DualTasking

Kejing Yan

2023-10-26

1. Initial baseline walk
2. walk w/ low obstacle
3. walk w/ medium obstacle
4. walk w/ high obstacle
5. typing while walk
6. texting while walk
7. typing while walk w/ low obstacle
8. texting while walk w/ low obstacle
9. typing while walk w/ medium obstacle
10. texting while walk w/ medium obstacle
11. typing while walk w/ high obstacle
12. texting while walk w/ high obstacle
13. final baseline walk

Create excel file

```
data <- read_excel("TWWT_fulldata.xlsx")
```

Create dataframe of averaged data

```
data_avg <- data.frame(matrix(nrow = 156, ncol = 23))

for (i in seq(1, 468, by = 3)) {
  j = (i+2)/3
  data_avg[j,1:2] = data[i,1:2]
  for (k in c(4:17, 21:23)) {
    num1 = as.numeric(data[i, k])
    num2 = as.numeric(data[i+1, k])
    num3 = as.numeric(data[i+2, k])
    if (is.na(num1) && is.na(num2) && is.na(num3)) {
      data_avg[j,k] = NA
    } else {
      count = 3
      if (is.na(num1)){
        num1 = 0
        count = count - 1
      }
      num2 = as.numeric(data[i+1, k])
      if (is.na(num2)){
        num2 = 0
        count = count - 1
      }
      num3 = as.numeric(data[i+2, k])
      if (is.na(num3)){
        num3 = 0
        count = count - 1
      }
      data_avg[j, k] = (num1+ num2+ num3)/count
    }
  }
  for (k in 18:20) {
    if (is.na(data[i,k]) != TRUE) {
        data_avg[j, k] <- data[i, k]
    } else if (is.na(data[i+1,k]) != TRUE){
      data_avg[j, k] <- data[i+1, k]
    } else {
      data_avg[j, k] <- data[i+2, k]
    }

  }
}

data_avg = data_avg[,-3]
colnames(data_avg) = colnames(data)[-3]
```

