SY19

November 21, 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

Chapter 1

Introduction

This report

Chapter 2

Ex 1- Phoneme Recognition

2.1 Context

In the context of speech recognition the aim is to predict which of the phonemes is pronounced my the subject. We have five phonemes to recognize :

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

2.2 Dataset Description

The study involved 4509 speeches pronounced by 50 male speakers. The method used for speech recognition is the Log-periodogram, an estimate of the spectral density of the sound signal.

Our dataset is composed of 4509 log-periodograms (observations) of length 256 (features). The column labelled `speakers` identifies the different speakers. We notice that some are labeled train and some test. Hence we have a training set composed of 3340 observations (74%) and a test set that comprises 1169 observations (26%). 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 2.1: Frequencies of phonemes in the train and test data set

Our dataset tells us which speaker the periodogram is extracted from. We could input this information in the model, however, the system may not be aware of the speaker in a real-case scenario, therefore, we will not consider the speaker as a feature.

```
data_set = read.csv("phoneme.data.txt")
head(data_set)
data_set.nb_features = 256
```

Splitting data into train set and test set.

```
train= data_set[1:3340,]
train.x = train[,-258]
train.y = train[,258]

test = data_set[3341:4509,]
test.x = test[,-258]
test.y = test[,258]
```

2.3 Classification

2.3.1 LDA- Linear Discriminant Analysis

Model Implementation

The function `lda` works the 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.

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

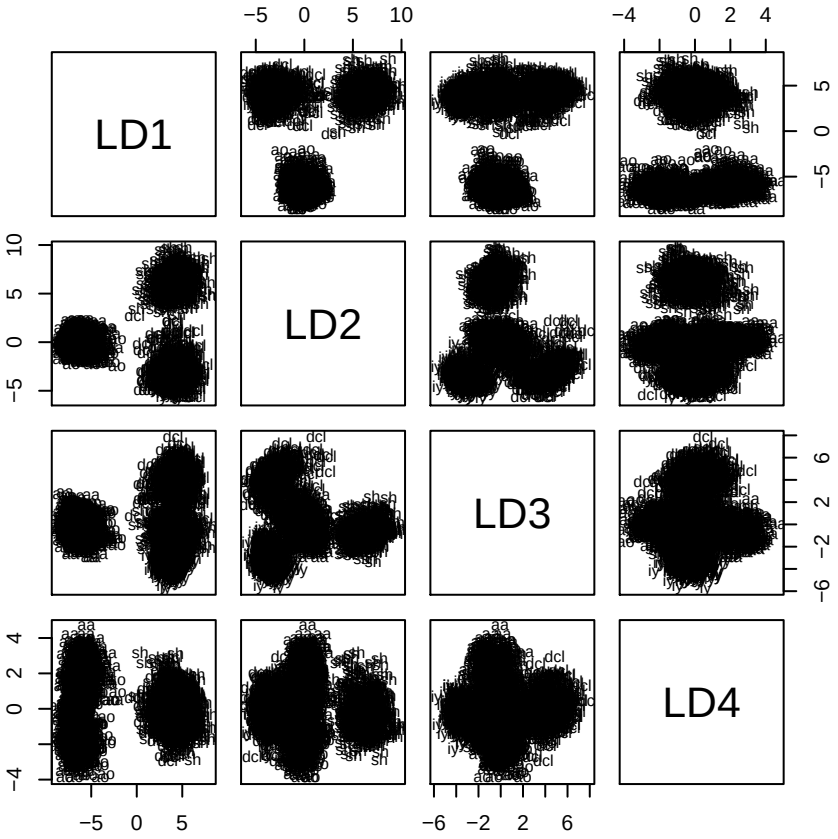

Figure 2.1: Phoneme LDA model

The prior probabilities tell us that in the train observation 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 of the test set of data. This function returns an element `class`, a list of the predicted phonemes. An element `posterior` with the posterior probability of the response of the k-th class. Last the linear discriminants are found in `x`.

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

The table perf of the predicted responses is shown below:

```
 g 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. This classifier performs quite well, but it does work well with the phonemes `aa` and `ao` which sound very similar.

The `ROCR` package can be used to produce ROC curves. The area under the ROC curve(AUC) is a criteria to judge the performance of the LDA. The larger the AUC the better the classifier.

