HarvardX: PH125.9x: Data Science: Capstone: Predicting the Chance of a Patient Falling Using Data from China

A Capstone Project for the `Professional Certificate in Data Science` offered by Harvard University (HarvardX) via EdX

John King'athia Karuitha

Wednesday December 23, 2020

Contents

Abstract

In this project, I use data from China to predict whether or not a patient will fall. In hospitals, falls by patients can be fatal and it is paramount to have an idea which patient is likely or unlikely to fall so as to take appropariate interventions. The dependent variable consists of a range of patient activities among them falling. The independent variables capture a range of physiological conditions such as heart rate and blood flow. The independent variables are highly correlated and hence, I start by performing principal components analysis (PCA) on the data. I then run 5 models- classification tree, random forest, K-Nearest Neighbours (KNN), Extreme gradient boosting (XGBoost), and multinomial logit. Finally, I assemble these models into one predictive algorithm (the ensemble). The random forest model offers the best specificity (0.8859) while extreme gradient boosting has the highest sensitivity (0.5437). The ensembled algorithm offers the best balanced accuracy (0.7048).

Introduction

In hospitals, predicting that a patient will fall goes a long way in ensuring their well being. Seriously ill and elderly patients could easily die if they attempt to take a walk but fall. In this exercise, I use a redacted dataset from China to predict the chance that a patient will fall[1]. I sourced the data from Kagle. I downloaded the data and loaded it in my github account. From the outset, it important to note that it is much more expensive to predict that a patient will NOT fall who ends up falling. It is much more acceptable to predict a patient will fall and they do NOT fall. From this perspective, maximizing specificity of the model is paramount in the choice of the right model.

Note that I partly adopt the Tidymodels framework in building the machine learning model. Tidymodels is a suit of packages for building machine learning models developed by R-Studio. `Max Kuhn`, the developer of `caret`, the other popular machine learning platform in R is the lead scientist in developing Tidymodels. Like the tidyverse, Tidymodels provides a consistent API for predictive modelling. Tidymodels has a smoother, more intuitive work-flow compared to caret and saves the data analyst of having to keep repeating data preprocessing steps.

The exercise proceeds as follows. In the next section, I download and load the required R packages and then present a summary of the final results. I then load and describe the data. I then explain the method of analysis, and then do the analysis. I then discuss the results and conclude.

Objective of the Exercise

The objective of the exercise is to build a machine learning model that maximizes the chances of telling whether or not a patient is likely to fall.

Summary of Results

The results show that the random forest model does best in predicting patients do not fall who actually do not fall (specificity). The random forest model offers the best specificity (0.8859) while extreme gradient boosting has the highest sensitivity (0.5437). The ensemble algorithm offers the best balanced accuracy (0.7048). Overall, we can rely on the models to predict which patients are at risk of falling and taking measures to support them.

The Data

In this section, I load the data into a file called fall. The data has seven variables. The dependent variable is `category` and has six levels.

1. Standing (coded 0)
2. Walking (coded 1)
3. Sitting (coded 2)
4. Falling (coded 3)
5. Cramps (coded 4)
6. Running (coded 5)

The rest of the variables are independent.

1. Time: monitoring time.
2. SL: sugar level.
3. EEG: Electroencephalogram (EEG) is a test that detects electrical activity in the brain.
4. BP: blood pressure.
5. HR: heart beat rate.
6. Circulation: blood circulation.

[1]The data is a redacted version of the one used in a study by Özdemir & Barshan (2017). However, the original dataset had over 300 features whilst the redacted version has 7 features

In total, the data has 16382 rows of data and 7 columns of data.

```
#####################################################################
# SECTION 3
#####################################################################
# Load the data ----

## Get the URL for the datafile
## The original source of the data is kagle.com, specifically https://www.kaggle.com/pitasr/falldata
## However, it is hard to download data straight from kagle without supplying login details
## hence, I loaded the data into my github account in the url below.

url <- "https://raw.githubusercontent.com/Karuitha/Final_Project_HarvardX/main/falldetection.csv"

## Download the dataset.

download.file(url = url, destfile = "fall.csv", method = "curl")

## load the dataset into R

fall <- read.csv("fall.csv") %>%

## Clean the column names by removing capital letters, spaces and special characters
  janitor::clean_names()

## The dependent variable is activity, classified as follows
## Fall detection data set of Chinese hospitals of old age patients.

### 0- Standing
### 1- Walking
### 2- Sitting
### 3- Falling
### 4- Cramps
### 5- Running

## Independent variables

### time: monitoring time
### sl: sugar level
### eeg: eeg monitoring rate- electroencephalogram (EEG) is a test that detects electrical activity in
### bp blood pressure
### hr: heart beat rate
### circulation: blood circulation

# Convert the dependent variable into a factor with category falling as the base
fall$activity <- factor(fall$activity, levels = c(3, 0, 1, 2, 4, 5))
```

Training set/ Test set split

I split the data into a training set and a test set. Here I use the function `initial_split` from `tidymodels` where I provide the data, the proportion of the data that goes into the training set and the dependent variable. After generating the index, I use the functions `training` and `testing` to specify the training and testing sets.

```
set.seed(123, sample.kind = "Rounding")
```

```r
# Specify the index to split data into training and testing set
index <- initial_split(fall, prop = 0.7, strata = activity)

# Sopecify the training set
fall_train <- training(index)

# Dimensions of the training set
dim(fall_train) # 11470 observations of 7 variables
```

```
## [1] 11470     7
```

```r
# Get the testing set
fall_test <- testing(index)

# Dimensions of the testing set
dim(fall_test) # 4912 observations of 7 variables
```

```
## [1] 4912    7
```

Note that any additional analysis will involve only the training set with the testing set reserved for evaluation of the final model. Next, I explore the training data

Exploratory Data Analysis- Training set

Data Structure

I first examine the structure of the data. Note that except for the dependent variable that is a factor with six levels, all the other variables are numeric. The training dataset has 11470 observations of 7 variables. The dependent variable is `activity`, a factor variable with 6 levels as follows.

0. Standing
1. Walking
2. Sitting
3. Falling
4. Cramps
5. Running

The level of interest in this case is to forecast whether or not a patient will fall. When weighing the options, it is better to predict that a patient will not fall when in fact they do not fall. While its also good to predict which patient will fall, the former has more weight. For this reason, I evaluate the model mainly using `specificity` and `balanced accuracy` rather than `sensitivity`.

The dependent variables are as follows.

1. time: monitoring time
2. sl: sugar level.
3. eeg: eeg monitoring rate- electroencephalogram (EEG) is a test that detects electrical activity in the brain.
4. bp: blood pressure.
5. hr: heart beat rate.
6. circulation: blood circulation.

I summarise the data below.

```r
## Overview of the training data.
## Structure of the training data
str(fall_train)
```

```
## 'data.frame':    11470 obs. of  7 variables:
```

```
##  $ activity   : Factor w/ 6 levels "3","0","1","2",..: 1 4 4 5 6 1 1 2 2 6 ...
##  $ time       : num  4723 4059 4774 7102 7015 ...
##  $ sl         : num  4020 2191 2788 14149 7337 ...
##  $ eeg        : num  -1600 -1146 -1263 -2381 -1700 ...
##  $ bp         : int  13 20 46 85 22 35 15 56 81 61 ...
##  $ hr         : int  79 54 67 120 95 157 196 249 133 214 ...
##  $ circluation: int  317 165 224 809 427 1519 1885 2826 847 1469 ...
```

```
## First 6 observations of the training dataset
head(fall_train) %>% knitr::kable(caption = "First Six Observations of the Training Set")
```

Table 1: First Six Observations of the Training Set

	activity	time	sl	eeg	bp	hr	circluation
1	3	4722.92	4019.64	-1600.00	13	79	317
2	2	4059.12	2191.03	-1146.08	20	54	165
3	2	4773.56	2787.99	-1263.38	46	67	224
5	4	7102.16	14148.80	-2381.15	85	120	809
6	5	7015.24	7336.79	-1699.80	22	95	427
7	3	8620.28	24949.90	-3198.06	35	157	1519

```
## last 6 observations of the training dataset
tail(fall_train) %>% knitr::kable(caption = "Last Six Observations of the Training Set")
```

Table 2: Last Six Observations of the Training Set

	activity	time	sl	eeg	bp	hr	circluation
16375	2	15767.90	115821.00	-4050.00	48	271	4426
16376	4	11468.80	87113.20	-2090.00	28	220	3450
16377	4	13625.90	61109.90	-4440.00	27	247	2754
16378	4	9280.68	11417.00	-3021.64	36	156	654
16379	3	8479.69	9455.54	-2932.85	17	138	554
16380	2	8872.53	27449.90	-2870.00	33	156	1364

```
## Number of rows in the training dataset
nrow(fall_train)
```

```
## [1] 11470
```

```
## Number of columns in the training dataset
ncol(fall_train)
```

