Université de Technologie de Compiegne

SY19

Machine Learning

First Assignment

Aladin TALEB Zineb SLAM

November 22, 2016

Abstract

This report is an assignment for the course SY19 : Machine Learning. It is focused on two exercises : a regression problem (Chapter I) and classification problem (Chapter II) . In each part, we will describe the methodology we used to build the best model for each dataset.
The content presented in this report is the result of the theoretical and practical courses of SY19 taught by Thierry Denoeux and a literature review. We put on a side note that we both have not followed SY09, and only one of us has already done SY02, which prevented us to go deeper in the explanations and analysis sometimes but we made sure to make researches every time it was needed.

Contents

0.1 Abbreviations 3

1 Breast Cancer Recurring Time **4**
1.1 Context 4
1.2 Dataset Description 4
1.2.1 Time 4
1.2.2 Features Description 4
1.2.3 Data Relevance 6
1.2.4 Relation between "feature" and "time" 7
1.3 Measures to Compare Models 9
1.3.1 Some Measures 9
1.3.2 Data Split 9
1.4 K-nearest neighbors (KNN) 11
1.4.1 Knn Model 11
1.4.2 The Validation Set Approach 14
1.5 Simple Linear Regression 18
1.5.1 Idea 18
1.5.2 Model Performance 19
1.6 Linear Regression with Features Selection 20
1.6.1 Idea 20
1.6.2 Model Performance 22
1.7 Linear Regression with Regularization 23
1.7.1 Ridge Regression 23
1.7.2 Lasso Regression 32
1.8 Linear Regression with Dimension Reduction 38
1.8.1 Idea 38
1.9 Models Comparison 40
1.9.1 MSE and Residuals 40
1.9.2 KNN neighbors and Linear Regression 42
1.9.3 Best subset Selection and Linear Regression Regularizations 43
1.9.4 Dimension Reduction 46

2 Phoneme Recognition **47**
2.1 Context 47
2.2 Dataset Description 47
2.3 Measure to Compare Models 48
2.4 Classification 48
2.4.1 LDA - Linear Discriminant Analysis 48
2.4.2 QDA - Quadratic Discriminant Analysis 52
2.4.3 Logistic Regression 54
2.5 Models Comparison 55

0.1 Abbreviations

MSE	Mean Squared Error
RSE	Resiual Standard Error
RSS	Residual Sum of Squares
AIC	Akaike information criterion
BIC	Bayesian information criterion
R^2	Adjusted R^2
LR	Linear Regression
CV	Cross Validation
LOOCV	Leave One Out Cross Validation

Chapter 1

Breast Cancer Recurring Time

1.1 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.

1.2 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.

1.2.1 Time

Let's first describe the distribution of the variable `Time`. To do so, we can use the R functions `boxplot` (figure 1.1) and `hist` (figure 1.2). 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.

1.2.2 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

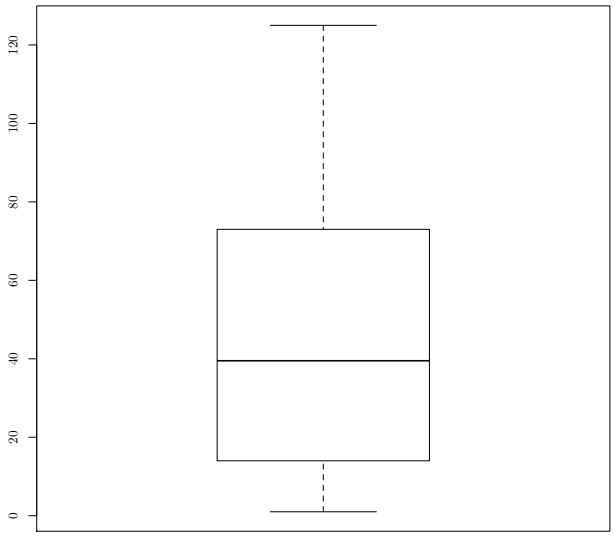

Figure 1.1: Box Plot

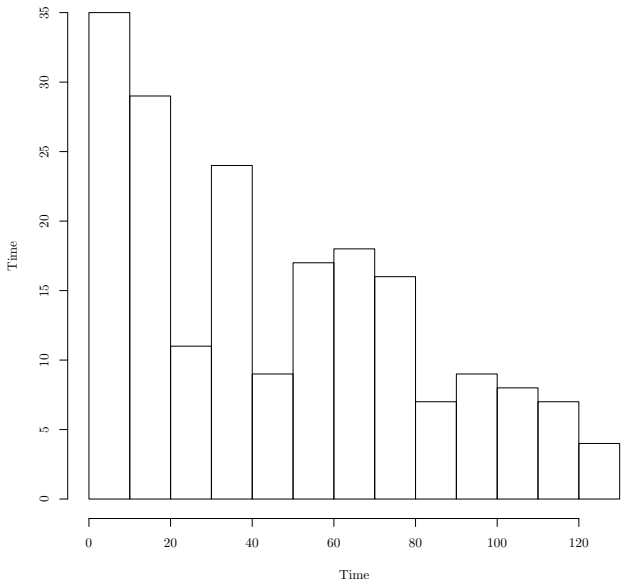

Figure 1.2: Histogram

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.

1.2.3 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)
```

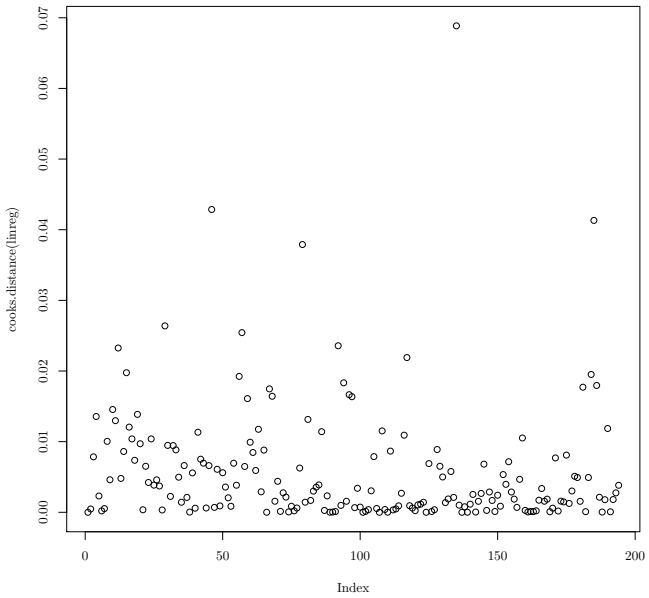

Figure 1.3: Cook's Distance

According to this plot (figure 1.3), 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.

1.2.4 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 1.4. 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.

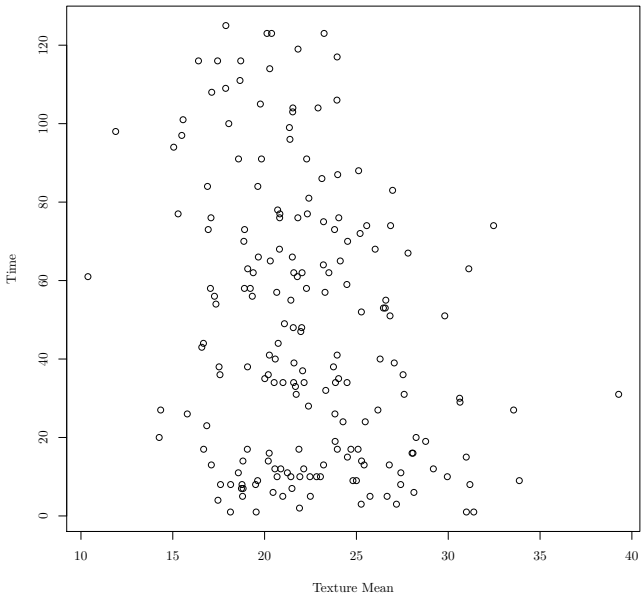

Figure 1.4: Plot of `Time` against `Texture Mean`

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 1.5) :

```
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 1.6). 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.

1.3 Measures to Compare Models

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

1.3.1 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.

1.3.2 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;

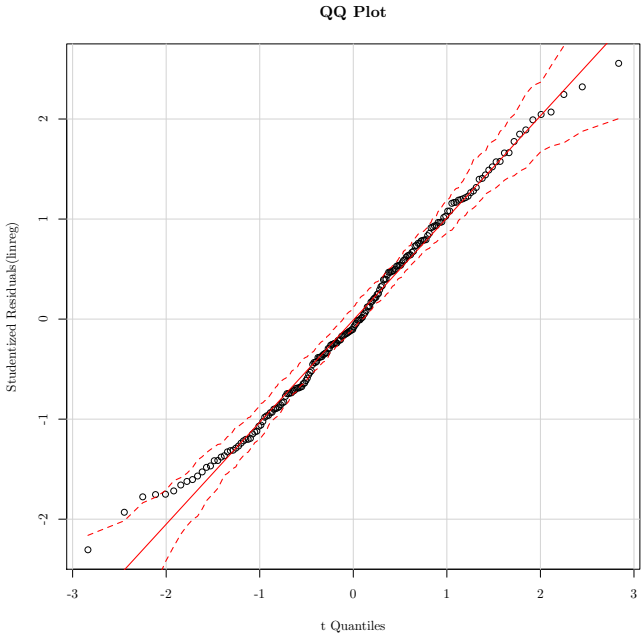

Figure 1.5: QQ-Plot

- 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]
```

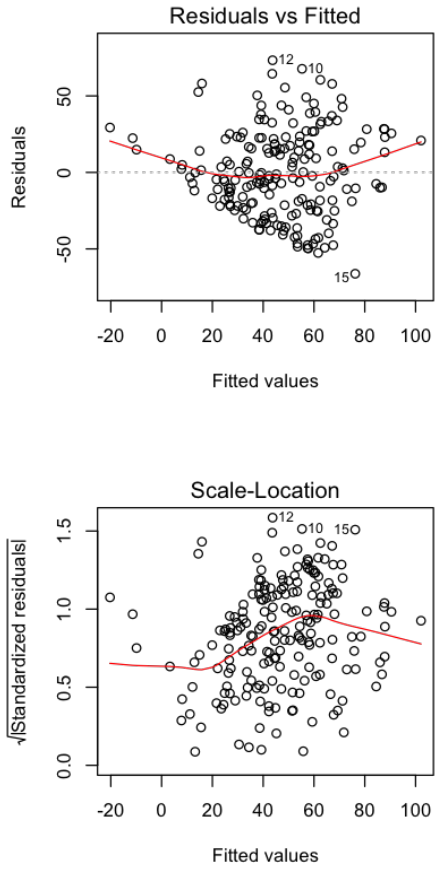

Figure 1.6

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

1.4 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.

1.4.1 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)
{
model.knn = knn.reg(train=train_set.x, test=test_set.x,
    y=train_set.y, k=k)
MSE[k] = mean((test_set.y - model.knn$pred)^2)
}

model.knn.best.k = which.min(MSE)
model.knn.best.MSE = MSE[model.knn.best.k]
```

The graph below (figure 1.7) 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.

