Human Activity Recognition

Kiran Joshi

July 20, 2020

Overview

Human Activity Recognition abbreviated as HAR is the main driver for a multimillion dollar fitness industry, as users would like to record and monitor the healthy lifestyles they lead. The raw data for this activity is obtained by accelerometers built into smart devices which collect the spacial coordinates if the device when an activity is performed with the smart device strapped to the arm of the user. This data collected as a time series is applied thru various transforms, to get the velocity,accelration and jerk values in the three coordinates.

The data set that we have is an extract from 30 subjects for 6 activities

1. WALKING
2. WALKING_UPSTAIRS
3. WALKING_DOWNSTAIRS
4. SITTING
5. STANDING
6. LAYING

We will try to predict these activities from the 561 predictors supplied to the model, and pick the right parameters to be supplied to get an accurate prediction with lowest error. We will also let the algorithm pick the variables of importance and report them.

Analysis

Lets load all the libraries required

Create the train and validation data from the UCI dataset and further split the train data into train set and test set, so that we can use the validation data only to obtain the models final predictions

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(doParallel)) install.packages("doParallel", repos = "http://cran.us.r-project.org")
if(!require(glmnet)) install.packages("glmnet", repos = "http://cran.us.r-project.org")
if(!require(party)) install.packages("party", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")

#create temp space
dload <- tempfile()
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00240/UCI%20HAR%20Dataset.zip"
download.file(url, dload)

#get column names
cnames <- fread(text = readLines(unzip(dload,"UCI HAR Dataset/features.txt")),
                col.names = c("colId", "features"))
```

```
#get activity names
anames <- fread(text = readLines(unzip(dload,"UCI HAR Dataset/activity_labels.txt")),
                col.names = c("actId", "activity"))
```

Observations and features cleanup

Lets now see how many fatures we have to predict the human activity

```
dim(cnames)
```

```
## [1] 561   2
```

There are 561 observations, which is a lot of predictors, we have to short list them or rely on a model that does the feature selection for us.

Now let us see if there are any duplicate feature names, given the number of features. If we do see a lot of duplicates, we may have to proceed with our analysis with feature indexes.

```
check <- as.data.frame(unique(cnames$features)) #get unique column names
names(check) <- "features"
# get a few duplicates
check %>% left_join(cnames) %>% group_by(features) %>% summarize(n=n()) %>% filter(n>1) %>% head
```

```
## Joining, by = "features"
```

```
## Warning: Column `features` joining factor and character vector, coercing into
## character vector
```

```
## # A tibble: 6 x 2
##   features                        n
##   <chr>                       <int>
## 1 fBodyAcc-bandsEnergy()-1,16     3
## 2 fBodyAcc-bandsEnergy()-1,24     3
## 3 fBodyAcc-bandsEnergy()-1,8      3
## 4 fBodyAcc-bandsEnergy()-17,24    3
## 5 fBodyAcc-bandsEnergy()-17,32    3
## 6 fBodyAcc-bandsEnergy()-25,32    3
```

There are a lot of duplicates, and hence we will not assign the names to the data set.

Lets now get the train and validation data

```
#get training data set into three objects
#x containing the predictors
#y containing the activites associated
#sid contains the subject ID associated with the training data which can be used for grouping
x_train <- fread(text = readLines(unzip(dload, "UCI HAR Dataset/train/X_train.txt")))
y_train <- fread(text = readLines(unzip(dload, "UCI HAR Dataset/train/y_train.txt")),
                 col.names = "activity")
sid_train <- fread(text = readLines(unzip(dload,"UCI HAR Dataset/train/subject_train.txt")),
                   col.names = "subjectId")

#now do the same for test data
#x containing the predictors
#y containing the activites associated
#sid contains the subject ID associated with the test data which can be used for grouping
x_test <- fread(text = readLines(unzip(dload, "UCI HAR Dataset/test/X_test.txt")))
y_test <- fread(text = readLines(unzip(dload, "UCI HAR Dataset/test/y_test.txt")),
                col.names = "activity")
```

```
sid_test <- fread(text = readLines(unzip(dload,"UCI HAR Dataset/test/subject_test.txt")),
                  col.names = "subjectId")
```

The outcome is stored in a different table, and on inspection it is a multinominal categorical non ordinal data. Thus we need to convert the data into a factor with 6 levels.

```
#since y is categorical data, lets convert it into a factor
y_train$activity <- as.factor(y_train$activity)
y_test$activity <- as.factor(y_test$activity)
```

Now we are ready to combine it into a dataframe to be used in fitting of prediction models.

```
#combine the x and y training and test data
data_train <- cbind(x_train,y_train)
data_valid <- cbind(x_test,y_test)
#make sure activity is a factor
class(data_train$activity)
```

```
## [1] "factor"
```

Let us now observe the data and see if needs to be cleaned up

```
head(data_train[,1:5])
```

```
##           V1          V2         V3         V4         V5
## 1: 0.2885845 -0.02029417 -0.1329051 -0.9952786 -0.9831106
## 2: 0.2784188 -0.01641057 -0.1235202 -0.9982453 -0.9753002
## 3: 0.2796531 -0.01946716 -0.1134617 -0.9953796 -0.9671870
## 4: 0.2791739 -0.02620065 -0.1232826 -0.9960915 -0.9834027
## 5: 0.2766288 -0.01656965 -0.1153619 -0.9981386 -0.9808173
## 6: 0.2771988 -0.01009785 -0.1051373 -0.9973350 -0.9904868
```

Initial inspection of the data set shows that there are no NA values to be cleaned up, and that all data for the predictors has been standardized. Let us confirm that with the code below.

```
x_list <- names(x_train)
#if any column has data <-1 or >+1 or NA then mark Clean = "NO" else "YES"
clean <- ifelse(sapply(x_list, function(x)
  any(data_train$x < -1 | data_train$x > 1 | is.na(data_train$x))),"NO", "YES")

all(clean == "YES")
```

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

Observations on data

1. There are no NA values that need to be imputed
2. The range of the values in the predictors appears to have been standardized to vary between -1 and +1
3. The data seems to be tidy, as all the fatures are extracted thru transforms

We will now split the train data into train set and test set as we will use the original test data as our validation data.

