything")
storeModelTestPreformance("Random Forest,Grid2",cm)
```

question 2.d: boosted trees

Next, a classification tree is grown for boosted trees. Boosted trees build trees one at a time, where each new tree helps to correct errors made by previously trained tree. It is a more itterative process. As with random forests first a standard GBM is grown and then an itterative tree is created using `h2o.grid`.

```
gbm.spam <- h2o.gbm(training_frame = as.h2o(Spam.train),
                    x = 1:57,
```

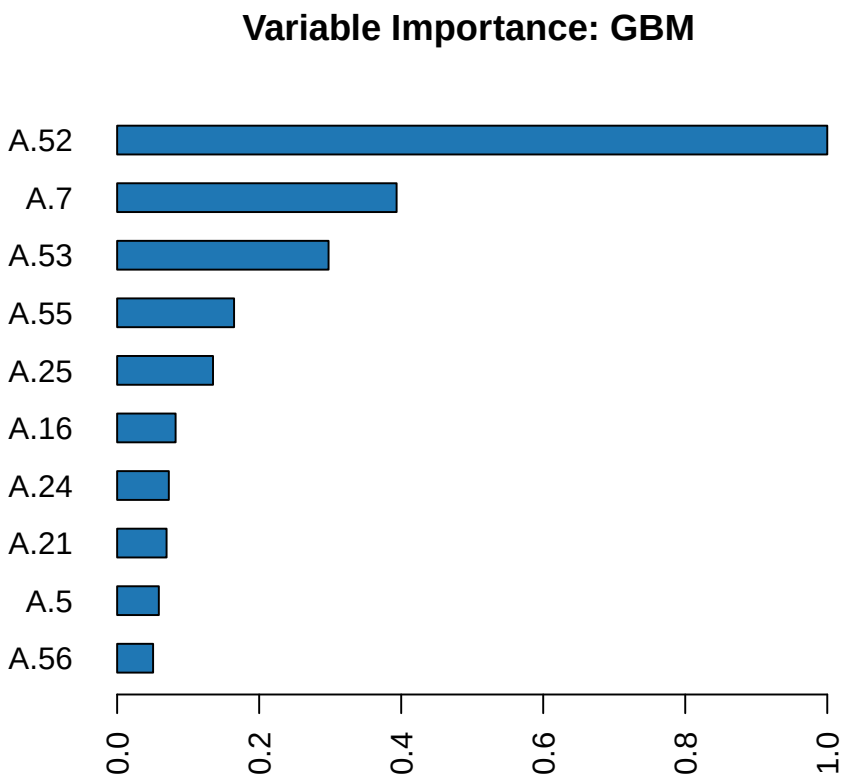
```
                    y = 58,
                    seed = 1)
#validation results
gbm.probs <- h2o.predict(object = gbm.spam, newdata = as.h2o(Spam.validation))
gbm.probs <- as.data.frame(gbm.probs)
cm <- confusionMatrix(gbm.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Boosted Tree",cm)
cm

#testing results
gbm.probs <- h2o.predict(object = gbm.spam, newdata = as.h2o(Spam.test))

gbm.probs <- as.data.frame(gbm.probs)
cm <- confusionMatrix(gbm.probs$predict, Spam.test$spam, mode = "everything")
storeModelTestPreformance("Boosted Tree",cm)
```

```
#we not introduce learning_rate and learn_rate instead of mtries. still generating
# 125 diffrent values over the full grid
learn_rate = seq(0.005, 0.5, length = 5)
ntrees = seq(50, 250,length = 5) #Number of trees
max_depth = seq(10, 50,length = 5) #Maximum tree depth

hyper_params <- list(ntrees = ntrees, max_depth = max_depth, learn_rate = learn_rate)

gbm.grid <- h2o.grid(algorithm = "gbm",
                     hyper_params = hyper_params,
                     x = 1:57,
                     y = 58,
                     training_frame = as.h2o(Spam.train),
                     nfolds = 0,
                     seed = 1)
gbm.best <- h2o.getModel(gbm.grid@model_ids[[1]])

#Validation results
gbm.probs <- h2o.predict(
  object = gbm.best,
  newdata = as.h2o(Spam.validation)
)
gbm.probs <- as.data.frame(gbm.probs)
```

