MAS3907 - Big Data Analytics

PROJECT - PREDICTING THE NATURAL LOGARITHM OF THE PER CAPITA CRIME RATE

Stephen Cole, Joseph Crone, Harry Johnston, Calum Doran

Academic year 2019/20

Contents

1 Introduction

In this report we will be analysing the **Boston** data-set, which concerns the value of housing in the Boston Standard Metropolitan Statistical Area in 1970, to build a model which predicts the natural logarithm of the per capita crime rate. However, we shall only be considering a random sample of size 400 from the full data-set, generated with seed 7045607 in R for fitting the model. The remaining portion of the data set will be used as test data for out-of-sample validation, to assess how well the given model is at predicting new values. A description of each variable can be found in Appendix A.

2 Exploratory Data Analysis

To ensure "fairness" in the variables we first need to put them on a common scale because some variables have readings over a greater range than others. This has been done by standardising them (taking away the overall mean and dividing by the standard deviation for each variable - shown below) to give each variable a mean of 0 and variance 1 in the data.

From now on in this report all the variables have been transformed in this manner. Hence, for prediction you would need to "un-standardise" them to get them on their original scale.

```
> #Means of each variable before any sacaling
> m
     lcrim         zn      indus       chas        nox         rm        age       disf 
-0.8294272 12.2837500 10.9275250  0.0675000  0.5505317  6.3190850 32.4297500  1.9975000 
       rad        tax    ptratio      black      lstat       medv 
 9.2925000 403.8075000 18.3842500 358.8000000 12.5022000 22.9177500 
> 
```

Figure 1: Means of our variables for our non-scaled Boston data

```
#Standard Deviations of unscaled data
sqrt(sv)
    lcrim        zn     indus      chas       nox        rm       age      disf       rad       tax   ptratio     black     lstat      medv 
2.1615286 23.9698821 6.9365817 0.2512001 0.1163354 0.7047096 28.2952858 0.9217471 8.6146663 167.3228400 2.1977151 88.7594105 7.2394017 9.4413855 
```

Figure 2: Standard Deviations of our non-scaled variables for our Boston data

Also, we need to consider the *dis* and *chas* variables as it is a ordered categorical variable with 4 and 2 levels respectively (numbered 1 to 4 and 0 to 1). They can either be treated as factors (with indicator variables) or just as a normal quantitative variable. We have decided on the latter as factors are not able to be scaled on R and we have to standardise our explanatory variables in order to perform regression. Therefore we treat our ordered categorical variables *disf* and *chas* as quantitative values as if they were continuous and scale them as above.

Before we start to build a model using this data we will need to look into the variables, as well as the relationship between them, in more depth.

	lcrim	zn	indus	chas	nox	rm	age	disf	rad
lcrim	1.00000000	-0.50953039	0.70681504	0.01846325	0.79162498	-0.29889959	-0.63466949	-0.66852581	0.85707860
zn	-0.50953039	1.00000000	-0.53107923	-0.03493407	-0.52172524	0.31003792	0.55833025	0.59437331	-0.30731648
indus	0.70681504	-0.53107923	1.00000000	0.03230054	0.75452404	-0.38992554	-0.63615909	-0.71918487	0.56975644
chas	0.01846325	-0.03493407	0.03230054	1.00000000	0.06087249	0.09729327	-0.08451994	-0.06477838	-0.02135836
nox	0.79162498	-0.52172524	0.75452404	0.06087249	1.00000000	-0.27580217	-0.73409313	-0.78563426	0.62133699
rm	-0.29889959	0.31003792	-0.38992554	0.09729327	-0.27580217	1.00000000	0.23078350	0.20882757	-0.21149049
age	-0.63466949	0.55833025	-0.63615909	-0.08451994	-0.73409313	0.23078350	1.00000000	0.74433406	-0.43642650
disf	-0.66852581	0.59437331	-0.71918487	-0.06477838	-0.78563426	0.20882757	0.74433406	1.00000000	-0.46604229
rad	0.85707860	-0.30731648	0.56975644	-0.02135836	0.62133699	-0.21149049	-0.43642650	-0.46604229	1.00000000
tax	0.81972051	-0.31627996	0.71348239	-0.05676726	0.67879314	-0.29572894	-0.49729277	-0.53551684	0.90083733
ptratio	0.39456912	-0.38626433	0.41463006	-0.13227714	0.22769632	-0.40479441	-0.24532660	-0.23742206	0.45894831
black	-0.47839662	0.17598887	-0.35421214	0.04328632	-0.40142594	0.13886056	0.26753893	0.32189685	-0.43314736
lstat	0.61265440	-0.41293834	0.60742601	-0.03177035	0.58118660	-0.61693227	-0.60857851	-0.51176570	0.49307669
medv	-0.45774487	0.36971350	-0.49948099	0.15450214	-0.43101012	0.71291311	0.38013301	0.29716664	-0.39480888

	tax	ptratio	black	lstat	medv
lcrim	0.81972051	0.3945691	-0.47839662	0.61265440	-0.4577449
zn	-0.31627996	-0.3862643	0.17598887	-0.41293834	0.3697135
indus	0.71348239	0.4146301	-0.35421214	0.60742601	-0.4994810
chas	-0.05676726	-0.1322771	0.04328632	-0.03177035	0.1545021
nox	0.67879314	0.2276963	-0.40142594	0.58118660	-0.4310101
rm	-0.29572894	-0.4047944	0.13886056	-0.61693227	0.7129131
age	-0.49729277	-0.2453266	0.26753893	-0.60857851	0.3801330
disf	-0.53551684	-0.2374221	0.32189685	-0.51176570	0.2971666
rad	0.90083733	0.4589483	-0.43314736	0.49307669	-0.3948089
tax	1.00000000	0.4775422	-0.43234634	0.55182562	-0.4866820
ptratio	0.47754220	1.0000000	-0.15872974	0.40984209	-0.5356023
black	-0.43234634	-0.1587297	1.00000000	-0.36033368	0.3389722
lstat	0.55182562	0.4098421	-0.36033368	1.00000000	-0.7417908
medv	-0.48668201	-0.5356023	0.33897219	-0.74179084	1.0000000

Figure 3: Correlation matrix of variables of our Boston data

2.1 Relationship between response and predictor variables

From looking at the results above, we can generally deduce that the response variable, the logarithm of the crime rate, is positively correlated with $indus, nox, rad, tax$ and $lstat$. It also appears to be negatively correlated with $age, dis, black$ and $medv$.
You could also see these relationships using by plotting the respective variables (this can also be used to see the linearity of the relationship). These plots are shown in the additional information in Appendix B.

2.2 The predictor variables

It is also important that we look at any potential strong correlation between the predictor variables used in the initial model as this could lead to problems with multi-co linearity.

Using the correlation matrix above, most of these predictor variables are reasonably uncorrelated however there is several with a value of greater than 0.7 in modulus. This includes tax and rad which has a very high value (over 0.9) but when you look at the scatter plots this appears to be more due to a high extreme value. Therefore, for now, we will proceed to build a model based on these variables and the data.

3 Least Squares Model

We will begin by fitting the model via least squares using the lm function. We can then obtain a summary of the fitted model by passing the returned object to the summary function:

```
Call:
lm(formula = y ~ ., data = boston_data)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.55851 -0.55424 -0.02385  0.47902  2.52856 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -0.8294272  0.0383218 -21.644  < 2e-16 ***
zn          -0.2663419  0.0561771  -4.741 2.99e-06 ***
indus        0.0652005  0.0785974   0.830  0.40731    
chas        -0.0052923  0.0398165  -0.133  0.89433    
nox          0.3800353  0.0815975   4.657 4.41e-06 ***
rm          -0.0645159  0.0627794  -1.028  0.30475    
age         -0.1437078  0.0697179  -2.061  0.03995 *  
disf        -0.1881279  0.0762045  -2.469  0.01399 *  
rad          1.2453262  0.1027847  12.116  < 2e-16 ***
tax          0.0007605  0.1154701   0.007  0.99475    
ptratio     -0.1238606  0.0550888  -2.248  0.02512 *  
black       -0.1253831  0.0454454  -2.759  0.00607 ** 
lstat        0.2039757  0.0725053   2.813  0.00516 ** 
medv         0.0800056  0.0770380   1.039  0.29968    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7664 on 386 degrees of freedom
Multiple R-squared:  0.8784,	Adjusted R-squared:  0.8743 
F-statistic: 214.4 on 13 and 386 DF,  p-value: < 2.2e-16
```

Figure 4: Summary for least squares model

```
> mean(fit_raw$coefficients^2) #MSE of non-scaled variables of least squares
[1] 0.9966208
> mean(fit$residuals^2) #MSE of scaled variables for least squares
[1] 0.566863
> 
```

Figure 5: The mean squared errors of both scaled and non-scaled variables for least squares estimate.

The table of p-values for the t-tests confirms our observations from the exploratory analysis; if we consider the explanatory variables one-at-a-time, there are several which would not be needed in a model containing all others. We see that we have a least squares model (remember the scaling in the variables if inputting values):

$$
\begin{aligned}
lcrim = & -0.8294272 - 0.2663419 \times zn + 0.0652005 \times indus - 0.0052923 \times chas \\
& + 0.3800353 \times nox - 0.0645159 \times rm - 0.1437078 \times age - 0.1881279 \times disf \\
& + 1.2453262 \times rad + 0.0007605 \times tax - 0.1238606 \times ptratio - 0.1253831 \times black \\
& + 0.2039757 \times lstat + 0.0800056 \times medv
\end{aligned}
$$

3.1 Cross Validation

To get an assessment of the predictive performance of the least squares model, we can estimate the test error using cross validation. In this project we will consider 10-fold cross validation.

```
> cv_lsq_errors
 [1] 0.4799388 0.6161233 0.6474747 0.4770355 0.5581088 0.9070497
 [7] 0.5119569 0.6599446 0.6085279 0.7168907
```

Figure 6: Test error rates

This shows the test error rates based on each fold. We get the average MSE over the entire data set by averaging the 10 estimates from our 10 folds. To allow for the fact that the 10 estimates were computed using test sets (i.e. folds) of different sizes, we weight the estimates by the sizes of the folds.

```
> (lsq_final_mse = weighted.mean(cv_lsq_errors, w=fold_sizes))
[1] 0.6144559
```

Figure 7: Average MSE

3.2 Out-of-Sample Validation

As previously mentioned, we have extra data which we have assigned to be test data. Using a validation set approach, we can find the test error using data which was not used in the model building process. Using the $predict$ function, the LSQ model and the test data, we can calculate the MSE for the test data.

```
> yhat_test = predict(fit, boston_data_test)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.6838521
> |
```

Figure 8: Output of the test error

4 Subset Selection

As it is clear that all the original predictor variables are not required in the final model, in order to make this as simple as possible, we will used several methods to try and discover the optimum number of variables needed and what these are.