```
set.seed(1,sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_ind <- createDataPartition(data_train$activity,times = 1,p=.2,list = FALSE)
train_set <- data_train[-test_ind,]
test_set <- data_train[test_ind,]
```

Model Selection process

From the understanding of the data, we have to go for a model that can predict multinominal outcomes.

We have learnt that the **logistic regression** models are best suited for binomial predictions. We also know that we have too many predictors **561** which is a lot for simpler glm models. Hence we look out for regressions that not only performs minimization of error, but does feature selection as well.

In my research, I did find three related models that caters to just our scenario, and we will evaluate them in detail now.

Ridge / LASSO / ElasticNet Regressions

Ridge regression is a regularization technique that drives the model coefficients to be small by adjusting the loss function to include a penalty term that is based on the coeffiecients. The ridge loss function includes a regularization hyperparameter α that determines how much weight should be placed on the penalty term. Different values of this hyperparameter will result in different models.

Goal: Minimize $Loss\left(\hat{\beta}_0, ..., \hat{\beta}_m\right) = \frac{1}{n} SSE + \lambda \sum_{i=1}^{m} \hat{\beta}_i^2$

LASSO regression is also a regularization method that encourages small parameter values by adding a penalty term to the loss function. This penalty term has a slightly different form from the one encountered in ridge regression.

Goal: Minimize $Loss\left(\hat{\beta}_0, ..., \hat{\beta}_m\right) = \frac{1}{n} SSE + \lambda \sum_{i=1}^{m} |\hat{\beta}_i|$

Elasticnet is a combination of both Ridge and LASSO, and and uses the α value to pick a regression between LASSO and Ridge When α is 0, the second term becomes 0 and hence becomes a Ridge model, and when α is set to 1, the first term becomes 0 and the model translates to LASSO

Goal: $\min_{\beta_0, \beta \in R^{p+1}} \frac{1}{2N} \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta\right)^2 + \lambda \left[(1-\alpha) \frac{\|\beta\|_2^2}{2} + \alpha \|\beta\|_1\right]$

These models are very expensive on the laptop, and takes a long time to train, thus two choices have been provided

1. The user can answer "Y" for the question, and We can cluster the R environment to run the train on all the CPU's of the computer and then coalesce the results.
2. The peer reviewer may be interested in just testing out the model and may not have the time to train, in which case there is a questionaire and if the reviewer puts in a "N", it looks for three models in the working directory, that can be downloaded from my google drive - Download Here.

```
musttrain <- ""
musttrain <- as.character(readline(prompt="Enter Y - if you want to Train or
             N - if you want to load model from saved rds file provided : "))
if(toupper(musttrain) == "Y") {

#set up a Ridge model
control <- trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 5)
#since the data is regularized, the weights applied are very close to zero for
#highest accuracy
```

```
grid <- expand.grid(alpha=0, #set to 0 for ridge model
                    lambda=10^-5)

#make a cluster of number of processors -1
cl <- makePSOCKcluster(7)
registerDoParallel(cl)

#Ridge model

set.seed(1,sample.kind = "Rounding")

ridge_model <- train(activity~.,
                     train_set,
                     method = "glmnet",
                     tuneGrid = grid,
                     trControl = control)



#Lasso model

grid <- expand.grid(alpha=1, #set to 1 for lasso model
                    lambda=10^-5)

set.seed(1,sample.kind = "Rounding")

lasso_model <- train(activity~.,
                     train_set,
                     method = "glmnet",
                     tuneGrid = grid,
                     trControl = control)

#Elasticnet model

grid <- expand.grid(alpha=seq(0,1,length=10), #set 0 to 1 for Elastic model
                    lambda=10^-5)

set.seed(1,sample.kind = "Rounding")

elastic_model <- train(activity~.,
                     train_set,
                     method = "glmnet",
                     tuneGrid = grid,
                     trControl = control)
##stop cluster
stopCluster(cl)

#saving the three models
saveRDS(ridge_model,"Ridge_model.rds")
saveRDS(lasso_model,"Lasso_model.rds")
saveRDS(elastic_model,"Elastic_model.rds")

