Statistical Learning (Mod. B)

Wu Xianlong MAT: 2038500

Contents

Data description and objective introduction **3**

Data inspection and preprocessing **4**

Representations of the dataset **9**

Removal of highly correlated features 9

Principle Component Analysis (PCA) 10

Multiclass classification **15**

K-Nearest Neighbor (KNN) 15

Multinomial Logistic Regression 19

Linear Discriminant Analysis (LDA) 23

Quadratic Discriminant Analysis (QDA) 26

Summary 28

Feature selection and Regularization **28**

LASSO 28

Ridge Regression 36

Elastic-net 47

Conclusion **47**

```
library(dplyr)
```

```
## Warning: il pacchetto 'dplyr' è stato creato con R versione 4.1.3
```

```
##
## Caricamento pacchetto: 'dplyr'
```

```
## I seguenti oggetti sono mascherati da 'package:stats':
##
##     filter, lag
```

```
## I seguenti oggetti sono mascherati da 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(corrplot) # get the correlation plots
```

```
## Warning: il pacchetto 'corrplot' è stato creato con R versione 4.1.3
```

```
## corrplot 0.92 loaded
```

```
library(glmnet)
```

```
## Warning: il pacchetto 'glmnet' è stato creato con R versione 4.1.3
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
library(caret)
```

```
## Warning: il pacchetto 'caret' è stato creato con R versione 4.1.3
```

```
## Caricamento del pacchetto richiesto: ggplot2
```

```
## Warning: il pacchetto 'ggplot2' è stato creato con R versione 4.1.3
```

```
## Caricamento del pacchetto richiesto: lattice
```

```
library(ggplot2)
library(gridExtra)
```

```
## Warning: il pacchetto 'gridExtra' è stato creato con R versione 4.1.3
```

```
##
## Caricamento pacchetto: 'gridExtra'
```

```
## Il seguente oggetto è mascherato da 'package:dplyr':
##
##     combine
```

```
library(mltools)
```

```
## Warning: il pacchetto 'mltools' è stato creato con R versione 4.1.3
```

```
library(data.table)
```

```
## Warning: il pacchetto 'data.table' è stato creato con R versione 4.1.3
```

```
##
## Caricamento pacchetto: 'data.table'

## I seguenti oggetti sono mascherati da 'package:dplyr':
##
##     between, first, last
```

```
library(nnet)
library(crosstable)
```

```
## Warning: il pacchetto 'crosstable' è stato creato con R versione 4.1.3
```

```
library(DMwR2)
```

```
## Warning: il pacchetto 'DMwR2' è stato creato con R versione 4.1.3

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(MASS)
```

```
##
## Caricamento pacchetto: 'MASS'

## Il seguente oggetto è mascherato da 'package:dplyr':
##
##     select
```

```
library(lattice)
```

Data description and objective introduction

The dataset that is used in this project comes from the UCI dataset which performs human activity recognition with the data collected from a smart phone(https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones). The data is collected from an experiment with 30 volunteers within an age bracket of 19-48 years. Each of them wears a smartphone and with the accelerometer and gyroscope embedded in the device, the 3-axial linear acceleration and 3-axial angular velocity were sampled with a frequency of 50Hz. Each of the volunteers have only 6 states(activities), which are: 'WALKING', 'WALKING_UPSTAIRS', 'WALKING_DOWNSTAIRS', 'SITTING', 'STANDING', 'LAYING.' The data has been pre-processed and separated as 70 percent of them are selected as training data and the rest 30 percent are the test data. \

As for the objective of this report, we would like to investigate the dataset from three perspectives: The first one is to find a low dimensional representation or transformation of the dataset so that the maximum amount of useful information can be kept and the second one is to find a good statistical model that can perform the classification task correctly and the third one is to further apply different regularization techniques such that the generalization ability of the model is guaranteed. Furthermore, some the techniques applied can be also considered as a way of performing feature selection such that the most relevant predictors can be found. \

Data inspection and preprocessing

Now, we first load the dataset and make the primary inspection of the dataset.

```
X_train_ori <- read.table('X_train.txt')
X_test_ori <- read.table('X_test.txt')
y_train <- read.table('y_train.txt')
y_test <- read.table('y_test.txt')
```

We first check the input training and test set, by checking the dimension and NA values \

```
cat('Dimension of the training set is: \n' , dim(X_train_ori))
```

```
## Dimension of the training set is:
##  7352 561
```

```
cat('\n')
```

```
cat('Dimension of the test set is: \n' , dim(X_test_ori))
```

```
## Dimension of the test set is:
##  2947 561
```

```
cat('\n')
```

```
cat('Number of NA in the training set is: \n', sum(colSums(is.na(X_train_ori))))
```

```
## Number of NA in the training set is:
##  0
```

```
cat('\n')
```

```
cat('Number of NA in the test set is: \n', sum(colSums(is.na(X_test_ori))))
```

```
## Number of NA in the test set is:
##  0
```

S We can see that there is not NA values in both the training and test set. Let's plot the bar plot to show the classes in a better manner. \

```
colnames(y_train) <- c('Activity')     # Change the names of the response
colnames(y_test) <- c('Activity')
cat('The column name of the training response is: \n')
```

```
## The column name of the training response is:
```

```
colnames(y_train)
```

```
## [1] "Activity"
```

```
cat('\n\n')
```

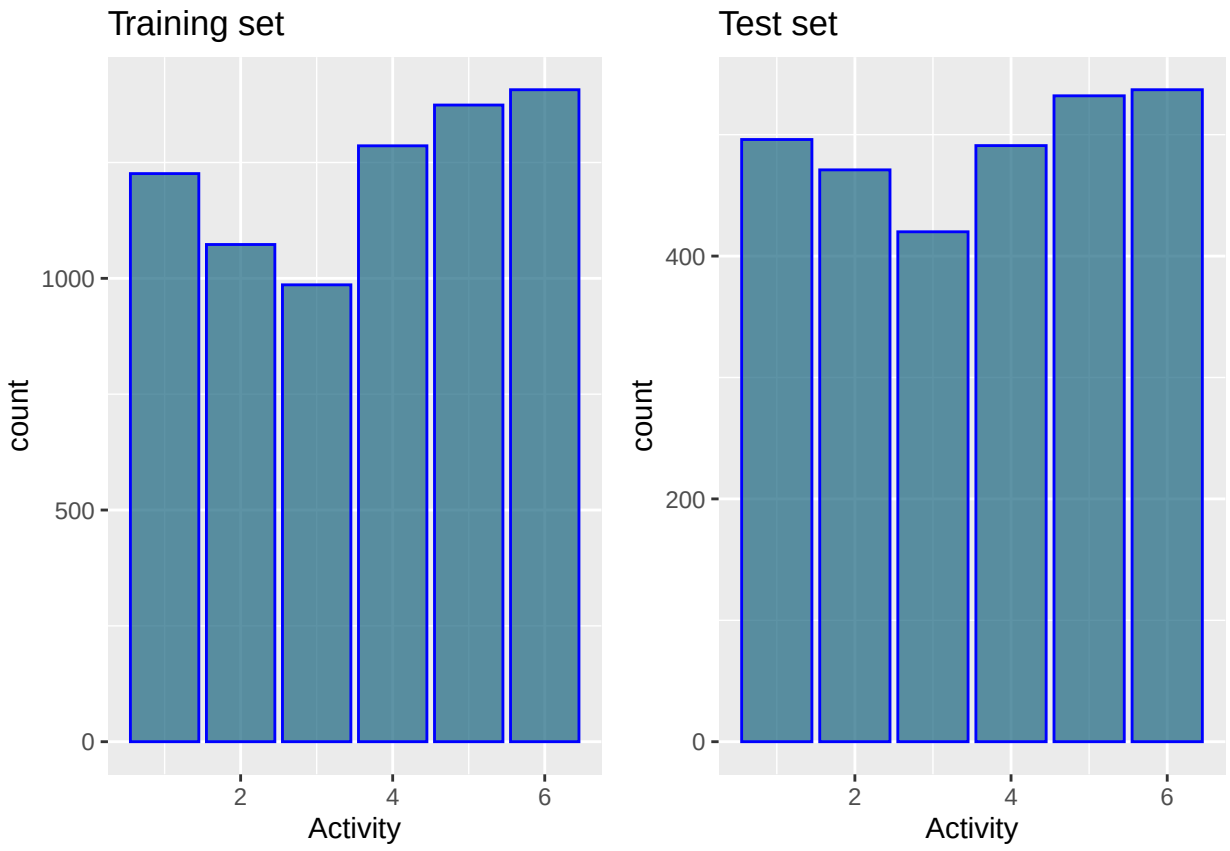
```
cat('The column name of the test response is: \n')
```

```
## The column name of the test response is:
```

```
colnames((y_test))
```

```
## [1] "Activity"
```

```
p1 <- ggplot(y_train, aes(x=Activity )) +
  geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7))

p2 <- ggplot(y_test, aes(x=Activity )) +
  geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7))

grid.arrange(p1 + labs(title = "Training set"), p2 + labs(title = "Test set"), ncol = 2)
```



From the bar plots above, we can see that for both training and test set, we have 6 different classes which

are labelled as integer number from 1 to 6 which correspond to the 6 possible activities of the monitored individual. Furthermore, we can see that the classes labelled as '2' and '3' are slightly less frequent compared to other classes, thus we will use the 'oversampling' technique to generate more data so that in the end all the classes are balanced. We can use the 'upSample' function, but before applying the function, we should merge the the dataset with the corresponding response. \

```
X_train_ori <- cbind(X_train_ori, y_train)
X_test_ori <- cbind(X_test_ori, y_test)
X_train_ori[1:5, ncol(X_train_ori)]
```

```
## [1] 5 5 5 5 5
```

```
X_test_ori[1:5, ncol(X_test_ori)]
```

```
## [1] 5 5 5 5 5
```

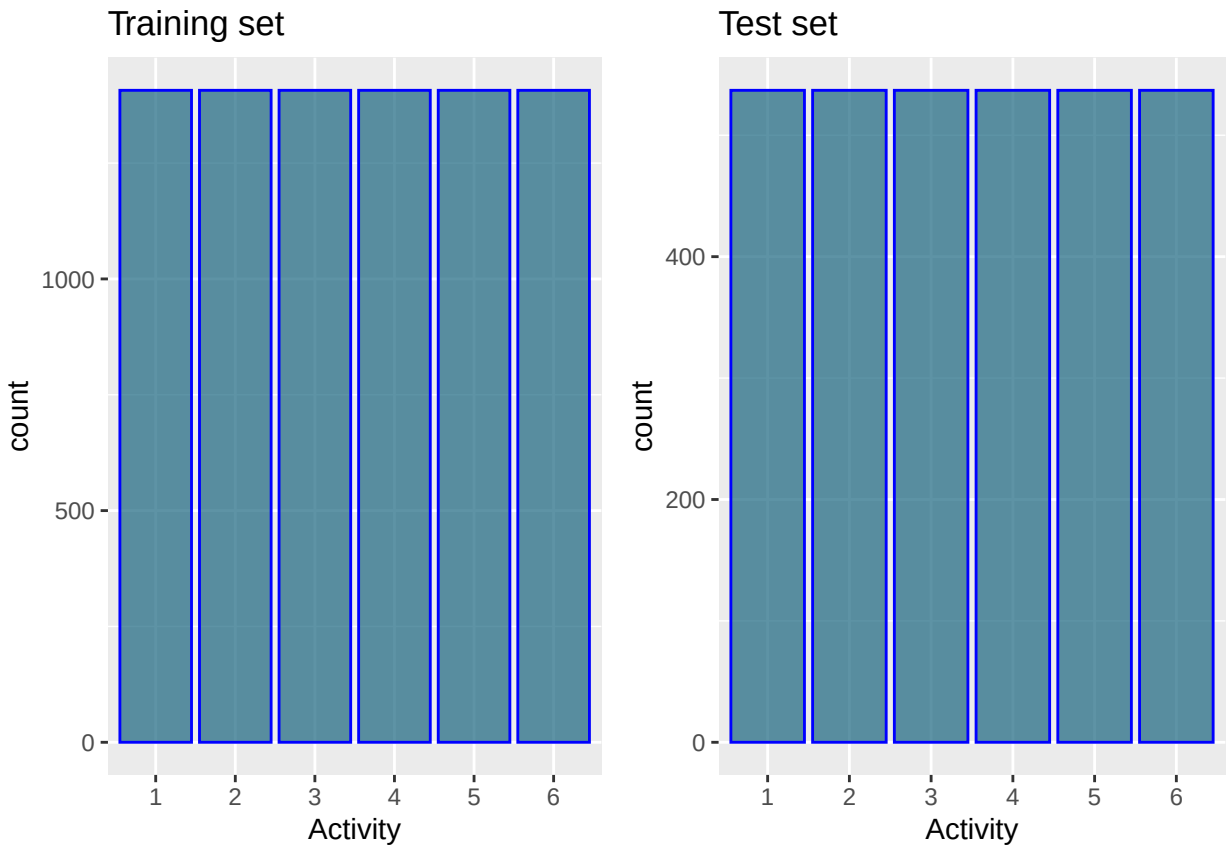
From the code output above, we can see that for both training and dataset the last column is the response. Now, we can perform the oversampling. \

```
set.seed(234)
X_train_ori <- upSample(x = X_train_ori[, -ncol(X_train_ori)],
                        y = as.factor(X_train_ori$Activity))

X_test_ori <- upSample(x = X_test_ori[, -ncol(X_train_ori)],
                       y = as.factor(X_test_ori$Activity))
```

Before accessing the result of the oversampling, let's extract the response and change the feature names for the sake of consistency. \

```
colnames(X_train_ori)[ncol(X_train_ori)] <- "Activity"
colnames(X_test_ori)[ncol(X_test_ori)] <- "Activity"

y_train <- data.frame(X_train_ori$Activity)
y_test <- data.frame(X_test_ori$Activity)

colnames(y_train)[ncol(y_train)] <- "Activity"
colnames(y_test)[ncol(y_test)] <- "Activity"
```

```
p1_upSampled <- ggplot(y_train, aes(x=Activity )) +
  geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7))

p2_upSampled <- ggplot(y_test, aes(x=Activity )) +
  geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7))

grid.arrange(p1_upSampled + labs(title = "Training set"), p2_upSampled + labs(title = "Test set"), ncol
```



We see that all the classes are now uniformly distributed. Now, as for the labels, they are just integer numbers that represent the human activities which are not ordinal, in order to get a better visualization, we insert the correct labels first. \