4.1 Best Subset Selection

The first method of this type we will use is **Best Subset Selection**. This involves taking every possible combination of predictor variables and using the least squares method again to find the optimum model for each total number of variables, k (from which the "perfect" model can be chosen).

To choose this "perfect" model we can use a variety of methods including **adjusted** R^2 (aim to maximise), **Mallow's** C_p **statistic** and **Bayes Information Criterion, BIC** (both aiming to minimise).

The following is the output for the value of these methods for each number of variables, k. Then this is automatically used to pick the best number of variables (and hence the model) for each of the 3 methods. As you can see this gives different values for each.

```
> bss_summary$adjr2 #Adjr2 values
 [1] 0.7225663 0.8408288 0.8589328 0.8650664 0.8707339 0.8725496 0.8741551 0.8750703 0.8750859 0.8749576 0.8749151 0.8745978 0.8742729
> bss_summary$cp    #Cp statistic of each component
 [1] 482.240440 108.604296  52.316325  33.924177  17.090601  12.386858   8.367219   6.520183   7.478075   8.881537  10.018099
[12]  12.000043  14.000000
> bss_summary$bic   #BIC values
 [1] -501.8901 -719.1456 -762.4607 -775.2621 -787.4482 -788.1316 -788.2302 -786.1800 -781.2629 -775.8877 -770.7900 -764.8172 -758.8258
> best_adjr2 #Components needed for highest adjR2
[1] 9
> best_cp    #Components needed for lowest CP
[1] 8
> best_bic   #lowest BIC
[1] 7
>
```

Figure 9: Output for best subset selection using each method for accuracy

Below we can get a better view of how each value changes with the number of variables in order to choose the perfect number to have in the model. Therefore, we have decided to have 7 variables as that gives close to the optimum value for each of the 3 methods (R^2_{adj}, C_p and BIC).

```
Selection Algorithm: exhaustive
          zn  indus chas nox rm  age disf rad tax ptratio black lstat medv
1  ( 1 )  " " " "   " "  " " " " " " " "  "*" " " " "     " "   " "   " " 
2  ( 1 )  " " " "   " "  "*" " " " " " "  "*" " " " "     " "   " "   " " 
3  ( 1 )  "*" " "   " "  "*" " " " " " "  "*" " " " "     " "   " "   " " 
4  ( 1 )  "*" " "   " "  "*" " " " " " "  "*" " " " "     " "   "*"   " " 
5  ( 1 )  "*" " "   " "  "*" " " " " "*"  "*" " " " "     " "   "*"   " " 
6  ( 1 )  "*" " "   " "  "*" " " " " "*"  "*" " " "*"     " "   "*"   " " 
7  ( 1 )  "*" " "   " "  "*" " " " " "*"  "*" " " "*"     "*"   "*"   " " 
8  ( 1 )  "*" " "   " "  "*" " " "*" "*"  "*" " " "*"     "*"   "*"   " " 
9  ( 1 )  "*" "*"   " "  "*" " " "*" "*"  "*" " " "*"     "*"   "*"   " " 
10  ( 1 ) "*" " "   " "  "*" "*" "*" "*"  "*" " " "*"     "*"   "*"   "*" 
11  ( 1 ) "*" "*"   " "  "*" "*" "*" "*"  "*" " " "*"     "*"   "*"   "*" 
12  ( 1 ) "*" "*"   "*"  "*" "*" "*" "*"  "*" " " "*"     "*"   "*"   "*" 
13  ( 1 ) "*" "*"   "*"  "*" "*" "*" "*"  "*" "*" "*"     "*"   "*"   "*" 
```

Figure 10: Which variables to use for each total number

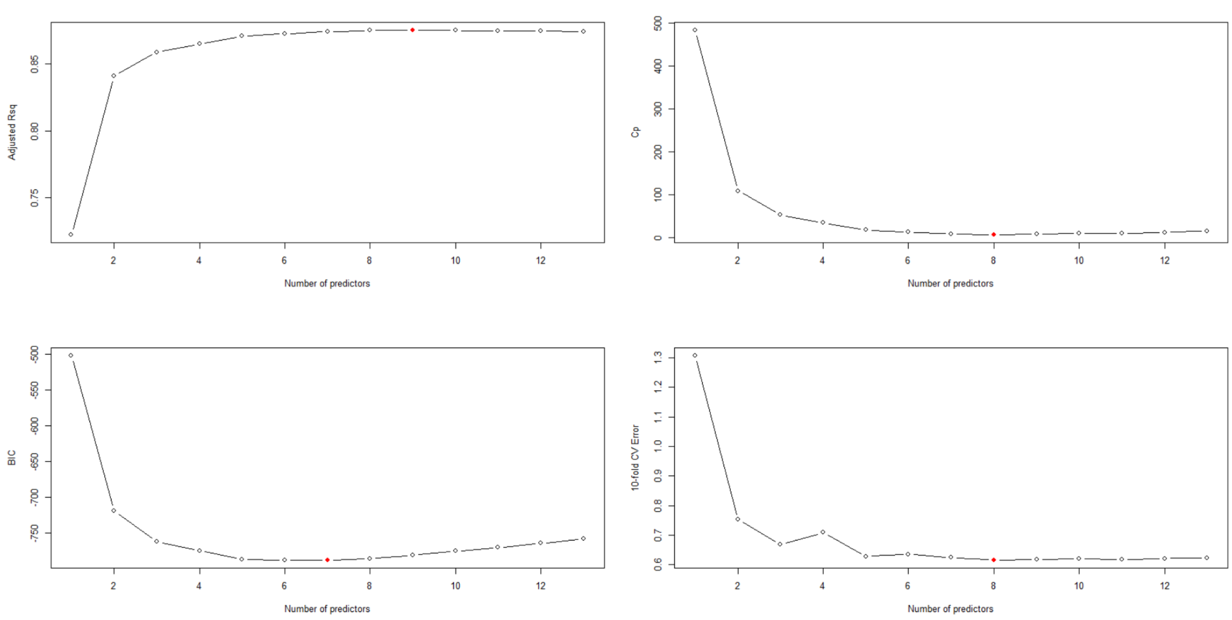

Figure 11: Graphs for each method, including cross-validation.

We find that we have the following fitted model using 7 explanatory variables:

$$\begin{aligned} lcrim = &-0.8294272 - 0.2900970 \times zn + 0.4369694 \times nox - 0.2655847 \times disf \\ &+ 1.2445161 \times rad - 0.1248727 \times ptratio - 0.1079895 \times black + 0.2363643 \times lstat \end{aligned}$$

4.1.1 Cross-validation

Again we will focus on 10-fold cross validation. We will need to repeat the procedure for all the models and then compare the average MSE for each of these models, and find the one minimising the test error.

```
> bss_mse
 [1] 1.3061835 0.7519041 0.6683897 0.7082830 0.6280319 0.6345905 0.6244488 0.6155674 0.6174723 0.6191997 0.6180251 0.6210629 0.6216705
>
> (best_cv = which.min(bss_mse))
[1] 8
>
```

Figure 12: Output for cross validation

Here we see that model 8 has the lowest test error with an error of 0.6155674. This is slightly higher than the test error of the least squares model which was 0.6144559. These result are shown graphically above in figure 11.

4.1.2 Out-of-Sample Validation

```
> #Out of sample for Best subset
> yhat_test = predict(bss_tmp_fit, boston_data_test)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.6977663
> |
```

Figure 13: Performing out-of-sample validation using the validation set approach.

5 Regularisation methods

This method works by smoothing the least squares function $L(\underline{\beta})$ to make sure that it has a unique minimum. This results in shrinking the estimates for the regression coefficients towards zero. We shall consider two regularisation methods - **ridge regression** and the **LASSO**.

We only want to shrink the coefficients of the explanatory variables, excluding the intercept, which is a measure of the mean response. Shrinkage methods lead to non-trivial relationships between the scale of the regression coefficients and the explanatory variables.

5.1 Ridge Regression

The ridge regression estimator $\hat{\underline{\beta}}_1^r$ is not an unbiased estimator for $\underline{\beta}_1$ when $\lambda > 0$. However, it can be used regardless of the relative sizes of n and p. For $\lambda > 0$, ridge regression will give a unique solution.