```
cm <- confusionMatrix(gbm.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Boosted Tree,Grid",cm)
```

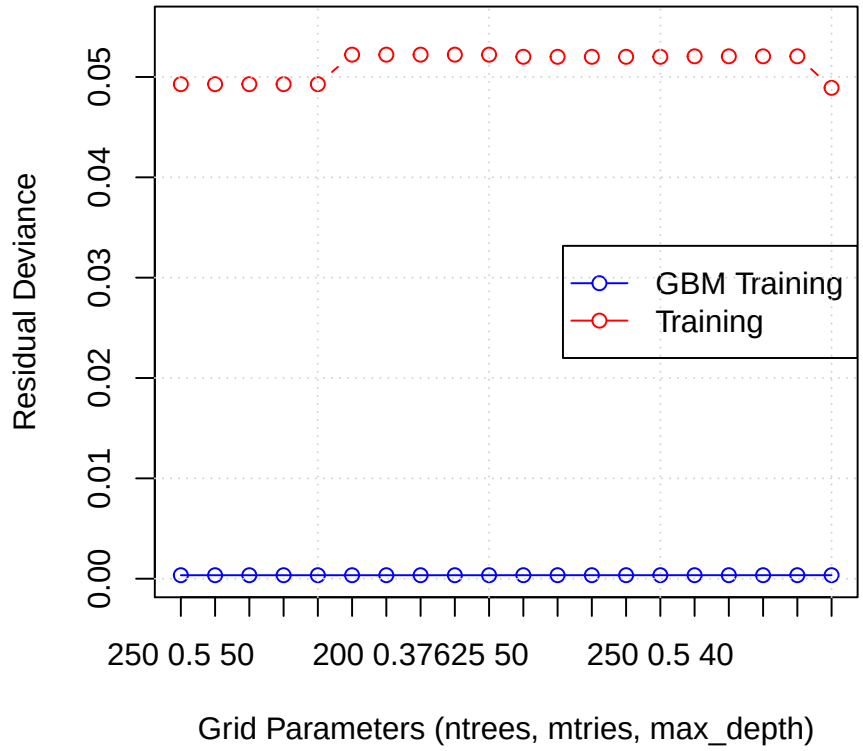
```
#testing results
gbm.probs <- h2o.predict(
  object = gbm.best,
  newdata = as.h2o(Spam.test)
)
```

```
gbm.probs <- as.data.frame(gbm.probs)

cm <- confusionMatrix(gbm.probs$predict, Spam.test$spam, mode = "everything")
storeModelTestPreformance("Boosted Tree,Grid",cm)

h2o.varimp_plot(gbm.best)
```




```
models=list()
n <- 20 #the number of models to plot

for(i in 1:n){
  models[i] = h2o.getModel(gbm.grid@model_ids[[i]])
}

# capture the requried x lables
xlabel = NULL
for(i in 1:n){
xlabel[i] = paste(models[[i]]@allparameters$ntrees,
                  models[[i]]@allparameters$learn_rate,
                  models[[i]]@allparameters$max_depth)
}

#capture the mse to quantify the OOB error
res_devOOB = NULL
for(i in 1:n){
res_devOOB[i] = h2o.mse(models[[i]])
}

#calculate the test set mse
res_devTest = NULL
for (i in 1:n) {
  res_devTest[i] <- h2o.mse(h2o.performance(h2o.getModel(gbm.grid@model_ids[[i]]),
                                            newdata = as.h2o(Spam.test)))
}
```

```r
# Plot of training errors for different models:
#trying to see if the out of bag error and the test errors give similar results
plot(res_devTest[1:n],
     xaxt = "n",
     xlab='Grid Parameters (ntrees, mtries, max_depth)',
     ylab='Residual Deviance',
     ylim=c(min(res_devOOB,res_devTest)*0.95,
            max(res_devOOB,res_devTest)*1.05),
     type = "b",
     col ="red")
axis(1,
     at=1:n,
     labels=xlabel[1:n])
lines(res_devOOB[1:n],
      type = "o",
      col ="blue")
legend("right",
       c("GBM Training","Training"),
       pch = 21,
       pt.bg = "white",
       lty = 1,
       col = c("blue", "red"))
grid(col = "lightgray", lty = "dotted")
```



Question 2.e

This question begings with selecting the best method based off accuracy overall from all testes preformed. We can then modify the ROC position if need be to inflence the Sensitivity spesificity ratio.

