Fall and Movement Detection Project

Introduction

Falls among the elderly is an important health issue. Fall detection and movement tracking are therefore instrumental in addressing this issue. This project responds to the challenge of classifying different movements as a part of a system designed to fulfill the need for a wearable device to collect data for fall and near-fall analysis. Four different fall trajectories (forward, backward, left and right), three normal activities (standing, walking and lying down) and near-fall situations are identified and detected.

Falls are a serious public health problem and possibly life threatening for people in fall risk groups. Data was collected by 3 researches wearable motion sensor units fitted to the subjects' body at six different positions. Fourteen volunteers perform a standardized set of movements including 20 voluntary falls and 16 activities of daily living (ADLs), resulting in a large dataset with 2520 trials. To reduce the computational complexity of training and testing the classifiers, data focus on the raw data for each sensor in a 4 s time window.

Dataset

Data set that can be found here (https://www.dropbox.com/s/e2hc5rinwkectsp/falldeteciton.csv?dl=1) contains around 16.4k records, each record contains multiple biometric measures: Sugar Level (SL), EEG monitoring rate (EEG), Blood Pressure (BP), Heart Beat Rate (HR), and Blood Circulation (CIRCLUATION). Record is classified also by activity with a timestamp, 6 activities was noted in this data set: Standing, Walking, Sitting, Falling, Cramps, and Running.

```
##################################################################
# Create dataset and validation set
##################################################################

# Note: this process could take few seconds

if(!require(ellipse)) install.packages("ellipse")
if(!require(data.table)) install.packages("data.table")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

# Download dataset file
dl <- tempfile()
download.file("https://www.dropbox.com/s/e2hc5rinwkectsp/falldeteciton.csv?dl=1", dl)
# load the CSV file from the temp file
dataset <- read.csv(dl, header=FALSE)
# set the column names in the dataset
colnames(dataset) <- c("ACTIVITY","TIME","SL","EEG","BP","HR","CIRCLUATION")
```

We are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be.

We will split the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

```
# create a list of 80% of the rows in the original dataset we can use for training
validation_index <- createDataPartition(dataset$ACTIVITY, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- dataset[-validation_index,]
```

```
# use the remaining 80% of data to training and testing the models
dataset <- dataset[validation_index,]
```

We now have training data in the dataset variable and a validation set we will use later in the validation variable.

Data Exploration and visualization

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the dim function.

```
dim(dataset)
```

```
## [1] 13109     7
```

Knowing the types is important as it will give us an idea of how to better summarize the data we have and the types of transforms we might need to use to prepare the data before we model it.

```
sapply(dataset, class)
```

```
##    ACTIVITY        TIME          SL         EEG          BP          HR
##    "factor"   "numeric"   "numeric"   "numeric"   "integer"   "integer"
## CIRCLUATION
##   "integer"
```

It is also always a good idea to actually eyeball our data.

```
head(dataset)
```

```
##   ACTIVITY    TIME       SL      EEG BP  HR CIRCLUATION
## 1  falling 4722.92  4019.64 -1600.00 13  79         317
## 2  sitting 4059.12  2191.03 -1146.08 20  54         165
## 3  sitting 4773.56  2787.99 -1263.38 46  67         224
## 4   cramps 8271.27  9545.98 -2848.93 26 138         554
## 7  falling 8620.28 24949.90 -3198.06 35 157        1519
## 8  falling 9238.73 39245.50 -2590.00 15 196        1885
```

The activity variable is a factor. This is a multi-class or a multinomial classification problem.

```
levels(dataset$ACTIVITY)
```

```
## [1] "cramps"   "falling"  "running"  "sitting"  "standing" "walking"
```

Let's now take a look at the number of instances (rows) that belong to each class

```
percentage <- prop.table(table(dataset$ACTIVITY)) * 100
cbind(freq=table(dataset$ACTIVITY), percentage=percentage)
```

```
##          freq percentage
## cramps   2796  21.328858
## falling  2871  21.900984
## running  1351  10.305897
## sitting  2002  15.271951
## standing 3687  28.125715
## walking   402   3.066595
```