```
library(pROC)
l   = length(model.lda.predicted$class)
ld1 = model.lda.predicted$x[1:l]
ld2 = model.lda.predicted$x[1+l:1+2*l]
ld3 = model.lda.predicted$x[2*l:3*l]
ld4 = model.lda.predicted$x[3*l:4*l]
roc_curve<-roc(test_set$g, as.vector(ld1))
plot(roc_curve)
```

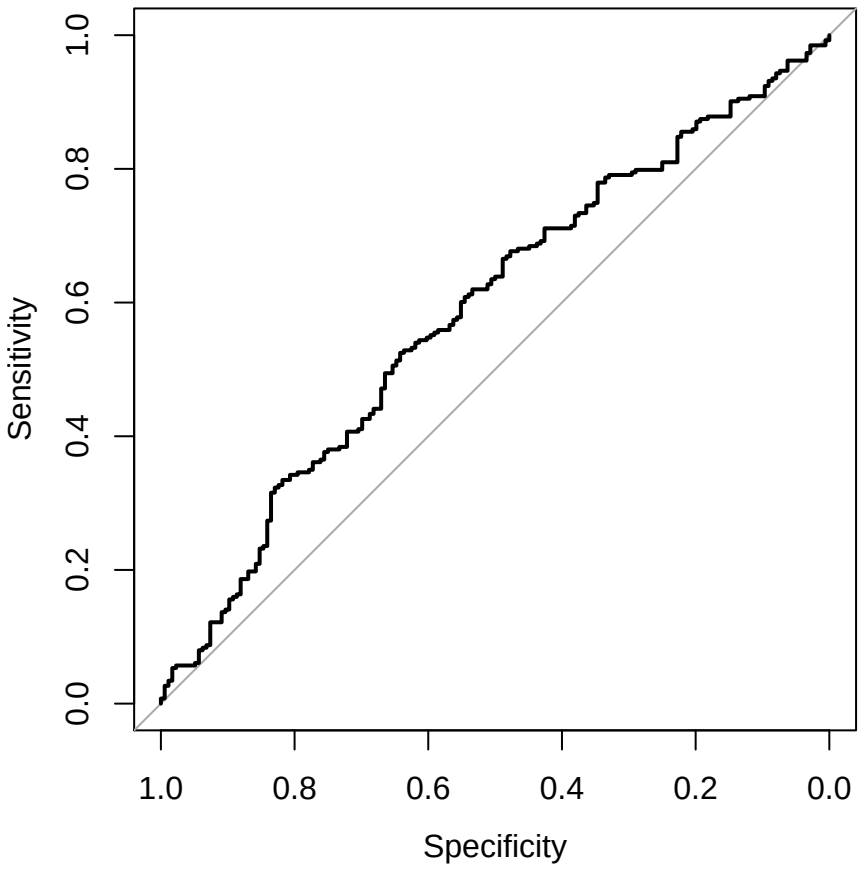

Figure 2.2: LDA ROC: Area under the curve: 0.588

For this data the AUC is 0.92, which is considered very good as it is close to the maximum.

Model Analysis

2.3.2 QDA- Quadratic Discriminant Analysis

Model Implementation

Model Analysis

2.3.3 Logistic Regression

Logistic Regression is a type of Linear Classification method that models the posterior probabilities with linear functions. When there are only two features

to deal with, the model is definined by :

$$\log \frac{\mathbb{P}(Y=0|X=x)}{\mathbb{P}(Y=1|X=x)} = \beta_0 + \beta x$$

If $\beta_0 + \beta x$ is positive, Y is most likely to be equal to 0; otherwise, Y is most likely to be equal to 1.

In a multinomial scenario, like our case, more complex methods have to be applied. In particular, the library `nnet` provides us with a function called `multinom` that uses a feedforward Neural Network with a single hidden layer to fit a multinomial logistic regression model. This kind of Neural Networks often runs a backpropagation method along with an iterative optimization algorithm to estimate the weights of each neuron. That is why we have to specify the number of maximum iterations.

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

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

Model Implementation

Model Analysis

2.4 Models Comparison

Chapter 3

Ex 2 - Breast Cancer Recurring Time

3.1 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.

3.2 Dataset Description

The very first step of our method consists in taking a look at the raw dataset to get precious 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.

3.2.1 Time

Let's first describe the distribution of the variable `Time`. To do so, we can use the R functions `boxplot` (figure 3.1) and `hist` (figure 3.2). According to these figures, our dataset mostly represents short reappearing times (lower than 40), and there are very few patients whose variable `Time` is higher than 80. However, we do not know if this distribution is also representative of the whole population. If this is is the case, our model should be able to work on other datasets, if not, our model will be biaised.

3.2.2 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, Standard Error and Mean of the three largest values (also called "Worst") of
  - Radius
  - Texture
  - Perimeter
  - Area
  - Smoothness
  - Compactness
  - Concavity
  - Concave points
  - Symmetry
  - Fractal dimension
- Tumor Size

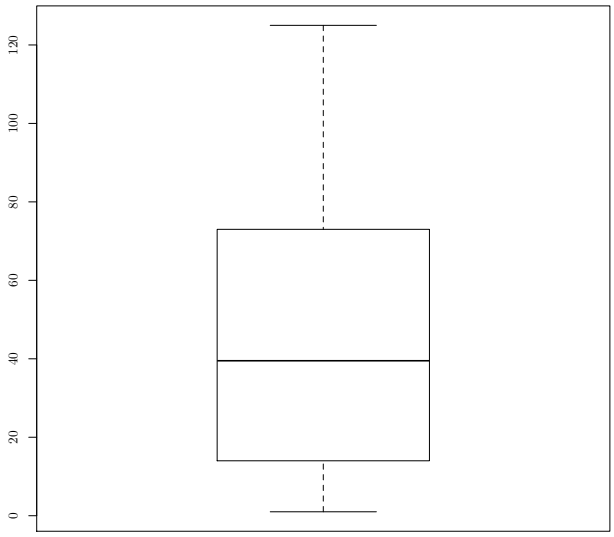

Figure 3.1: Box Plot

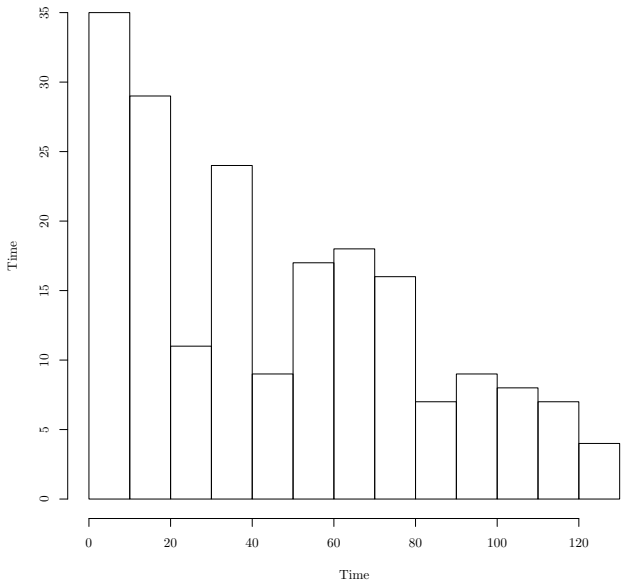

Figure 3.2: Histogram

Feature Correlation

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

- The mean of each parameters is smaller than the "worst" value;
- The radius, the perimeter and the area are most likely to be linked together;
- The compactness can be computed with the perimeter and the area thanks to the given formula : $Compactness = \frac{perimeter^2}{area-1}$

These dependent features might be a cause of model low performance.

3.2.3 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 i-th 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)
```

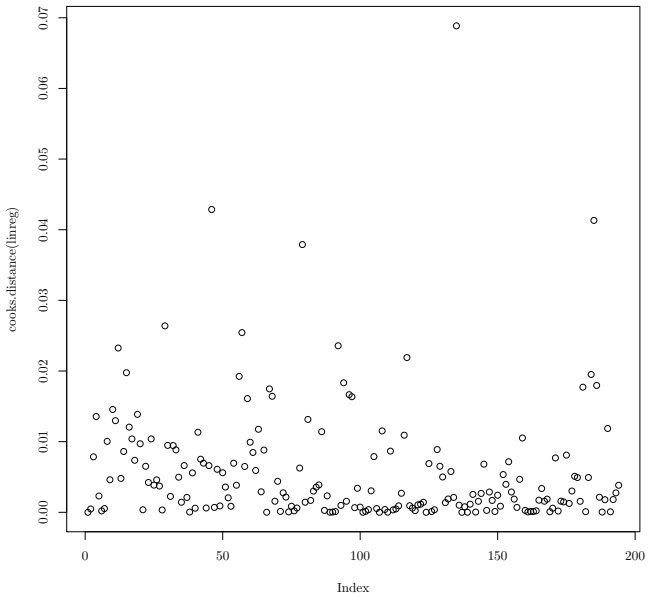

Figure 3.3: Cook's Distance

According to this plot (figure 3.3), 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.