```
rownames(modelTestPreformance) <- c() # clean up the row data from the model preformance
kable(modelTestPreformance[order(modelTestPreformance$Accuracy, decreasing = TRUE), ])
```

	modelName	Accuracy	Kappa	TruePositiveRate	TrueNegativeRate	FalsePositiveRate	FlaseN
12	Random Forest,Grid2	0.942	0.8720364	0.9644513	0.9008499	0.0991501	
14	Boosted Tree,Grid	0.942	0.8713689	0.9706337	0.8895184	0.1104816	
11	Random Forest,Grid	0.940	0.8677954	0.9613601	0.9008499	0.0991501	
10	Random Forest	0.938	0.8628555	0.9644513	0.8895184	0.1104816	
13	Boosted Tree	0.938	0.8630337	0.9629057	0.8923513	0.1076487	
1	GLM	0.934	0.8549508	0.9536321	0.8980170	0.1019830	
3	GLM Forwards	0.934	0.8549508	0.9536321	0.8980170	0.1019830	
6	Repeated CV GLM	0.934	0.8549508	0.9536321	0.8980170	0.1019830	
2	GLM Backward	0.926	0.8375790	0.9459042	0.8895184	0.1104816	
4	GLM Bothways	0.926	0.8375790	0.9459042	0.8895184	0.1104816	
5	CV Lasso GLM	0.926	0.8365241	0.9536321	0.8753541	0.1246459	
7	LDA	0.890	0.7521663	0.9489954	0.7818697	0.2181303	
8	LDA, wilks	0.886	0.7431542	0.9459042	0.7762040	0.2237960	
9	QDA	0.821	0.6415785	0.7496136	0.9518414	0.0481586	

print the validation error results

```
rownames(modelValidationPreformance) <- c() # clean up the row data from the model preformance
kable(modelValidationPreformance[order(modelValidationPreformance$Accuracy, decreasing = TRUE), ])
```

	modelName	Accuracy	Kappa	TruePositiveRate	TrueNegativeRate	FalsePositiveRate	FlaseN
14	Boosted Tree,Grid	0.9500000	0.8950820	0.9655172	0.9263158	0.0736842	
11	Random Forest,Grid	0.9486111	0.8924939	0.9586207	0.9333333	0.0666667	
12	Random Forest,Grid2	0.9486111	0.8922330	0.9632184	0.9263158	0.0736842	
10	Random Forest	0.9444444	0.8832827	0.9632184	0.9157895	0.0842105	
13	Boosted Tree	0.9444444	0.8835658	0.9586207	0.9228070	0.0771930	
1	GLM	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	
3	GLM Forwards	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	
6	Repeated CV GLM	0.9263889	0.8443084	0.9609195	0.8736842	0.1263158	
2	GLM Backward	0.9236111	0.8384333	0.9586207	0.8701754	0.1298246	
4	GLM Bothways	0.9236111	0.8384333	0.9586207	0.8701754	0.1298246	
5	CV Lasso GLM	0.9208333	0.8321472	0.9609195	0.8596491	0.1403509	
7	LDA	0.8930556	0.7695761	0.9678161	0.7789474	0.2210526	
8	LDA, wilks	0.8930556	0.7695761	0.9678161	0.7789474	0.2210526	
9	QDA	0.8569444	0.7138889	0.7954023	0.9508772	0.0491228	

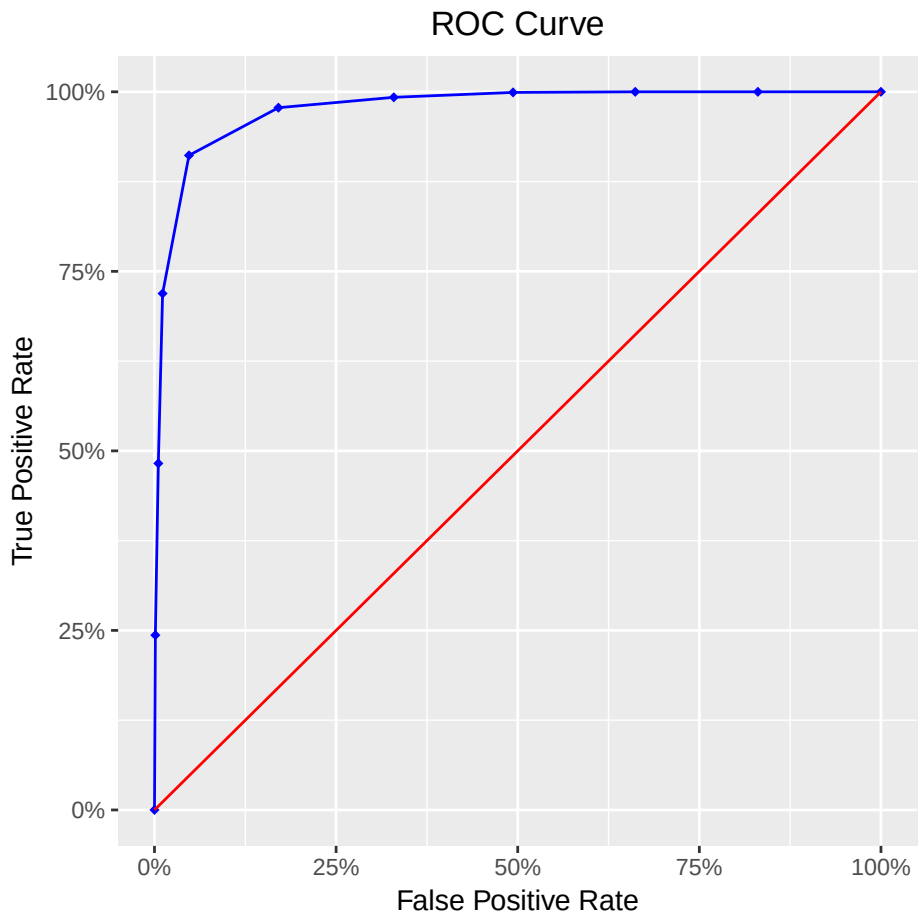
Some plots.