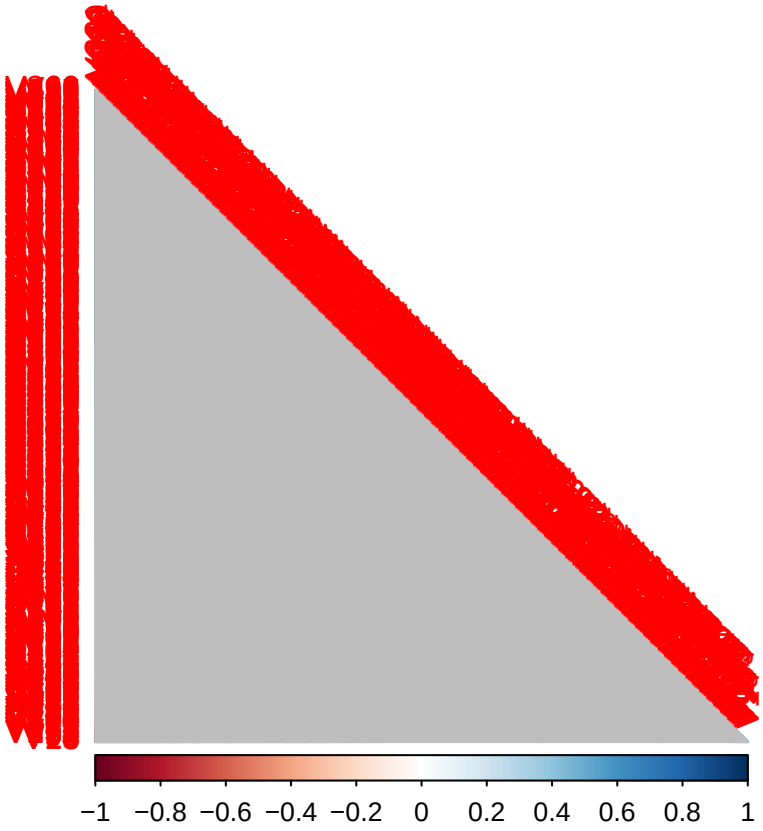
```
# Define a dictionary of old and new values
replace_dict <- c("1" = "WALKING", "2" = "WALKING_UPSTAIRS", "3" = "WALKING_DOWNSTAIRS", "4" = "SITTING

# Replace values in the response
y_train$Activity <- ifelse(y_train$Activity %in% names(replace_dict), replace_dict[y_train$Activity], y
y_test$Activity <- ifelse(y_test$Activity %in% names(replace_dict), replace_dict[y_test$Activity], y_tes

X_train_ori$Activity <- ifelse(X_train_ori$Activity %in% names(replace_dict), replace_dict[X_train_ori$
X_test_ori$Activity <- ifelse(X_test_ori$Activity %in% names(replace_dict), replace_dict[X_test_ori$Acti


cat('The new labels are: \n', unique(X_train_ori$Activity))
```

```
## The new labels are:
##  WALKING WALKING_UPSTAIRS WALKING_DOWNSTAIRS SITTING STANDING LAYING
```

The next task we should perform is get the correlations between the features: \

```
X_train_unlabelled <- X_train_ori[, -ncol(X_train_ori)]
X_test_unlabelled <- X_test_ori[, -ncol(X_test_ori)]
```

```
cor_mat = cor(X_train_unlabelled)
corrplot(cor_mat, method = 'square', order = 'FPC', type = 'lower', diag = FALSE)
```



From the plot, we can see that there is almost no correlations between different features, however, since there is 561 features in our dataset thus in order to confirm the observation, we now select the sub-matrix that contains the entries that are greater than 0.95 in absolute values. \

```
abs_cor_mat <- abs(cor_mat)    # get the matrix in absolute values
masked <- abs_cor_mat[which(abs_cor_mat > 0.95)] %>% as.array()   # get the part that larger than 0.95
ordered <- sort(unique(masked), decreasing = TRUE)
cat('The first 10 largest unique entries of the correlation matrix in absolute values are: \n', ordered
```

```
## The first 10 largest unique entries of the correlation matrix in absolute values are:
##  1 1 1 0.9999997 0.9999996 0.9999993 0.9998738 0.9997545 0.9996918 0.9996436
```

```
cat('\n\n')
```

```
cat('The size of the array is : \n', dim(ordered))
```

```
## The size of the array is :
##  1621
```

From the codes above we can see that it is NOT true that there is almost no correlations between the features, in fact, the size of the array is 1749, hence the plot above has this specific pattern simply because

the size of the matrix is too much and we do not have enough margin to make them visible. Indeed, from the top 10 largest entries in absolute values, we see all of them show perfect/almost perfect correlations. One thing should we should keep in mind is that notice the largest absolute value of the correlation coefficients is 1 which might have two possible sources, the first source is simply the entries on the diagonal of the correlation matrix as all of them indicate the self-correlations of each individual feature which is for sure equals to 1 and thus guaranteed to be present in the top 10 largest coefficients, the second possible source is the features that have perfect positive/negative correlations which is not visible to us since only the unique values are considered, thus in our dataset there is a possibility that there is perfect correlations between different features but they are just 'covered' by the self-correlations of each individual feature. \

Another observation is that, besides the entry equals to 1 that is discussed above, we can see that the second and third largest entries are also equal to 1, considering only the unique values are considered, thus this happened due to the precision of R, indeed, we can easily verify it with the following code which gives us the 'FALSE' as output. \

```
ordered[2] == ordered[3]
```

```
## [1] FALSE
```

Representations of the dataset

Within this section, we would like to transform the dataset into a lower dimensional representation as by doing we can save both the memory and the computation time without losing too much information of the data. Furthermore, since co-linearity exits in the dataset, thus approach such as Linear Discriminant Analysis (LDA) require a preprocessing of the dataset. \

In this report, two approaches of transformation or modification of the dataset is used. The first one is the most intuitive which is removing the highly correlated features as they do not provide useful information and the second approach is the Principle Component Analysis which projects the dataset onto a lower dimensional space and tries to maintain the maximum information. \

Removal of highly correlated features

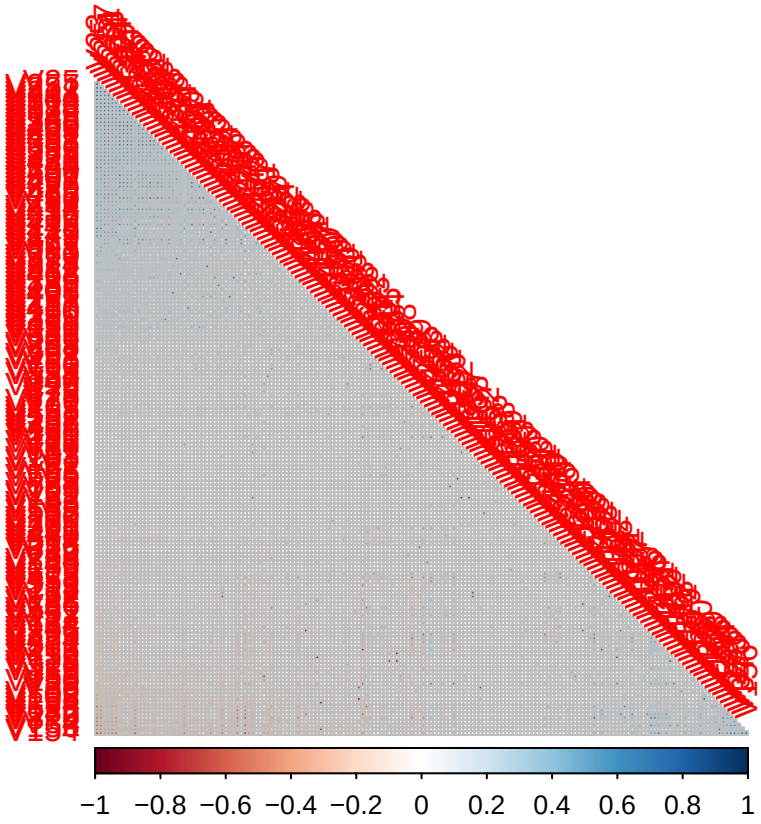
As mentioned before, the size of the array of the unique entries with a correlation coefficient larger than 0.95 in absolute value is 1749, thus, we will remove the features that are highly correlated as co-linearity does not provide us more or only provides limited information. \

```
col_rm <- findCorrelation(cor_mat, cutoff=0.8)   # Get the mask:  columms to remove
col_name <- colnames(X_train_unlabelled)                    # Get all the features
X_train <- X_train_unlabelled[-col_rm]             # Apply the mask
X_test <- X_test_unlabelled[-col_rm]
cor_mat_selected <- cor(X_train)        # Get the new correlation matrix
cat('The number of features left after removal of highly correlated features is: \n' , dim(cor_mat_sele
```

```
## The number of features left after removal of highly correlated features is:
##  173
```

From the output above, we can see that only 173 features are below the threshold and we can plot again the correlation plot. \

```
corrplot(cor_mat_selected, method = 'square', order = 'FPC', type = 'lower', diag = FALSE)
```



From the new plot, we can see that the pattern of the correlation between different features can be seen now, which provides another evidence to the speculation made before. \

Since some of our model has the assumption of normal distributed and some techniques such as PCA that is sensitive to the variance of the variables, the dataset will be standardized first. \

Principle Component Analysis (PCA)

With the method above, we removed the variables that are highly correlated, however, we could also apply the Principle Component Analysis (PCA). PCA is a really useful dimension reduction technique in statistics as it finds the directions (the principle components) so that by projecting the data onto the principle components, the maximum variance can be preserved or in another word, the maximum amount information can be preserved. As it searches for the maximum variance, thus it can be sensitive to the variance, thus we will standardize the data. \

```
X_train_std <- scale(X_train_ori[, -ncol(X_train_ori)])
X_test_std <- scale(X_test_ori[, -ncol(X_test_ori)])
```

```
set.seed(123)
pca_train <- prcomp(X_train_std, center = TRUE, scale. = TRUE)
cumsum(pca_train$sdev^2 / sum(pca_train$sdev^2))
```

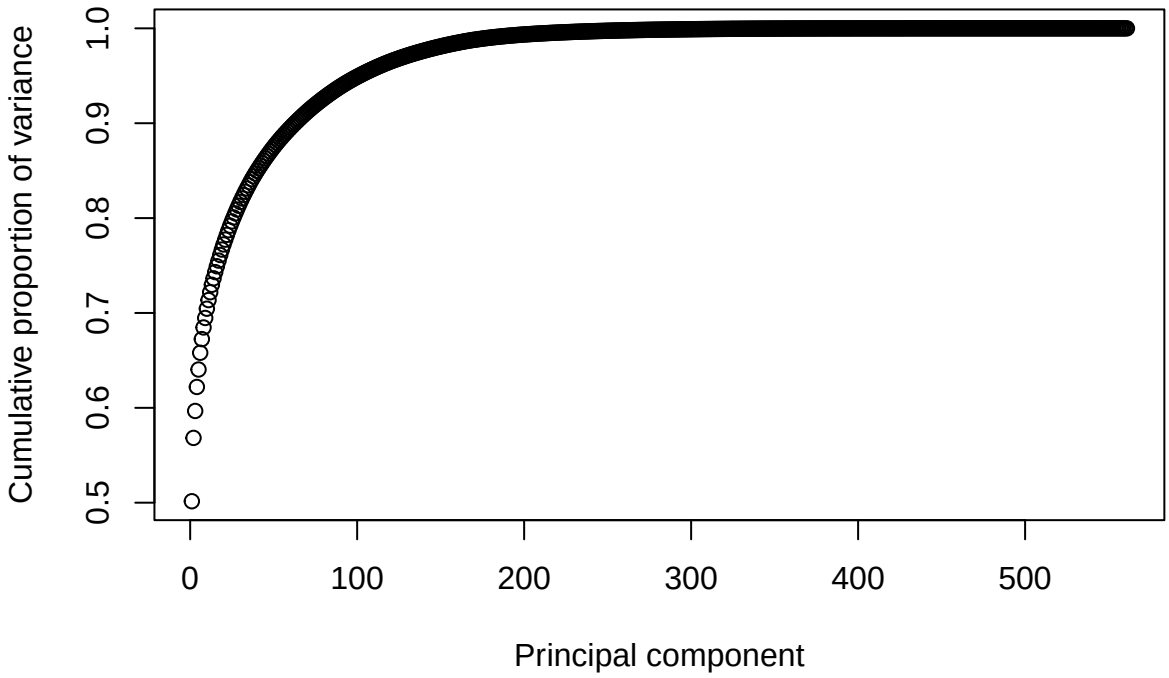
```
##   [1] 0.5015346 0.5682887 0.5966956 0.6219874 0.6403826 0.6580244 0.6724202
```

```
##   [8] 0.6847195 0.6947112 0.7045048 0.7135884 0.7218382 0.7296934 0.7365158
##  [15] 0.7430401 0.7492128 0.7551955 0.7610798 0.7668065 0.7721944 0.7772881
##  [22] 0.7823287 0.7872437 0.7919932 0.7964867 0.8007235 0.8049435 0.8090320
##  [29] 0.8130133 0.8169765 0.8206860 0.8242556 0.8277438 0.8311550 0.8344986
##  [36] 0.8377913 0.8410071 0.8439905 0.8468946 0.8497663 0.8524952 0.8551954
##  [43] 0.8578458 0.8604441 0.8629901 0.8654502 0.8678929 0.8703052 0.8726329
##  [50] 0.8748972 0.8771324 0.8792951 0.8814284 0.8834917 0.8855028 0.8874970
##  [57] 0.8894425 0.8913560 0.8932492 0.8950986 0.8969174 0.8986913 0.9004505
##  [64] 0.9021874 0.9038955 0.9055733 0.9072475 0.9088716 0.9104619 0.9120430
##  [71] 0.9136109 0.9151509 0.9166498 0.9180991 0.9195402 0.9209620 0.9223550
##  [78] 0.9237364 0.9250991 0.9264358 0.9277439 0.9290420 0.9303340 0.9315858
##  [85] 0.9328297 0.9340403 0.9352424 0.9364346 0.9376045 0.9387318 0.9398399
##  [92] 0.9409398 0.9420211 0.9430736 0.9440959 0.9451012 0.9461004 0.9470859
##  [99] 0.9480387 0.9489775 0.9499063 0.9508238 0.9517287 0.9526114 0.9534846
## [106] 0.9543496 0.9552007 0.9560291 0.9568499 0.9576668 0.9584779 0.9592742
## [113] 0.9600430 0.9608092 0.9615580 0.9623001 0.9630303 0.9637346 0.9644239
## [120] 0.9651106 0.9657770 0.9664269 0.9670664 0.9676979 0.9683160 0.9689241
## [127] 0.9695240 0.9701147 0.9706951 0.9712710 0.9718322 0.9723826 0.9729220
## [134] 0.9734543 0.9739735 0.9744906 0.9750014 0.9755010 0.9759995 0.9764872
## [141] 0.9769727 0.9774442 0.9779112 0.9783680 0.9788214 0.9792651 0.9797073
## [148] 0.9801419 0.9805651 0.9809823 0.9813921 0.9817977 0.9821950 0.9825877
## [155] 0.9829703 0.9833492 0.9837227 0.9840869 0.9844462 0.9848000 0.9851498
## [162] 0.9854884 0.9858239 0.9861537 0.9864755 0.9867815 0.9870861 0.9873785
## [169] 0.9876563 0.9879266 0.9881893 0.9884412 0.9886918 0.9889354 0.9891752
## [176] 0.9894036 0.9896290 0.9898526 0.9900653 0.9902732 0.9904791 0.9906790
## [183] 0.9908766 0.9910655 0.9912506 0.9914317 0.9916068 0.9917773 0.9919450
## [190] 0.9921078 0.9922645 0.9924202 0.9925725 0.9927202 0.9928635 0.9930052
## [197] 0.9931462 0.9932780 0.9934082 0.9935346 0.9936577 0.9937778 0.9938952
## [204] 0.9940097 0.9941197 0.9942293 0.9943369 0.9944399 0.9945394 0.9946380
## [211] 0.9947361 0.9948307 0.9949248 0.9950169 0.9951068 0.9951958 0.9952839
## [218] 0.9953716 0.9954566 0.9955400 0.9956219 0.9957020 0.9957808 0.9958585
## [225] 0.9959357 0.9960115 0.9960852 0.9961573 0.9962279 0.9962969 0.9963644
## [232] 0.9964319 0.9964986 0.9965649 0.9966287 0.9966916 0.9967538 0.9968142
## [239] 0.9968745 0.9969333 0.9969917 0.9970490 0.9971052 0.9971599 0.9972135
## [246] 0.9972667 0.9973192 0.9973707 0.9974220 0.9974713 0.9975200 0.9975681
## [253] 0.9976147 0.9976609 0.9977059 0.9977499 0.9977928 0.9978355 0.9978772
## [260] 0.9979182 0.9979591 0.9979992 0.9980384 0.9980772 0.9981156 0.9981531
## [267] 0.9981899 0.9982256 0.9982610 0.9982956 0.9983295 0.9983624 0.9983953
## [274] 0.9984275 0.9984588 0.9984893 0.9985190 0.9985483 0.9985774 0.9986062
## [281] 0.9986346 0.9986626 0.9986900 0.9987166 0.9987431 0.9987692 0.9987950
## [288] 0.9988205 0.9988456 0.9988702 0.9988944 0.9989179 0.9989411 0.9989638
## [295] 0.9989862 0.9990083 0.9990301 0.9990514 0.9990726 0.9990935 0.9991139
## [302] 0.9991340 0.9991539 0.9991731 0.9991920 0.9992107 0.9992293 0.9992476
## [309] 0.9992658 0.9992838 0.9993012 0.9993186 0.9993351 0.9993516 0.9993678
## [316] 0.9993838 0.9993995 0.9994151 0.9994305 0.9994457 0.9994607 0.9994755
## [323] 0.9994899 0.9995040 0.9995178 0.9995314 0.9995445 0.9995575 0.9995703
## [330] 0.9995826 0.9995948 0.9996070 0.9996189 0.9996303 0.9996414 0.9996525
## [337] 0.9996633 0.9996736 0.9996837 0.9996936 0.9997033 0.9997128 0.9997222
## [344] 0.9997313 0.9997401 0.9997485 0.9997568 0.9997649 0.9997727 0.9997803
## [351] 0.9997879 0.9997951 0.9998019 0.9998086 0.9998153 0.9998218 0.9998280
## [358] 0.9998340 0.9998399 0.9998458 0.9998512 0.9998565 0.9998616 0.9998666
## [365] 0.9998715 0.9998762 0.9998808 0.9998852 0.9998896 0.9998938 0.9998979
## [372] 0.9999019 0.9999058 0.9999096 0.9999134 0.9999170 0.9999205 0.9999239
## [379] 0.9999272 0.9999305 0.9999335 0.9999364 0.9999391 0.9999417 0.9999443
```

```
## [386] 0.9999468 0.9999492 0.9999514 0.9999535 0.9999556 0.9999577 0.9999596
## [393] 0.9999616 0.9999635 0.9999651 0.9999667 0.9999683 0.9999698 0.9999712
## [400] 0.9999725 0.9999738 0.9999751 0.9999763 0.9999775 0.9999786 0.9999796
## [407] 0.9999806 0.9999816 0.9999825 0.9999834 0.9999843 0.9999851 0.9999859
## [414] 0.9999866 0.9999874 0.9999881 0.9999888 0.9999894 0.9999901 0.9999907
## [421] 0.9999913 0.9999919 0.9999925 0.9999930 0.9999935 0.9999940 0.9999945
## [428] 0.9999949 0.9999954 0.9999958 0.9999962 0.9999966 0.9999969 0.9999973
## [435] 0.9999976 0.9999979 0.9999982 0.9999985 0.9999987 0.9999990 0.9999992
## [442] 0.9999994 0.9999995 0.9999996 0.9999997 0.9999997 0.9999997 0.9999998
## [449] 0.9999998 0.9999998 0.9999998 0.9999999 0.9999999 0.9999999 0.9999999
## [456] 0.9999999 0.9999999 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [463] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [470] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [477] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [484] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [491] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [498] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [505] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [512] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [519] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [526] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [533] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [540] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [547] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [554] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [561] 1.0000000
```