}
```

```
#load models
if(toupper(musttrain) == "N") {
  ridge_model <-readRDS("Ridge_model.rds")
  lasso_model <-readRDS("Lasso_model.rds")
  elastic_model <-readRDS("Elastic_model.rds")
}
```

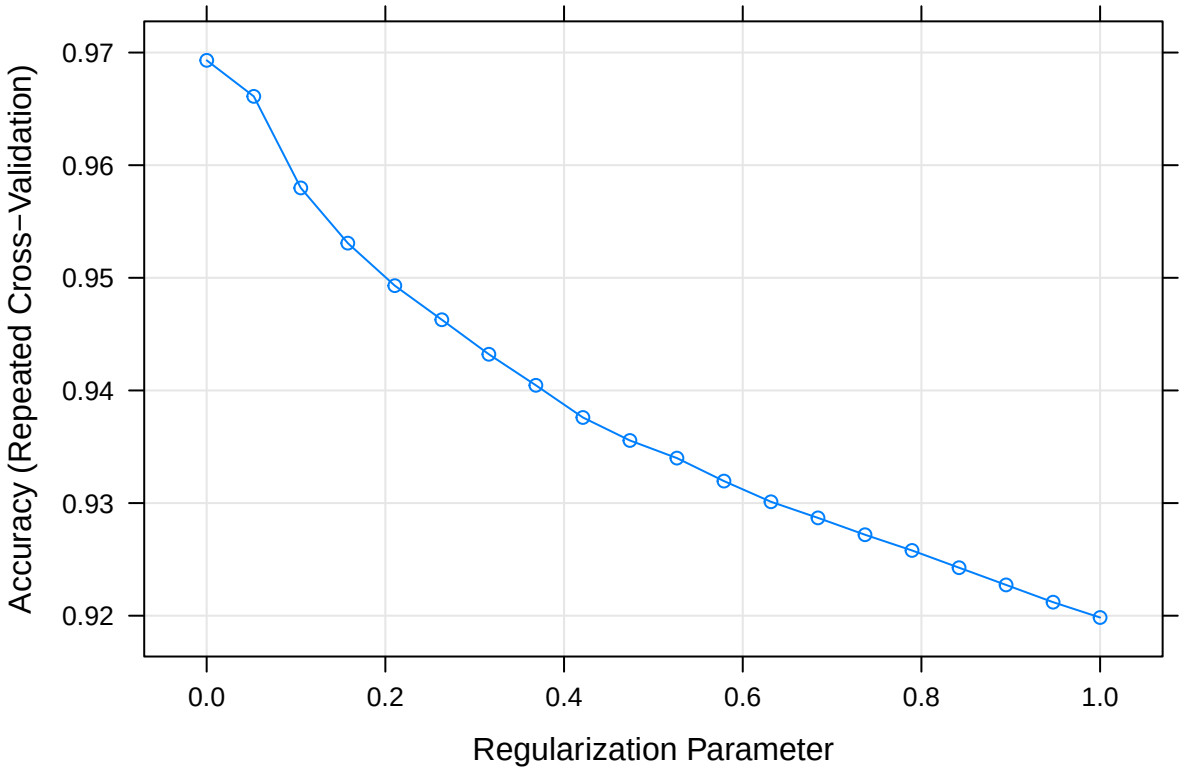
```
  ridge_model <-readRDS("Ridge_model.rds")
  lasso_model <-readRDS("Lasso_model.rds")
  elastic_model <-readRDS("Elastic_model.rds")
```

Lets now Analyze the different models and see which would be a good model to adapt.

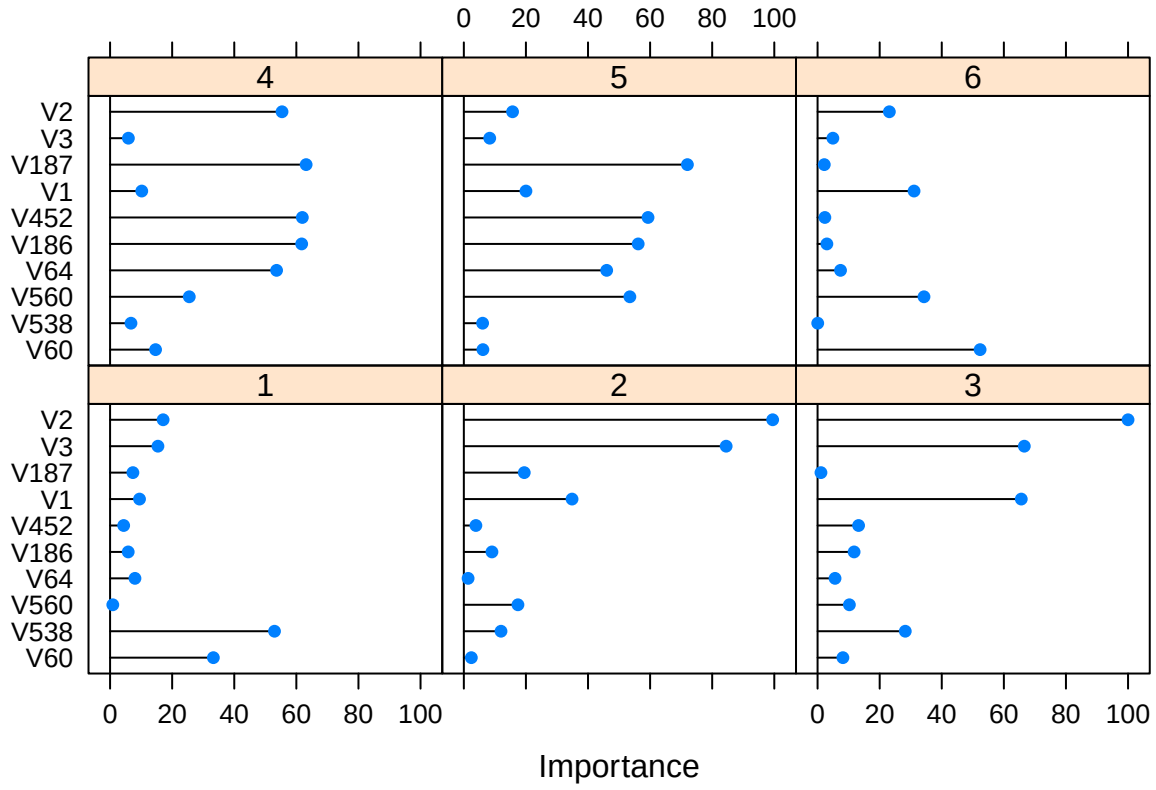
```
#get the details from the model
ridge_model
```

```
## glmnet
##
## 5878 samples
##  561 predictor
##    6 classes: '1', '2', '3', '4', '5', '6'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 5292, 5290, 5290, 5291, 5290, 5290, ...
## Resampling results across tuning parameters:
##
##   lambda      Accuracy   Kappa
##   0.00010000  0.9693080  0.9630532
##   0.05272632  0.9661096  0.9592022
##   0.10535263  0.9579795  0.9494128
##   0.15797895  0.9530796  0.9435115
##   0.21060526  0.9493033  0.9389633
##   0.26323158  0.9462758  0.9353170
##   0.31585789  0.9432127  0.9316264
##   0.36848421  0.9404576  0.9283069
##   0.42111053  0.9375998  0.9248640
##   0.47373684  0.9355588  0.9224057
##   0.52636316  0.9339944  0.9205216
##   0.57898947  0.9319526  0.9180611
##   0.63161579  0.9301147  0.9158464
##   0.68424211  0.9286855  0.9141240
##   0.73686842  0.9271882  0.9123190
##   0.78949474  0.9257938  0.9106388
##   0.84212105  0.9242625  0.9087942
##   0.89474737  0.9227318  0.9069497
##   0.94737368  0.9212003  0.9051041
##   1.00000000  0.9198395  0.9034636
##
## Tuning parameter 'alpha' was held constant at a value of 0
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0 and lambda = 1e-04.
```

```
#as we can see the highest accuracy is obtained with the lowest lambda
plot(ridge_model)
```




