Human Activity Recognition Using Smartphone Data

Farid Saifuddin

Submitted as part of edX HarvardX: PH125.9x Data Science: Capstone Course

1 Executive Summary

This project demonstrates the application of multiple Machine Learning algorithms for identification of human activities from smartphone sensors data. The algorithms are K-Nearest Neighbors, Quadratic Discriminant Analysis, Recursive Partitioning and Regression Trees (RPART) and Random Forest. The data are collected from two smartphone sensors: accelerometer and gyroscope and are time-series in nature. Six activities from 30 subjects were recorded. The data has been transformed into multiple domains and attributes as input for the algorithms. The overall accuracy ranges from 0.8 to 0.93. The Random Forest algorithm outperforms all other algorithms with overall accuracy of 0.93.

The project workflow includes data import, exploration and visualization, pre-processing, algorithm training, final algorithm modeling, validating the algorithm on test data set and result analysis. There are 561 features in the input data with 6 classes of activity to predict. The pre-processing managed to reduce the number of features to be less than 27% of original number of features. The **caret** package is utilized to train each algorithm using a subset of the training data to fine-tune the model parameters prior to defining the final model using the entire training data set. The validation results were analyzed and compared among each other by evaluating at the overall accuracy and class-specific sensitivity, specificity as well as balanced accuracy.

1.1 Data Source, Experiment, Transformation and Dimension

The dataset is downloaded from Kaggle dataset of Human Activity Recognition with Smartphones and is stored in one of project sub-folder. There are two sets of data, training and test data sets. The features in the data set are transformations of time-series data that was recorded on 30 subjects performing six activities (LAYING, SITTING, STANDING, WALKING, WALKING_DOWNSTAIRS, WALKING_UPSTAIRS) wearing a smartphone on their waist. The video of the experiment is available on this Youtube page. The training and test data are resulted from random partitioning of these 30 subjects with 70% portion allocated to training data (21 subjects) and the remaining 30% or 9 subjects are assigned as test data. This is interesting as the algorithm will have to be trained from the features of 21 persons in training data set group and to be validated with completely different persons in the test dataset.

The recorded time-series data are 3-axial linear acceleration from phone's accelerometer sensor and 3-axial angular velocity from the gyroscope sensor, all at a constant rate of 50Hz. A noise filter was applied and data were re-sampled with each sample contain a window of 2.56 second or 128 time-series points. The 2.56 second window is a fixed-width sliding window with 50% overlap to the next window.

Because machine learning algorithms do not work directly with time-series data, transformations were performed in each window. The first one is to separate acceleration data into gravity and body acceleration **(tBodyAcc-XYZ and tGravityAcc-XYZ)** by applying a low-pass filter. This is followed by time-derivation of body acceleration and angular velocity to obtain **tBodyAccJerk-XYZ and tBodyGyroJerk-XYZ**. The non-dimensional magnitude of these four outcomes were then calculated to produce **tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag**. All existing

time-domain data were later transformed into frequency domain using Fast Fourier Transform. At this point, eight 3-axial (24 signals) and nine magnitude data have been produced from the transformation, making a total of 33 signals.

The final features were resulted by estimating up to 17 statistical and or physical attributes on the 33 signals. The statistical attributes are *mean, standard deviation, median absolute deviation, maximum, minimum, signal magnitude area, energy, interquartile range, entropy, autoregression coefficients and correlation.* The physical properties are *index of the frequency component with largest magnitude, mean frequency, skewness of the frequency domain signal, kurtosis of the frequency domain signal, energy of a frequency interval and angle between two vectors.* Some physical attributes are applicable only to the frequency domain signals. There are 14 frequency intervals/bands which energy was estimated. These finally made 561 features.

There are 7352 rows in the training dataset. With 21 subjects and 6 activities, on average there are 58 records per person per activity.

1.2 Data Exploration and Visualization

The data distribution for each subject and activities in all training and test data set is as follows:

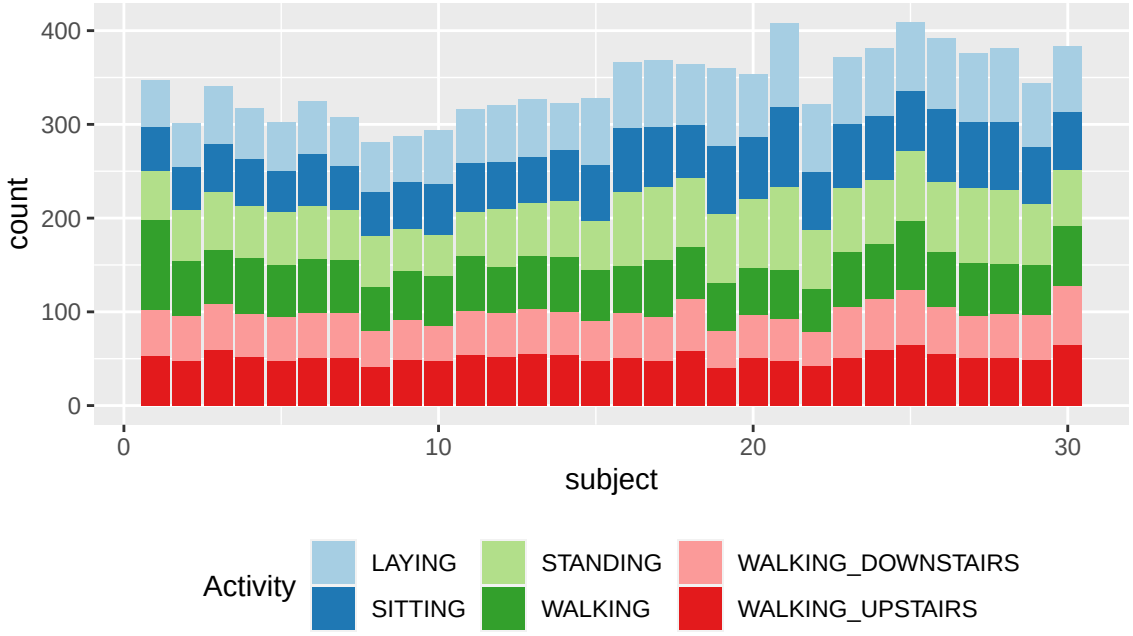

Figure 1: Data count for each subject with activity color code

Each subject has around 300 to 400 data (observations) in total with an average of more than 50 observations for each activity.