3.2.4 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 `Time` against each feature. An example of such plot is shown in figure 3.4. Unfortunately, most plots picture

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.

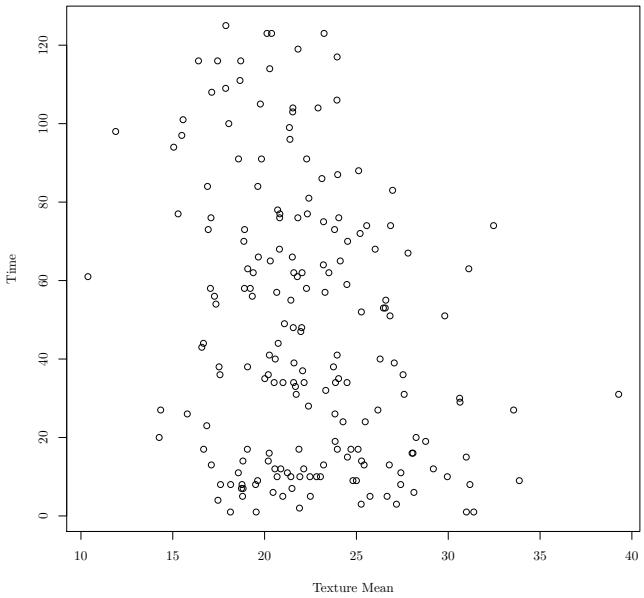

Figure 3.4: Plot of `Time` against `Texture Mean`

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 against the quantiles.

In R, we can use the library `car` to easily draw the QQ-Plot (figure 3.5) :

```
1 library(car)
2 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 3.6). 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.

3.3 Measures to Compare Models

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

3.3.1 Some Measures

A first way to assess the performance of a model F is to compute the Mean Squared Error (MSE). Let $\hat{Time}$ be the estimate of the variable $Time$ using the model F. MSE is defined as :

$$MSE(F) = mean_i(Time_i - \hat{Time}_i)^2$$

We can also use adjusted R^2 score.

3.3.2 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 66% of the dataset dedicated to the model fitting;

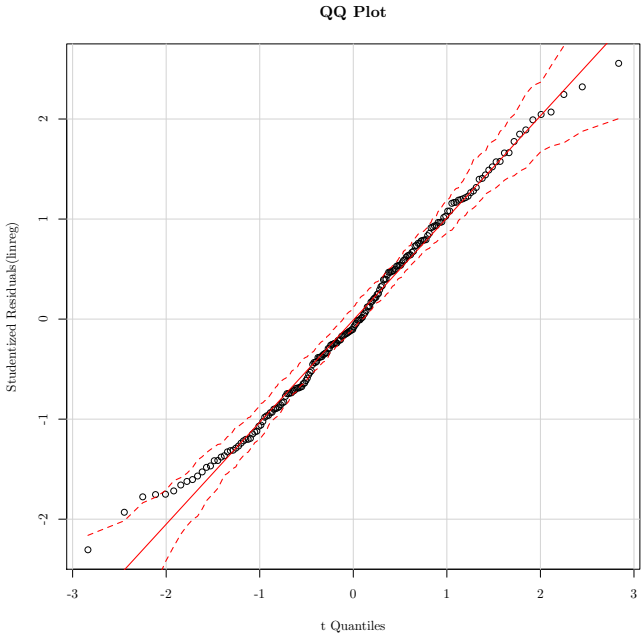

Figure 3.5: QQ-Plot

- Test Set : The remaining 34% only used at the end to provide some kind of objective measure of the model performance.

```
n = dim(data_set)[1]
train_id = sample(1:n, n * 2/3)

train_set = data_set[train_id,]
train_set.x = train_set[,-33]
train_set.y = train_set[,33]

test_set = data_set[-train_id,]
test_set.x = test_set[,-33]
test_set.y = test_set[,33]
```

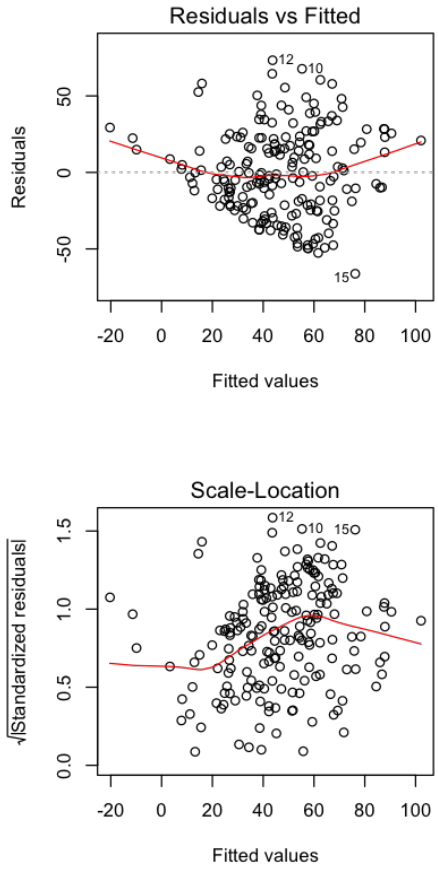

Figure 3.6

Once this data split is done, we can finally dive into the model building.

3.4 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. The KNN model first identifies the k closest points to each point of the test observation then it estimates the response using the average of the k closest training responses.

3.4.1 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 120), 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