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

Missing Data

As the summary below shows, the training data has no missing data points. Hence, I explore the data using visualizations next using the `missmap` function from the Amelia package.

```
# ************************************
# Exploratory data analysis: missing data
## Check for missing data.
sapply(fall_train, is.na) %>%
```

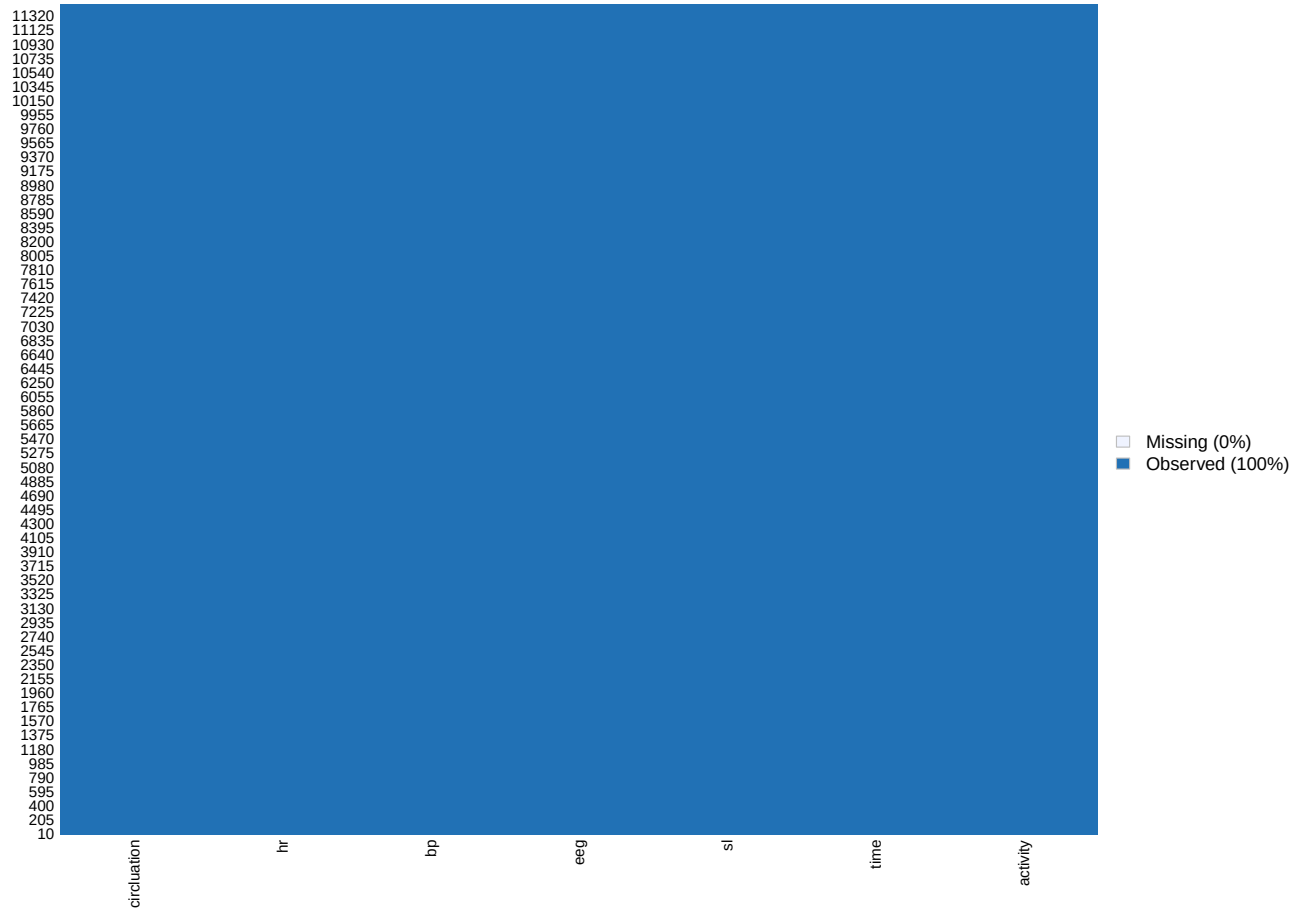
```r
## Get the colsums of the logical dataframe.
## The colsums represent missing data for each column.
colSums() %>%

## Make a tibble of column names and missing values.
dplyr::tibble(variables = names(fall), missing = .) %>%

## Arrange missing values in descending order of missingness
dplyr::arrange(desc(missing)) %>%

## Get the top 7 (number of columns)
## Our tibble has no missing data.
head(7) %>%

## make a nice table
knitr::kable(caption = "Missing Data in the Training Set")
```

Table 3: Missing Data in the Training Set

variables	missing
activity	0
time	0
sl	0
eeg	0
bp	0
hr	0
circluation	0

```
## Visualizing missingness of data
Amelia::missmap(fall_train, main = "Figure 1: Missingness Map- Training Set")
```

Figure 1: Missingness Map– Training Set




```
## Again there is no missing data
```

Other Summary Statistics

The independent variable is categorical with six categories. The category of interest is 3 (falling). While the other categories are important, maximizing the prediction for falling remains the most important objective as the other activities are not as harmful.

```
## Summary statistics for the dependent variables
summary(fall_train$activity) %>% knitr::kable(caption = "Summary of Dependent Variable")
```

Table 4: Summary of Dependent Variable

	x
3	2507
0	3226
1	358
2	1752
4	2444
5	1183

Table 5 below shows the summary statistics for the independent variables (features). The data shows that that there is a wide variation in the dataset with the standard deviation of the variables ranging from 48.38601 to 130029.85851.

```
##################################################
# Exploratory data analysis: summary statistics
fall_train %>%

  ## Deselect the activity column
  select(-activity) %>%

  ## Make a table of summary statistics
  skimr::skim() %>%

  ## Remove some uninformative columns
  dplyr::select(-contains(c("missing", "complete", "hist", "skim_type"))) %>%

  ## Rename remaining columns
  dplyr::rename(Variable = skim_variable,

                Mean = numeric.mean, SD = numeric.sd,

                Min = numeric.p0, Q1 = numeric.p25,

                Median = numeric.p50,

                Q3 = numeric.p75, Max = numeric.p100) %>%

  ## make a nice table
  knitr::kable(caption = "Summary Statistics for the `Fall` dataset", align = "l")
```

Table 5: Summary Statistics for the `Fall` dataset

Variable	Mean	SD	Min	Q1	Median	Q3	Max
time	10920.84471	5273.26154	1.95423e+03	7271.703	9768.170	13438.300	50895.5
sl	75111.23505	130029.85851	4.22242e+01	9877.298	31405.700	79959.275	2426140.0
eeg	-5920.05876	125364.84763	-1.26260e+07	-5613.955	-3364.555	-2160.512	1410000.0
bp	58.26957	48.38601	0.00000e+00	25.000	44.000	78.000	533.0
hr	211.18187	129.96850	3.30000e+01	119.000	180.000	266.750	986.0
circluation	2878.03444	3825.94978	5.00000e+00	587.000	1581.000	3539.000	52210.0

Data Visualization

In this section, I visualize the training dataset. First, I visualize the correlation matrix that shows extremely high correlation between the independent variables, for instance between `circulation` and `heart rate`. In its current form, running models using collinear data is likely yo yield unstable coefficients and hence unstable predictions. For this reason, I will run principal components analysis on the independent variables.

```r
# Exploratory data analysis: data visualization
fall_train[,-1] %>% GGally::ggpairs() +

  ## Add title and caption
  labs(title = "Figure 2: Correlation and Distribution of Independent Variables",

       caption = "Source: Author's Computations") +

       ## Add themes and adjust the font size plot title
       ggthemes::theme_clean() + theme(plot.title = element_text(size = 8)) +

       ## Adjust font sizes of axis text
       theme(axis.text = element_text(size = 6))
```