As described in the data description on Kaggle, the value for all 561 features are already standardized with data range of -1 to 1, as demonstrated by the following command:

```
range(train_set[,1:561])
```

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

The following plots provide some insights into some of the features in he training data set:

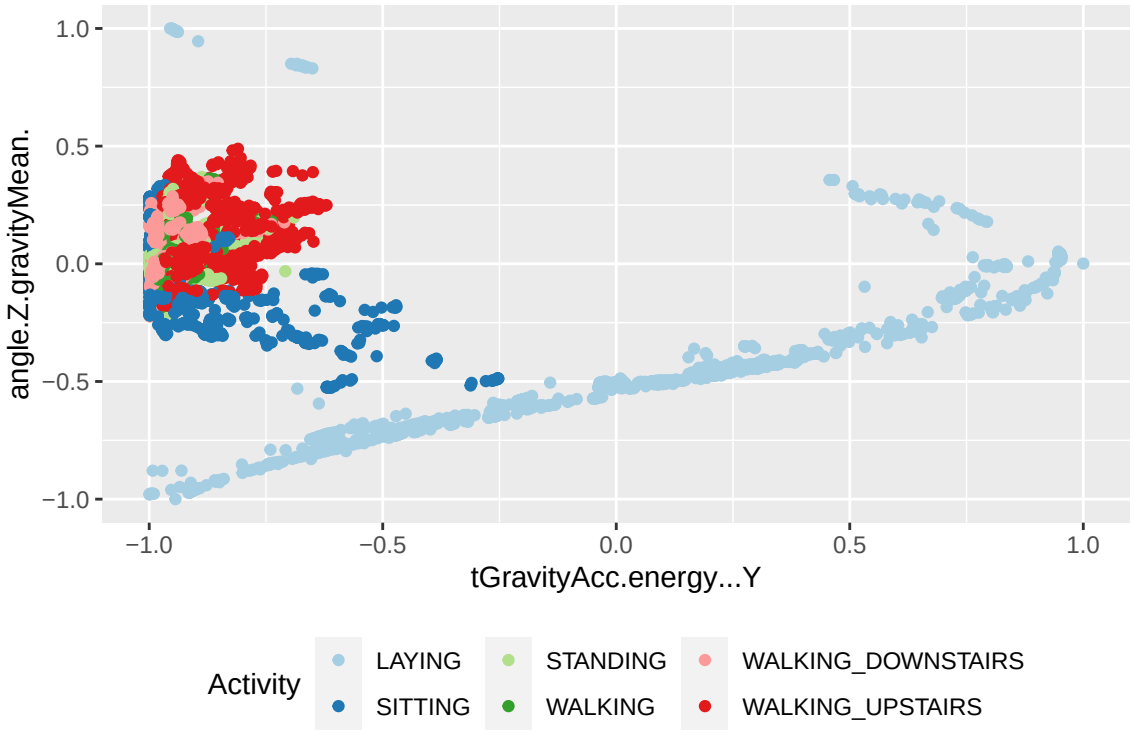

Figure 2: Plot of tGravityAcc-energy-Y vs. angle-Z-gravity-Mean

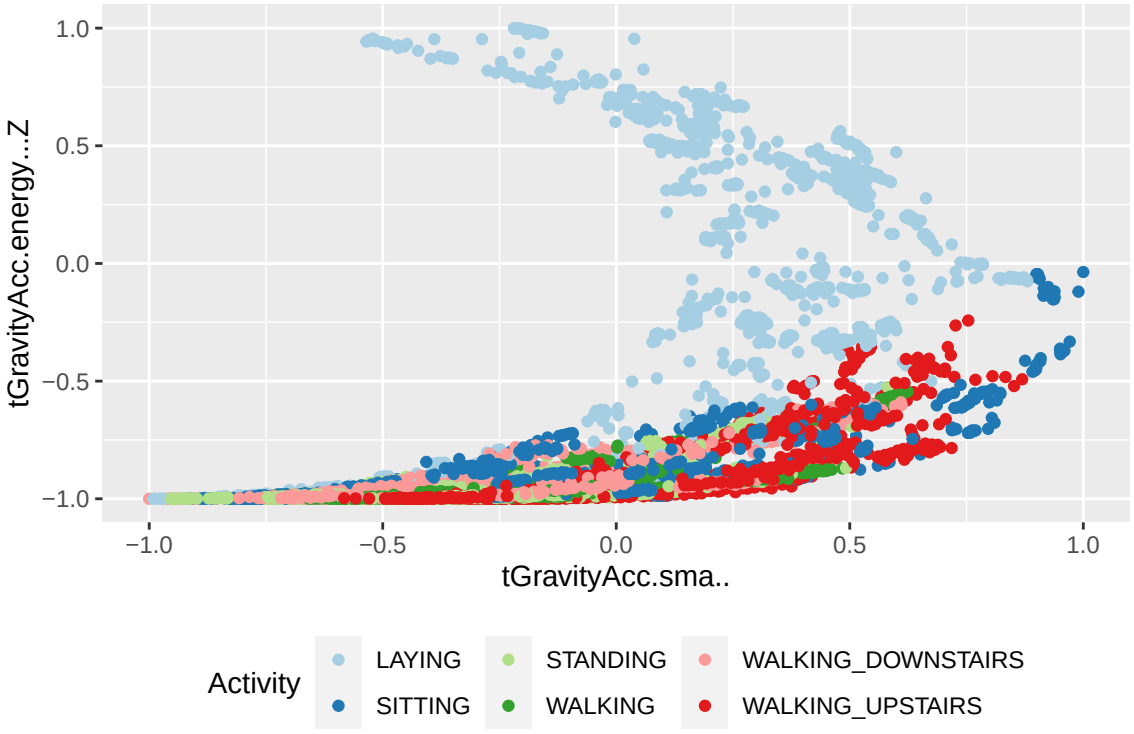

Figure 3: Plot of tGravityAcc-sma vs. tGravityAcc-energy-Z

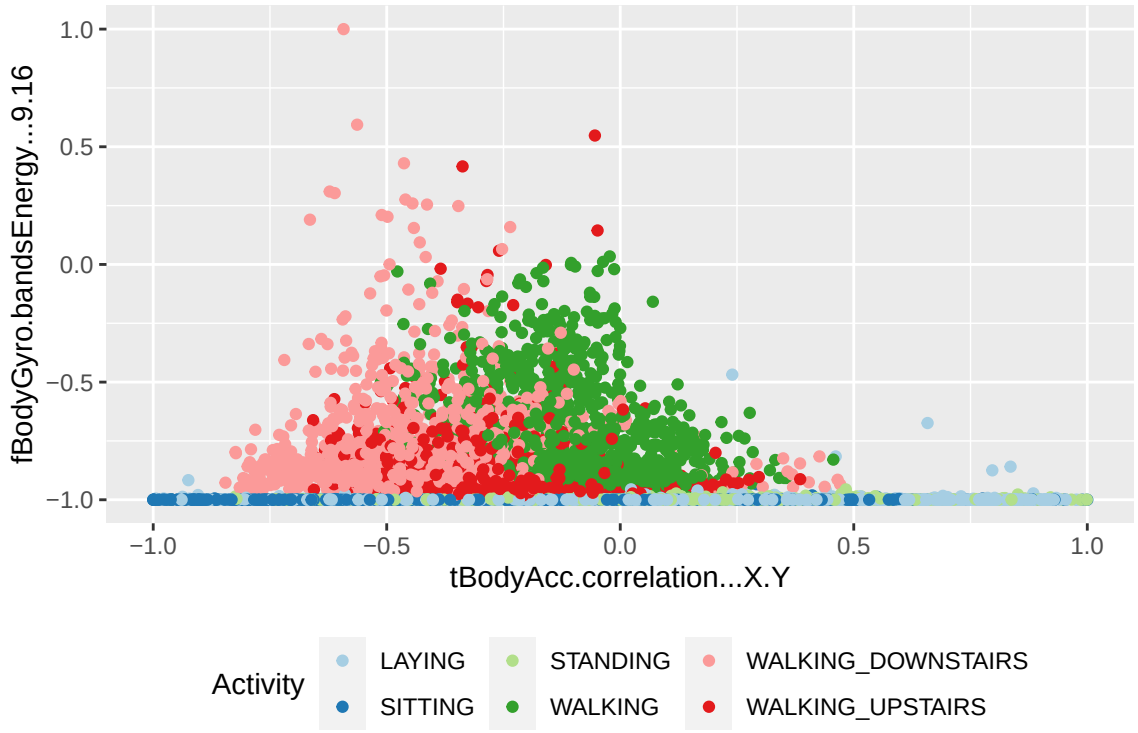

Figure 4: Plot of tBodyAcc-correlation-XY vs. fBodyGyro-bandsEnergy-9-16

The plots confirm the range of values to be between -1 and 1. While it is hard to pick which features that are able to provide visual insights to clearly separate the 6 activities, the 3 plots gave some valuable clues. In Figures 2 and 3, "Laying" is almost perfectly isolated from other activities. In all 3 plots, the 5 activities other than "Laying" are harder to isolate.