```
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_train = which.min(MSE)
best_k__train_mse = MSE[best_k_train]
```

As you will notice in the code, we labeled the best_k and the MSE as train because we chose our best parameters based on the test directly and not the train.
The graph below (figure 3.7) shows the **Train 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.

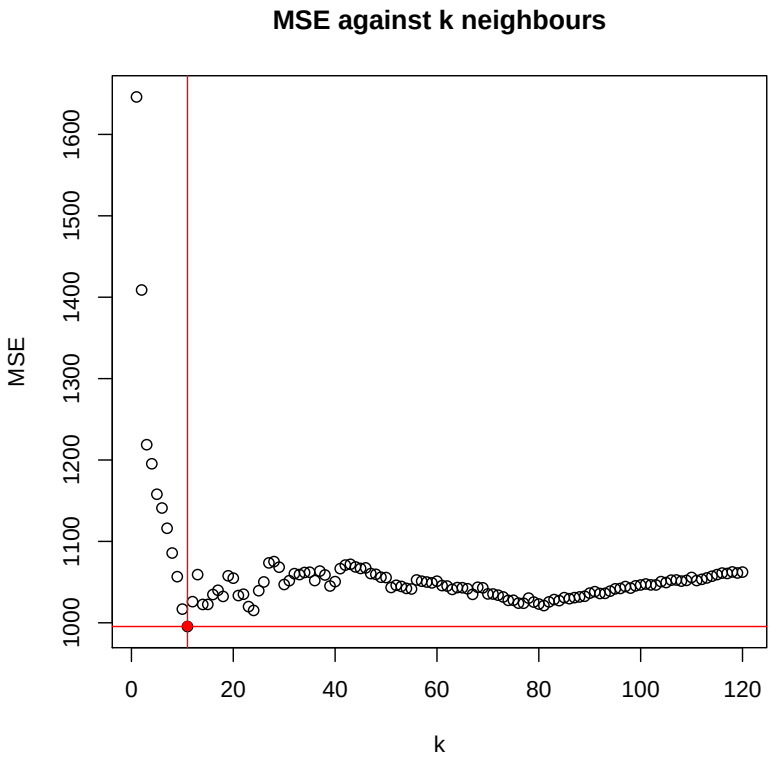

Figure 3.7: Train MSE against K neighbours

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

best_k_Train= 11
best_k_Train_MSE = 995.433185

Now that we have the k that minimizes the Train MSE we call KNN algorithm with this best k and plot the predicted values against the real values. The figure 3.8 shows the result.

```
best_reg_test = knn.reg(train= cancer.train.x, test =
    cancer.test.x, y=cancer.train.y, k = best_k_train)
plot(cancer.test.y, best_reg_test$pred, xlab='y',
    ylab='y-hat', main='y-hat (Predicted) against y')
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).

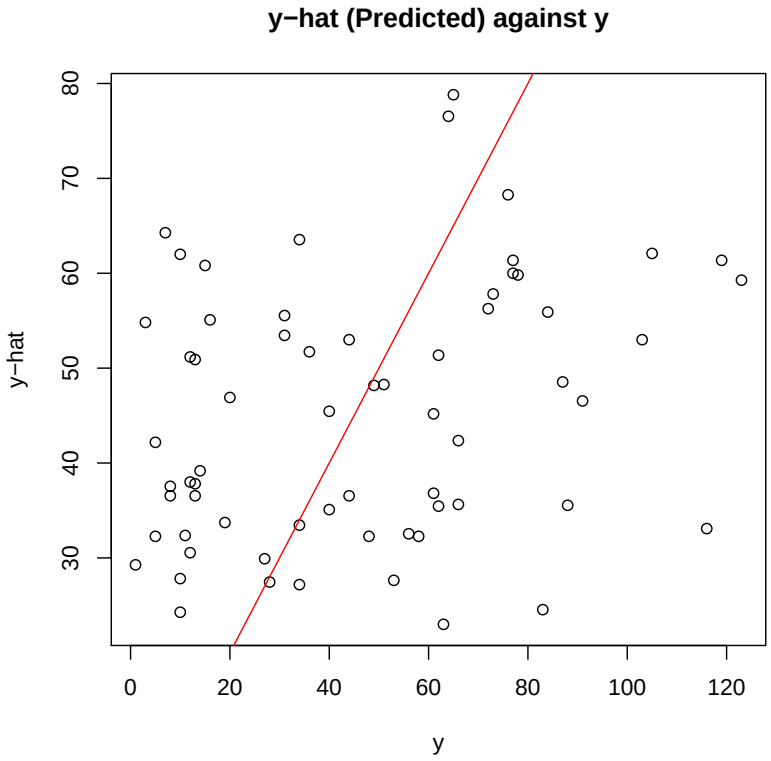

Figure 3.8: Predicted values against the real values

Model Analysis

We notice that the predicted $\hat{y}$ diverge a lot from the real values y. We expected those results since the MSE is around 967 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 k we use the method of cross validation on the train data then we predict the response on the test.

3.4.2 The Validation Set Approach

There are two main methods in cross validation: **the validation set approc**h and the **cross validation leave one out (LOOCV)** which is a particular case of the **K-fold cross validation**. As we do not have that much observations (n=198) we can afford the computation of cross validation leave one out, but before we will argument our choice.

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. We validate our model using the best MSE.

Leave-One-Out Cross-Validation

Like the cross validation the LOOCV involves splitting the set of observations in two parts. The main difference is that we have a single observation (x_1, y_1) in the test data and the $n-1$ remaining is used for the train data. The MSE in this case is $MSE_1 = (y_1 - \hat{y})^2$. This provides an unbiased estimate for the test error since the size of the train model is approximately the one of 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 as it was stated before the CV runs the train 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 set approach. The only disadvantage of the LOOCV is it is computation time which can be very time consuming n is large.

An alternative to LOOCV that has a smaller computation time is k-Fold Cross Validation. This approach is 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 has only 198 observations, we can then afford the computation of the LOOCV.