Ridge regression produces a different estimate $\hat{\underline{\beta}}_1^r$ for every value of λ, which we regard as the tuning parameter and select a good value. We use cross validation, evaluate the error over a grid of values for $\hat{\lambda}$ and choose the value which minimises the error. This process can be shown in Appendix B - Figure B.4

For our values of our tuning parameter, we shall choose a grid that ranges from $\hat{\lambda} = 10^5$ (lots of shrinkage) to $\hat{\lambda} = 10^{-3}$ (very little shrinkage). From the program list we see that we obtain the following coefficient with the corresponding tuning parameter:

```
#Some examples of coefficients for different levels of shrinkage
beta1_hat[,1] #Lots of shrinkage
 (Intercept)            zn         indus          chas           nox            rm           age          disf           rad           tax       ptratio         black         lstat
3.294272e-01 -2.465697e-05  3.374267e-05  6.501369e-07  3.683696e-05 -1.409403e-05 -3.096161e-05 -3.218198e-05  3.958314e-05  3.804513e-05  1.745400e-05 -2.238739e-05  2.906908e-05
        medv
2.129902e-05
beta1_hat[,75] #Some shrinkage
(Intercept)           zn        indus         chas          nox           rm          age         disf          rad          tax      ptratio        black        lstat         medv
0.829427190 -0.261999861  0.041374519  0.005604361  0.376310685 -0.043965372 -0.145608838 -0.197022741  0.994386742  0.213208766 -0.088748368 -0.140328715  0.209888910  0.078079123
beta1_hat[,100] #Very little shrinkage. Very similar to that of least squares estimate
(Intercept)           zn        indus         chas          nox           rm          age         disf          rad          tax      ptratio        black        lstat         medv
0.829427190 -0.266833258  0.063565619 -0.004999089  0.380641216 -0.063930080 -0.143220628 -0.188238135  1.238303793  0.007301692 -0.122996342 -0.125801596  0.204851143  0.080577923
```

Figure 14: Examples of ridge regression coefficients with different levels of shrinkage.

This shows that increasing the tuning parameter shrinks the regression coefficient towards 0. This can be plotted to show how the coefficients shrink with varying $\hat{\lambda}$, shown in Appendix B - Figure B.5. From looking at the graph, we can clearly see that all regression coefficients are essentially 0 when $\log \hat{\lambda}$ is around 6.

As previously mentioned, we need to find a good value for the tuning parameter that minimises the error which is done by cross-validation. This is accomplished from the following:

```
> #Extracting the value of lambda which corresponds to the minimum
> (lambda_min = ridge_cv_fit$lambda.min)
[1] 0.04977024
> #The tuning parameter which is the minimum.
> (i=which(ridge_cv_fit$lambda == ridge_cv_fit$lambda.min))
[1] 79
> #Corresponding mean MSE
> ridge_cv_fit$cvm[i]
[1] 0.6199296
>
```

Figure 15: Finding a good value of $\hat{\lambda}$ which minimises the error

We see that the 79^{th} point of our cross-validation ridge regression fit has the value which minimises the error, resulting in a mean MSE of 0.6199296 shown in Figure 14. We choose our tuning parameter to be 0.04977024 and we find that the coefficients for this regression is:

$$
\begin{aligned}
\textit{lcrim} = & -0.8294271897 - 0.2677136275 \times \textit{zn} + 0.0458406365 \times \textit{indus} + 0.0004068605 \times \textit{chas} \\
& + 0.3807799386 \times \textit{nox} - 0.0548999412 \times \textit{rm} - 0.1424292146 \times \textit{age} - 0.1925059818 \times \textit{disf} \\
& + 1.1148724598 \times \textit{rad} + 0.1185417718 \times \textit{tax} - 0.1070468379 \times \textit{ptratio} - 0.1328564108 \times \textit{black} \\
& + 0.2101482417 \times \textit{lstat} + 0.0833100559 \times \textit{medv}
\end{aligned}
$$

Compared with our least squares solution in Section 3, we can see that most variables have shrunk towards 0, most notably *indus, chas, rm and ptratio*, which is at least 13% smaller. However, *indus and chas* are 30% and 92% smaller respectively and has shrunk more significantly in comparison to the least squares estimate.

We can use the predict function, shown in the program list in Appendix B - Figure B.3, to find a prediction at our new set of explanatory variables with our tuning parameter. We obtain the following results:

$i^{th}row$	$\hat{y}$ value
410	2.3581811
52	-2.4282134
87	-2.1371956
376	1.9339514
...	...

The table to the left shows how the different values of our explanatory variables in the i^{th} row affects our predicted value for our response variable $\hat{y}$. A larger set of predicted values can be seen in Appendix B - Figure B.6

An issue with ridge regression is that is will always contain all the explanatory variables, 13 variables in this case. This is due to shrinking the coefficients but does is never equal to 0. This makes our interpretation of the fitted model very difficult. However, we are only concerned with predicting the values so it is not so much an issue in that respect. If the amount of subsets was an issue, we can just simply remove the variables that are close to 0. On the other hand, if reducing subsets was the goal then the LASSO method, which we shall discuss next, would be the ideal tool for model building using a shrinking method.

5.1.1 Out-of-Sample Validation

```
> #Out of sample for Ridge
> yhat_test = predict.glmnet(ridge_fit, X1_test, s = lambda_min)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.6842727
>
```

Figure 16: Performing out-of-sample validation using the validation set approach.

5.2 The LASSO

Similar to the ridge regression model, we will be using a range from 10^5 to 10^{-3} for our tuning parameter. We will also use the $glmnet$ function to perform the LASSO. Below we see how we obtain different coefficients using different tuning parameters.

From this we see that when a tuning parameter of $\lambda = 10^5$ is used, the coefficients all shrink to zero. Also, when the tuning parameter is $\lambda = 10^{-3}$ all coefficients are non-zero. We can see the effect of varying the tuning parameter λ has on the coefficients, shown in Appendix - Figure B.9. From this graph we see that when $\log\lambda \approx 1$, all coefficients shrink to zero.

We now need to find the value of the tuning parameter that minimises the error. We can achieve this using the following function.

```
> grid[1]
[1] 1e+05
> beta1_hat[,1]
(Intercept)          zn       indus        chas         nox          rm         age        disf         rad         tax     ptratio 
 -0.8294272   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000   0.0000000 
      black       lstat        medv 
  0.0000000   0.0000000   0.0000000 
> grid[75]
[1] 0.1047616
> beta1_hat[,75]
(Intercept)          zn       indus        chas         nox          rm         age        disf         rad         tax     ptratio 
-0.82942719 -0.16301615  0.02787166  0.00000000  0.45586197  0.00000000 -0.11432882 -0.20493860  1.14570341  0.00000000  0.00000000 
      black       lstat        medv 
-0.05332096  0.13699629  0.00000000 
> grid[100]
[1] 0.001
> beta1_hat[,100]
 (Intercept)           zn        indus         chas          nox           rm          age         disf          rad          tax      ptratio 
-0.829427190 -0.264591171  0.064292447 -0.003587522  0.380959801 -0.059960036 -0.142721101 -0.190093366  1.244362268  0.000000000 -0.122234253 
       black        lstat         medv 
-0.123788438  0.200877799  0.072156714 
> 
```

Figure 17: Examples of LASSO coefficients with different levels of shrinkage.

```
> (lambda_min = lasso_cv_fit$lambda.min)
[1] 0.00367838
> (i = which(lasso_cv_fit$lambda == lasso_cv_fit$lambda.min))
[1] 93
> lasso_cv_fit$cvm[i]
[1] 0.5991631
> 
```

Figure 18: Finding a good value of $\hat{\lambda}$ which minimises the error

We see that the 93^{rd} point of our cross-validation LASSO fit has the value which minimises the error, we can also find the mean MSE for that particular value of the tuning parameter. The value of this tuning parameter is 0.5991631. Using this we can find the coefficients of the fitted model and also see which explanatory values drop out of the model.

```
> coef(lasso_fit, s=lambda_min)
14 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) -0.82942719
zn          -0.26061455
indus        0.06263766
chas         .         
nox          0.38281362
rm          -0.04910525
age         -0.14073802
disf        -0.19326884
rad          1.23975952
tax          .         
ptratio     -0.11734337
black        0.12016095
lstat        0.19561037
medv         0.05524359
> 
```

Figure 19: Regression coefficients for LASSO model.

Here we have the values of the coefficients if we use the value of the tuning parameter which minimises the error. We also see that the explanatory variables *chas* and *tax* have been dropped from the model.

5.2.1 Out-of-Sample Validation

```
> #Out of sample for LASSO
> yhat_test = predict.glmnet(lasso_fit, X1_test, s = lambda_min)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.6745852
> 
```

Figure 20: Performing out-of-sample validation using the validation set approach.

6 Dimension reduction methods

This method involves controlling the variance of our estimators for the regression coefficients. Rather than regressing on our explanatory variables, we shall regress on $m < p$ linear transformations of them, each of which is a good predictor for a response.

6.1 Principal Component Regression (PCR)

We first need to standardise the explanatory variables onto a common scale as there are variables with a large variance relative to others. If our variables are not scaled then they will dominate the first few principal components, resulting in the domination of the principal component regression. This can be seen on the program list in Appendix B- Figure B.11

