Human Activity Recognition Using Smart Phone Data

My Vien

December 09, 2021

Contents

1 Summary

Most of us have been familiar and seen technology devices like Fitbit or Apple Watch. They are all-in-one fitness bands, smartwatches and accesories that can monitor, track human activities for steps, exercise, heart rates, weights and many more. Those devices are primarily using accelerometer and gyroscope which are triaxial sensors to generate a signal which can be processed and analized as an activity. In this project, we would like to use sequences of signals recorded by smart phones for building a machine learning classifier of detecting Human Activities.

```
# load the data into R
data <- read_csv("https://raw.githubusercontent.com/tmvien/human-activites/master/train.csv")
# checking Null value
sum(is.na(data))
```

```
## [1] 0
```

2 Data Description

This data is acquired from **Kaggle**. The data record was from an experiment on a group of 30 volunteers from 19-48 years old. Each volunteer wearing a smartphone on their waist and performed 6 acitivities including: WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING. The smart phones have equiped with accelerometer and gyroscope and are able to capture 3-axial linear acceleration and 3-axial angular velocity. For more details on how the data were processed with more research

papers, please refer to this **link**. The training data is 7352, 563 dimension with 1 column as the response variable. Some columns are not neccessary for model training and will be excluded. There is no missing values.

3 Data Analysis

We first change the order of response variable in the data to first column and remove column 562 "subject" which indicate anonymous volunteers who participated in the experiment. This column is unneccessary for data visualization and modeling. Furthermore, since we plan to use KNN algorithm, we will need to normalize the data, make sure that all features are in the same range.

```
# change order of columns so that target variable will be the first column
data <- data[,c(ncol(data),1:561)]
# function to bring features range between 0 and 1
range01 <- function(x){(x-min(x))/(max(x)-min(x))}

for (col in colnames(data)[-1]) {
  data[, col] <- range01(data[, col])
}
```

3.1 Data Visualization

3.1.1 Univariate Analysis

In this part, we will mostly explore the distribution of features. First, let's see how many training examples we have in each class.

```
data %>%
  ggplot(aes(x=Activity, fill=Activity)) +
  geom_bar(stat = "count") +
  scale_fill_viridis_d() +
  theme(
    axis.text.x=element_text(angle = 15)
    , panel.background = element_rect(fill="#E8EDFB")
    , legend.position='none'
  )
```