Now finally, we can take a look at a summary of each attribute.

```
summary(dataset)
```

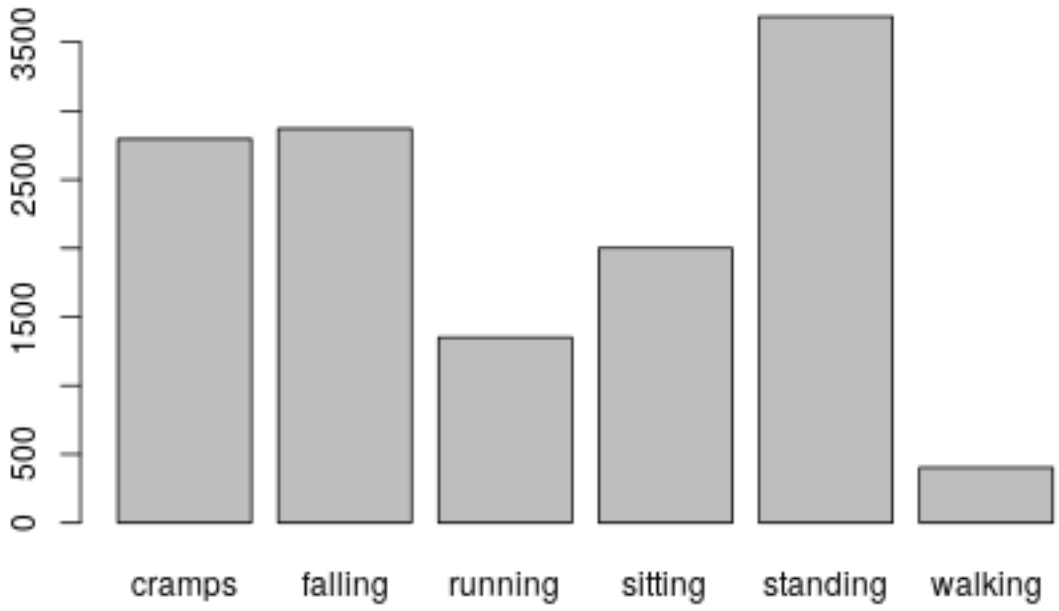
```
##      ACTIVITY          TIME             SL                EEG
##  cramps  :2796   Min.   : 2012   Min.   :     42.2   Min.   :-2508800
##  falling :2871   1st Qu.: 7289   1st Qu.:  10027.3   1st Qu.:   -5612
##  running :1351   Median : 9770   Median :  31433.7   Median :   -3360
##  sitting :2002   Mean   :10946   Mean   :  75290.1   Mean   :   -4517
##  standing:3687   3rd Qu.:13482   3rd Qu.:  80738.7   3rd Qu.:   -2160
##  walking : 402   Max.   :50896   Max.   :2426140.0   Max.   : 1410000
##        BP              HR         CIRCLUATION
##  Min.   :  0.0   Min.   : 33.0   Min.   :    5
##  1st Qu.: 25.0   1st Qu.:119.0   1st Qu.:  587
##  Median : 44.0   Median :180.0   Median : 1585
##  Mean   : 58.4   Mean   :211.5   Mean   : 2894
##  3rd Qu.: 79.0   3rd Qu.:271.0   3rd Qu.: 3539
##  Max.   :533.0   Max.   :981.0   Max.   :52210
```

It is helpful with visualization to have a way to refer to just the input attributes and just the output attributes.

```
x <- dataset[,3:7]
y <- dataset[,1]
```