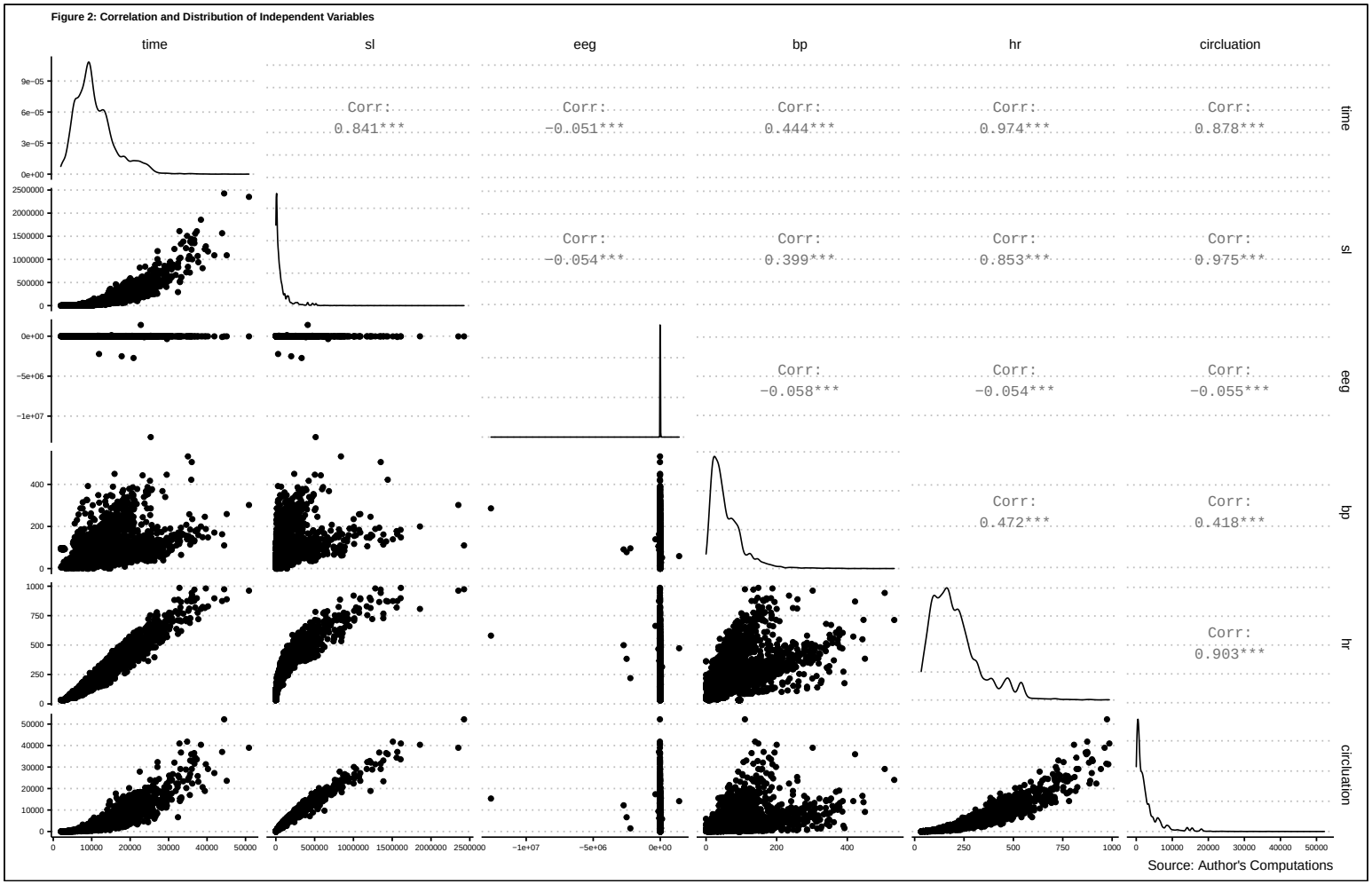


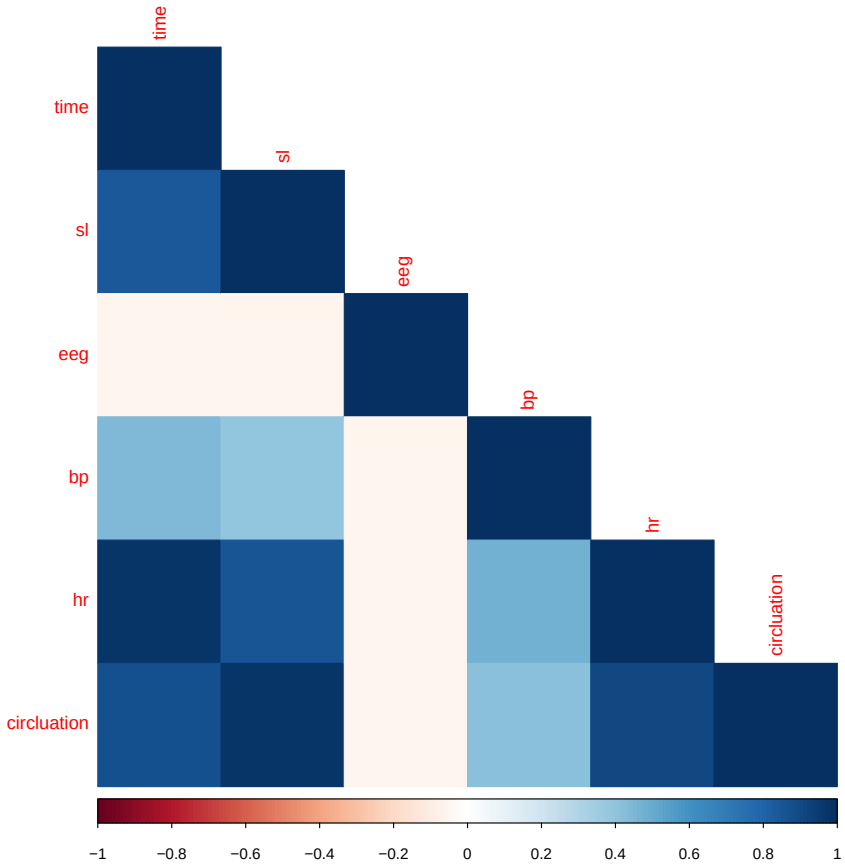
Figure 2: Correlation and Distribution of Independent Variables
time
sl
eeg
bp
hr
circluation
Corr: 0.841***
Corr: −0.051***
Corr: 0.444***
Corr: 0.974***
Corr: 0.878***
Corr: −0.054***
Corr: 0.399***
Corr: 0.853***
Corr: 0.975***
Corr: −0.058***
Corr: −0.054***
Corr: −0.055***
Corr: 0.472***
Corr: 0.418***
Corr: 0.903***
Source: Author's Computations

Correlation among the variables

Figure 2 shows the high degree of correlation between the independent variables which could make a model have unstable coefficients. I intend to correct the multicollinearity by running principal components analysis.

```
## Visualize the correlation
## Run correlation on independent variables and call the corrplot
cor(fall_train[,-1]) %>% corrplot::corrplot(method = "color",

            ## Specify the corrplot type and add title
            type = "lower", title = "Figure 3: A Visual of the Correlation Matrix - Training Set")
```

Figure 3: A Visual of the Correlation Matrix – Training Set



Class Balance/ Imbalance

Here I check for data balance and find that class 1 has very low prevalence that is likely to affect predictive accuracy on the test set. I intend to correct for the low prevalence by up-sampling the classes that have very low prevalence in order to create a balanced dataset.

```
## Check for possible class imbalance on the dependent variable
## Make a table of class counts
table(fall_train$activity)
```

```
##
##    3    0    1    2    4    5
## 2507 3226  358 1752 2444 1183
```

```
## Make a proportionate table of class counts
prop.table(table(fall_train$activity))
```

```
##
##          3          0          1          2          4          5
## 0.21857018 0.28125545 0.03121186 0.15274629 0.21307759 0.10313862
```

```
# Class 1 has a problem of low prevalence.
```

Method

I adopt the following strategy;

- First, given that the data has high collinearity, I run a principal components analysis (PCA) and generate uncorrelated variables.
- I deal with the problem of imbalanced data by up-sampling the under-represented classes.
- I run seven machine learning models and then make an ensemble.
- I compare the performance of the models and choose the most optimal.

Principal Components Analysis and Handling Class Imbalance

In this section, I do principal components analysis (PCA) and balance the datasets. Note that I have applied the transformation to deal with extreme values that exist in our data.

```
## Create a PCA recipe
pca_recipe <- recipe(activity ~ ., data = fall_train)

## Do the PCA analysis
pca_trans <- pca_recipe %>%

  ## Ensure all predictors have a mean of zero
  step_center(all_predictors()) %>%

  ## Ensure all predictors have a standard deviation of one
  step_scale(all_predictors()) %>%

  ## Run the principal components analysis
  step_pca(all_predictors()) %>%

  ## Apply adjustment to deal with outliers
  step_spatialsign(all_predictors()) %>%
```

```
  ## Adjust data to deal with missing values
  step_upsample(activity, over_ratio = 1) %>%

  ## Appply alll the steps above and generate new dataset.
  prep()
```

Visualizing the Principal Components

Here, I extract the standard deviations, compute the variance and the cumulative variance for the PCs. The data shows that the first principal component accounts for about 66% of the variability while the second PC accounts for 17% of the variation.

```
## Check the names of the dependent variables
names(pca_trans)
```

```
## [1] "var_info"       "term_info"      "steps"          "template"
## [5] "levels"         "retained"       "tr_info"        "orig_lvls"
## [9] "last_term_info"
```

```
## Access the standard deviations
sdev <- pca_trans$steps[[3]]$res$sdev

# View the standard deviations output
sdev
```

```
## [1] 1.9926884 0.9989131 0.8655107 0.4871137 0.1755044 0.1190608
```

```
## The contribution of PCA to the total variation
variance_explained <- (sdev ^ 2) / sum(sdev ^ 2)

# The variance explained by each principal component
variance_explained
```

```
## [1] 0.661801145 0.166304549 0.124851469 0.039546628 0.005133631 0.002362579
```

