Classifying physical activity using wearable sensors

The ability to classify physical activity in free-living individuals is of great scientific interest to the life sciences. Wearable technologies, relying predominantly on accelerometer output, have been used extensively in research environments; the Sensewear Armbands and the IDEEA activity monitoring system are two notable and extinct examples. The appeal of wearable technologies is in their objectivity, they are not plagued by the bias which renders self-report questionnaires close to useless.

The challenge with wearable devices lies in translating raw sensor outputs to accurate and precise measurements with biological and behavioural meaning. This challenge is not easily overcome and traditional linear models are limited in their ability to capture the variability of human movement. Machine learning approaches, which are advanced statistical techniques able to capture more complex patterns in data, are showing great promise for the assessment of physical activity.

The purpose of this post is to explore the accuracy of popular machine learning algorithms for the classification of physical activity sensor outputs.

```
#packages
library(caret)
library(GGally)
library(class)
library(data.table)
library(rpart)
library(rpart.plot)
library(party)
library(rattle)
library(xgboost)
library(formattable)
library(dplyr)
library(tidyr)
library(tibble)
library(ggthemes)
library(randomForest)
library(MASS)
library(GeNetIt)
library(sjPlot)
```

```
#STEP 1: read in all data
setwd("~/Desktop/GitHub/Data-science-projects/Classification of physical activity /MHEALTHDATASET")

files <- list.files(pattern = "*.log", full.names = T)## Specify files using list.files.
f <- lapply(files, fread)## Read data into a list using data.table::fread
for(i in seq_along(f)) {

    f[[i]]$ID <- rep(files[[i]], nrow(f[[i]])) ##Add the participant number to df

}
f <- bind_rows(f)# Bind rows together to have a data.frame with an ID column
```

Dataset

The MHEALTH (Mobile HEALTH) dataset (Banos et al., 2015) comprises body motion and physiological recordings for volunteers performing physical activities. The dataset is available in full at the UCI machine learning repository.

The study recruited 10 participants and each participant performed the following tasks:

1: Standing still (1 min) 2: Sitting and relaxing (1 min) 3: Lying down (1 min) 4: Walking (1 min) 5: Climbing stairs (1 min) 6: Waist bends forward (20x) 7: Frontal elevation of arms (20x) 8: Knees bending (crouching) (20x) 9: Cycling (1 min) 10: Jogging (1 min) 11: Running (1 min) 12: Jump front & back (20x) Sensors located on the chest, right wrist and left ankle were used to measure the acceleration, the rate of turn and the magnetic field orientation during the 12 activities. The sensor positioned on the chest provided 2-lead electrocardiogram measurements. The variables in the dataset are as follows:

Column 1–3: acceleration from the chest sensor (X, Y, Z axis) Column 4 & 5: electrocardiogram signal (lead 1 + 2) Column 6–8: acceleration from the left-ankle sensor (X, Y, Z axis) Column 9–11: gyro from the left-ankle sensor (X, Y, Z axis) Column 13–14: magnetometer from the left-ankle sensor (X, Y, Z axis) Column 15–17: acceleration from the right-lower-arm sensor (X, Y, Z axis) Column 18–20: gyro from the right-lower-arm sensor (X, Y, Z axis) Column 21–23: magnetometer from the right-lower-arm sensor (X, Y, Z axis) Column 24: Label of activity

```
#rename collumns
cn<-c("acceleration_chest_X", "acceleration_chest_Y", "acceleration_chest_Z", "ECG_1", "ECG_2",
"acceleration_ankle_X", "acceleration_ankle_Y", "acceleration_ankle_Z", "gyro_ankle_X",
"gyro_ankle_Y", "gyro_ankle_Z", "magnetometer_ankle_X", "magnetometer_ankle_Y", "magnetometer_ankle_Z",
"acceleration_right_lower_arm_X","acceleration_right_lower_arm_Y","acceleration_right_lower_arm_Z",
"gyro_right_lower_arm_X","gyro_right_lower_arm_Y","gyro_right_lower_arm_Z","magnetometer_right_lower_arm_X",
"magnetometer_right_lower_arm_Y","magnetometer_right_lower_arm_Z","Label",
"ID")
names(f)[1:25]<-cn

f<- droplevels(f[f$Label!="0",])
f$Label<-as.factor(f$Label)
```