By running the program list we can extract the loading matrix and examine the directions our new variables represent:

```
> c #attr - proportion of variation of the standardised explanatory variables X1 that is explained by each of the PCs.
             Comp 1      Comp 2      Comp 3      Comp 4      Comp 5      Comp 6       Comp 7       Comp 8       Comp 9     Comp 10     Comp 11      Comp 12     Comp 13
zn       0.259573121 -0.10416046 -0.34732714  0.27398457 -0.36268122  0.40794220 -0.260309762  0.535412943 -0.008269173 -0.25261975  0.05037762 -0.003881659  0.08513466
indus   -0.345571313  0.11723292  0.02358157 -0.01916156  0.04692171  0.14014929  0.386515267  0.543951850  0.035301270  0.48836515 -0.29024975 -0.065726516  0.26548244
chas     0.009711657  0.32694943  0.29137055  0.87278528  0.09916009 -0.17569517  0.009072404  0.012679919 -0.025534431  0.01107464  0.03427795  0.053344041 -0.02487959
nox     -0.335679981  0.26487288  0.03389959 -0.03962442 -0.17807143  0.17309845  0.134141532 -0.058298436 -0.460613834 -0.20748614  0.30759806 -0.613905298 -0.06830067
rm       0.215175935  0.47476798 -0.33794805 -0.13409666  0.19963927 -0.10394369 -0.394913164  0.064113433 -0.203318792  0.50029670  0.30434587  0.033351095  0.00167209
age      0.307679777 -0.27585868 -0.20942725  0.14164903  0.05820572 -0.02168390  0.572555273 -0.020540003  0.214762637  0.20584593  0.56557425 -0.144138178 -0.06376367
disf     0.306015964 -0.35762663 -0.13864487  0.21331112  0.02006338  0.01508866 -0.067442788 -0.260984429 -0.348799639  0.32546241 -0.48043772 -0.426070829  0.02231727
rad     -0.304022050  0.03931452 -0.45008140  0.18444151  0.18932913  0.20524952 -0.019101461 -0.418917067  0.158410382 -0.11264353  0.02668780  0.049493468  0.61209436
tax     -0.325975659  0.01257859 -0.40093515  0.14982119  0.12231216  0.30032442  0.099191121 -0.112578849 -0.021866932  0.07123828 -0.14672291  0.231085636 -0.70868690
ptratio -0.216063265 -0.35343919 -0.07424443  0.04623530  0.66651414 -0.23380246 -0.247387127  0.369204131 -0.008374399 -0.21574479  0.10323992 -0.252473372 -0.05756273
black    0.197689895 -0.01961769  0.45237954 -0.05034647  0.41138366  0.74022227 -0.073000452 -0.079861773 -0.032676196  0.07912607  0.11884661  0.046795651  0.02891930
lstat   -0.320554067 -0.19447637  0.17006010  0.09085334 -0.29626236  0.06897671 -0.439939718 -0.101911590  0.505573457  0.37618894  0.17918996 -0.290631667 -0.10994871
medv     0.280364980  0.45180194 -0.11431771 -0.09111092  0.17034512  0.04398639  0.078510399  0.001873323  0.542615171 -0.20934700 -0.30817641 -0.454164747 -0.13887182
attr(,"explvar")
    Comp 1     Comp 2     Comp 3     Comp 4     Comp 5     Comp 6     Comp 7     Comp 8     Comp 9    Comp 10    Comp 11    Comp 12    Comp 13
47.6844780 12.8566832  9.4877204  6.9378805  6.3987539  4.9556678  3.0299295  2.4321197  1.6026213  1.5424851  1.4409198  1.1574988  0.4732421
>
```

Figure 21: Loading matrix of standardised explanatory variables

The first PC is essentially a contrast of *zn, rm, age, disf, black* and *medv* with *indus, nox, rad, tax, ptratio* and *lstat*. The second PC could be interpreted as the value of owner-occupied homes with lots of rooms per dwelling that takes into account the mean distance to the five Boston employment centres and the amount of teachers in the area.

The attribute **explvar** gives the proportion of variation of the standardised explanatory variables that is explained by each of the PCs.

```
> theta1_hat #comp 4,5,7,11 are close to 0 which suggests that this is a dimension in which X1 shows variation not strongly associated
      Comp 1    Comp 2     Comp 3   Comp 4     Comp 5    Comp 6     Comp 7     Comp 8    Comp 9    Comp 10    Comp 11    Comp 12   Comp 13
y -0.7547909 0.3022933 -0.3994171 0.095897 0.06137171 0.1837499 0.03397497 -0.6595738 0.2263901 -0.1663112 0.09117509 -0.1465238 0.7053512
>               #with the response.
> summary(pcr_fit) #Cumulative proportion of variation in X1 explained by different numbers of PCs
Data:   X dimension: 400 13
        Y dimension: 400 1
Fit method: svdpc
Number of components considered: 13
TRAINING: % variance explained
   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
X    47.68    60.54    70.03    76.97    83.37    88.32    91.35    93.78    95.39     96.93     98.37     99.53    100.00
y    75.59    78.86    83.07    83.25    83.31    83.78    83.79    86.73    86.96     87.08     87.11     87.18     87.84
>
```

Figure 22: Regression coefficients $\underline{\theta}_1$ for the transformed explanatory variables and cumulative proportion of variation explained by PCs.

From looking at Figure 19, we can see that if we used the first 3 PCs then 70.03% of the variation in our explanatory variables would be explained and 83.07% of the variation in our response would be explained. If we used all PCs then we would have the same R^2 as our least squares shown in Figure 4, which also contains all the explanatory variables.

To reduce the number of dimensions, we need to choose a number, m, of PCs such that $m < 13$. In order to achieve this, we use cross-validation with 10 folds, which can be seen on the program list previously mentioned. We can then extract the MSE of each number of PCs and obtain the following:

```
> MSEP(pcr_cv_fit)
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV           4.684    1.159    1.006   0.8078   0.7943   0.8076   0.7874    0.790   0.6526   0.6457    0.6372    0.6456    0.6471    0.6157
adjCV        4.684    1.158    1.004   0.8058   0.7919   0.8067   0.7859    0.789   0.6507   0.6439    0.6335    0.6428    0.6446    0.6130
>
```

Figure 23: Mean Squared Errors of each number of PCs.

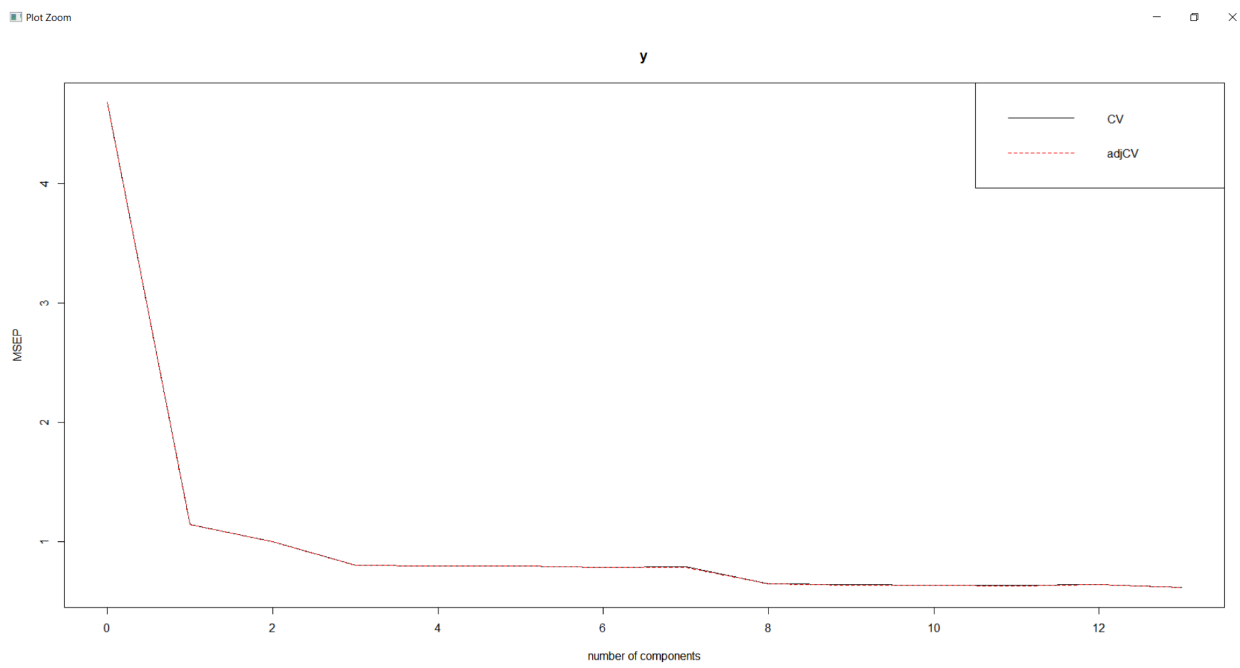

Figure 24: Plot of the MSE of each number of PCs.

From looking at the results, we can see that there is an elbow at three components; can be seen more clearly on figure 21. This suggests that once the first three PCs are used as regressors, little is gained by adding more. So we select $m = 3$, giving a model which uses the first three PCs as our transformed explanatory variables. The MSE for this model was 0.8078.

We then obtain the following regression coefficients in our model:

$$\begin{aligned} lcrim = & -0.82942719 - 0.08868202 \times zn + 0.28685393 \times indus - 0.02487406 \times chas \\ & + 0.31989741 \times nox + 0.11608856 \times rm - 0.23197528 \times age - 0.28370905 \times disf \\ & + 0.42112783 \times rad + 0.40998626 \times tax + 0.08589480 \times ptratio - 0.33583298 \times black \\ & + 0.11523748 \times lstat - 0.02937981 \times medv \end{aligned}$$

In this model, the coefficients for *zn, chas, ptratio* and *medv* are very close to 0, and so do not have a significant impact on the response. This model can be used to predict the value of *lcrim* by simply inputting the values of the explanatory variables.

The issue with PCR is that there is no guarantee that the directions in which our explanatory variables are most effective in explaining the variation in our response.

6.1.1 Out-of-Sample Validation

```
> #Out of sample for PCR
> yhat_test = predict(pcr_cv_fit, boston_data_test, ncomp = 3)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.8750313
> 
```

Figure 25: Performing out-of-sample validation using the validation set approach.

