SY19

November 20, 2016

Part I

Abreviations

MSE	Mean Squared Error
RSE	Resiual Standard Error
RSS	Residual
R	
BIC	
LR	Linear Regression
CV	Cross Validation
LOOCV	Leave One Out Cross Validation

Abstract

Part II

Introduction

Part III

Ex 1- Phoneme

0.1 Context

0.2 Dataset Description

In the context of speech recognition the aim is to predict which of the phoneme is the subject pronouncing. We have five phonemes:

$$g = \begin{cases} \textbf{"sh"} \, a \, sin \, "she" \\ \textbf{"dcl"} \, a \, sin \, "dark" \\ \textbf{"iy"} \, as \, the \, vowel \, in \, "she" \\ \textbf{"aa"} \, as \, the \, vowel \, in \, "dark" \\ \textbf{"ao"} \, as \, the \, first \, vowel \, in \, "water" \end{cases}$$

The study was on 4509 speeches pronounced my 50 male speakers.the method used for speech recognition is the Log-periodogram. our data sets is composed of 4509 log-periodograms (observations) of length 256 (features).The column labelled "speaker" identifies the different speakers. We notice that some are labeled train and some test. Hence we have a training set composed of 3340 observations and 1169 of the test. The column g shows the responses. The frequencies of each phoneme for the 4509 speeches are shown in the table below.

Phonemes	aa	ao	dcl	iy	sh
Train	519	759	562	852	648
Test	176	263	195	311	224
Total	695	1022	757	1163	872

Table 1: Frequencies of phonemes in the train and test data set

```
1  data_set = read.csv("phoneme.data.txt")
2  head(data_set)
3  data_set.nb_features = 256
4
5  }
```

Splitting data into train set and test set.

```
1  train= data_set[1:3340,]
2  train.x = train[,-258]
3  train.y = train[,258]
4
5  test = data_set[3341:4509,]
6  test.x = test[,-258]
7  test.y = test[,258]
```

3

0.3 Classification

0.3.1 Logistic Regression

```
1  library("nnet")
2  model.lr = multinom(formula = g ~ . - g - speaker -
       row.names, data=train, MaxNWts=2000, maxit=1000)
```

```
1  model.lr.predicted = predict(model.lr,newdata=test_set)
2  perf = table(test_set$g,model.lr.predicted)
3  perf
4  sum(diag(perf))/dim(test_set)[1]
```

Model Implementation

Model Analysis

0.3.2 LDA- Linear Discriminant Analysis

Model Implementation

The function lda works same was as lm for linear regression. It returns the group means that are the average of each predictor in each class. The coefficients of linear discriminants output are used to form the LDA decision rule. The prior probability is the percentage of the response for each class in the observation.

```
1  library(MASS)
2  model.lda = lda(g ~ . - g - speaker - row.names, data=train)
3  summary(model.lda)
```

```
Prior probabilities of groups:
aa          ao          dcl         iy          sh
0.1553892   0.2272455   0.1682635   0.2550898   0.1940120
```

```
        Length Class  Mode
prior      5    -none- numeric
counts     5    -none- numeric
means   1280    -none- numeric
scaling 1024    -none- numeric
lev        5    -none- character
svd        4    -none- numeric
N          1    -none- numeric
call       3    -none- call
terms      3    terms  call
xlevels    1    -none- list
```
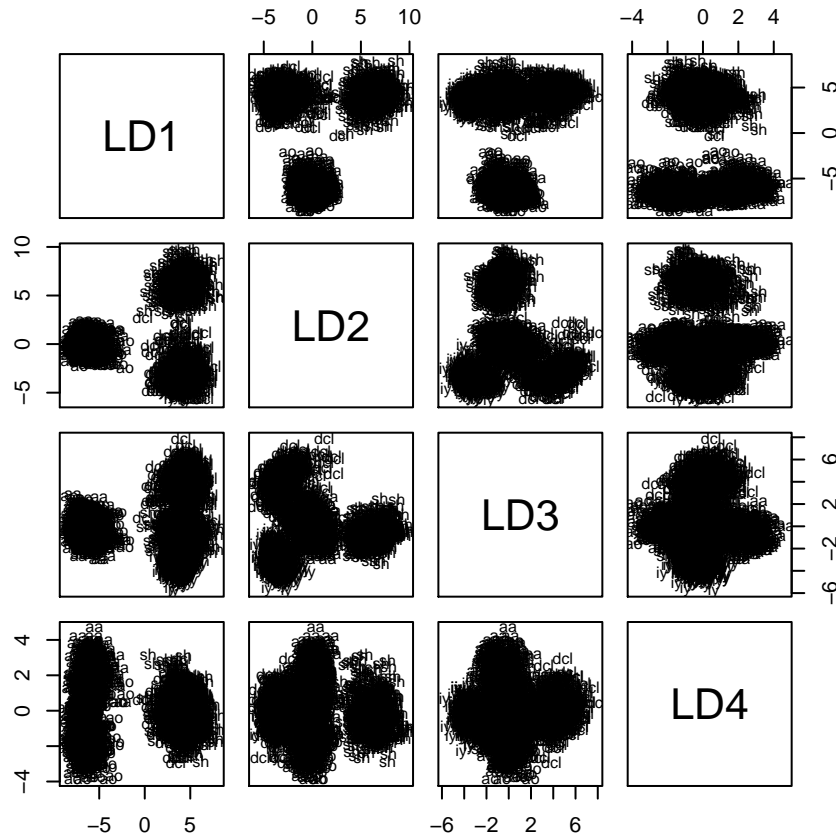
Figure 1: Phoneme LDA model

The prior probabilities tell us that 15% of phonemes are aa , 23% ao, 17% dcl, 25% iy and 19% sh.

Now that we trained the model we can call the predict function to predict the responses for the test set of data. The predict() function returns an element "class", a list of the predicted phonemes. An element "posterior" witch the posterior probability of the response of the kth class. Last the linear discriminants are found in "x".

```
1 model.lda.predicted = predict(model.lda,newdata=test_set)
2 perf = table(test_set$g,model.lda.predicted$class)
3 perf
4 sum(diag(perf))/dim(test_set)[1]
```

The table perf of the predicted responses is shown below:

```
aa   ao dcl  iy  sh
aa  129  47   0   0   0
ao   39 223   0   1   0
dcl   0   0 190   5   0
iy    0   0   2 309   0
sh    0   0   0   0 224
```

```
0.919589392643285
```

Here is an example of how this table is read, for the second line when the phoneme pronounced is aa , the speech recognition detects that 129 is detected as aa and 47 as ao, in other words only 27% is misclassified. In order to compute the total error rate we divide the sum of the diagonal terms (which are the true phonemes detected) and divide it by the number total of observations We find that in 92% of the cases the speech recognition detected right the phoneme.

The ROCR package can be used to produce ROC curves

```
1  library(pROC)
2  l   = length(model.lda.predicted$class)
3  ld1 = model.lda.predicted$x[1:l]
4  ld2 = model.lda.predicted$x[1+l:1+2*l]
5  ld3 = model.lda.predicted$x[2*l:3*l]
6  ld4 = model.lda.predicted$x[3*l:4*l]
7  roc_curve<-roc(test_set$g, as.vector(ld1))
8  plot(roc_curve)
```
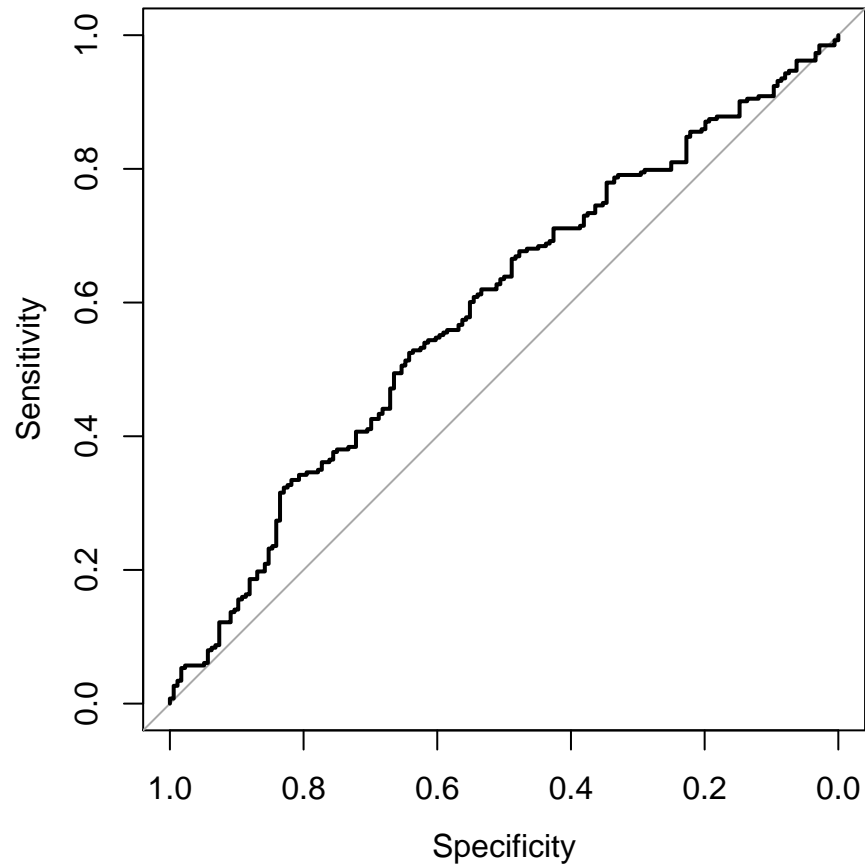
Figure 2: LDA ROC: Area under the curve: 0.588

(AUC) The area under the (ROC) curve is a criteria to judge the performance of the LDA. For this data the AUC is 0.92, which is considered very good as it is close to the maximum.

Model Analysis

0.3.3 QDA- Quadratic Discriminant Analysis

Model Implementation

Model Analysis

0.4 Models Comparison

7

Part IV

Ex 2 - Breast Cancer Recurring Time

0.5 Context

This part aims to build the best model to predict the recurring time of breast cancer based on about 30 features computed from a breast mass. This regression problem will take advantage of a given dataset describing about 200 patient cases.

0.6 Dataset Description

We take a look at the raw dataset to get first hints on how each feature contributes to the recurring time.

The dataset comprises 194 patient cases, each of which is described through 32 features and the cancer recurring time `Time` that we have to predict.

0.6.1 Time

Let's first describe the evolution of the variable `Time`. To do so, we can use the R functions `boxplot` and `hist`.

0.6.2 Features Description

Each patient is represented with a set of 32 features extracted and computed from a digitized image of a breast mass.

The data description we were given does not specify the units, but we do not need them for the following analysis.

Here are the 32 features we are provided with:

- Lymph Node Status

- Mean of

    - radius
    - texture
    - perimeter
    - area
    - smoothness
    - compactness
    - concavity
    - concave points
    - symmetry
    - fractal dimension

- Standard Error of

    - radius

- texture
- perimeter
- area
- smoothness
- compactness
- concavity
- concave points
- symmetry
- fractal dimension

- Mean of the three largest values (also called "Worst") of

  - radius
  - texture
  - perimeter
  - area
  - smoothness
  - compactness
  - concavity
  - concave points
  - symmetry
  - fractal dimension

- Tumor Size

Feature Correlation

Based on the definition of the parameters described above, we already know that many features are correlated :

- The mean of each parameters is smaller than the "worst" value;

- The radius, the perimeter and the area might be linked together;

- The compactness can be computed with the perimeter and the area thanks to the given formula : $Compactness = \frac{perimeter^2}{area-1}$

0.6.3   Data Relevance

We should first check that every patient is relevant to our study, in other words, that there is no abnormal observation in the dataset. Cook's Distance is an interesting measure to verify this important criteria, it can be computed after a simple Linear Regression.