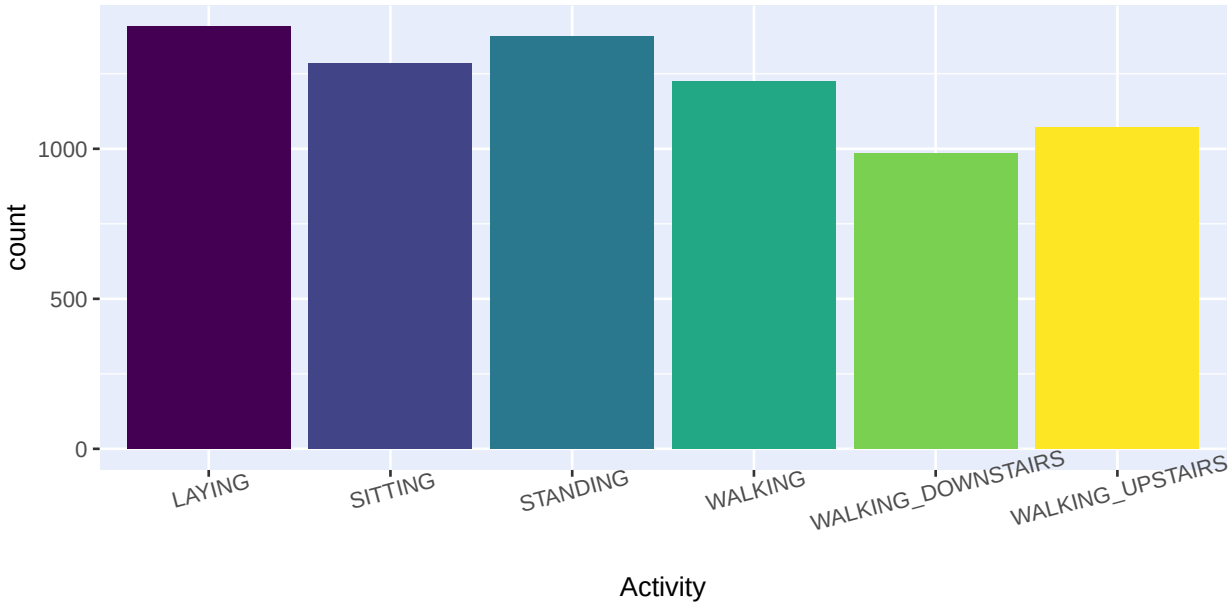

Figure 1: Number of Training Examples in Each Class

The dataset looks fairly balanced between classes. We have fewer examples in walking downstair class compared to others but it should be fine as long as no class has examples dominating others.

Below is the distribution of the Fast Fourier Transform (FFT) magnitude of three-dimensional signals entropy recorded from gyroscope and calculated using the Euclidean norm. We notice that distributions of WALKING_UPSTAIRS, WALKING_DOWNSTAIRS AND WALKING are similar while LAYING, SITTING AND STANDING distributions are close and resembled each other. Using only this features, we could easily classifying them into two groups: WALKING and NOT WALKING.

```
data %>%
  ggplot(aes(x=`fBodyBodyGyroMag-entropy()`, fill=Activity)) +
  geom_density(alpha=0.4) +
  scale_fill_viridis_d() +
  labs(x = "Feature 536-fBodyBodyGyroMag-entropy()") +
  theme(
    panel.background = element_rect(fill="#E8EDFB")
  )
```

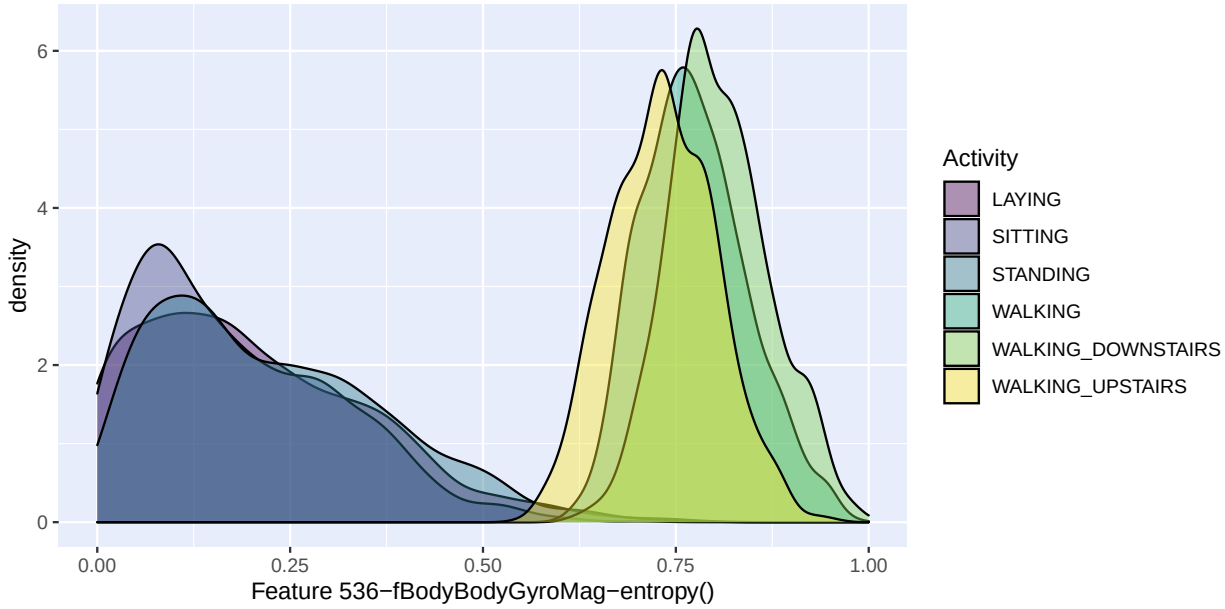

Figure 2: Distribution of FFT Entropy Magnitude (L2 Norm) of 3-Dimensional Signals ih Each Class