Model implementation

```
library("kknn")
model_kknn = train.kknn(Time ~., data= cancer.train, kmax =
    30, ks = NULL, distance = 2, kernel = "optimal")
best_k = 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 $\hat{y}$ against the real ones y.

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

The results are shown in figure 3.9 .

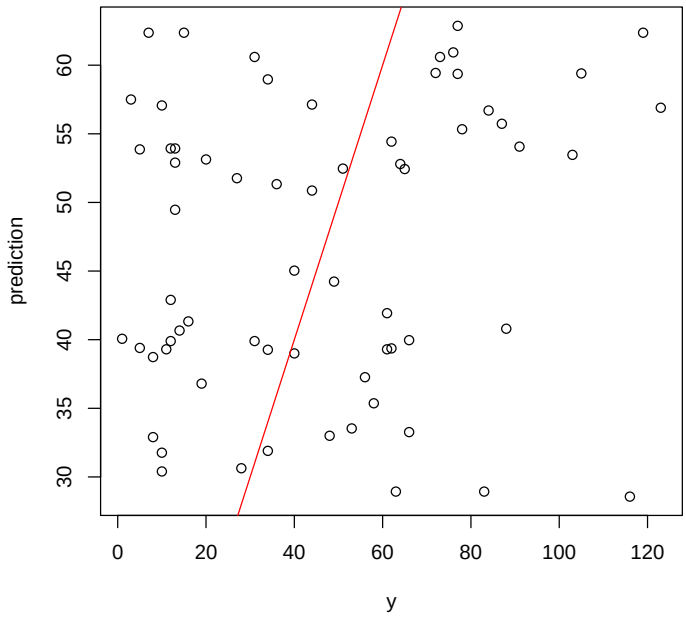

Figure 3.9: Predicted values against Real values

best_k= 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 MSE we obtained when we computed with the best train k. However this result is normal since in the first approach we run our knn directly on the test so we underestimated the MSE, while the model with the cross validation is not biased. In other words, in general the $k = 30$ will guarantee a smaller MSE than the $k = 11$ on any test observation.

3.5 Simple Linear Regression

3.5.1 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` depends on the other features in a linear way. A linear regression model

has the following form :

$$Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$$

The regression coefficients β_i are choosen so that they minimize the MSE. Both analytical and optimization methods (Gradient Descent, Stochastical Gradient Descent, ...) can be used to find the best coefficient estimates.

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

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

3.5.2 Model Performance

This model has a MSE approximatively equal to 1285. The raw residuals distribution is pictured on figure 3.10, it shows a very spread out distribution that should be improved.

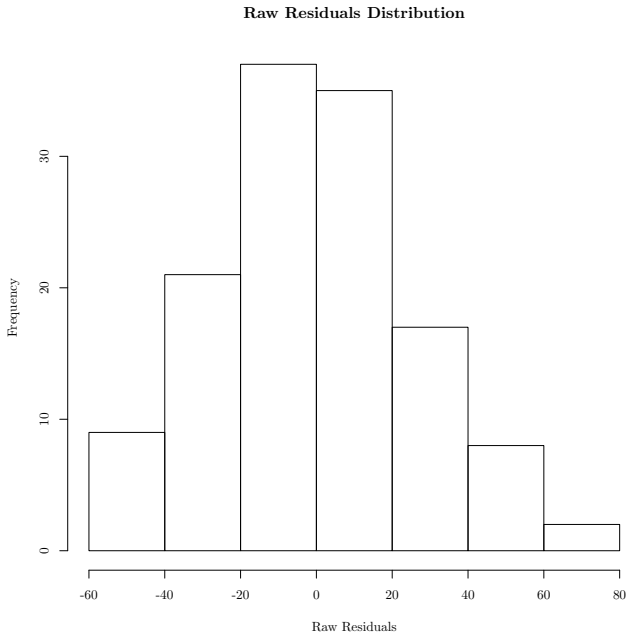

Figure 3.10

3.6 Linear Regression with Features Selection

3.6.1 Idea

A simple method to improve the performance of the previous 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.

Feature subset selection algorithms exist to extract the best subset of features from the dataset. Such method builds a linear model based on multiple subsets of features and compute a performance score to compare them. Our dataset contains a quite small set of features to deal with, therefore we can use an exhaustive feature subset selection algorithm which will apply a linear regression on each and every subset available.

In R, the following function is available to fit the linear models :

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

We can then use the function `plot` to compare the models according to a given scale (in this case, the BIC measure). The result is shown on figure 3.11.

```
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`

3.6.2 Model Performance

The MSE of this model is approximatively equal to 1067, which is better than the full-featured model. The raw residual distribution, shown on figure 3.12, is not as spread out as the previous linear regression model.

3.7 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.

3.7.1 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.

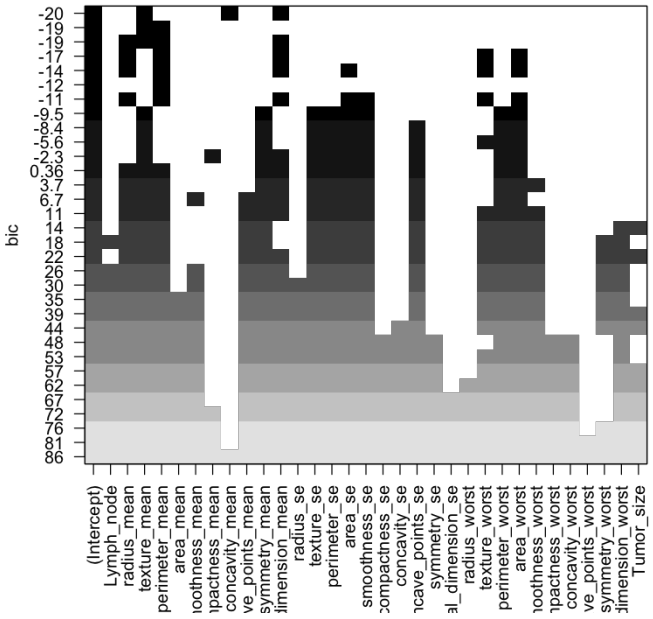

Figure 3.11: BIC-score of each feature subset

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

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:

$$RSS + \lambda \sum_{j} \beta_j^2 \tag{3.1}$$

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 3.1.
When applying the penalty on the coefficients those with different scales (for instance a perimeter in m and the other one in Km) will be "treated" differently , because the penalized term is a sum of squares of all the coefficients. An alternative to get the penalty applied uniformly across the predictors is to standardize the independent predictors first with the function scale.

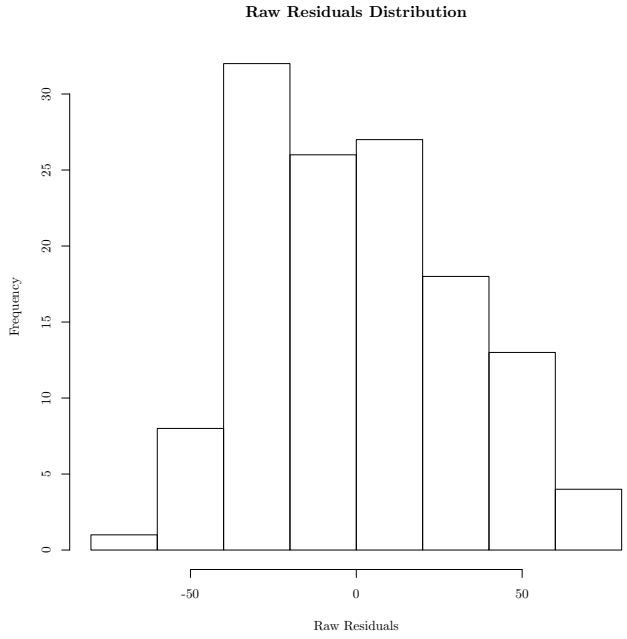

Figure 3.12

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.

```
x.train = model.matrix(Time~.,cancer.train)[,-1]
y.train = cancer.train$Time
scale(x.train, center = TRUE, scale = TRUE)

x.test =model.matrix(Time~.,cancer.test)[,-1]
y.test = cancer.test$Time
scale(x.test, center = TRUE, scale = TRUE)
```

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.

```
library(glmnet)
grid = 10^seq(10, -2, length=100)
ridge.mod = glmnet(x.train, y.train, alpha=0, lambda=grid)
```

```
plot(ridge.mod)
```