Here, I make the scree plot showing the contribution of each principal component to the overall variability in the data.

```r
## Plot a scree plot
## Create dataframe of principal components and variance explained by each PC>
data.frame(pc = paste("pc", 1:length(variance_explained)), variance = variance_explained) %>%

  # Call ggplot and supply the axes
  ggplot(aes(x = pc, y = variance, fill = pc)) +

  ## Add the geoms- geom_col and geom_label
  geom_col() + geom_label(show.legend = FALSE, mapping = aes(label = round(variance, 4))) +

  # Add a title
  ggtitle("Figure 4: Skree Plot - Contribution of Each PC to Total Variability") +

  # Remove title
  theme(legend.position = "none") +

  # Add a pleasant theme
  ggthemes::theme_clean() +

  # Add labels and caption
  labs(y = "Variance", caption = "Source: Author's Construction")
```

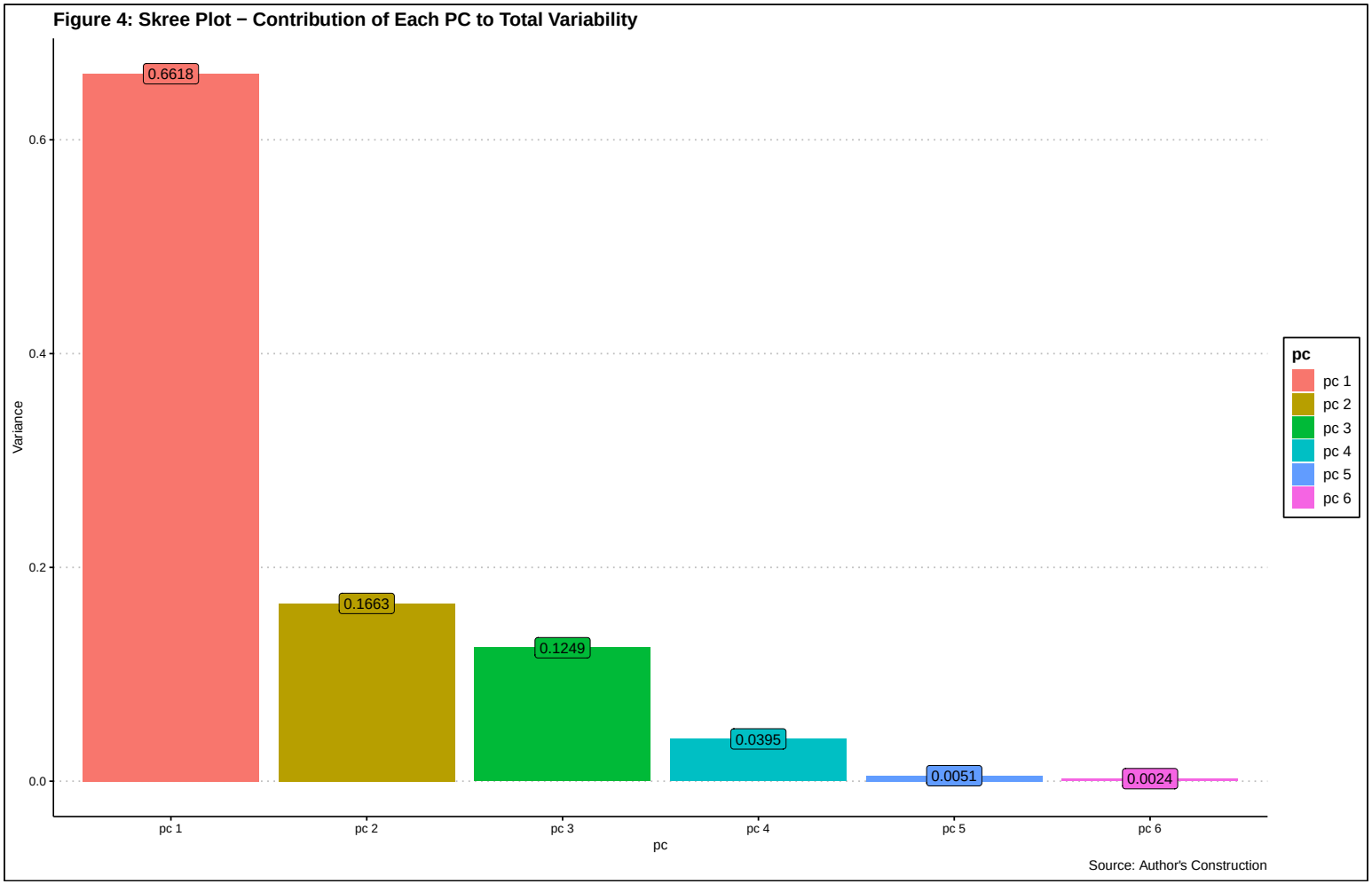


Figure 4: Skree Plot − Contribution of Each PC to Total Variability
0.6618
0.1663
0.1249
0.0395
0.0051
0.0024
0.6
0.4
0.2
0.0
Variance
pc 1
pc 2
pc 3
pc 4
pc 5
pc 6
pc
pc
pc 1
pc 2
pc 3
pc 4
pc 5
pc 6
Source: Author's Construction

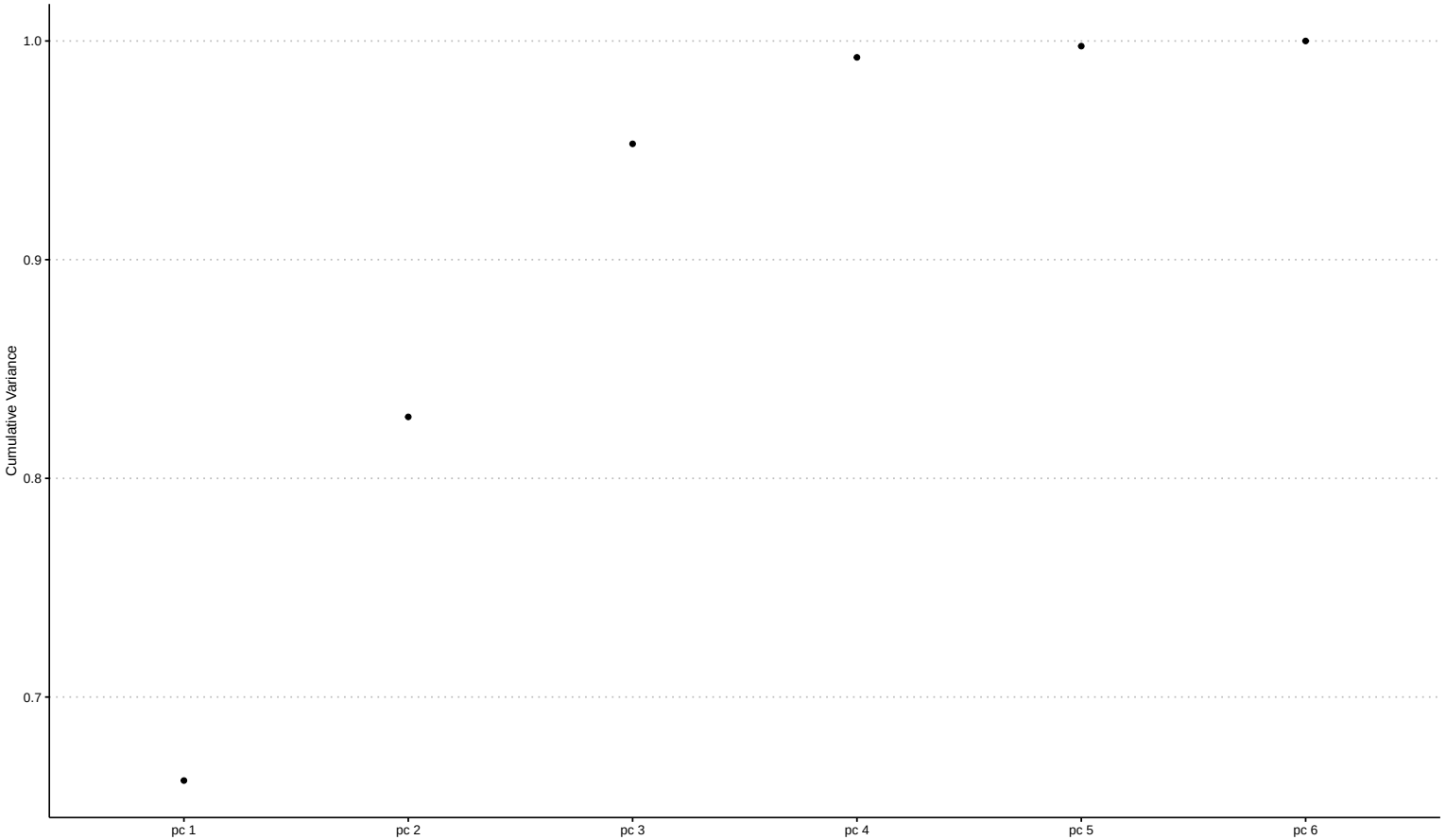
In this section I make the plot for the cumulative variance. both the summary and the graph show that the first four principal components account for over 99% of the variability.

```
## Plot a scree plot
## The cumulative variance captured by the PCAs
variance_explained_cum <- cumsum(sdev ^ 2) / sum(sdev ^ 2)

## View the cumulative variance explained
variance_explained_cum
```

```
## [1] 0.6618011 0.8281057 0.9529572 0.9925038 0.9976374 1.0000000
```

```r
## Create a dataframe of principal components and cumulative variance
data.frame(pc = paste("pc", 1:length(variance_explained_cum)),

           variance = variance_explained_cum) %>%

  ## Specify the axes
  ggplot(aes(x = pc, y = variance, group = pc)) +

  # Add a geom
  geom_point() +

  # Add a title
  ggtitle("Figure 5: Skree Plot - Cumulative Contribution of Each PC to Total Variability") +

  # remove legend
  theme(legend.position = "none") +

  # Add a nice theme
  ggthemes::theme_clean() +

  # Add axes labels and title
  labs(y = "Cumulative Variance", x = "Principal Components", caption = "PC1 is the contribution of PC1 to overall variance. PC2 is the co
```