After the removal of null values and some cleaning of the dataframe, data are split into a training and testing data set at a 70:30 ratio.

```
set.seed(150)
sample <- floor(0.7 * nrow(f))
train_ind <- sample(seq_len(nrow(f)), size = sample)
train <- f[train_ind, ]
test <- f[-train_ind, ]
train$ID<-NULL
test$ID<-NULL
```

Here's the structure of the data:

```
str(train)
```

```
## Classes 'data.table' and 'data.frame':   240236 obs. of  24 variables:
##  $ acceleration_chest_X          : num  -9.97 -11.89 -7.28 -9.02 -9.83 ...
##  $ acceleration_chest_Y          : num  0.648 -0.138 0.165 -1.293 1.448 ...
##  $ acceleration_chest_Z          : num  3.01 -1.08 -2.52 2.1 -1.06 ...
##  $ ECG_1                         : num  0 -0.2177 -0.1172 0.0126 -0.2386 ...
##  $ ECG_2                         : num  -0.02093 -0.00837 -0.12559 -0.15489 -0.12977 ...
##  $ acceleration_ankle_X          : num  2.326 3.906 -0.224 3.946 -2.086 ...
##  $ acceleration_ankle_Y          : num  -9.66 -18.85 -10.62 -9.04 -8.15 ...
##  $ acceleration_ankle_Z          : num  0.54 -11.91 -2.38 -3.16 -2.51 ...
##  $ gyro_ankle_X                  : num  -0.347 0.395 -0.458 0.805 0.453 ...
##  $ gyro_ankle_Y                  : num  -0.809 -0.531 -0.737 -0.563 -0.809 ...
##  $ gyro_ankle_Z                  : num  0.4813 0.6287 -0.5894 -0.0118 -0.3752 ...
##  $ magnetometer_ankle_X          : num  0.723 135.87 -26.667 -30.364 -27.551 ...
##  $ magnetometer_ankle_Y          : num  1.277 -0.441 5.221 -15.293 -17.89 ...
##  $ magnetometer_ankle_Z          : num  -0.0272 -10.217 -7.5997 -4.7232 -8.2167 ...
##  $ acceleration_right_lower_arm_X: num  -5.63 -1.2 -1.43 -2.49 -4.04 ...
##  $ acceleration_right_lower_arm_Y: num  -1.56 -7.56 -4.96 -8.96 -14.32 ...
##  $ acceleration_right_lower_arm_Z: num  8.661 1.684 3.992 0.521 4.7 ...
##  $ gyro_right_lower_arm_X        : num  -0.796 -0.176 -0.469 -0.237 -0.333 ...
##  $ gyro_right_lower_arm_Y        : num  0.571 -0.554 -0.93 -0.727 -0.702 ...
##  $ gyro_right_lower_arm_Z        : num  0.5 0.976 -0.228 0.838 0.815 ...
##  $ magnetometer_right_lower_arm_X: num  -60.44 21.22 3.33 -8.3 4.35 ...
##  $ magnetometer_right_lower_arm_Y: num  24.21 -14.21 9.78 -4.04 4.4 ...
##  $ magnetometer_right_lower_arm_Z: num  -53.44 34.5 19.46 -13.12 -8.98 ...
##  $ Label                         : Factor w/ 12 levels "1","2","3","4",..: 7 4 8 5 4 10 9 6 11 7 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(test)
```