```
# Fit statistics
blr_model_fit_stats(glm.fit)
```

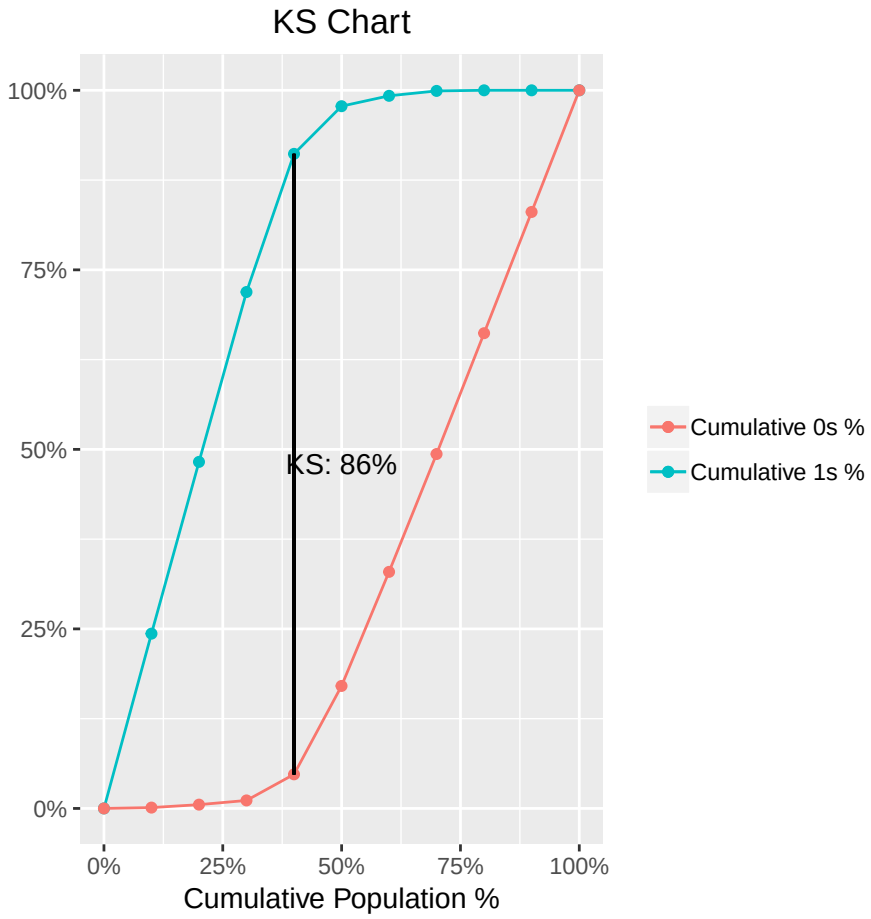
```
##                              Model Fit Statistics
## -------------------------------------------------------------------------------
## Log-Lik Intercept Only:     -1947.742    Log-Lik Full Model:            -544.307
## Deviance(2823):              1088.614    LR(57):                       2806.869
##                                          Prob > LR:                       0.000
## MCFadden's R2                   0.721    McFadden's Adj R2:               0.691
```