Figure 5: Skree Plot − Cumulative Contribution of Each PC to Total Variability



PC1 is the contribution of PC1 to overall variance. PC2 is the contribution of PC1 and PC2 to overall variance, and so on
Source: Author's Construction

Next, I extract the summary statistics. Note that the new dataset for PCAs has 19356 rows of data. Note that the classes are now evenly balanced with each class having 3226 observations. Also, the summary shows that the variability between the variables has reduced markedly.

```
## Extract the transformed dataset
fall_train <- pca_trans %>% juice()

## The number of rows in the training dataset containing pcas
pca_trans %>% juice() %>% nrow()
```

```
## [1] 19356
```

```
## Checking class balances by dependent variable
table(pca_trans %>%

        ## Extract transformed data
        juice() %>%

        ## Select dependent variable to check for balance in classes
        select(activity)) %>%

        ## Make a nice table and add title
        knitr::kable(caption = "Distribution of Classes")
```

Table 6: Distribution of Classes

Var1	Freq
3	3226
0	3226
1	3226
2	3226
4	3226
5	3226

```
## Summary of the PCAs
pca_trans %>%

  ## Extract transformed data
  juice() %>%

  # Select all variables except activity
  select(-activity) %>%

  ## Summarize the data
  skim_without_charts() %>%

  ## Select some variables in the resulting table
  select(-skim_type, -n_missing, -complete_rate) %>%

  ## Make a nice table
  knitr::kable(caption = "Summary Statistics for PCAs in the training set")
```

Table 7: Summary Statistics for PCAs in the training set

skim_variable	numeric.mean	numeric.sd	numeric.p0	numeric.p25	numeric.p50	numeric.p75	numeric.p100
PC1	-0.3381147	0.7243397	-0.9990787	-0.9438338	-0.7515406	0.3747073	0.9993459
PC2	-0.0069751	0.0601367	-0.9376360	-0.0444654	-0.0052108	0.0282205	0.9973387
PC3	0.0470980	0.4965560	-0.9949437	-0.1684862	0.1871634	0.3274623	0.9892141
PC4	-0.0534431	0.2996874	-0.9980425	-0.2296614	0.0180149	0.1861670	0.5795946
PC5	0.0006080	0.1262368	-0.8911030	-0.0438459	0.0089687	0.0498038	0.9193233

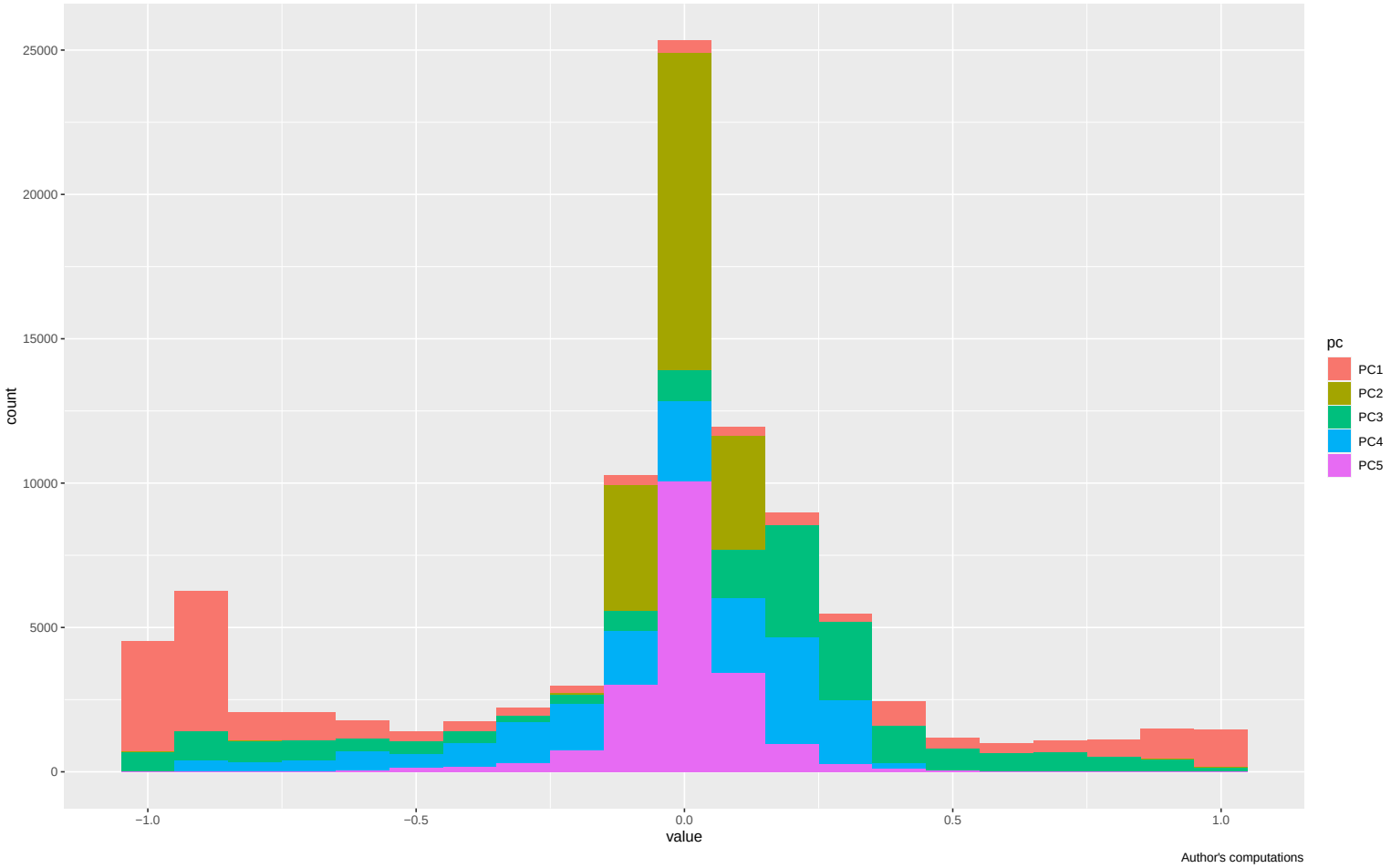
In this section, I examine the distribution of the values of each principal components to see the extent to which outliers exist.

```r
## Checking for extreme values
pca_trans %>% juice() %>%

  ## Convert the data to tidy format
  pivot_longer(-activity, names_to = "pc", values_to = "value") %>%

  ## Filter for values that meet threshold of 7.5 to filter out extreme values
  filter(value > 7.5)
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: activity <fct>, pc <chr>, value <dbl>
```