```
## Classes 'data.table' and 'data.frame':   102959 obs. of  24 variables:
##  $ acceleration_chest_X          : num  -9.77 -9.86 -9.69 -9.69 -9.62 ...
##  $ acceleration_chest_Y          : num  0.2788 0.1156 0.2106 0.0756 -0.0138 ...
##  $ acceleration_chest_Z          : num  0.73 0.8 0.566 0.933 0.851 ...
##  $ ECG_1                         : num  -0.0251 0.0251 0 0.7493 -0.2177 ...
##  $ ECG_2                         : num  -0.0251 0.0167 -0.0209 0.046 -0.1758 ...
##  $ acceleration_ankle_X          : num  2.42 2.39 2.63 2.4 2.45 ...
##  $ acceleration_ankle_Y          : num  -9.53 -9.6 -9.54 -9.61 -9.39 ...
##  $ acceleration_ankle_Z          : num  0.402 0.481 0.372 0.596 0.472 ...
##  $ gyro_ankle_X                  : num  -0.21 -0.2 -0.2 -0.202 -0.202 ...
##  $ gyro_ankle_Y                  : num  -0.889 -0.869 -0.869 -0.88 -0.88 ...
##  $ gyro_ankle_Z                  : num  -0.509 -0.507 -0.507 -0.491 -0.491 ...
##  $ magnetometer_ankle_X          : num  0.568 0.211 0.744 0.383 0.202 ...
##  $ magnetometer_ankle_Y          : num  0.912 0.548 1.093 0.547 0.365 ...
##  $ magnetometer_ankle_Z          : num  -0.886 -1.02 -0.747 -0.735 -0.729 ...
##  $ acceleration_right_lower_arm_X: num  -2.99 -2.88 -2.82 -2.92 -2.85 ...
##  $ acceleration_right_lower_arm_Y: num  -9.2 -9.19 -9.12 -9.19 -9.11 ...
##  $ acceleration_right_lower_arm_Z: num  1.52 1.55 1.86 1.52 1.75 ...
##  $ gyro_right_lower_arm_X        : num  -0.0588 -0.0588 -0.0784 -0.0784 -0.0686 ...
##  $ gyro_right_lower_arm_Y        : num  -0.934 -0.934 -0.934 -0.934 -0.932 ...
##  $ gyro_right_lower_arm_Z        : num  -0.345 -0.345 -0.341 -0.341 -0.347 ...
##  $ magnetometer_right_lower_arm_X: num  0.72 0.355 0.359 0.722 0.357 ...
##  $ magnetometer_right_lower_arm_Y: num  0.17803 -0.37003 -0.00713 0.35226 -0.18858 ...
##  $ magnetometer_right_lower_arm_Z: num  0.374 -0.35 -0.354 -0.35 -0.352 ...
##  $ Label                         : Factor w/ 12 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can see that's a fair amount of data. It's important to see if any of the variables have little predictive power, so they can be removed to limit the computing load. The nearZeroVar function is useful to identify predictors with zero variance or very few unique values. We can see the output below and it looks like all of the columns are likely to assist in classification.