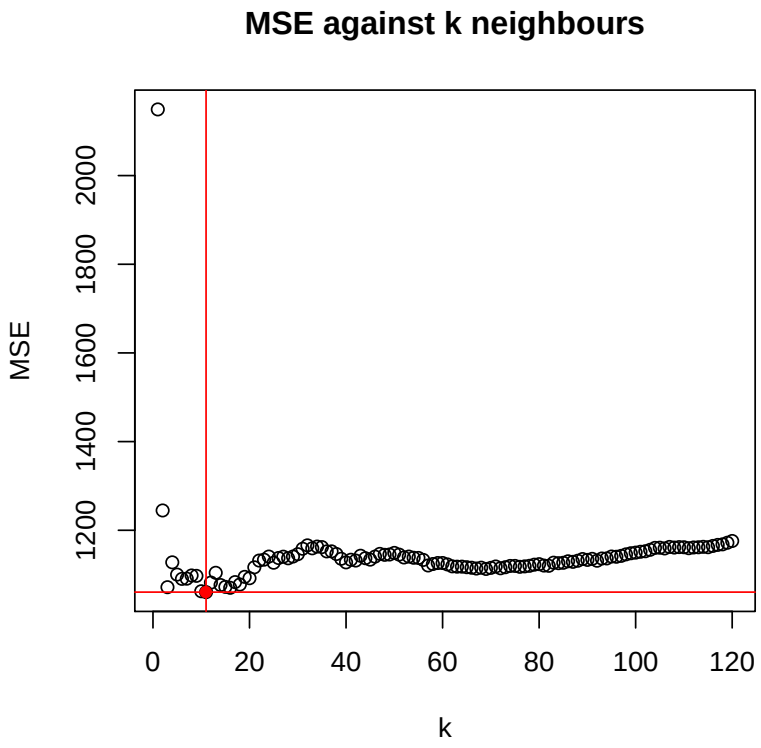

Figure 1.7: MSE against K neighbours *(minimum in red)*

We are looking for the k neighbors that has the smallest MSE so we use the function `which.min` that returns the index of the minimum value of MSE.

$$\text{model.knn.best.k} = 11$$
$$\text{model.knn.best.MSE} = 1060.46$$

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 as the residuals. The corresponding code is shown below:

```
1 model.knn.best = knn.reg(train=train_set.x,
      test=test_set.x, y=train_set.y, k=model.knn.best.k)
2 plot(test_set.y,model.knn.best$pred, xlab='y',
      ylab='y-hat', main='y-hat (Predicted) against y')
3 abline(0,1, col='red')
4
5 model.knn.best.residuals = test_set.y - model.knn.best$pred
6
```

```r
hist(model.knn.best.residuals, freq=FALSE,
    main="Distribution of Residuals in Knn best case")
residuals.mean = mean(model.knn.best.residuals)
residuals.stdev = sqrt(var(model.knn.best.residuals))
curve(dnorm(x, mean=residuals.mean, sd=residuals.stdev),
    col="darkblue", lwd=2, add=TRUE, yaxt="n")
```

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).

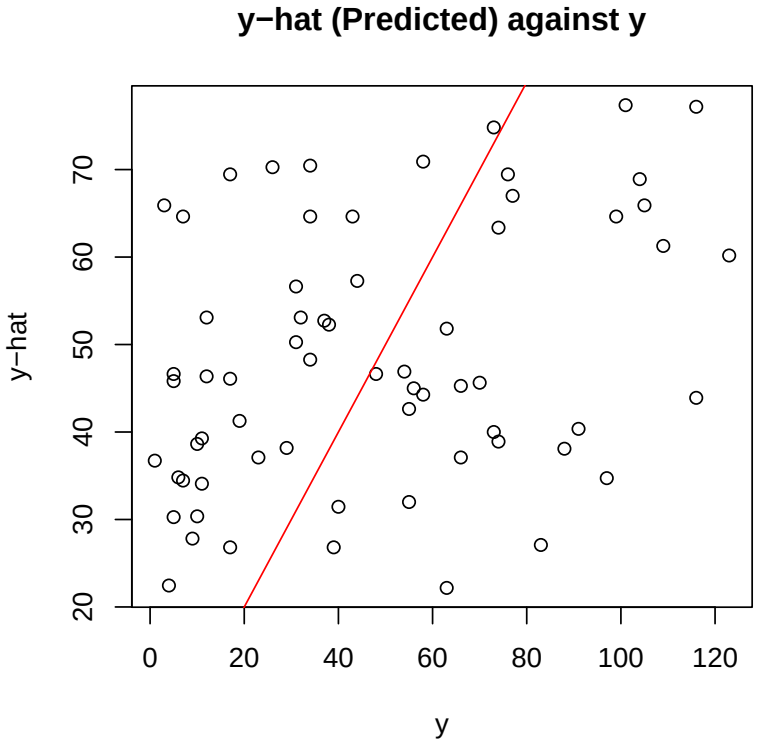

Figure 1.8: Predicted values against the real values

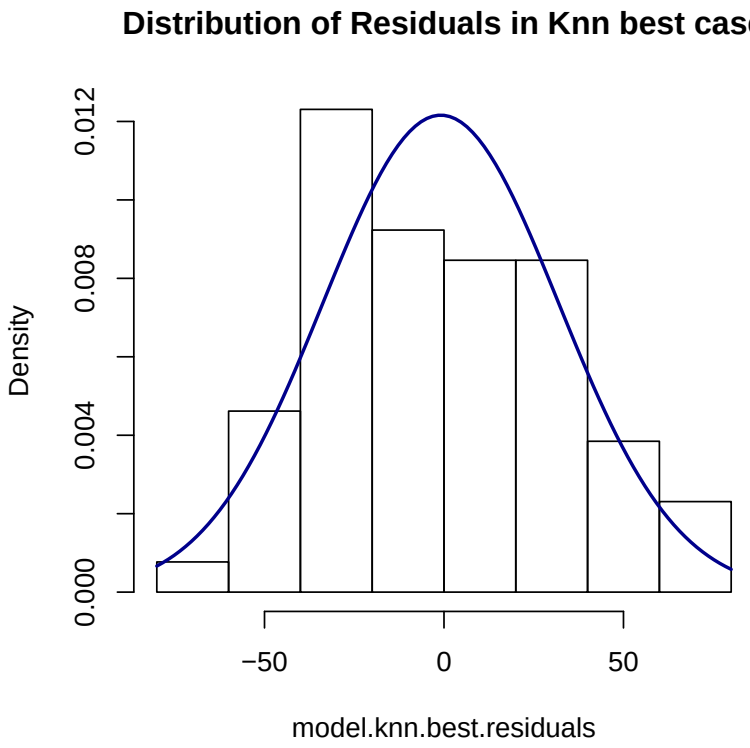

Figure 1.9: Distribution of Residuals in Knn best case

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.

1.4.2 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= train_set, kmax =
    30, ks = NULL, distance = 2, kernel = "optimal")
model.kknn.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.

```
library("kknn")
model.kknn = train.kknn(Time ~., data= train_set, kmax =
    30, ks = NULL, distance = 2, kernel = "optimal")
model.kknn.best.k = model.kknn$best.parameters$k

model.kknn.best = knn.reg(train=train_set.x,
    test=test_set.x, y=train_set.y, k=model.kknn.best.k)
plot(test_set.y, model.kknn.best$pred, xlab='y',
    ylab='prediction')
abline(0,1, col='red')

residuals = test_set.y - model.kknn.best$pred
errors = residuals^2
model.kknn.best.MSE = mean(errors)

hist(residuals, freq=FALSE, main="Distribution of Residuals
    in Knn LOOCV best case")
residuals.mean = mean(residuals)
residuals.stdev = sqrt(var(residuals))
curve(dnorm(x, mean=residuals.mean, sd=residuals.stdev),
    col="darkblue", lwd=2, add=TRUE, yaxt="n")
```

$$\text{model.kknn.best.k} = 30$$
$$\text{model.kknn.best.MSE} = 1146.05$$

We run a the same plot again with the predicted reponses against the real ones 1.10 and the histogram of the residuals 1.11. .