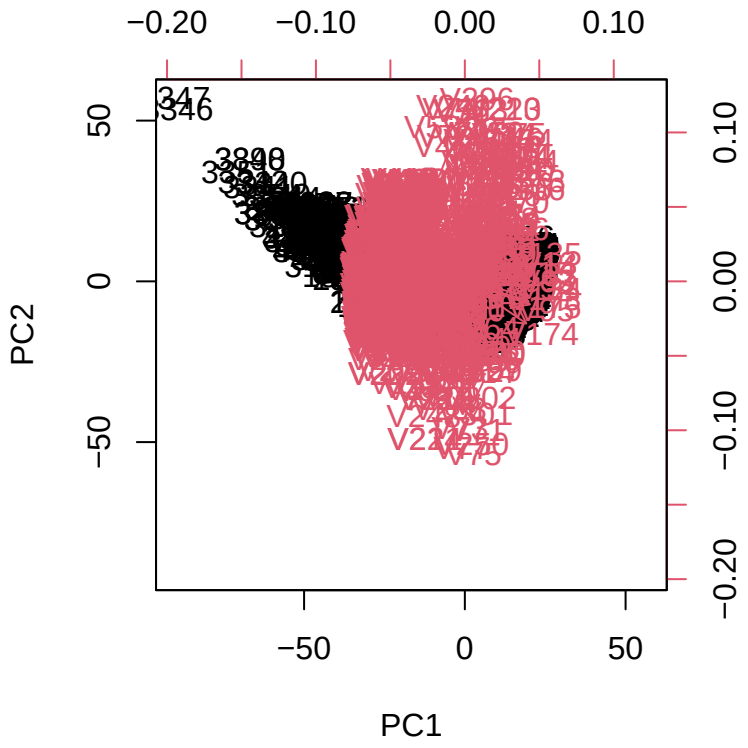
From the result above, we can see that with the first 180 principle components, around 99 percent of the variance can be preserved and we can see it better with the following plot. \

```
plot(cumsum(pca_train$sdev^2/sum(pca_train$sdev^2)), xlab = "Principal component", ylab = "Cumulative pr
```



Furthermore, we can use the biplot to get an intuition how PCA projects the dataset onto the first two principle components. \

```
biplot(pca_train, scale = 0)
```



Now, we project our dataset onto the first 180 principle components.

```
X_train_pca <- as.data.frame(predict(pca_train, newdata = X_train_std)[, 1:180])
X_test_pca <- as.data.frame(predict(pca_train, newdata = X_test_std)[, 1:180])
```

Now, we can put the assign again the labels of the data.

```
X_train <- cbind(X_train, y_train)
X_test <- cbind(X_test, y_test)
X_train_pca <- cbind(X_train_pca, y_train)
X_test_pca <- cbind(X_test_pca, y_test)
X_train[1:5, ncol(X_train)]
```

```
## [1] "WALKING" "WALKING" "WALKING" "WALKING" "WALKING"
```

```
X_test[1:5, ncol(X_test)]
```

```
## [1] "WALKING" "WALKING" "WALKING" "WALKING" "WALKING"
```

```
X_train_pca[1:5, ncol(X_train_pca)]
```

```
## [1] "WALKING" "WALKING" "WALKING" "WALKING" "WALKING"
```

```
X_test_pca[1:5, ncol(X_test_pca)]
```

```
## [1] "WALKING" "WALKING" "WALKING" "WALKING" "WALKING"
```

Multiclass classification

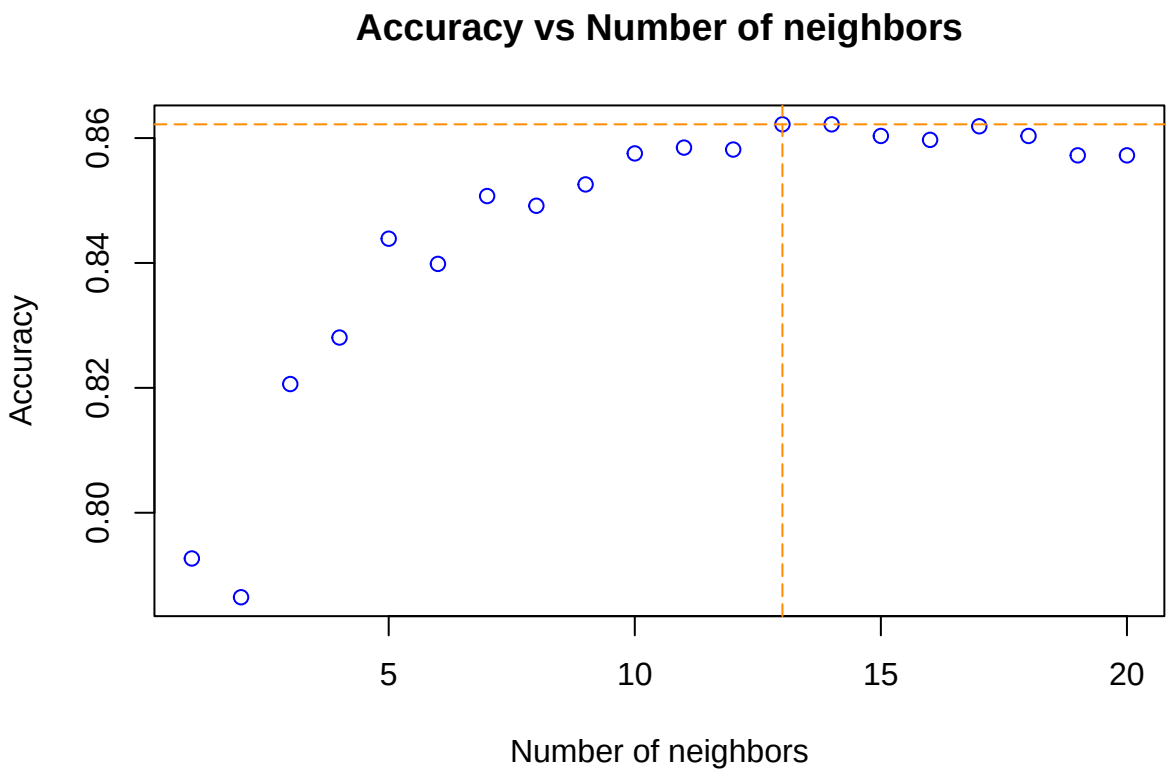
With the removal of highly correlated features and PCA, we obtain two different dataset, and now let's compare which of the preprocessing technique can perform with different models. Since we are facing a multi-class classification problem, thus models such as the multinomial Logistic Regression, LDA are used. As for the evaluation of the model performance, the mean error and the confusion matrix are used as they are easy to implement and also powerful for the assessment of the classification task. \

K-Nearest Neighbor (KNN)

The KNN model is a model that clusters the dataset by considering the labels of its K closest neighbors. However, the number of neighbors to be considered is a hyper-parameter and thus we will run a for loop between 1 and 20 to get the best number of neighbors to be considered. \

First, we use the dataset that removed the highly correlated features.

```
set.seed(123)
accuracy_KNN <- c()              # accuracy vector
k_closest<- c()                  # number of nearest neighbors
for(k in seq(1,20)) {
pred_KNN <- kNN(Activity ~ .,train = X_train, test = X_test,  k = k)
accuracy_KNN <- append(accuracy_KNN, mean(as.matrix(pred_KNN) == y_test))
k_closest <- append(k_closest, k)
}
plot(k_closest, accuracy_KNN, type = "p", col="blue", xlab="Number of neighbors", ylab="Accuracy", main=
abline(h = max(accuracy_KNN), v = which.max(accuracy_KNN), col = "darkorange", lty = 5)
```




```
cat('The optimal number of neighbors to be considered is: ', which.max(accuracy_KNN))
```

```
## The optimal number of neighbors to be considered is:  13
```

```
cat('\n\n')
```

```
cat('The best accuracy of KNN is: ', max(accuracy_KNN))
```

```
## The best accuracy of KNN is:  0.8621974
```

We can see that for our dataset, the number of neighbors to be considered is 13, and with 13 nearest neighbors, the KNN model reaches an accuracy of 0.86. In order to check the specific mistakes that is made by the model, we can create a 6X6 confusion matrix which can be achieved with the 'table' function in R. \

```
set.seed(123)
pred_KNN_best <- kNN(Activity ~ .,train = X_train, test = X_test,  k = which.max(accuracy_KNN))  # fit
table(X_test$Activity, pred_KNN_best)
```

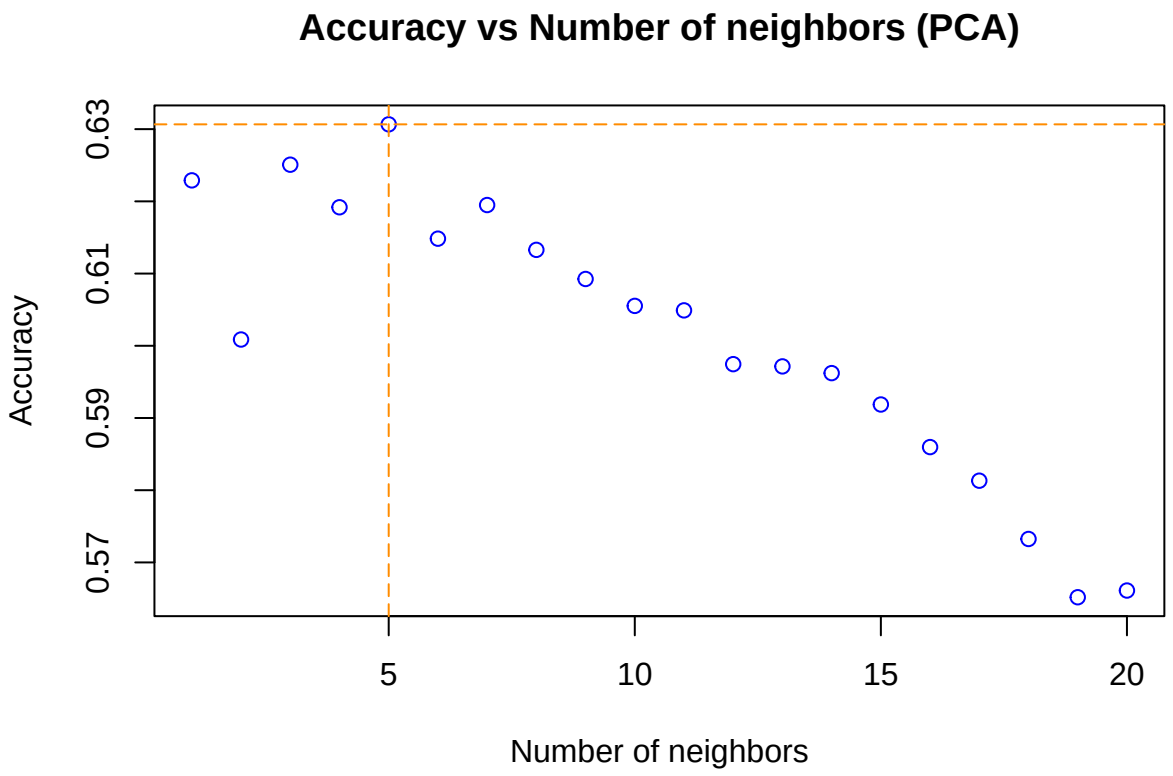
```
##                     pred_KNN_best
##                      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING                479      43       14       0                  0
##   SITTING                 4     416      113       0                  0
##   STANDING                0      46      491       0                  0
```

```
##   WALKING                 0       0        0     520                   9
##   WALKING_DOWNSTAIRS      0       0        0      82                 398
##   WALKING_UPSTAIRS        0       0        0      53                  10
##                     pred_KNN_best
##                      WALKING_UPSTAIRS
##   LAYING                           1
##   SITTING                          4
##   STANDING                         0
##   WALKING                          8
##   WALKING_DOWNSTAIRS              57
##   WALKING_UPSTAIRS               474
```