6.2 Partial Least Squares

We can use the PLS method which considers the direction that explains the variation in the response variable as well as the explanatory variables. The program list to carry out PLS can be found in Appendix B - Figure B.13

The components themselves have now changed, for example the first PC is now the negative of the first using the PCR method, however the amount of the variation explained in X has now decreased with the first 3 components explaining 65.99% compared to over 70% in the previous section.

```
> #print the directions
> (C = unclass(plsr_load))
            Comp 1       Comp 2      Comp 3      Comp 4      Comp 5       Comp 6        Comp 7       Comp 8       Comp 9      Comp 10
zn      -0.257120120  0.180609816  0.20508886 -0.50563069 -0.27375767  0.2445502848  0.2714242919  0.009533166 -0.541452967  0.380061440
indus    0.350161292 -0.009303529 -0.30835773 -0.14129375 -0.28885563  0.1044527860  0.2410593959  0.176539591  0.024283537 -0.357787160
chas    -0.004340805  0.144413128 -0.34072703  0.33868548 -0.61415324  1.0981936642 -0.8369225937  0.414248376 -0.212095886  0.055343737
nox      0.345433515  0.147329014 -0.21689810  0.22533912 -0.08229227 -0.1316281880  0.1904076097  0.071626667 -0.224887012 -0.530110460
rm      -0.196418770  0.612694474 -0.38459225 -0.07836483  0.08196885 -0.1459023823  0.0098546428  0.002660523  0.320475202  0.168070716
age     -0.312946557 -0.017043047  0.44883489 -0.27827799  0.07006716  0.1809822662 -0.1142269851 -0.047069816 -0.079661459 -0.583363065
disf    -0.314789479 -0.116572048  0.53181394 -0.25687870 -0.07261154 -0.0008210674 -0.1918067080  0.479630452  0.004596605 -0.212092262
rad      0.319317986  0.354226030  0.53770646 -0.10166409 -0.02123736  0.3064252984  0.0223853027  0.061646853  0.106293037  0.111039615
tax      0.338723979  0.265227035  0.36961174 -0.27076092 -0.33054019 -0.2694362474 -0.0777121570 -0.085923884  0.006372292  0.016755241
ptratio  0.208531492 -0.288632649  0.25970356 -0.42986284 -0.04164962  0.6205397205 -0.2365749810 -0.624904206  0.578092924 -0.130532879
black   -0.206681903 -0.186138826  0.08097427  0.44903278 -0.93026864  0.0409190314  0.4283913542 -0.344658459  0.288557922  0.009092775
lstat    0.313888314 -0.309461158  0.15828416  0.09131733  0.04630864  0.0572512602  0.0004398792 -0.107131576 -0.340866380  0.410435382
medv    -0.266390539  0.499642180 -0.30962880  0.23380366  0.02886223  0.1548001212  0.0799810940 -0.587817567  0.080749363 -0.184636777
           Comp 11      Comp 12     Comp 13
zn      -0.17911268  0.164830581 -0.15650714
indus    0.30265622 -0.488489538  0.18994939
chas     0.01705804  0.042302713  0.02271391
nox     -0.25626487  0.411268626 -0.12810792
rm      -0.42193639  0.142154436  0.48587288
age      0.45321227  0.124176919  0.55833049
disf    -0.39186457 -0.378586204 -0.17875437
rad      0.03945755  0.006031335 -0.01492884
tax      0.13913413 -0.086677733 -0.03651063
ptratio -0.22295191  0.135012767 -0.10578711
black   -0.06341903  0.083621756  0.12094523
lstat   -0.44828747 -0.314203145  0.53029923
medv     0.05054888 -0.504311943 -0.18478863
attr(,"explvar")
   Comp 1    Comp 2    Comp 3    Comp 4    Comp 5    Comp 6    Comp 7    Comp 8    Comp 9   Comp 10   Comp 11   Comp 12   Comp 13
47.481415 11.080941  7.425291  6.464938  4.988110  3.122204  4.297328  2.079608  5.712757  2.127087  2.222128  1.490025  1.508167
>
```

Figure 26: Components and the variation explained

```
> summary(plsr_fit)
Data:   X dimension: 400 13
        Y dimension: 400 1
Fit method: kernelpls
Number of components considered: 13
TRAINING: % variance explained
         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
X          47.48    58.56    65.99    72.45    77.44    80.56    84.86    86.94    92.65     94.78     97.00     98.49    100.00
y_train    79.14    85.17    86.71    87.42    87.57    87.74    87.81    87.83    87.84     87.84     87.84     87.84     87.84
>
```

Figure 27: Variation explained in response ($\underline{y}$) and explanatory (X)

The big improvement with this method is the proportion of the response variable that is explained. This is now at 86.71% for 3 components compared to 83% for PCR. Hence, this means that the PLS method allows you to use fewer components, reducing the dimension, for greater variation explained.

```
> #corresponding MSE values
> MSEP(plsr_cv_fit)
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV           4.684   0.9967   0.7225   0.6689   0.6333   0.6269   0.6253   0.6216   0.6167   0.6167    0.6158    0.6156    0.6157    0.6157
adjCV        4.684   0.9956   0.7211   0.6666   0.6308   0.6244   0.6223   0.6185   0.6139   0.6140    0.6131    0.6130    0.6130    0.6130
>
```

Figure 28: MSE for each number of PCs

In order to choose the number of PCs to use for this model we will use cross validation, as before, to find the optimum value. Using these values and the plot below we can see that the MSE stabilises after 3 PCs and therefore these will be used in our model.

This gives a value for the MSE of 0.6689 however this cannot be compared directly to the value for PCR earlier because we did not use the same subsets of variables during cross validation.

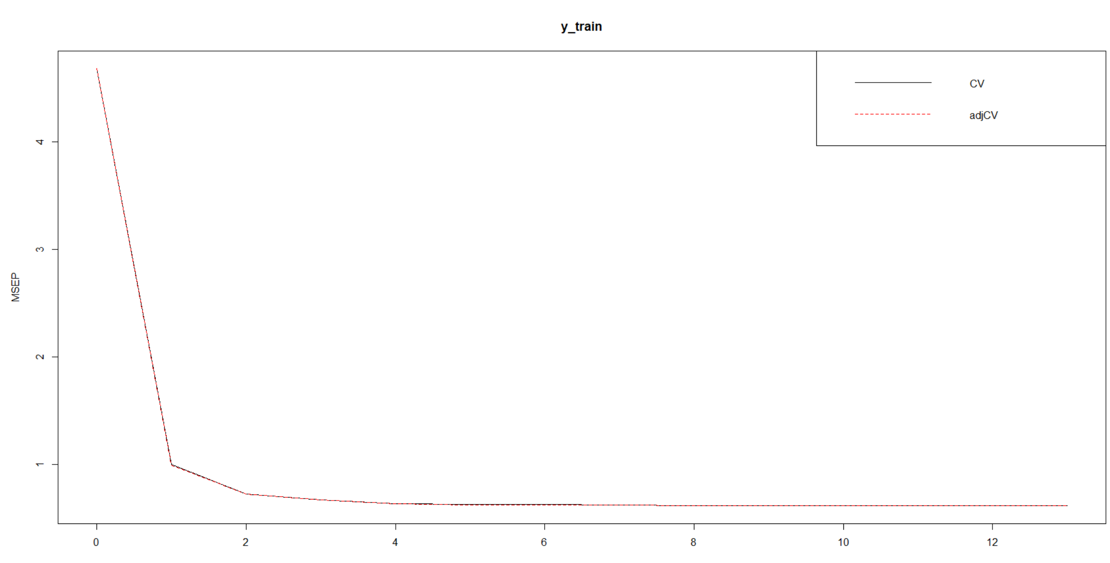

Figure 29: Plot of component number against the MSE

The following model is therefore produced:

$$\begin{aligned} lcrim = &- 0.82942719 - 0.190529803 \times zn + 0.051757064 \times indus - 0.002752944 \times chas \\ &+ 0.356237206 \times nox + 0.005132704 \times rm - 0.125400241 \times age - 0.163561447 \times disf \\ &+ 0.829790998 \times rad + 0.480158534 \times tax - 0.034835706 \times ptratio - 0.163207837 \times black \\ &+ 0.186364905 \times lstat - 0.046128195 \times medv \end{aligned}$$

In this model, the coefficients for *chas, rm, ptratio* and *medv* are very close to 0, and so do not have a significant impact on the response. This model is quite similar in terms of effect of explanatory variables, i.e. *chas* makes little difference to response in both models, but the coefficients are significantly different.

6.2.1 Out-of-Sample Validation

```
> #Out of sample for PLS
> yhat_test = predict(plsr_cv_fit, boston_data_test, ncomp = 2)
> test_error = mean((boston_data_test$y_test - yhat_test)^2)
> test_error
[1] 0.8041665
> 
```

Figure 30: Performing out-of-sample validation using the validation set approach.

7 Final Model

Now that we have produced a range of models using various methods, we now need to decide on the best option for a final model to explain the natural logarithm of the per capita crime rate in the Boston area.
To decide on this, we can use the simplicity of the model, as well as, the MSE value obtained by cross validation from each method (even though some of these can't be compared directly due to the different fold index used for the dimension reduction methods) and the validation set approach using the remaining data that was not used in the sample.
Looking at the out-of-sample validation for the best subset selection method we see that the prediction is worse than the least squares model, with the least squares model having a test error of 0.6838521 and the best subset model having a slightly higher 0.6977663. Looking at the ridge regression and LASSO model we see that prediction is about the same as the least squares model, although the LASSO model is marginally better, with the test error being 0.6842727 and 0.6745852 respectively. The PCR and PLS model performed significantly worse than the least squares model with test errors of 0.8750313 and 0.8041665.

