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

Part IV

Ex 2 - Breast Cancer Recurring Time

0.1 Introduction

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.

0.2 Dataset Description

We first take a look at the original 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`.

0.2.1 Time

0.2.2 Features Description

Feature Correlation

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

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

0.2.3 Data Relevance

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

TALK ABOUT COOK'S DISTANCE

According to this graph, no point is located beyond the critical Cook's boundary. This means that we can potentially use each and every patient case of our dataset to build our regression model.

0.2.4 Relation between "feature" and "time"

GRAPH : each Feature vs. Time

QQ-PLOT : Non Linearities

Heteroscedasticity

0.3 Measures to Compare Models

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

0.3.1 Some Measures

TALK ABOUT MSE, R2 AND OTHER STUFF

0.3.2 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 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.

0.4 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 x0. The KNN model first identifies the k closest points to x0 from the training data. Then estimates

0.4.1 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 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

```
1 library(FNN)
2 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.

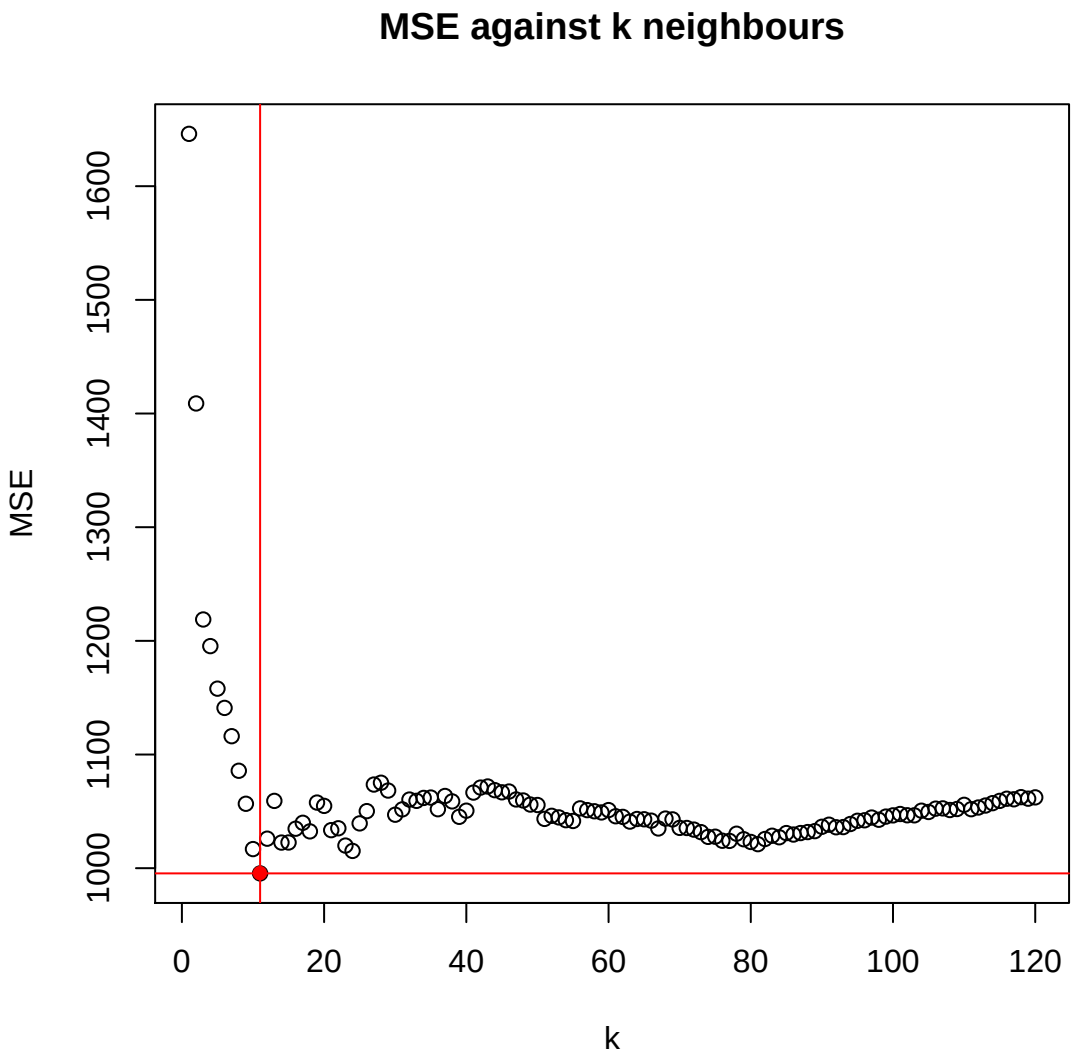

Figure 1: MSE against K neighbours

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

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

```
best_reg_test = knn.reg(train= cancer.train.x, test =
    cancer.test.x, y=cancer.train.y, k = best_k_test)
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).