From the confusion matrix we can see that the model can separate really well the WALKING and NON-WALKING(LAYING, SITTING, STANDING) activities as most of the entries for the upper triangular and lower triangular parts of the confusion matrix are 0. If we check further the sub-blocks of the WALKING and NON-WALKING activities, we can see that sometimes the model incorrectly labels 'SITTING' as 'STANDING'. As for the NON-WALKING sub-block, the model can label pure 'WALKING' activity almost perfectly, but sometimes it has difficulty identifying walking on the stairs especially 'WALKING_DOWNSTAIRS' as around 20 percent of the time it is labelled as other type of walking activities. \

Let's see the dataset selected according to PCA. \

```
set.seed(123)
accuracy_KNN_pca <- c()              # accuracy vector
k_closest_pca<- c()                  # number of nearest neighbors
for(k in seq(1,20)) {
pred_KNN_pca <- kNN(Activity ~ .,train = X_train_pca, test = X_test_pca,  k = k)
accuracy_KNN_pca <- append(accuracy_KNN_pca, mean(as.matrix(pred_KNN_pca) == y_test))
k_closest_pca <- append(k_closest_pca, k)
}
plot(k_closest, accuracy_KNN_pca, type = "p", col="blue", xlab="Number of neighbors", ylab="Accuracy", m
abline(h = max(accuracy_KNN_pca), v = which.max(accuracy_KNN_pca), col = "darkorange", lty = 5)
```




```
cat('The optimal number of neighbors to be considered for PCA dataset is: ', which.max(accuracy_KNN_pca
```

```
## The optimal number of neighbors to be considered for PCA dataset is:  5
```

```
cat('\n\n')
```

```
cat('The best accuracy of KNN for PCA dataset is: ', max(accuracy_KNN_pca))
```

```
## The best accuracy of KNN for PCA dataset is:  0.6306642
```

```
set.seed(123)
pred_KNN_best_pca <- kNN(Activity ~ .,train = X_train_pca, test = X_test_pca,  k = which.max(accuracy_K
table(X_test_pca$Activity, pred_KNN_best_pca)
```

```
##                     pred_KNN_best_pca
##                      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING                417      83       37       0                  0
##   SITTING                 3     415      116       0                  0
##   STANDING                2      71      464       0                  0
##   WALKING                 3     141       99     235                 13
##   WALKING_DOWNSTAIRS      2     101      109      73                175
##   WALKING_UPSTAIRS        1      60      140      15                  4
##                     pred_KNN_best_pca
```

```
##                       WALKING_UPSTAIRS
##   LAYING                             0
##   SITTING                            3
##   STANDING                           0
##   WALKING                           46
##   WALKING_DOWNSTAIRS                77
##   WALKING_UPSTAIRS                 317
```

The optimal number of neighbors for the PCA dataset is 5, however, the result we obtained shows that KNN performs poorly on the dataset selected with PCA and the separability of WALKING and NON-WALKING blocks is not present.

Multinomial Logistic Regression

As mentioned before, the labels are non ordinal, thus we chose the multinomial logistic regression as it is a good option for this type of problem. Now we can fit the multinomial logistic regression model.

```
set.seed(123)
fit_mul_log <- multinom(Activity ~ ., data = X_train, MaxNWts=10000, maxit=1000)  # fit the model
```

```
## # weights:  1050 (870 variable)
## initial  value 15126.033439
## iter  10 value 4013.232234
## iter  20 value 1229.419156
## iter  30 value 682.106354
## iter  40 value 511.149962
## iter  50 value 463.433609
## iter  60 value 434.991241
## iter  70 value 411.264471
## iter  80 value 388.558500
## iter  90 value 371.112752
## iter 100 value 357.990490
## iter 110 value 347.140407
## iter 120 value 339.222562
## iter 130 value 332.029280
## iter 140 value 324.488310
## iter 150 value 318.813468
## iter 160 value 315.144497
## iter 170 value 312.377195
## iter 180 value 309.339072
## iter 190 value 305.499417
## iter 200 value 302.224942
## iter 210 value 299.483614
## iter 220 value 296.663226
## iter 230 value 293.734672
## iter 240 value 290.374630
## iter 250 value 285.887464
## iter 260 value 282.811309
## iter 270 value 278.850136
## iter 280 value 276.183607
## iter 290 value 274.281846
## iter 300 value 272.842774
```

```
## iter 310 value 272.150936
## iter 320 value 271.411545
## iter 330 value 270.690972
## iter 340 value 269.932574
## iter 350 value 268.913963
## iter 360 value 267.794707
## iter 370 value 266.801890
## iter 380 value 265.947756
## iter 390 value 265.204363
## iter 400 value 264.681622
## iter 410 value 264.082223
## iter 420 value 263.551409
## iter 430 value 263.133259
## iter 440 value 262.884297
## iter 450 value 262.673845
## iter 460 value 262.476856
## iter 470 value 262.284564
## iter 480 value 262.162440
## iter 490 value 261.960450
## iter 500 value 261.744883
## iter 510 value 261.583906
## iter 520 value 261.410381
## iter 530 value 261.220249
## iter 540 value 261.001499
## iter 550 value 260.799501
## iter 560 value 260.661681
## iter 570 value 260.475504
## iter 580 value 260.382790
## iter 590 value 260.300593
## iter 600 value 260.238971
## iter 610 value 260.142732
## iter 620 value 260.008411
## iter 630 value 259.880042
## iter 640 value 259.739969
## iter 650 value 259.586241
## iter 660 value 259.467039
## iter 670 value 259.266739
## iter 680 value 259.118995
## iter 690 value 258.961041
## iter 700 value 258.861313
## iter 710 value 258.787219
## iter 720 value 258.716160
## iter 730 value 258.661396
## iter 740 value 258.533689
## iter 750 value 258.460854
## iter 760 value 258.297811
## iter 770 value 258.191950
## iter 780 value 258.114353
## iter 790 value 258.021672
## iter 800 value 257.911798
## iter 810 value 257.837642
## iter 820 value 257.730023
## iter 830 value 257.616445
## iter 840 value 257.545842
```

```
## iter 850 value 257.475542
## iter 860 value 257.427993
## iter 870 value 257.386502
## iter 880 value 257.337513
## iter 890 value 257.297358
## iter 900 value 257.278363
## iter 910 value 257.261023
## iter 920 value 257.233011
## iter 930 value 257.190621
## iter 940 value 257.155120
## iter 950 value 257.141075
## iter 960 value 257.122256
## iter 970 value 257.098799
## iter 980 value 257.065989
## iter 990 value 257.021809
## iter1000 value 256.989287
## final  value 256.989287
## stopped after 1000 iterations
```

```
pred_mul_log <- predict(fit_mul_log, X_test[,-ncol(X_test)], type="class")       # prediction
accuray_mul_log <- mean(as.matrix(pred_mul_log) == y_test)                       # get the accuracy sco
cat('\n')
```

```
cat('The accuracy of the multinomial logistic regression is: \n', accuray_mul_log)
```

```
## The accuracy of the multinomial logistic regression is:
##  0.9028554
```

From the output we can see that the algorithm reaches the preset maximum iterations, but if we check the loss value, the drop is quite small and hence even if we allow more iterations, the improvement will be not large. Even without arriving at the optimum value, the model still manages to fit the data extremely well as it gets an accuracy of more than 0.90. Let's check the confusion matrix for a better understanding of the errors. \

```
table(X_test$Activity, pred_mul_log)
```

```
##                     pred_mul_log
##                      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING                535       1        0       0                   1
##   SITTING                 0     472       64       1                   0
##   STANDING                3      39      493       2                   0
##   WALKING                10       0        1     450                  44
##   WALKING_DOWNSTAIRS      4       0       20       6                 477
##   WALKING_UPSTAIRS        6       1       16      21                  11
##                     pred_mul_log
##                      WALKING_UPSTAIRS
##   LAYING                            0
##   SITTING                           0
##   STANDING                          0
##   WALKING                          32
##   WALKING_DOWNSTAIRS               30
##   WALKING_UPSTAIRS                482
```

We can see that all the activities can be labelled correctly. Especially the 'LAYING' activity which can be labelled almost perfectly. Furthermore, unlike the result with KNN, the separability between WALKING and NON-WALKING activities is not as significant but it happens rarely as most of those errors happen to be in the same block. \

```
set.seed(123)
fit_mul_log_pca <- multinom(Activity ~ ., data = X_train_pca, MaxNWts=10000, maxit=1000)  # fit the mod
```

```
## # weights:  1092 (905 variable)
## initial  value 15126.033439
## iter  10 value 3188.267306
## iter  20 value 2745.669256
## iter  30 value 2542.255209
## iter  40 value 2393.276889
## iter  50 value 2147.852793
## iter  60 value 1412.283356
## iter  70 value 993.528809
## iter  80 value 858.811585
## iter  90 value 768.794634
## iter 100 value 697.629135
## iter 110 value 595.109074
## iter 120 value 450.403992
## iter 130 value 345.507161
## iter 140 value 312.547268
## iter 150 value 286.444257
## iter 160 value 268.593131
## iter 170 value 249.013500
## iter 180 value 237.346454
## iter 190 value 226.062797
## iter 200 value 220.244011
## iter 210 value 212.647666
## iter 220 value 205.765399
## iter 230 value 198.361366
## iter 240 value 190.653967
## iter 250 value 184.300023
## iter 260 value 176.103041
## iter 270 value 168.762777
## iter 280 value 158.612565
## iter 290 value 149.799898
## iter 300 value 140.187555
## iter 310 value 132.023636
## iter 320 value 122.696600
## iter 330 value 113.951423
## iter 340 value 108.622333
## iter 350 value 103.025470
## iter 360 value 98.287278
## iter 370 value 92.240404
## iter 380 value 87.673356
## iter 390 value 83.635189
## iter 400 value 77.685386
## iter 410 value 73.436531
## iter 420 value 67.765254
## iter 430 value 61.717144
## iter 440 value 50.245482
```

```
## iter 450 value 30.034777
## iter 460 value 0.551787
## iter 470 value 0.007456
## final  value 0.000081
## converged
```

```
pred_mul_log_pca <- predict(fit_mul_log_pca, X_test_pca[,-ncol(X_test_pca)], type="class")       # pred
accuray_mul_log_pca <- mean(as.matrix(pred_mul_log_pca) == y_test)                       # get the accu
cat('\n')
```

```
cat('The accuracy of the multinomial logistic regression of PCA dataset is: \n', accuray_mul_log_pca)
```

```
## The accuracy of the multinomial logistic regression of PCA dataset is:
##  0.7759156
```

```
table(X_test_pca$Activity, pred_mul_log_pca)
```

```
##                     pred_mul_log_pca
##                      LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING                493       1        0       6                   8
##   SITTING                10     510        3       6                   5
##   STANDING               13     368      105      21                  12
##   WALKING                21      13        3     481                  14
##   WALKING_DOWNSTAIRS     15      21       10       6                 464
##   WALKING_UPSTAIRS       18      28        6      24                  14
##                     pred_mul_log_pca
##                      WALKING_UPSTAIRS
##   LAYING                           29
##   SITTING                           3
##   STANDING                         18
##   WALKING                           5
##   WALKING_DOWNSTAIRS               21
##   WALKING_UPSTAIRS                447
```

We can see that the accuracy has improved a lot compared to KNN as an accuracy of 0.77 is achieved, however, it is still much lower compared to the dataset after the removal of highly correlated features. Furthermore, we can also see that the classification of the 'STANDING' activity is really bad as more than 70 percent of the time it is labelled wrongly. \

Linear Discriminant Analysis (LDA)

The next possible model is the Linear Discriminant Analysis which searches a linear combination of features to perform the task.

```
set.seed(123)
fit_lda <- lda(X_train$Activity ~ ., data = X_train)
pred_lda <- predict(fit_lda, X_test, type = "class")
accuray_lda <- mean(as.matrix(pred_lda$class) == y_test)
cat('\n')
```

```
cat('The accuracy with LDA is: \n', accuray_lda)
```

```
## The accuracy with LDA is:
##  0.9413408
```

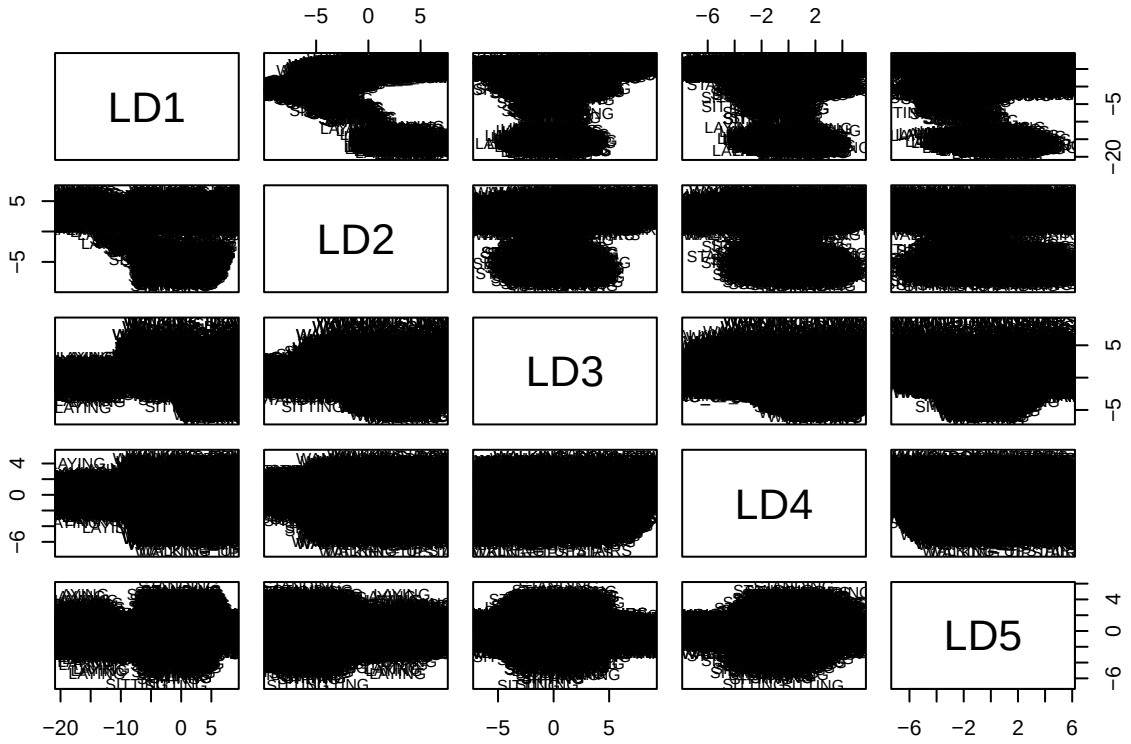
The result is so far the best of all the models, it reaches more than 0.94 of the accuracy. Let's have a look at the confusion matrix and see if there is any difference with previous ones. \

```
table(X_test$Activity, pred_lda$class)
```

```
##
##                     LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING               537       0        0       0                   0
##   SITTING                0     468       68       0                   0
##   STANDING               0      38      499       0                   0
##   WALKING                0       0        0     520                  10
##   WALKING_DOWNSTAIRS     0       0        0      10                 514
##   WALKING_UPSTAIRS       0       0        0      33                   9
##
##                     WALKING_UPSTAIRS
##   LAYING                           0
##   SITTING                          1
##   STANDING                         0
##   WALKING                          7
##   WALKING_DOWNSTAIRS              13
##   WALKING_UPSTAIRS               495
```