```
#get the variables of importance
plot(varImp(ridge_model,scale = TRUE),top = 10)
```



We see from the first plot that the highest accuracy of **96.9 %**is obtained from the lowest lambda. and from the second plot we see that for for activities 2,3 and 4 (walking upstairs,downstairs and sitting) the most important predictor is V2 and V3

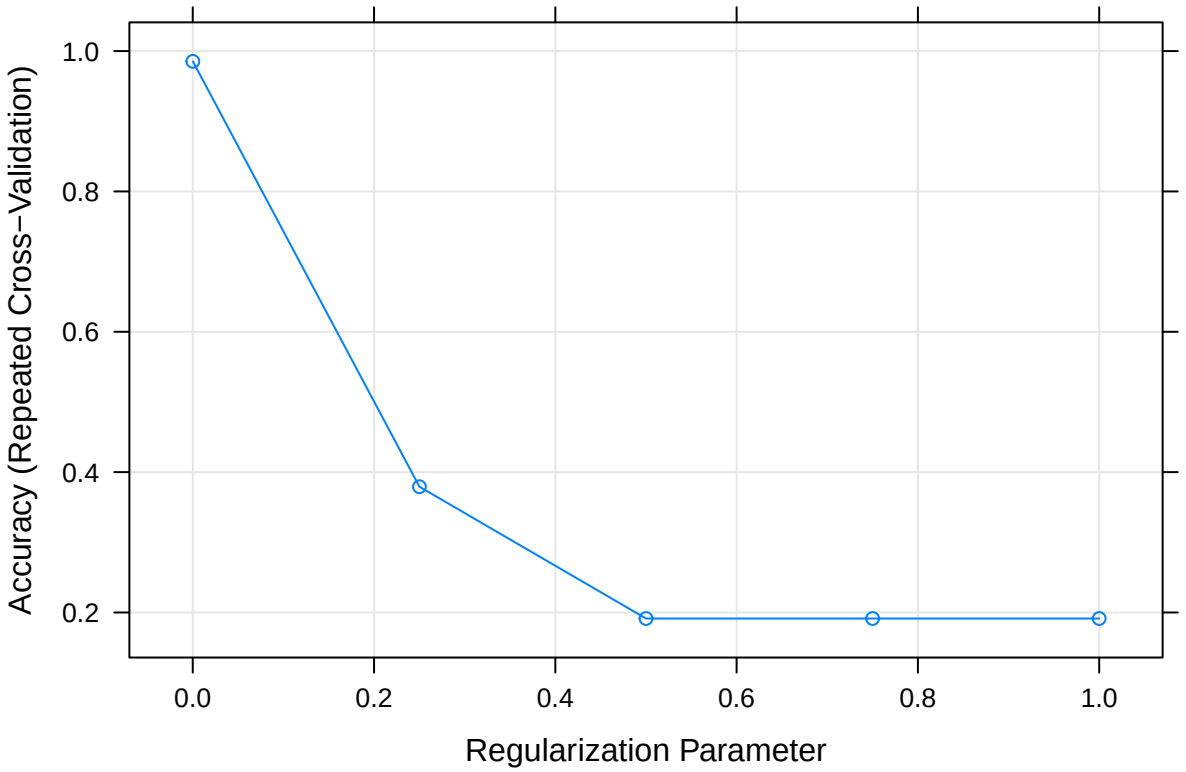
Lets analyze the Lasso model now

```
#get the details from the model
lasso_model
```

```
## glmnet
##
## 5878 samples
##  561 predictor
##    6 classes: '1', '2', '3', '4', '5', '6'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 5292, 5290, 5290, 5291, 5290, 5290, ...
## Resampling results across tuning parameters:
##
##   lambda    Accuracy   Kappa
##   0.000100  0.9853672  0.9823864
##   0.250075  0.3791768  0.2356006
##   0.500050  0.1913913  0.0000000
##   0.750025  0.1913913  0.0000000
##   1.000000  0.1913913  0.0000000
##
```

```
## Tuning parameter 'alpha' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 1 and lambda = 1e-04.
```