Next figure is the boxplot of FFT mean L2 norm magnitude of 3-dimensional signals recorded by accelerometer. Using this feature, we can further isolate WALKING_DOWNSTAIR class from the WALKING group we have as above.

```
data %>%
  ggplot(aes(x = Activity, y=`fBodyAccMag-mean()`, fill=Activity)) +
  geom_boxplot(alpha = 0.4) +
  labs(y = "Feature 530-fBodyAccMag-mean()") +
  theme(
    axis.text.x=element_text(angle = 10)
    , panel.background = element_rect(fill="#E8EDFB")
    , legend.position='none'
  )
```

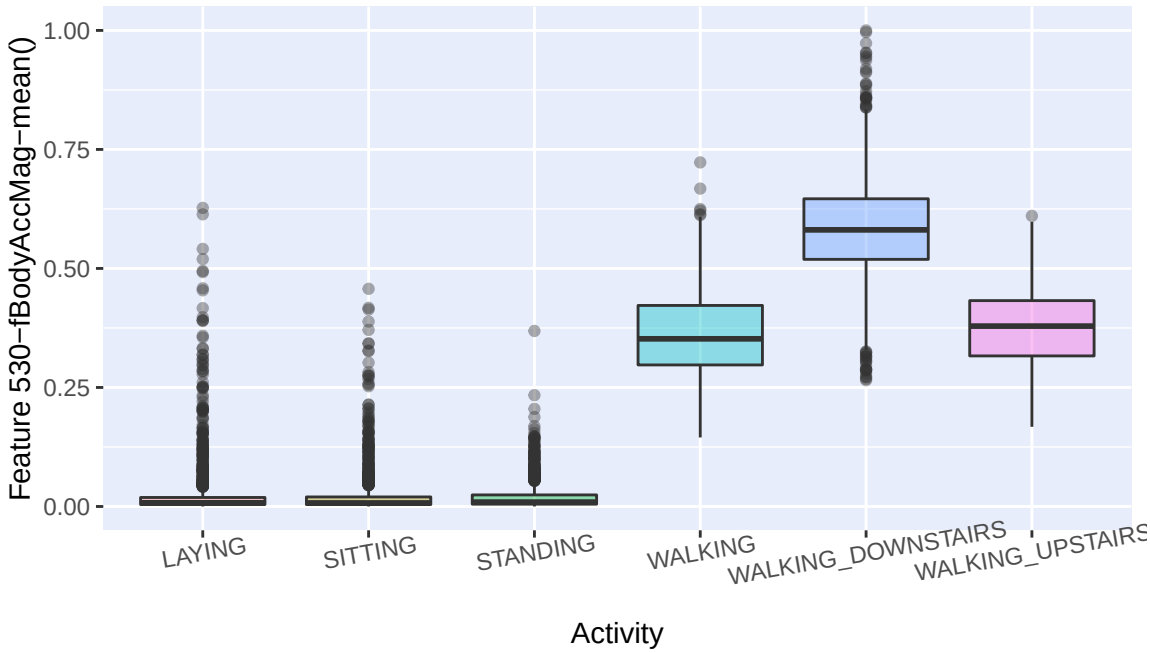

Figure 3: Boxplot of FFT Mean Magnitude (L2 Norm) of 3-Dimensional Signals in Each Class

Figure 4 is is the feature depicting the angle between vector X (X is used to denote signal in X direction) and gravity mean. By applying this 560th feature, we will be able to set apart LAYING class from the rest because of how seperated of this class from others.

```
data %>%
  ggplot(aes(x=`angle(X,gravityMean)`, fill=Activity)) +
  geom_density(alpha=0.6) +
  scale_fill_viridis_d() +
  labs(x = "Feature 560-angle(X,gravityMean)") +
  theme(
    panel.background = element_rect(fill="#E8EDFB")
  )
```