Out of all the model options we chose the LASSO model, explained in detail in Section 5.2, as this has the smallest MSE value. It also has the lowest has out-of-sample validation test error with a value of 0.6745852. The method is useful for both variable selection, shrinkage and prediction. Therefore our final model is:

$$\begin{aligned} lcrim = &- 0.82942719 - 0.26061455 \times zn + 0.06263766 \times indus \\ &+ 0.38281368 \times nox - 0.04910525 \times rm - 0.14073802 \times age - 0.19326884 \times disf \\ &+ 1.23975952 \times rad - 0.11734337 \times ptratio + 0.12016095 \times black \\ &+ 0.19561037 \times lstat + 0.0.05524359 \times medv \end{aligned}$$

This model has removed *chas* and *tax* (which were the first to be removed under subset selection). Using this we can see that the crime rate increases when: *indus*, *nox*, *rad*, *black*, *lstat* and *medv* increase and/or *zn*, *rm*, *disf* and *ptratio* decrease. We see that the most significant variable is *rad* showing us that the index of accessibility to radial highways has the most impact on per capita crime rate in the Boston area

Appendix A

Variables

- ***lcrim*** (y) - per capita crime rate by town.
- ***zn*** - proportion of residential land zoned for lots over 25,000 sq.ft.
- ***indus*** - proportion of non-retail business acres per town.
- ***chas*** - Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
- ***nox*** - nitrogen oxides concentration (parts per 10 million).
- ***rm*** - average number of rooms per dwelling.
- ***age*** - proportion of owner-occupied units built prior to 1940.
- ***dis*** - weighted mean of distances to five Boston employment centres.
- ***rad*** - index of accessibility to radial highways.
- ***tax*** - full-value property-tax rate per $10,000.
- ***ptratio*** - pupil-teacher ratio by town.
- ***black*** - $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town.
- ***lstat*** - lower status of the population (percent).
- ***medv*** - median value of owner-occupied homes in $1000s

Appendix B

Additional

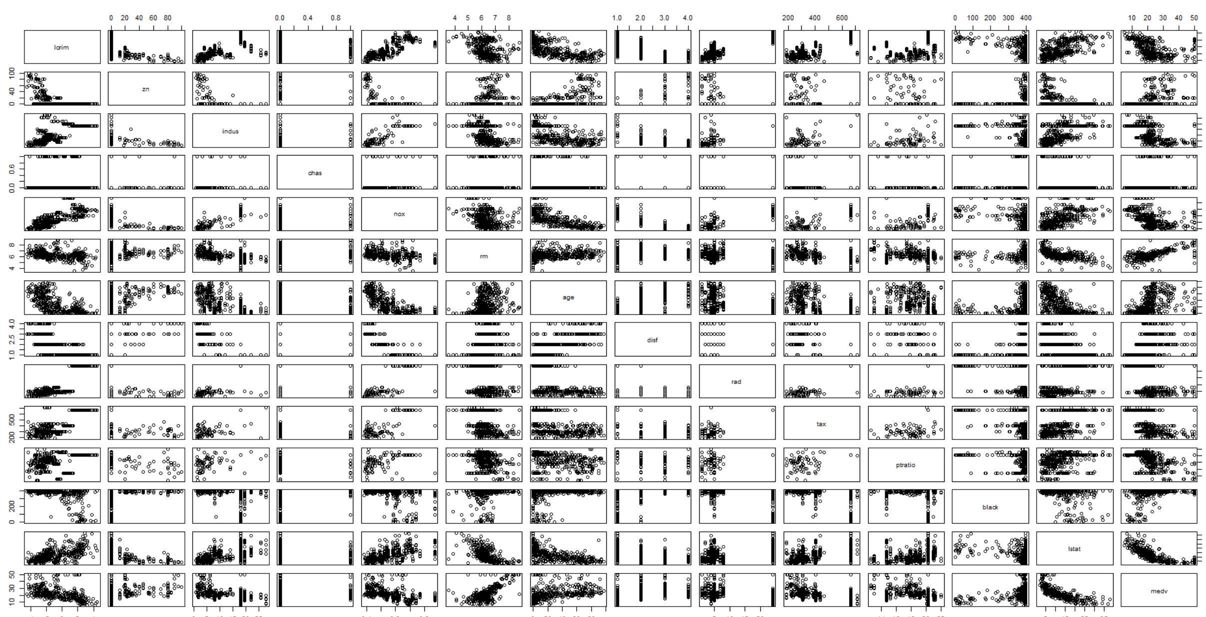

Figure B.1: Scatter plot matrix

```
summary(fit1)

#Best subset selection; using least squares
bss = regsubsets(y~.,data = boston_data,method = "exhaustive",nvmax=13)
bss_summary = summary(bss)
bss_summary
names(bss_summary)
bss_summary$adjr2 #Adjr2 values
bss_summary$cp     #Cp statistic of each component
bss_summary$bic    #BIC values

best_adjr2 = which.max(bss_summary$adjr2) #Need highest adjR^2 value; 9 components
best_cp = which.min(bss_summary$cp)     #Smallest Cp and BIC; 8 components
best_bic = which.min(bss_summary$bic)    #7 components

best_adjr2 #Components needed for highest adjR2
best_cp    #Components needed for lowest CP
best_bic   #lowest BIC

par(mfrow=c(1,3))
plot(1:13,bss_summary$adjr2, xlab="Number of predictors",ylab="Adjusted R_squared",type="b")
points(best_adjr2,bss_summary$adjr2[best_adjr2],col="red",pch=16)
plot(1:13,bss_summary$cp, xlab="Number of predictors",ylab="Cp",type="b")
points(best_cp,bss_summary$cp[best_cp],col="red",pch=16)
plot(1:13,bss_summary$bic, xlab="Number of predictors",ylab="BIC",type="b")
points(best_bic,bss_summary$bic[best_bic],col="red",pch=16)
#Very little difference between M7 to M13 for adjr2 and cp
#choose 7 predictors
coef(bss,7)
```

Figure B.2: Program list for Best Subset Selection

```r
## cross-validation
# 10-fold cross validation
nfolds = 10
(n = nrow(boston_data_train))
(p = ncol(boston_data_train) - 1)
# Sample fold-assignment index
fold_index = sample(nfolds, n, replace=TRUE)
fold_sizes = numeric(nfolds)
for(k in 1:nfolds) fold_sizes[k] = length(which(fold_index==k))
fold_sizes
#important to keep the same!!!!

cv_bss_errors = matrix(NA, p, nfolds)

for(k in 1:nfolds) {
  # Fit models M_1,...,M_p by best-subset selection using all but the k-th fold
  bss_tmp_fit = regsubsets(y_train ~ ., data=boston_data_train[fold_index!=k,], method="exhaustive", nvmax=p)
# For each model M_m where m=1,...,p:
for(m in 1:p) {
  # Compute fitted values for the k-th fold
  bss_tmp_predict = predict(bss_tmp_fit, boston_data_train[fold_index==k,], m)
  # Work out MSE for the k-th fold
  cv_bss_errors[m, k] = mean((boston_data_train[fold_index==k,]$y_train - bss_tmp_predict)^2)
  }
}
cv_bss_errors

bss_mse = numeric(p)
# For models M_1,...,M_p:
for(m in 1:p) {
bss_mse[m] = weighted.mean(cv_bss_errors[m,], w=fold_sizes)
}
bss_mse

(best_cv = which.min(bss_mse))

## Create multi-panel plotting device
par(mfrow=c(2,2))
## Produce plots, highlighting optimal value of k
plot(1:p, bss_summary$adjr2, xlab="Number of predictors", ylab="Adjusted Rsq", type="b")
points(best_adjr2, bss_summary$adjr2[best_adjr2],col="red", pch=16)
plot(1:p, bss_summary$cp, xlab="Number of predictors", ylab="Cp", type="b")
points(best_cp, bss_summary$cp[best_cp], col="red", pch=16)
plot(1:p, bss_summary$bic, xlab="Number of predictors", ylab="BIC", type="b")
points(best_bic, bss_summary$bic[best_bic], col="red", pch=16)
plot(1:p, bss_mse, xlab="Number of predictors", ylab="10-fold CV Error", type="b")
points(best_cv, bss_mse[best_cv], col="red", pch=16)
```

Figure B.3: Program list for Best Subset Selection cross-validation

```r
library("glmnet")

set.seed(7045607)
sampid = sample(dim(Boston)[1],400)
BostonNew = Boston[sampid,]

X1_raw = as.matrix(BostonNew[,2:14])
x1 = scale(X1_raw) # Standardised explanantory variables
y = BostonNew$lcrim # response variable
boston_data = data.frame(y,x1) #Matrix of response and explanatory variables

grid = 10^seq(5,-3,length=100) #Grid of values for the tuning parameter
ridge_fit = glmnet(x1,y,alpha=0,standardize = FALSE, lambda = grid) #alpha=0 means we are doing ridge reg
#Forcing the function to use values lambda=10^5 (lots of shrinkage) to lambda=10^(-3) (very little shrinkage).

#Ridge regression coefficients
beta1_hat = coef(ridge_fit)

#Some examples of coefficients for different levels of shrinkage
beta1_hat[,1] #Lots of shrinkage
beta1_hat[,75] #Some shrinkage
beta1_hat[,100] #Very little shrinkage. Very similar to that of least squares estimate

par(mfrow=c(1,1))
#plot to show how estimated coefficients vary with lambda
plot(ridge_fit,xvar="lambda",col=1:13,label=TRUE) #number at top are number of coeff that are non-zero.

#Using cross-validation to choose appropriate value for tuning parameter
ridge_cv_fit = cv.glmnet(x1,y,alpha=0,standardize = FALSE,lambda=grid, foldid = fold_index) #k=10 folds with the same index
plot(ridge_cv_fit) #mean MSE plotted with error bars which covers the mean +- 1 Standard error

#Extracting the value of lambda which corresponds to the minimum
(lambda_min = ridge_cv_fit$lambda.min)

#The tuning parameter which is the minimum.
(i=which(ridge_cv_fit$lambda == ridge_cv_fit$lambda.min))

#Corresponding mean MSE
ridge_cv_fit$cvm[i]
#Regression coefficients where s argument is used for tuning parameter
coef(ridge_fit, s=lambda_min)
predict(ridge_fit, newx = as.matrix(x1), s=lambda_min)
```