```
#first we must remove the features with little predictive power
tt <- nearZeroVar(train, saveMetrics = T)

tt
```

```
##                                freqRatio percentUnique zeroVar   nzv
## acceleration_chest_X            1.047619  41.292312559   FALSE FALSE
## acceleration_chest_Y            1.055556  61.832115087   FALSE FALSE
## acceleration_chest_Z            1.062500  62.983066651   FALSE FALSE
## ECG_1                           1.056725   1.109742087   FALSE FALSE
## ECG_2                           1.032511   1.239614379   FALSE FALSE
## acceleration_ankle_X            1.094595  56.396626650   FALSE FALSE
## acceleration_ankle_Y            1.772947  28.930718127   FALSE FALSE
## acceleration_ankle_Z            1.305556  61.404202534   FALSE FALSE
## gyro_ankle_X                    1.046667   0.546129639   FALSE FALSE
## gyro_ankle_Y                    1.045420   0.618142160   FALSE FALSE
## gyro_ankle_Z                    1.035941   0.585257830   FALSE FALSE
## magnetometer_ankle_X            1.130575  49.855974958   FALSE FALSE
## magnetometer_ankle_Y            1.130575  52.847616510   FALSE FALSE
## magnetometer_ankle_Z            1.130575  51.547644816   FALSE FALSE
## acceleration_right_lower_arm_X  1.157895  53.129006477   FALSE FALSE
## acceleration_right_lower_arm_Y  1.250000  41.717727568   FALSE FALSE
## acceleration_right_lower_arm_Z  2.209302  44.363875522   FALSE FALSE
## gyro_right_lower_arm_X          1.340675   0.481193493   FALSE FALSE
## gyro_right_lower_arm_Y          1.026316   0.589836661   FALSE FALSE
## gyro_right_lower_arm_Z          1.033987   0.436653957   FALSE FALSE
## magnetometer_right_lower_arm_X  1.147914  57.778601042   FALSE FALSE
## magnetometer_right_lower_arm_Y  1.147914  53.776702909   FALSE FALSE
## magnetometer_right_lower_arm_Z  1.147914  56.845352070   FALSE FALSE
## Label                           1.004124   0.004995088   FALSE FALSE
```

Classification tree

We'll start with probably the most simple classification algorithm. A classification tree analyses the training data and constructs a set of rules, or questions, which can ultimately be used to arrive at a predicted classification. It may be likened to a flowchart, where each 'decision node' brings you closer to the final decision, termed a 'leaf node'. The classification tree splits at each decision node using the Gini index, which can be broadly thought of as the probability of mislabeling an element in each class.