In Figure 4, all non moving activities ("Laying", "Sitting" and "Standing") are all at the bottom of the chart while the moving activities ("Walking", "Walking Downstairs", "Walking Upstairs") spread upward with apparent horizontal clustering introduced by correlation between body acceleration in X and Y axis. Walking in a flat surface has relatively highest correlation, while Walking Downstairs has lowest correlation and Walking Upstairs is in the middle. However, there are big overlaps between them.

As and additional information, the "t" in the beginning of the feature name denotes the feature is an attribute in time domain. When a feature name begins with "f", the feature is an attribute in frequency domain.

In the following figure we will visualize the distance between features in the order of activities. This will help us figure out how the features behave in general between the same and different activities. Because the computing time can be very long if we include all features and observations in the training set, the figure is generated only for randomly sampled 200 features sand 200 observations. The image provides three major dimensions with some striking distances in details within major groups. There are some uniformities as expected within each group of moving and non moving activities. However, we can not see clearly the boundary between six activities in the distance image:

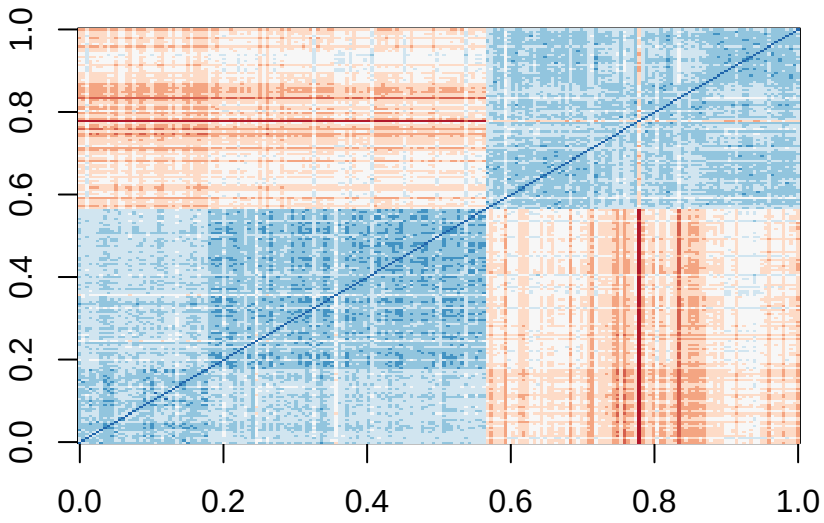

Figure 5: Image of distance for randomly selected 200 features and 200 observations in the training set

2 Methods and Analysis

2.1 Pre-processing

Value normalization and standardization were no longer necessary because the source dataset has been standardized. The first pre-processing step is to check if there is feature(s) with zero or near-zero variance to be removed. This is because features with zero or near-zero variance are not useful for prediction. The following code from **caret** package confirms there is no feature with zero or near-zero variance. The lowest variance has 34% unique value in the feature. In the following script, x is the predictor matrix for the observations in the training set:

```
# Detect and characterize zero and near-zero variance
nzv <- nearZeroVar(x, saveMetrics= TRUE)
# Count zero variance features
sum(nzv$zeroVar)
## [1] 0
# Count near-zero variance features
sum(nzv$nzv)
## [1] 0
# Investigate range of unique value percentage in the features
range(nzv$percentUnique)
## [1]   0.34 100.00
```

2.1.1 Removal of Highly Correlated Features

The 561 features are originated from 6 signals coming from 2 sensors (accelerometer & gyroscope). Transformations and attribute generations that finally produced 561 features may include some processes that their outputs are of high correlation, either positive or negative correlation. The highly correlated features will not give any prediction improvement. Hence, we will identify and remove any highly correlated feature and take only features that are not highly correlated for prediction.

First, a correlation matrix of the features is created:

```
xCor <- cor(as.matrix(x))
```

The summary of the correlation matrix shows that the correlation(s) can be as high as 1:

```
summary(xCor[upper.tri(xCor)])
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -1.00   -0.06    0.25    0.26    0.66    1.00
```

Function `findCorrelation` from **caret** package will give column indexes that has correlation equal or higher than the given cut-off. A cut-off value of 0.75 is selected :

```
highlyCorX <- findCorrelation(xCor, cutoff = .75)
```

And the features with correlation of less than 0.75 will be kept for the training and prediction. The number of feature is now only 150:

```
x_lowCor <- x[,-highlyCorX]
ncol(x_lowCor)
## [1] 150
```

A new training set is created with the features that have less than 0.75 (absolute) correlation:

```
x_lowCor <- x[,-highlyCorX]
ncol(x_lowCor)
## [1] 150
```

Similar feature removal is also applied to the test data, and a new test set data is created by removing the columns that are identified by `findCorrelation` command:

```
tx <- as.matrix(test_set[,1:561])
ty <- test_set$Activity
tx_lowCor <- tx[,-highlyCorX] # filter out features with cor > .75
ftest_set <- data.frame(ty, as.data.frame(tx_lowCor))
```

In conclusion, this pre-processing step has reduced the number of features by more than 70%, from 561 down to only 150 features. In the subsequent steps, we will train the algorithms using the newly created training set that only has 150 features. The final algorithm models will also be validated using the new test set that only has 150 features. This will enhance computing efficiency in each steps.

2.2 Algorithm Training Using The Caret Package

The advantage of training the algorithm using caret package is the ability to optimize parameters to be used for certain algorithm (tuning parameters) using the `train` function. Each algorithm has it's own tuning parameters. The final selected model that maximize the accuracy will be stored under model's `bestTune` component that we can evaluate. The parameter tuning process within `train` function automatically uses cross-validation to come up with selection of optimum parameters. Caret has default values to evaluate for each algorithm. It also allows us to provide customized values in the `tuneGrid` argument. Caret also allow us to customize the cross-validation parameter using `trainControl` function that is taken in the `trControl` argument of `train` function. We can visually inspect the result of parameter tuning using `ggplot`function.

In addition to customizing `train` arguments, to prevent longer computing time, we will not use all the training data in the parameter tuning stage. This is because if, for example, we use 10-fold cross-validation and 10 values in tuning parameters, `caret` will perform 100 times iteration. Instead, we can randomly select a certain numbers of observations from the training data to be used in the `train` function.

2.3 K-Nearest Neighbors

The k-nearest neighbors algorithm works by evaluating the distance between all observations based on the features. The k parameter is the number of nearest neighbors from the point to be predicted that will control the prediction. This is the parameter to be tuned in the `train` function of `caret` package with k as `seq(1, 30, 2)`. Only randomly selected 1000 observation to be used for training with 5 cross-validation to decide the optimum k.