Kmeans Clustering

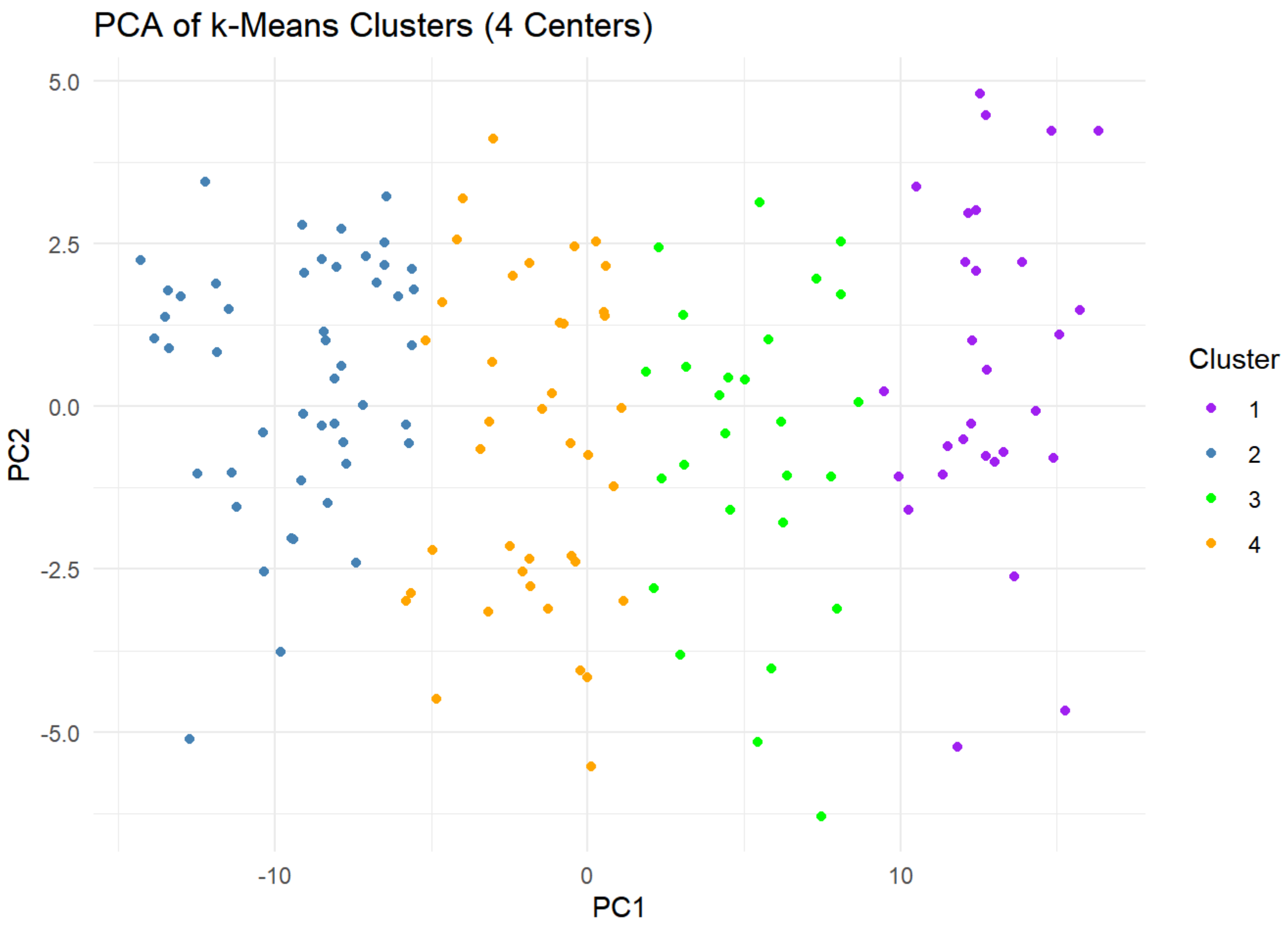
```
#copy data into a new dataframe with 17 independent variables
cluster.data <- data_avg[,-c(1,2,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,1:14] <- scale(cluster.data[,c(1:14,16:17)])
```

```
## Warning in matrix(value, n, p): 数据长度[2496]不是矩阵列数[14]的整倍数
```

```
cluster.data <- na.omit(cluster.data)

#cluster with 4 centers (conjecturing that four groups will correspond to four different obstac
le height)
cl <- kmeans(cluster.data, centers = 4)

#copy cluster result into dataframe
plot_data <- na.omit(data_avg[,-c(22)])
plot_data$Cluster <- as.factor(cl$cluster)

plot_data$Obstacle <- as.factor(plot_data$Obstacle)
plot_data$Task <- as.factor(plot_data$Task)
```

Cluster with 4 centers

```
# Perform PCA
pca_result <- prcomp(cluster.data, rank. = 2)  # reduce to 2 dimensions for visualization

loadings <- pca_result$rotation

# Add PCA results to the original data frame
plot_data$PC1 <- pca_result$x[, 1]
plot_data$PC2 <- pca_result$x[, 2]

# Use ggplot2 to create the PCA scatter plot colored by clusters
ggplot(plot_data, aes(x = PC1, y = PC2, color = as.factor(Cluster))) +
  geom_point() +
  scale_color_manual(values = c("purple", 'steelblue', 'green', 'orange','pink','grey','blac
k','darkblue','darkgreen','yellow','blue','red')) + # Customize colors if needed
  theme_minimal() +
  labs(title = "PCA of k-Means Clusters (4 Centers)", color = "Cluster")
```