```r
#DECISION TREE
DT1 <- rpart(Label ~ ., data = train, method = 'class')

#test DT
DTTEST <- predict(DT1, test, type = 'class')
DTRES <- confusionMatrix(DTTEST, test$Label)
```

The constructed tree did not perform particularly well. The overall accuracy was 62.5%.

```r
DTRES$overall[1]
```

```
##  Accuracy 
## 0.6257539
```

We can demonstrate the class specific accuracy (or lack of) with a confusion matrix; this shows the classification errors across all classes of activity.

```r
DTRES
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5    6    7    8    9   10   11   12
##         1  8701  103    0  257  181  805 1107  672    2  144   29   44
##         2     9 6328    0   13  146   53 1372    9    2  932  749  261
##         3     0    1 9219    0    0    0    0    0    0   24  206   52
##         4   163    0    0 5496 1786  515  140 1265   18  801  300  148
##         5    89    2    0  897 4451  441   19 1662   23   30    1   27
##         6    90    0    0 1449  569 5418   11  802   43 1312  591  479
##         7     0  933    0  624  268   79 5478   43  158  129  140   83
##         8     0  933    0  193  871 1105    0 3570  449  546  305  128
##         9     0    0    0  126  223   56   41  796 8463  723 1216  507
##         10    0    0    0   15   80    1    0   96    1 1866  388  582
##         11    0  928    7  135  566    0  741    5   13 2673 5437  780
##         12    0    0    0    0    0    0    0    0    0    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.6258
##                  95% CI : (0.6228, 0.6287)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5896
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.96122  0.68574  0.99924  0.59707  0.48693  0.63944
## Specificity           0.96439  0.96217  0.99698  0.94522  0.96599  0.94342
## Pos Pred Value        0.72237  0.64088  0.97022  0.51693  0.58244  0.50334
## Neg Pred Value        0.99614  0.96885  0.99993  0.95983  0.95080  0.96686
## Prevalence            0.08792  0.08963  0.08961  0.08940  0.08878  0.08229
## Detection Rate        0.08451  0.06146  0.08954  0.05338  0.04323  0.05262
## Detection Prevalence  0.11699  0.09590  0.09229  0.10326  0.07422  0.10455
## Balanced Accuracy     0.96281  0.82395  0.99811  0.77114  0.72646  0.79143
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.61488  0.40022  0.92270   0.20327   0.58075
## Specificity           0.97388  0.95183  0.96068   0.98760   0.93752
## Pos Pred Value        0.69036  0.44074  0.69649   0.61604   0.48179
## Neg Pred Value        0.96389  0.94360  0.99219   0.92681   0.95719
## Prevalence            0.08653  0.08664  0.08908   0.08916   0.09093
## Detection Rate        0.05321  0.03467  0.08220   0.01812   0.05281
## Detection Prevalence  0.07707  0.07867  0.11802   0.02942   0.10961
## Balanced Accuracy     0.79438  0.67603  0.94169   0.59543   0.75914
##                      Class: 12
## Sensitivity            0.00000
## Specificity            1.00000
## Pos Pred Value             NaN
## Neg Pred Value         0.96998
## Prevalence             0.03002
## Detection Rate         0.00000
## Detection Prevalence   0.00000
## Balanced Accuracy      0.50000
```

Classification trees generally do not match the predictive power of more advanced algorithms, their top down approach is ‘greedy’ at each node, which limits the predictive accuracy of the overall decision tree. By aggregating many decision trees, the predictive power can be substantially improved.

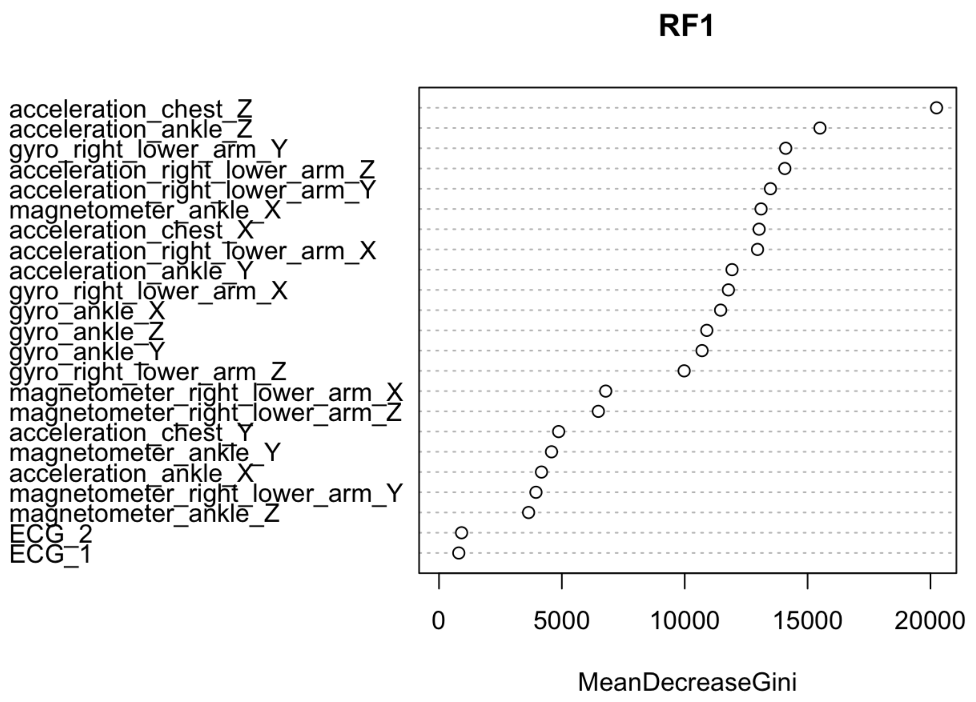
Random forest

Random Forest is another popular tree based machine learning classifier. It utilises many decision trees trained on bootstrapped samples from training data. Each of the trees in the forest is trained on a random sample of predictors and observations, hence the name! The number of predictors is user defined but recommended to be close to the square root of the number of predictor variables and each tree considers a limited number of observations. Whilst ignoring data might seem illogical, it has the effect of decorrelating the trees in the forest and therefore reduces the variance in the overall model.