The maximum accuracy is reached at k equal 7:

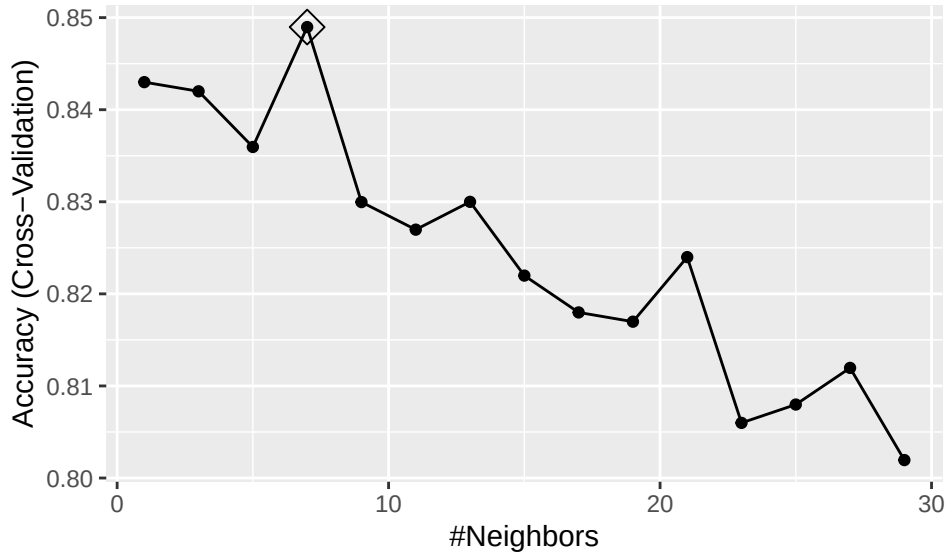

Figure 6: Parameter tuning for kNN

The resulted k number is then passed on to `knn3` function to get the final knn model from the entire training set. Once the final model is obtained, the algorithm is then validated by making prediction on the test data set. The confusion matrix is calculated following the prediction. The kNN algorithm gave an overall accuracy of 0.87.

2.4 Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) is a version of Naive Bayes generative model with assumption that the conditional distributions $p_{X|Y=1}(x)$ and $p_{X|Y=0}(x)$ are multivariate normal. For each class (activity) QDA needs to calculate the average and standard deviation for each feature and the correlation between each features. With 6 classes and 150 predictors the QDA estimates 900 means and standard deviations plus 6.7×10^4 correlations. The reduction of features from 561 down to 150 helps reducing the number of parameters calculated in QDA.

To visualize how QDA partition works, the following figure is a QDA partition plot on two features displayed in Figure 1:

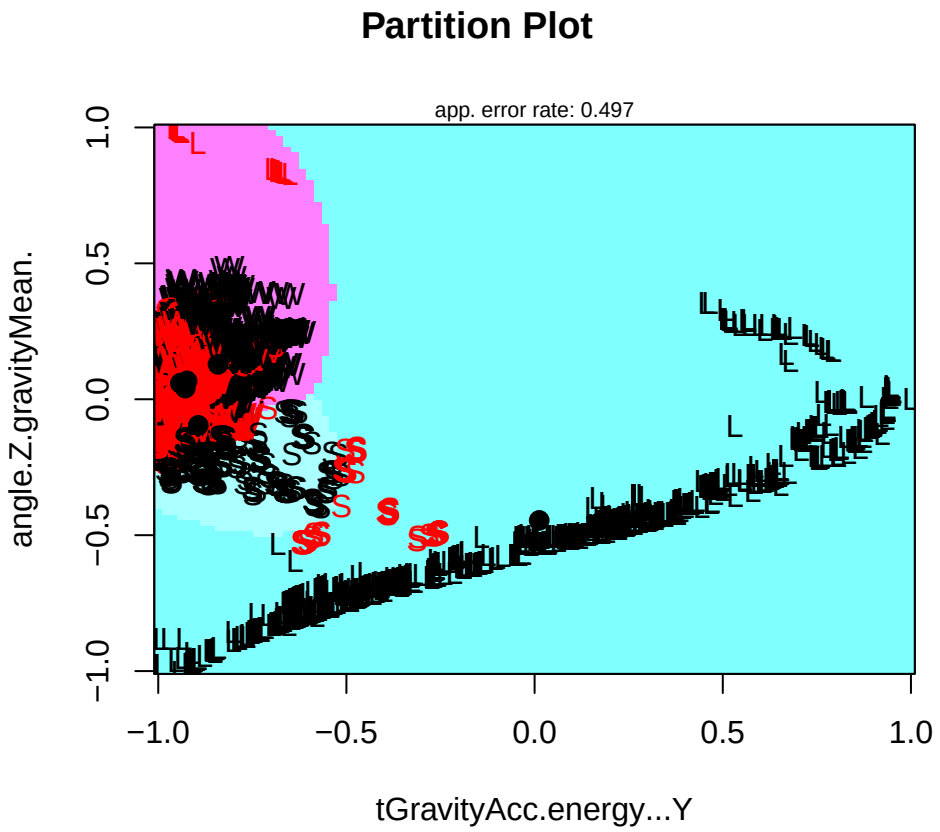

Figure 7: QDA partition plot

The `train` function in caret package does not provide tuning parameter for QDA. Hence, we will use the output QDA model from the `train` function as final model. The entire training data set will be used with 5 cross-validation. The overall accuracy from the validation using the test set is 0.91.

2.5 Classification and Regression Trees (RPART)

Recursive Partitioning and Regression Trees (RPART) works by building a decision tree. The algorithm starts with one partition and the regression trees recursively create partitions to grow the decision tree. So it starts with the entire features, define two new partition by finding a predictor j and value s. These two new partitions are called $R_1(j, s)$ and $R_2(j, s)$ that splits the observations in the current partition by evaluating if x_j is bigger than s:

$$R_1(j, s) = \{\mathbf{X} \mid x_j < s\} \text{ and } R_2(j, s) = \{\mathbf{X} \mid x_j \geq s\}$$

The j and s are picked by finding the pair that minimizes the residual sum of square (RSS):

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

The `train` function will fine-tune the complexity parameter *cp* which is the minimum value for how much residual sum of squares (RSS) must improve to grow another partition. This is done using `rpart` method. The *cp* values to be evaluated is from 0 to 0.02 with 0.001 increment. Only 2000 randomly selected observation from the training data set were used for these parameter tuning. For complexity parameter *cp*, the maximum accuracy is reached at *cp* equal 0.003:

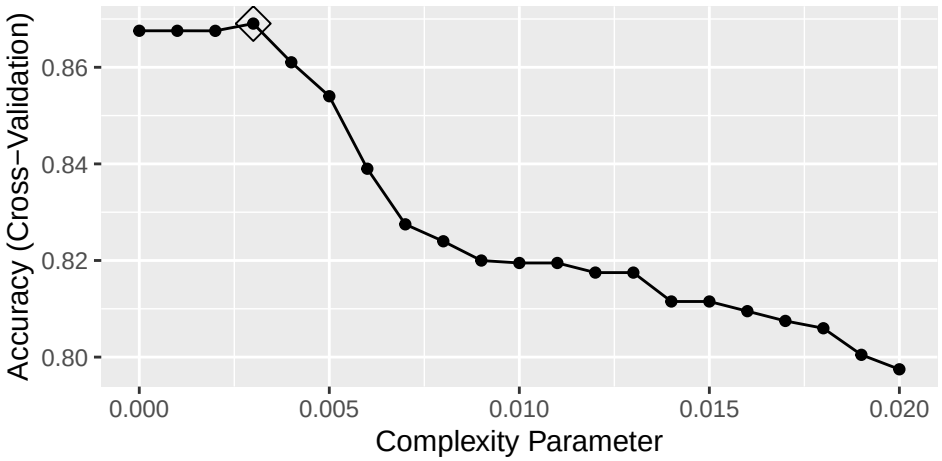

Figure 8: Parameter tuning for RPART complexity parameter

Not only *cp* that can be used to decide to make more partition. Another parameter is the minimum number of observations required in a partition before making next partition. This argument is called minsplit with default value = 20.