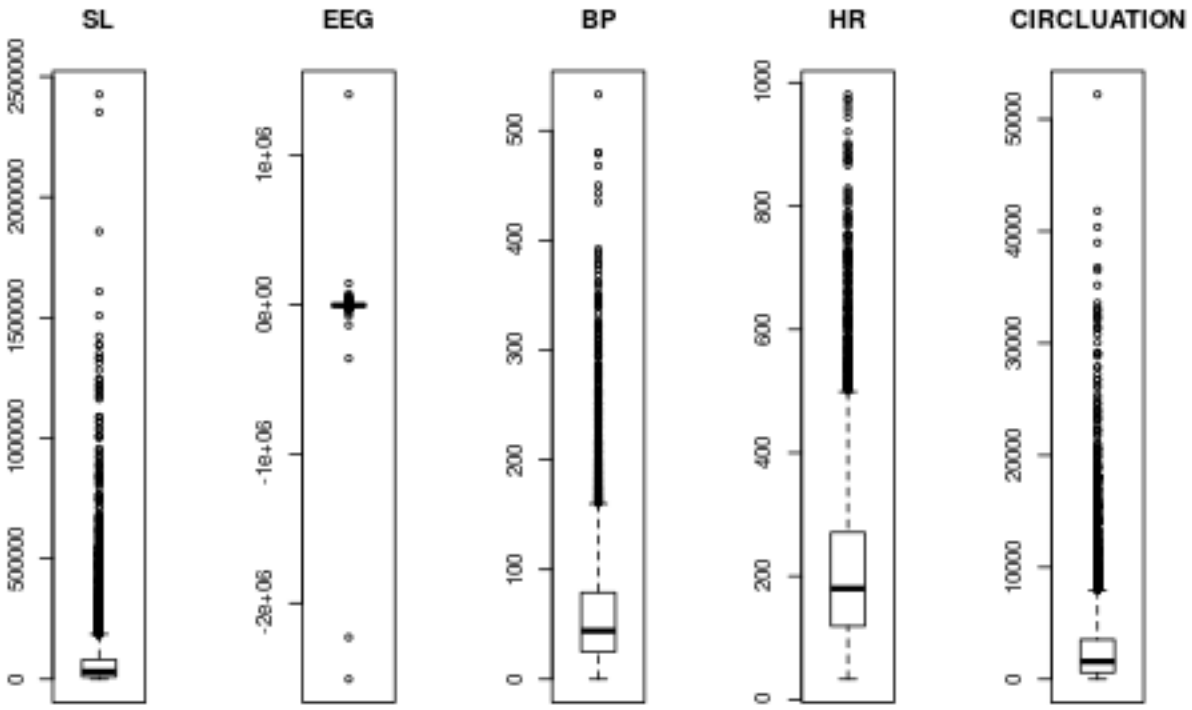
We can create a barplot of the Activity class variable to get a graphical representation of the class distribution.

```
plot(y)
```



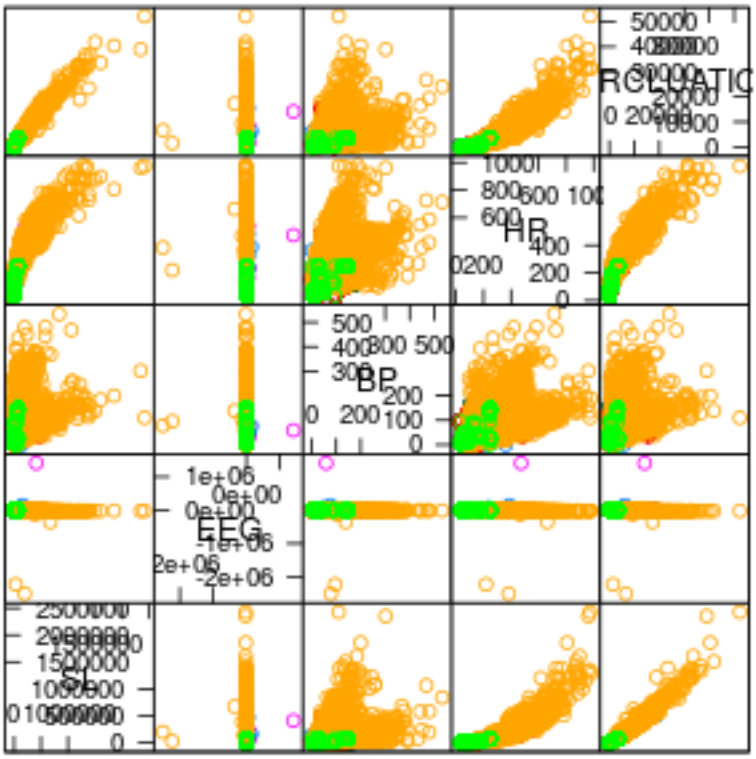
Given that the input variables are numeric, we can create scatterplots of each.

```
par(mfrow=c(1,5))
for(i in 1:5) {
  boxplot(x[,i], main=names(x)[i])
}
```



First let's look at scatterplots of all pairs of attributes and color the points by class.