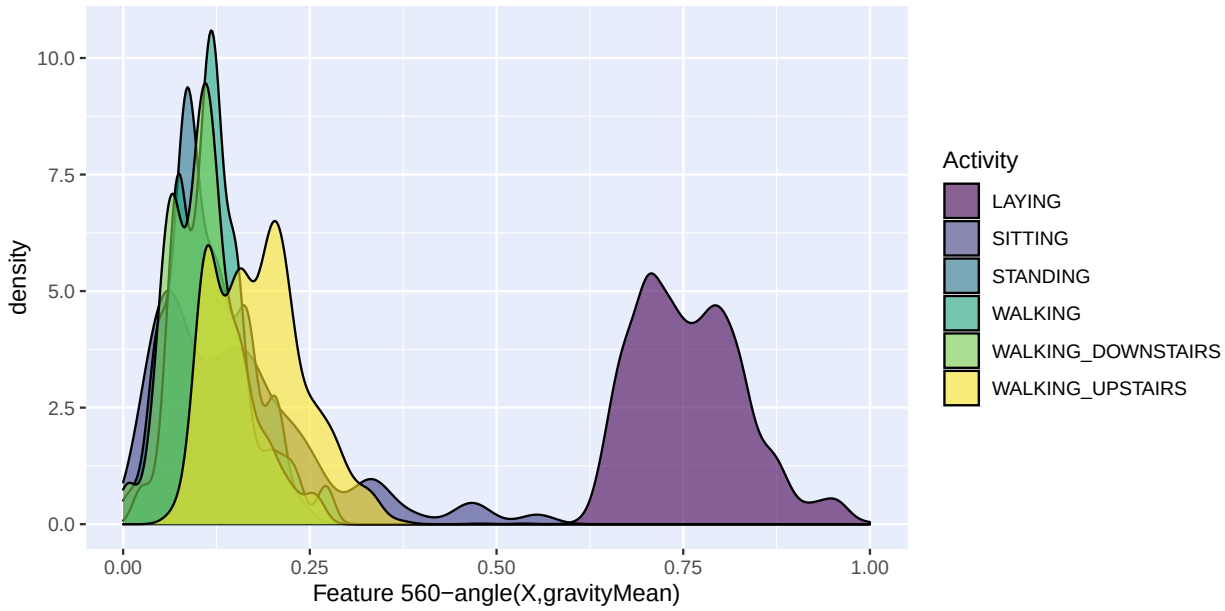

Figure 4: Distribution of Angle between X and Gravity Mean in Each Class

3.1.2 Experiment with PCA (Principal Components Analysis)

Now, let's perform Principal Component Analysis and select 2 largest principal components. The scree plot shows that we will be able to have ~70% of proportion of variance explained if choosing the first 2 PCs. We then can visualize them in a 2D plot to see if the classes are seperable by using only 2 PCs.

```
# A pca object
d.pca <- prcomp(data[,-1])
# scree plot
g1 <- fviz_eig(d.pca)
# 2PCs visualization
g2 <- fviz_pca_ind(d.pca, geom.ind = "point", pointshape = 21,
             pointsize = 2,
             fill.ind = data$Activity,
             col.ind = "black",
             palette = "jco",
             addEllipses = TRUE,
             label = "var",
             col.var = "black",
             repel = TRUE,
             legend.title = "Diagnosis") +
  theme(plot.title = element_text(hjust = 0.5))
grid.arrange(g1, g2, ncol=2)
```