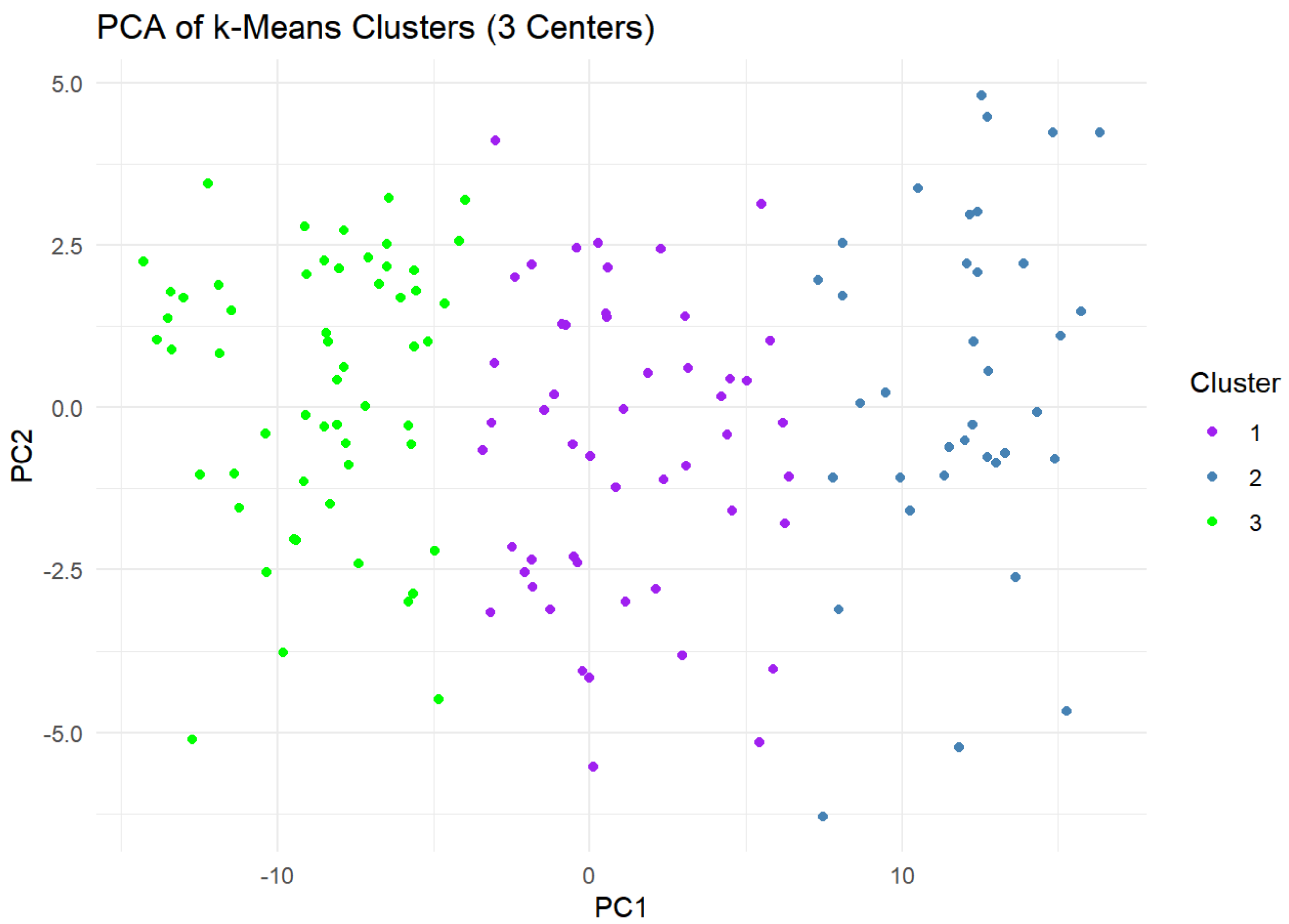

```
#copy data into a new dataframe with 17 independent variables
cluster.data <- data_avg[,-c(1,2,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,1:14] <- scale(cluster.data[,c(1:14,16:17)])
```

```
## Warning in matrix(value, n, p): 数据长度[2496]不是矩阵列数[14]的整倍数
```

```
cluster.data <- na.omit(cluster.data)

#cluster with 3 centers (conjecturing that four groups will correspond to three different task)
cl <- kmeans(cluster.data, centers = 3)

#copy cluster result into dataframe
plot_data <- na.omit(data_avg[,-c(22)])
plot_data$Cluster <- as.factor(cl$cluster)

plot_data$Obstacle <- as.factor(plot_data$Obstacle)
plot_data$Task <- as.factor(plot_data$Task)
```