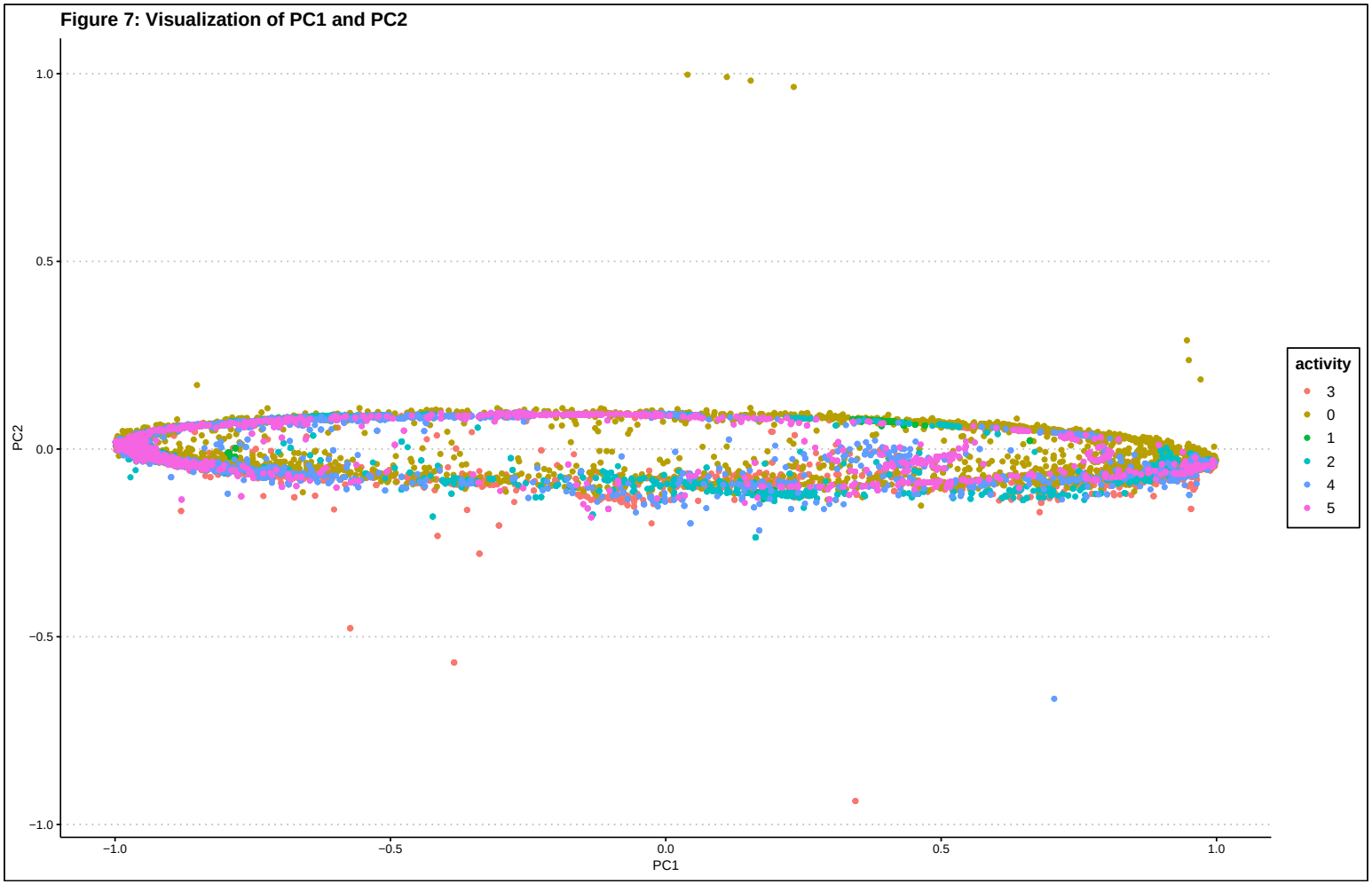
```r
## Plot histogram for PCAs
pca_trans %>% juice() %>%

  ## Convert data to tidy format
  pivot_longer(-activity, names_to = "pc", values_to = "value") %>%

  ## Filter for values within limits to avoid extreme values
  filter(value <= 7.5 & value >= -7.5) %>%

  ## Plot the data by supplying axes
  ggplot(mapping = aes(x = value, fill = pc), color = "black") +

  ## Add the geom and specify binwidth
  geom_histogram(binwidth = 0.1) +

  ## Add titles and labels
  labs(title = "Figure 6: Histogram of PCs", caption = "Author's computations")
```

Figure 6: Histogram of PCs



Next, I visualize the first 2 PCs

```
pca_trans %>%

  ## Extract transformed variables - the principal components
  juice() %>%

  ## Plot the first two principal components
  ggplot(mapping = aes(x = PC1, y = PC2, color = activity)) +

   ## Add a geom and a pleasant theme
   geom_point() + ggthemes::theme_clean() +

  ## Add labels and titles
  labs(title = "Figure 7: Visualization of PC1 and PC2")
```



Next, I apply the same transformation I have made on the training set to the testing set. Here, I use a function `bake`.

```
## Transform the testing data similar to the training data
fall_test <- pca_trans %>%

  ## Use of bake function to generate testing test transformed exactly like the training set
  bake(fall_test)
```

Running the ML models

I now run machine learning models and evaluate each of them in the following order.

1. Classification tree.
2. Random Forest.
3. K-Nearest Neighbours.
4. Extreme Gradient Boosting.
5. Generalized Multinomial Logit Model.
6. Ensemble of all the models above.

Note that in evaluating the models, I will focus mainly on the specificity given that it is more expensive to predict no-fall in a patient who falls that it is to predict a fall in a patient who does not fall. Hence, a model that predicts that a patient will not fall with a higher precision is better than one that predicts a fall with equal level of precision. However, I will also, side by side, consider overall accuracy and specificity in case of ties. Also, given that our model has more than two classes, I consider the specificity of the main class `fall`. While the other classes are important, our case will be specific to predicting which patients fall or do not fall. The baseline accuracy on the test set, guessing the most frequent outcome, is 0.2200733.

In all cases, I run a 10-fold cross validation and set a random seed as follows.

```
## Set seed to be used in the models
seeds <- set.seed(123, sample.kind = "Rounding")

## Set up cross validation parameters
control <- trainControl(method = "repeatedcv",

                        repeats = 10,

                        seeds = seeds)
```

Classification tree

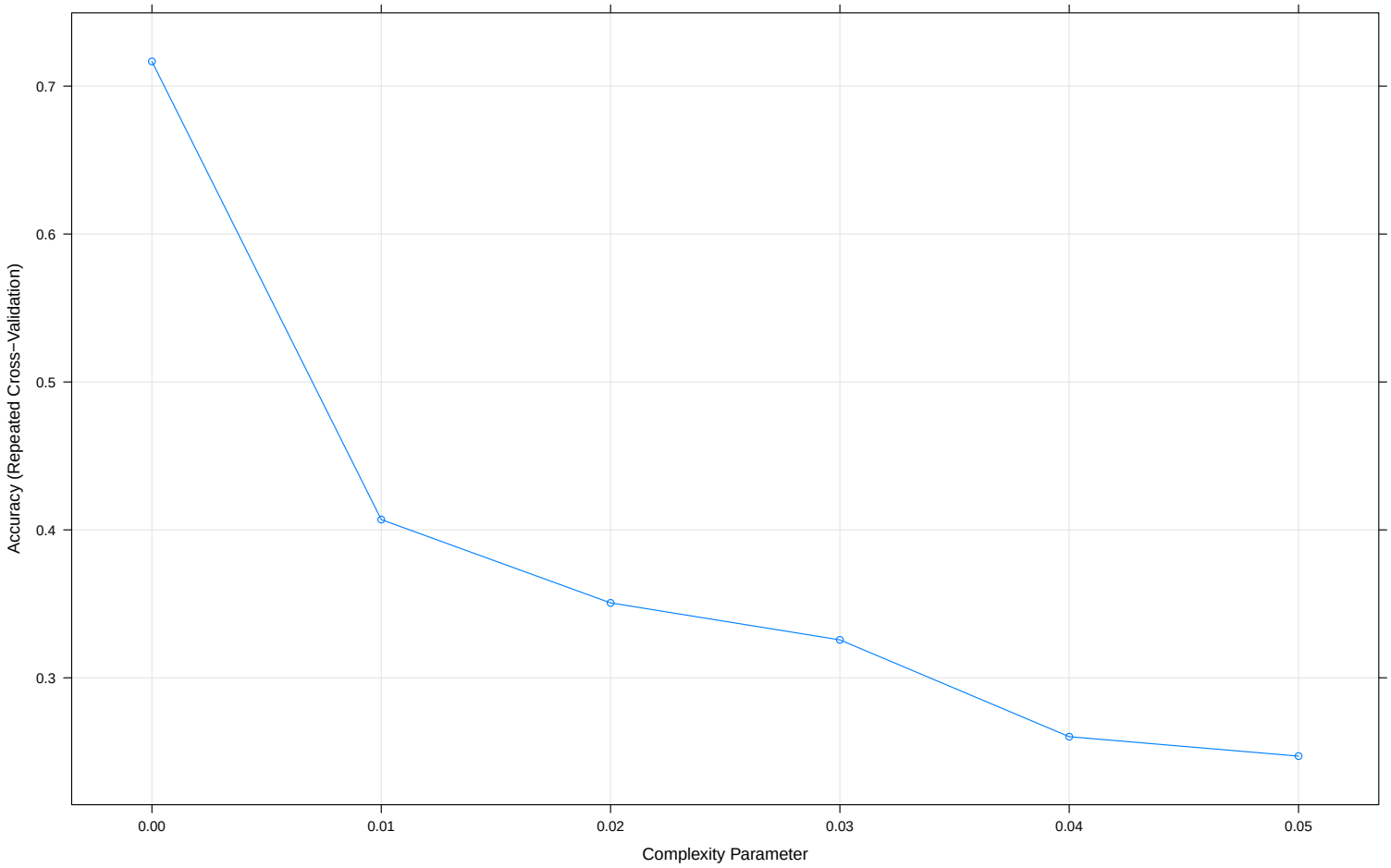
In this section, I run the classification tree using the code chunk below. The tree model has overall accuracy 0.5138 against a no information rate (NIR) of 0.2814.The model has a specificity of 0.86453 and a sensitivity of 0.44866. Note that the optimal complexity parameter is near zero.

```
# The classification tree model
tree <- caret::train(activity ~ .,

                        data = fall_train,

                        # specify engine to use
                        method = "rpart",

                        # Set up cross validation
                        trControl = control,

                        # Set up metric to use
                        metric = "Accuracy",

                        # Tuning parameters
                        tuneGrid = expand.grid(cp = seq(0, 0.05, 0.01)))

# make predictions on the test set
```

```r
tree_prediction <- predict(tree, newdata = fall_test)

# Generate confusion matrix on the test set
confusionMatrix(tree_prediction, fall_test$activity)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 485 154   2 115 198  50
##          0 124 798   2  51 141  72
##          1  16  25 100  52  18   3
##          2 163  93  25 443  69  28
##          4 205 173   6  51 449 103
##          5  88 139   9  38 175 249
##
## Overall Statistics
##
##                Accuracy : 0.5138
##                  95% CI : (0.4998, 0.5279)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3943
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity           0.44866   0.5774  0.69444  0.59067  0.42762  0.49307
## Specificity           0.86453   0.8895  0.97609  0.90918  0.86069  0.89812
## Pos Pred Value        0.48307   0.6717  0.46729  0.53959  0.45491  0.35673
## Neg Pred Value        0.84749   0.8432  0.99063  0.92496  0.84688  0.93925
## Prevalence            0.22007   0.2814  0.02932  0.15269  0.21376  0.10281
## Detection Rate        0.09874   0.1625  0.02036  0.09019  0.09141  0.05069
## Detection Prevalence  0.20440   0.2419  0.04357  0.16714  0.20094  0.14210
## Balanced Accuracy     0.65659   0.7335  0.83527  0.74992  0.64416  0.69559
```

```
plot(tree)
```




```
#rpart.plot(tree$finalModel)
```