```
#rf
train$Label<-as.factor(train$Label)
RF1 <- randomForest(Label ~ ., ntree = 501, mtry = 5, data = train)
#testing the Rf
RFTEST <- predict(RF1, test)
test$predition <- predict(RF1, test)
```

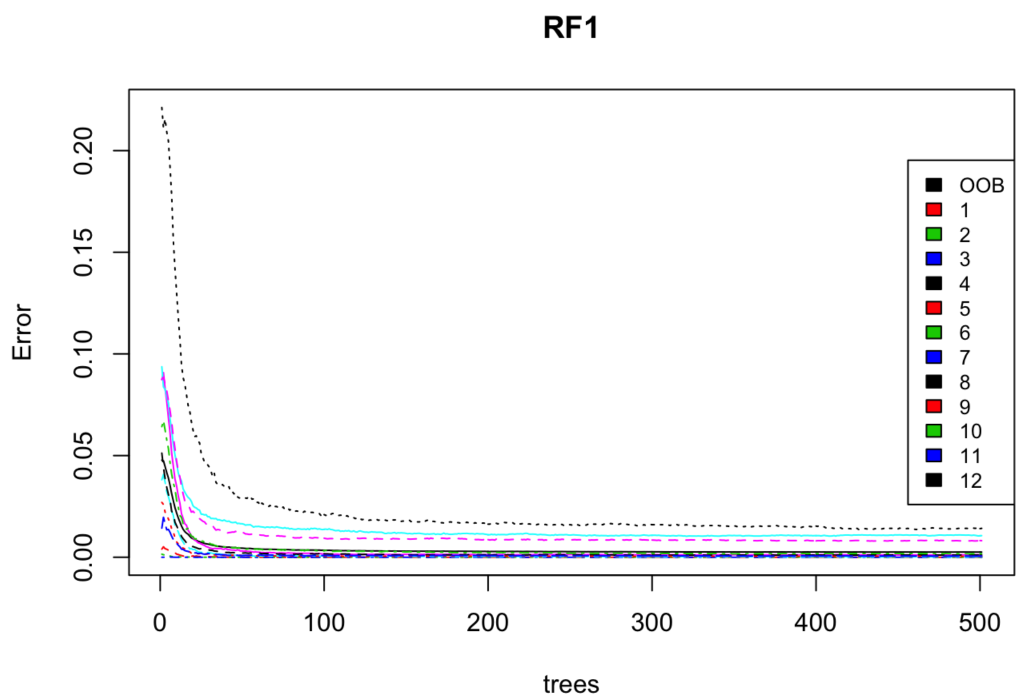
We can see that the random forest's node purity is most influenced by the Z axis acceleration at the chest and ankle and ECG data is the least influential.

```
varImpPlot(RF1, type = 2)
```



The plot below details the error rate for each of the 12 physical activity categories plotted against the number of trees.

```
plot(RF1, type="l")
legend("right", colnames(RF1$err.rate),col=1:4,cex=0.8,fill=1:4)
```



The power of the random forest algorithm is clear, classifying nearly 100% of observations correctly in the test data set.

```
forestResults <- confusionMatrix(RFTEST,test$Label)
forestResults$overall[1]
```