Another parameter that can be fine-tuned is maximum tree depth using `rpart2` method. Hence, we performed second step training for RPART maximum depth with range from 5 to 30 with 5 increment. The accuracy is best at maxdepth = 15:

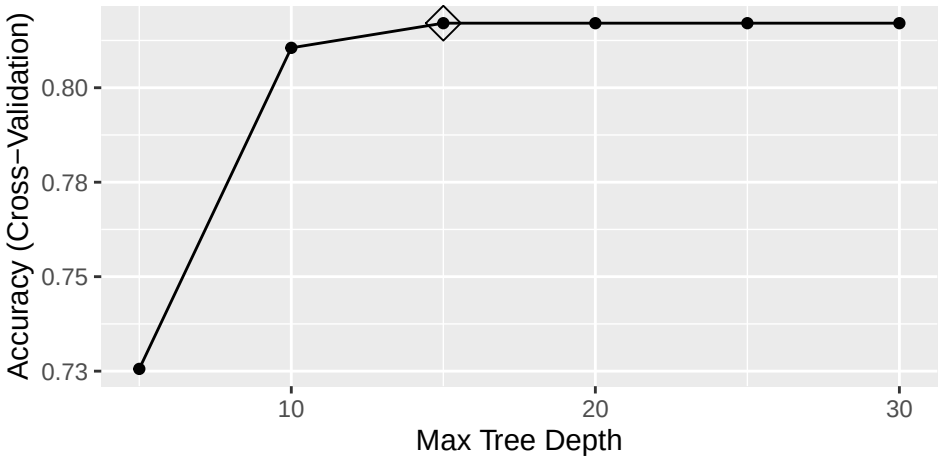

Figure 9: Parameter tuning for RPART maximum tree depth

The fine-tuned parameters were then passed on to build the final model for the algorithm using the entire training dataset. The final model tree can be displayed as follows:

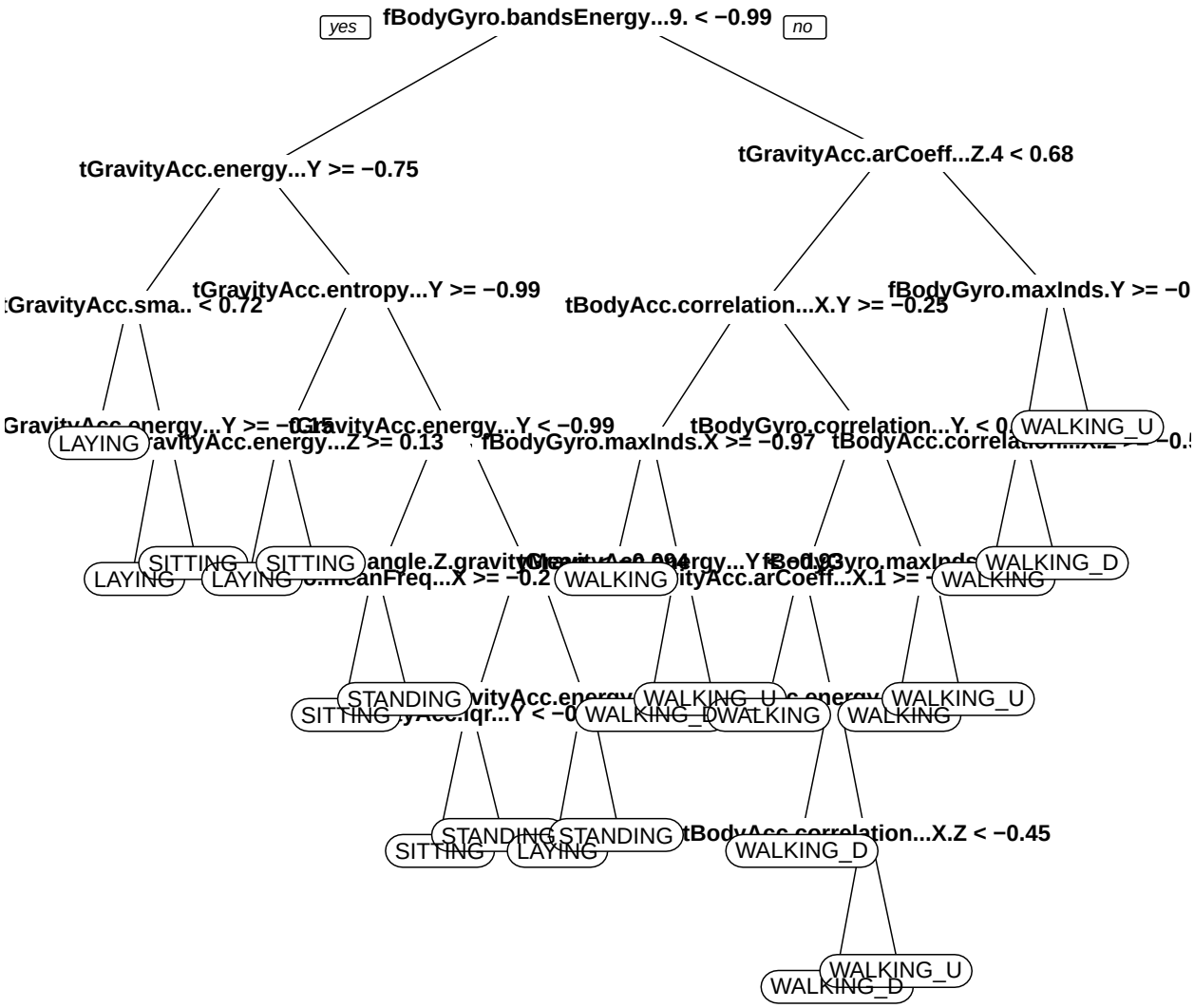

Figure 10: Decision tree for RPART final model

Unfortunately, the length of feature names prevents us to have readable labels especially in the bottom part of the decision tree. Nonetheless, we can identify which feature splits first and which class came to terminal nodes first. The beauty of the decision tree is this interpretability aspect as it is very similar to daily practice of decision tree in human life.

However, this algorithm is hard to train and often under-performs compared to other algorithms. The overall accuracy from the algorithm validation on the test dataset is 0.8.

2.6 Random Forest

Random Forest approach to classification is by constructing multiple decision trees with randomness and perform final classification by voting from the forest of the constructed decision trees. The randomness in individual tree construction is created by sampling the observations in the training set with replacement, or by randomly selecting a portion of features to be included in the building of each tree.