The Random Forest Model

The random forest model does way better than the tree in terms of accuracy with an overall accuracy of 0.6179 against a no information rate of 0.2814, a specificity level of 0.8859, and a sensitivity level of 0.5171.

```
## The random forest model
## Set up the tuning parameters
tunegrid <- expand.grid(.mtry= sqrt(ncol(fall_train)))

## Set up the random forest model
rf_default <- caret::train(activity ~ ., data = fall_train,

              # Set up engine, cross validation and tuning parameters
              method = "rf", tunegrid = tunegrid, trControl = control)

## make predictions on the test set
rf_prediction <- predict(rf_default, newdata = fall_test)

## generate confusion matrix
confusionMatrix(rf_prediction, fall_test$activity)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    3    0    1    2    4    5
##          3  562   96    2  115  175   32
##          0  102 1067    2   40  123   69
##          1   10   11   90   33   13    3
##          2  152   49   44  498   47   19
##          4  200   91    4   49  565  120
##          5   55   68    2   15  127  262
##
## Overall Statistics
##
##                Accuracy : 0.6197
##                  95% CI : (0.606, 0.6333)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5209
##
##  Mcnemar's Test P-Value : 0.001044
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity            0.5199   0.7721  0.62500   0.6640   0.5381  0.51881
## Specificity            0.8904   0.9048  0.98532   0.9253   0.8799  0.93941
## Pos Pred Value         0.5723   0.7605  0.56250   0.6156   0.5491  0.49527
## Neg Pred Value         0.8679   0.9102  0.98864   0.9386   0.8751  0.94456
## Prevalence             0.2201   0.2814  0.02932   0.1527   0.2138  0.10281
## Detection Rate         0.1144   0.2172  0.01832   0.1014   0.1150  0.05334
## Detection Prevalence   0.1999   0.2856  0.03257   0.1647   0.2095  0.10770
## Balanced Accuracy      0.7051   0.8384  0.80516   0.7946   0.7090  0.72911
```

K-Nearest Neighbours (KNN)

Although the KNN model does not perform as well as the random forest model, it is way batter than the tree model. The overall accuracy for the KNN is 0.5189 against a no information rate of 0.2134. The sensitivity is 0.5105 while the specificity is 0.8587. Finally, the balanced accuracy is quite good at 0.6846.

```
## K-Nearest Neighbours (KNN)
## Set up the model and cross validation
knn <- train(activity ~ ., data = fall_train, method = "knn",

             trControl = control)


# Generate the confusion matrix
confusionMatrix(fall_test$activity, predict(knn, newdata = fall_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 530  49  19 185 211  87
##          0 200 616  33 125 194 214
##          1   0   0 111  29   3   1
##          2 107  14  62 493  35  39
##          4 172  60  39  88 480 211
##          5  40  32   9  32  90 302
##
## Overall Statistics
##
##                Accuracy : 0.5155
##                  95% CI : (0.5014, 0.5295)
##     No Information Rate : 0.2136
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.406
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity            0.5052   0.7990  0.40659   0.5179  0.47384  0.35363
## Specificity            0.8574   0.8150  0.99289   0.9351  0.85381  0.94998
## Pos Pred Value         0.4903   0.4457  0.77083   0.6573  0.45714  0.59802
## Neg Pred Value         0.8645   0.9561  0.96602   0.8897  0.86199  0.87474
## Prevalence             0.2136   0.1570  0.05558   0.1938  0.20623  0.17386
## Detection Rate         0.1079   0.1254  0.02260   0.1004  0.09772  0.06148
## Detection Prevalence   0.2201   0.2814  0.02932   0.1527  0.21376  0.10281
## Balanced Accuracy      0.6813   0.8070  0.69974   0.7265  0.66382  0.65180
```

Extreme Gradient Boosting (XGBoost)

In this section, I run the extreme gradient boosting (XGBoost) model. XGBoost refers to the engineering goal to push the limit of computations resources for boosted tree algorithms.

```r
# The XGboost model
## Set up the tuning parameters
tune_grid <- expand.grid(nrounds = 200,
                         max_depth = 5,
                         eta = 0.05,
                         gamma = 0.01,
                         colsample_bytree = 0.75,
                         min_child_weight = 0,
                         subsample = 0.5)

## Set up model, cross validation and tuning parameters
xgb_model <- train(activity ~., data = fall_train, method = "xgbTree",
                trControl = control,
                tuneGrid = tune_grid,
                tuneLength = 10)

## generate confusion matrix for the test set predictions
confusionMatrix(fall_test$activity, predict(xgb_model, newdata = fall_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 497 128  24 199 165  68
##          0 123 860  34 102 132 131
##          1   0   0 110  29   2   3
##          2  86  66  60 479  43  16
##          4 176 125  32  92 460 165
##          5  43  57  13  33  85 274
##
## Overall Statistics
##
##                Accuracy : 0.5456
##                  95% CI : (0.5316, 0.5596)
##     No Information Rate : 0.2516
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4354
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity            0.5373   0.6958  0.40293  0.51285  0.51860  0.41705
## Specificity            0.8535   0.8580  0.99267  0.93188  0.85342  0.94571
## Pos Pred Value         0.4598   0.6223  0.76389  0.63867  0.43810  0.54257
## Neg Pred Value         0.8883   0.8935  0.96581  0.89068  0.88944  0.91309
## Prevalence             0.1883   0.2516  0.05558  0.19015  0.18058  0.13375
## Detection Rate         0.1012   0.1751  0.02239  0.09752  0.09365  0.05578
## Detection Prevalence   0.2201   0.2814  0.02932  0.15269  0.21376  0.10281
## Balanced Accuracy      0.6954   0.7769  0.69780  0.72236  0.68601  0.68138
```