Cook's distance aims to study the influence of each observation on the regression coefficient estimates. To do so, this method uses a straight-forward approach that consists in computing the difference between the original coefficient estimates $\hat{\beta}$ and the coefficient estimates without taking into account the ith observation $\hat{\beta}_{-i}$. The difference is then normalized using the number of parameters and the standard deviation estimate.

A value higher than 1 often indicates an outlier that should be removed from the dataset.

In R, we can use the following code to compute and plot the Cook's distance of each observation :

```
linreg = lm(Time ~ ., data=data_set)
cooks.distance(linreg)
```

According to plot (figure 5), no observation is located beyond the critical Cook's boundary of 1. This means that we can potentially use each and every patient case of our dataset to build our regression model.

0.6.4   Relation between "feature" and "time"

In this section, we will take a first look at the relationship between the variable `Time` and the features used to describe a patient case.

A first way to do it is to separately plot each relationship. An example of such plot is shown in figure 6.

Unfortunately, most plots show very scattered points that do not seem to follow any specific model. The variance is so significant that we cannot even estimate the type of function that links a feature and the variable `Time` together. It could be a simple linear model with a high variance that we may be able to estimate, or more complex models with non-linearities and feature correlations.

To have better clues on the type of model we should be dealing with, we can draw a QQ-Plot that plots the Studentized Residuals with the quantiles.

In R, we can use the library `car` to easily draw the QQ-Plot (figure 7) :

```
library(car)
qqPlot(linreg, main="QQ Plot")
```

The bottom tail of the QQ-Plot seems to deviate from the linear line, which is a sign of the error's non-normality. This may mean that the error does not follow a normal model, or that the model is actually non-linear.

We can also analyze the Residuals-Fitted plot and the Scale-Location plot to get a better understanding on the model. To do, we can simply apply the

function `plot` on the linear model (figure 8).

It turns out that both plots show non-normal scatters of the residuals. In particular, the points displayed in the Residuals-Fitted plot seem to follow a fan shape. This is a sign of a non-constant variance, also called heteroscedasticity.

0.7 Measures to Compare Models

Before building any model, we have to properly define the measures we will later use to compare them.

0.7.1 Some Measures

A first way to assess the performance of a model is to compute the Mean Squared Error (MSE). We can also use adjusted R^2 score.

0.7.2 Data Split

These measures should not be applied on a set whose data was also used to train the model. Indeed, this would include a biais that might distort our conclusions. To cope with this problem, we have to split the dataset into two disjointed sets :

- Training Set : About 75% of the dataset dedicated to the building the model;

- Test Set : The remaining 25% only used at the end to provide some kind of objective measure of the model performance.

Once it is done, we can finally dive in the model building.

0.8 K-nearest neighbors (KNN)

We start our analysis with a very simple model called the KNN. Given a positive integer k and a test observation x0. The KNN model first identifies the k closest points to x0 from the training data. Then estimates

0.8.1 Knn Model

The KNN model in R is done by calling the function reg of the package knn. As we will see in the following sections, For most prediction algorithms, we first have to build the prediction model on the training data, and then use the model to test our predictions. However, the KNN function does both in a single step. In order to find the best k we set a maximum number of neighbors to be considered (in our model it is 20), then we calculate the MSE for each k which is the mean of the squared difference between the real value of Time and the predicted one. All the steps are detailed in the code below.

Model Implementation

```r
library(FNN)
k_max = 120;
MSE = rep(0,k_max)

for( k in 1:k_max)
{
    reg = knn.reg(train=cancer.train.x, test=cancer.test.x,
        y=cancer.train.y, k=k)
    MSE[k] = mean((cancer.test.y - reg$pred)^2)
}

plot(1:k_max, MSE, xlab='k', ylab='MSE', main='MSE against
    k neighbours')
points(x = best_k_test, y = best_k_mse, col = "red", pch =
    16)
abline(h = best_k_mse, col='red')
abline(v = best_k_test, col='red')

best_k_test = which.min(MSE)
best_k_mse = MSE[best_k_test]
```

The graph below shows the MSE plotted against the values of k in a range from 0 to 120. Graphically we notice that a minimum is reached between 10 and 20.

MSE against k neighbours
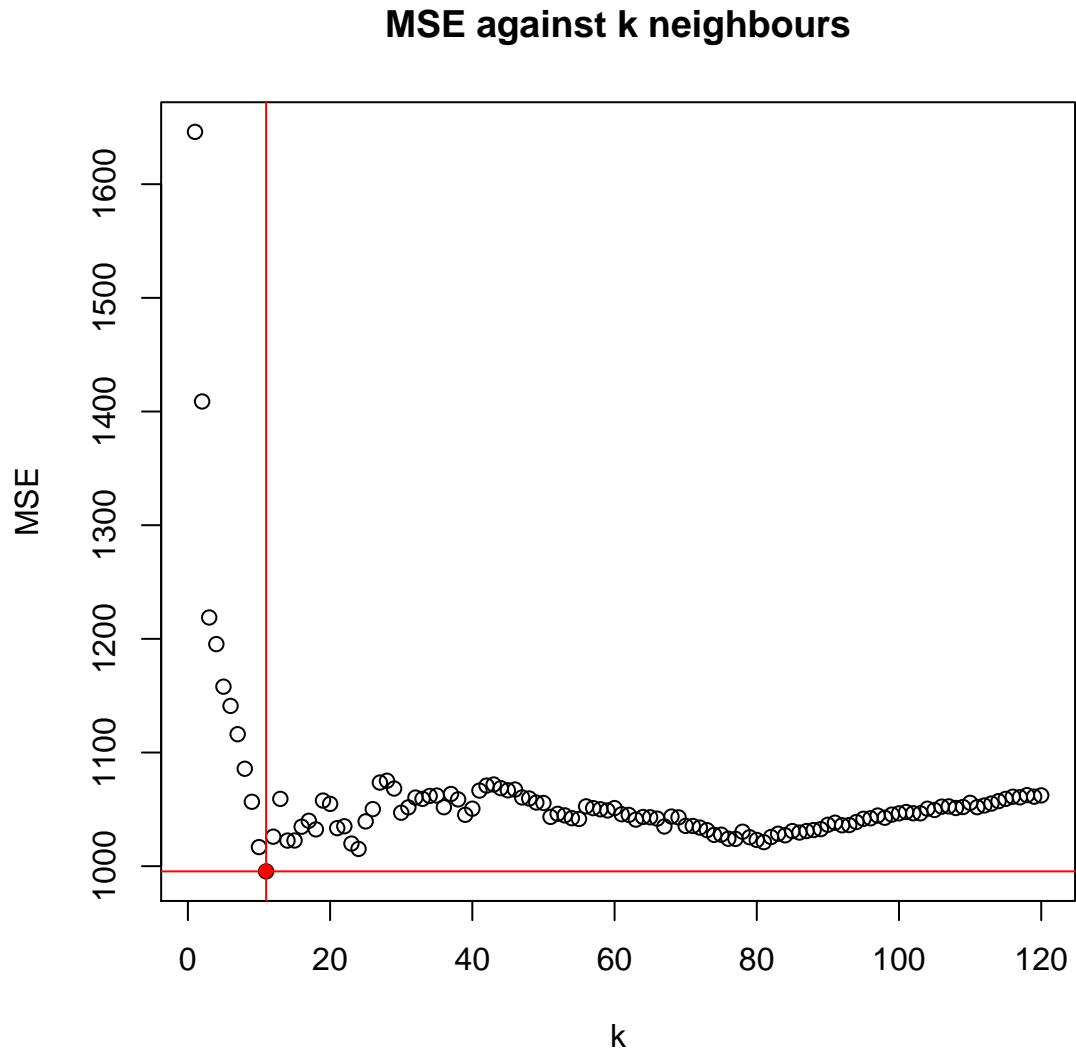
Figure 9: MSE against K neighbours

We use the function **which.min** that returns the index of the minimum MSE value.

best_k_test= 11
best_k_test_MSE = 995.433185

Now that we have the k that minimizes the MSE we call KNN algorithm

14

with this best k and plot the predicted values against the real values. The figure
above shows the result.

```
1  best_reg_test = knn.reg(train= cancer.train.x, test =
       cancer.test.x, y=cancer.train.y, k = best_k_test)
2  plot(cancer.test.y, best_reg_test$pred, xlab='y',
       ylab='y-hat', main='y-hat (Predicted) against y')
3  abline(0,1, col='red')
```

The red line is the function y=x; so further are the points from this line the
further are the predicted values ($\hat{y}$) from the real one (y).

y-hat (Predicted) against y
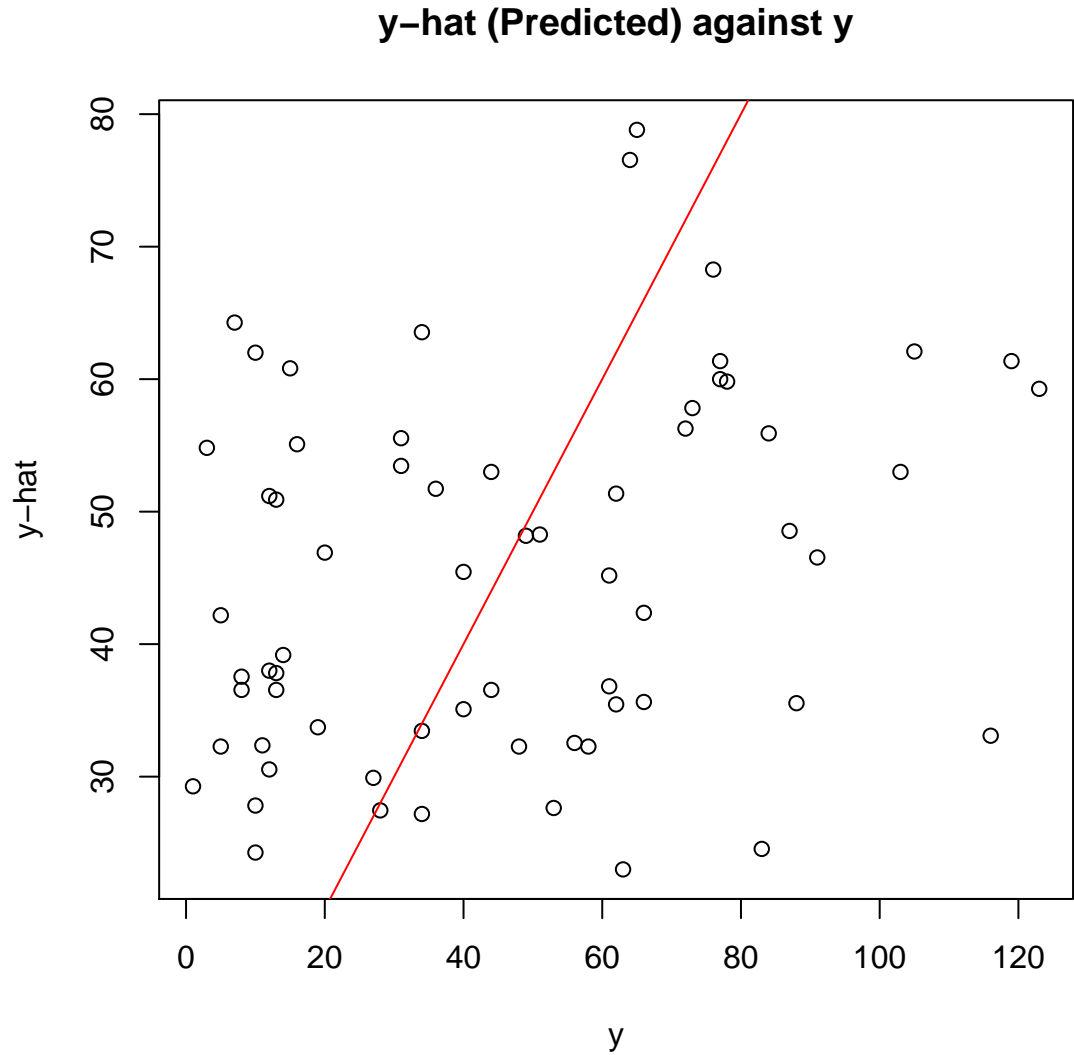
Figure 10: MSE against K neighbours

Model Analysis

WE notice that the predicted $\hat{y}$ diverge a lot the real values y. We expected those results since the MSE=967.386966 which is quite high.