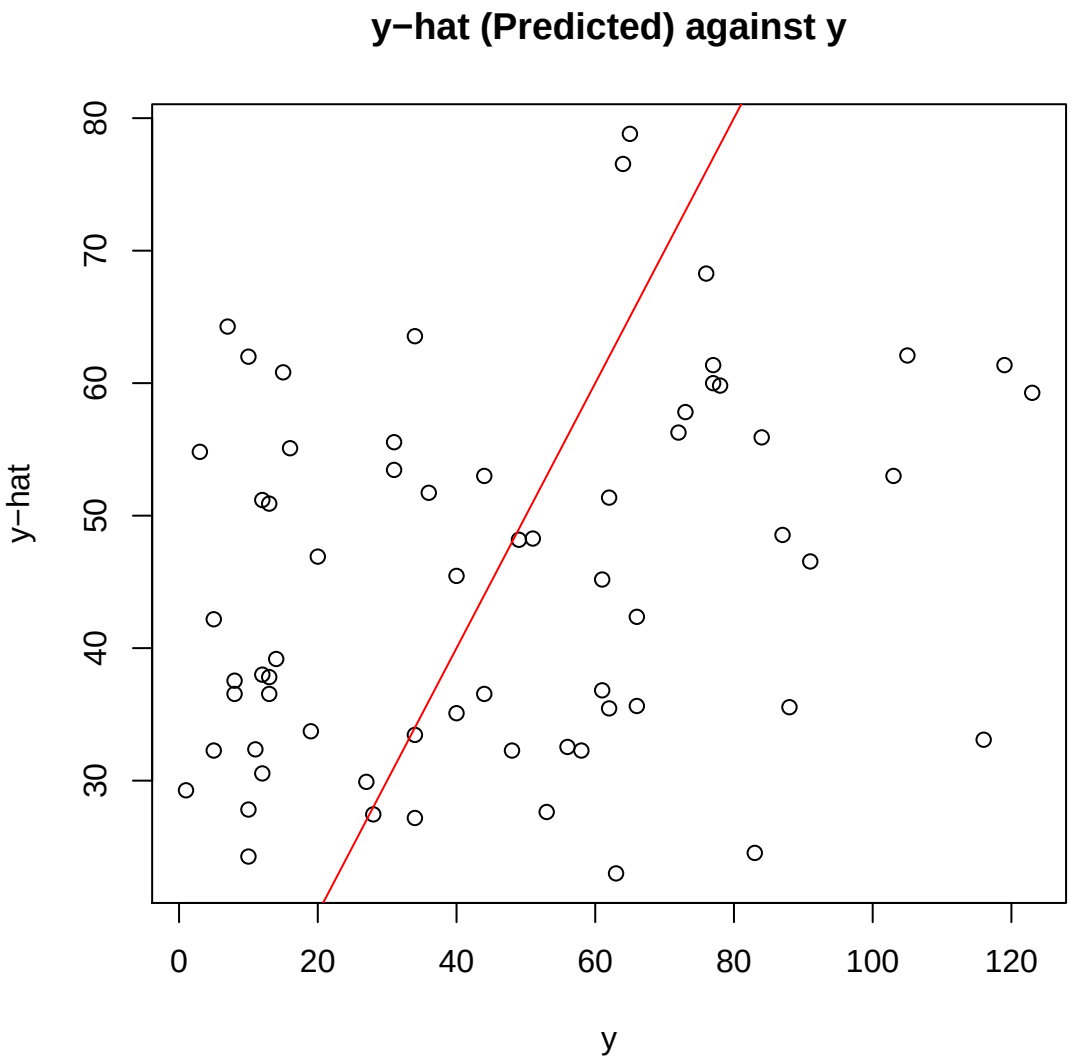

Figure 2: MSE against K neighbours

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.

0.4.2 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

```
library("kknn")
model_kknn = train.kknn(Time ~., data= cancer.train, kmax =
    30, ks = NULL, distance = 2, kernel = "optimal")
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.