The number of features in each tree can be tuned with `train` function in the caret package as `mtry` parameter to get maximum accuracy. For this training, only 2000 randomly selected observations will be used. Here is the tuning result of `mtry` that gave maximum accuracy at 40:

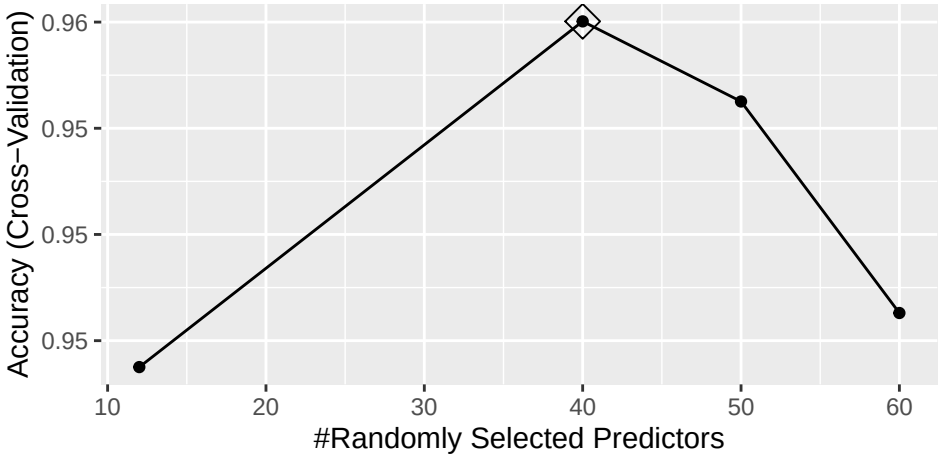

Figure 11: Random forest mtry parameter tuning

Another parameter we can optimize is `nodesize` which is the minimum size of terminal nodes in one tree. However, this has to be trained separately using `rborist` package. The minimum node size will also have impact on the maximum node size and the tree depth. Here is the tuning result that shows maximum accuracy at minimum node size of 1:

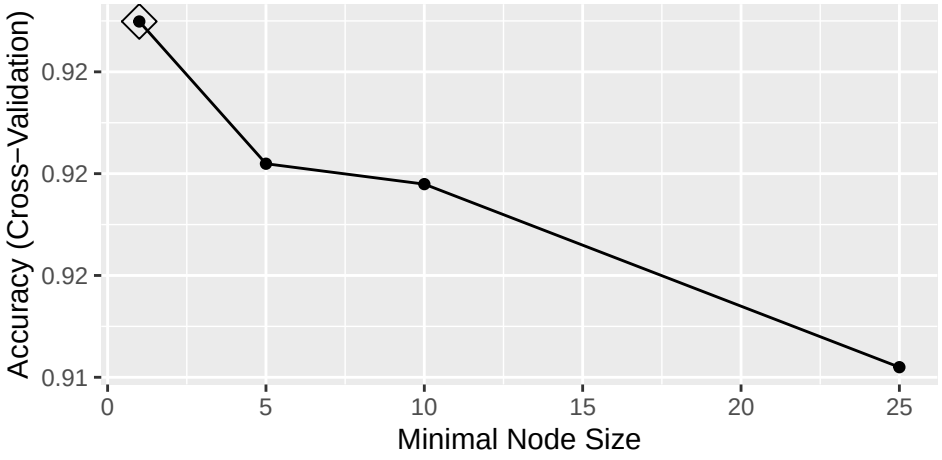

Figure 12: Random forest nodesize parameter tuning

The two tuned parameters were then passed on to develop the final Random Forest model using the entire training set. The default number of trees in the `randomForest` function is 500. We can check the error rate against the number of trees in the final model:

Figure 13: Error rate with number of trees

In Figure 13, apart from the 6 classes, there is another plot for OOB (out of bag). This is a measure for those samples that are not included in a tree.

The `randomForest` function also assessed the importance of each feature by calculating the mean decrease in accuracy and Gini over specific and overall classes when a feature is excluded in a tree. The overall variable importance of the final model in both decrease of accuracy and Gini is shown below, limited to top-10 variables only:

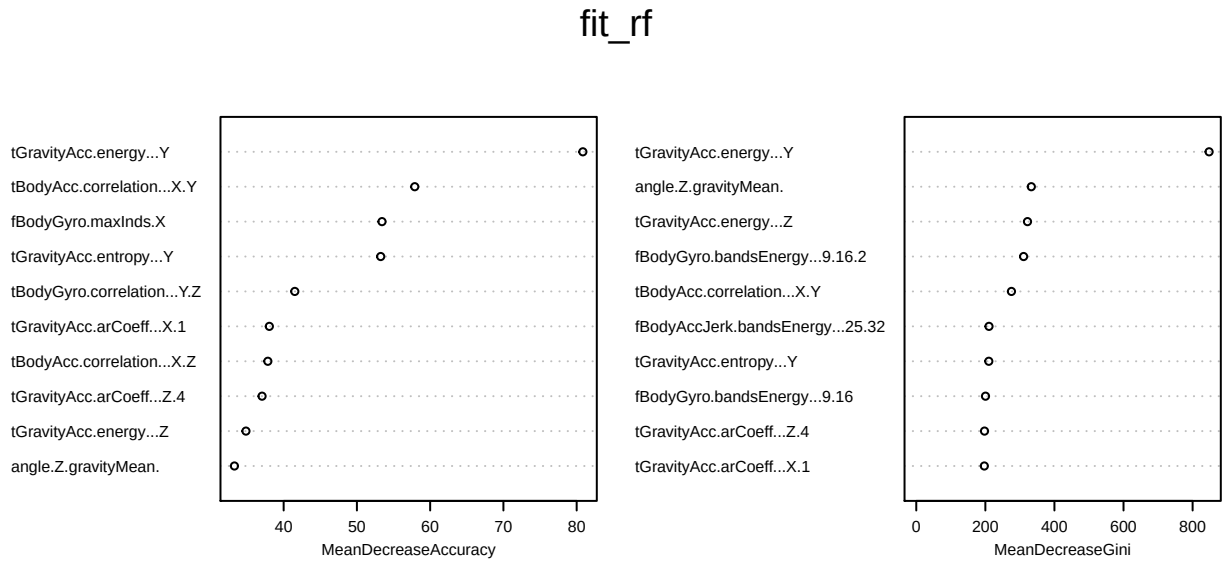

Figure 14: Random Forest final model variable importance (Top 10)

The overall accuracy from the algorithm validation on the test dataset is 0.93.

3 Results and Discussion

We will compare and discuss the result of all four algorithms in solving the same problem using the same data: recognizing human activities from smartphone sensors signals. This will include the overall accuracy, sensitivity, specificity and balanced accuracy. When necessary, confusion matrix tables are evaluated to allow more detailed evaluation.

3.1 Overall Accuracy

The overall accuracy is simply defined as the overall proportion that is predicted correctly, in this case refers to prediction to all six classes of activities in the data set. Overall accuracy for all four methods are presented in the following chart:

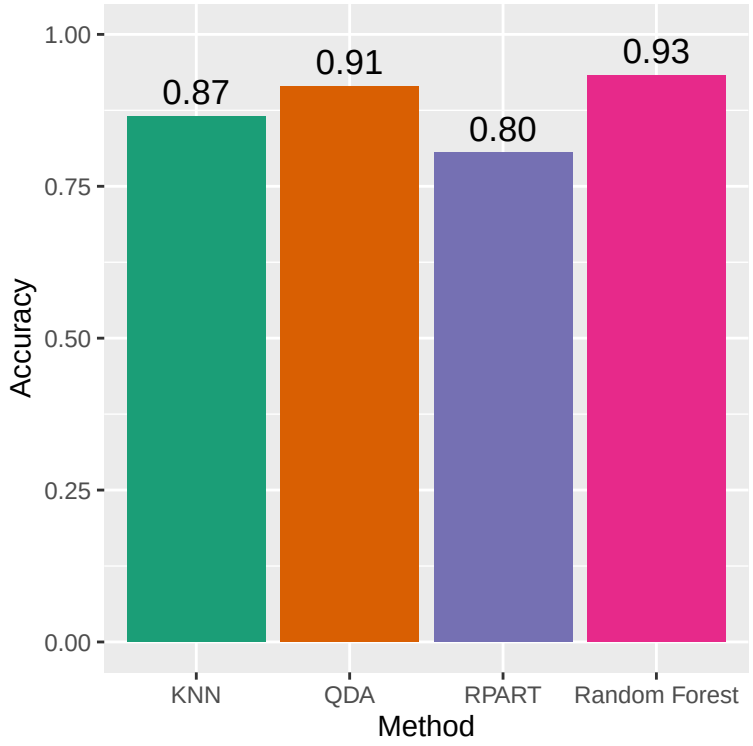

Figure 15: Overall accuracy for each algorithm

As expected, the Random Forest method outperforms all other methods in predicting the activity type from the smartphone sensors data. The Recursive Partitioning and Regression Trees (RPART) performance is the lowest among the four methods despite it's high-interpretability. The QDA performance that was trained without parameter tuning is surprisingly close to the Random Forest. The KNN performance represents the median accuracy value among the four. To understand further, we will evaluate the sensitivity, specificity and balanced accuracy for each class.

3.2 Sensitivity

Sensitivity is the ability of an algorithm to predict a positive outcome when the actual outcome is positive: $\hat{Y} = 1$ when $Y = 1$ or proportion of the Actual Positive cases that are predicted Positive. Here is the sensitivity chart for each class by all 4 algorithms:

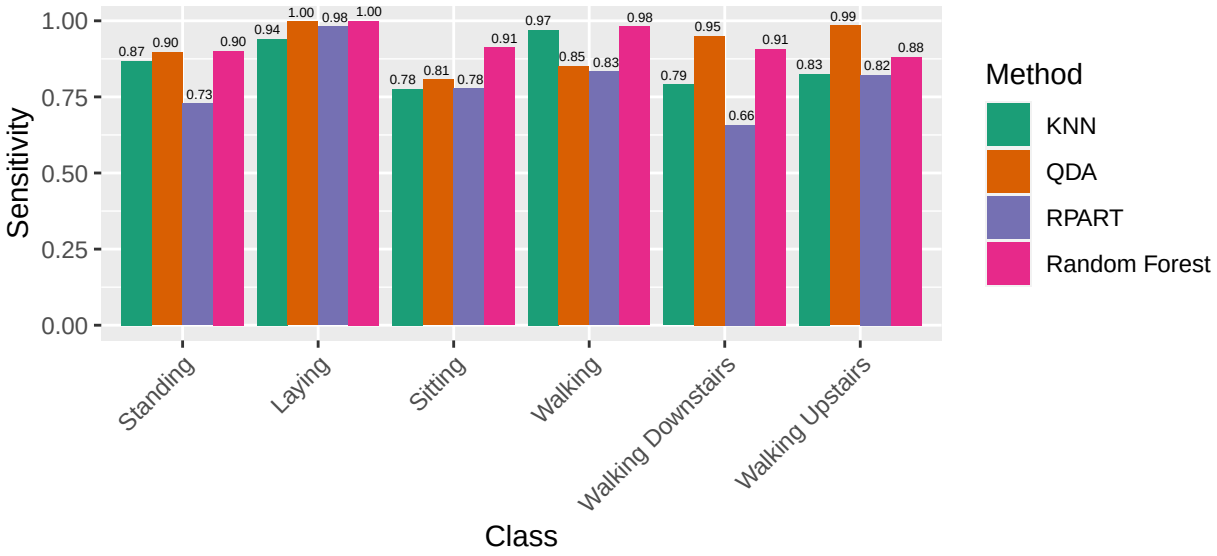

Figure 16: Sensitivity for each class and algorithm

From the sensitivity chart, we see that all algorithm doing perfect or almost perfect job in recognizing "Laying" activity. With exception of Random Forest, most algorithm gave relatively low sensitivity for identifying "Sitting" activity. The sensitivity has more variation in other activities with RPART mostly performs the lowest. Surprisingly, QDA outperforms Random Forest in recognizing both non-flat walking. The following confusion matrix tables for QDA and Random Forest provide more detail on the miss-match between the actual and predicted activities:

Table 1: QDA confusion matrix table

Predicted	Reference					
	Laying	Sitting	Standing	Walking	W_Downstairs	W_Upstairs
Laying	536	1	0	0	0	0
Sitting	1	396	29	0	0	0
Standing	0	87	477	0	0	0
Walking	0	0	5	422	1	1
W_Downstairs	0	3	6	63	399	6
W_Upstairs	0	4	15	11	20	464

Table 2: Random Forest confusion matrix table

Predicted	Reference					
	Laying	Sitting	Standing	Walking	W_Downstairs	W_Upstairs
Laying	537	1	0	0	0	0
Sitting	0	448	49	0	0	0
Standing	0	41	479	0	0	0
Walking	0	0	1	487	6	34
W_Downstairs	0	0	0	6	381	22
W_Upstairs	0	1	3	3	33	415

By comparing both tables, we can see that when the actual activity is *walking downstairs*, QDA made 21 false predictions: 1 as *walking* and 20 as *walking upstairs*. This is much better compared to Random Forest that made 39 false prediction, 6 as *walking* and 33 as *walking upstairs*.

In the actual case of *walking upstairs*, which QDA made 0.99 sensitivity compared to 0.88 for Random Forest, QDA only made 7 false prediction(1 as *walking* and 6 as *walking downstairs*). On the other hand, Random Forest made 56 false prediction, eight times larger than QDA, with 34 as *walking* and 22 as *walking downstairs*.

In a grouped activities of *walking* and *non walking*, both QDA and Random Forest exhibit similar behavior. When actual activity group is *non walking*, both algorithm made false predictions to as *walking* type of activity. However, when the actual cases are *walking* type of activities, they both never made false prediction as *non-walking* type of activities.

3.3 Specificity

Specificity is the proportion of Actual Negative Cases that are called Negative, or the ability of an algorithm to predict a negative outcome when that actual case is negative. Here is the specificity chart:

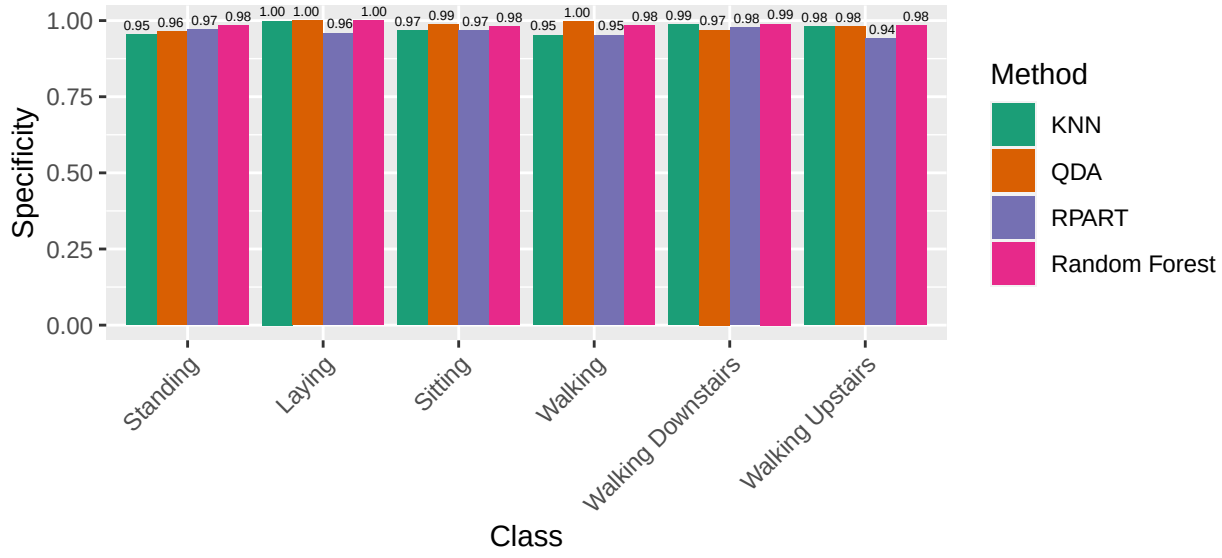

Figure 17: Specificity for each class and algorithm

In general, the numbers are higher than the sensitivity numbers. This is partly caused by the total data counts involved in specificity calculation. Each of six activities has approximately similar prevalence in the data set i.e one sixth of the total observations for each. The sensitivity refers to proportion of this one sixth. On the other hand, for specificity, the proportion is referred to the rest, or the five sixth of the entire observations (2455) in the test data set.

The lowest specificity is 0.94 performed by RPART for actual *not walking upstairs* activity. Or, 6 percent out of 471 number of activities other than *walking upstairs*, were falsely predicted as *walking upstairs* by RPART algorithm. For actual activity of *laying*, three out of four algorithm did perfect specificity: they correctly predicted as *not laying* when the actual activity is *not laying*.

It is interesting to compare between QDA and Random Forest that have highest overall accuracy and sensitivities in most activities. They have tied performance for specificity in two activities: *laying* (specificity = 1) and *walking upstairs* (specificity = 0.98). QDA performed better than Random Forest for *sitting* (0.99 vs. 0.98) and *walking* (1 vs. 0.98). Random Forest performed better for the remaining 2 activities: *standing* (0.98 vs. 0.96) and *walking downstairs* (0.99 vs. 0.97).

3.4 Balanced Accuracy

Balanced accuracy is defined as the average of specificity and sensitivity. This is particularly useful when the prevalence in the observations are not balanced and overall accuracy is highly influenced by high or low prevalence. The balanced accuracy chart for all algorithms in each activities is shown below:

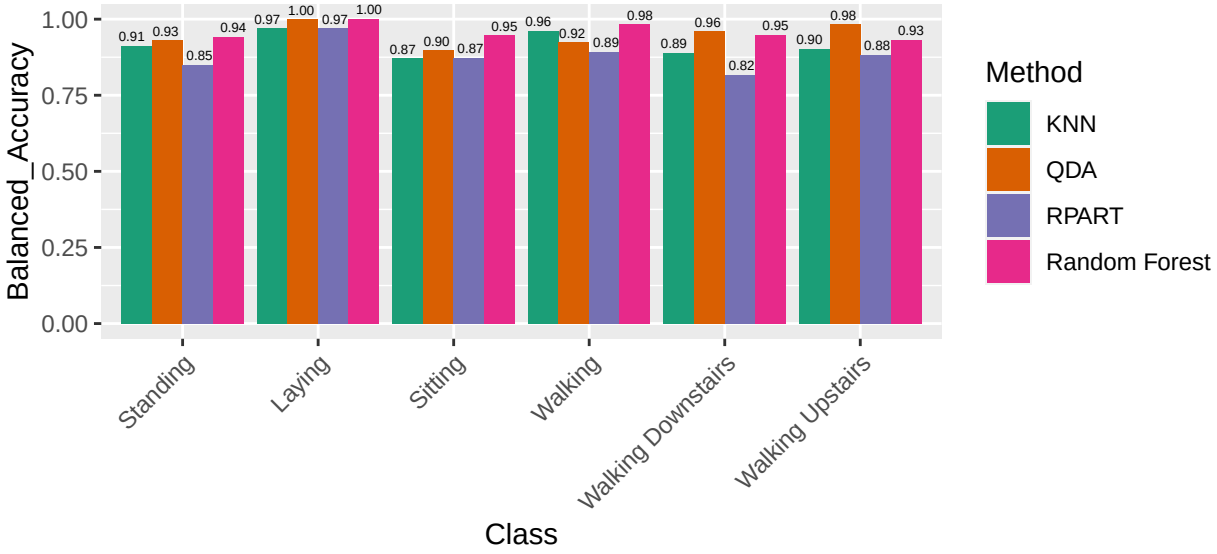

Figure 18: Balanced accuracy for each class and algorithm

The balanced accuracy here has similar weight between sensitivity and specificity. The 4 algorithms have highest balanced accuracy in *Laying* activity. When comparing between QDA and Random Forest, in balanced accuracy, they have tied performance in *laying* activity. QDA performed better in *walking upstairs* and *downstairs*, and Random Forest beat QDA for the rest: *standing, sitting* and *walking.*

3.5 Consideration for Better Tuning and Training

While most of algorithm perform well, better training and parameter tuning may further improve their performance. With limited computing capability, training and tuning often take long computing time even with limited observation samples in this study. With more powerful computer, Random Forest can be trained more with more parameter tuning in order to achieve better performance. Hyper-parameter tuning may be explored to get the most optimum combination of parameters that gives highest accuracy using cross-validation with training dataset.

Alternative QDA algorithm package(s) in R may be evaluated, especially those equipped with parameter tuning in order to further explore the performance of this algorithm that in some cases beat Random Forest.

Another possible way to improve performance for all algorithm is during data pre-processing stage by setting higher correlation threshold in eliminating features with high correlation. This study uses 0.75 absolute correlation as threshold. This can be modified to 0.90 and improvement of performances for each algorithm may be evaluated with the higher threshold and hence more features involved.

In this application, the data set was partitioned on the basis of the experiment subject. The algorithms learn from set of signals of a group of persons and applied / validated the learning using signals from completely different people. This is a good advantage where we can prove that the learning is not personal. It would be interesting to test the algorithm with personal learning and validation where the data for learning and validation come from one person doing repeated activities.

3.6 Consideration for Application

Selection of algorithm may depend on it's application or level of complexity in real-life problem. If the challenge is only to detect movement, then we may look into the algorithm that give highest grouped sensitivity and specificity. On the other hand, when activities to be recognized involves multiple different moving and non moving activities, multiple algorithm may be involved including making an ensemble to enhance the desired accuracy.

4 Conclusion and Way Forward

Four machine learning algorithms have been successfully applied to recognize 6 different activities using pre-processed time-series signals from smartphone sensors attached to the subjects. There are 561 features as a result of signal transformation and feature estimation in time and frequency domain. A data pre-processing was performed to eliminate highly-correlated features. Using 0.75 absolute correlation as a threshold, only 150 features were used for for developing machine learning algorithm model and validation. The overall accuracy range is between 0.80 to 0.93. The two highest performing algorithms are Random Forest and Quadratic Discriminant Analysis. Recursive Partitioning and Regression Trees (RPART) gave the lowest overall accuracy (0.80) and k-Nearest Neighbor performs quite well with 0.87 overall accuracy.

Better algorithm training with more powerful computing devices and alternative R models and packages inside `caret` package could probably improve the performance of the algorithms. Application of the algorithm can be further adjusted depending on the objectives of the application in real-life world.