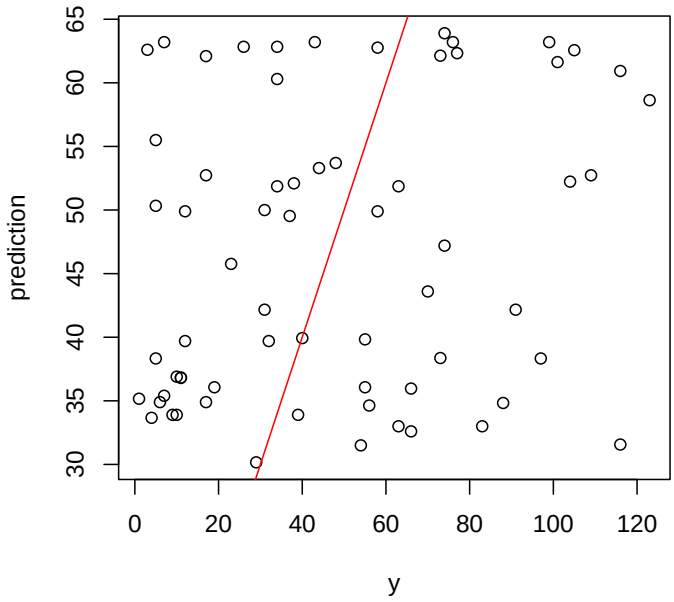

Figure 1.10: Predicted values against Real values with LOOCV

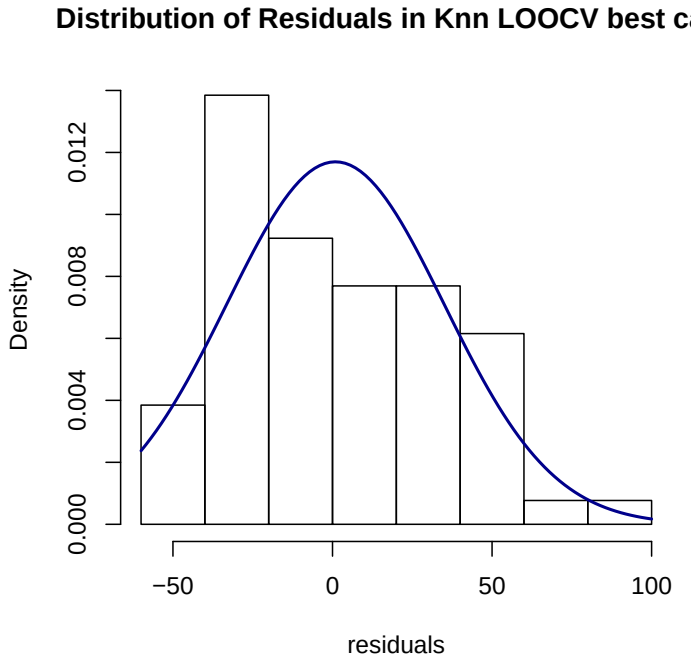

Figure 1.11: Distribution of Residuals in Knn with LOOCV

Model Analysis

One might argue that the prediction was not improved with the LOOCV since the test MSE obtained is higher that the MSE we obtained when we computed using the test set directly. However this result was expected since in the first approach we run our knn directly on the test so we underestimated the MSE, in other words the first MSE is specific to that test set of data (biased), in contrast to our the LOOCV model. In other words, in general the $k = 30$ will guarantee a smaller MSE than the $k = 11$ on any test observation. The LOOCV has also an advantage over the first approach because it will always give the same result of best k.

1.5 Simple Linear Regression

1.5.1 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)
```

1.5.2 Model Performance

This model has a MSE approximatively equal to 1285. The raw residuals distribution is pictured on figure 1.12, it shows a very spread out distribution that should be improved.

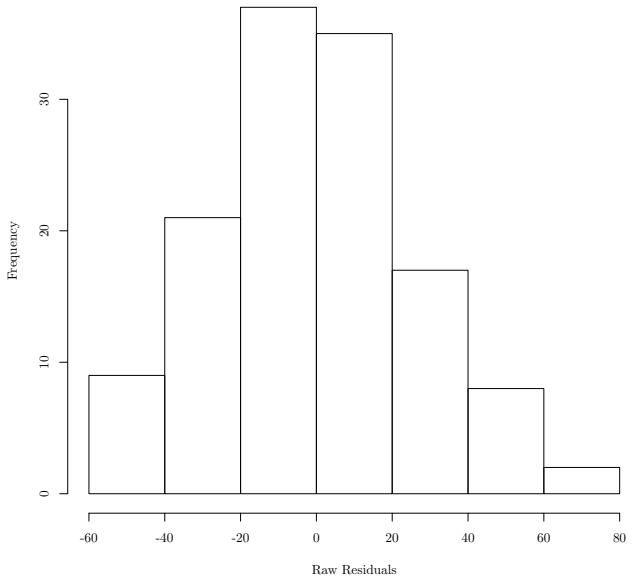

Figure 1.12

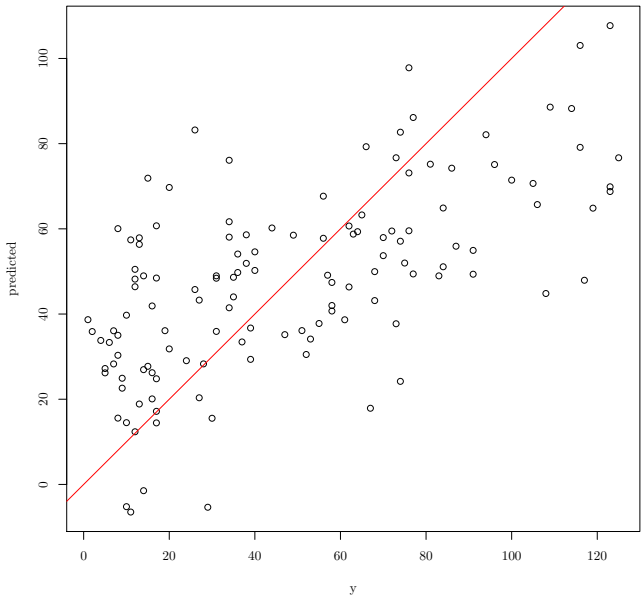

Figure 1.13

1.6 Linear Regression with Features Selection

1.6.1 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 our case we use the BIC measurements.

$$BIC = \frac{1}{n}(RSS + log(n)d\hat{\sigma}^2)$$

The BIC will be smaller when the RSS is small, so we select the model with the

smallest BIC. The result is shown on figure 1.14.

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

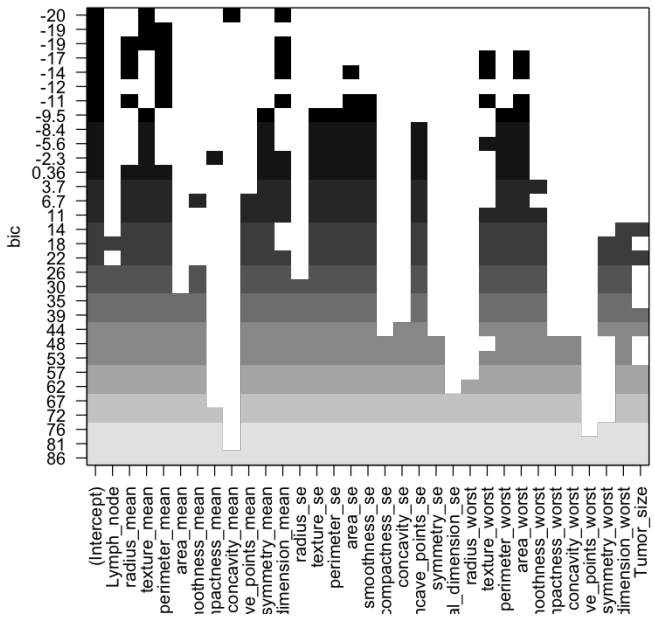

Figure 1.14: BIC-score of each feature subset

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`

1.6.2 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 1.15, is not as spread out as the previous linear regression model.

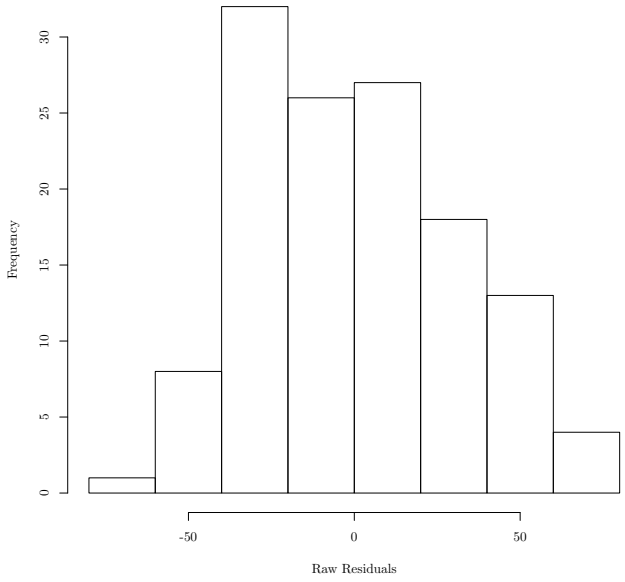

Figure 1.15

1.7 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.

1.7.1 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:

$$RSS + \lambda \sum_{j} \beta_j^2 \tag{1.1}$$

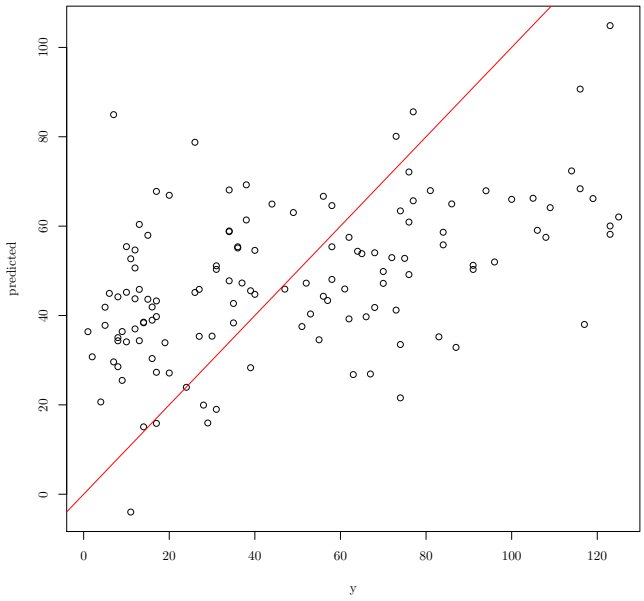

Figure 1.16

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 1.1.
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.

Model Performance

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.

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

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