We can see that the confusion matrix is similar to the others, the performance of the classification of 'WALKING_DOWNSTAIRS' has improved.

```
set.seed(123)
plot(fit_lda)
```




```
set.seed(123)
fit_lda_pca <- lda(X_train_pca$Activity ~ ., data = X_train_pca)
pred_lda_pca <- predict(fit_lda_pca, X_test_pca, type = "class")
accuray_lda_pca <- mean(as.matrix(pred_lda_pca$class) == y_test)
cat('\n')
```

```
cat('The accuracy with LDA for PCA dataset is: \n', accuray_lda_pca)
```

```
## The accuracy with LDA for PCA dataset is:
##  0.955928
```

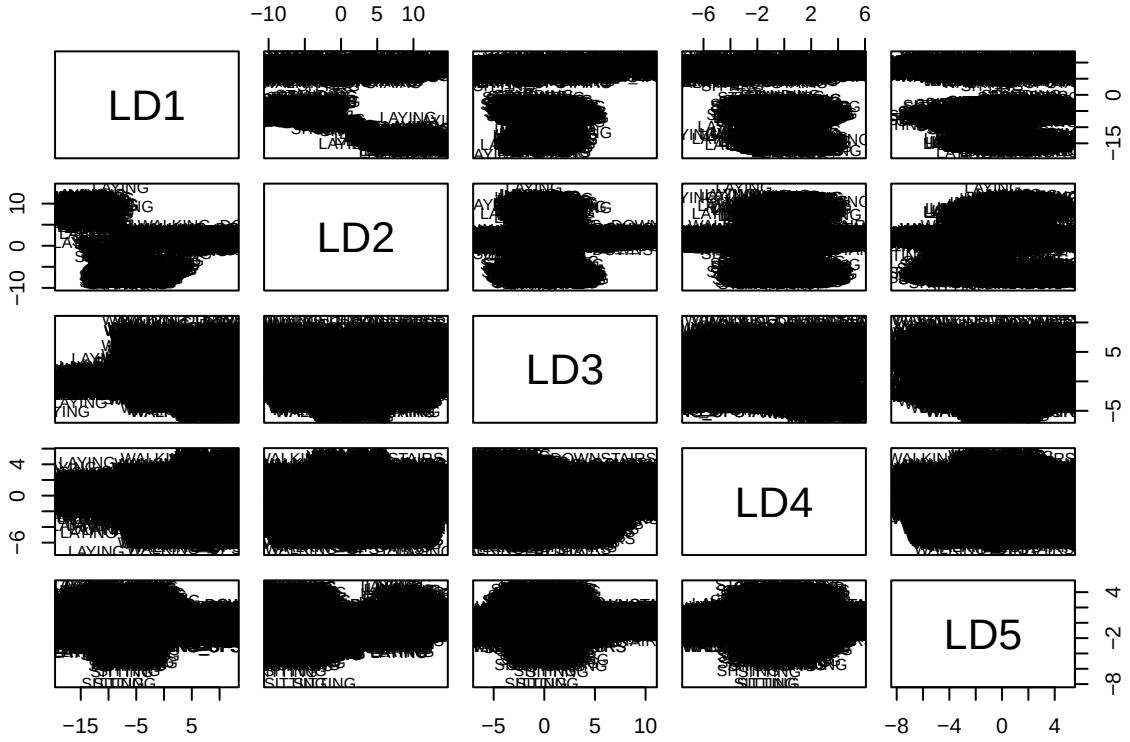
```
table(X_test_pca$Activity, pred_lda_pca$class)
```

```
##
##                     LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING               537       0        0       0                   0
##   SITTING                0     479       57       0                   0
##   STANDING               0      50      487       0                   0
##   WALKING                0       0        0     533                   1
##   WALKING_DOWNSTAIRS     0       0        0       3                 519
##   WALKING_UPSTAIRS       0       0        0      11                   1
##
##                     WALKING_UPSTAIRS
##   LAYING                           0
```

```
##    SITTING                          1
##    STANDING                         0
##    WALKING                          3
##    WALKING_DOWNSTAIRS              15
##    WALKING_UPSTAIRS               525
```

With the PCA dataset, the LDA performs better, and comparing with other models over the PCA dataset, we can see that LDA is way better than other models. Furthermore, the separation between the WALKIND and NON-WALKING blocks can be observed clearly. Comparing with LDA over the other dataset, we can see that the WALKING block can be predicted better as less errors are made over this block. \

```
set.seed(123)
plot(fit_lda_pca)
```



Quadratic Discriminant Analysis (QDA)

Another technique that is similar to LDA is QDA which stands for Quadratic Linear Discriminant Analysis. Unlike the linear one, QDA searches for the quadratic combinations of the features.

```
set.seed(123)
fit_qda <- qda(X_train$Activity ~ ., data = X_train)
pred_qda <- predict(fit_qda, X_test, type = "class")
accuray_qda <- mean(as.matrix(pred_qda$class) == y_test)
cat('\n')
```

```
cat('The accuracy with QDA is: \n', accuray_qda)
```

```
## The accuracy with QDA is:
##  0.9146493
```

The result is also really well, however, it is slightly worse than the LDA which might indicate that our dataset is more 'linear' than 'quadratic'. \

```
table(X_test$Activity, pred_qda$class)
```

```
##
##                     LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING               535       1        0       0                   0
##   SITTING                1     459       69       0                   3
##   STANDING               0      45      465       5                   6
##   WALKING                0       0        0     441                  84
##   WALKING_DOWNSTAIRS     0       0        0       2                 519
##   WALKING_UPSTAIRS       0       0        0       2                   7
##
##                     WALKING_UPSTAIRS
##   LAYING                           1
##   SITTING                          5
##   STANDING                        16
##   WALKING                         12
##   WALKING_DOWNSTAIRS              16
##   WALKING_UPSTAIRS               528
```

The confusion matrix again shows some separability of the different activity blocks, we can see that sometimes the mis-classifications of the NON-WALKING activities are made on the WALKING activities, and all the errors are made within the same activity block. \

```
set.seed(123)
fit_qda_pca <- qda(X_train_pca$Activity ~ ., data = X_train_pca)
pred_qda_pca <- predict(fit_qda_pca, X_test_pca, type = "class")
accuray_qda_pca <- mean(as.matrix(pred_qda_pca$class) == y_test)
cat('\n')
```

```
cat('The accuracy with QDA for PCA dataset is: \n', accuray_qda_pca)
```

```
## The accuracy with QDA for PCA dataset is:
##  0.7647424
```

```
table(X_test_pca$Activity, pred_qda_pca$class)
```

```
##
##                     LAYING SITTING STANDING WALKING WALKING_DOWNSTAIRS
##   LAYING               537       0        0       0                   0
##   SITTING               71       0      446       0                   1
##   STANDING               2       0      500       1                   3
##   WALKING                0       0        0     380                 150
```

```
##   WALKING_DOWNSTAIRS      0       0        0       0                519
##   WALKING_UPSTAIRS        0       0        0       0                  9
##
##                       WALKING_UPSTAIRS
##   LAYING                             0
##   SITTING                           19
##   STANDING                          31
##   WALKING                            7
##   WALKING_DOWNSTAIRS                18
##   WALKING_UPSTAIRS                 528
```

We can see for QDA, the PCA dataset is also not a good option and again it failed to label the 'SITTING' activity. \

Summary

From the results obtained above, we can conclude that overall the dataset with removal of highly correlated features is a better preprocessing choice as all of our models perform well on that dataset and all of them show the separability between the WALKING and NON-WALKING blocks. However, it is worth stating that there is also some activities that can be more likely to be wrongly predicted as other activities, such as: 'SITTING' are misclassified as 'STANDING' and vice versa, 'WALKING' are misclassified as 'WALKING_DOWNSTAIRS', but this phenomenon is not really problematic as it rarely happens. On the other hand, for the data selected with PCA, overall it performs poorly compared to the other one expect with LDA which actually outperforms the same method with the other dataset. However, other than that, all the models fail to predict well with PCA dataset and actually with the multinomial logistic regression and QDA, both models fail completely to predict 'STANDING' and 'SITTING' activities respectively. \

Thus, as a final conclusion of this chapter, it is better to preprocess the dataset by simply removing the highly correlated features it provides a better dataset for the performance of different models. If the correct prediction of the WALKING block activities such as 'WALKING', 'WALKING_DOWNSTAIRS', and 'WALKING_UPSTAIRS', it is advised to preprocess with PCA and use LDA for the prediction as this combination rarely makes mistake on this block. \

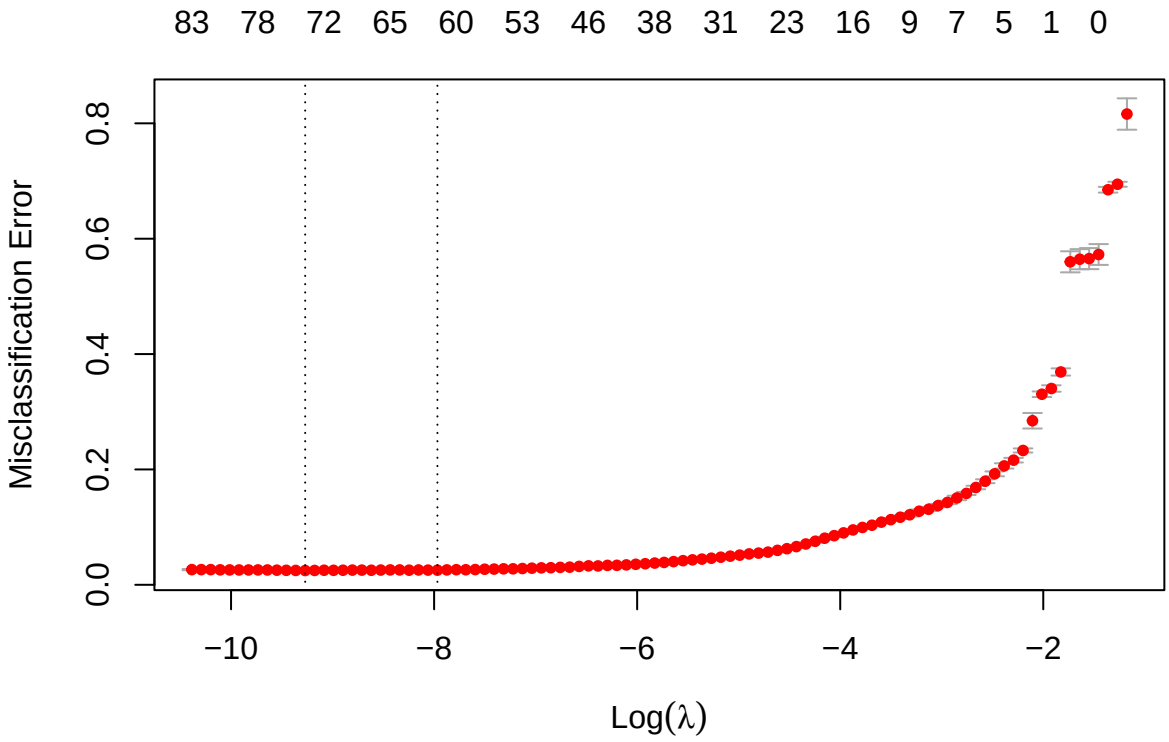
Feature selection and Regularization

As discussed above, if we preprocess the dataset by removing highly correlated features, all of our models can perform well, thus we will use this dataset for the subsequent task. Now, we want to apply some regularization terms so that the model can generalize well, furthermore, depending on the method we use, it is also possible to select the most relevant features for the prediction. The main regularization approaches we want to try are: Ridge, LASSO, and Elastic-net. As all of them requires hyper-parameters thus we will use the cross-validation technique to find them. \

LASSO

LASSO is one of the most popular regularization in Machine Learning and Statistics. It used the L1 penalty terms to force the sparsity which can be also considered as a sort of feature selection technique. \

```
set.seed(123)
cv_fit_lasso <- cv.glmnet(data.matrix(X_train[,-ncol(X_train)]), data.matrix(y_train), family= 'multinor
plot(cv_fit_lasso)
```




```
cv_fit_lasso
```

```
##
## Call:  cv.glmnet(x = data.matrix(X_train[, -ncol(X_train)]), y = data.matrix(y_train),      type.meas
##
## Measure: Misclassification Error
##
##        Lambda Index Measure        SE Nonzero
## min 0.0000942    88 0.02476 0.0009776      73
## 1se 0.0003466    74 0.02535 0.0018449      61
```