Cluster with 3 centers

```
# Perform PCA
pca_result <- prcomp(cluster.data, rank. = 2)  # reduce to 2 dimensions for visualization

loadings <- pca_result$rotation

# Add PCA results to the original data frame
plot_data$PC1 <- pca_result$x[, 1]
plot_data$PC2 <- pca_result$x[, 2]

# Use ggplot2 to create the PCA scatter plot colored by clusters
ggplot(plot_data, aes(x = PC1, y = PC2, color = as.factor(Cluster))) +
  geom_point() +
  scale_color_manual(values = c("purple", 'steelblue', 'green', 'orange','pink','grey','blac
k','darkblue','darkgreen','yellow','blue','red')) + # Customize colors if needed
  theme_minimal() +
  labs(title = "PCA of k-Means Clusters (3 Centers)", color = "Cluster")
```



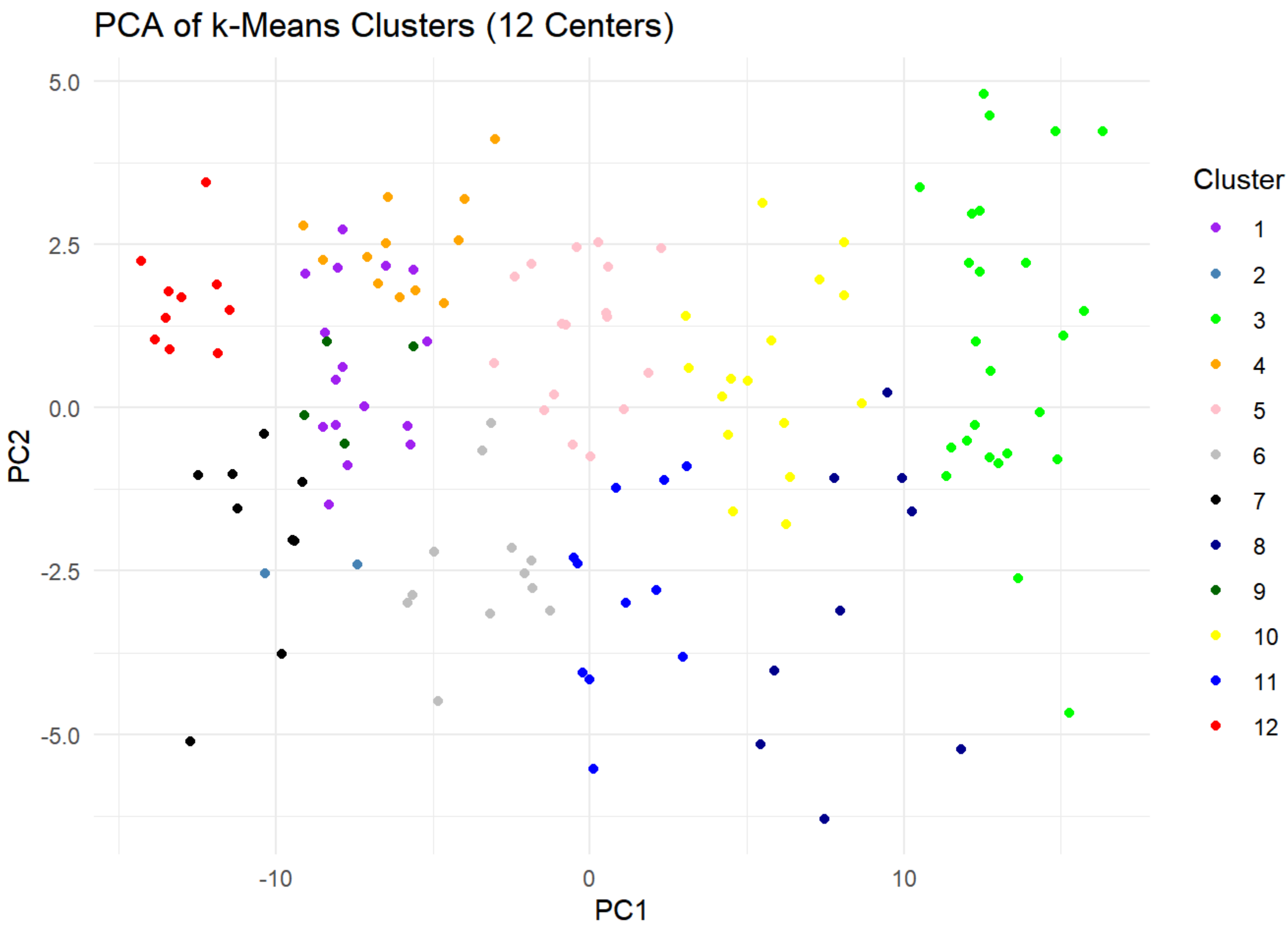
Cluster with 12 centers

```
#copy data into a new dataframe with 17 independent variables
cluster.data <- data_avg[,-c(1,2,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,1:14] <- scale(cluster.data[,c(1:14,16:17)])
```

```
## Warning in matrix(value, n, p): 数据长度[2496]不是矩阵列数[14]的整倍数
```

```
cluster.data <- na.omit(cluster.data)

#cluster with 12 centers (conjecturing that four groups will correspond to all 12 different con
ditions)
cl <- kmeans(cluster.data, centers = 12)

#copy cluster result into dataframe
plot_data <- na.omit(data_avg[,-c(22)])
plot_data$Cluster <- as.factor(cl$cluster)

plot_data$Obstacle <- as.factor(plot_data$Obstacle)
plot_data$Task <- as.factor(plot_data$Task)
```

```
# Perform PCA
pca_result <- prcomp(cluster.data, rank. = 2)  # reduce to 2 dimensions for visualization

loadings <- pca_result$rotation

# Add PCA results to the original data frame
plot_data$PC1 <- pca_result$x[, 1]
plot_data$PC2 <- pca_result$x[, 2]

# Use ggplot2 to create the PCA scatter plot colored by clusters
ggplot(plot_data, aes(x = PC1, y = PC2, color = as.factor(Cluster))) +
  geom_point() +
  scale_color_manual(values = c("purple", 'steelblue', 'green', 'orange','pink','grey','blac
k','darkblue','darkgreen','yellow','blue','red')) + # Customize colors if needed
  theme_minimal() +
  labs(title = "PCA of k-Means Clusters (12 Centers)", color = "Cluster")
```