```
best_reg_train = knn.reg(train= cancer.train.x, test =
    cancer.test.x, y=cancer.train.y, k = best_k_train)
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
mse= mean(errors)
```

The results are shown below:

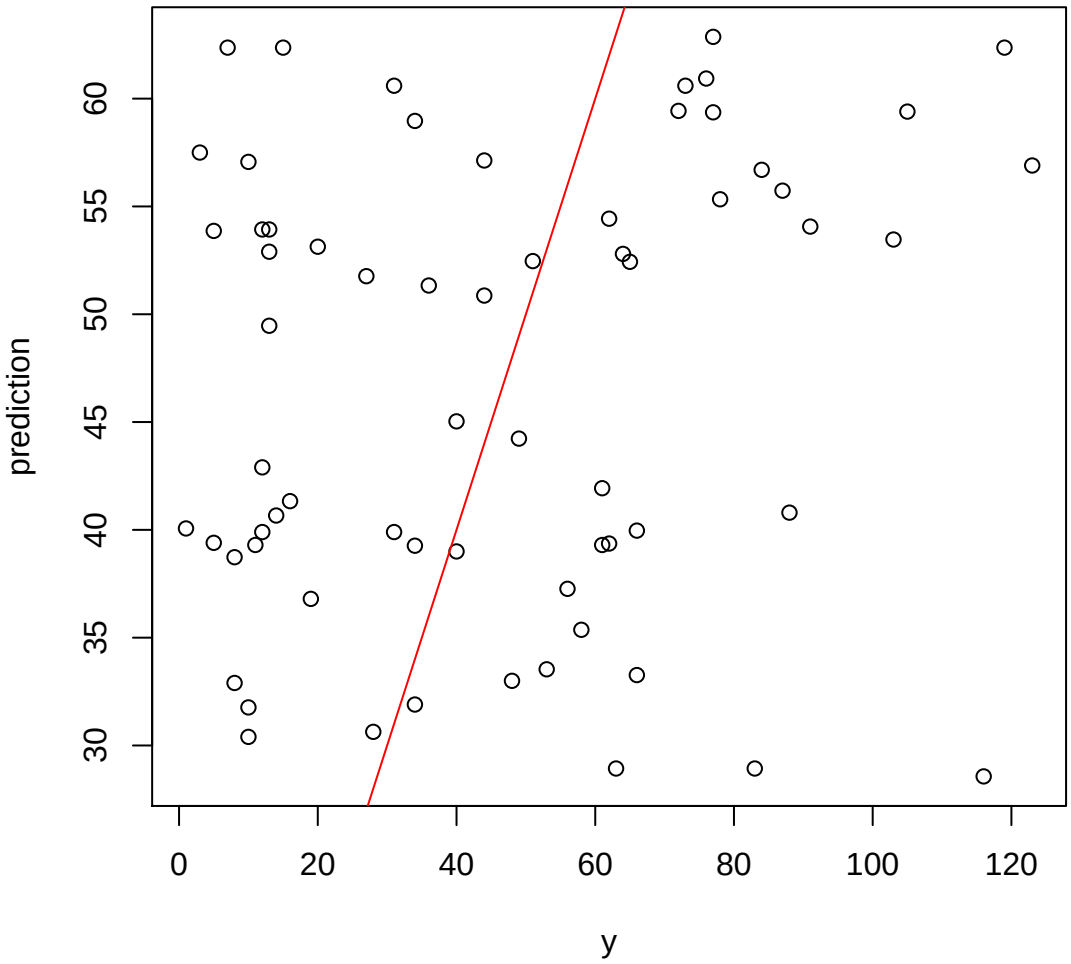

Figure 3: MSE against K neighbours

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

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.

0.5 Simple Linear Regression

0.5.1 Idea

0.5.2 Build the Model

0.5.3 Model Analysis

HIGH P VALUES

0.6 Linear Regression with Features Selection

0.6.1 Idea

0.6.2 Build the Model

EXHAUSTIVE

0.6.3 Model Analysis

0.7 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.

0.7.1 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.