Here, we can clearly see that the minimum lambda and the lambda with one standard error from the minimum. With both choice, we can decrease the number of features significantly. Let's check the coefficients. \

```
temp <- coef(cv_fit_lasso, s = cv_fit_lasso$lambda.min)
beta <- Reduce(cbind, temp)
beta <- beta[apply(beta != 0, 1, any),]
colnames(beta) <- names(temp)
beta
```

```
## 166 x 6 sparse Matrix of class "dgCMatrix"
##                          1             2             3             4           5
## (Intercept) 26.040948922 -35.719138575 -3.321198e+01 10.286731901 12.30525698
## V1           .             .             .            -4.517802387  .
```

```
## V2           .            8.150127856 -1.096259e+01  .           10.76215598
## V3           0.840590043  .            .             1.375063744  .
## V23          .           -1.472011800  .             5.879181584 -0.18716547
## V25          .            .           -6.631746e-01  0.587278806  4.42628628
## V28          .            1.849807703 -7.990411e-01  .            .
## V29          .            .           -1.758819e-01 -1.721035704  7.45920269
## V31          .            1.855155383 -5.580509e+00  .           -1.05697556
## V32          .            3.922937234 -4.720647e+00  .            .
## V33          .            3.978656386 -4.895952e+00 -0.578224772  1.23370726
## V34          .            .            2.556201e+00  .            .
## V36          .            1.922810753 -2.729539e+00  .            .
## V37          .            3.352985749  .             .            .
## V38          1.004565803  0.153146457 -1.531465e-01  5.197321171 -7.84556352
## V39          .           -0.921126630  2.344044e-01  .           -0.20442955
## V40         -1.352692142  .            3.476643e-01  1.324167675 -1.74773626
## V56         -5.177437708  .            4.110248e+00  0.275795593 -4.77941194
## V58         14.867782690  .           -7.990280e+00  .           -2.56191492
## V59         11.830821425  .           -1.819392e+01  .            .
## V60          3.894006127 -0.715538898  1.509862e+01  .            .
## V61          0.200756681  .            2.538634e+00  .            .
## V62          .            0.593999142  .            -6.206928013  .
## V63          .           -0.113588006  .            -4.261849920  .
## V64          1.903992066  4.394473519 -8.892931e-01 -1.260774156  .
## V65          .           -1.058670597  1.192451e+00  .            .
## V66          .            .            1.717990e-01  .           -5.62220077
## V73          .            .            9.154158e-01  .           -0.67471974
## V77          .            .            5.069870e-01  .            .
## V78          .           -0.337226254  1.020442e-01  .           -0.12088543
## V79          .           -0.040038552  8.750007e-03  .           -0.43535695
## V80          .            0.165981701  .             .            .
## V81          .            .            .             .            .
## V82          .           -2.011238093  1.621569e-01  .            .
## V83          .            0.490975174 -2.507507e+00  .            .
## V107         .            0.696113521  .            0.505665060 -5.16420672
## V108         .           -0.108553011  2.682187e+00  .            .
## V109         .            .            1.680448e+00  .            .
## V112         .            0.526745344 -2.314645e+00  .            .
## V113         .            .            1.772919e+00  .            .
## V115         .            .           -7.978002e-02  0.002990957  .
## V116         .           -1.417566998  1.610213e+00  .            .
## V117         .           -1.217446158  .            1.006720845  .
## V118         .            0.184431046 -7.923542e-01  1.419156570  .
## V119         .            0.107049025 -1.739405e+00  0.036628215  .
## V120         .            0.656678460 -2.983591e-01  .           -1.36030591
## V121         .            1.992951249  .             .           -0.58571972
## V122         .           -1.348214079  8.931728e+00  1.461865429  .
## V123         .           -2.843964693  .            -2.283653824  0.24276687
## V143         .           -3.273582662  2.540020e-01  .            .
## V144         .            .           -1.531927e+00  .            .
## V145         .            .           -5.582877e-01  .            .
## V148         .           -0.184965663  5.067299e-01  .            .
## V149         .            1.080395524  .             .            .
## V150         .           -4.028247401  4.041447e-01 -2.625200307  0.68209157
## V152         .            3.264166011 -3.132855e+00  .            .
```

```
## V156         .            -0.112626247  1.578998e+00  .            .
## V157         .            -0.108885470  3.561553e+00  1.629676394  .
## V158         .             .            1.012602e+00 -2.586561945  .
## V159         .             0.243072893 -3.457762e-01  .            .
## V160         .            -0.237983199  .             0.651013444 -3.31565411
## V161         .             0.074613324  .             .            .
## V162         .             .            .             .            .
## V163         .            -1.001045467  1.398427e+00  .            .
## V186         .             2.628949415 -8.899615e+00 -0.786615397  .
## V187         .             .           -8.670343e+00  .            .
## V188         .             0.493167378 -2.493628e+00  .            .
## V189         .             0.549399237  .             0.341578568  .
## V191         0.123250771   .           -4.044010e+00  .            .
## V192         .            -3.462804737  .             .            .
## V193         .            -1.970535401  1.774278e+00  .            .
## V196         .             .            1.581444e+00 -0.954962673  .
## V197         .             .            1.351352e+00  0.014544631 -0.27271211
## V198         .             .           -1.976489e+00  1.990327861  .
## V199         .             0.477511992  .             2.939295130  .
## V200         .             0.265671430 -3.235214e-01  1.284017566  .
## V218         .            -3.076260130  .             1.191281774  .
## V223         .             .            .             2.139124361 -2.13412264
## V226         0.126631946  -0.106268439  .            -0.025666901  .
## V231         .             .           -7.729640e+00  .            .
## V237         .             .           -4.793825e-01  .            0.42855895
## V238         .             0.143877182 -7.956690e-01  .            .
## V239         .             .           -1.943503e+00  .           -0.37483386
## V244         .             .           -6.601186e+00  .            .
## V248         .             .           -1.168125e-01  0.831631595  .
## V250         .            -0.381632897  .             .            .
## V252         .             0.416210228  .             .           -0.35506238
## V257         .           -15.586101832  .             0.652961403  .
## V262         .             0.559400095  .             .            .
## V263         .             2.160451633  .             .            .
## V264         .             0.072424407 -6.220709e-01  .           -2.01813924
## V265         .             0.412065640 -1.462656e+00  .            .
## V279         .             .            .             .            0.70651292
## V280         .            -0.013561050  .             .            .
## V291         .             0.424887458 -1.268052e-05  .            .
## V292         .             .            6.812459e-04  .            .
## V293         .             .           -7.364368e-01  .            .
## V294         0.005563297  -0.005563297  4.997701e-01  0.005563297 -5.00631733
## V295         .             .            .             .            .
## V296         .            -0.745681165  .             .            .
## V298         .             0.027867247 -1.149294e+00 -2.010889202  .
## V299         .            -0.242046881  4.444445e-01  .           -3.91776268
## V301         .            -0.115533835  .             .           -2.13048758
## V304         .             .            .             .            7.17166147
## V310         .             .            .            -1.344105234  1.71362678
## V319         .             .            .             2.222879897  .
## V324         .             .           -2.864570e-01  .            .
## V338         .             .            .             .            .
## V357         .             .            .             1.524828987  .
## V358         .            -2.753293284  .             0.084224636  .
```

```
## V359          .            -4.556515934  .             0.489391933  0.12698813
## V370          .             .            6.345107e-01  .           -1.22022902
## V371          .             .            2.140905e-01  .            .
## V372          .             .            1.194696e-01  0.811926294  .
## V377          .             .           -3.202747e-01  .            0.26585205
## V379          .             .            .             1.438535782 -1.02366064
## V381          .             .           -5.999811e-01  .           -2.37836574
## V384          .             .            .             .            8.18681249
## V388          .             .            .             .            4.10694337
## V389          .             .            .            -3.287467541  .
## V403          .             .            .             .            .
## V414          .             .            .             .            .
## V417          .             .            .             .            0.30193104
## V436          .            -4.524876698  .             0.035283074  0.02994111
## V437          .             0.233322358 -8.304409e+00  .            .
## V438          .             .            .             0.860698210  .
## V449          .             0.578265071  .             0.040865197  .
## V450          .             .            5.388021e-01  2.146910986  .
## V451          .            -0.267118364  2.904225e-01  2.867646813  .
## V452          .             0.978514945 -7.953332e-01  .           -1.57748453
## V453          .            -0.531022463  1.062375e-01  .            .
## V454          .             0.536455878 -4.474314e-02  .            .
## V456          0.144781862  -0.070483934  .             .            .
## V458          0.005778846   .           -3.652063e-01  .           -0.16618182
## V460          .             0.269212220  .             .           -2.06217273
## V461          .             .           -7.117538e+00  .            3.46918661
## V462          .             .            .             5.712016980  .
## V463          .             .            .             .            .
## V465          .             .            .             .            .
## V468          .             .            .             0.301934906  .
## V475          .             .            .             4.867625467  .
## V482          .             .            .             .            .
## V489          .             .            .             0.428432770  .
## V490          .             .            .            -0.227563993  5.98120209
## V491          .             .            .             .            .
## V493          .             .            .             .           -0.10685605
## V507          .            -0.505939259  .             .            0.40203432
## V512          .            -0.135323904  .             .            .
## V513          .             .            .             0.758832965  .
## V514          .             1.104285756  .            -1.916839892 -0.53890651
## V520          .             .            .             .            .
## V525         -0.387799731   .            9.680247e-02  .            .
## V526          .             .           -4.948896e-01  .            .
## V528          .            -1.374852021  .             1.002009428  1.21934686
## V533          .             1.407051822 -2.228785e+00  .            2.35739974
## V538          .             .            .             2.191828019 -1.17527073
## V541          .             0.719947498  .             0.454161601 -0.13131947
## V546          .           -14.906535582  .             0.589270348  .
## V551          .            -0.995627974  .             .            .
## V552          .            -0.349895455  .             .            .
## V554          .             0.309217669  .             .           -1.55042699
## V555          .             .           -2.814721e-01  0.097303066 -0.04362877
## V556          .            -0.003262458  .             0.132609892  .
## V557          .            -1.056440972  9.509630e-04  .            0.18003954
```

```
## V558          .            -0.003368490  .             .            0.10440804
## V561          .            -2.324010257  4.117300e+00  .            .
##                         6
## (Intercept)  20.298184818
## V1           -5.498703131
## V2           -1.974330404
## V3            .
## V23           4.416722999
## V25           .
## V28           .
## V29           .
## V31           0.363465028
## V32           .
## V33           .
## V34          -0.011218216
## V36           .
## V37           1.976325489
## V38          -1.139690274
## V39           .
## V40           .
## V56           .
## V58           6.483423345
## V59           .
## V60           .
## V61          -1.149152639
## V62          -0.337220608
## V63           1.325553025
## V64           .
## V65           .
## V66           .
## V73           4.627507414
## V77           .
## V78           .
## V79           0.004677379
## V80           .
## V81          -0.313619797
## V82          -0.377752758
## V83           .
## V107         -1.017139518
## V108          .
## V109          .
## V112          .
## V113         -0.676982485
## V115         -3.368540154
## V116         -3.264221558
## V117         -0.601234969
## V118         -2.303262716
## V119         -1.946447579
## V120          .
## V121          0.080470032
## V122          .
## V123          .
## V143         -0.701589756
## V144          .
```

```
## V145          .
## V148          .
## V149        -0.044952768
## V150          .
## V152          0.131852711
## V156        -1.108729044
## V157        -1.195396680
## V158          .
## V159          .
## V160          0.931931633
## V161          0.157543886
## V162        -0.066942227
## V163        -0.019137193
## V186          2.587644713
## V187          7.160425800
## V188          3.066786116
## V189          .
## V191          2.556246644
## V192          0.321594034
## V193          0.073736280
## V196          .
## V197          .
## V198        -3.107242254
## V199        -3.410155579
## V200        -0.923211078
## V218          1.380924115
## V223          .
## V226          .
## V231          0.915166464
## V237          1.757070167
## V238        -0.602351731
## V239          0.006614510
## V244          0.394594212
## V248          .
## V250          0.838857310
## V252          .
## V257          0.826560271
## V262          .
## V263        -0.520933081
## V264          0.217438052
## V265          .
## V279          .
## V280          .
## V291          .
## V292        -0.897353536
## V293          .
## V294        -1.919062967
## V295        -2.452148499
## V296        -2.223105740
## V298          0.388193012
## V299          .
## V301          .
## V304          4.332144890
## V310          .
```

```
## V319          .
## V324          0.730390160
## V338          1.190027592
## V357          .
## V358          0.522409517
## V359          .
## V370         -1.213149815
## V371          .
## V372         -1.057618724
## V377          .
## V379          .
## V381          .
## V384          .
## V388          .
## V389          3.353012870
## V403         -0.114862344
## V414        -10.718004748
## V417          .
## V436          .
## V437          .
## V438          1.146023473
## V449          .
## V450         -0.994955837
## V451         -0.580767536
## V452          .
## V453          .
## V454          .
## V456          .
## V458          2.077875570
## V460          2.092128791
## V461          0.388631668
## V462          2.922906688
## V463          0.282165745
## V465          0.386313637
## V468          .
## V475          .
## V482          0.713123140
## V489          5.272616008
## V490          .
## V491          5.228011758
## V493          1.797275242
## V507          .
## V512          0.428908135
## V513          .
## V514          .
## V520          0.996224064
## V525          .
## V526          0.410628598
## V528         -1.996964927
## V533          .
## V538          .
## V541          .
## V546          .
## V551          .
```

```
## V552          .
## V554          .
## V555          .
## V556          .
## V557         -0.064128072
## V558          .
## V561          .
```

```
set.seed(123)

# Make predictions on the test set
pred_lasso = predict(cv_fit_lasso, newx= data.matrix(X_test[,-ncol(X_test)]), type='class')

# Accuracy
mean(pred_lasso == data.matrix(y_test))
```

```
## [1] 0.9391682
```

The accuracy now is almost 0.94 which is higher than that without LASSO. The features: V38 tBodyAcc-correlation()-X,Y, V294 fBodyAcc-meanFreq()-X appear to be the most relevant features for the classifying all the labels while labels such as V81 tBodyAccJerk-mean()-X, V551 fBodyBodyGyroJerkMag-maxInds, V552 fBodyBodyGyroJerkMag-meanFreq() appear to be less relevant for the prediction.

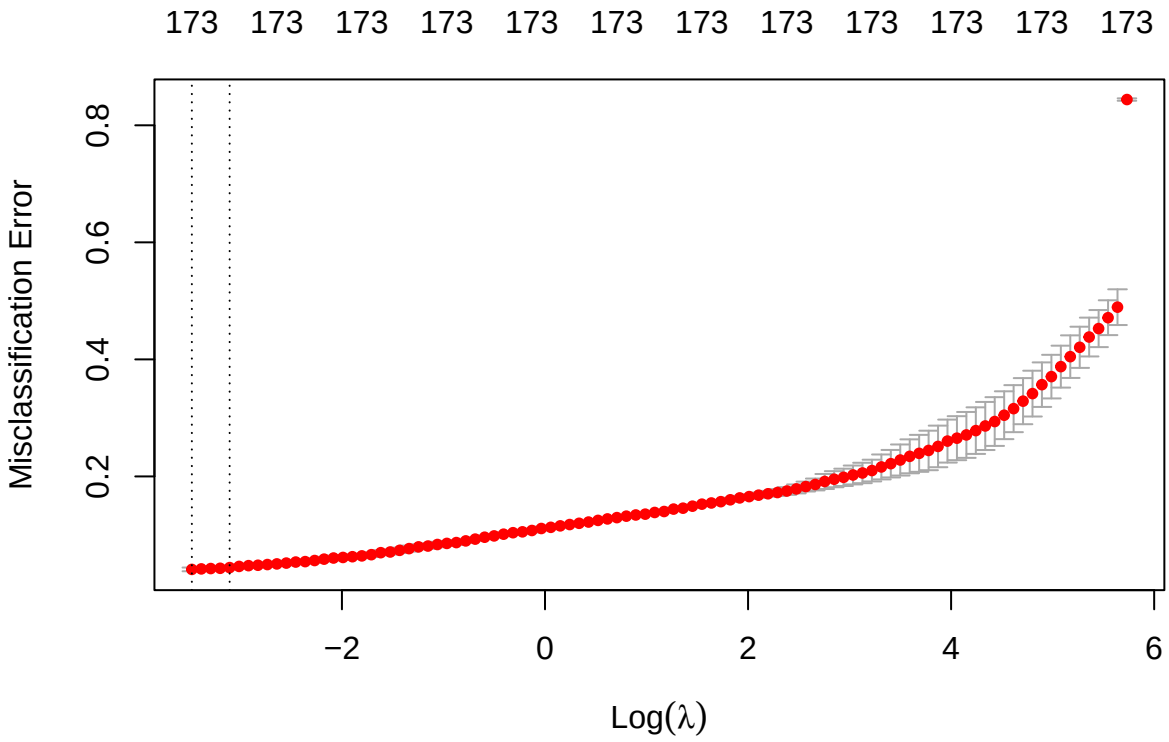
```
table(data.matrix(y_test),pred_lasso)
```

```
##    pred_lasso
##       1   2   3   4   5   6
##   1 537   0   0   0   0   0
##   2   0 464  71   0   0   2
##   3   0  37 500   0   0   0
##   4   0   0   0 521  15   1
##   5   0   0   2  11 499  25
##   6   0   0   2  25   5 505
```