This approach of finding the best k was quite optimistic. Actually we tried finding the best k while minimizing the MSE in **the test data**. Therefore the model is very specific to our test data which yields to a high bias. The solution

is to find the best k among the **training data** and then use the best k in the test data.

To find the best **unbiased** k number of neighbors we use the method of cross validation on the train data.

0.8.2 The Validation Set Approach

There are two methods in cross validation: **cross validation leave one out** and **K-fold cross validation**.In the following parts we will test with both validation methods. As we do not have that much predictors we can afford the computation of cross validation leave one out.

Cross Validation

The cross validation approach is based on dividing the provided data in 2 sets: a training set and a validation set. The model is fit on the training set, then the fitted model is used to predict responses of observations in the validation set. The resulting validation set error rate is assessed using MSE.

Leave-One-Out Cross-Validation

Like the cross validation the LOOCV involves splitting the set of observations in two parts. The test data has a single observation (x1, y1) and the n-1 remaining is used for the training data. The MSE in this case is $MSE_1 = (y_1 - \hat{y})^2$. This provides an unbiased estimate for the test error since it is training the model on approximately all the data of the observation. However it is highly variable as it is based in one observation. The LOOCV repeats the procedure n times fitting each time a different set of observations. The LOOCV test MSE is computed with calculating the average of the n test error estimates.

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} MSE_i$$

The LOOCV will always give the same results in contrast to the CV that depends on the randomness of how the data are split. Furthermore the CV runs the training approach on around the half of the size of the original data while the LOOCV repeats the validation set approach n times using n-1 observations. Hence the LOOCV yields to a not overestimated test error rate compared to the validation. Nevertheless the disadvantaged of the LOOCV is it can be very time consuming when n the number of observations is large.

An alternative to LOOCV that has a smaller computation time is k-Fold Cross Validation. This approach based on dividing the training observations on k groups, each time one group will be considered as the test set and the k-1 left as the training set. Therefore the CV becomes:

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$$

We can see that the k-Fold Cross-Validation fits the model k times instead of n, which reduces considerably the computation time. Our original data have 198 observations so we can afford the computation of the LOOCV.

Model implementation

```
1  library("kknn")
2  model_kknn = train.kknn(Time ~., data= cancer.train, kmax =
       30, ks = NULL, distance = 2, kernel = "optimal")
3  best_k_train = model_kknn$best.parameters$k
```

After deducting the best k neighbors from the model we use it on the test observations to predict the values of Time. We then compute the MSE and plot the predicted values $\hat{y}$ on the real ones y.

```
1  best_reg_train = knn.reg(train= cancer.train.x, test =
       cancer.test.x, y=cancer.train.y, k = best_k_train)
2  plot(cancer.test.y, best_reg_train$pred, xlab='y',
       ylab='prediction')
3  abline(0,1, col='red')
4  errors = (cancer.test.y - best_reg_train$pred)^2
5  mse= mean(errors)
```

The results are shown below:
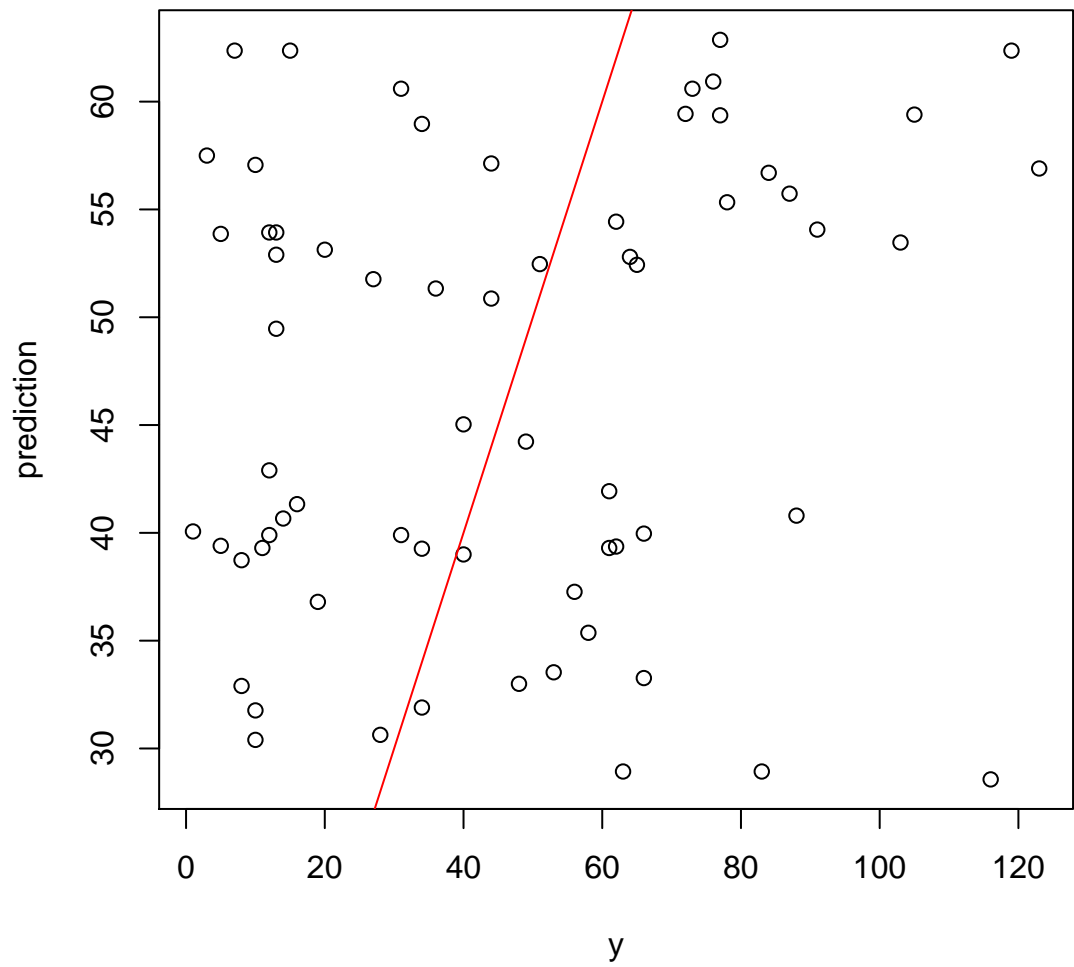
Figure 11: MSE against K neighbours

best_k_train= 30
best_k_test_MSE = 1047.085675

Model Analysis

One can argue that the prediction is not better since the test MSE obtained is
higher that the one we had when we computed with the best test k. However

19

the model with the validation approach is not biased, in other words, in general the k=30 will guarantee a smaller MSE than the k=11 on any test observation.

0.9 Simple Linear Regression

0.9.1 Idea

Our next attempt consists in using the same linear model we used in the feature analysis section. This model takes advantage of the simple assumption that `Time` linearly depends on the other features.

To build the model, we can use the function `lm` :

```
model.linreg = lm(Time ~ ., data=train_set)
```

0.9.2 Model Performance

The MSE of this model is approximatively equal to 1285. The raw residual distribution is shown on figure 12.

0.10 Linear Regression with Features Selection

0.10.1 Idea

A simple method to improve the performance of a simple Linear Regression is to select a subset of features that better describes the distribution of `Time`.

Once the simple linear model is fitted, we can use the function `summary` to display the value of each coefficient.

```
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept)            5.722e+01  1.587e+02   0.361   0.7191
Lymph_node            -1.752e-01  7.034e-01  -0.249   0.8038
radius_mean            3.001e+01  4.445e+01   0.675   0.5012
texture_mean          -3.913e-01  2.043e+00  -0.192   0.8485
perimeter_mean        -3.794e+00  6.651e+00  -0.570   0.5697
area_mean             -1.328e-01  1.354e-01  -0.981   0.3291
smoothness_mean       -7.124e+02  9.050e+02  -0.787   0.4331
compactness_mean      -8.767e+01  3.554e+02  -0.247   0.8057
concavity_mean        -2.654e+02  2.728e+02  -0.973   0.3332
concave_points_mean    1.236e+03  5.911e+02   2.090   0.0392 *
symmetry_mean         -3.199e+01  2.647e+02  -0.121   0.9041
fractal_dimension_mean 1.818e+03  1.666e+03   1.091   0.2781
radius_se             -2.897e+01  9.953e+01  -0.291   0.7716
texture_se            -1.784e+00  1.246e+01  -0.143   0.8865
perimeter_se           8.817e+00  1.289e+01   0.684   0.4955
```

area_se	-3.871e-01	4.212e-01	-0.919	0.3604
smoothness_se	2.987e+03	2.535e+03	1.178	0.2415
compactness_se	7.135e+02	7.864e+02	0.907	0.3665
concavity_se	3.798e+02	7.170e+02	0.530	0.5975
concave_points_se	-7.782e+02	1.450e+03	-0.537	0.5926
symmetry_se	-1.239e+03	7.861e+02	-1.576	0.1182
fractal_dimension_se	-5.979e+03	6.280e+03	-0.952	0.3434
radius_worst	9.300e-01	1.367e+01	0.068	0.9459
texture_worst	-1.307e+00	1.964e+00	-0.666	0.5072
perimeter_worst	-1.026e+00	1.449e+00	-0.708	0.4805
area_worst	8.705e-02	7.021e-02	1.240	0.2181
smoothness_worst	-3.761e+02	4.088e+02	-0.920	0.3599
compactness_worst	-7.031e+01	9.964e+01	-0.706	0.4821
concavity_worst	-3.011e+01	7.311e+01	-0.412	0.6814
concave_points_worst	-2.181e+02	2.345e+02	-0.930	0.3546
symmetry_worst	1.586e+02	1.425e+02	1.113	0.2687
fractal_dimension_worst	1.028e+03	7.100e+02	1.448	0.1509
Tumor_size	-1.188e+00	1.915e+00	-0.620	0.5364

The last column contains the P-value of each coefficient, which is a measure to test the hypothesis that this particular coefficient is null. A P-value lower than 5% allows us to conclude that the coefficient is not equal to zero. In our case, the feature concavity_points_mean is not null, but we cannot make such assumptions for the other parameters. Therefore, we are not able to select a subset of interesting features based on the P-values.

Our dataset contains a small set of features to deal with, therefore we can use an exhaustive feature subset selection algorithm. Such method build a linear model based on each subsets and compute a performance score to compare them.