```
## Accuracy
## 0.997669
```

```
forestResults
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5    6    7    8    9   10   11   12
##         1  9052    0    0    0    0    1    0    0    0    0    0    0
##         2     0 9228    0    0    0    0    0    0    0    0    0    0
##         3     0    0 9226    0    0    0    0    0    0    0    0    0
##         4     0    0    0 9203    6    0    0    1    0    0    0    1
##         5     0    0    0    2 9131    3    0    4    1    0    0    5
##         6     0    0    0    0    0 8463    1    2    0    0    0    0
##         7     0    0    0    0    0    5 8908    5    0    0    0    0
##         8     0    0    0    0    3    1    0 8908    2    0    0    0
##         9     0    0    0    0    0    0    0    0 9169    0    0    0
##         10    0    0    0    0    1    0    0    0    0 9098   84   11
##         11    0    0    0    0    0    0    0    0    0   82 9277   18
##         12    0    0    0    0    0    0    0    0    0    0    1 3056
##
## Overall Statistics
##
##                Accuracy : 0.9977
##                  95% CI : (0.9974, 0.998)
##     No Information Rate : 0.0909
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           1.00000  1.00000  1.00000  0.99978  0.99891  0.99882
## Specificity           0.99999  1.00000  1.00000  0.99991  0.99984  0.99997
## Pos Pred Value        0.99989  1.00000  1.00000  0.99913  0.99836  0.99965
## Neg Pred Value        1.00000  1.00000  1.00000  0.99998  0.99989  0.99989
## Prevalence            0.08792  0.08963  0.08961  0.08940  0.08878  0.08229
## Detection Rate        0.08792  0.08963  0.08961  0.08939  0.08869  0.08220
## Detection Prevalence  0.08793  0.08963  0.08961  0.08946  0.08883  0.08223
## Balanced Accuracy     0.99999  1.00000  1.00000  0.99985  0.99937  0.99939
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.99989  0.99865  0.99967   0.99107   0.99092
## Specificity           0.99989  0.99994  1.00000   0.99898   0.99893
## Pos Pred Value        0.99888  0.99933  1.00000   0.98956   0.98934
## Neg Pred Value        0.99999  0.99987  0.99997   0.99913   0.99909
## Prevalence            0.08653  0.08664  0.08908   0.08916   0.09093
## Detection Rate        0.08652  0.08652  0.08905   0.08837   0.09010
## Detection Prevalence  0.08662  0.08658  0.08905   0.08930   0.09108
## Balanced Accuracy     0.99989  0.99930  0.99984   0.99502   0.99493
##                      Class: 12
## Sensitivity            0.98868
## Specificity            0.99999
## Pos Pred Value         0.99967
## Neg Pred Value         0.99965
## Prevalence             0.03002
## Detection Rate         0.02968
## Detection Prevalence   0.02969
## Balanced Accuracy      0.99433
```

K-nearest neighbours

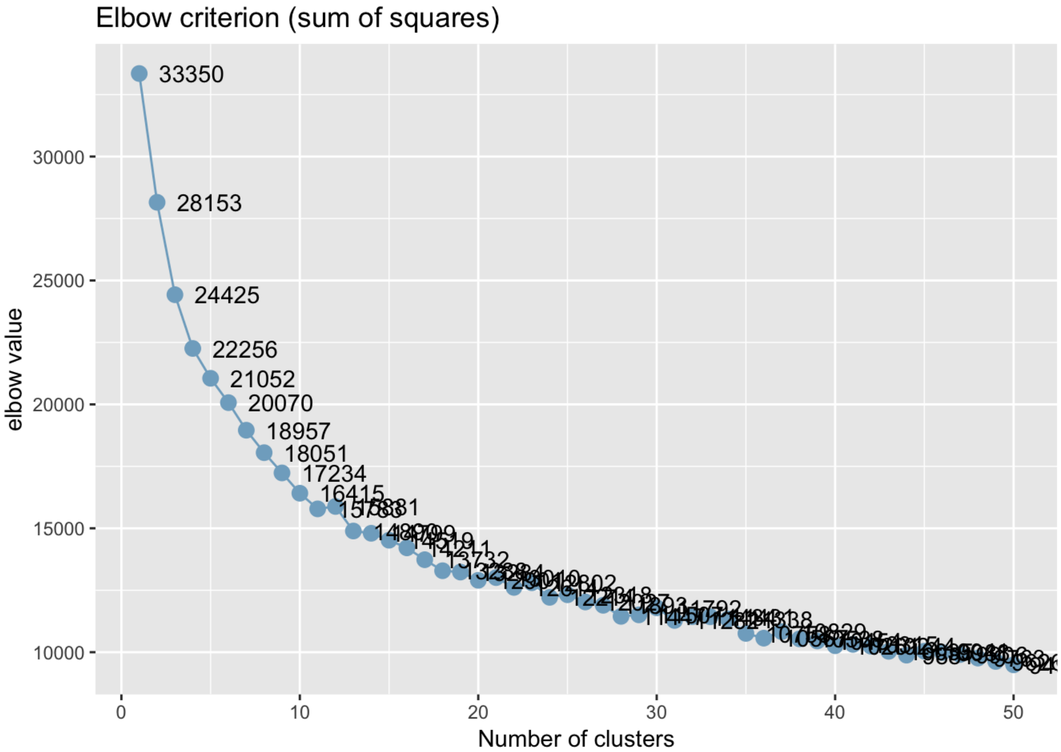
The final algorithm is K-Nearest Neighbours (KNN). The KNN algorithm classifies any given data point based on similarity to its neighbouring points, measured by euclidian distance. An important consideration for KNN is the number of neighbours considered for classification (k). It is also important to normalise the data for KNN, as outlined below.