Gaussian Mixture (GMM)

```
#copy data into a new dataframe with 17 independent variables
cluster.data <- data_avg[,-c(1,2,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,1:14] <- scale(cluster.data[,c(1:14,16:17)])
```

```
## Warning in matrix(value, n, p): 数据长度[2496]不是矩阵列数[14]的整倍数
```

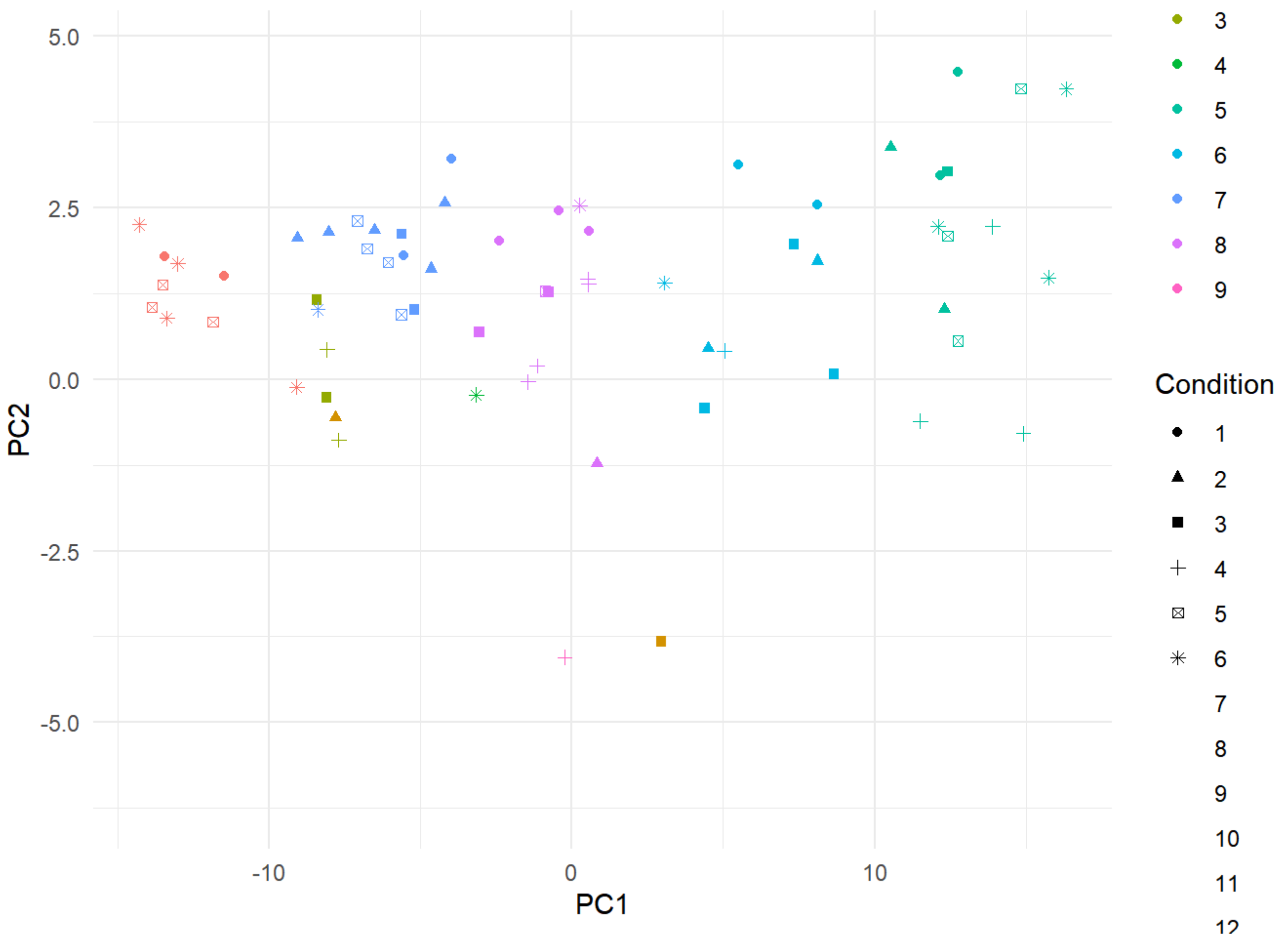
```
cluster.data <- na.omit(cluster.data)

gmm.result <- Mclust(cluster.data)
summary(gmm.result)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VII (spherical, varying volume) model with 9 components:
##
##  log-likelihood   n  df       BIC       ICL
##       -3261.592 143 170 -7366.868 -7369.637
##
## Clustering table:
##  1  2  3  4  5  6  7  8  9
## 16 12 12 13 27 19 21 19  4
```

```
pca_result <- prcomp(cluster.data, rank. = 2)

plot_data <- na.omit(data_avg[,-c(22)])
plot_data$Cluster <- as.factor(gmm.result$classification)

plot_data$Obstacle <- as.factor(plot_data$Obstacle)
plot_data$Task <- as.factor(plot_data$Task)

plot_data$PC1 <- pca_result$x[, 1]
plot_data$PC2 <- pca_result$x[, 2]

ggplot(plot_data, aes(x = PC1, y = PC2, color = factor(Cluster), shape = factor(Condition))) +
    geom_point() +
    theme_minimal() +
    labs(color = "Cluster", shape = "Condition")
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 13. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 77 rows containing missing values (`geom_point()`).
```