We use the following function to fit the linear models :

```
model.linreg.regsubsets = regsubsets(Time ~ .,
    data=train_set, method = "exhaustive", nvmax = 32)
```

Then the function plot to compare the models according to a given scale (in this case, the BIC measure). The result is shown on figure **??**.

```
plot(model.linreg.regsubsets, scale="bic")
```

According to this plot, the best BIC is reached with a model that only uses the following features :

- texture_mean
- fractal_dimension_mean
- concavity_mean

0.10.2 Model Performance

The MSE of this model is approximatively equal to 1067. The raw residual distribution is shown on figure 13.

0.11 Linear Regression with Regularization

In this section we will discuss some methods that will help us shrink the model by reducing the number of parameters. We will use the **glmnet** package in order to build the ridge regression and the lasso in R.

0.11.1 Ridge Regression

The Linear Regression with least squares estimates the parameters $\beta_0, \beta_1...\beta_p$ that to minimize the term of the RSS.

$$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$$

Ridge Regression works the same way as the least squares in the sense that it also tries to minimize the RSS but is also has another term $\lambda \sum j\beta_j^2$ called the **shrinkage penalty** where $\lambda \geq 0$ is the **tuning parameter**. The formula is:

$$RSS + \lambda \sum_{j} \beta_j^2 \tag{1}$$

If λ=0 we are in the same case of a least squares estimates. The higher λ gets, the higher will be the penalty. Hence Ridge regression will try to minimize the parameters β_j in order to minimize the term 1.

Model Implementation

The glmnet function takes for parameters the matrix x of predictors and vector y of responses. The model.matrix will help us transform out data sets into matrix. This function not only gives out a matrix but it also converts the qualitative variables into dummy variables.

```
1  x.train = model.matrix(Time~.,cancer.train)[,-1]
2  y.train = cancer.train$Time
3
4  x.test =model.matrix(Time~.,cancer.test)[,-1]
5  y.test = cancer.test$Time
```

The glmnet function takes in parameter the train data, the pramater alpha that indicates whether it is a Ridge or Lasso regression that we want to perform (alpha=0 for Ridge). By default glmnet chooses an automatic range of κ, however here we chose a wide range $\lambda \in [10^2, 10^{10}]$ to cover all possibilities.

```
1  library(glmnet)
2  grid = 10^seq(10, -2, length=100)
3  ridge.mod = glmnet(x.train, y.train, alpha=0, lambda=grid)
4  plot(ridge.mod)
```
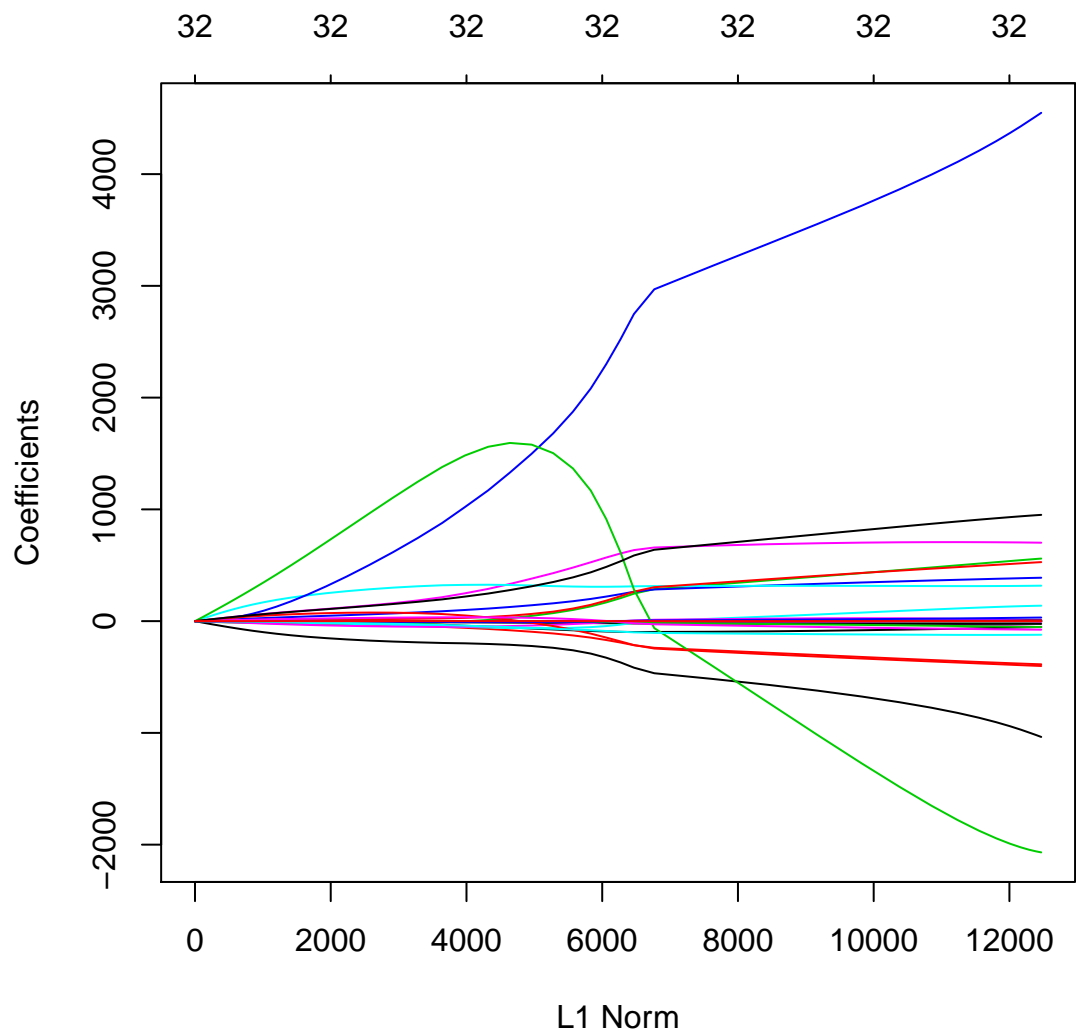
Figure 14: Coefficients β_j against L1 Norm

The figure below shows that the higher the norm L1 is the smaller are the coefficients β_j.

In order to find the best tuning parameter λ we perform a cross validation on the training data. cv.glmnet function runs a 10 fold cross validation on the data.

```
1  cv.out = cv.glmnet(x.train, y.train,alpha=0)
2  plot(cv.out)
3  best_lambda = cv.out$lambda.min
```
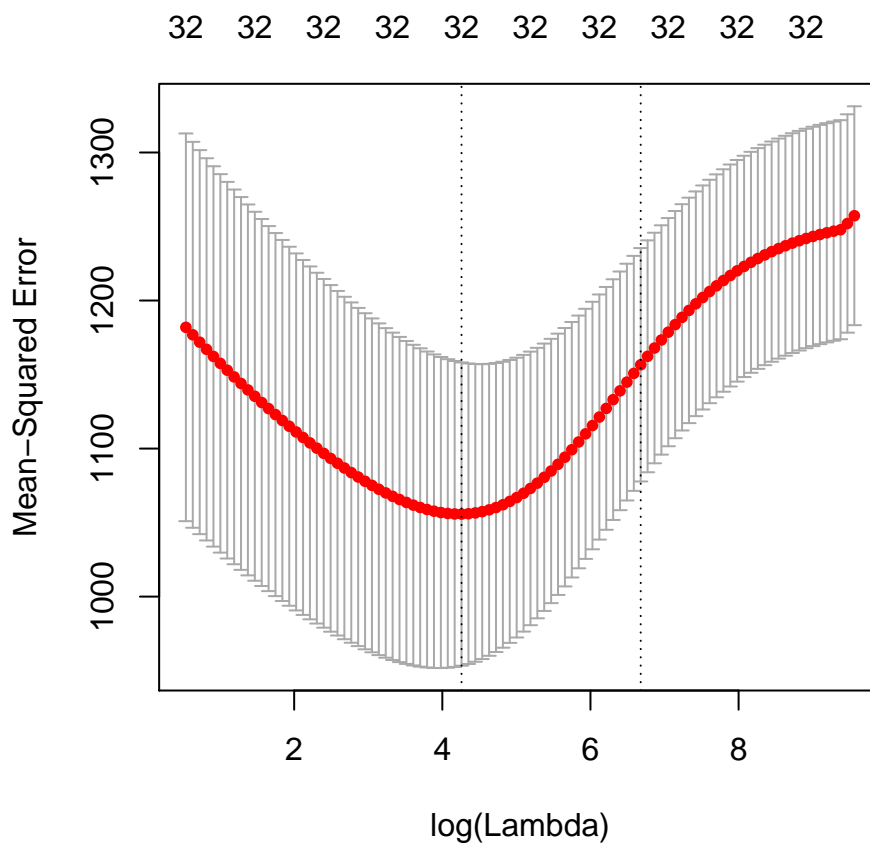

Figure 15: MSE against $\log(\lambda)$

The value λ that yields to the smallest MSE is shown in the graph near $log(\lambda) = 4$ and $log(\lambda) = 5$

```
1  fit.ridge = glmnet(x.train, y.train, lambda=best_lambda,
       alpha=0)
2  ridge.pred = predict(fit.ridge, s=best_lambda, newx=x.test)
3  MSE = mean((y.test - ridge.pred)^2)
4  residuals = y.test - ridge.pred
```

```
5   plot(x=y.test, y=ridge.pred)
6   abline(0,1, col='red')
7   plot(x=y.test,y=residuals)
```