Multinomial Logit Model

In the multinomial logit model below, the overall accuracy is very poor- at 0.2828 against a no information rate of 0.2814. However the model has good specificity at 0.8630 and a very low sensitivity of 0.2970. Consequently, the balanced accuracy is also low at 0.57995. Overall this model lacks good predictive power going by the balanced accuracy and sensitivity relative to the other models. However, it has specificity that is reasonable.

```
## Set up the multinomial logit model
multinom <- multinom(activity ~ ., data = fall_train)
```

```
## # weights:  42 (30 variable)
## initial  value 34681.296286
## iter  10 value 33291.048162
## iter  20 value 32947.236234
## iter  30 value 31766.916187
## iter  40 value 31532.958068
## final  value 31532.892228
## converged
```

```
## Predict the multinomial logit model on the test set
multinom_predict <- predict(multinom, newdata = fall_test)

## Get confusion matrix
confusionMatrix(multinom_predict, fall_test$activity)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 321 126   3  98 219  79
##          0 193 663   6  94 146  84
##          1 299 353 108 359 283 108
##          2  74  67   6  67  61  33
##          4  38  29   0  25  61  32
##          5 156 144  21 107 280 169
##
## Overall Statistics
##
##                Accuracy : 0.2828
##                  95% CI : (0.2702, 0.2956)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : 0.4174
##
##                   Kappa : 0.1554
##
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity           0.29695   0.4797  0.75000  0.08933  0.05810  0.33465
## Specificity           0.86296   0.8518  0.70596  0.94210  0.96789  0.83935
## Pos Pred Value        0.37943   0.5590  0.07152  0.21753  0.32973  0.19270
## Neg Pred Value        0.81308   0.8070  0.98942  0.85165  0.79078  0.91673
## Prevalence            0.22007   0.2814  0.02932  0.15269  0.21376  0.10281
## Detection Rate        0.06535   0.1350  0.02199  0.01364  0.01242  0.03441
```

```
## Detection Prevalence  0.17223   0.2414  0.30741  0.06270  0.03766  0.17854
## Balanced Accuracy     0.57995   0.6658  0.72798  0.51571  0.51299  0.58700
```

Ensemble

Ensemble 1: With multinomial logit In this section, I assemble all the models and use a voting method to build an ensemble. The prediction is the value that receives the most votes from each of the models. In the ensemble, the overall accuracy is 0.5928, a specificity of 0.8804, and a sensitivity of 0.5291. The model does a good job in predicting who will not fall but rather poorly in predicting who will fall. Part of the reason for the low sensitivity maybe the inclusion of the multinomial logit model that had extremely low sensitivity. I remove the multinomial logit model from the ensemble and build a new ensemble next.

```
## make a dataframe with predictions on the test on all the models
ensemble <- tibble(tree = predict(tree, newdata = fall_test), rf = predict(rf_default, newdata = fall_t


## Create a new colum with outcome being the most popular outcome for the models
ensemble$ensemble_all <- apply(ensemble, 1, function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
})

## Convert the ensembled column into a factor
ensemble$ensemble_all <- factor(ensemble$ensemble_all, levels = levels(ensemble$rf))

## Compute the confusion matrix on the model
confusionMatrix(ensemble$ensemble_all, fall_test$activity)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 577 117   1 104 192  40
##          0  91 968   0  30 118  50
##          1  17  20 106  50  26   8
##          2 148  63  34 497  58  29
##          4 182 111   1  45 491  95
##          5  66 103   2  24 165 283
##
## Overall Statistics
##
##                Accuracy : 0.5949
##                  95% CI : (0.581, 0.6086)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4942
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity            0.5338   0.7004  0.73611   0.6627  0.46762  0.56040
## Specificity            0.8815   0.9181  0.97462   0.9202  0.88762  0.91831
```

```
## Pos Pred Value         0.5597   0.7701  0.46696   0.5995  0.53081  0.44012
## Neg Pred Value         0.8701   0.8867  0.99189   0.9380  0.85979  0.94800
## Prevalence             0.2201   0.2814  0.02932   0.1527  0.21376  0.10281
## Detection Rate         0.1175   0.1971  0.02158   0.1012  0.09996  0.05761
## Detection Prevalence   0.2099   0.2559  0.04621   0.1688  0.18831  0.13090
## Balanced Accuracy      0.7076   0.8093  0.85537   0.7914  0.67762  0.73935
```

Ensemble 2: Without Mutinomial logit The multinomial logit performs poorly and hence I remove it in making the second ensemble. However, both the sensitivity and specificity barely change with this tweak given that the other models subsume the mistakes made by the multinomial logit model.

```
## make a dataframe with predictions on the test on all the models
ensemble2 <- tibble(tree = predict(tree, newdata = fall_test), rf = predict(rf_default, newdata = fall_


## Create a new colum with outcome being the most popular outcome for the models
ensemble2$ensemble_all <- apply(ensemble2, 1, function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
})

## Convert the ensembled column into a factor
ensemble2$ensemble_all <- factor(ensemble2$ensemble_all, levels = levels(ensemble2$rf))

## Compute the confusion matrix on the model
confusionMatrix(ensemble$ensemble_all, fall_test$activity)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   0   1   2   4   5
##          3 577 117   1 104 192  40
##          0  91 968   0  30 118  50
##          1  17  20 106  50  26   8
##          2 148  63  34 497  58  29
##          4 182 111   1  45 491  95
##          5  66 103   2  24 165 283
##
## Overall Statistics
##
##                Accuracy : 0.5949
##                  95% CI : (0.581, 0.6086)
##     No Information Rate : 0.2814
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4942
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 3 Class: 0 Class: 1 Class: 2 Class: 4 Class: 5
## Sensitivity            0.5338   0.7004  0.73611   0.6627  0.46762  0.56040
## Specificity            0.8815   0.9181  0.97462   0.9202  0.88762  0.91831
## Pos Pred Value         0.5597   0.7701  0.46696   0.5995  0.53081  0.44012
```

```
## Neg Pred Value         0.8701   0.8867  0.99189   0.9380  0.85979  0.94800
## Prevalence             0.2201   0.2814  0.02932   0.1527  0.21376  0.10281
## Detection Rate         0.1175   0.1971  0.02158   0.1012  0.09996  0.05761
## Detection Prevalence   0.2099   0.2559  0.04621   0.1688  0.18831  0.13090
## Balanced Accuracy      0.7076   0.8093  0.85537   0.7914  0.67762  0.73935
```

Model Evaluation

I present the overall results in the table 8 below. Overall, the random forest model does best in terms of specificity and overall accuracy followed by the ensemble without the multinomial logit model. In terms of sensitivity, the extreme gradient boosting model performs best followed by the ensemble without the multinomial logit model. Ensemble with Multinomial logit model has the highest balanced accuracy. The results are in the table below.

```
## make a table of results
tribble(~ Model, ~ Specificity, ~ Sensitivity, ~ BalancedAccuracy, ~ Accuracy, ~ NoInformationRate, "Cl
        "Random Forest Model", "0.8859", "0.5171", "0.7015", "0.6179", "0.2814", "K Nearest Neighbours"
```

Table 8: Results of the Machine Learning Models

Model	Specificity	Sensitivity	BalancedAccuracy	Accuracy	NoInformationRate
Classification Tree	0.8645	0.4487	0.6566	0.5138	0.2814
Random Forest Model	0.8859	0.5171	0.7015	0.6179	0.2814
K Nearest Neighbours	0.8587	0.5105	0.6846	0.5189	0.2134
Extreme Gradient Boosting	0.8563	0.5437	0.7000	0.5450	0.2484
Multinomial Logit	0.8630	0.2970	0.5800	0.2828	0.2814
Ensemble with Multinomial	0.8815	0.5282	0.7049	0.5930	0.2814
Ensemble without Muntinomial	0.8804	0.5291	0.7048	0.5928	0.2814

Conclusion

In this project, I have built models to predict whether or not a patient falls. I have developed several models- the classification tree, the random forest model, the K-nearest neighbour model, extreme gradient boosting, multinomial logit models and two ensembles- one with the multinomial logit model and one without the multinomial logit model. In evaluating the models I use specificity- the accuracy in predicting that a patient will not fall when in fact, they do not fall as it less expensive to predict that a patient will fall and they do not fall. Overall, the random forest model does the best in terms of specificity, while extreme gradient boosting has the best sensitivity. The challenges I have encountered in this exercise include computing power where the models take too long to run. The computational power has affected my ability to explore more models and fine tune the parameters for better results.

References

Anava, O., & Levy, K. (2016). k*-nearest neighbours: From global to local. In Advances in neural information processing systems (pp. 4916-4924).

Chen, Z., & Fan, W. D. (2019). A multinomial logit model of pedestrian-vehicle crash severity in North Carolina. International journal of transportation science and technology, 8(1), 43-52.

Özdemir, A. T., & Barshan, B. (2014). Detecting falls with wearable sensors using machine learning techniques. Sensors, 14(6), 10691-10708.

Shaikhina, T., Lowe, D., Daga, S., Briggs, D., Higgins, R., & Khovanova, N. (2019). Decision tree and random forest models for outcome prediction in antibody incompatible kidney transplantation. Biomedical Signal Processing and Control, 52, 456-462.