```
## ML (Cox-Snell) R2:                0.623    Cragg-Uhler(Nagelkerke) R2:        0.840
## McKelvey & Zavoina's R2:          0.997    Efron's R2:                        0.783
## Count R2:                         0.934    Adj Count R2:                      0.837
## BIC:                           1550.636    AIC:                            1204.614
## ---------------------------------------------------------------------------------
```

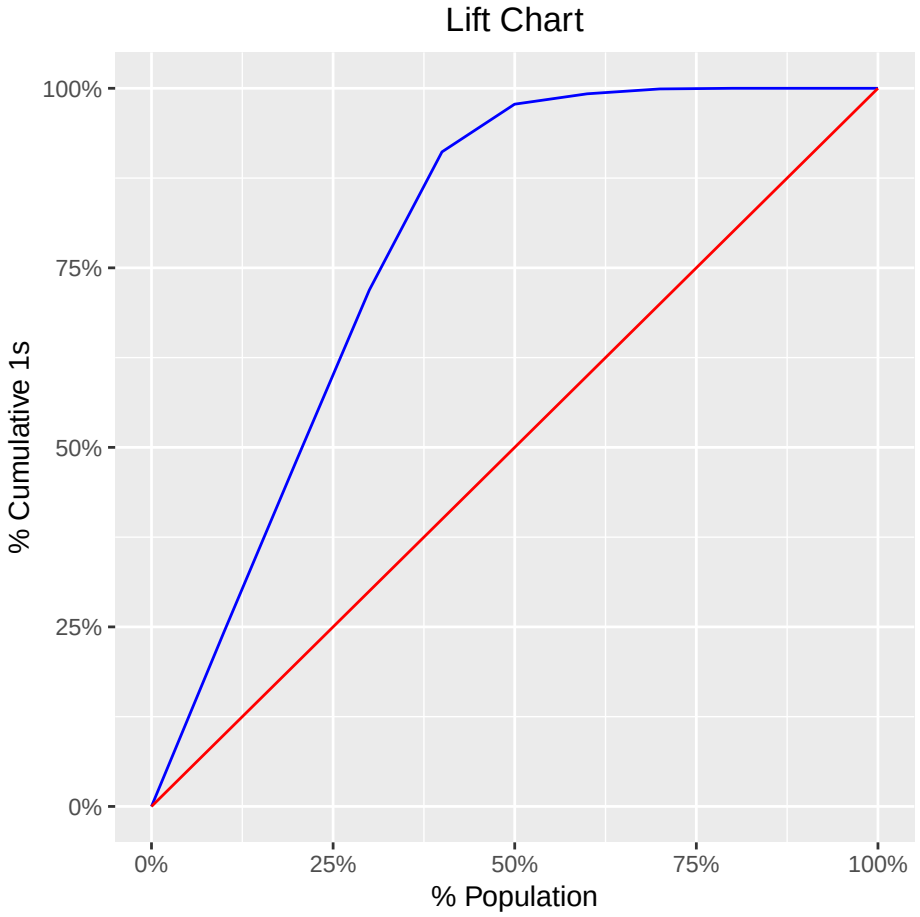
```
# Roc
glm.fit %>%
  blr_gains_table() %>%
  blr_roc_curve(xaxis_title = "False Positive Rate", yaxis = "True Positive Rate")
```




```
# KS chart
glm.fit %>%
  blr_gains_table() %>%
  blr_ks_chart()
```




```
# lift
glm.fit %>%
  blr_gains_table() %>%
  plot()
```



create a glm in h2o so we can plot it's roc.

```
glm.h2o <- h2o.glm(x = 1:57,
                   y = 58,
                   training_frame = as.h2o(Spam.train),
                   seed = 1234567,         # Seed for random numbers
                   family = "binomial",    # Outcome variable
                   lambda_search = FALSE,  # Optimum regularisation lambda
                   alpha = 0,            # Elastic net regularisation
                   nfolds = 1)             # N-fold cross validation