```
scale(test_set.x, center = TRUE, scale = TRUE)
```

The glmnet function takes in parameter the train data, the parameter **alpha** indicates whether it is a Ridge or Lasso regression (alpha=0 for Ridge). By default `glmnet` chooses an automatic range of λ, however we chose a wide range with the function **grid** that takes the minimum value the maximum and the length and returns a grid. We chose $\lambda \in [10^{-2}, 10^{10}]$ to cover all possibilities.

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

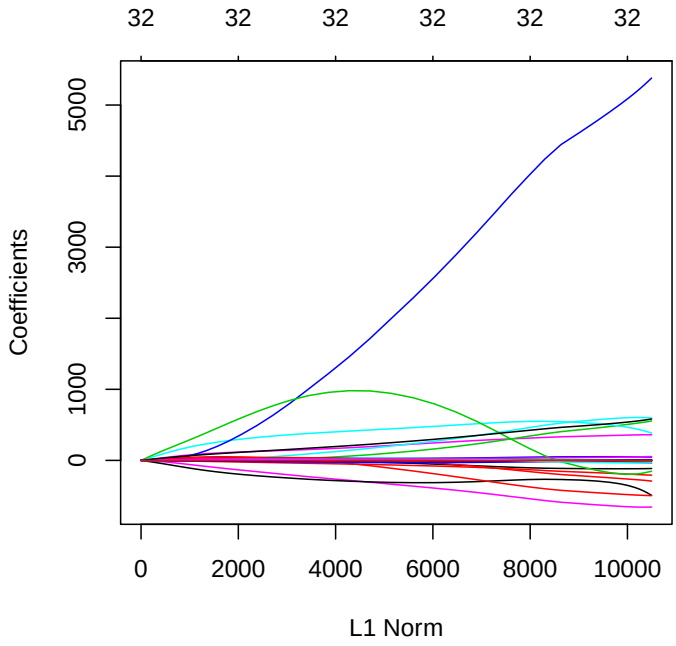

Figure 1.17: Coefficients β_j against L1 Norm

Each curve of the 1.17 corresponds to the ridge regression coefficient estimate for one of the 33 predictors, plotted against $L1Norm$. It is to be noted that the higher the L1 Norm, the closer to zero 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.

```
model.ridge.cv.out = cv.glmnet(train_set.x,
    train_set.y,alpha=0)
model.ridge.best.lambda = model.ridge.cv.out$lambda.min
plot(model.ridge.cv.out)
```

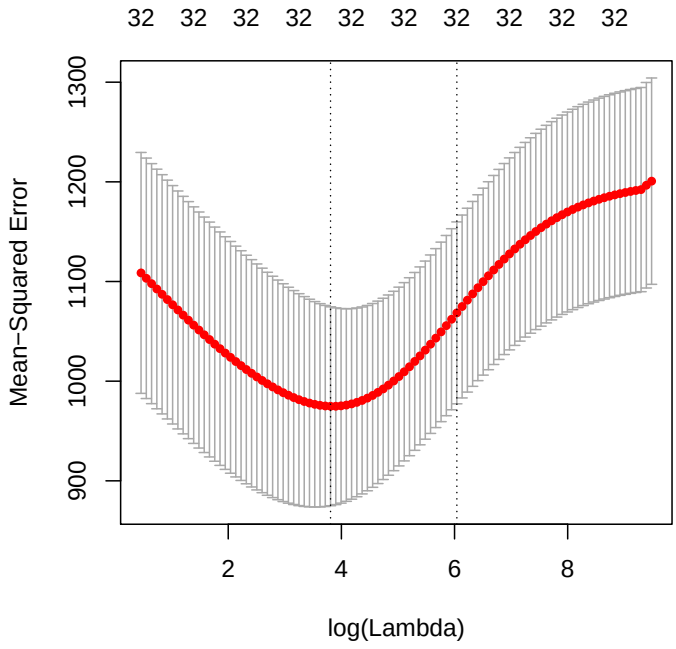

Figure 1.18: MSE against $\log(\lambda)$

The value λ that yields to the smallest MSE is shown in the graph 1.18 near $log(\lambda) = 4$ and $log(\lambda) = 5$

```
model.ridge.best = glmnet(train_set.x, train_set.y,
    lambda=model.ridge.best.lambda, alpha=0)
model.ridge.best.pred = predict(model.ridge.best,
    s=model.ridge.best.lambda, newx=test_set.x)

residuals = test_set.y - model.ridge.best.pred
errors = residuals^2
model.ridge.best.MSE = mean(errors)

plot(x=test_set.y, y=model.ridge.best.pred)
abline(0,1, col='red')
```

```
10
11 hist(residuals, freq=FALSE, main="Distribution of Residuals
       in Ridge")
12 residuals.mean = mean(residuals)
13 residuals.stdev = sqrt(var(residuals))
14 curve(dnorm(x, mean=residuals.mean, sd=residuals.stdev),
       col="darkblue", lwd=2, add=TRUE, yaxt="n")
```

model.ridge.best.lambda = 45.07
model.ridge.best.MSE = 1063.3

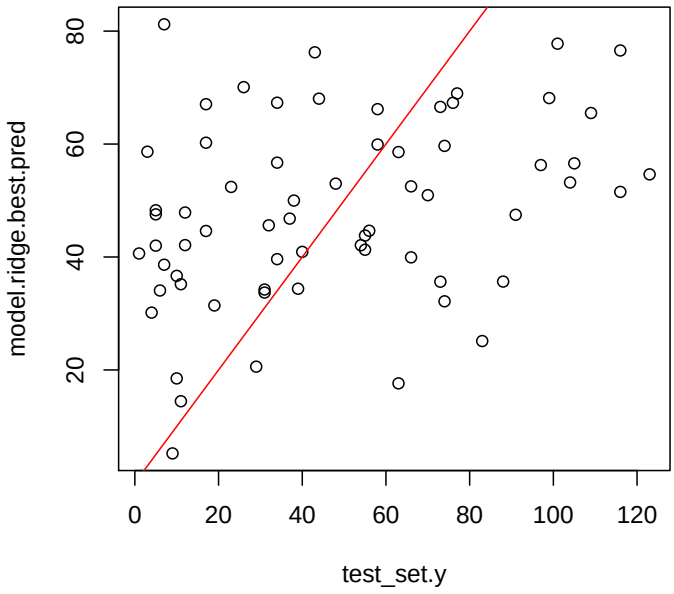

Figure 1.19: predicted Time $\hat{y}$ against real responses y

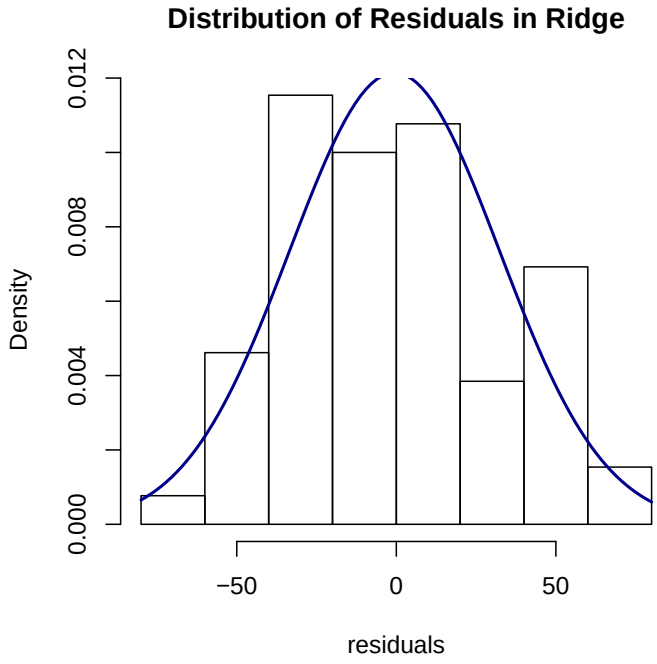

Figure 1.20: Residuals against Time(y)

We have a smaller MSE compared to Linear Regression. However the scattered $\hat{y}$ in the extremes let us think that maybe the model is not really linear. The residuals in 1.20 are getting centered and it looks more like a normal distribution.

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

```
predict(model.ridge, type="coefficients",
    s=model.ridge.best.lambda)[1:33,]
```

(Intercept)	62.5015644263887
Lymph_node	-0.100938631782285
radius_mean	-0.49162598035402
texture_mean	-0.546588424261081
perimeter_mean	-0.0736547875729137
area_mean	-0.00430679044764489
smoothness_mean	96.8117931071261
compactness_mean	4.59693750491085
concavity_mean	-14.6877128128471
concave_points_mean	-6.59290982908008
symmetry_mean	33.6374215458585
fractal_dimension_m...	237.875435541426
radius_se	-0.554934245779119
texture_se	-4.18461238448654
perimeter_se	-0.148476465484444
area_se	-0.0120040219753369
smoothness_se	150.189441839833
compactness_se	6.11836242882208
concavity_se	-88.7802623855958
concave_points_se	-146.9592219009
symmetry_se	26.0064126060648
fractal_dimension_se	401.735755303345
radius_worst	-0.171542834544902
texture_worst	-0.33291548057428
perimeter_worst	-0.0333700857746032
area_worst	-0.00111259498281325
smoothness_worst	49.8677352921429
compactness_worst	4.02281481898061
concavity_worst	-7.31642217426913
concave_points_worst	5.14294581484783
symmetry_worst	17.695513572159
fractal_dimension_wo...	85.6833237064771
Tumor_size	-0.415565750547081

Figure 1.21: Ridge coefficients β_j for ($\lambda = 45.07$)

Model Analysis

We notice that some coefficients are very close to 0 such as `area_se` (-0.004), 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.

We stated in the previous part that the test MSE of Ridge regression improved compared to the Linear Regression one. However this didn't come with no expenses, in fact Ridge regression will have a larger bias. 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 test MSE, the squared bias and the cube root of the variance. *We computed the cube root in order to plot it with the bias in the same graph.*