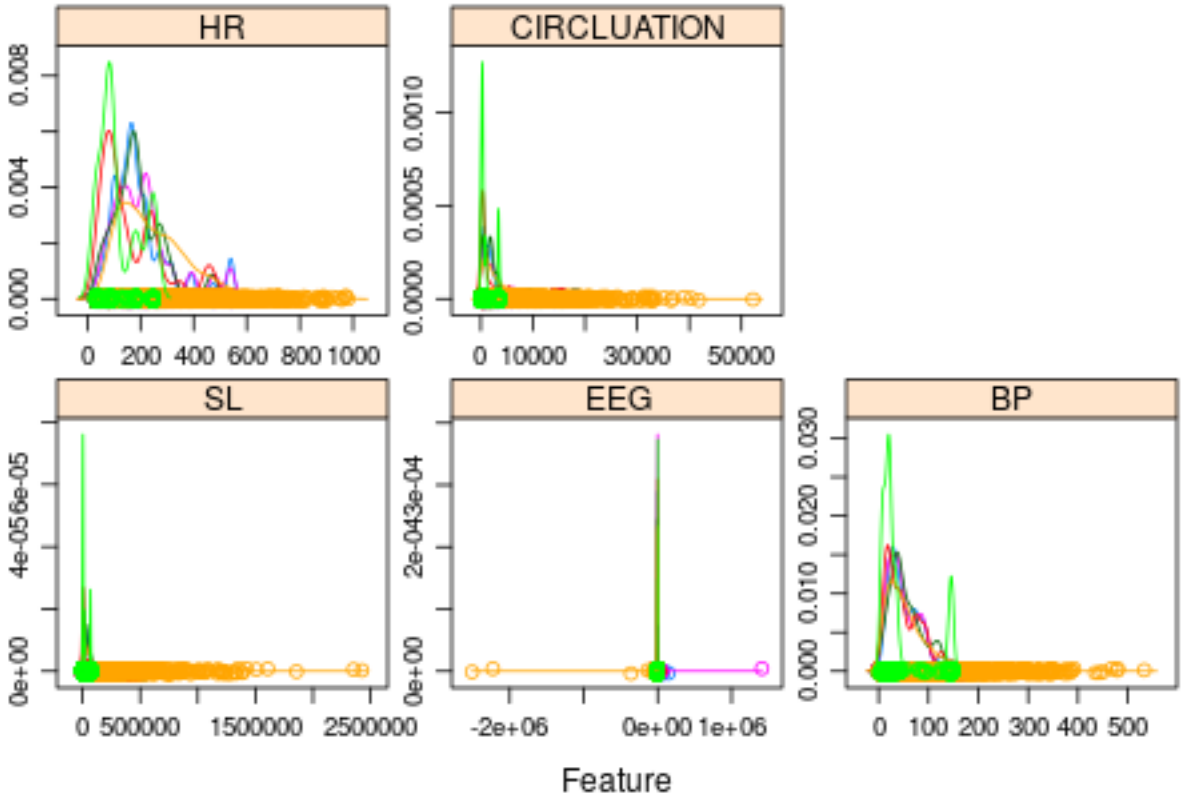
```
featurePlot(x=x, y=y, plot="pairs")
```



Scatter Plot Matrix

Next we can get an idea of the distribution of each attribute broken down by class value. We will use some probability density plots to give nice smooth lines for each distribution.

```
scales <- list(x=list(relation="free"), y=list(relation="free"))
featurePlot(x=x, y=y, plot="density", scales=scales)
```



We will 10-fold crossvalidation to estimate accuracy.