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

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

$$\lambda$$

=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.

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

x.test =model.matrix(Time~.,cancer.test)[,-1]
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.

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

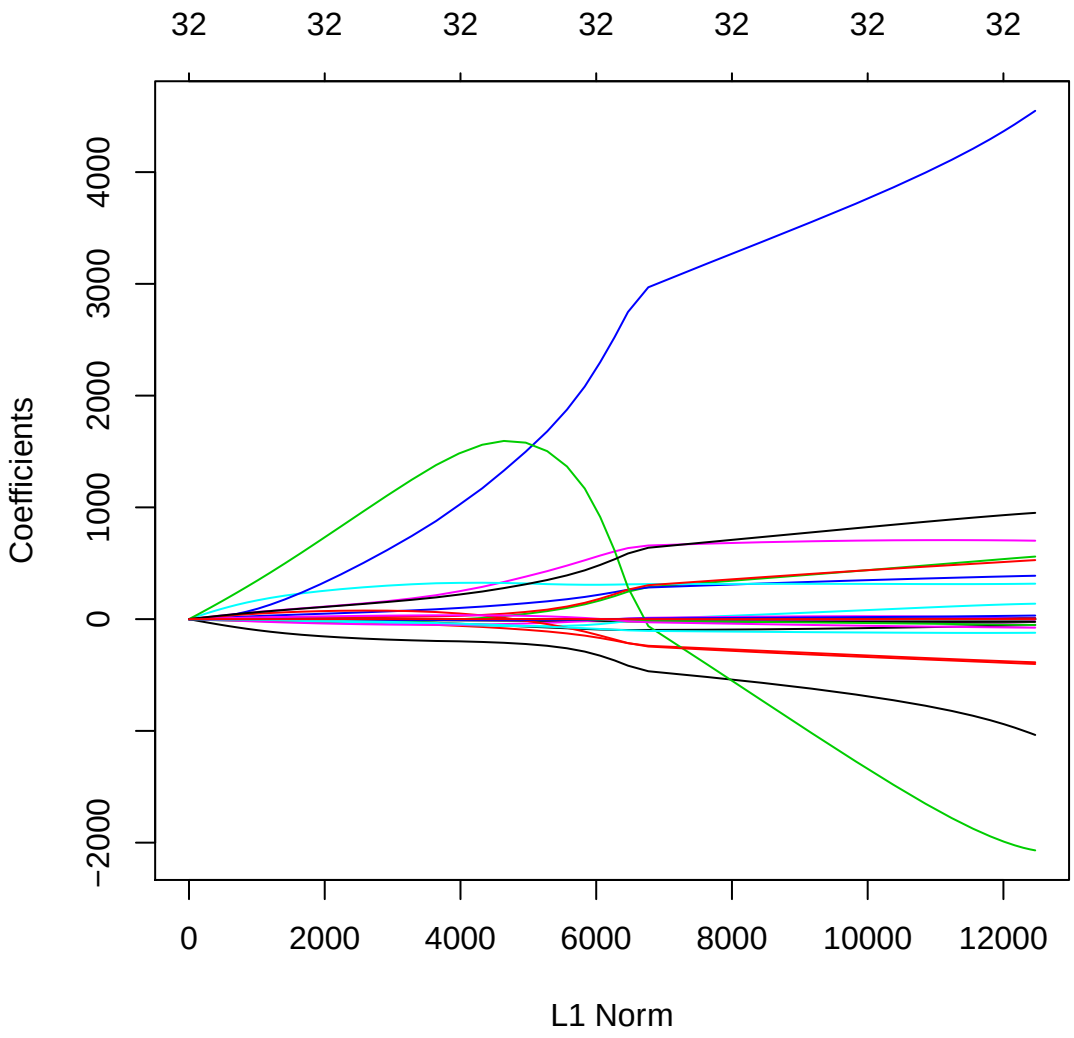

Figure 4: Coefficients β_j against L1 Norm

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.

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

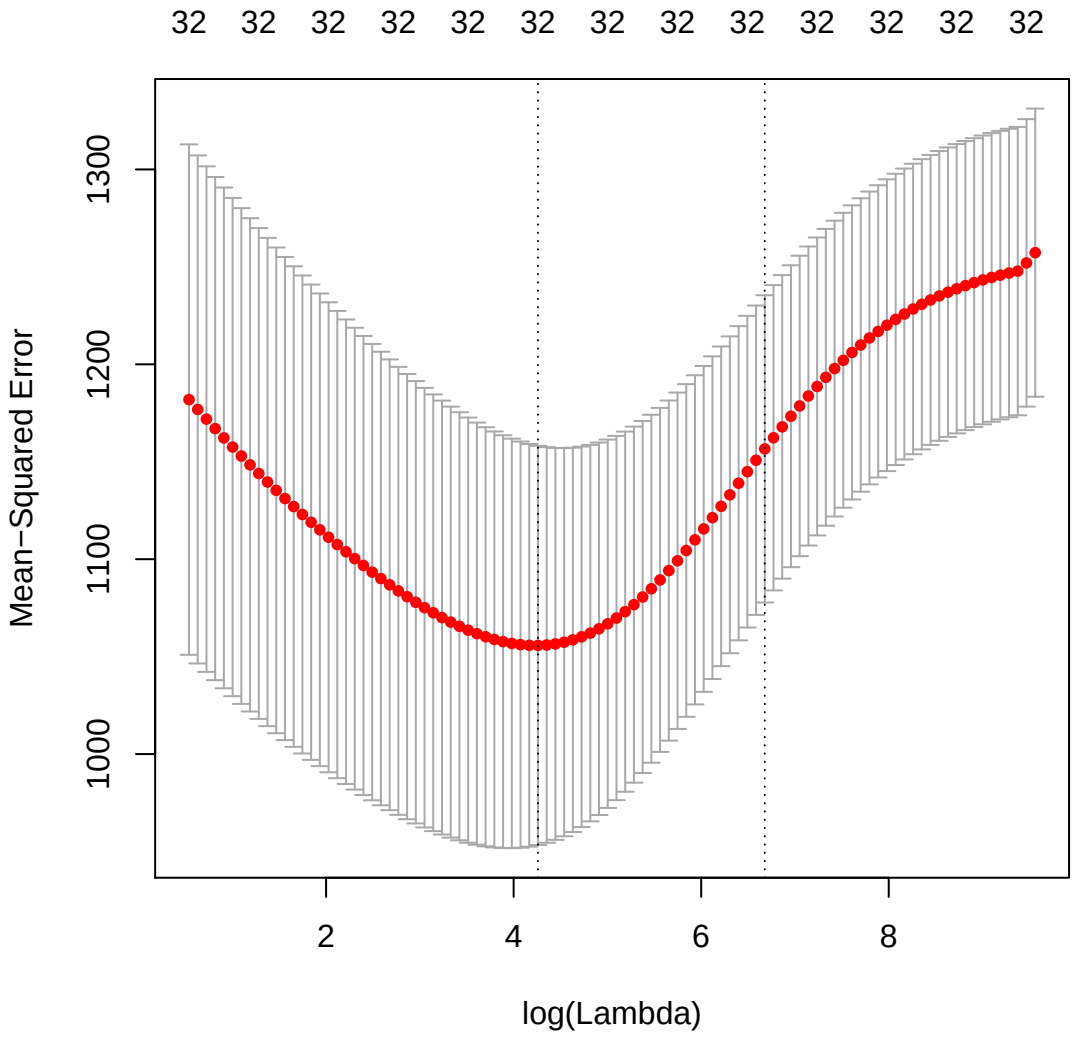

Figure 5: MSE against $\log(\lambda)$

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