```
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)
```

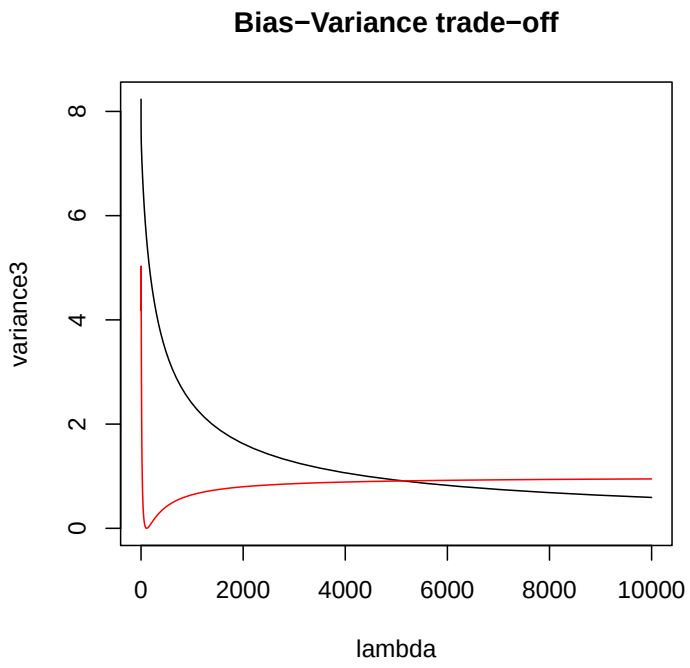

Figure 1.22: Bias Variance trade-off, squared bias(red), root squared variance(black)

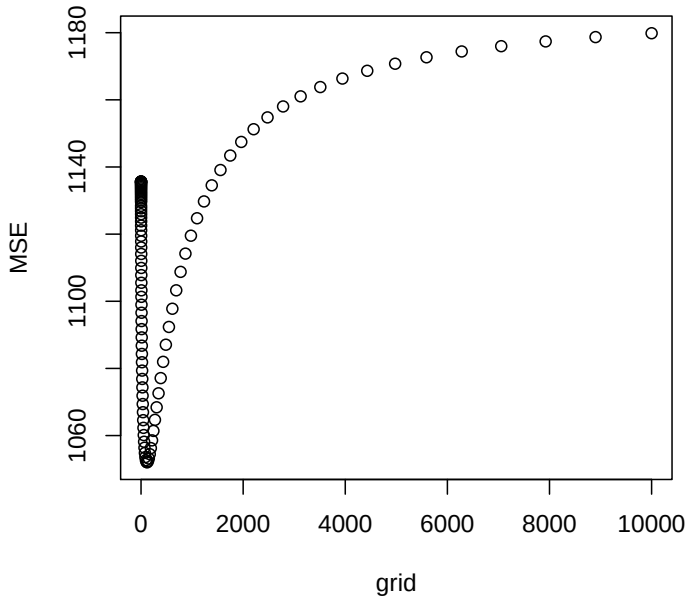

Figure 1.23: MSE against λ

At $\lambda = 0$ the variance is high but there is no bias,which corresponds to the least squared approach. 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. In fact 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.

1.7.2 Lasso Regression

As it was discussed on the previous section, Ridge regression's disadvantage is that it does not perform a variable selection. Lasso Regression however enables variable selection by minimizing the term:

$$RSS + \lambda \sum_j |\beta_j^2| \tag{1.2}$$

The only difference with the term of Ridge is that we now 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 by reducing the number of variables. To call the lasso model we use the same function **glmnet** but with the parameter **alpha=1**.

Model Performance

```
#Grid of 100 values of Lamba from 10^-2 to 10^10
grid = 10^seq(10,-2, length=100)
model.lasso = glmnet(train_set.x, train_set.y, alpha=1,
    lambda=grid)
plot(model.lasso)
```

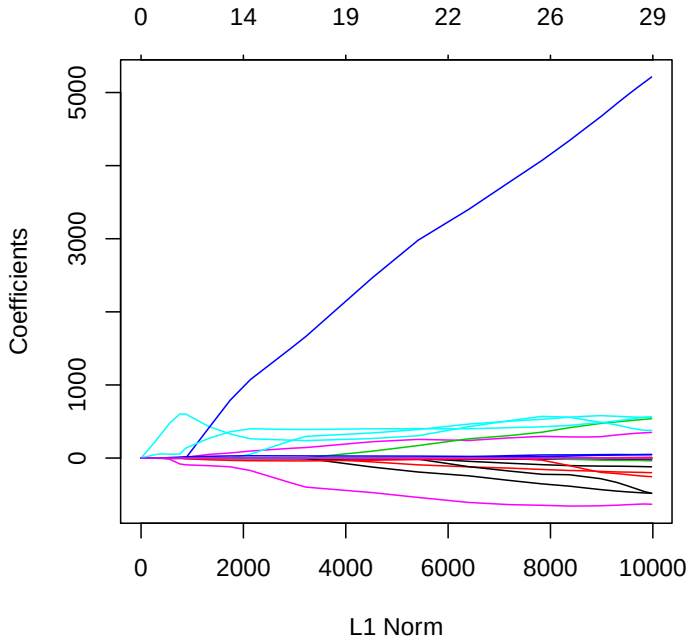

Figure 1.24: Coefficients against L1 Norm

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 pa-

rameter. **cv.glmnet** does k-fold cross-validation for glmnet (by default k=10) it then produces a plot and returns a value for lambda.

```
model.lasso.cv.out = cv.glmnet(train_set.x, train_set.y,
    lambda=grid, alpha=1)
plot(model.lasso.cv.out)
model.lasso.best.lambda = model.lasso.cv.out$lambda.min
```

After running the cross validation we get the best lambda that minimized the term 1.2. We now can run a prediction on the test set with the best lambda and compute the residuals and the test MSE.

```
model.lasso.best = glmnet(train_set.x, train_set.y,
    lambda=model.lasso.best.lambda, alpha=1)
model.lasso.best.pred = predict(model.lasso.best,
    s=model.lasso.best.lambda, newx=test_set.x)

residuals = test_set.y - model.lasso.best.pred
errors    = residuals^2
model.lasso.MSE = mean(errors)
```

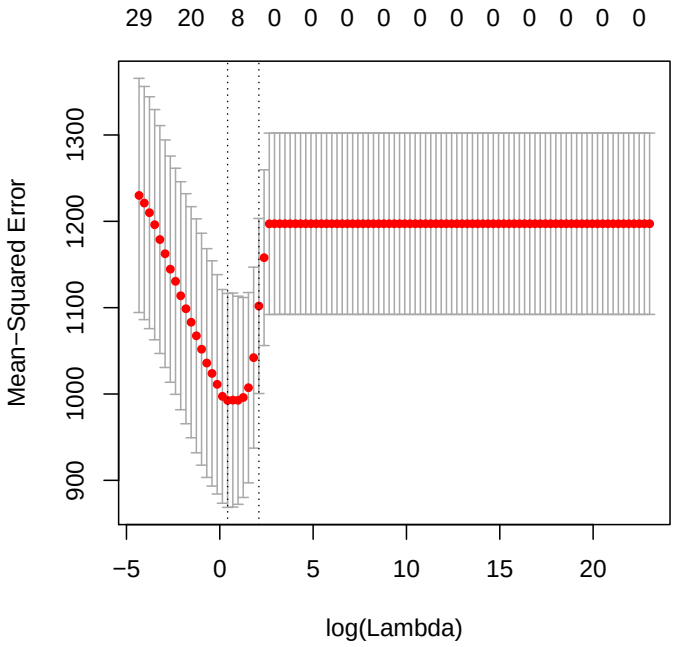

Figure 1.25: MSE against $log(\lambda)$

model.lasso.best.lambda = 3.51
model.lasso.best.MSE = 1045.92

```
# predicted values against real values
plot(x=test_set.y, y=model.lasso.pred)
#y = x
abline(0,1, col='red')

# histogram of residuals and the normal distribution
hist(residuals, freq=FALSE, main="Distribution of Residuals
    in Lasso")
residuals.mean = mean(residuals)
residuals.stdev = sqrt(var(residuals))
curve(dnorm(x, mean=residuals.mean, sd=residuals.stdev),
    col="darkblue", lwd=2, add=TRUE, yaxt="n")
dev.off()
```

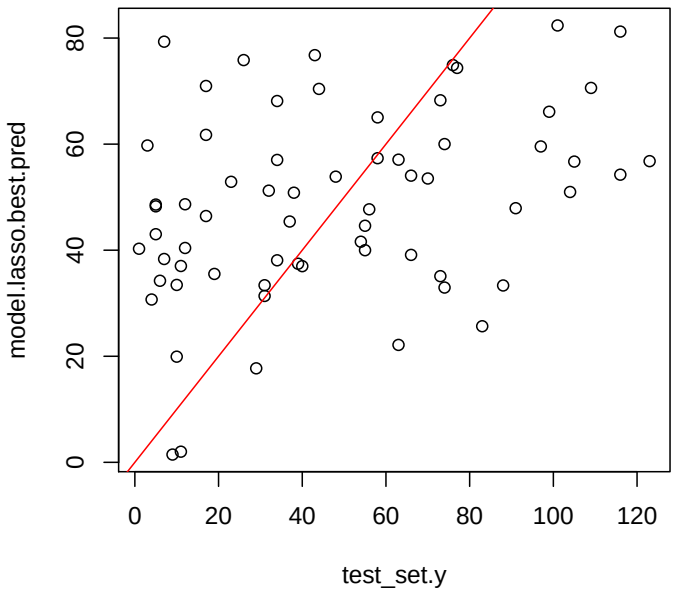

Figure 1.26: Predicted responses $\hat{y}$ against real values y

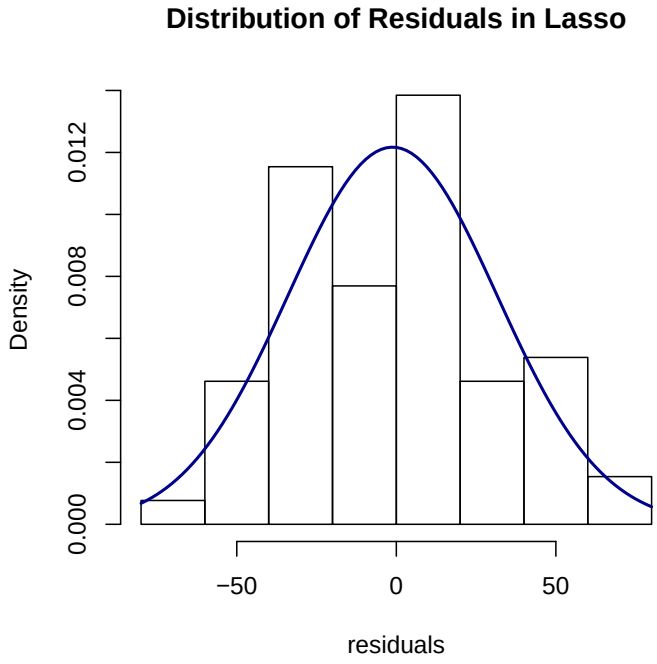

Figure 1.27: Distribution of Residuals in Lasso

The MSE is slightly inferior to Ridge Regression. Nevertheless the residuals are not well distributed which means high residuals are common, especially between (-50 and -25).