$$\text{best}\_\lambda = 3.522385$$
$$\text{best\_test\_MSE} = 1017.18$$
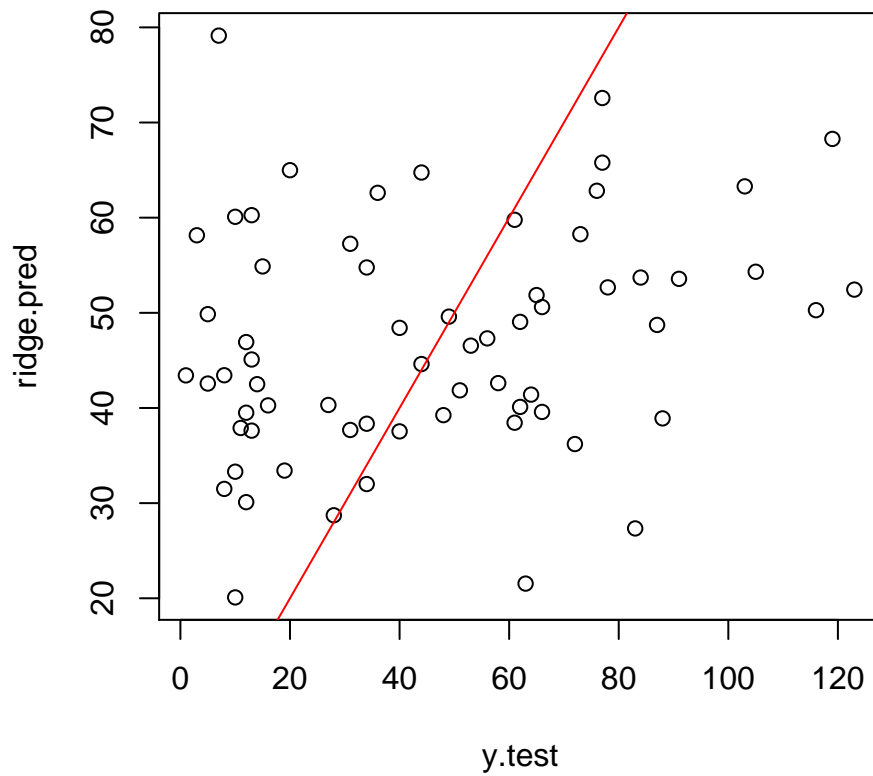
Figure 16: predicted Time $\hat{y}$ against real responses y
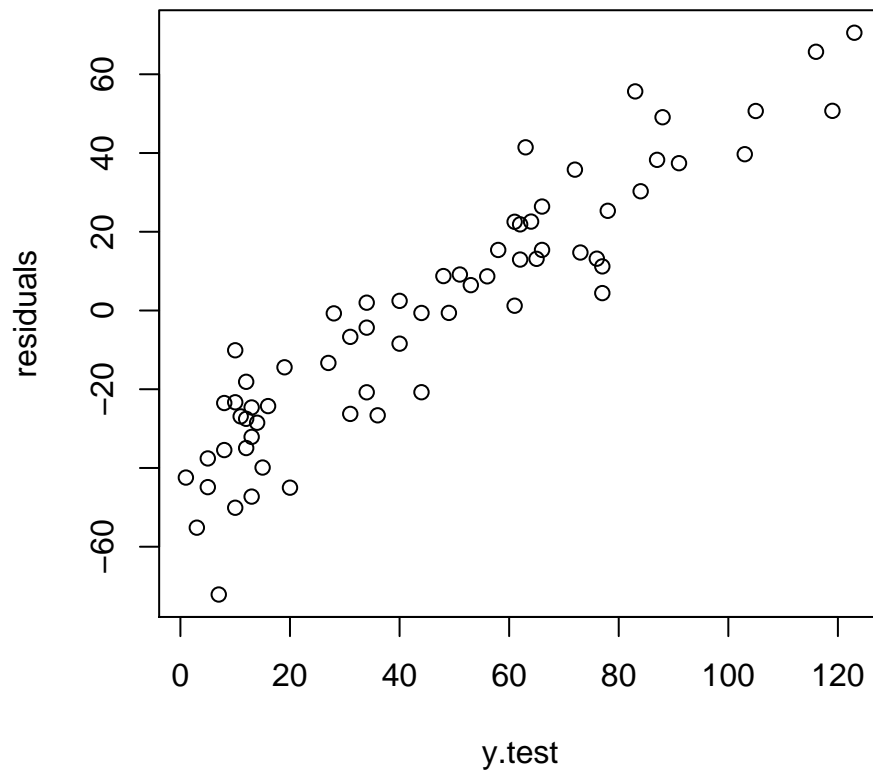
Figure 17: Residuals against Time(y)

Performing a Ridge regression didn't really improve out MSE, we notice that we still have the same shape when we plot the predicted response against the real values. In fact we still have the extreme values scattered away from the line y=x while the values in the middle are grouped around it. The residuals shows most residuals are among -20 and 40, however we can't neglect the important number of observation that have residuals below -20.

After predicting the responses with the best_λ, we can also to get the coefficients β_j for this model.

```
1  predict(ridge.mod, type="coefficients",
       s=best_lambda)[1:33,]
```

(Intercept)	50.4600029742775
Lymph_node	-0.0740382365112173
radius_mean	-0.385497956046896
texture_mean	-0.439640231767502
perimeter_mean	-0.0611376994572613
area_mean	-0.00352287155395527
smoothness_mean	87.6824334874088
compactness_mean	-4.01389237413323
concavity_mean	-16.8784966228312
concave_points_mean	-21.0103808557697
symmetry_mean	37.9942559603596
fractal_dimension_m...	215.765069490171
radius_se	-1.30277609111998
texture_se	-4.36181426998151
perimeter_se	-0.367716257339956
area_se	-0.0122846670311223
smoothness_se	192.720973139687
compactness_se	-13.1530380711173
concavity_se	-30.7162937829027
concave_points_se	-129.927319653512
symmetry_se	15.0202933696317
fractal_dimension_se	524.172237262352
radius_worst	-0.10822285260215
texture_worst	-0.233696210472122
perimeter_worst	-0.0264959094156711
area_worst	-0.000866271952319244
smoothness_worst	62.5696518012362
compactness_worst	2.80025635825785
concavity_worst	-1.65992645320788
concave_points_worst	-8.85978373663052
symmetry_worst	20.2867760331075
fractal_dimension_wo...	85.4647011613642
Tumor_size	0.0260172046297538

Figure 18: coefficients β_j for ($\lambda = 64.48$)

Model Analysis

So we notice that some coefficients are very close to 0 such as area_se (-0.012), but none of them is null. In fact Ridge Regression only shrinks the coefficients and does not perform any variable selection; that is why we are going to use Lasso in the following part.

After comparing both MSE of the least squares and Ridge Regression we notice that Ridge did not really improve our prediction. The question is why do we still affirm that Ridge improves over the least squares? It all can be summarized in **biais-variance trade-off**. Let's take the following example where we we fit the model for each λ and predict the responses for $\lambda \in [10^{-1}, 10^4]$. For each value of λ we compute the MSE, the squared bias and the cube root of the variance. We computed the cube root in order to be able to scale purposes.

```
1  max = 100
2  biais2<-rep(0,max)
3  variance<-rep(0,max)
4  mse<-rep(0,max)
5  grid=10^seq(4,-1, length=max)
6  for( i in 1:max)
7  {
8      fit.ridge = glmnet(x.train, y.train, lambda=grid[i],
             alpha=0)
9      ridge.pred = predict(fit.ridge, s=grid[i] ,newx=x.test)
10     mse[i] = mean((ridge.pred - y.test)^2) #MSE
11     biais2[i] = (mean(ridge.pred - y.test))^2 #squared bias
12     variance[i] = (var(ridge.pred))^(1/3) #variance
13 }
14 plot(grid, variance,type='l', xlab="lambda",
        main="Biais-Variance trade-off")
15 lines(grid, biais2,col='red')
16 plot(mse)
```

Bias–Variance trade-off
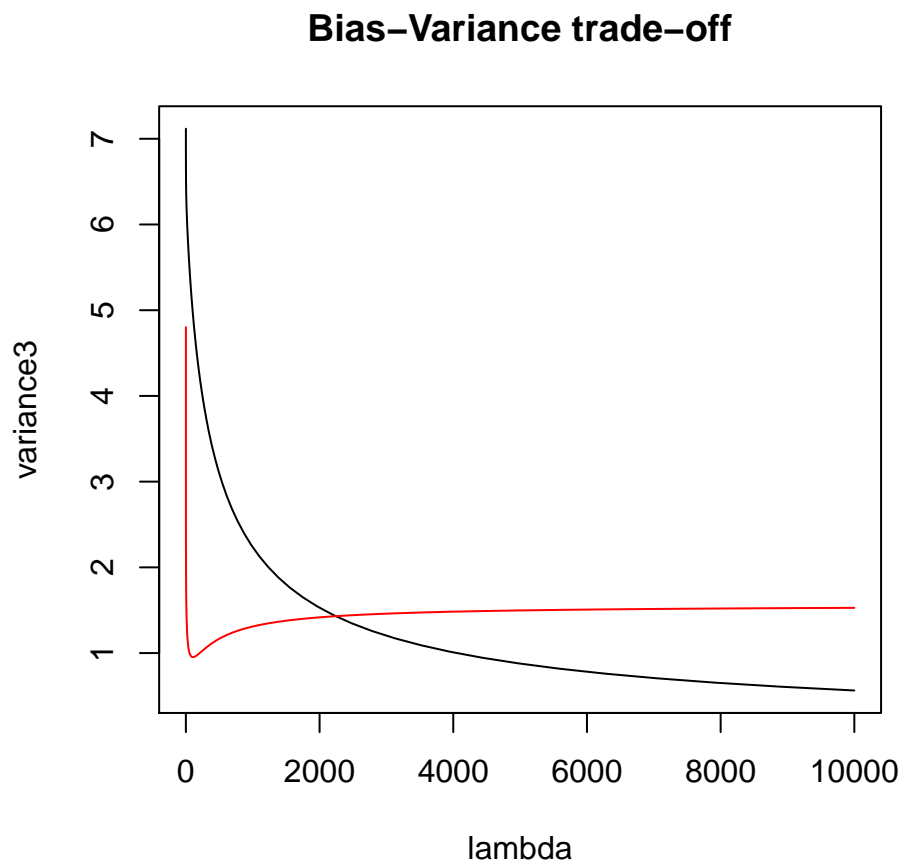
Figure 19: Bias Variance trade-off, squared bias(red), root squared variance(black)
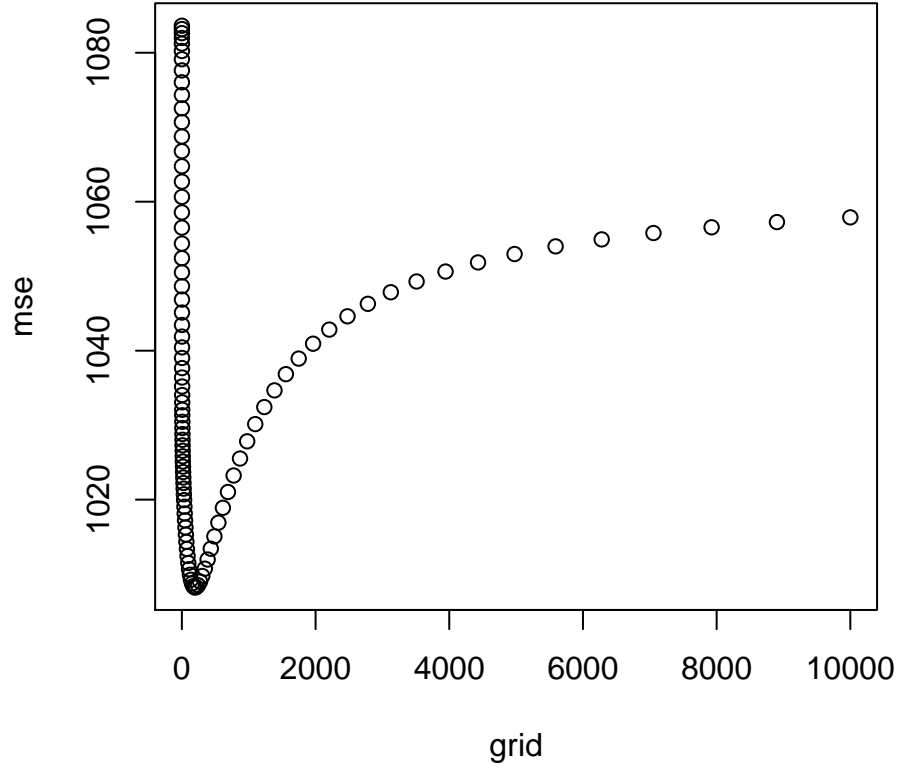
Figure 20: MSE against λ

At $\lambda = 0$ (least squares), the variance is high but there is no bias. As λ increases the variance and bias decrease but at some point the variance continues to decrease while the bias starts increasing. Which means that the shrinkage of the coefficients by λ reduces the variances on the expense of the bias, which can be explained by the fact that λ in underestimating the coefficients by shrinking them. Let us compare now those results to the MSE curve.

$$MSE = (E[\Theta] - \Theta)^2 + Var(\Theta) = (Bias[\Theta])^2 + Var(\Theta)$$

Hence if the variance and the bias decrease significantly in the beginning the MSE will decrease too, and when the variance will decrease less and the bias will start increasing the MSE will increase too. We should also note that at at $\lambda = 0$ (least squares) the MSE is high.