```
#knn
#normalise
normfunc<-function(x){
    return((x-min(x))/(max(x) - min(x)))
}
train<-train[1:102959,]
trainlabel<-train$Label
testlabel<-test$Label

#normalise data
train<-as.data.frame(lapply(train[,1:23], normfunc))
train<-as.data.frame(lapply(train[,1:23], as.numeric))
test<-as.data.frame(lapply(test[,1:23], normfunc))
test<-as.data.frame(lapply(test[,1:23], as.numeric))
```

To determine the optimal k values I used the sjc.elbow function to create a plot of the error rate against number of clusters. The plot below seems to indicate that the elbow value is approximately 10.

```
#what is optimal k
sjc.elbow(train, steps = 50)
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```



Now we fit the KNN model:

```
m1<-knn(train = train, test = test, cl = trainlabel, k=10)
```

The accuracy of the KNN algorithm is comparable to the random forest, resulting in 96% classification accuracy. Please note the size of the training data was reduced to be exactly the same size as the test set (n=102,959)

```
cmat<-table(testlabel, m1)
cc<-confusionMatrix(cmat)
cc$overall[1]
```

```
##  Accuracy
## 0.9853048
```

```
cc
```

```
## Confusion Matrix and Statistics
##
##          m1
## testlabel    1    2    3    4    5    6    7    8    9   10   11   12
##        1  9052    0    0    0    0    0    0    0    0    0    0    0
##        2     0 9228    0    0    0    0    0    0    0    0    0    0
##        3     0    0 9226    0    0    0    0    0    0    0    0    0
##        4     5    0    0 9193    2    3    1    0    0    0    1    0
##        5    15    6    0  194 8734   27   30  129    4    2    0    0
##        6    15    0    0    0    1 8414   41    2    0    0    0    0
##        7     5    0    0    0    0   31 8849   24    0    0    0    0
##        8     2    0    0    2    4   14   24 8874    0    0    0    0
##        9     0    0    0    0    0    0    0    4 9168    0    0    0
##        10    0    0    0   14    1    0    2    1    0 9051  110    1
##        11    0    0    0    9    0    0    0    0    0  406 8942    5
##        12    0    1    0   48   28    5    8    7    0  219   60 2715
##
## Overall Statistics
##
##                Accuracy : 0.9853
##                  95% CI : (0.9846, 0.986)
##     No Information Rate : 0.094
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9839
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity           0.99538  0.99924  1.00000  0.97178  0.99590  0.99058
## Specificity           1.00000  1.00000  1.00000  0.99987  0.99568  0.99938
## Pos Pred Value        1.00000  1.00000  1.00000  0.99870  0.95548  0.99304
## Neg Pred Value        0.99955  0.99993  1.00000  0.99715  0.99962  0.99915
## Prevalence            0.08833  0.08970  0.08961  0.09188  0.08518  0.08250
## Detection Rate        0.08792  0.08963  0.08961  0.08929  0.08483  0.08172
## Detection Prevalence  0.08792  0.08963  0.08961  0.08940  0.08878  0.08229
## Balanced Accuracy     0.99769  0.99962  1.00000  0.98582  0.99579  0.99498
##                      Class: 7 Class: 8 Class: 9 Class: 10 Class: 11
## Sensitivity           0.98816  0.98153  0.99956   0.93521   0.98124
## Specificity           0.99936  0.99951  0.99996   0.99862   0.99552
## Pos Pred Value        0.99327  0.99484  0.99956   0.98595   0.95514
## Neg Pred Value        0.99887  0.99822  0.99996   0.99331   0.99817
## Prevalence            0.08698  0.08781  0.08908   0.09400   0.08851
## Detection Rate        0.08595  0.08619  0.08905   0.08791   0.08685
## Detection Prevalence  0.08653  0.08664  0.08908   0.08916   0.09093
## Balanced Accuracy     0.99376  0.99052  0.99976   0.96692   0.98838
##                      Class: 12
## Sensitivity            0.99779
## Specificity            0.99625
## Pos Pred Value         0.87836
## Neg Pred Value         0.99994
## Prevalence             0.02643
## Detection Rate         0.02637
## Detection Prevalence   0.03002
## Balanced Accuracy      0.99702
```