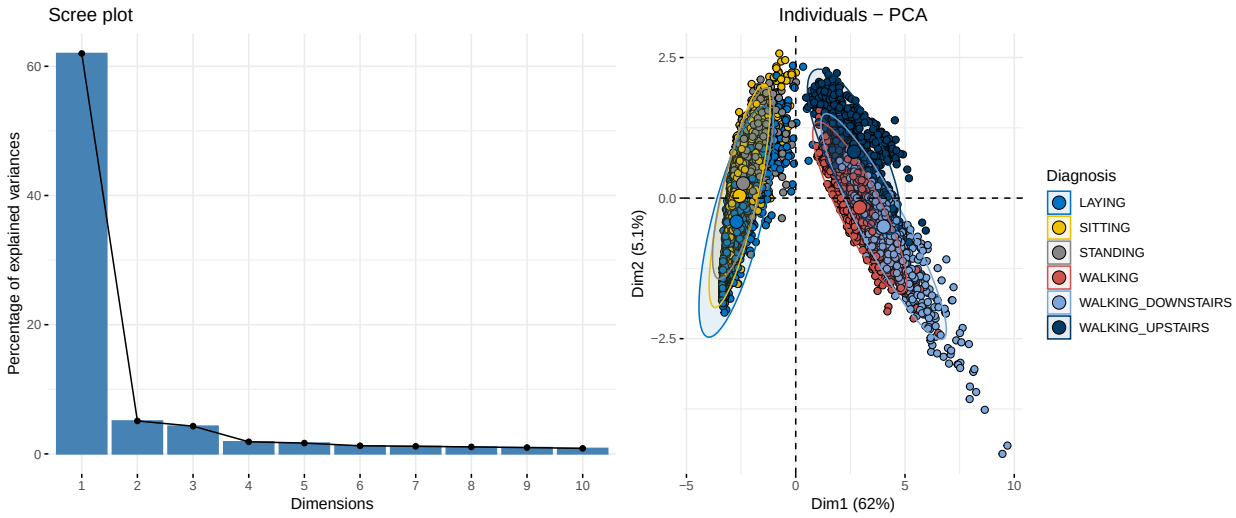

Figure 5: Scree Plot, 2D PCA-plot from 561 features

From the 2D plot on the right hand side of **figure 5**, we can see that by using 2PCs, the classes are not well seperable and had overlappings between them. The classes LAYING, SITTING, STANDING look clustered together and not be able to be identifiable.

3.1.3 More Visualization with t-SNE

We have tried to visualize the data with 2PCs but the result does not seem so well. There is another dimension reduction technique that can be used for exploring high-dimensional data, that is, t-SNE. While PCA is a linear transformation, t-SNE (t-Distributed Stochastic Neighbor Embedding) is a non-linear dimensionality reduction technique which is able to capture complex relationship between data points. T-SNE works by measuring the distances between all the data points to the interest point, plot them into the probability curve.

```
tSNE.fit <- data[,-1] %>%
  Rtsne()
(tSNE.df <- tSNE.fit$Y %>%
  as.data.frame() %>%
  rename(tSNE1="V1",
         tSNE2="V2") %>%
  mutate(Activity=data$Activity)) %>%
  ggplot(aes(x = tSNE1,
             y = tSNE2)) +
  geom_point(aes(fill = Activity, alpha = 0.4), colour = "black", shape = 21, size = 3) +
  scale_color_viridis_d() +
  theme(
    legend.position="bottom"
    , panel.background = element_rect(fill="#E8EDFB")
  )
```