To sum up, in linear regression we can have a low bias but a high variance, this

31

is where the ridge regression improves over the least squares because it shrinks the variance on the expense of the bias.

0.11.2   Lasso Regression

As it was discussed on the previous section, Ridge regression disadvantage is that it does not perform a variable selection. Lasso Regression tries to minimize the term:

$$RSS + \lambda \sum_j |\beta_j^2| \tag{2}$$

The only difference with the term of Ridge is that now we have $|\beta_j^2|$ instead of β_j^2. This alternative will reduce the β_j that are close to zero to null. Hoping we have a more interpretable model with reducing the number of variables. To call the lasso model we use the same function glmnet but with alpha=1.

Model Implementation

```
1  grid = 10^seq(10,-2, length=100)
2  lasso.mod = glmnet(x.train, y.train, alpha=1,lambda=grid)
3  plot(lasso.mod)
```
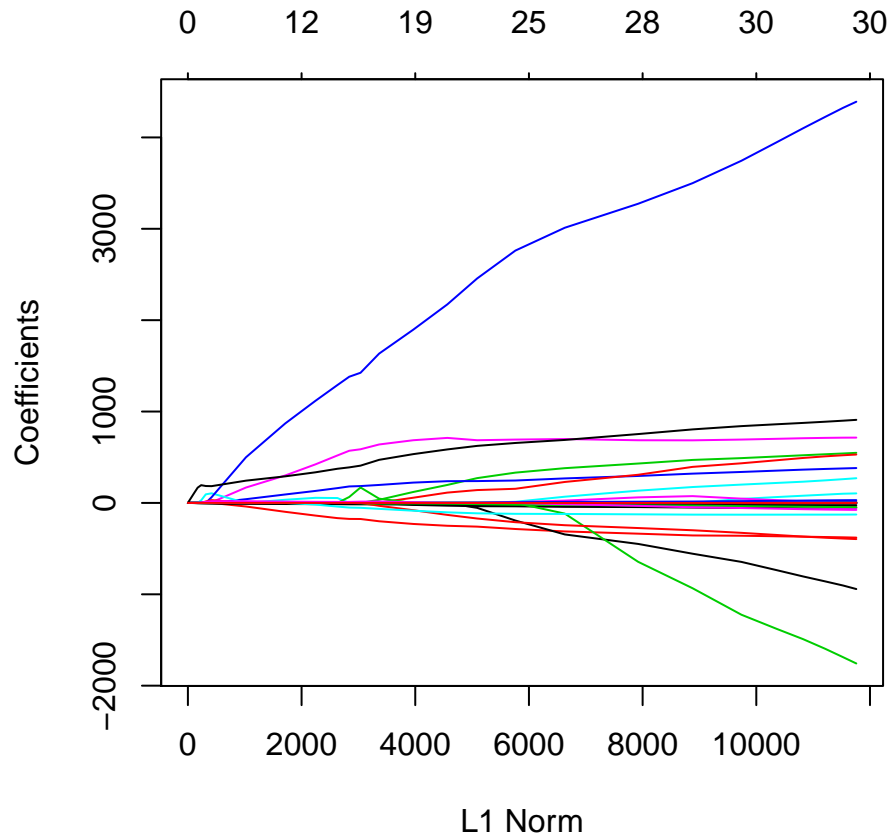
Figure 21: MSE against λ

We notice that for some values of λ the coefficients are null. We now perform the 10 fold cross validation on the training set to deduct the best tuning parameter.

```
1  cv.out = cv.glmnet(x.train, y.train, alpha=1)
2  plot(cv.out)
3  best_lambda = cv.out$lambda.min
4  lasso.pred = predict(lasso.mod, s=best_lambda , newx=x.test)
5  mean((lasso.pred-y.test)^2)
6  plot(x=y.test, y=lasso.pred)
7  abline(0,1, col='red')
```
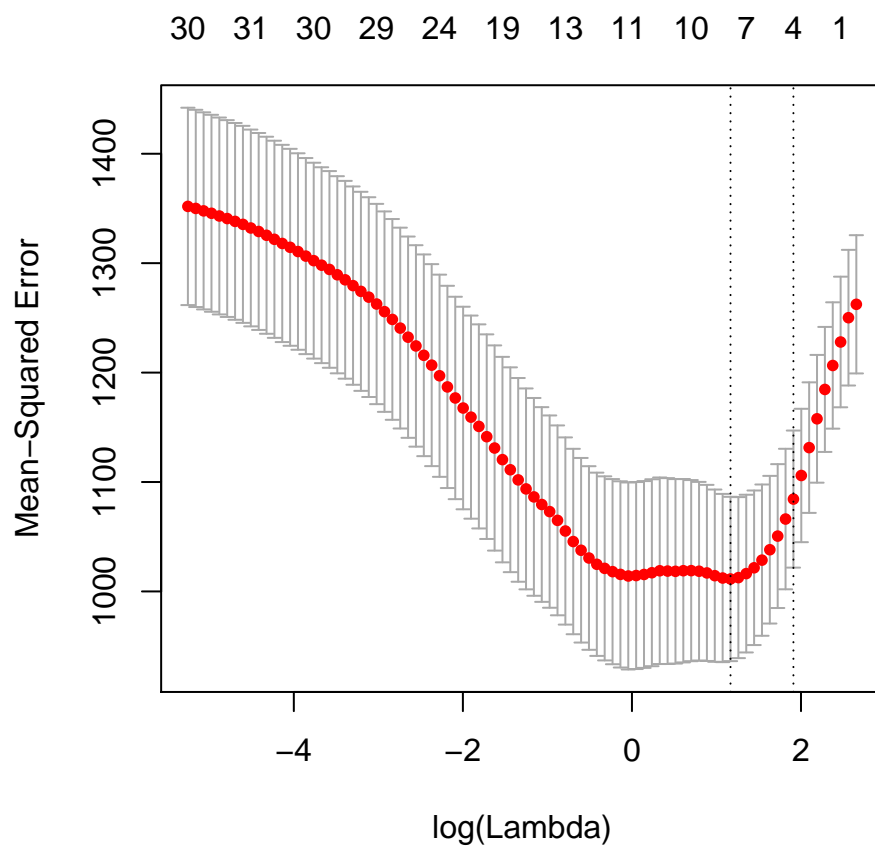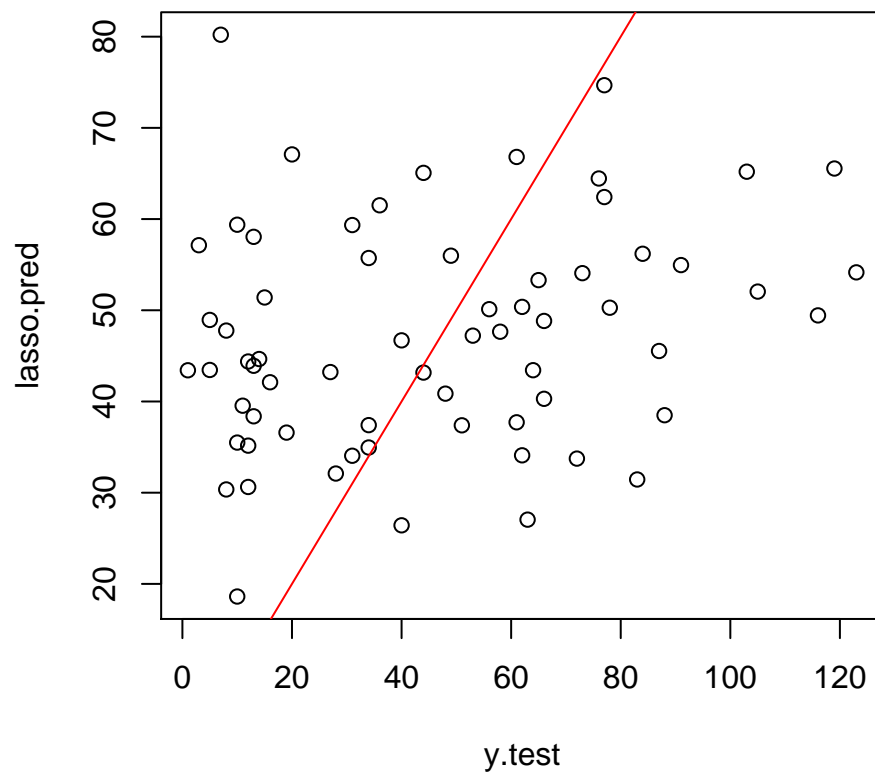
Figure 22: MSE against λ

Figure 23: MSE against λ

```
1  predict(lasso.mod, type="coefficients",
       s=best_lambda)[1:33,]
```

(Intercept)	50.4600029742775
Lymph_node	-0.0740382365112173
radius_mean	-0.385497956046896
texture_mean	-0.439640231767502
perimeter_mean	-0.0611376994572613
area_mean	-0.00352287155395527
smoothness_mean	87.6824334874088
compactness_mean	-4.01389237413323
concavity_mean	-16.8784966228312
concave_points_mean	-21.0103808557697
symmetry_mean	37.9942559603596
fractal_dimension_m...	215.765069490171
radius_se	-1.30277609111998
texture_se	-4.36181426998151
perimeter_se	-0.367716257339956
area_se	-0.0122846670311223
smoothness_se	192.720973139687
compactness_se	-13.1530380711173
concavity_se	-30.7162937829027
concave_points_se	-129.927319653512
symmetry_se	15.0202933696317
fractal_dimension_se	524.172237262352
radius_worst	-0.10822285260215
texture_worst	-0.233696210472122
perimeter_worst	-0.0264959094156711
area_worst	-0.000866271952319244
smoothness_worst	62.5696518012362
compactness_worst	2.80025635825785
concavity_worst	-1.65992645320788
concave_points_worst	-8.85978373663052
symmetry_worst	20.2867760331075
fractal_dimension_wo...	85.4647011613642
Tumor_size	0.0260172046297538

Figure 24: coefficients β_j for $(\lambda = 64.48)$

We can clearly see that Lasso performed a variable selection by setting some

coefficients to 0.