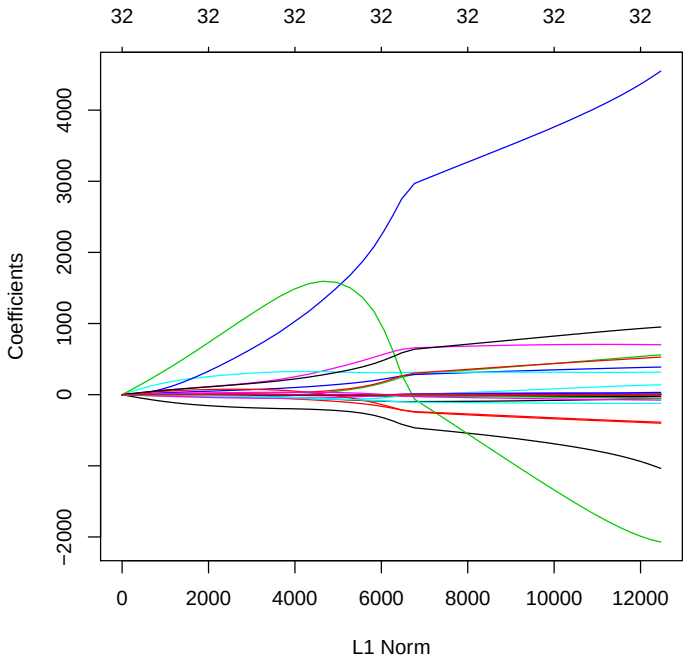

Figure 3.13: Coefficients β_j against L1 Norm

The figure 3.13 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.

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

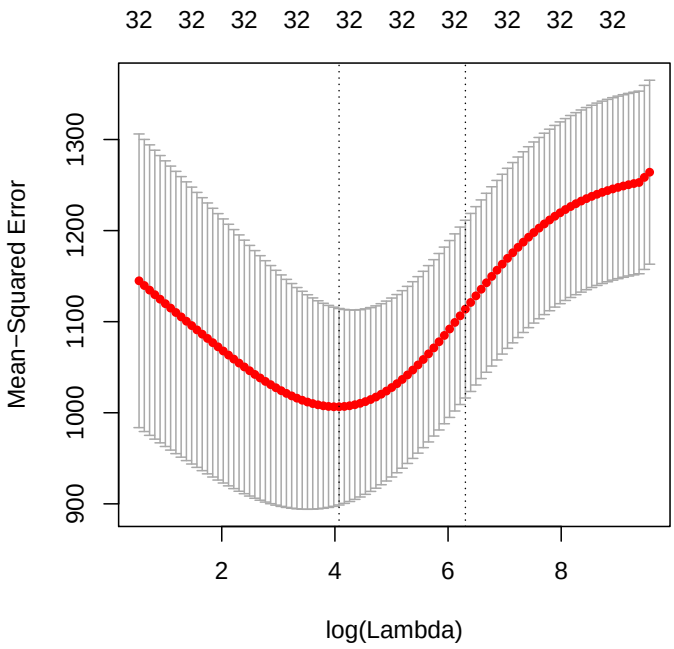

Figure 3.14: MSE against $\log(\lambda)$

The value λ that yields to the smallest MSE is shown in the graph 3.14 near $log(\lambda) = 4$ and $log(\lambda) = 5$

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

$$\text{best\_}\lambda = 58.75693$$
$$\text{best\_test\_MSE} = 1015.485$$

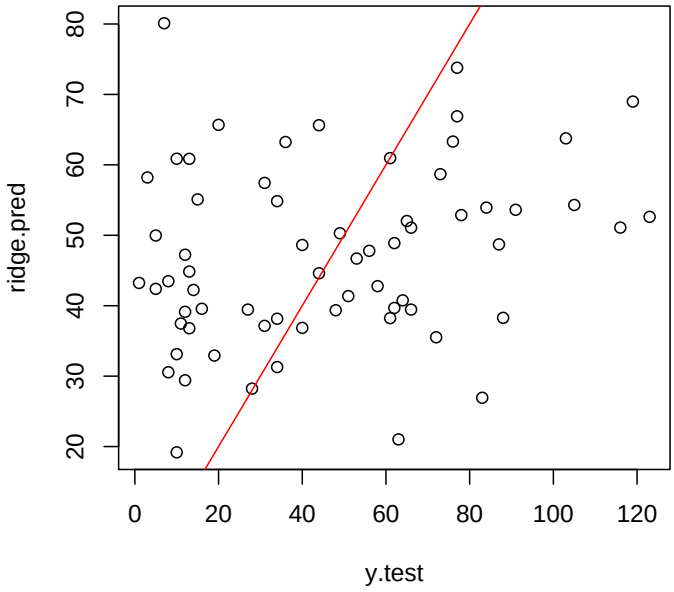

Figure 3.15: predicted Time $\hat{y}$ against real responses y

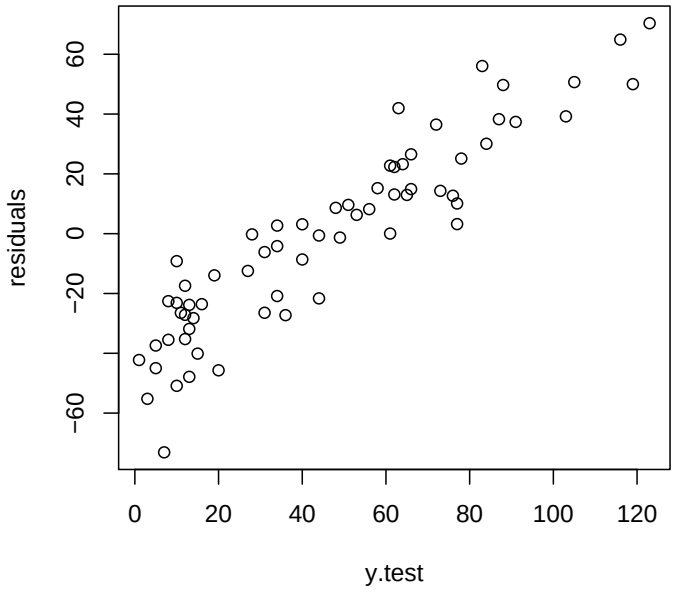

Figure 3.16: Residuals against Time(y)

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.