```
predict(model.lasso, type="coefficients",
    s=model.lasso.best.lambda)[1:33,]
```

(Intercept)	79.0308812608241
Lymph_node	0
radius_mean	0
texture_mean	-1.02108853748218
perimeter_mean	-0.339703271620677
area_mean	-0.00128975783963866
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave_points_mean	0
symmetry_mean	0
fractal_dimension_m...	511.808342216284
radius_se	0
texture_se	-5.70256925538677
perimeter_se	0
area_se	0
smoothness_se	0
compactness_se	0
concavity_se	-34.3617192851664
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	0
compactness_worst	0
concavity_worst	0
concave_points_worst	0
symmetry_worst	8.55146169876415
fractal_dimension_wo...	53.1202672054075
Tumor_size	0

Figure 1.28: Lasso coefficients β_j for ($\lambda = 3.25$)

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

unselected predictors' coefficients to 0

Model Analysis

Lasso selected only 8 predictors out of 33 with a slightly improved test MSE compared to Ridge that uses all the predictors.Therefore we selected 25% of variables. The features selected for recurrence Time prediction are:

- texture_mean
- perimeter_mean
- smoothness_mean
- fractal_dimension_mean
- texture_se
- smoothness_se
- smoothness_worst
- symmetry_worst
- fractal_dimension_worst

Those are the features that with $\lambda = 3.25$ their coefficients β was not null.

1.8 Linear Regression with Dimension Reduction

1.8.1 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 a 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`. pcr works same way as the lm with some additional parameters, scale helps standardizing the data and setting the validation="CV" will perform a 10 fold cross validation on the model.

```
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 1.29, the model with only 4 components yields to the lowest MSE. However, since $M = 4$ is found after running a random cross validation method, this number may be different if we run this algorithm another time.

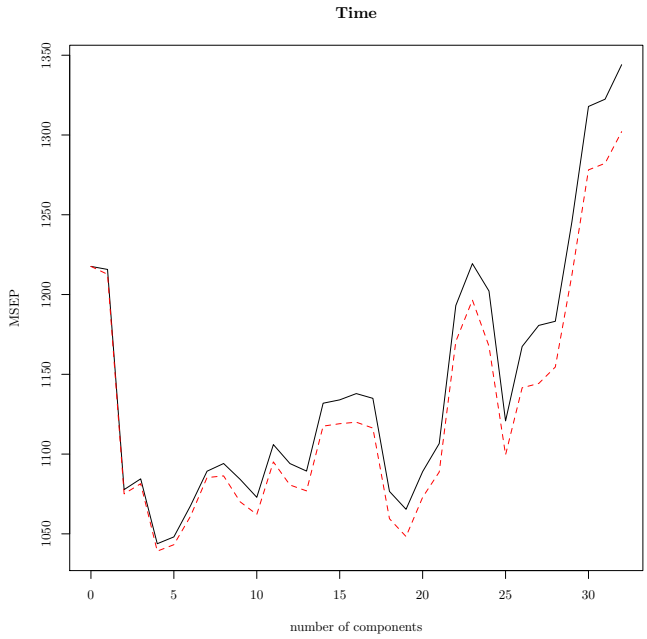

Figure 1.29: PCR

The function `summary` tells us that 4 components are enough to explain 75% of the features' variance, which seems to be enough for this dataset. Indeed, this model performs better that the other linear regression methods : the MSE is approximatively equal to 967.

The test MSE obtained is very competitive, however even though we know that the number of components is 4 PCR does not explicitly show the predictors.

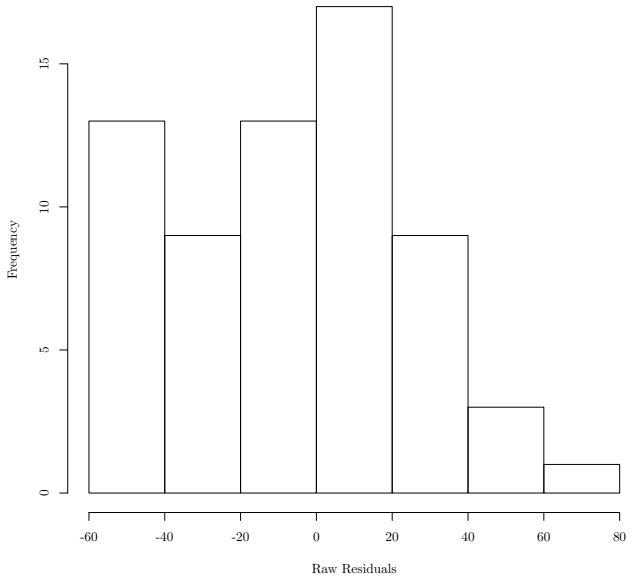

Figure 1.30: PCR Raw Residuals Distribution

Therefore PCR's results are harder to interpret as it does not perform a variable selection.

1.9 Models Comparison

Comparing the MSE and the raw residuals distribution is the first approach to take for judging and comparing the performance of each model.

1.9.1 MSE and Residuals

Model	Mean Squared Error (MSE)
KNN neighbours (with LOOCV)	1146
Linear Regression	1285
Best Subset	1067
Ridge Regression	1063
Lasso	1048
PCR	967

Table 1.1: Models' MSE

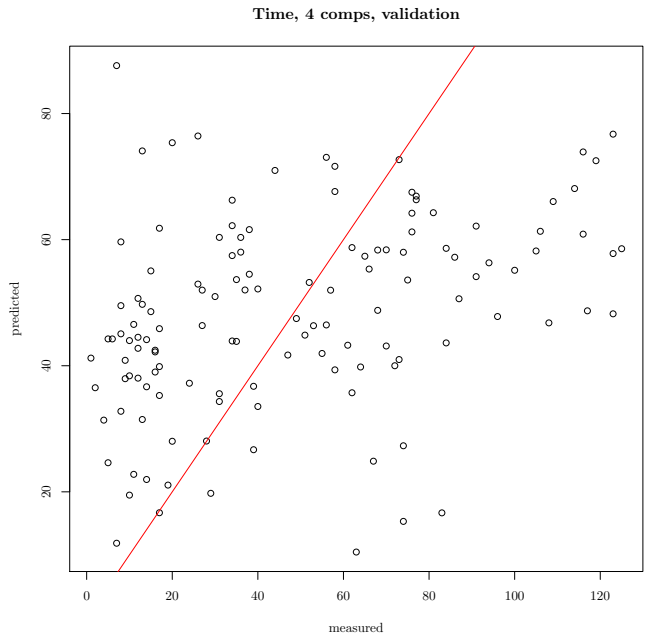

Figure 1.31: PCR Predicted values against Real values

We notice that the best MSE is found with the PCR model.

It is also important to compare the residuals distribution of our models.If the residuals are normal, it means that our assumption is valid and model inference (confidence intervals, model predictions) should also be valid then.

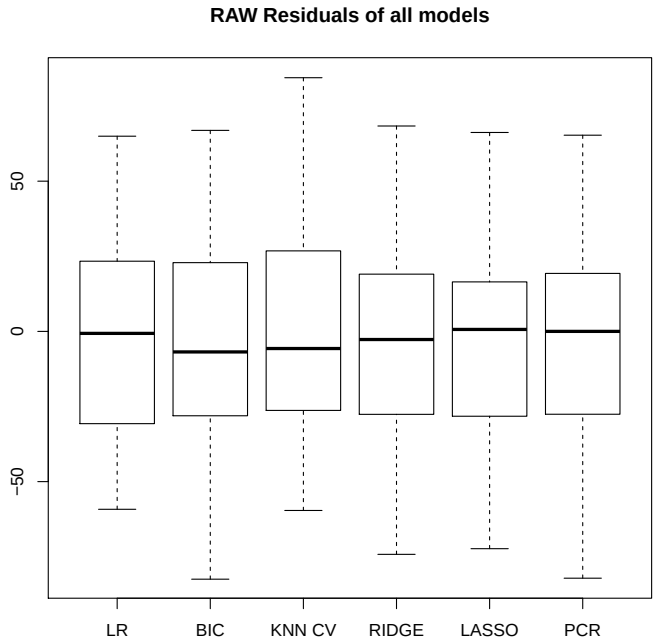

Figure 1.32: BoxPlots of residuals off all models

Two main information can be retained from this raw residuals box-plots:

- Range: It seems like the Linear Regression and the KNN with CV have the largest range of residuals, while Lasso shows a striking decrease in the range of residuals.
- Median: Lasso's raw residuals are normally distributed. This was noticeable in the previous histogram in 1.20. Other residuals are close to normal distribution except the BIC, KNN and Lasso.

1.9.2 KNN neighbors and Linear Regression

The linear regression is a **parametric method**, it comes with the hypothesis that there is a relationship between the response Y and the p predictors X_i that can be expressed like $Y = \beta_1 X_1 + \beta_2 X + 2 + ... + \beta p X_p$. Their main advantages is that it is easy to use and to interpret however when the relationship between the predictors and the response is not linear the the results might not be satisfying. The KNN neighbors however is a non parametric method. In other words it makes no assumptions on the response Y. If we compare between the **test MSE** obtained by the KNN (1146) and the one obtained by the Linear Regression (1285), we see that the KNN does a better job. These arguments the fact that

the model is not really linear. A remark that was made when observing the QQ plots in the first section.