```
plot(lasso_model)
```

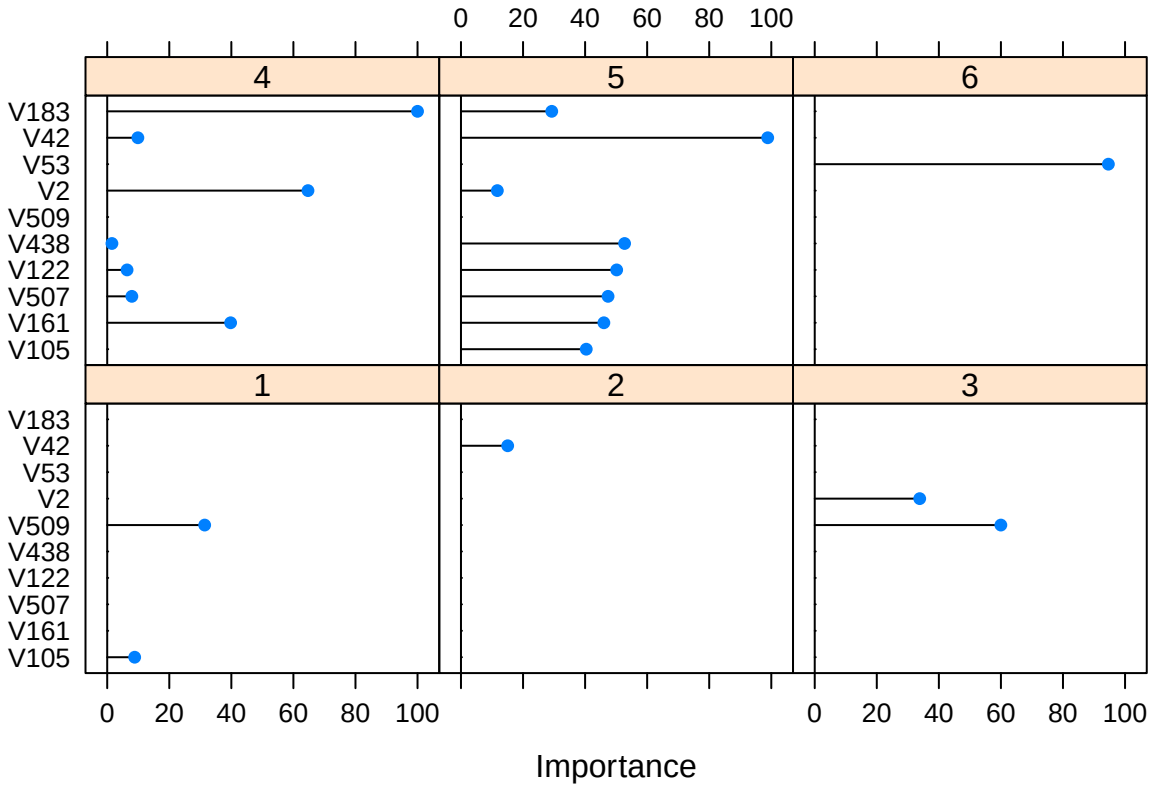



```
#get the variables of importance
varImp(lasso_model)
```

```
## glmnet variable importance
##
##   variables are sorted by maximum importance across the classes
##   only 20 most important variables shown (out of 561)
##
##           1       2     3         4      5     6
## V183  0.000  0.0000  0.00 100.00000 29.253  0.00
## V42   0.000 15.0561  0.00   9.87673 98.853  0.00
## V53   0.000  0.0000  0.00   0.00000  0.000 94.63
## V2    0.000  0.0000 33.83  64.69632 11.724  0.00
## V509 31.409  0.0000 59.99   0.00000  0.000  0.00
## V438  0.000  0.0000  0.00   1.51662 52.726  0.00
## V122  0.000  0.0000  0.00   6.39541 50.174  0.00
## V507  0.000  0.0000  0.00   7.93502 47.397  0.00
## V161  0.000  0.0000  0.00  39.77004 46.055  0.00
## V105  8.848  0.0000  0.00   0.00000 40.368  0.00
## V48   0.000  0.0000  0.00  37.57951  0.000  0.00
## V187  0.000  2.9777  0.00  28.37493 33.636  0.00
```

```
## V244  2.919  0.5057  0.00   0.03083 32.326  0.00
## V121  0.000  0.0000  0.00  16.01003 31.225  0.00
## V491  0.000 31.0437  0.00   0.00000  0.000  0.00
## V247  1.980  0.0000  0.00  13.76426 30.932  0.00
## V248  0.000  0.0000  0.00  30.39762  2.902  0.00
## V90   0.000  0.0000 29.50   0.00000  0.000  0.00
## V412  0.000 27.8593 19.49   0.00000  0.000  0.00
## V38  26.841  0.0000 25.61   0.00000  8.106  0.00
```

```
plot(varImp(lasso_model,scale = TRUE),top = 10)
```



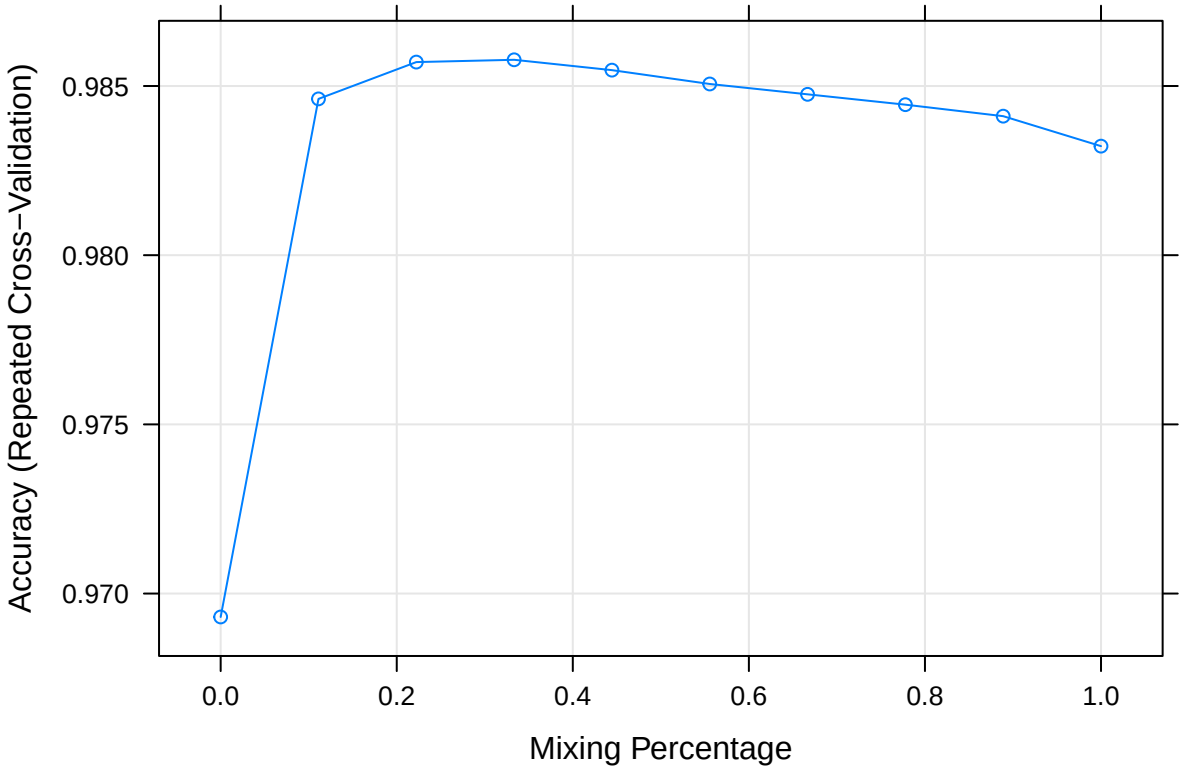
The highest accuracy of **98.53** %is obtained from the lowest lambda. and from the second plot that similarly for for activities 2,3 and 4 (walking upstairs,downstairs and sitting) the most important predictor is V2 and V3.