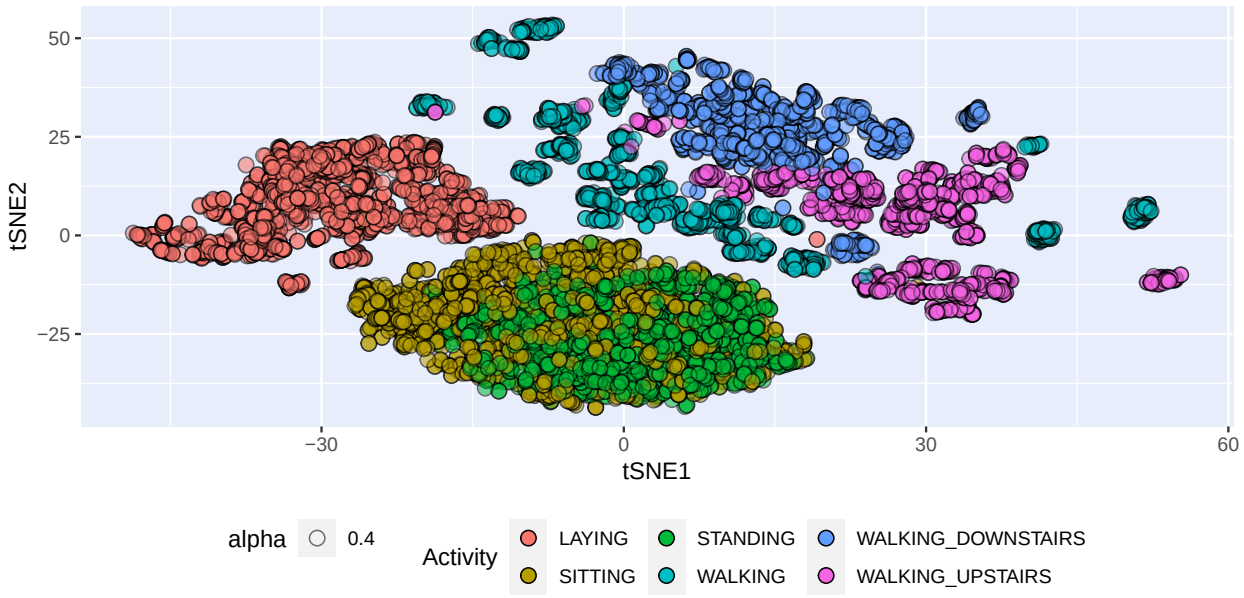

Figure 6: t-SNE transformation

From **figure 6** we can see that t-SNE produces a better plot than PCA technique even though STANDING and SITTING classes still look clustered together.

3.2 Models

Now, we will build a classification machine for this dataset. First step is to convert the response variable which is Activity column to numerical value. Second, we will split the whole dataset into train and valid set. Using 10 fold cross validation, we fit the train data into different algorithms and perform optimization for the best parameters that give lowest misclassification error rate (or accuracy score). We will experiment with 4 different models for this task: Mutinomial Ridge Logistic Regression, Mutinomial Lasso Logistic Regression, Multinomial Elastic Net Logistic Regression and KNN algorithm.

```
# encode response variable with number
y <- ifelse(data$Activity == "WALKING", 1,
            ifelse(data$Activity == "WALKING_UPSTAIRS", 2,
                   ifelse(data$Activity == "WALKING_DOWNSTAIRS", 3,
                          ifelse(data$Activity == "SITTING", 4,
                                 ifelse(data$Activity == "STANDING", 5, 6)
                                 )
                          )
                   )
            )
y <- factor(y, labels = c("WALKING", "WALKING_UPSTAIRS", "WALKING_DOWNSTAIRS", "SITTING", "STANDING", "
# checking if there is any mismatch
sum(which(data$Activity == "WALKING") != which(y==1))
```

```
## [1] 0
```

```r
sum(which(data$Activity == "WALKING_UPSTAIRS") != which(y==2))
```

```
## [1] 0
```

```r
sum(which(data$Activity == "WALKING_DOWNSTAIRS") != which(y==3))
```

```
## [1] 0
```

```r
sum(which(data$Activity == "SITTING") != which(y==4))
```

```
## [1] 0
```

```r
sum(which(data$Activity == "STANDING") != which(y==5))
```

```
## [1] 0
```

```r
sum(which(data$Activity == "LAYING") != which(y==6))
```

```
## [1] 0
```

The code chunk below creates predictors matrix X, response y, training indices and a matrix to store models' performance and time for cross-validating and training.

```r
# create matrix for train data
X <- as.matrix(data[,-1])
n <- dim(X)[1]
p <- dim(X)[2]
# rm the original data
rm(data)
# using 80% of data for train and 20% for validation
train <- sample(n, n*0.8)
# number of K fold cross-validation
K <- 10
# a matrix for storing misclassification rate
m.score <- matrix(0, ncol = 4, nrow = 2)
colnames(m.score) <- c("MRidge", "MLasso", "MElastic", "KNN")
rownames(m.score) <- c("Model", "Time")
```