#Validation results
gbm.probs <- h2o.predict(
  object = glm.h2o,
  newdata = as.h2o(Spam.validation)
)
gbm.probs <- as.data.frame(gbm.probs)
```

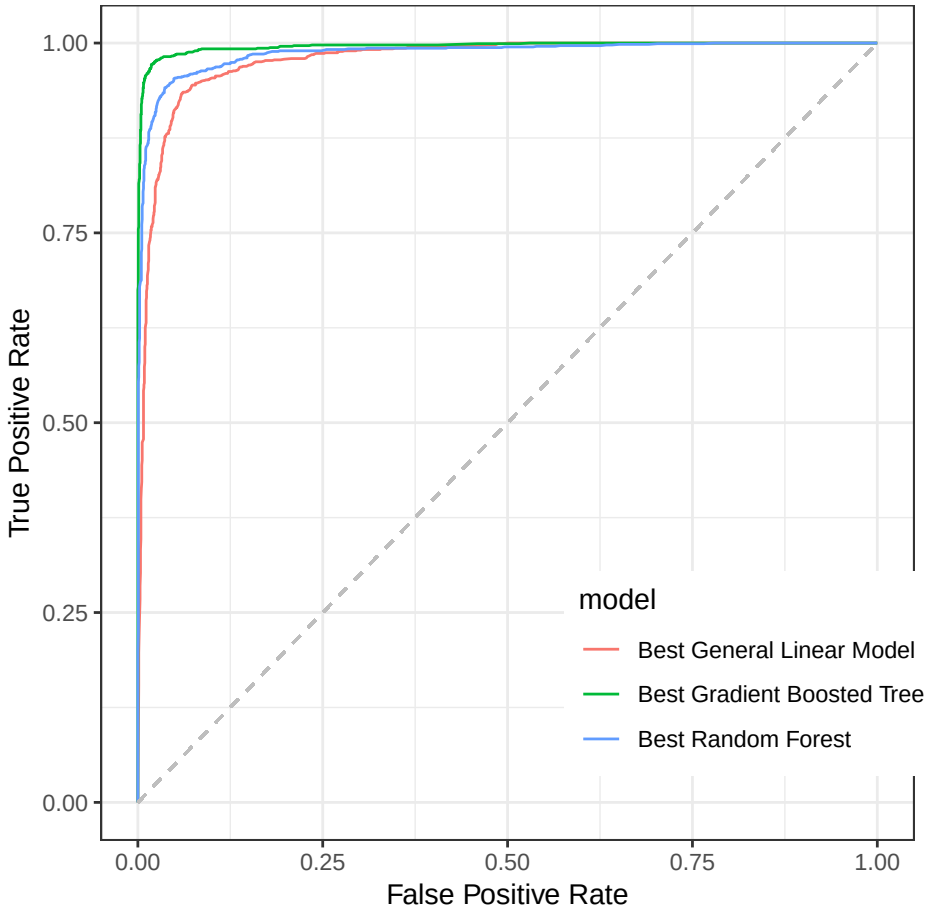
```
cm <- confusionMatrix(gbm.probs$predict, Spam.validation$spam, mode = "everything")
storeModelValidationPreformance("Repeated CV, GLM",cm)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction email spam
##      email   413   30
##      spam     22  255
##
##                Accuracy : 0.9278
##                  95% CI : (0.9064, 0.9456)
##     No Information Rate : 0.6042
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8483
##  Mcnemar's Test P-Value : 0.3317
##
##             Sensitivity : 0.9494
##             Specificity : 0.8947
##          Pos Pred Value : 0.9323
##          Neg Pred Value : 0.9206
##               Precision : 0.9323
##                  Recall : 0.9494
##                      F1 : 0.9408
##              Prevalence : 0.6042
##          Detection Rate : 0.5736
##    Detection Prevalence : 0.6153
##       Balanced Accuracy : 0.9221
##
##        'Positive' Class : email
##
```

this next graph's implementation was inspired from here: https://stackoverflow.com/questions/44034944/how-to-directly-plot-roc-of-h2o-model-object-in-r

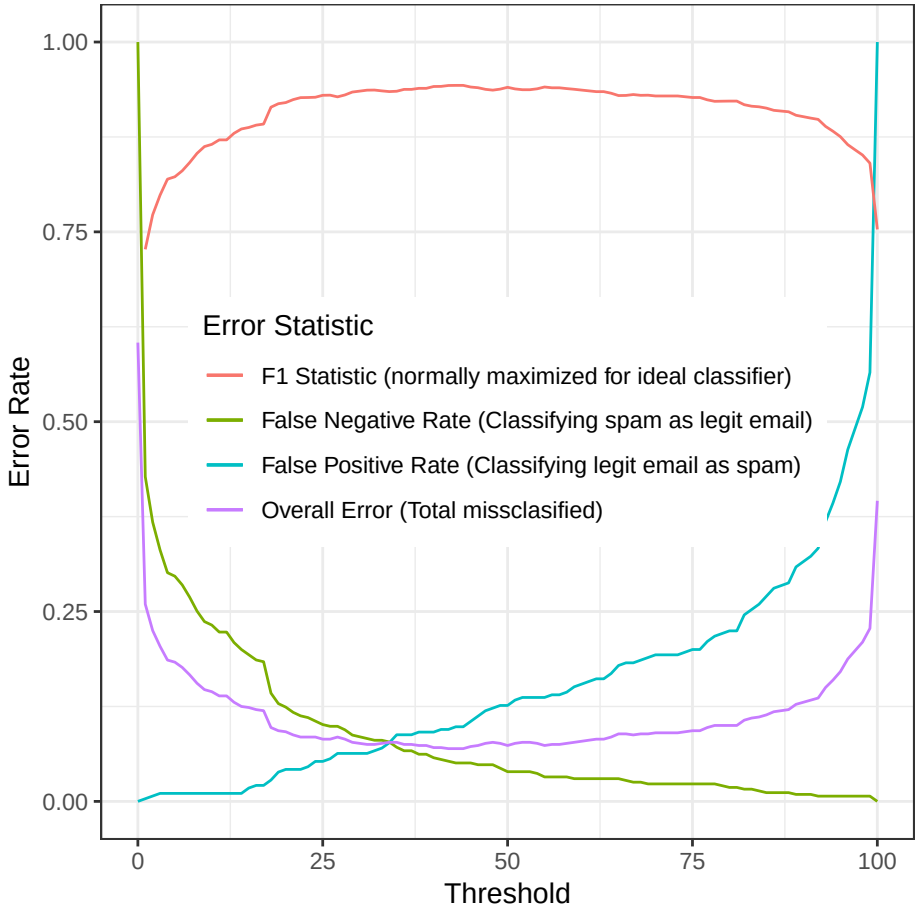
```
# for example I have 4 H2OModels
list(gbm.spam, rf.best2, glm.h2o) %>%
  # map a function to each element in the list
  purrr::map(function(x) x %>% h2o.performance() %>%
        # from all these 'paths' in the object
        .@metrics %>% .$thresholds_and_metric_scores %>%
        # extracting true positive rate and false positive rate
        .[c('tpr','fpr')] %>%
        # add (0,0) and (1,1) for the start and end point of ROC curve
        tibble::add_row(tpr=0,fpr=0,.before=T) %>%
        tibble::add_row(tpr=0,fpr=0,.before=F)) %>%
  # add a column of model name for future grouping in ggplot2
  purrr::map2(c('Best Gradient Boosted Tree','Best Random Forest','Best General Linear Model'),
        function(x,y) x %>% tibble::add_column(model=y)) %>%
  # reduce four data.frame to one
  purrr::reduce(rbind) %>%
  # plot fpr and tpr, map model to color as grouping
  ggplot(aes(fpr,tpr,col=model))+
  geom_line()+
  geom_segment(aes(x=0,y=0,xend = 1, yend = 1),linetype = 2,col='grey')+
  xlab('False Positive Rate')+
  ylab('True Positive Rate')+
  theme_bw() +
  theme(legend.position=c(0.8,0.2))
```



Lastly we will influence the threshold value to try and get the impact on the Error rate. First we will plot it for the GLM. Then it is done for the Boosted tree