```
#get the details from the model
elastic_model
```

```
## glmnet
##
## 5878 samples
##  561 predictor
##    6 classes: '1', '2', '3', '4', '5', '6'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 5292, 5290, 5290, 5291, 5290, 5290, ...
## Resampling results across tuning parameters:
```

```
##
##   alpha      Accuracy   Kappa
##   0.0000000  0.9693080  0.9630532
##   0.1111111  0.9846192  0.9814860
##   0.2222222  0.9857068  0.9827955
##   0.3333333  0.9857744  0.9828768
##   0.4444444  0.9854685  0.9825087
##   0.5555556  0.9850602  0.9820172
##   0.6666667  0.9847544  0.9816490
##   0.7777778  0.9844486  0.9812809
##   0.8888889  0.9841082  0.9808710
##   1.0000000  0.9832230  0.9798051
##
## Tuning parameter 'lambda' was held constant at a value of 1e-05
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.3333333 and lambda = 1e-05.
```

```
plot(elastic_model)
```

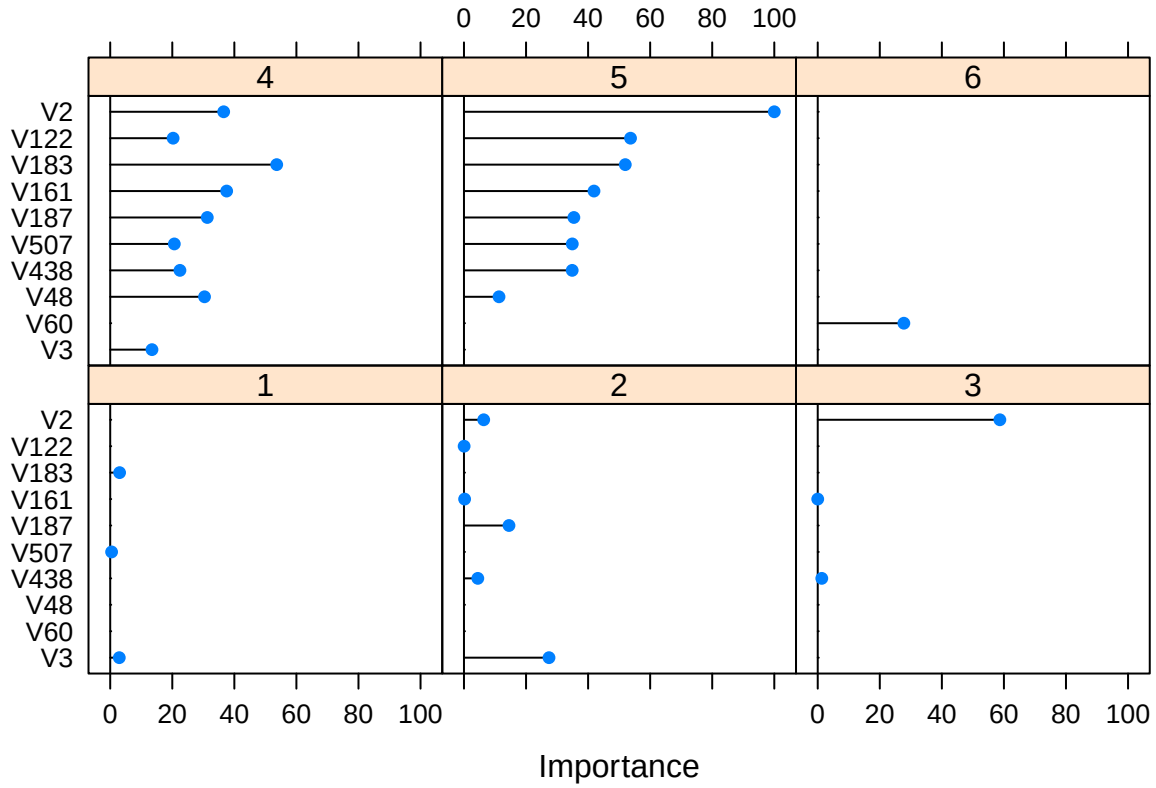



```
#get the variables of importance
varImp(elastic_model)
```

```
## glmnet variable importance
##
##   variables are sorted by maximum importance across the classes
##   only 20 most important variables shown (out of 561)
##
```

```
##             1        2         3      4       5      6
## V2     0.0000  6.35461 58.690928 36.566 100.000  0.000
## V122   0.0000  0.05358  0.000000 20.269  53.678  0.000
## V183   3.0242  0.00000  0.000000 53.659  51.983  0.000
## V161   0.0000  0.20258  0.004644 37.523  41.900  0.000
## V187   0.0000 14.50592  0.000000 31.279  35.412  0.000
## V507   0.4425  0.00000  0.000000 20.655  34.907  0.000
## V438   0.0000  4.45633  1.306785 22.468  34.874  0.000
## V48    0.0000  0.00000  0.000000 30.394  11.288  0.000
## V60    0.0000  0.00000  0.000000  0.000   0.000 27.755
## V3     2.9185 27.40266  0.000000 13.458   0.000  0.000
## V42    0.0000  6.38363  4.173932 16.806  24.364  5.835
## V560   0.0000  5.43926  3.804453  9.823  23.474  7.993
## V1     0.0000  0.00000 23.306441  5.887   0.000  0.000
## V247   3.2838  0.00000  0.000000 21.970  23.266  0.000
## V244   2.9063  2.89016  1.344821 10.919  23.201  0.000
## V51    0.0000  6.05303  3.037619 15.292  22.733  5.664
## V162   0.0000  2.92936  0.000000  8.668  22.403  0.000
## V121   0.0000  0.00000  0.000000 15.392  22.387  0.000
## V538 22.1899  0.00000  4.156800  2.682   8.792  0.000
## V82    0.0000  0.05051  0.000000 10.833  22.117  0.000
```

```
plot(varImp(elastic_model,scale = TRUE),top = 10)
```



We can also compare the different models by using the resampling feature of the caret package and merge the results into one pane for easier contrast.