For Multinomial Logistic Regression models, we use package "glmnet" for cross-validation and training. With KNN model, we will use package "caret". The cv.glmnet function from "glmnet" package will automatically select a set of lambdas for cross-validation. However, we will pick a sequence of odd K nearest numbers ranging from 3 to 31 for the cross-validation on our KNN model. The valid accuracy score together with time for cross-validation and training will be recorded.

```r
######### Ridge
# define alpha
a = 0
# record time
start.time      <-    proc.time()
```

```
cv.rd <- cv.glmnet(X[train,], y[train], family = "multinomial", nfolds=K, type.measure = "class", alpha=
rd.fit <- glmnet(X[train,], y[train], lambda = cv.rd$lambda.min,  family = "multinomial", alpha = a)
end.time        <-    proc.time() - start.time
m.score[2,1]    <-    end.time['elapsed']
y.hat.rd        <-    predict(rd.fit, newx = X[-train,], type = "class")
m.score[1,1]    <-    mean(y[-train] == y.hat.rd)
######### Lasso
# define alpha
a = 1
# record time
start.time      <-    proc.time()
cv.ls   <- cv.glmnet(X[train,], y[train], family = "multinomial", nfolds=K, type.measure = "class", alpl
ls.fit  <- glmnet(X[train,], y[train], lambda = cv.ls$lambda.min,  family = "multinomial", alpha = a)
end.time        <-    proc.time() - start.time
m.score[2,2]    <-    end.time['elapsed']
y.hat.ls        <-    predict(ls.fit, newx = X[-train,], type = "class")
m.score[1,2]    <-    mean(y[-train] == y.hat.ls)
########## ElasticNet
# define alpha
a = 0.5
# record time
start.time      <-    proc.time()
cv.el <- cv.glmnet(X[train,], y[train], family = "multinomial", nfolds=K, type.measure = "class", alpha=
el.fit <- glmnet(X[train,], y[train], lambda = cv.el$lambda.min,  family = "multinomial", alpha = a)
end.time        <-    proc.time() - start.time
m.score[2,3]    <-    end.time['elapsed']
y.hat.el        <-    predict(el.fit, newx = X[-train,], type = "class")
m.score[1,3]    <-    mean(y[-train] == y.hat.el)

########### KNN
# Define training control
train.control   <- trainControl(method = "cv", number = K)
# record time
start.time      <-    proc.time()
# Cross validate and train the model
knn.fit <- train(x=X[train,], y=y[train], method = "knn",
                 trControl = train.control,
                 metric = "Accuracy",
                 tuneGrid = expand.grid(.k=seq(3, 31, 2)))
end.time        <-    proc.time() - start.time
m.score[2,4]    <-    end.time['elapsed']
y.hat.knn       <-    predict(knn.fit,newdata = X[-train,] , type = "raw")
m.score[1,4]    <-    mean(y[-train] == y.hat.knn)
```

Figure 7 illustrates the cross-validation process of all 4 models. Keep in mind that **during the cross-validation process**, Multinomial Logistic Regression models from "glmnet" package use misclassification rate metric while "caret" uses accuracy score. Nevertheless, the idea of both metric is cross-related. Misclassification is just the opposite of accuracy score. Accuracy indicates how often the classifier correct is and misclassification rate indicates how often it is wrong. Nevertheless, the accuracy score is still our metric for the valid dataset.

```
knn.result <- knn.fit$results[,c("k", "Accuracy")]
par(mfrow=c(2,2))
```

```
plot(cv.rd, main="Multinomial Ridge Logistic")
plot(cv.ls, main="Multinomial Lasso Logistic")
plot(cv.el, main="Multinomial Elastic Net Logistic")
plot(knn.result$k, knn.result$Accuracy, type = "b",
     col = "blue", ylab = "Accuracy", xlab ="k",
     main = "KNN")
```