Figure B.4: Program list for Ridge Regression

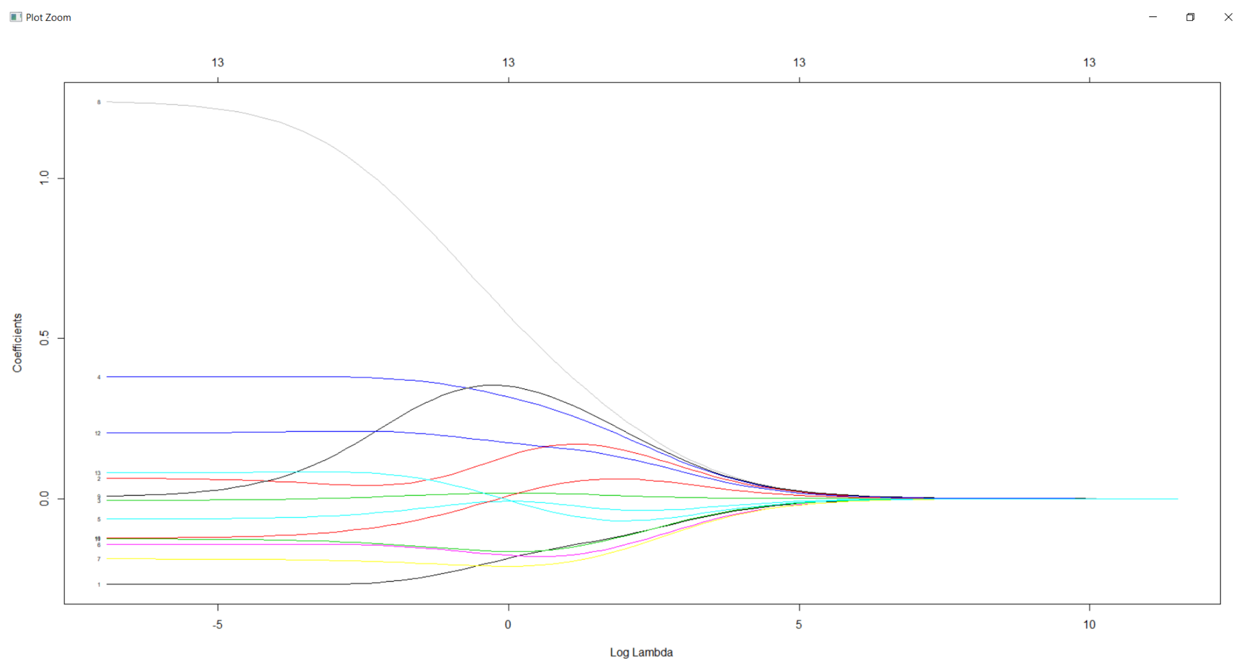

Figure B.5: Plot of the estimated ridge regression coefficients with varying $\hat{\lambda}$

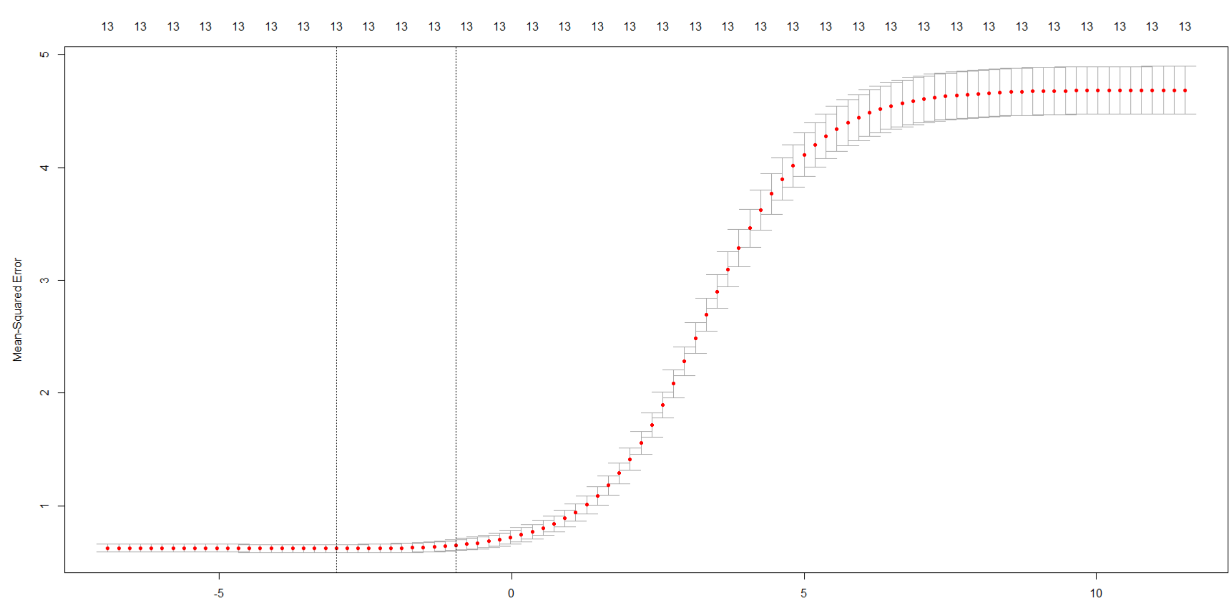

Figure B.6: Cross-validation scores for our sample of the Boston data, using ridge regression

```
> predict(ridge_fit, newx = as.matrix(x1), s=lambda_min)
            1
410  2.3581811
52  -2.4282134
87  -2.1371956
376  1.9339514
493 -0.6833196
472  1.3447380
71  -2.6556972
4   -2.6221071
30  -1.6257229
164 -0.7231525
499 -0.9023909
56  -3.7917516
158 -0.5980470
140 -0.7331431
422  2.3502017
372  2.1404216
267 -0.7010826
202 -3.4531400
90  -2.2379571
234 -1.2448997
252 -2.5300308
381  1.9994734
162 -0.6965231
262 -0.8319441
226 -1.1983584
374  2.7308056
43  -2.6808657
247 -2.4446429
421  2.3464703
221 -1.0936840
231 -1.0771226
93  -2.2926970
76  -1.8913044
478  2.2783360
433  2.0183400
201 -3.9827382
19  -1.6649506
112 -1.1724772
126 -1.2733612
344 -2.6107115
446  3.0180786
463  1.9208379
480  1.8492188
246 -2.0074114
338 -1.9836179
299 -3.3333152
70  -2.6111818
```

```
33  -0.9872337
175 -1.5118614
400  2.5095958
79  -1.9599667
465  1.6452313
34  -1.3638391
98  -2.3593631
170 -0.4892896
271 -2.3170417
380  2.2339098
260 -0.8562579
292 -3.5835642
132 -0.9118761
345 -2.7638532
454  2.2242161
89  -2.1676318
194 -3.6742391
471  1.5725267
348 -3.6443038
403  2.2923004
504 -1.8779079
273 -2.4199192
436  2.8973640
192 -2.6538636
129 -0.8377784
467  2.4164478
409  2.3426148
159 -0.5989522
36  -1.7191912
386  2.6238887
11  -1.3741276
78  -1.8481731
165 -0.5482916
99  -2.5053135
364  2.4488136
184 -1.9371582
481  1.1786706
265 -0.8295436
55  -3.2382890
102 -1.6814570
392  2.3041770
385  2.8351158
40  -3.5859239
317 -1.2909457
432  2.3142984
274 -2.4746682
169 -0.4221236
8   -1.2866201
31  -1.2515587
```

```
475  1.9757167
183 -1.9735461
82  -2.6394810
417  2.8198854
346 -2.5977618
280 -2.2257063
127 -0.8584402
86  -2.2717848
305 -2.1048578
363  2.3823749
303 -2.3154006
337  2.0649062
501 -0.8284244
404  2.2740298
434  2.6080501
240 -2.4232638
329 -2.1588485
155  0.4853180
420  2.7514542
177  1.6136903
188 -2.3317420
151  0.4233811
448  2.3144387
73  -2.6801692
320 -1.5718701
411  2.3170598
351 -3.5815304
27  -1.4952068
152  0.4990392
370  1.9074467
26  -1.3662104
466  1.7004400
215 -1.3853595
128 -0.7508415
459  2.1366793
340 -1.8757781
10  -1.4414874
146  1.0652919
15  -1.6691099
291 -3.6445259
479  1.9846751
336 -2.1752341
423  2.0532743
138 -0.8685269
186 -1.8342133
355 -3.8098886
157  0.8834355
302 -2.2449638
235 -1.1676388
```

Figure B.7: Large sample of our predicted values for Ridge regression

```r
library(glmnet)

load('Boston.RData')

## Set the seed using one of your group members login ID
set.seed(7045607)
# Randomly sample 400 rows from the 506 rows in the full data
sampid = sample(dim(Boston)[1], 400)
traindata = Boston[sampid, ]
testdata = Boston[-sampid, ]

#set the response variable
y_train = traindata$lcrim
y_test = testdata$lcrim

#create matrix with the rest of the data
X1_raw_train = traindata[,2:14]
X1_raw_train = as.matrix(X1_raw_train)
X1_train = scale(X1_raw_train)

X1_raw_test = testdata[,2:14]
X1_raw_test = as.matrix(X1_raw_test)
X1_test = scale(X1_raw_test)

#data frame with response and data
boston_data_train = data.frame(y_train, X1_train)
boston_data_test = data.frame(y_test, X1_test)

grid = 10^seq(5, -3, length=100)
lasso_fit = glmnet(X1_train, y_train, alpha = 1, standardize = F, lambda = grid)

beta1_hat = coef(lasso_fit)

grid[1]
beta1_hat[,1]
grid[75]
beta1_hat[,75]
grid[100]
beta1_hat[,100]

plot(lasso_fit, xvar="lambda", col=1:13, label=T)

lasso_cv_fit = cv.glmnet(X1_train, y_train, alpha=1, standardize=F, lambda = grid, foldid = fold_index)
plot(lasso_cv_fit)

(lambda_min = lasso_cv_fit$lambda.min)
(i = which(lasso_cv_fit$lambda == lasso_cv_fit$lambda.min))
lasso_cv_fit$cvm[i]

coef(lasso_fit, s=lambda_min)
```

Figure B.8: Program list for LASSO.