```
results <- data.frame()
probs <- predict(glm.fit, newdata = Spam.validation[-58], type = "response")

for (i in 0:100){
  pred <- rep("email", length(probs))
  pred[probs > i/100] <- "spam"
  (cm <- confusionMatrix(factor(pred), Spam.validation$spam, mode = "everything"))
  (results <- rbind(results,
                    data.frame(
                              index = i,
                              FalsePositiveRate = 1 - cm$byClass["Specificity"],
                              FlaseNegativeRate = 1 - cm$byClass["Sensitivity"],
                              OverallError = 1 - cm$overall["Accuracy"],
                              F1 = cm$byClass["F1"])))

}

ggplot(results, aes(x=index, y=error)) +
  geom_line(aes(y = FalsePositiveRate,
                colour = "False Positive Rate (Classifying legit email as spam)")) +
  geom_line(aes(y = FlaseNegativeRate,
```

```
                colour = "False Negative Rate (Classifying spam as legit email)")) +
  geom_line(aes(y = OverallError, colour = "Overall Error (Total missclasified)")) +
  geom_line(aes(y = F1,
                colour = "F1 Statistic (normally maximized for ideal classifier)")) +
  xlab('Threshold') +
  ylab('Error Rate') +
  labs(colour = "Error Statistic") +
  theme_bw() +
  theme(legend.position=c(0.5,0.5))
```