1.9.3 Best subset Selection and Linear Regression Regularizations

Ridge & Lasso

The lasso's advantage over Ridge Regression is the variable selection; the model is thus easier to interpret. The MSE in Lasso is also slightly improved. In fact Lasso works better when there isn't an important number of predictors which corresponds to our case since we only have 33. In order to compare between their variance bias trade off. We run the ridge regression with all the different values of λ but for Lasso we run the regression with the new data set that contains only 8 variables.
In order to create the new data_set of 8 predictors we used the function **cbind** that binds columns and **data.frame** for storing data tables. cbind did not keep the columns'names so we renamed them using the function **colnames**.

```
newdata_set =  data.frame(cbind(data_set$texture_mean,
    data_set$perimeter_mean, data_set$area_mean,
data_set$fractal_dimension_mean,
data_set$texture_se, data_set$concavity_se,
    data_set$symmetry_worst,
data_set$fractal_dimension_worst, data_set$Time))

colnames(newdata_set) <- c("texture_mean",
    "perimeter_mean", "area_mean", "fractal_dimension_mean",
    "texture_se",
"concavity_se","symmetry_worst", "fractal_dimension_worst",
    "Time" )

n      = nrow(newdata_set)
p      = ncol(newdata_set)
napp   =  round(2*n/3)
ntst   = n-napp
train     = sample(1:n, napp)
newtrain_set    = newdata_set[train,]
newtest_set     = newdata_set[-train,]

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

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

Now that we have our new data_set, let's compare between the test MSE and variance bias trade off variance in function of the tuning parameter λ

```
#Number of itterations
max          = 100
#Ridge: squared Biais, root square variance and MSE
biais2_r     = rep(0,max)
variance3_r = rep(0,max)
MSE_r        = rep(0,max)

#Lasso: squared Biais, root square variance and MSE
biais2_l     = rep(0,max)
variance3_l = rep(0,max)
MSE_l        = rep(0,max)

#Grid of lamba between 10^-1 and 10^4
grid         = 10^seq(4,-1, length=max)
for( i in 1:max)
{
   # Ridge Regression
   model.ridge = glmnet(train_set.x, train_set.y,
       lambda=grid[i], alpha=0)
   model.ridge.pred = predict(model.ridge, s=grid[i]
       ,newx=test_set.x)
   MSE_r[i] = mean((model.ridge.pred - test_set.y)^2) #MSE
   biais2_r[i] = (mean(model.ridge.pred - test_set.y))^2
       #squared bias
   variance3_r[i] = (var(model.ridge.pred))^(1/3) #sqrt
       cubic of variance

   #Lasso Regression with 8 predictors
   model.lasso = glmnet(newtrain_set.x , newtrain_set.y,
       lambda=grid[i], alpha=1)
   model.lasso.pred = predict(model.lasso, s=grid[i],
       newx=newtest_set.x)
   MSE_l[i] = mean((model.lasso.pred - test_set.y)^2) #MSE
   biais2_l[i] = (mean(model.lasso.pred - test_set.y))^2
       #squared bias
   variance3_l[i] = (var(model.lasso.pred))^(1/3) #sqrt
       cubic of variance
}
#Red for Ridge
#Variance
plot(grid,variance3_r,type="l", xlab="lambda", ylab="root
    squared Var and squared
    Biais",col="red",ylim=range(biais2_r, variance3_r,
    biais2_l, variance3_l),
    main="Bias(dashed)-Variance(line) trade-off of
    Ridge(Red) and Lasso(Blue)")
#Biais
```

```
lines(grid,biais2_r,col="red", lty=2)

#Blue for Lasso
#Variance
lines(grid,variance3_l,col="blue")
#Biais
lines(grid,biais2_l,col="blue",lty=2)
```

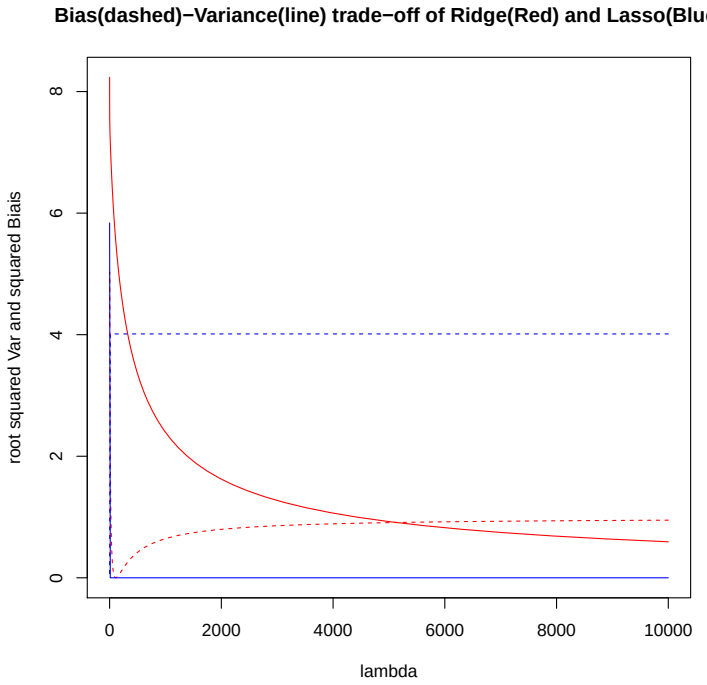

Figure 1.33: Bias(dashed line)-Variance(line) Trade off of Ridge(Red) and Lasso(Blue)

As it was stated in the subsection above 1.7.1 both Ridge and Lasso influence on the bias and the variance same way. It is clear from the curves above that Lasso decreases the variance more on the expense of the bias which explains why the Ridge's vaiance is higher than the Lasso's one but its bias is smaller. These leads us to conclude that neither of the models performs better than the other. Lasso will be more pertinent to use if we have few predictors as it will set some coefficients to null and so the model will be easier to interpret. Nevertheless if we have lot of predictors Ridge might be a better idea because all the coefficients will be small.

Regularization & Best subset

Both Lasso and Best Subset perform a variable selection, their MSE are close but with the Best Subset we only kept 3 variable which is 38% less predictors. How can we compare between those two approaches?
First Lasso adds up a new tuning parameter to penalize the coefficients which is why it is named shrinkage method. In contrast to the Best subset selection that doesn't add up anything. Therefore the regression coefficients obtained by Lasso are biased.

1.9.4 Dimension Reduction

Last but not least, dimension reduction is competitive as it has the lowest MSE and its raw residuals are distributed normally. PCR can be an interesting approach if we aim to **predict** data, however it won't be of much utility if the goal is the **inference**. Since the PCR doesn't explicitly select variables, the relationship between the response and a given predictor is not clear and Lasso would be a better choice.

Chapter 2

Phoneme Recognition

2.1 Context

In the context of speech recognition the aim is to predict which of the phonemes is pronounced by the subject. In this case, 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
```

2.3 Measure to Compare Models

Before building any model, we have to properly define the measures we will use later to compare their performance. In a classification problem, "accuracy" is often used. It is defined by :

$$A = \frac{\text{Number of Good Predictions}}{\text{Number of Cases}}$$

This is a very simple measure that may cause some issues when applied on classification problems that deal with skewed classes. For instance, let A and B, two classes. A contains 95% of the dataset, while B contains the remaining 5%. Let M a pretty bad classifier which classifies the whole dataset in class A. According to the definition stated above, this classifier has an accuracy of 95%, which is excellent. In this case, more specific performance measures have to be applied.

However, our current classification problem do not deal with skewed class since all classes are almost equally-represented in the dataset. Therefore, we will be able to run this performance measure without any important issue. This measure will be aplied on the test set in order not to include any biais.

```
train_set= data_set[1:3340, 2:258]
train_set.x = train_set[,1:256]
train_set.y = train_set[,257]

test_set = data_set[3341:4509, 2:258]
test_set.x = test_set[,1:256]
test_set.y = test_set[,257]
```

2.4 Classification

2.4.1 LDA - Linear Discriminant Analysis

The first model we are going to use takes advantage of Bayes' Theorem. Let $f_k(x)$ the class-conditional density of X when $Y = k$, and π_k the prior probability of class k. Bayes' Theorem proposes the following statement :

$$\mathbb{P}(Y = k|X = x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^{K} f_l\pi_l}$$

X belongs to the class whose conditional probability is the highest.

If the class distribution of the dataset is representative of the real distribution, we can estimate π_k by calculating the proportion of k class elements in the dataset. How can we now estimate $f_k(x)$? Linear Discriminant Analysis assumes that all $f_k(x)$ are Gaussian distributions whose covariance matrix Σ are the same.

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 ~ ., data=train_set)
summary(model.lda)
```

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

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 not work well with the phonemes `aa` and `ao` which sound very similar.

Improvements with PCA

The dataset has a high number of features (256) that may prevent the model from fitting well. We can cope with this performance issue by applying a dimension reduction method, such as Principal Component Analysis (PCA). The function`prcomp` is available in R to compute the principal components of the 256 features. Since PCA depends on the scaling of the inputs, it is important to scale and center each feature.

```
pca = prcomp(train_set.x, center = TRUE, scale. = TRUE)
```

The main question is now : how many principal components should we use to replace our main features ? The best number of components M yields to the highest accuracy. The following code finds the best M for the test set.