```
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)
```

$$\text{best\_}\lambda = 3.522385$$
$$\text{best\_test\_MSE} = 1017.18$$

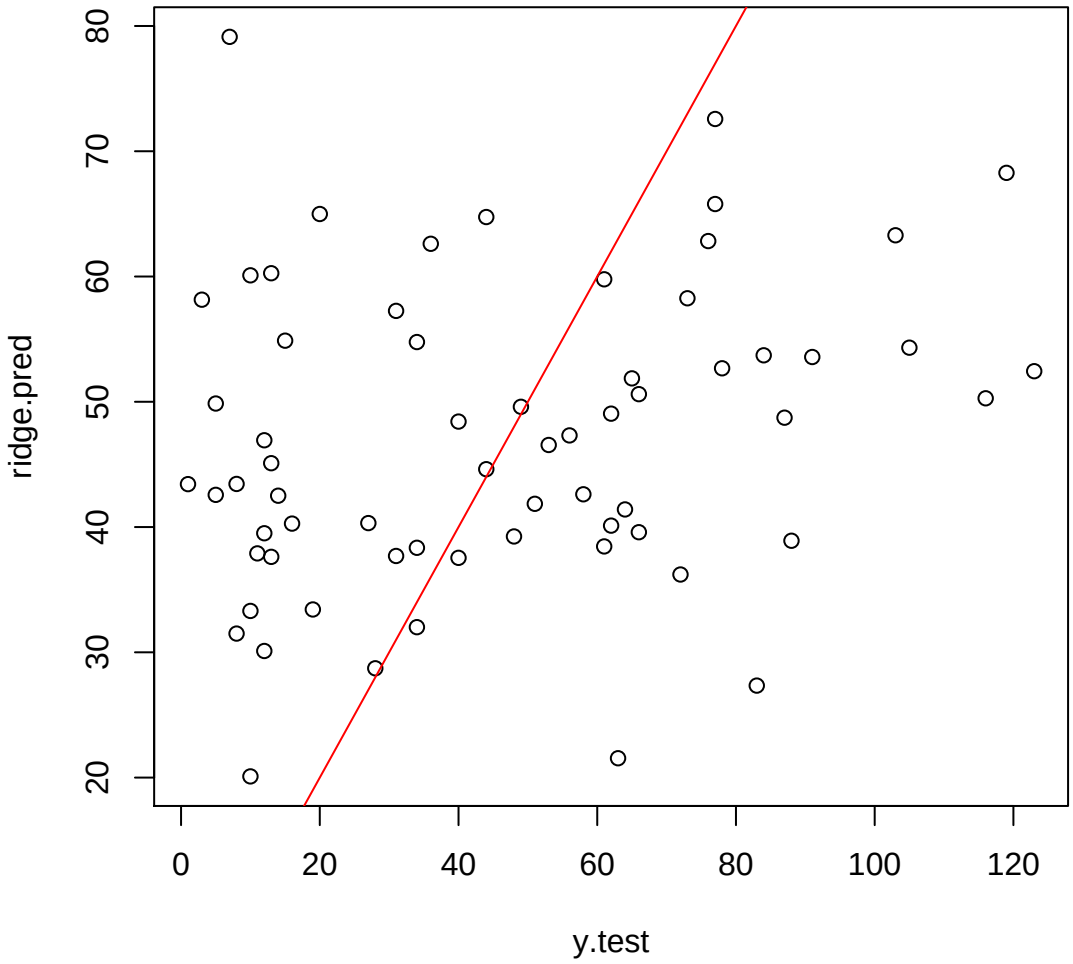

Figure 6: predicted Time $\hat{y}$ against real responses y

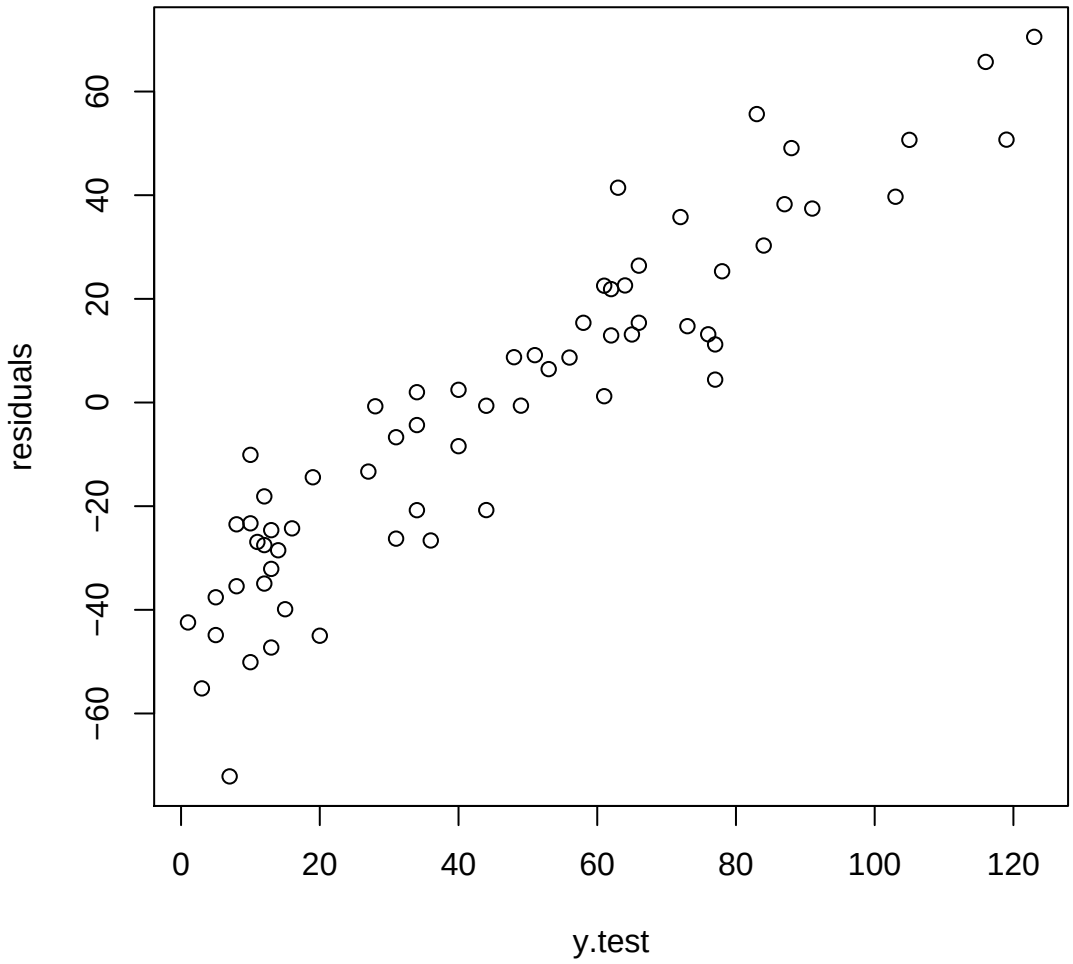

Figure 7: Residuals against Time(y)

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)	50.460029742775
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 8: 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.