Ridge Regression

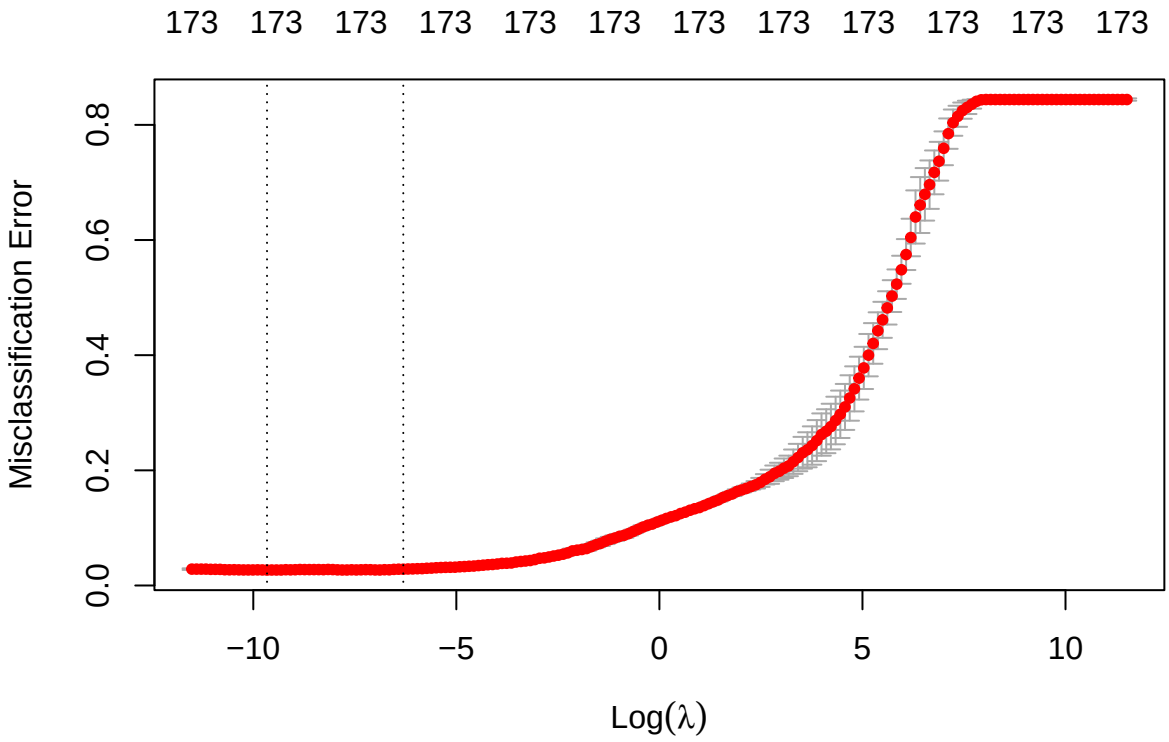
Ridge Regression also penalizes the coefficient with a L2 norm. Unlike LASSO that can force some coefficients to be zero, the Ridge Regression forces the coefficients to be close to zero.

```
set.seed(123)

cv_fit_ridge <- cv.glmnet(data.matrix(X_train[,-ncol(X_train)]), data.matrix(y_train), family= 'multinor
plot(cv_fit_ridge)
```



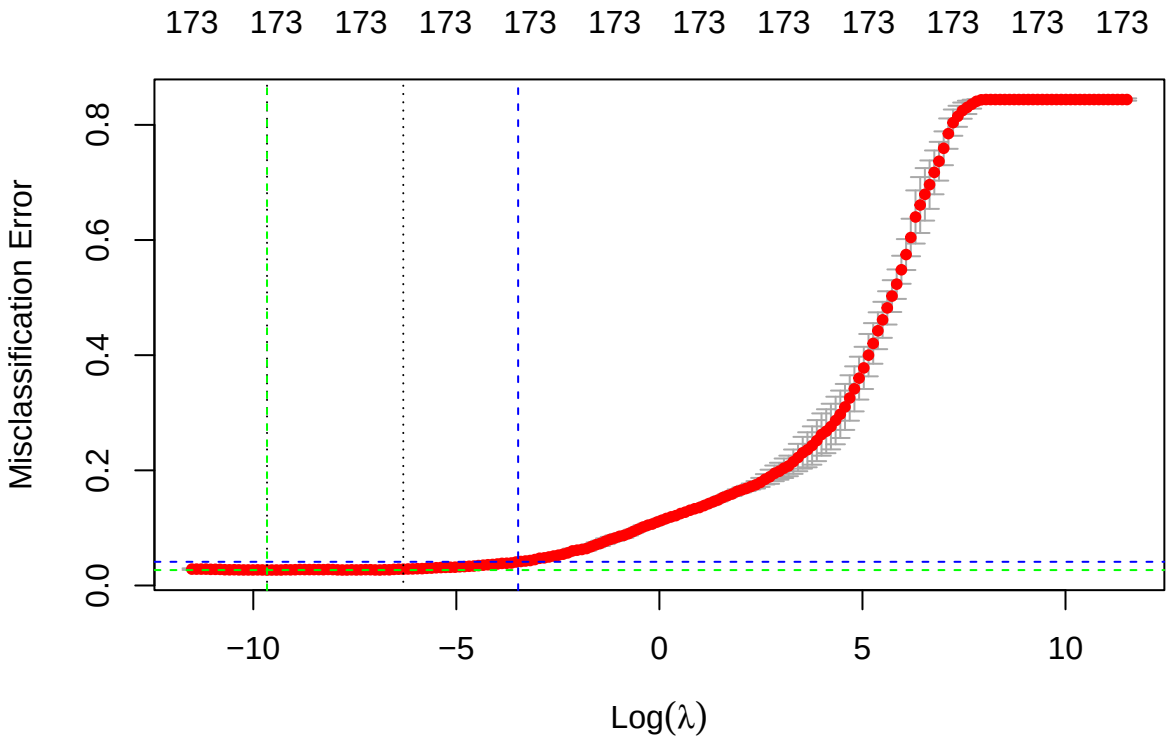
From the plot, we can see that the minimum is at the edge of the plot, and the trend of decreasing still seems quite strong, thus we will apply a larger search grid than the default one so that the error can decrease further. \

```
set.seed(123)
grid = 10^seq(5,-5,length=200) ##get lambda sequence
cv_fit_ridge_grid <- cv.glmnet(data.matrix(X_train[,-ncol(X_train)]), data.matrix(y_train), family= 'mu
plot(cv_fit_ridge_grid)
```



Now the minimum is quite far from the boundary and let's compare the lambdas we obtained with these two grids.

```
set.seed(123)
plot(cv_fit_ridge_grid)

lambda_ridge_default <- cv_fit_ridge$lambda.min
i <- which(cv_fit_ridge$lambda == cv_fit_ridge$lambda.min)
mse_min_default <- cv_fit_ridge$cvm[i]
j <- which(cv_fit_ridge_grid$lambda == cv_fit_ridge_grid$lambda.min)
mse_min_grid <- cv_fit_ridge_grid$cvm[j]
lambda_ridge_grid <- cv_fit_ridge_grid$lambda.min
abline(h=mse_min_default, v=log(lambda_ridge_default),col= 'blue', lty=2)  # selected by default lambda
abline(h=mse_min_grid, v=log(lambda_ridge_grid),col= 'green', lty=2)  # selected by defined grid of lam
legend(17, 185, legend=c("default", "search_grid"),
       col=c("blue", "green"), lty=1:2, cex=0.8)
```