Model Analysis

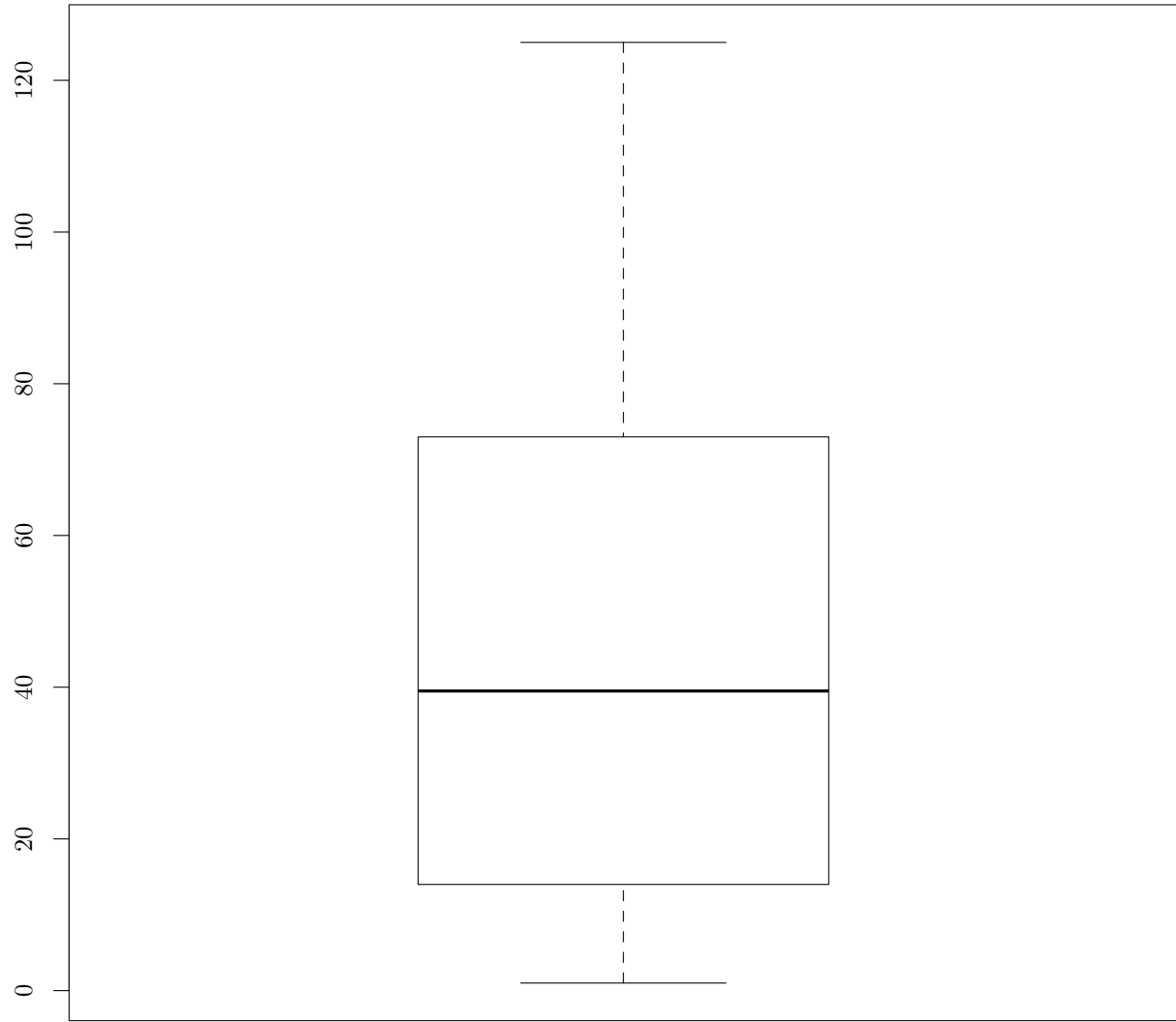# 0.12 Models Comparaison

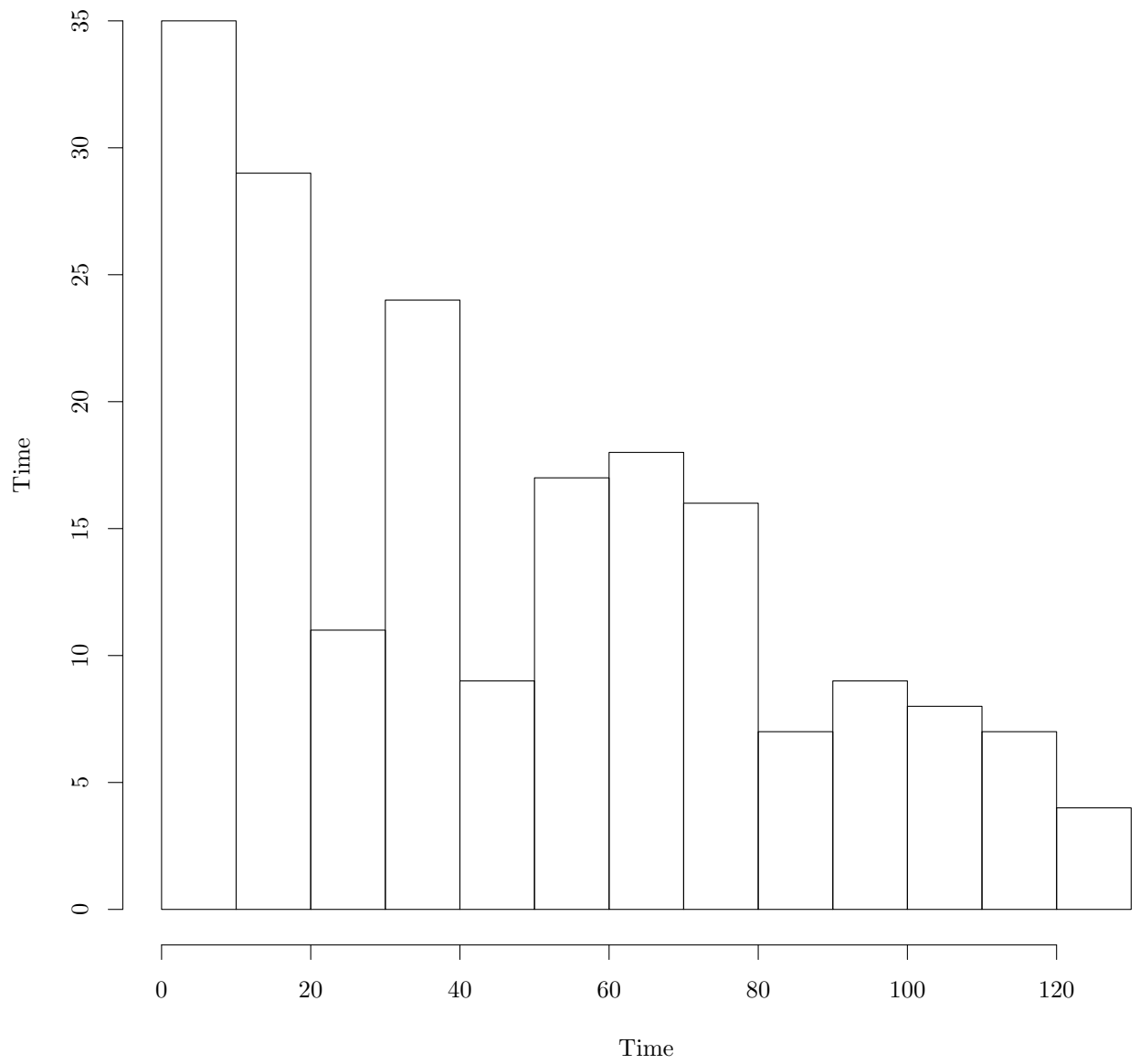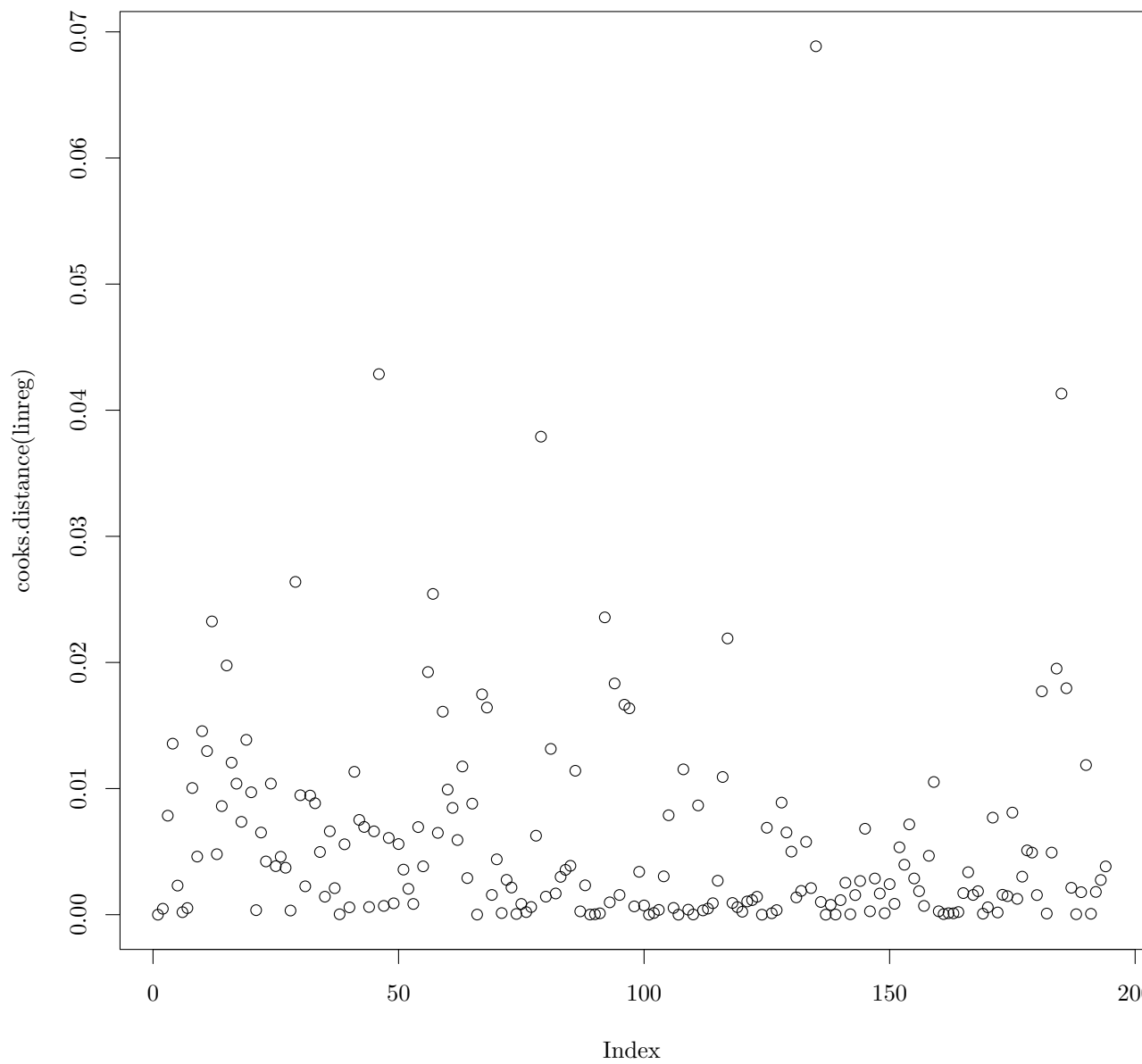*USE TEST SET TO COMPARE MODEL*