```r
#we can compare the models by combining them to an array and listing out
#their properties.
models <- list(ridge = ridge_model,lasso = lasso_model,elastic = elastic_model)
resample <- resamples(models)
summary(resample)
```

```
##
## Call:
## summary.resamples(object = resample)
##
## Models: ridge, lasso, elastic
## Number of resamples: 50
##
## Accuracy
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ridge   0.9540816 0.9646673 0.9685648 0.9693080 0.9757152 0.9829932    0
## lasso   0.9744463 0.9817197 0.9847069 0.9853672 0.9897916 0.9948980    0
## elastic 0.9762309 0.9829932 0.9863946 0.9857744 0.9881104 0.9966044    0
##
## Kappa
##              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## ridge   0.9447370 0.9574696 0.9621576 0.9630532 0.9707643 0.9795305    0
## lasso   0.9692357 0.9779944 0.9815920 0.9823864 0.9877115 0.9938593    0
## elastic 0.9713896 0.9795288 0.9836224 0.9828768 0.9856888 0.9959126    0
```

We can see that the Elastic model had the max accuracy of **99.66 %** which is a nice to have in the predictive world.

Till now we have only dealt with the Train set of data and obtained the accuracies of the model. Let us now apply the model on the Test set and obtain the accuracies to make sure that the model we are about to pick would generate good results from data that was not used for training.

```r
#Using the test set to predict accuracy
#ridge accuracy
ridge_predict <- predict(ridge_model,test_set)
mean(as.numeric(as.character((ridge_predict))) ==
     as.numeric(as.character(test_set$activity)))
```

```
## [1] 0.972863
```

```r
#lasso accuracy
lasso_predict <- predict(lasso_model,test_set)
mean(as.numeric(as.character((lasso_predict))) ==
     as.numeric(as.character(test_set$activity)))
```

```
## [1] 0.9850746
```

```r
#elasticnet accuracy
elastic_predict <- predict(elastic_model,test_set)
mean(as.numeric(as.character((elastic_predict))) ==
     as.numeric(as.character(test_set$activity)))
```

```
## [1] 0.9857531
```

The Elastic Model tops the list with **99.3 %** accuracy

Let us now check to see where the errors are, for that we will use the confusion matrix and see where the predictions went wrong.

```r
#confusion matricies
cm_ridge <- confusionMatrix(ridge_predict,test_set$activity)
#ridge cm
cm_ridge$table
```

```
##           Reference
## Prediction   1   2   3   4   5   6
##          1 244   0   2   0   0   0
##          2   1 214   0   0   0   0
##          3   1   1 196   0   0   0
##          4   0   0   0 240  17   0
##          5   0   0   0  18 258   0
##          6   0   0   0   0   0 282
```

```r
#ridge error (misclassification %)
1-sum(diag(cm_ridge$table))/sum(cm_ridge$table)
```

```
## [1] 0.02713704
```

```r
cm_lasso <- confusionMatrix(lasso_predict,test_set$activity)
#lasso cm
cm_lasso$table
```

```
##           Reference
## Prediction   1   2   3   4   5   6
##          1 246   0   0   0   0   0
##          2   0 214   0   1   0   0
##          3   0   1 198   0   0   0
##          4   0   0   0 250  13   0
##          5   0   0   0   7 262   0
##          6   0   0   0   0   0 282
```

```r
#lasso error (misclassification %)
1-sum(diag(cm_lasso$table))/sum(cm_lasso$table)
```

```
## [1] 0.01492537
```

```r
cm_elastic <- confusionMatrix(elastic_predict,test_set$activity)
#elastic cm
cm_elastic$table
```

```
##           Reference
## Prediction   1   2   3   4   5   6
##          1 246   0   0   0   0   0
##          2   0 215   0   0   0   0
##          3   0   0 198   0   0   0
##          4   0   0   0 249  12   0
##          5   0   0   0   9 263   0
##          6   0   0   0   0   0 282
```

```r
#elastic error (misclassification %)
1-sum(diag(cm_elastic$table))/sum(cm_elastic$table)
```

```
## [1] 0.01424695
```

From what we can see, all the three models have struggled with predicting standing (activity 5), This may be due to the fact that there are too many variables of importance that are contributing to the prediction of activity 5.

The errors have been minimal for the Elastic Model as expected.

Considering all the exploratory analysis we have done so far, we are now comfortable to pick the best fit of the Elastic model

We can use the coefficients from the best model to get an idea about the predictors considered for the various activities

```
#picking the model
elastic_model$bestTune
```

```
##       alpha lambda
## 4 0.3333333  1e-05
```

```
elastic_best <- elastic_model$finalModel
coefficients <- coef(elastic_best,s=elastic_model$bestTune$lambda)
predictor <- data.frame(predictors = names(coefficients[[1]][,1]),
                        y1 = coefficients[[1]][,1],y2 = coefficients[[2]][,1],
                        y3 = coefficients[[3]][,1],y4 = coefficients[[4]][,1],
                        y5 = coefficients[[5]][,1],y6 = coefficients[[6]][,1])