```
max = 100
biais2<-rep(0,Kmax)
variance<-rep(0,Kmax)
mse<-rep(0,Kmax)
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)
```

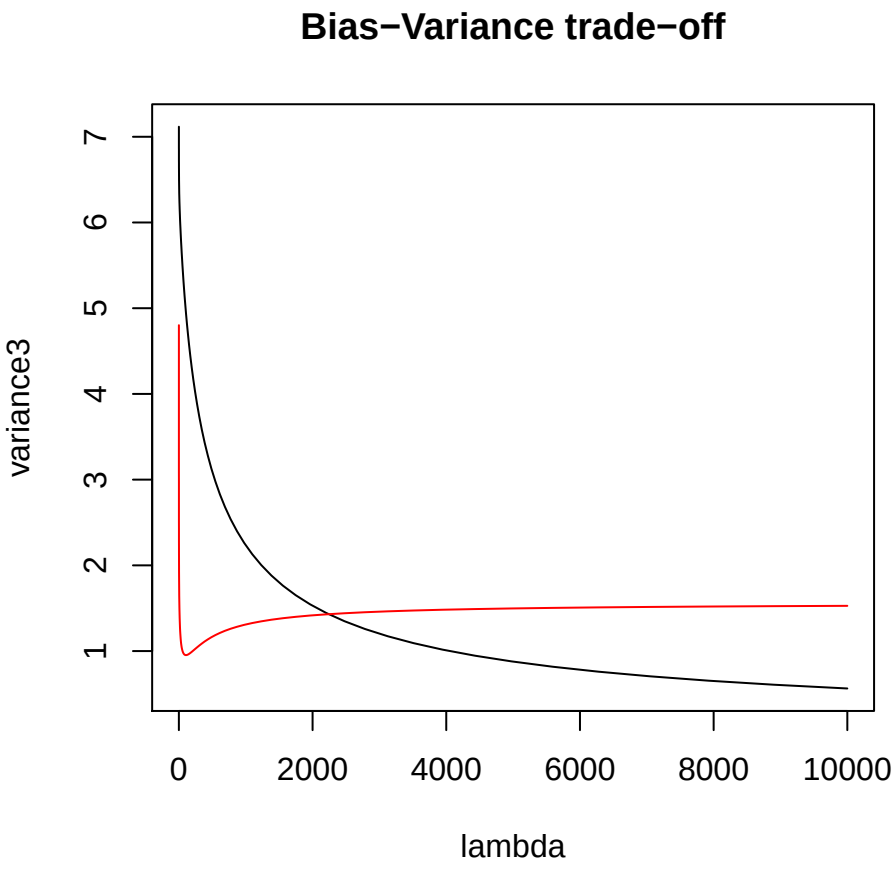

Figure 9: Bias Variance trade-off, squared bias(red), root squared variance(black)

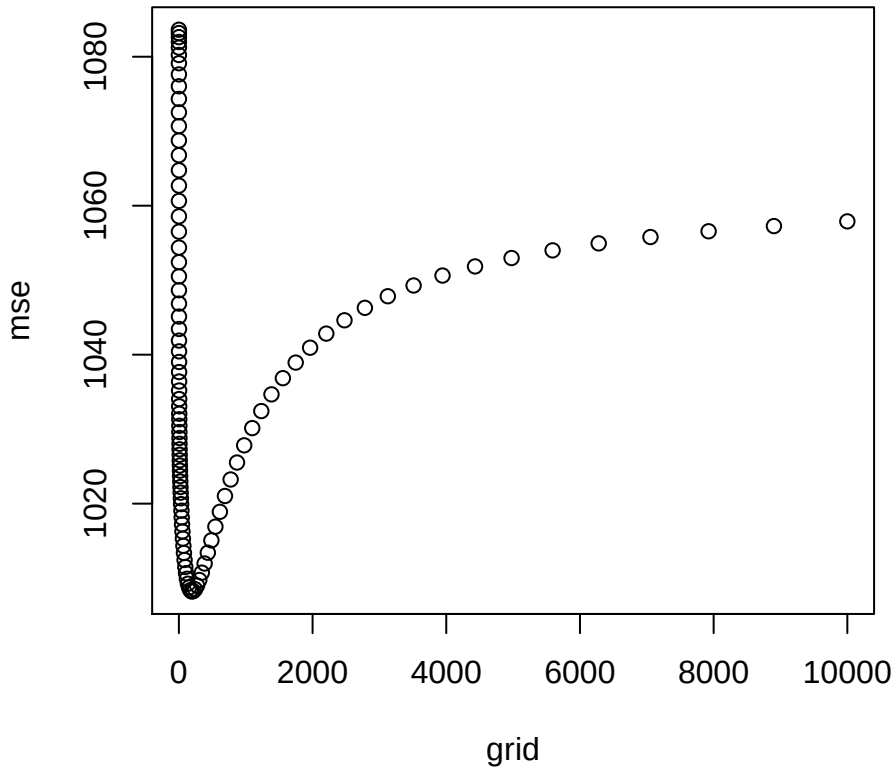

Figure 10: MSE against λ

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.

0.7.2 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:

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

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)
```