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

(Intercept)	48.9783494205899
Lymph_node	-0.0796231361072258
radius_mean	-0.395758970057055
texture_mean	-0.469991574403947
perimeter_mean	-0.0635689055806551
area_mean	-0.00365245681222613
smoothness_mean	95.1340975435298
compactness_mean	-5.41777262342363
concavity_mean	-18.8788872243882
concave_points_mean	-21.8056348232133
symmetry_mean	41.359863583279
fractal_dimension_m...	228.867988039387
radius_se	-1.16881937756599
texture_se	-4.81931663217402
perimeter_se	-0.37340368115239
area_se	-0.0119811087528577
smoothness_se	231.993485614537
compactness_se	-15.3778242479491
concavity_se	-33.5032494393098
concave_points_se	-138.617199733536
symmetry_se	15.6487187834847
fractal_dimension_se	586.528252622921
radius_worst	-0.0920779375222904
texture_worst	-0.247869970089008
perimeter_worst	-0.0259355498933169
area_worst	-0.000778327462223526
smoothness_worst	66.2335842414962
compactness_worst	2.74997566980902
concavity_worst	-2.11475226928221
concave_points_worst	-10.2228618437314
symmetry_worst	21.8421235897495
fractal_dimension_wo...	92.8905950895525
Tumor_size	0.04086192938721

Figure 3.17: Ridge coefficients β_j for ($\lambda = 58.75$)

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 summa-

rized 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.

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

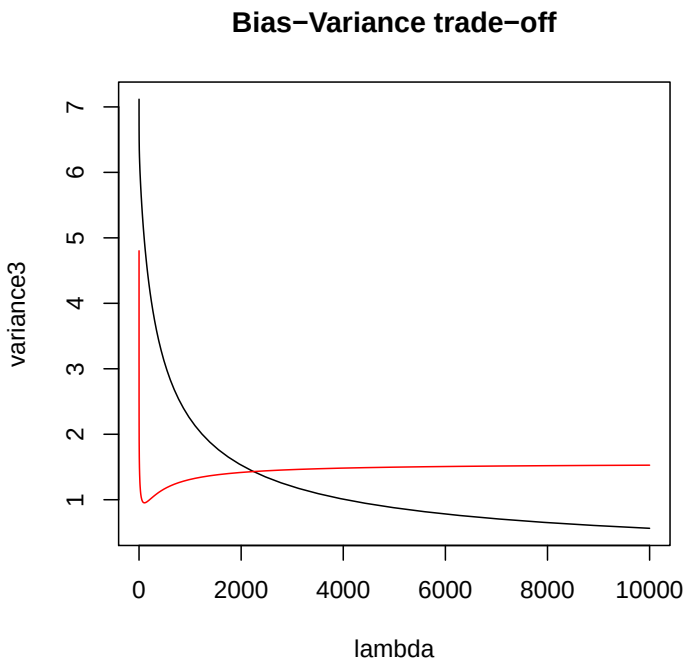

Figure 3.18: Bias Variance trade-off, squared bias(red), root squared variance(black)

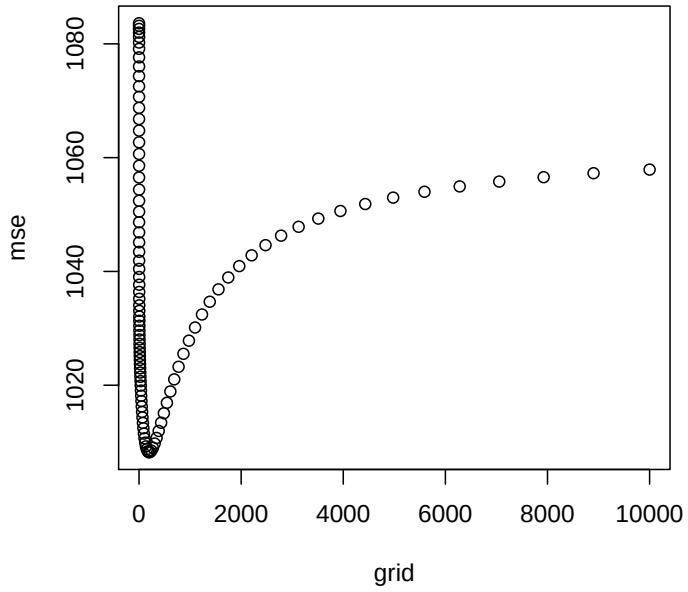

Figure 3.19: MSE against λ

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

3.7.2 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:

$$RSS + \lambda \sum_j |\beta_j^2| \tag{3.2}$$

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

```
grid = 10^seq(10,-2, length=100)
lasso.mod = glmnet(x.train, y.train, alpha=1,lambda=grid)
plot(lasso.mod)
```

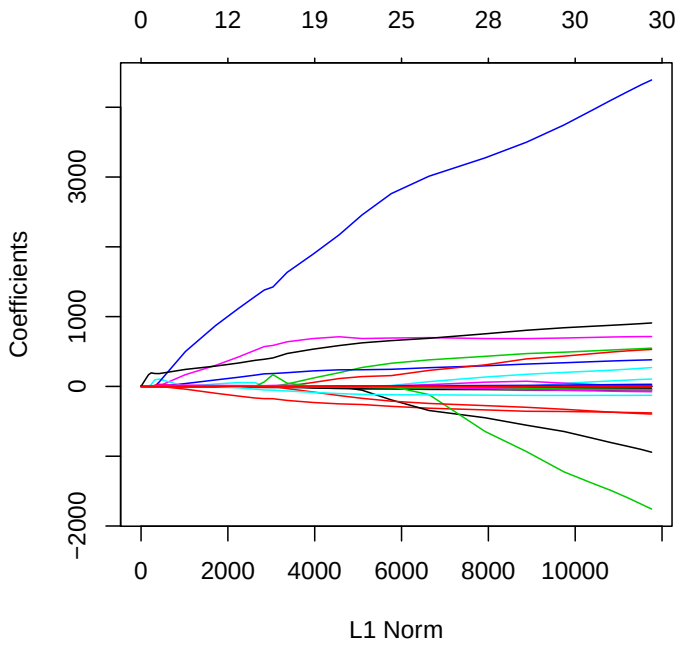

Figure 3.20: MSE against λ

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.

```
cv.out = cv.glmnet(x.train, y.train, alpha=1)
```

```
plot(cv.out)
best_lambda = cv.out$lambda.min
lasso.pred = predict(lasso.mod, s=best_lambda , newx=x.test)
mean((lasso.pred-y.test)^2)
plot(x=y.test, y=lasso.pred)
abline(0,1, col='red')
```