Figure 3: Box Plot

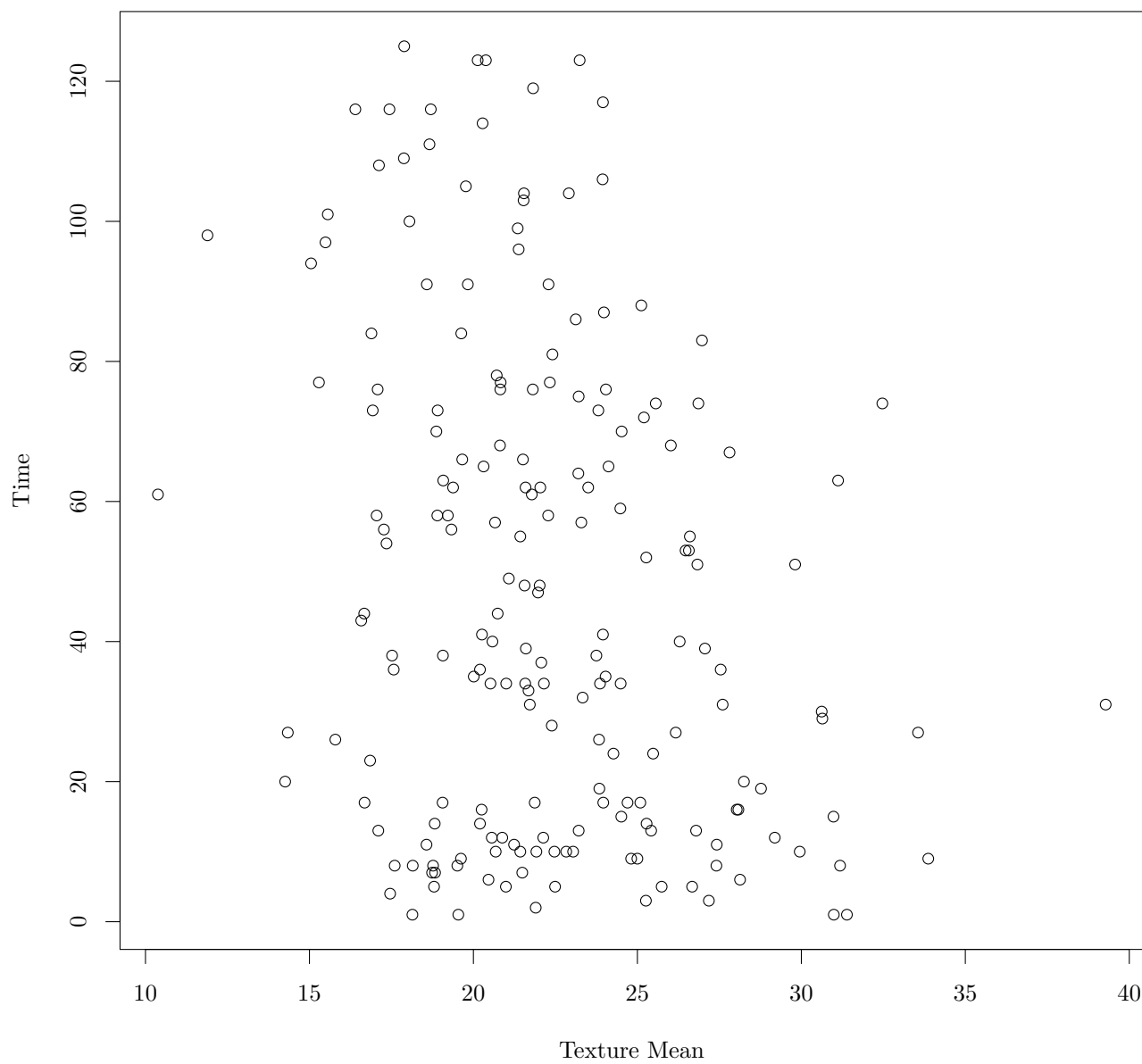Figure 4: Histogram

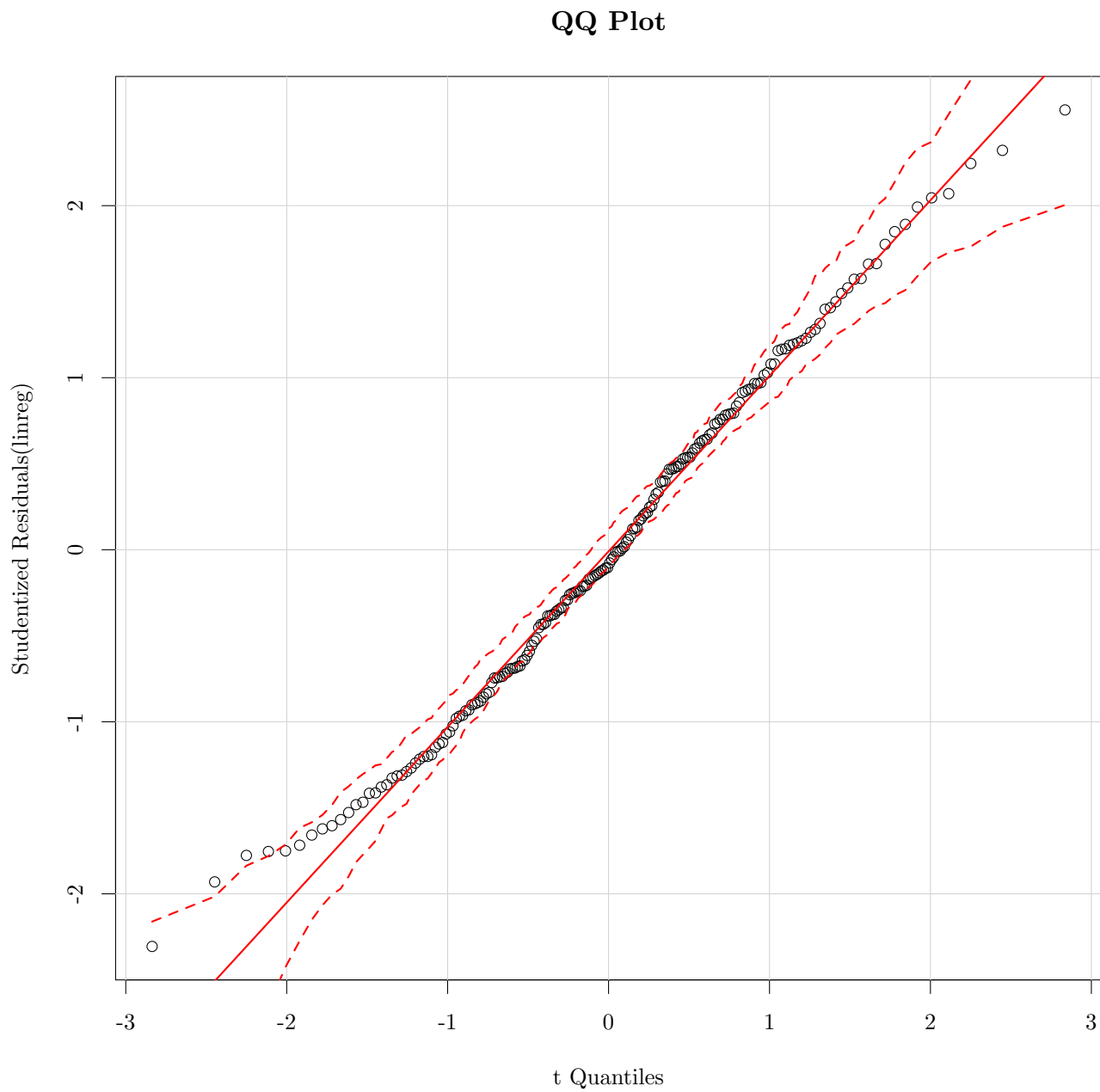Figure 5: Cook's Distance
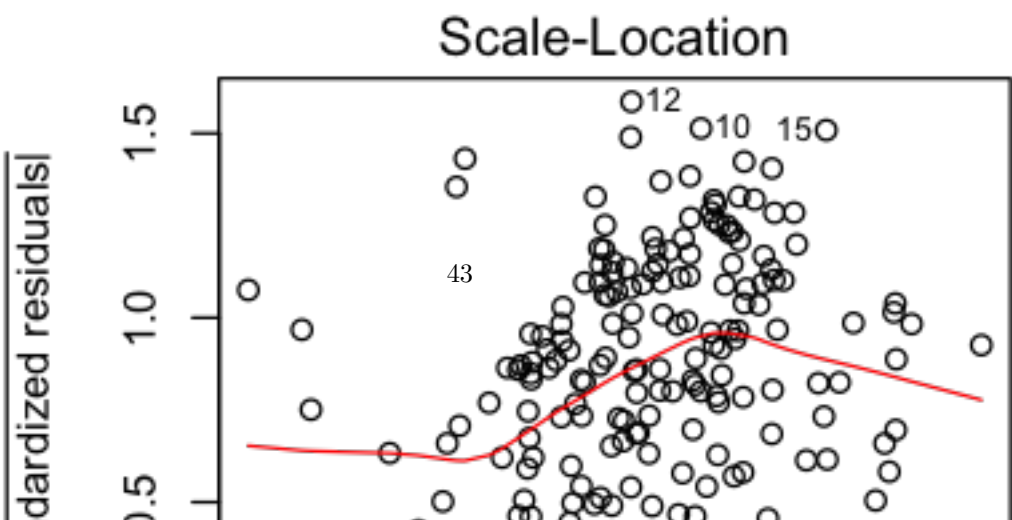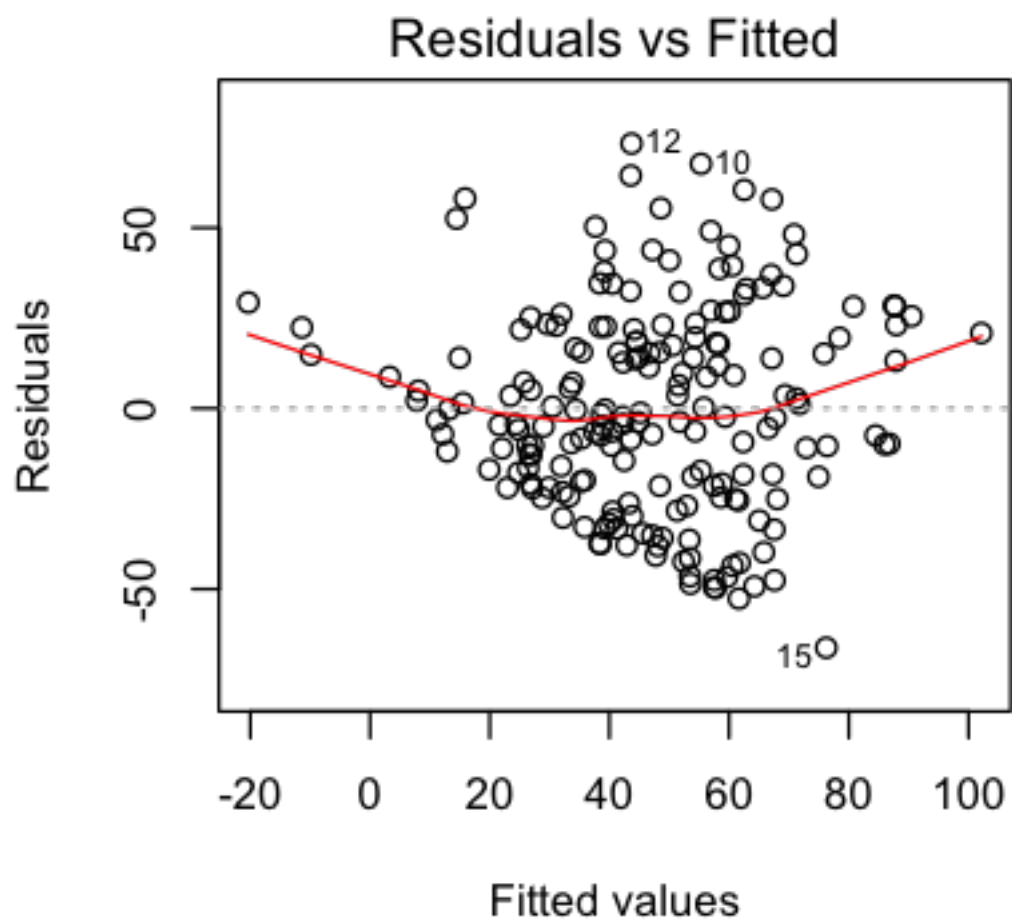
Figure 6: Plot of `Time` and `Texture Mean`

QQ Plot

Figure 7: QQ-Plot

42

Residuals vs Fitted

Scale-Location

43

Raw Residuals Distribution

Figure 12

Raw Residuals Distribution

Figure 13