This will split our dataset into 10 parts, train in 9 and test on 1 and release for all combinations of train-test splits. We will also repeat the process 3 times for each algorithm with different splits of the data into 10 groups, in an effort to get a more accurate estimate.

```
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

We are using the metric of "Accuracy" to evaluate models. This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the metric variable when we run build and evaluate each model next.

Modeling

We don't know which algorithm would be good on this problem or what configurations to use. Let's evaluate 4 different algorithms:

```
1- Linear Discriminant Analysis (LDA)
2- Classification and Regression Trees (CART).
3- k-Nearest Neighbors (kNN).
4- Random Forest (RF)
```

This is a good mixture of simple linear (LDA), nonlinear (CART, kNN) and complex nonlinear method (RF). We reset the random number seed before reach run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

a) linear algorithms

```
set.seed(7)
fit.lda <- train(ACTIVITY~., data=dataset, method="lda", metric=metric, trControl=control)
```

b) nonlinear algorithms

CART

```
set.seed(7)
fit.cart <- train(ACTIVITY~., data=dataset, method="rpart", metric=metric, trControl=control)
```

kNN

```
set.seed(7)
fit.knn <- train(ACTIVITY~., data=dataset, method="knn", metric=metric, trControl=control)
```

Random Forest

```
# This may take few minutes to complete
set.seed(7)
fit.rf <- train(ACTIVITY~., data=dataset, method="rf", metric=metric, trControl=control)
```

Results

We now have 4 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate.

We can report on the accuracy of each model by first creating a list of the created models and using the summary function.

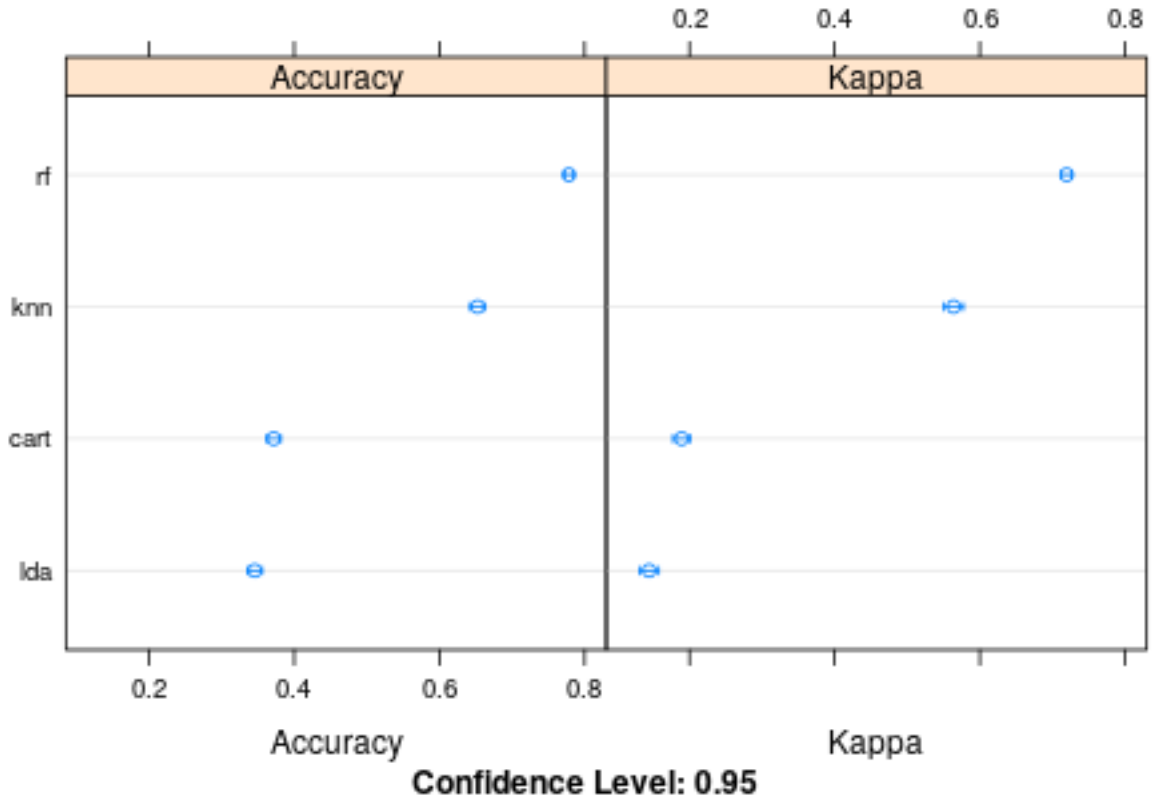
```
results <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, rf=fit.rf))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, cart, knn, rf
## Number of resamples: 10
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda  0.3272311 0.3384193 0.3436308 0.3456392 0.3545671 0.3727134    0
## cart 0.3501144 0.3641051 0.3713315 0.3718061 0.3786921 0.3992366    0
## knn  0.6186117 0.6496949 0.6516768 0.6527591 0.6617059 0.6694656    0
## rf   0.7597254 0.7742612 0.7810830 0.7783215 0.7840931 0.7862595    0
##
```