Random Forest

```
cluster.data <- data_avg[,-c(1,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,c(2:15,17:18)] <- scale(cluster.data[,c(2:15,17:18)])
cluster.data <- na.omit(cluster.data)
names(cluster.data) <- make.names(names(cluster.data))
rf_model <- randomForest(Condition ~ ., data = cluster.data)
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = Condition ~ ., data = cluster.data)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 11.61324
##                     % Var explained: 17.05
```

```
predictions <- predict(rf_model, newdata=cluster.data)
predictions
```

```
##         1         2         3         4         5         6         7         8 
##  2.750600  3.652233  4.691533  5.297267  5.145100  6.473600  7.551533  8.184500 
##         9        10        11        12        13        14        15        16 
##  8.864067  9.465067 10.569400 11.134033  9.411267  2.455500  2.795033  3.478200 
##        17        18        19        20        21        22        23        24 
##  4.382500  5.009500  5.717267  7.054400  7.971333  7.925267  9.409700 10.357833 
##        25        26        27        28        29        30        31        32 
## 11.188200 11.223033  3.227300  4.168667  4.522633  6.389833  5.725533  6.011867 
##        33        34        35        36        37        38        39        40 
##  7.582600  8.222433  9.348733  9.842167 10.242033 10.908467  9.435700  2.592567 
##        41        42        43        44        45        46        47        48 
##  3.253933  4.583400  6.060633  5.039867  5.009200  6.782067  6.956733  9.228333 
##        49        50        51        52        53        54        55        56 
##  9.487767  9.896967 11.330733 10.102233  2.647467  3.158933  3.704600  5.958767 
##        57        58        59        60        61        62        63        64 
##  4.628367  5.395400  6.868633  6.641833  8.577767  8.988367  9.761433  9.950967 
##        65        79        80        81        82        83        84        85 
## 10.642900  4.559767  2.556533  3.630000  5.149300  5.626100  5.777800  6.862000 
##        86        87        88        89        90        91        92        93 
##  8.192233  8.020133  9.290600 10.225500 11.093567  9.371867  3.306467  2.986267 
##        94        95        96        97        98        99       100       101 
##  3.694533  4.122133  5.283267  6.249467  7.958700  8.551000  9.215833  9.896133 
##       102       103       104       105       106       107       108       109 
##  9.680067 11.218867  8.852833  4.286767  3.620000  3.291067  4.261267  5.041933 
##       110       111       112       113       114       115       116       117 
##  5.511900  7.409767  7.036200  8.422733  7.693933  9.583367  9.483167 11.570900 
##       118       119       120       121       122       123       124       125 
##  3.332367  2.487100  3.525567  6.048233  5.413967  6.788300  6.328200  8.283767 
##       126       127       128       129       130       131       132       133 
##  8.622667  9.452533  9.504667 11.246733  9.374167  3.321367  4.179033  5.784500 
##       134       135       136       137       138       139       140       141 
##  5.755100  6.158467  6.021667  7.171433  7.335100  9.211733  9.429367  9.408467 
##       142       143       144       145       146       147       148       149 
## 10.369233 11.224233  2.250767  3.864167  3.908533  4.998133  5.630733  5.477900 
##       150       151       152       153       154       155       156 
##  8.136400  8.388833  9.015967  8.908767  9.829000 11.163833  9.932800
```

```
# Predicting on the same dataset used for training just as an example
predictions <- predict(rf_model, cluster.data)

predictions_rounded <- round(predictions)

# Assuming that Condition is a numeric column that represents integer classes
actual_values <- cluster.data$Condition
accuracy <- sum(predictions_rounded == actual_values) / length(actual_values)

# Calculate RMSE
rmse <- sqrt(mean((predictions - actual_values)^2))

accuracy
```