```
probs <- predict(glm.fit, newdata = Spam.train[-58], type = "response")
glm.pred <- rep("email", length(probs))
glm.pred[probs > 0.85] <- "spam"
confusionMatrix(factor(glm.pred), Spam.train$spam, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email  1683  310
##      spam     23  865
##
##                Accuracy : 0.8844
##                  95% CI : (0.8722, 0.8959)
```

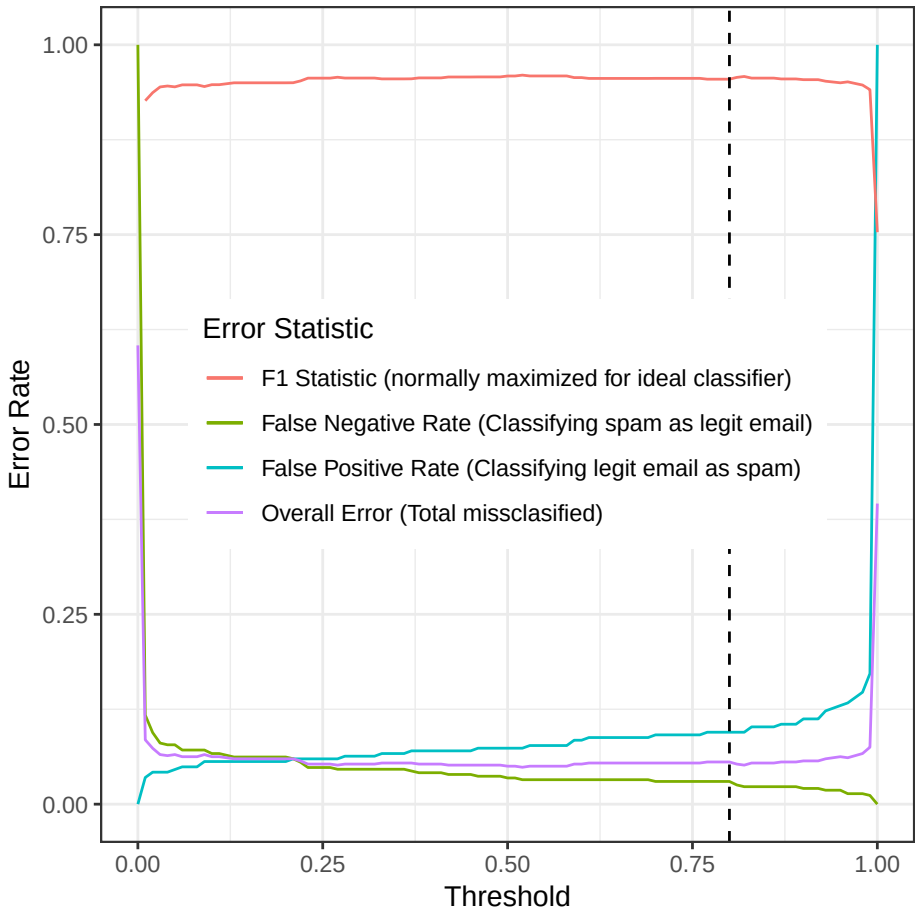
```
##     No Information Rate : 0.5922
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7512
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9865
##             Specificity : 0.7362
##          Pos Pred Value : 0.8445
##          Neg Pred Value : 0.9741
##               Precision : 0.8445
##                  Recall : 0.9865
##                      F1 : 0.9100
##              Prevalence : 0.5922
##          Detection Rate : 0.5842
##    Detection Prevalence : 0.6918
##       Balanced Accuracy : 0.8613
##
##        'Positive' Class : email
##
```

Lets do the same process for the best gbm

```
probs <- h2o.predict(object = gbm.best, newdata = as.h2o(Spam.validation))
```

```
results <- data.frame()

#we need to convert the h2o frame to a matrix to do the comparison
probs <- as.matrix(probs$spam)

#itterate through all elements and extract the values we want to plot
for (i in 0:100){
  pred <- rep("email", length(probs))
  pred[probs > i/100] <- "spam"
  (cm <- confusionMatrix(factor(pred), Spam.validation$spam, mode = "everything"))
  (results <- rbind(results,
                    data.frame(
                              index = i/100,
                              FalsePositiveRate = 1 - cm$byClass["Specificity"],
                              FlaseNegativeRate = 1 - cm$byClass["Sensitivity"],
                              OverallError = 1 - cm$overall["Accuracy"],
                              F1 = cm$byClass["F1"]))))
}
#generate the plot
ggplot(results, aes(index)) +
  geom_line(aes(y = FalsePositiveRate,
                colour = "False Positive Rate (Classifying legit email as spam)")) +
  geom_line(aes(y = FlaseNegativeRate,
                colour = "False Negative Rate (Classifying spam as legit email)")) +
  geom_line(aes(y = OverallError, colour = "Overall Error (Total missclasified)")) +
  geom_line(aes(y = F1,
                colour = "F1 Statistic (normally maximized for ideal classifier)")) +
  geom_vline(xintercept=0.80, linetype = "dashed") +
  xlab('Threshold') +
  ylab('Error Rate') +
```

```
  labs(colour = "Error Statistic") +
  theme_bw() +
  theme(legend.position=c(0.5,0.5))
```




```
gbm.probs <- h2o.predict(object = gbm.best, newdata = as.h2o(Spam.test))
```

```
gbm.probs <- as.data.frame(gbm.probs)

gbm.pred <- rep("email", length(gbm.probs$spam))
gbm.pred[gbm.probs$spam > 0.8] <- "spam"

cm <- confusionMatrix(as.factor(gbm.pred), Spam.test$spam, mode = "everything")
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction email spam
##      email   631   47
##      spam     16  306
##
##                Accuracy : 0.937
##                  95% CI : (0.9201, 0.9513)
##     No Information Rate : 0.647
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.8593
##  Mcnemar's Test P-Value : 0.0001571
##
##             Sensitivity : 0.9753
##             Specificity : 0.8669
##          Pos Pred Value : 0.9307
##          Neg Pred Value : 0.9503
##               Precision : 0.9307
##                  Recall : 0.9753
##                      F1 : 0.9525
##              Prevalence : 0.6470
##          Detection Rate : 0.6310
##    Detection Prevalence : 0.6780
##       Balanced Accuracy : 0.9211
##
##        'Positive' Class : email
##
```