```
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lda  0.1176480 0.1359365 0.1417388 0.1444597 0.1560947 0.1811119    0
## cart 0.1642353 0.1797209 0.1882149 0.1893551 0.1972607 0.2244229    0
## knn  0.5205722 0.5597064 0.5631758 0.5638486 0.5751322 0.5855330    0
## rf   0.6966219 0.7143750 0.7230854 0.7196534 0.7267789 0.7296901    0
```

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model.

```
dotplot(results)
```



The results for just the RF model can be summarized. This gives a nice summary of what was used to train the model and the mean and standard deviation (SD) accuracy achieved.

```
print(fit.rf)
```

```
## Random Forest
##
## 13109 samples
##     6 predictor
##     6 classes: 'cramps', 'falling', 'running', 'sitting', 'standing', 'walking'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 11798, 11799, 11798, 11797, 11798, 11798, ...
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa
##   2     0.7783215  0.7196534
##   4     0.7694729  0.7085159
##   6     0.7613101  0.6982913
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

The RF was the most accurate model. Now we want to get an idea of the accuracy of the model on our validation set.This will give us an independent final check on the accuracy of the best model. We can run the LDA model directly on the validation set and summarize the results in a confusion matrix.

```
predictions <- predict(fit.rf, validation)
confusionMatrix(predictions, validation$ACTIVITY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction cramps falling running sitting standing walking
##   cramps      432     102      89      17       15       1
##   falling     150     513      29      83       11       0
##   running      67       3     189       0        3       0
##   sitting      28      91      13     377        4      19
##   standing     20       6      15       2      887       0
##   walking       1       2       2      21        1      80
##
## Overall Statistics
##
##                Accuracy : 0.7571
##                  95% CI : (0.742, 0.7717)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6928
##  Mcnemar's Test P-Value : 2.932e-08
##
## Statistics by Class:
##
##                      Class: cramps Class: falling Class: running
## Sensitivity                 0.6189         0.7155        0.56083
## Specificity                 0.9130         0.8932        0.97514
## Pos Pred Value              0.6585         0.6527        0.72137
## Neg Pred Value              0.8984         0.9180        0.95085
## Prevalence                  0.2133         0.2191        0.10296
## Detection Rate              0.1320         0.1567        0.05775
## Detection Prevalence        0.2004         0.2401        0.08005
## Balanced Accuracy           0.7660         0.8043        0.76798
##                      Class: sitting Class: standing Class: walking
## Sensitivity                  0.7540          0.9631        0.80000
## Specificity                  0.9441          0.9817        0.99149
## Pos Pred Value               0.7086          0.9538        0.74766
## Neg Pred Value               0.9551          0.9855        0.99368
## Prevalence                   0.1528          0.2814        0.03055
## Detection Rate               0.1152          0.2710        0.02444
## Detection Prevalence         0.1625          0.2841        0.03269
```

```
## Balanced Accuracy            0.8491          0.9724        0.89575
```

Conclusion

We have built a an algorithm to predict fall and movement type.

We have tested multiple methods to build models to predict and found that RF was the best one to use, it gives the best accuracy for our example.