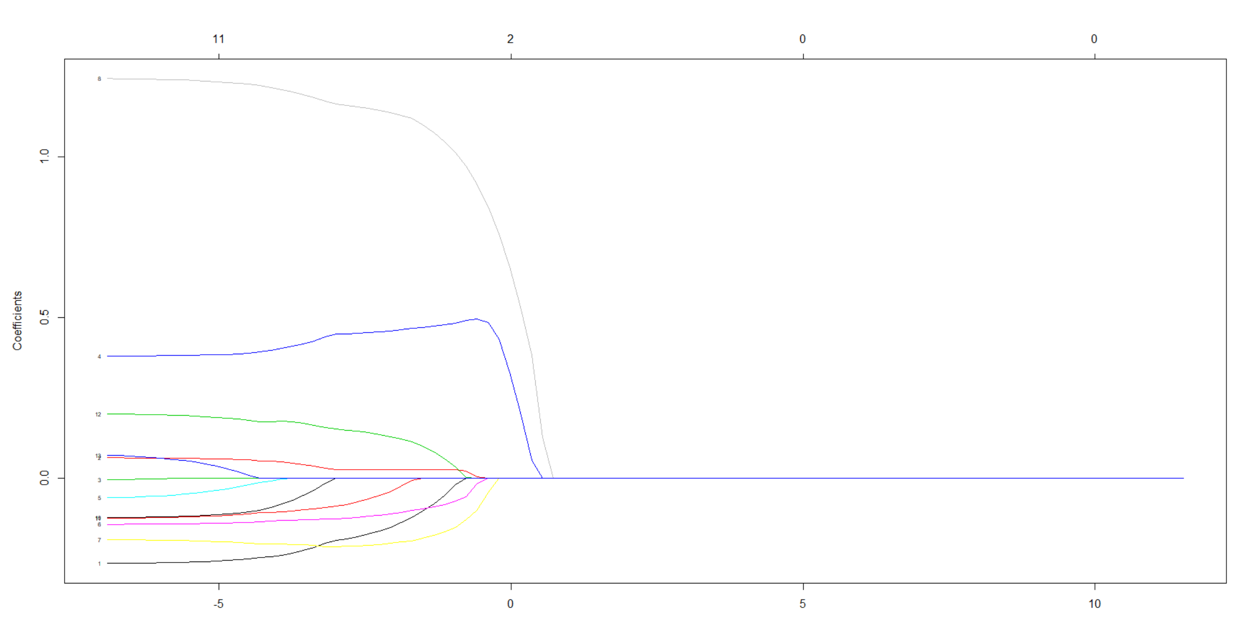

Figure B.9: Plot of the estimated LASSO regression coefficients with varying $\hat{\lambda}$

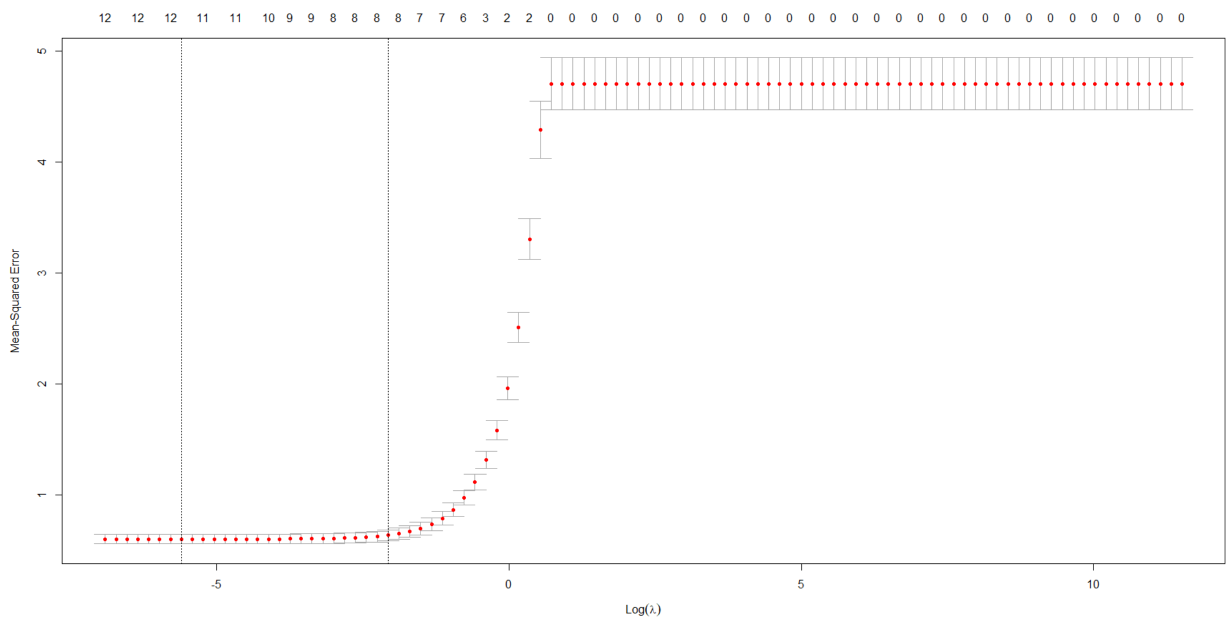

Figure B.10: Cross-validation scores for our sample of the Boston data, using LASSO.

```
install.packages("pls")
library("pls")

set.seed(7045607)
sampid = sample(dim(Boston)[1],400)
BostonNew = Boston[sampid,]

X1_raw = as.matrix(BostonNew[,2:14])
x1 = scale(X1_raw) # Standardised explanantory variables
y = BostonNew$lcrim # response variable
boston_data = data.frame(y,x1) #Matrix of response and explanatory variables

fit = lm(y~.,data = boston_data)
summary(fit) #Some explanatory variables are not significant when considered individually
#Do not need all variables.

#Principal Components regression
pcr_fit = pcr(y~.,data=boston_data,scale = FALSE)
pcr_load = loadings(pcr_fit)
c = unclass(pcr_load)
c #attr - proportion of variation of the standardised explanatory variables X1 that is explained by each of the PCs.

pcr_yload = Yloadings(pcr_fit)
theta1_hat = unclass(pcr_yload)
theta1_hat #comp 4,5,7,11 are close to 0 which suggests that this is a dimension in which X1 shows variation not strongly associated
           #with the response.
summary(pcr_fit) #Cumulative proportion of variation in X1 explained by different numbers of PCs

pcr_cv_fit = pcr(y~.,data=boston_data,scale=FALSE,validation="CV")
MSEP(pcr_cv_fit)
plot(pcr_cv_fit, plottype = "validation", legend = "topright",val.type="MSEP") #Elbow at three components

coef(pcr_fit, intercept = TRUE, ncomp=3)
predict(pcr_fit,newdata=x1,ncomp=3)
```

Figure B.11: Program list for PCR

```
> coef(pcr_fit, intercept = TRUE, ncomp=3)
, , 3 comps

                      y
(Intercept) -0.82942719
zn          -0.08868202
indus        0.28685393
chas        -0.02487406
nox          0.31989741
rm           0.11608856
age         -0.23197528
disf        -0.28370905
rad          0.42112783
tax          0.40998626
ptratio      0.08589480
black       -0.33583298
lstat        0.11523748
medv        -0.02937981
```

Figure B.12: Regression coefficients for PCR for 3 components

```r
library(pls)
#load dataset
load('Boston.RData')
## Set the seed using one of your group members login ID
set.seed(7045607)
# Randomly sample 400 rows from the 506 rows in the full data
sampid = sample(dim(Boston)[1], 400)
traindata = Boston[sampid, ]
testdata = Boston[-sampid, ]

#set the response variable
y_train = traindata$lcrim
y_test = testdata$lcrim

#create matrix with the rest of the data
X1_raw_train = traindata[,2:14]
X1_raw_train = as.matrix(X1_raw_train)
X1_train = scale(X1_raw_train)

X1_raw_test = testdata[,2:14]
X1_raw_test = as.matrix(X1_raw_test)
X1_test = scale(X1_raw_test)

#data frame with response and data
boston_data_train = data.frame(y_train, X1_train)
boston_data_test = data.frame(y_test, X1_test)

#fit the model using PLS
plsr_fit = plsr(y_train ~ ., data = boston_data_train, scale=FALSE)
#examine the directions defined by the PLS
plsr_load = loadings(plsr_fit)
#print the directions
(C = unclass(plsr_load))

summary(plsr_fit)

#extract the coefficients of the transformed variables
plsr_yload = Yloadings(plsr_fit)
#print the coefficients
(theta1_hat = unclass(plsr_yload))

#fit model, applying 10-fold cross validations
plsr_cv_fit = plsr(y_train ~ ., data = boston_data_train, scale=FALSE, validation="CV")
#plot the cross validation scores
plot(plsr_cv_fit, plottype = "validation", legend = "topright", val.type = "MSEP")

#corresponding MSE values
MSEP(plsr_cv_fit)

coef(plsr_fit, intercept=T, ncomp=3)
```

Figure B.13: Program list for PLS

```
                       y
(Intercept) -0.829427190
zn          -0.190529803
indus        0.051757064
chas        -0.002752944
nox          0.356237206
rm           0.005132704
age         -0.125400241
disf        -0.163561447
rad          0.829790998
tax          0.480158534
ptratio     -0.034835706
black       -0.163207837
lstat        0.186364905
medv         0.046128195
```

Figure B.14: Coefficients for PLS model using 3 PCs

Appendix C

Contribution to Projects

- **Stephen Cole** - Section 1, Section 2 (Code), Section 3 (Code), Section 4 (Code), Section 5 - intro (Write-up), Section 5.1 (Code and Write-up), Section 6 - intro (Write up), Section 6.1 (Code and Write-up).
- **Calum Doran** - Section 1, Section 3.2 (Code and Write up), Section 4.1.2, Section 5.1.1, Section 5.2 (Code and Write up), Section 5.2.1, Section 6.1.1, Section 6.2 (Code), Section 6.2.1, Section 7.
- **Joseph Crone** - Section 3 (Write-up), Section 3.1 (Code and Write-up), Section 4.1.1 (Code and Write-up), Section 7
- **Harry Johnston** - Section 2 (Write-up), Section 6.2 (Write-up), Section 7.