The point indicated by the blue line is the minimum point selected by default grid while the point indicated by the green line is the minimum point selected by defined grid, indeed the error keeps decreasing for awhile which means we get a better lambda value.

```
cat('The selected lambda corresponding to the minimum error is: \n', cv_fit_ridge_grid$lambda.min)
```

```
## The selected lambda corresponding to the minimum error is:
##  6.36825e-05
```

```
cat('\n\n')
```

```
cat('The selected lambda corresponding to the 1se is: \n', cv_fit_ridge_grid$lambda.1se)
```

```
## The selected lambda corresponding to the 1se is:
##  0.001825183
```

```
temp <- coef(cv_fit_ridge_grid, s = cv_fit_ridge_grid$lambda.min)
beta <- Reduce(cbind, temp)
beta <- beta[apply(beta != 0, 1, any),]
colnames(beta) <- names(temp)
beta
```

```
## 174 x 6 sparse Matrix of class "dgCMatrix"
##                       1            2           3          4          5
```

```
## (Intercept)  6.27517992 -55.339065630 -14.220866423 23.80246883 22.54574582
## V1          -1.67797991   0.137638081   7.174951910 -3.48332864  2.28076788
## V2           4.78633989  12.637510834 -11.173607383 -7.60756643  7.37400049
## V3          -1.21662127   1.868553890   0.113025586  1.86452846  0.31162128
## V23         -0.41074210  -1.921546967  -0.592339090  2.51722642 -1.79807912
## V25          0.06192559  -1.408219675  -1.952404118  1.09568392  2.75358431
## V28          0.30042096   2.965586648  -2.540097564  0.13219169 -1.06133769
## V29          0.10920984   0.051747751  -0.903064983 -2.95350076  4.80777181
## V31         -0.63121021   2.682411756  -4.884306108  2.58684934 -1.00620449
## V32          0.20349899   4.492675516  -4.247942440  0.21562327 -0.02692650
## V33          0.56518101   4.732627585  -4.647783479 -1.91444296  2.10179965
## V34          0.62938010  -0.475329526   2.034544402  0.37553355 -0.87098293
## V36          0.49648905   3.200846437  -3.683374632 -0.56158520 -0.09454967
## V37          0.96353189   2.253653652  -1.558080474 -1.60417221 -1.79491236
## V38          1.39527827   0.393388471  -0.005753386  3.41706375 -4.45240929
## V39          0.81078052  -0.648344936   0.592779553  0.24358005 -0.90911710
## V40         -0.98575322   0.205222663   0.792062621  1.38248939 -1.57135679
## V56         -2.77149455   0.059474427   2.515864556  1.05633268 -2.91212282
## V58          8.21689378  -2.219184519  -4.575231433 -0.89616312 -2.03799472
## V59          8.54376983   1.798829220 -10.133764264  0.50338351  0.20353575
## V60          2.46719598  -7.270870990   9.944670771  1.50691566 -1.03298166
## V61          0.11902738   0.417932648   3.564996061 -0.41553417 -0.79707152
## V62          0.48280754   3.344984233   1.517730047 -5.77851903  1.17817877
## V63          0.41186303   0.260523723  -0.120163582 -3.45259639  0.64499161
## V64          1.77747438   4.134060786  -1.495980136 -1.69771241 -2.06972859
## V65         -0.66806764  -1.007639145   1.180845090  0.03950967  0.31511900
## V66         -0.16368789   0.928475047   1.515928475  2.27438538 -4.66276460
## V73         -0.47916007   0.083603373   1.073530509 -1.30088346 -2.32890615
## V77         -0.68299991   0.200868030   0.818355229  0.03650437 -0.74887673
## V78          0.12181366  -0.332334288   0.131825727  0.11860532 -0.37144466
## V79          0.10679523  -0.018495183   0.099913100  0.13669758 -0.48840044
## V80         -0.07730051   0.195449069  -0.021654231 -0.06695212  0.17845391
## V81          0.07512618  -1.443599727   0.857707046  1.30060767  0.12251598
## V82          1.20130648  -2.681837072   0.664897542  0.18670503  0.53509255
## V83          2.27750486   1.812331991  -2.548332326 -0.77778140 -0.25874481
## V107        -0.28260403   2.980853145   0.098200948  2.53698587 -4.73428975
## V108        -0.42651756  -0.608594721   2.954253267  0.30247838 -1.37877720
## V109        -0.18639350  -0.210582612   2.553616229  0.47450792 -1.79360777
## V112         0.26164413   1.493315199  -1.798882329  0.52391807 -0.35825643
## V113        -1.11305098  -0.337511499   1.395720514  0.51743271  0.73338572
## V115        -0.29926030   0.529559562  -1.116716771  1.64488781  2.15904277
## V116         0.14072026  -2.388417843   1.802124666  0.69992647  2.03137349
## V117         0.44061641  -1.482547816   0.421045241  1.10468343  0.72832317
## V118         0.03316473   0.628685713  -0.409715080  3.18619898 -1.31127393
## V119         1.11991010   0.828874019  -1.200504356  1.13446096  0.23057948
## V120         0.23603904   0.874425967  -0.360281571  0.87409923 -2.33795684
## V121        -0.12637541   4.753100480  -1.689724834 -1.32256497 -1.89744778
## V122        -2.14183655  -4.236035813   7.321660951  1.82258254 -1.46922727
## V123         1.95520747  -2.915748705   3.008369119 -3.32832993  0.63565733
## V143        -0.18417679  -2.407033537   1.650797132  0.95766615  1.27285148
## V144        -0.18637997   0.231035629  -1.472874078 -0.71737876  1.24074110
## V145        -0.56773815  -0.254518880  -1.166920755  1.50383686  0.25243183
## V148        -0.62493532  -0.317602058   1.750665532  0.89386933  0.14919457
## V149         0.99940073   1.566586270   0.295801990  0.13055440 -0.62229564
```

```
## V150        0.21443225 -2.168057090  1.796616908 -2.97073294  2.36809312
## V152       -0.35380128  3.634510974 -3.795432150  0.44613417 -1.26181761
## V154        0.71842143  0.846782519  0.518743515 -1.76999067 -0.75943873
## V156        0.45192210 -0.450025583  0.961321874  1.32920146 -0.12447586
## V157        0.16844227 -2.068127972  2.568387881  2.00402873 -0.03896309
## V158       -0.23390432  0.107390470  1.126490784 -2.03050791  0.18520089
## V159        0.24965969  0.456695283 -0.202292842 -0.16942545 -0.22441899
## V160        0.60600946 -0.507812312 -0.157424844  1.12059892 -2.64226087
## V161       -0.53366774  2.377752976 -0.910170322 -1.85573574  0.30367468
## V162        1.02659405 -0.182286207  0.387539295  0.32586417 -0.26229746
## V163       -0.78438348 -3.757884371  3.131377882  0.91698323  0.33099669
## V186        1.76228721  5.353931233 -5.757373962 -2.13863612 -1.26572228
## V187       -0.12950341  2.233276346 -5.481662834 -0.31604047 -1.31341066
## V188        0.17828961  2.012096979 -2.440124056 -0.44849224 -1.44726682
## V189       -0.22006104  0.790269992 -0.684374405  0.61734519 -0.38667164
## V191        3.37781794  1.198780668 -3.672560878 -2.12214093 -2.76315874
## V192        1.32405928 -3.073094676  1.338001389 -0.80384814 -0.46150787
## V193        0.68740446 -2.357811684  1.970830773 -0.56693552 -0.93326010
## V196       -0.01214671 -1.638131114  1.338486524 -0.54184404  0.74903390
## V197       -0.02205333 -0.686803214  1.296571338  0.42527139 -0.67865291
## V198        0.52788632  0.593509379 -1.594677533  2.33073864  1.18477347
## V199       -0.33259284  0.519254127  0.011939285  3.20909859  0.16212853
## V200       -0.87641910  0.949899308  0.250186200  1.96377672 -0.31202864
## V218       -0.09572330 -5.815865529 -0.998260850  2.63435250  1.54133632
## V223        0.21080586  0.293353748  0.370439460  2.65262948 -2.89959428
## V226        0.72075278 -0.524644204 -0.332306184 -0.03093195 -0.03990791
## V231       -0.43229953  2.082714843 -4.371851850  0.65782896  0.38359995
## V237       -0.63019250 -0.165593546 -1.096868935 -0.78648247  0.95430918
## V238        0.52582267  0.636393666 -0.655495922  0.36574916 -0.36759694
## V239        0.43622610  0.620368235 -1.649062884  0.25406829 -0.59894443
## V244        0.50160589  1.822786113 -4.563710137  0.20072340  0.94610744
## V248        0.14075122 -0.715386342 -0.788629750  1.19478930 -0.37440673
## V250        0.70944380 -0.189736930  0.207448265 -0.95715889 -0.79130936
## V252       -0.58148055  0.499466978  0.146232222  0.56310199 -0.87025083
## V257       -0.44545817 -7.810439934  3.992478629  2.28806491  0.13249847
## V262       -0.61409359  3.322627516 -3.546261939 -0.64272013  0.97487055
## V263        0.08803154  5.690204936 -4.393928418 -0.51714737  0.43664302
## V264       -0.18632753  2.951450375 -2.843622798  0.88825860 -1.48297980
## V265        0.38424520  3.190634335 -3.056893694  0.04672119 -0.07599588
## V278       -0.68218647  0.887067279 -1.927875929  0.77871638  0.76566752
## V279       -0.30181947 -1.094895730  0.375603172  0.20581750  1.22150572
## V280        0.05041487 -0.511735475 -0.201570422  0.94747404 -0.50593134
## V291        0.65339231  0.710241034  0.407102100 -1.40556033 -0.52824748
## V292        0.51119058 -0.064461115 -0.005621883  0.42845703  1.23582196
## V293        0.69311951 -0.202951208 -0.804467622  0.01529638  0.38082033
## V294        0.87308165  0.959425235  1.680799410  1.49554418 -3.67451322
## V295        0.42126509  0.551600478  0.749508576  0.23976775  0.93551241
## V296        0.43824335 -0.345997726  0.323342300  0.80628773  0.64807789
## V298        0.51519218  0.321767025 -0.878809419 -1.67396103  0.53898874
## V299        0.77799201  0.229675691  0.959423846  0.28005326 -2.77541906
## V301       -0.20478452  0.078845877  0.276411209  0.98883504 -1.46574407
## V304       -0.45784816 -1.046865358 -1.783016908 -2.80959020  4.62118533
## V310       -0.55447568  0.958515724 -0.610185427 -2.51780458  2.01911379
## V319       -0.35306544 -0.251190619 -1.600008142  1.57111963  0.68633118
```

```
## V324       -0.27720976   7.274987992  -7.095018840  0.23245559 -1.19831731
## V338       -0.11891431   0.239654044  -1.001708032 -0.71807969  0.32990461
## V357       -0.69092315  -4.476170468   2.402583575  1.93180805  0.43561285
## V358       -0.46806901  -3.567713976   1.368135937  0.86544331  0.03062750
## V359       -0.46497517  -5.064214472   1.897375621  1.50335622  1.51853257
## V370       -0.05230632   0.354837754   1.036381703  0.73796749 -1.33826414
## V371        0.22395425  -0.073043095   0.079063835 -0.37560350  0.34825938
## V372       -0.14832415  -0.013948980   0.145416967  1.10716257 -0.10688808
## V377       -0.56512982  -0.677953821  -1.200763536  0.23671786  1.94534932
## V379       -0.58998864  -0.132453899  -0.197559210  1.21702832 -1.14024090
## V381        0.12059696   0.172532156  -0.320935744  1.54532287 -3.22733317
## V384       -0.35294161  -0.385026668  -2.022313529 -1.33559987  5.40649197
## V385       -0.37856642  -0.875256345  -0.120386641 -0.74598295  1.21152091
## V388       -0.61819773  -1.292767250  -2.308351333 -1.86148417  6.53666922
## V389       -0.49165022   0.074111157   0.642834853 -3.40450558 -0.95842864
## V400       -0.59356265  -2.345781028   0.062019816  2.48063216 -0.09791452
## V403       -0.34501379   0.300372781  -0.318760565  0.44030189  0.75567455
## V414       -0.56714902  -2.361884947   0.534456965  2.77207412  6.69690463
## V417       -0.29872607  -0.405181652  -0.129394769  0.33682018  1.12814173
## V436       -0.46793836  -4.549753305   2.138083997  1.42740959  1.75460248
## V437        0.23182865   3.416316210  -4.883903543  0.81189213  0.14460643
## V438       -0.57872813  -2.008711457  -0.430269805  0.97281246 -0.20831798
## V449        0.89501588   0.619969585   0.004668023  0.02925368 -0.84972278
## V450       -0.50246691  -0.313275349   0.400285280  2.18336752 -0.11494977
## V451        0.36133613  -0.126281710   0.445683068  2.42193968 -1.09924932
## V452        1.00257610   0.664286871  -1.199547488  0.18617735 -1.66365448
## V453        0.78288691  -0.697475833   0.172276278  0.22397750  0.45214951
## V454        1.18586421  -0.004529284  -0.908102500  0.13170937  0.75374057
## V456        1.14361226  -0.546618619  -0.499376805  0.18793256 -0.16904218
## V458        0.74300243  -0.233237763  -0.719685049  0.13650848 -1.57123954
## V460        0.08964277   0.250269846   0.045112577  0.01908251 -2.33700659
## V461       -0.83494359   5.467252692  -8.301037579 -1.01578825  3.53571322
## V462       -0.36372386  -3.426179023   0.804066781  4.21374746 -3.31067071
## V463       -0.35785008  -2.758249861   0.983569493  0.62631572 -0.85898600
## V465       -0.45338787  -7.384881947   6.068923681  0.15032327 -1.54171380
## V468       -0.61709269   0.299688949  -2.386817783  1.39319916  0.86149275
## V475       -0.33822272  -2.209832967  -0.513233763  3.41887630 -1.17649155
## V476       -0.21545179  -1.926964293   0.448584850 -0.76755596  1.92800657
## V477       -0.21787944  -0.587602237  -0.002830080  2.11866666  1.21988249
## V479       -0.70165910  -8.890668740   6.726710672  4.72920862  3.28499258
## V482       -0.31315545   3.254587988  -2.850571041 -1.76902619 -1.45992802
## V489       -0.55705213  -2.100676913  -0.148157921  0.50431034 -1.59708110
## V490       -0.38361287  -0.791229883  -0.806524397 -3.04162593  5.24711324
## V491       -0.43854484   2.014378825  -2.518998887 -1.43211299 -1.07061775
## V493       -0.81241875  -2.994199837   0.950919674  0.23377626 -0.91734209
## V496       -0.62416289  -3.143871845   2.763496591  0.67511461  0.35006207
## V507        0.90181289  -2.352099853   0.411082189  0.35994251  1.19865256
## V512        0.08688283  -0.992512918  -0.881667125  0.53951084 -0.01029619
## V513       -0.36813734  -0.088856769  -0.161670487  1.84974737 -1.68905236
## V514        1.14041014   1.500545288   0.199128127 -2.18039334 -0.97077609
## V520       -0.39318642  -1.733995903  -0.731945392  0.56469638  0.67252270
## V525       -0.70696000  -0.261409644  -0.112980386  1.13408107 -0.59293443
## V526        0.19949650   0.257156306  -0.628841745 -1.00580739 -0.26474134
## V528       -0.64886574  -1.215470397   0.134614730  1.46596143  1.96652117
```

```
## V533         0.31604504   3.381518356  -3.430038872 -1.69471408  2.36315815
## V538        -0.53204083  -0.541176361  -0.597485676  2.50667854 -1.68968472
## V541         0.37650283   0.719327932   0.004099925  0.46966577 -1.12975461
## V546        -0.12409356  -8.685168322   5.281787388  2.02152008  0.63093974
## V551        -1.48451339  -0.945559953   0.016364758  0.64186899  0.31801577
## V552        -1.31736501  -0.998506909  -0.243563615  0.41785630  1.54754917
## V554        -0.44886723   0.874027769   0.605853560  0.60078757 -1.89332962
## V555         0.40395160   0.123740643  -0.271166716 -0.02077430 -0.18337348
## V556         0.11757030  -0.033776873   0.007067000  0.55967118 -0.27388032
## V557        -0.34471073  -0.634692184   0.544945266  0.17989555  0.40590518
## V558         0.19845895  -0.135376765  -0.231527204 -0.32197063  0.26300997
## V561        -3.32320286  -1.737426931   4.489345797  1.51903834 -1.12349128
##                       6
## (Intercept) 16.93653748
## V1          -4.43204932
## V2          -6.01667739
## V3          -2.94110794
## V23          2.20548086
## V25         -0.55057002
## V28          0.20323595
## V29         -1.11216366
## V31          1.25245971
## V32         -0.63692884
## V33         -0.83738181
## V34         -1.69314560
## V36          0.64217402
## V37          1.73997950
## V38         -0.74756781
## V39         -0.08967808
## V40          0.17733534
## V56          2.05194571
## V58          1.51168001
## V59         -0.91575404
## V60         -5.61492976
## V61         -2.88935040
## V62         -0.74518157
## V63          2.25538161
## V64         -0.64811403
## V65          0.14023303
## V66          0.10766359
## V73          2.95181580
## V77          0.37614902
## V78          0.33153424
## V79          0.16348972
## V80         -0.20799611
## V81         -0.91235715
## V82          0.09383547
## V83         -0.50497832
## V107        -0.59914618
## V108        -0.84284216
## V109        -0.83754027
## V112        -0.12173865
## V113        -1.19597647
## V115        -2.91751308
```

```
## V116       -2.28572703
## V117       -1.21212044
## V118       -2.12706041
## V119       -2.11332021
## V120        0.71367417
## V121        0.28301252
## V122       -1.29714386
## V123        0.64484472
## V143       -1.29010443
## V144        0.90485608
## V145        0.23290909
## V148       -1.85119206
## V149       -2.37004775
## V150        0.75964776
## V152        1.33040590
## V154        0.44548193
## V156       -2.16794399
## V157       -2.63376783
## V158        0.84533009
## V159       -0.11021768
## V160        1.58088966
## V161        0.61814614
## V162       -1.29541385
## V163        0.16291004
## V186        2.04551392
## V187        5.00734103
## V188        2.14549654
## V189       -0.11650809
## V191        3.98126194
## V192        1.67639001
## V193        1.19977207
## V196        0.10460144
## V197       -0.33433327
## V198       -3.04223027
## V199       -3.56982770
## V200       -1.97541448
## V218        2.73416086
## V223       -0.62763428
## V226        0.20703746
## V231        1.68000764
## V237        1.72482827
## V238       -0.50487264
## V239        0.93734470
## V244        1.09248731
## V248        0.54288230
## V250        1.02131312
## V252        0.24293019
## V257        1.84285609
## V262        0.50557759
## V263       -1.30380371
## V264        0.67322115
## V265       -0.48871115
## V278        0.17861121
## V279       -0.40621120
```

```
## V280         0.22134833
## V291         0.16307235
## V292        -2.10538658
## V293        -0.08181739
## V294        -1.33433725
## V295        -2.89765430
## V296        -1.86995354
## V298         1.17682250
## V299         0.52827425
## V301         0.32643647
## V304         1.47613530
## V310         0.70483617
## V319        -0.05318661
## V324         1.06310233
## V338         1.26914337
## V357         0.39708915
## V358         1.77157624
## V359         0.60992524
## V370        -0.73861650
## V371        -0.20263087
## V372        -0.98341833
## V377         0.26178000
## V379         0.84321433
## V381         1.70981693
## V384        -1.31061029
## V385         0.90867144
## V388        -0.45586873
## V389         4.13763842
## V400         0.49460622
## V403        -0.83257486
## V414        -7.07440174
## V417        -0.63165942
## V436        -0.30240440
## V437         0.27926012
## V438         2.25321491
## V449        -0.69918439
## V450        -1.65296077
## V451        -2.00342785
## V452         1.01016165
## V453        -0.93381436
## V454        -1.15868237
## V456        -0.11650722
## V458         1.64465144
## V460         1.93289889
## V461         1.14880350
## V462         2.08275935
## V463         2.36520073
## V465         3.16073666
## V468         0.44952961
## V475         0.81890470
## V476         0.53338062
## V477        -2.53023739
## V479        -5.14858404
## V482         3.13809271
```

```
## V489         3.89865772
## V490        -0.22412016
## V491         3.44589564
## V493         3.53926474
## V496        -0.02063854
## V507        -0.51939030
## V512         1.25808256
## V513         0.45796958
## V514         0.31108587
## V520         1.62190864
## V525         0.54020339
## V526         1.44273767
## V528        -1.70276119
## V533        -0.93596860
## V538         0.85370905
## V541        -0.43984185
## V546         0.87501467
## V551         1.45382382
## V552         0.59403007
## V554         0.26152796
## V555        -0.05237775
## V556        -0.37665128
## V557        -0.15134309
## V558         0.22740568
## V561         0.17573693
```

```
set.seed(123)

# Make predictions on the test set
pred_ridge = predict(cv_fit_ridge_grid, newx= data.matrix(X_test[,-ncol(X_test)]), type='class')

# Accuracy
mean(pred_ridge == data.matrix(y_test))
```

```
## [1] 0.937306
```

We can see that now the accuracy is over 0.94 which is slightly better than LASSO and the most relevant features are those with coefficients appear to be high in magnitude in terms of the absolute values, for example: V2 tBodyAcc-mean()-Y and V191 tBodyGyroJerk-arCoeff()-Y,2. Furthermore, we can get the confusion matrix: \

```
table(data.matrix(y_test),pred_ridge)
```

```
##    pred_ridge
##       1   2   3   4   5   6
##   1 537   0   0   0   0   0
##   2   1 462  71   0   0   3
##   3   0  41 496   0   0   0
##   4   0   0   0 520  10   7
##   5   0   0   0   8 511  18
##   6   0   1   2  25  15 494
```

Elastic-net

LASSO uses L1 penalty to force sparsity and Ridge Regression uses L2 penalty to force the coefficients to be close to zero, the Elastic-net takes into both types of penalty which hopes to obtain an intermediate solution. \

Before fitting the model with Elastic-net, first we have to fix the amount of alpha we want to use, hence, we will use a search grid between 0 and 10 with a total of 10 different points to find the best alpha. \

```
set.seed(123)
alpha_grid = seq(0, 1,length=10)
cv_alpha <- lapply(alpha_grid, function(a){
  cv.glmnet(data.matrix(X_train[,-ncol(X_train)]), data.matrix(y_train), nfold=3,  alpha=a, family= 'mu
})
```

```
min_loss <- c()
for (i in 1:10) {
  min_loss <- append(min_loss, i)
}
alpha <- min_loss[which.min(min_loss)]
alpha
```

```
## [1] 1
```

The best alpha we found is 1 which means the LASSO regression, thus we don't need to proceed the following steps as if we use the exact same parameter as with LASSO, the Elastic-net and LASSO will give us exactly the same result. \

Conclusion

Within this project, we tried two different types of representations of dataset and with each representation we used four different methods check the performance. Overall, we can see that the representation with the removal of highly co-related features appear to be better than PCA except the case that we use LDA. With the former representation, all of our models performs well and almost all of them can reach an accuracy of around 0.90, and we can see that there is a sort of separability between different activity blocks, namely: WALKING and NON-WALKING blocks. The prediction errors primarily stay within their own block which indicates that it is easy to separate them. Sometimes the models experience some minor difficulties labeling correctly the within the blocks, such as mis-classifying 'SITTING' as 'STANDING' and 'WALKING_DOWNSTAIRS' as 'WALKING'. With PCA, only LDA is able to outperform the models with the other representation and in fact PCA with LDA achieved the highest accuracy among all the possible combinations of the representations and models. Even though it still suffers from the difficulty of correctly classifying within the NON-WALKING block, it can actually classify the WALKING block much better, hence, if our objective focus more on the correctness of the NON-WALKING block, then transforming the dataset with PCA first and then apply LDA appears to be the best option. On the other hand, if we focus more on the separability between different activity blocks, then removing the highly correlated features before fitting the models is highly recommended as all after the feature removal all the models can separate the different blocks. \

As for the second part of the project, we applied different regularization approaches and hope to find a way to guarantee a better generalization of our models. Since all of the techniques used are model dependent thus we fixed the logistic regression. Furthermore, three different approaches have been tested, and all of them actually improve in accuracy which might indicate that without regularization, the model actually overfits the data. All three techniques performed well, LASSO and Elastic-net turned out to be equivalent for this

specific problem as after the cross-validation, the best alpha value for the Elastic-net appears to be equals to 1 which indicates a LASSO regularization. With LASSO, not only the accuracy of improved but also the number of parameters needed has dropped significantly. And finally, the ridge also provide really good result but slightly worse than LASSO and Elastic-net, considering the feature selection ability provided by LASSO, Ridge Regression is not really the best option from the computation and storage point of view as it cannot shrink the parameters to zero. \

```
rm(list = ls())
```