head(predictor)
```

```
##              predictors        y1          y2          y3         y4
## (Intercept) (Intercept) 1.3835947  1.55228295 11.13809688 -2.7215175
## V1                   V1 0.0000000  0.00000000  3.33676381 -0.8427791
## V2                   V2 0.0000000 -0.90978369  8.40273132 -5.2351003
## V3                   V3 0.4178393 -3.92321654  0.00000000 -1.9267627
## V4                   V4 0.0000000  0.78100376  0.02989568  0.0000000
## V5                   V5 0.2514993  0.03487206  0.00000000  0.0000000
##                        y5       y6
## (Intercept) -22.993524661 11.64107
## V1            0.000000000  0.00000
## V2           14.316916829  0.00000
## V3            0.000000000  0.00000
## V4           -0.001534673  0.00000
## V5           -0.034535355  0.00000
```

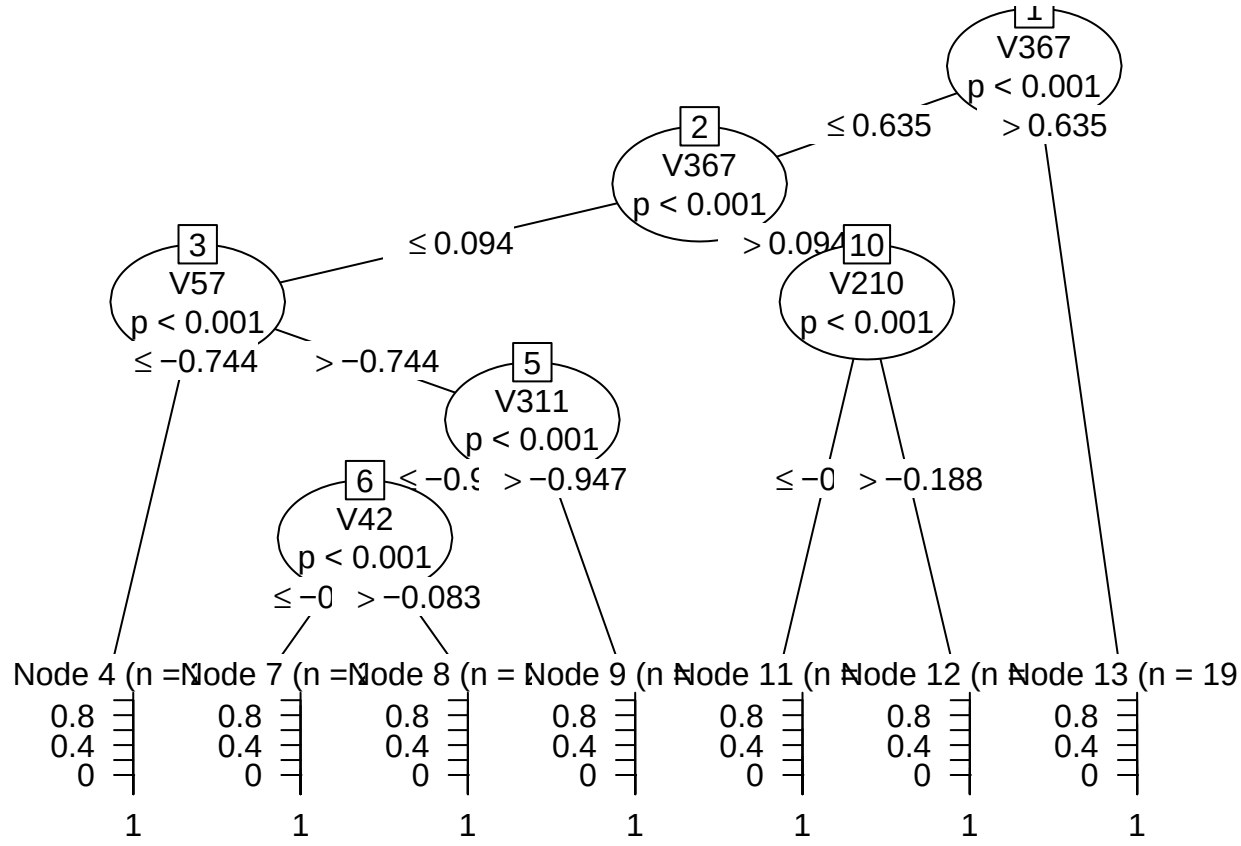
The predictions can be obtained by multiplying the predictor value with their coefficients and summing them all up with the intercept value.

Decision Tree

If we would have approached the problem with decision trees instead of logistic methods, we would have arrrived at a classification based on the value of the predictor being lesser than or greater than certain values to make our decision trees, and then prune it to a desired number of branches to base our predictions.

I have provided a small example to compare and contrast the two methodologies.

```
#plot decision tree
#we will now use ctree from the party package to view the decision tree
#we have used some pruning parameters to make the tree look pretty
# this is for an example purpose only and not actually used for modelling
tree <- ctree(activity~.,data = test_set,controls = ctree_control(mincriterion = 0.99,
                                                                  minsplit = 400))
plot(tree)
```



As we can see from the bar graphs shown at the bottom nodes, some activities are relatively easy to predict just with a few variables. The errors in this particular case is very high.

Now we are ready to do our run on the validataion data set (which we have not used for training or testing our models) and get the accuracy of predictions.

Validation

```
#Result
#validation set checks
elastic_predict_valid <- predict(elastic_model,data_valid)
#elastic accuracy
mean(as.numeric(as.character((elastic_predict_valid))) ==
               as.numeric(as.character(data_valid$activity)))
```

```
## [1] 0.956566
```

```
#create confusion matrix
cm_elastic_valid <- confusionMatrix(elastic_predict_valid,data_valid$activity)
#elastic cm
cm_elastic_valid$table
```

```
##            Reference
## Prediction   1   2   3   4   5   6
##          1 494  24   7   0   1   0
##          2   0 445  15   3   0   0
##          3   2   2 397   0   0   0
##          4   0   0   0 426  11   0
```

```
##          5   0   0   1  61 520   0
##          6   0   0   0   1   0 537
```

```
#error (misclassification)
1-sum(diag(cm_elastic_valid$table))/sum(cm_elastic_valid$table)
```

```
## [1] 0.043434
```

Results

We have obtained an accuracy of **95.65 %** on the validation set. This confirms no over training and an adequate model to present the results.

The confusion matrix shows that the model had some challenges with the classification of Standing activity, and that again may be due to too many predictor identified to be responsible for more than 90% of the predictability of the activity.

We can probably better the results marginally by running the cross-validation against the validation set, but since no operation was allowed on the validation set, the final results are as presented.

Conclusion

From the various models applied on the HAR(Human Activity Recognition) data, we were able to classify the activities with an accuracy of **95.65 %** which is a significant value for machine learning modules. We were able to achieve this by evaluating the various techniques available for multivariate classification, and fine tuning them with cross validation and regression methods that would minimize the prediction errors.

We have also explored setting up clusters for incorporating a fit and train a model for data that had a large value of predictors. The models are also provided as a RDS objects, which could be used to perform peer reviews, and can be downloaded from the link provided.

Download Here

Some of the other methodology tried to pick the effects failed due to the nature and size of information provided for analysis which include linear regression, k - nearest neighbor and random forest.

References

Ridge and Lasso regression: http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/