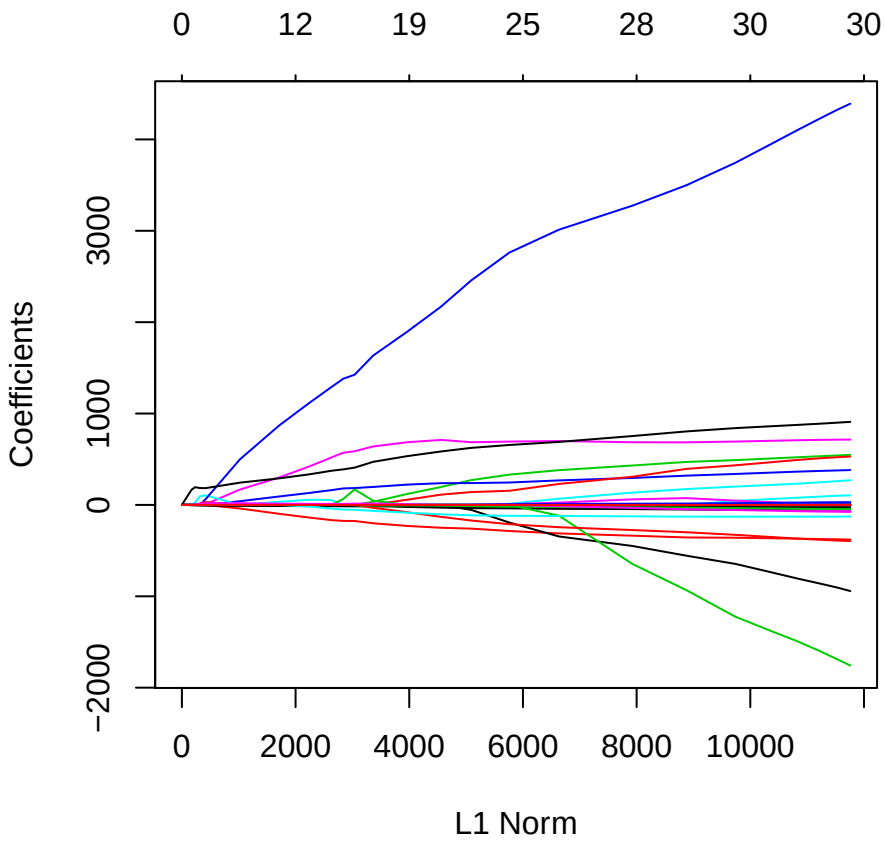

Figure 11: MSE against λ

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')
```

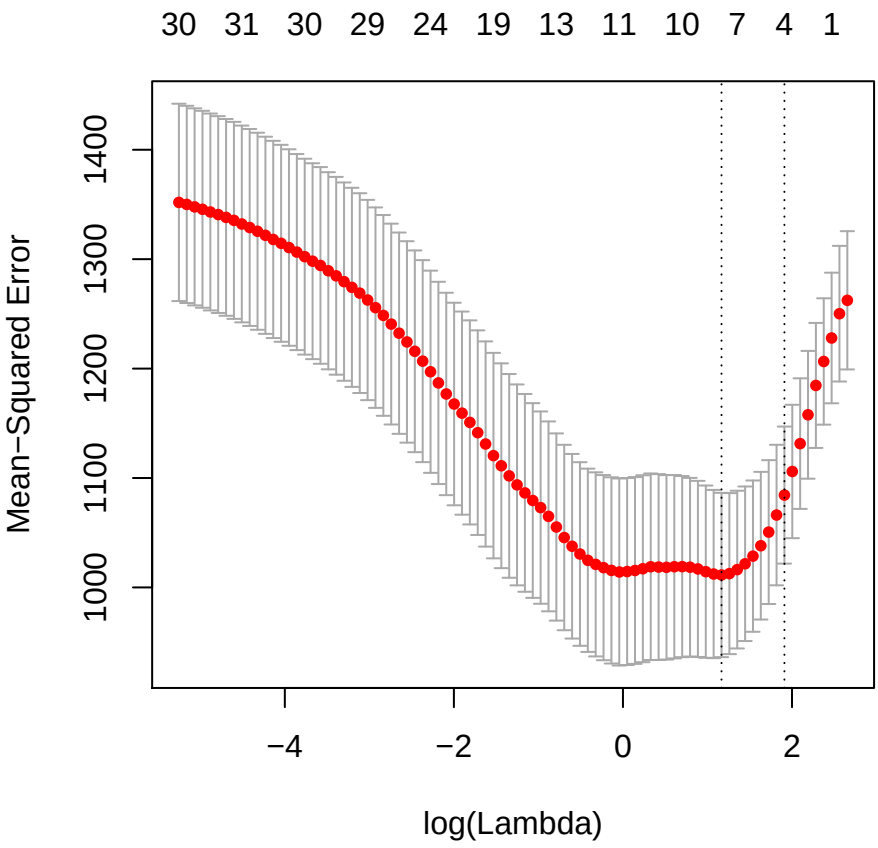

Figure 12: MSE against λ

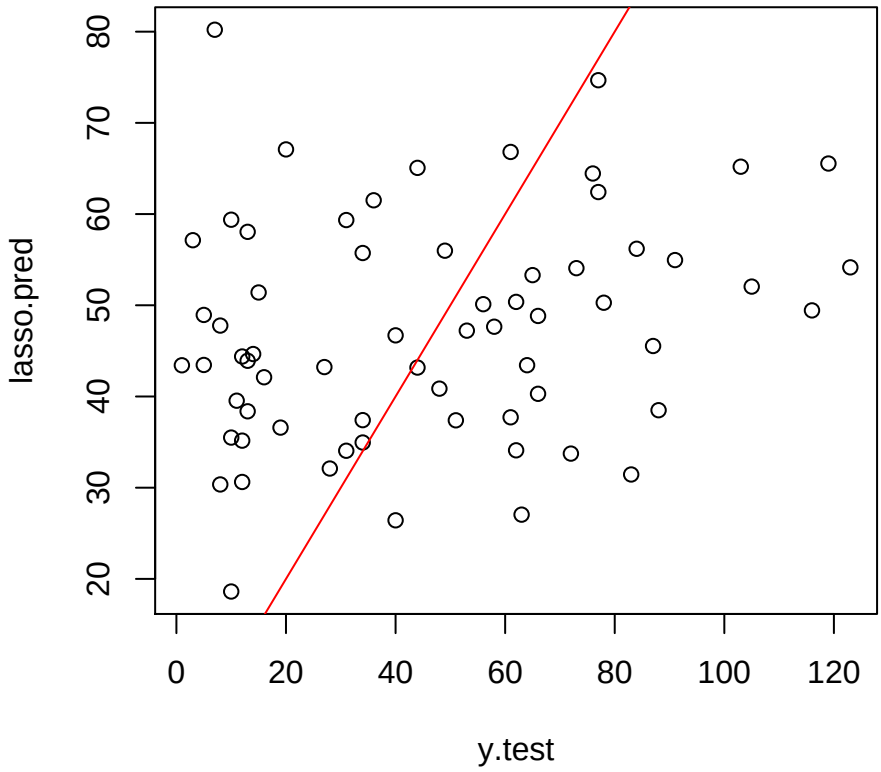

Figure 13: MSE against λ

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

(Intercept)	50.460029742775
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 14: 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

0.8 Models Comparaison

USE TEST SET TO COMPARE MODEL