```
accs = matrix(0, 50, 1)

for (M in 2:50) {
    train_set.pca.x = as.data.frame(pca$x[,1:M])
    train_set.pca = as.data.frame(cbind(train_set.y,
        train_set.pca.x))

    test_set.pca.x = predict(pca, newdata = test_set.x)[,1:M]
    test_set.pca = as.data.frame(cbind(test_set.y,
        test_set.pca.x))

    model.lda.pca = lda(train_set.y ~ ., data =
        train_set.pca)
    model.lda.pca.predicted =
        predict(model.lda.pca,newdata=test_set.pca)

    perf = table(test_set$g,model.lda.pca.predicted$class)
    accs[M] = sum(diag(perf))/dim(test_set)[1]
}

which.max(accs[-1])
plot(accs[-1])
```

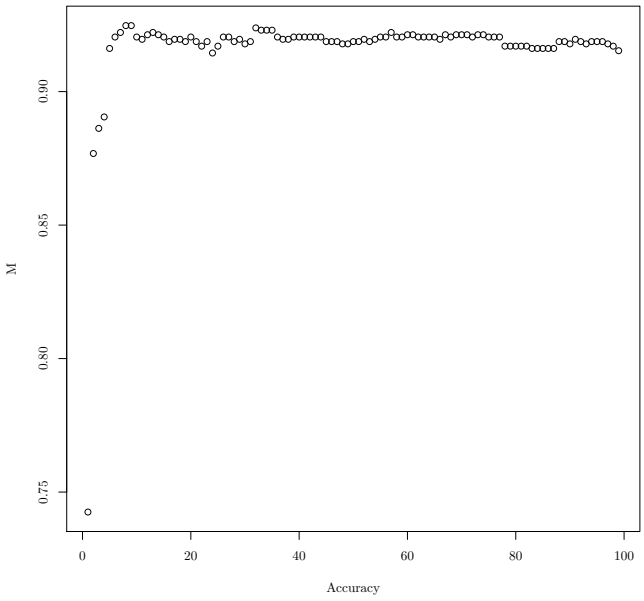

Figure 2.1: Best M for LDA + PCA on Test Set

The best M on the test set is 8 (figure 2.1), which yields to an accuracy of 92.2%.

g	aa	ao	dcl	iy	sh
aa	131	45	0	0	0
ao	40	223	0	0	0
dcl	0	0	191	4	0
iy	0	0	1	309	1
sh	0	0	0	0	224

However, this method raises a biais since we computed M based on the test set. A proper way to find M is using a k fold cross validation technique on the training set. The following code aims to find the best M in the range $[2:50]$.

```
1  library(caret)
2
3  accs = matrix(0, 50, 1)
4  for (M in 2:50) {
5     a.train_set.pca.x = as.data.frame(pca$x[,1:M])
6     a.train_set.pca = as.data.frame(cbind(train_set.y,
          a.train_set.pca.x))
7
8     folds = createFolds(train_set.pca$train_set.y)
9
```

```
    acc = 0;
    for (k in 1:10) {
        validation_indexes = folds[[k]]
        a.train_set.x =
            a.train_set.pca.x[-validation_indexes,]
        a.train_set = a.train_set.pca[-validation_indexes,]

        a.validation_set.x =
            a.train_set.pca.x[validation_indexes,]
        a.validation_set =
            a.train_set.pca[validation_indexes,]

        model.lda.pca = lda(a.train_set$train_set.y ~ ., data
            = a.train_set)
        model.lda.pca.predicted =
            predict(model.lda.pca,newdata=a.validation_set)

        perf =
            table(a.validation_set$train_set.y,model.lda.pca.predicted$class)
        acc = acc + sum(diag(perf))/dim(a.validation_set)[1]
    }

    acc = acc / 10
    accs[M] = acc
}
```

According to the graph shown in figure 2.2, the best M is 42, which yields to an accuracy score of 92.04% :

```
g    aa  ao dcl  iy  sh
aa  129  47   0   0   0
ao   39 224   0   0   0
dcl   0   0 189   6   0
iy    0   0   1 310   0
sh    0   0   0   0 224
```

2.4.2 QDA - Quadratic Discriminant Analysis

This method is very similar to the LDA, the only difference being that the covariance matrices Σ_k are not necessarily equal.

Model Implementation

In R, the function `qda` creates a classifier based on the QDA methods. It works the same way as `lda` :

```
model.qda = qda(g ~ ., data=train_set)
```

g	aa	ao	dcl	iy	sh
aa	27	143	0	3	3
ao	2	259	0	1	1
dcl	0	0	165	29	1
iy	0	0	0	311	0
sh	0	0	1	1	222

This model performs worse than the LDA : the accuracy is about 84.2%. It does not manage to make the difference between class `aa` and `ao`. This may mean that the covariance matrices Σ_k are almost equal and that we will not be able to make this model more accurate than LDA.

Improvements with PCA

We know that the error rate of QDA highly depends on the size of the dataset and the number of features. Therefore, we may be able to improve its performance using PCA using the same 10 fold cross validation technique we used earlier.

In this case, the algorithm returns $M = 12$ (figure 2.3) which leads to 92% accuracy. We successfully improved QDA's performance but the accuracy is still not better than LDA, as expected.

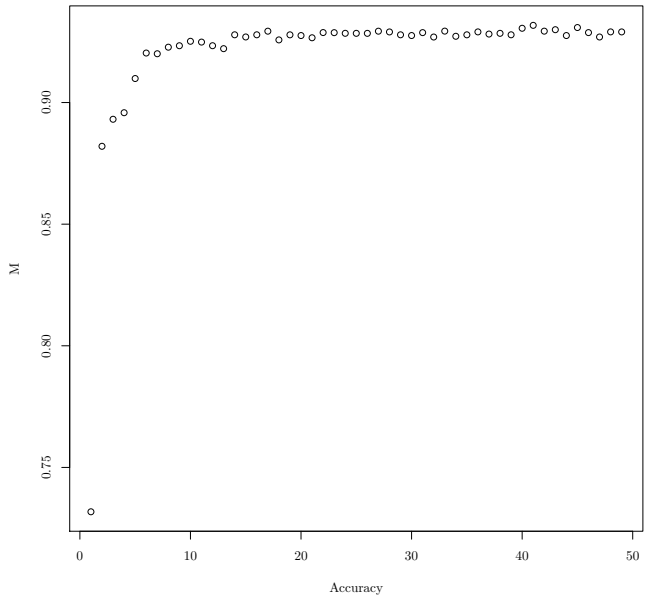

Figure 2.2: Best M for LDA + PCA on Training Set with Cross Validation

g	aa	ao	dcl	iy	sh
aa	124	52	0	0	0
ao	35	228	0	0	0
dcl	0	0	192	3	0
iy	0	0	2	308	1
sh	0	0	0	0	224

2.4.3 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

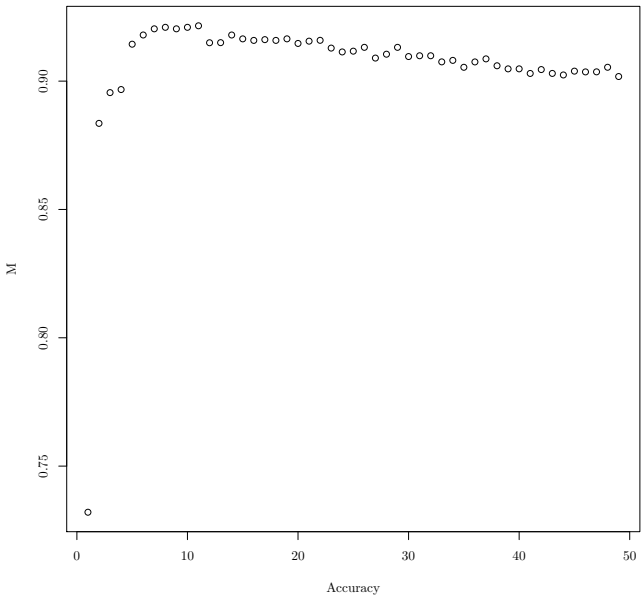

Figure 2.3: Best M for QDA + PCA on Training Set with Cross Validation

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 maximum number of iterations.

```
library("nnet")
model.lr = multinom(formula = g ~ . , 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]
```

Neural Networks are known to be quite long to train, that is why it takes several seconds to fit the model. The error rate is similar to QDA (accuracy is about 86%), however it performs better on `aa` and `ao`:

g	aa	ao	dcl	iy	sh
aa	114	54	4	2	2
ao	47	197	8	10	1
dcl	0	1	192	2	0
iy	0	2	12	292	5
sh	0	1	6	1	216

Improvements with PCA

Once again, we can use PCA to reduce the input dimension in order to make the neural network fit better and faster. However, since `multinom` runs significantly slower than the other methods, we will only find the best M in the range $[2, 20]$ using a 5 fold cross validation method.

The best accuracy is reached at $M = 17$, accuracy on the test set being approximatively 91% :

g	aa	ao	dcl	iy	sh
aa	122	54	0	0	0
ao	34	228	0	1	0
dcl	0	0	190	5	0
iy	0	4	3	303	1
sh	1	0	1	1	221

2.5 Models Comparison

According to our comparison measure (accuracy), the best model we found is LDA with PCA. LDA is the most interesting model so far because of its simplicity and its performance with and without PCA transformation. QDA

and LR does not perform well with the full set of features, but they are as good as LDA when a PCA feature subset selection is applied.

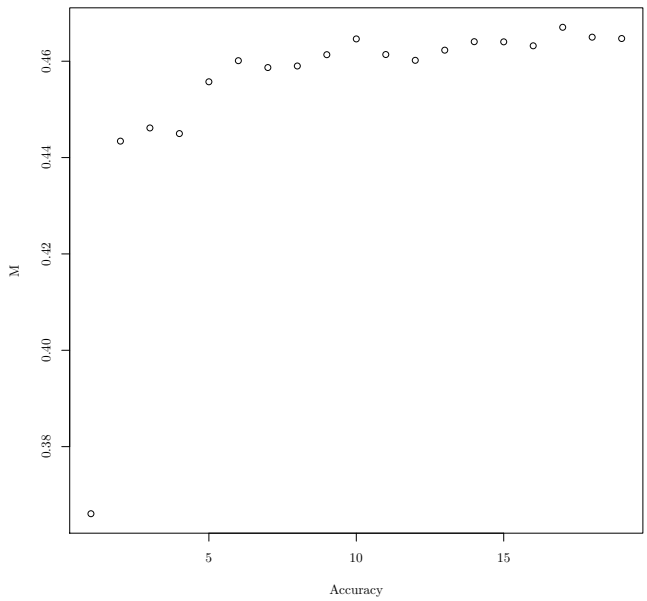

Figure 2.4: Best M for LR + PCA on Training Set with Cross Validation