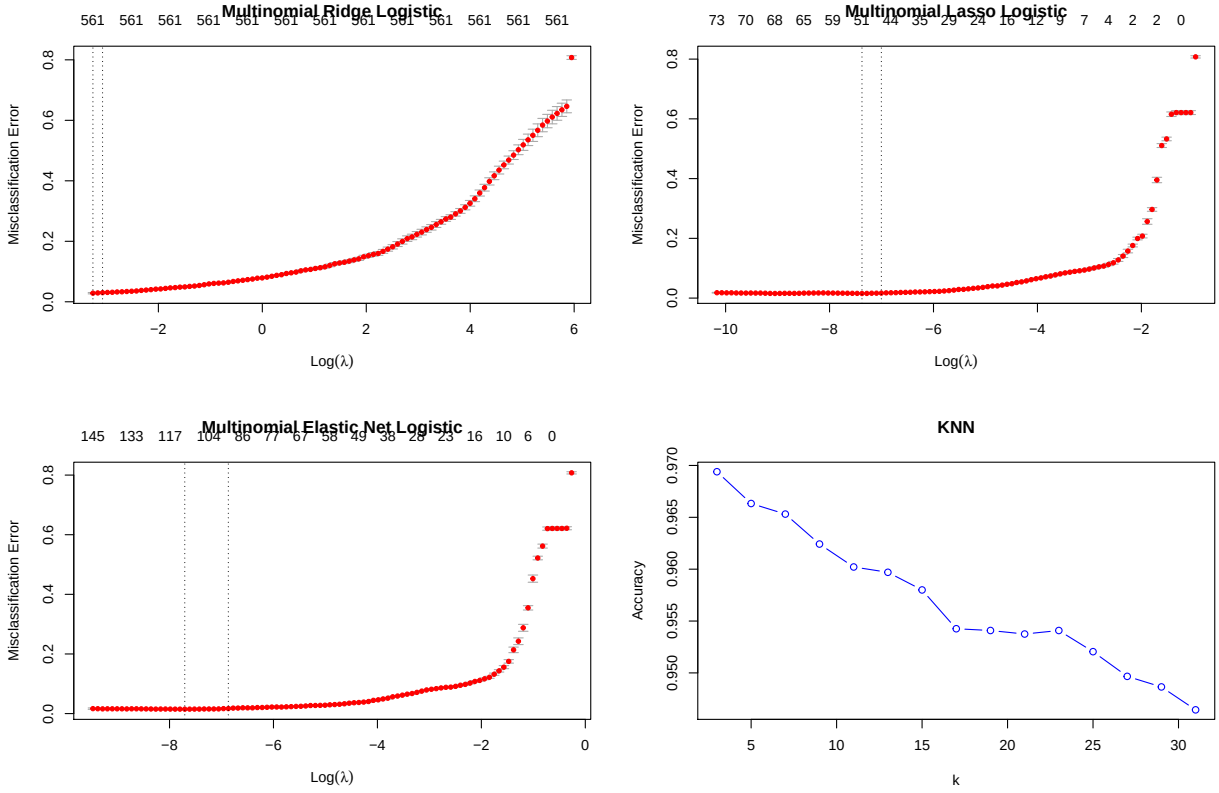

Figure 7: 10 Fold Cross Validation

The best K number from cross-validation for KNN model is 3. The lambdas from cross-validation of multinomial logistic are 0.03841, 0.000626, 0.0004499 for Ridge, Lasso and Elastic Net, respectively. Since misclassification rate is the opposite of accuracy score, the cross-validation graphs between Multinomial Logistic Models and KNN look opposite.

The accuracy score and time for cross-validation and training are as table below:

	MRidge	MLasso	MElastic	KNN
Model	0.9619307	0.9789259	0.9830048	0.9734874
Time	1681.8840000	1108.5630000	1059.9370000	4828.7000000

4 Conclusion

KNN is the slowest algorithm and the performance score does not excel other models. Taking into account both the performance score and time spent for each model, MElastic is the best model with highest accuracy

score. Moreover, time for training and cross-validation on MElastic is acceptable compared to other models. Considering that companies like Fitbit or Apple has much huger computational resources and power, the performance score could be even improved and training time can become negligible.