$$\text{best\_}\lambda = 3.20946$$
$$\text{best\_test\_MSE} = 1020.8685$$

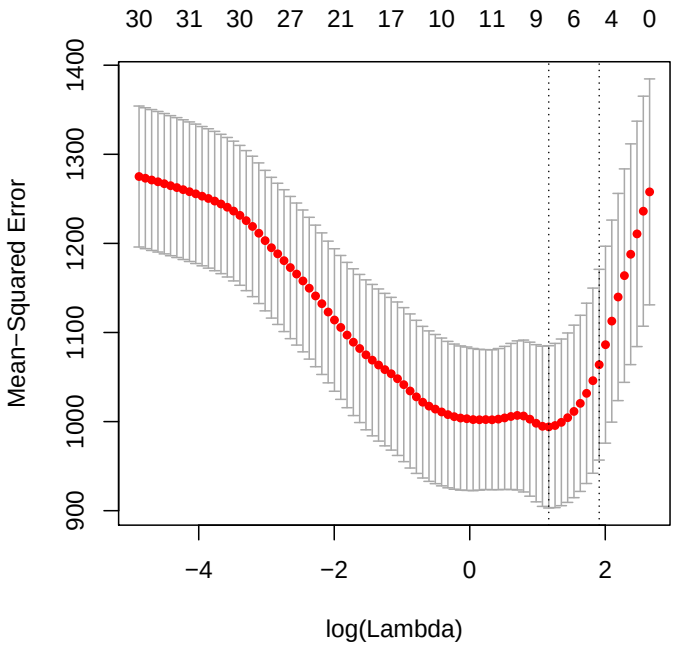

Figure 3.21: MSE against λ

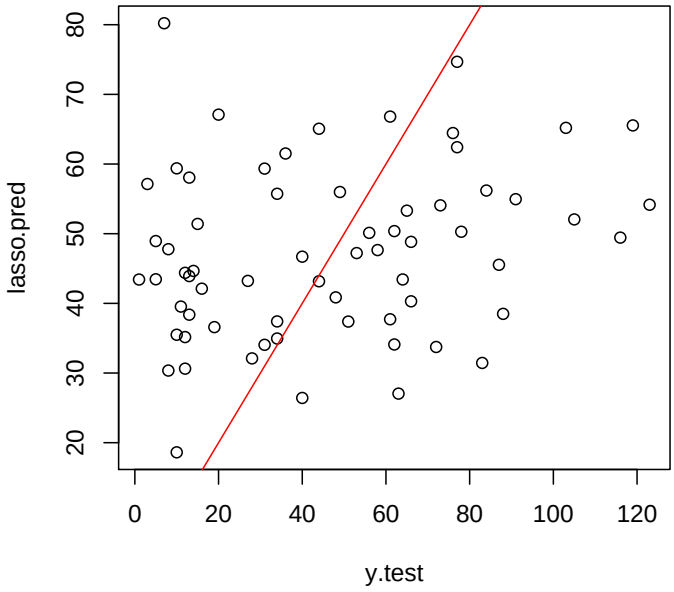

Figure 3.22: MSE against λ

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

(Intercept)	84.8180909916685
Lymph_node	0
radius_mean	0
texture_mean	-0.867374006274355
perimeter_mean	-0.344351326027745
area_mean	0
smoothness_mean	2.35445420561878
compactness_mean	0
concavity_mean	0
concave_points_mean	0
symmetry_mean	0
fractal_dimension_m...	97.7980483397943
radius_se	0
texture_se	-7.52383830040945
perimeter_se	0
area_se	0
smoothness_se	22.0540181849659
compactness_se	0
concavity_se	0
concave_points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	21.0985486091439
compactness_worst	0
concavity_worst	0
concave_points_worst	0
symmetry_worst	14.3799505743644
fractal_dimension_wo...	184.532513675901
Tumor_size	0

Figure 3.23: Lasso coefficients β_j for ($\lambda = 3.20$)

We can clearly see that Lasso performed a variable selection by setting the unselected predictors' coefficients to 0. Therefore only 10 predictors were selected out of 33.

3.8 Linear Regression with Dimension Reduction

3.8.1 Idea

In the Introduction, we mentioned the fact that some features are actually correlated. This means that they carry redundant information that make the model more complex, thus harder to fit. In this section, we will apply a Dimension Reduction method that decreases the number of features while keeping the information needed to predict `Time`.

Principal Component Analysis (PCA) is one such method. It consists in transforming the set of p features into a set of M orthogonal vectors ($M < p$) using linear transformations. The new set of vectors (called components) contains as much information as the initial set. "Information" is here defined in terms of variance, in other words, each new component should be able to explain as much variance as possible, while being orthogonal to the others.

Once the set of principal components is found, it can be used as an input to a simple linear regression to build what is called a Principal Component Regression (PCR) model. The number of components M should be taken so that the model built with PCR has a minimum error value; once again, a 10 fold cross validation method can be used to determine the best M.

PCR is available in R with the package `pls` :

```
library(pls)
model.pcr = pcr(Time ~ ., data=train_set, scale=TRUE,
    validation="CV")
```

We can then plot the MSE for each set of components :

```
validationplot(model.pcr, val.type = "MSEP")
```

According to figure 3.24, the model with only 4 components yields to the lowest MSE. The function `summary` tells us that 4 components are enough to explain 75% of the features' variance.

3.9 Models Comparaison

3.9.1 KNN neighbors and Linear Regression

USE TEST SET TO COMPARE MODEL

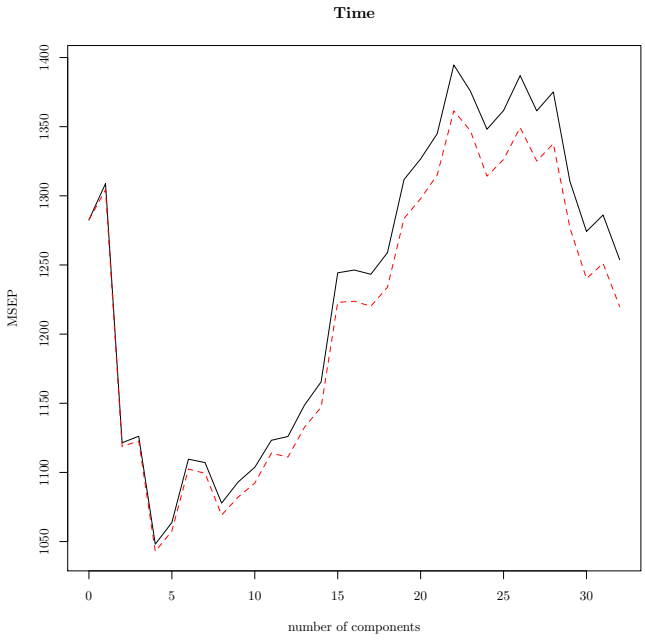

Figure 3.24: PCR