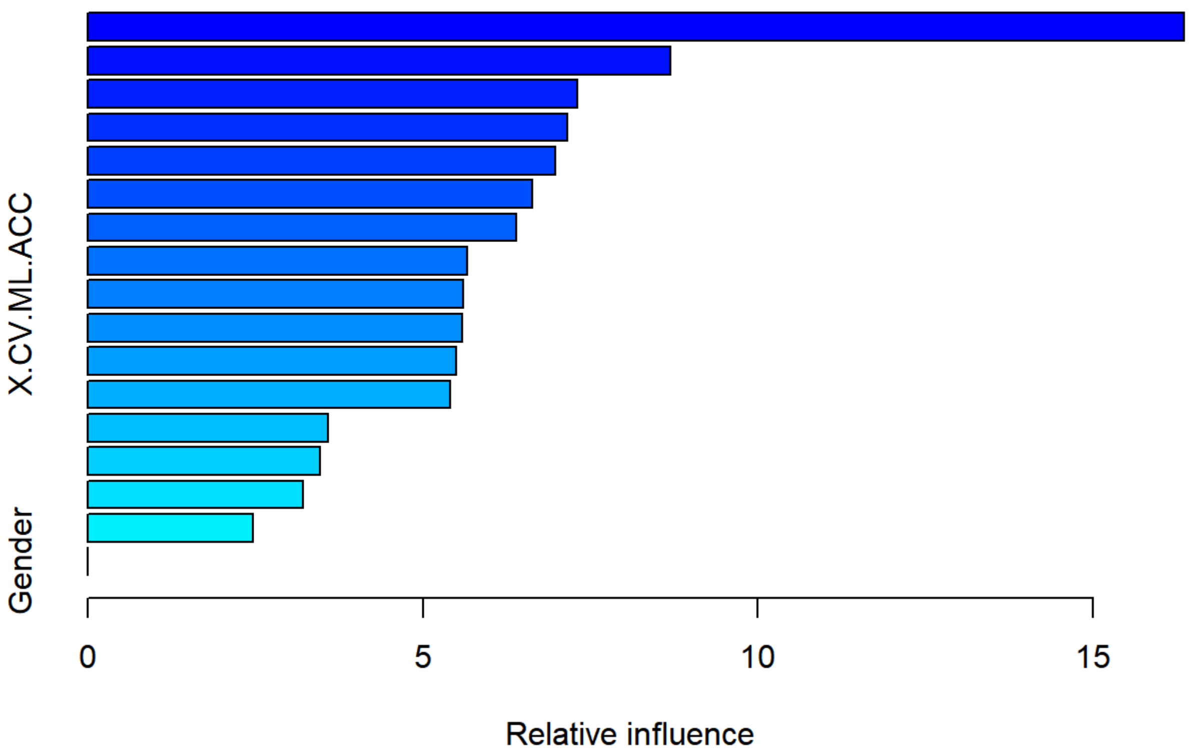
```
## [1] 0.3006993
```

GBM

```
cluster.data <- data_avg[,-c(1,18,19,22)]
cluster.data$Gender <- ifelse(data_avg$Gender == "female", 1, 0)
cluster.data[,c(2:15,17:18)] <- scale(cluster.data[,c(2:15,17:18)])
cluster.data <- na.omit(cluster.data)
names(cluster.data) <- make.names(names(cluster.data))

gbm_model <- gbm(Condition ~ ., data=cluster.data, distribution="gaussian", n.trees=100, interaction.depth=3)
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
## : variable 15: Gender has no variation.
```

```
summary(gbm_model)
```



```
##                                                              var   rel.inf
## X.CV.Stride.Time..sec..           X.CV.Stride.Time..sec.. 16.352851
## Step.Time..sec..                         Step.Time..sec..  8.690061
## Velocity..cm..sec..                   Velocity..cm..sec..  7.308770
## X.CV.Stride.Length..cm..         X.CV.Stride.Length..cm..  7.155209
## Total.D..Support..sec..           Total.D..Support..sec..  6.980389
## ML.ACC                                             ML.ACC  6.633434
## X.CV.Stride.Width..cm..           X.CV.Stride.Width..cm..  6.396592
## Stride.Width..cm..                     Stride.Width..cm..  5.661162
## X.CV.ML.ACC                                   X.CV.ML.ACC  5.596619
## X.CV.Total.D..Support..sec.. X.CV.Total.D..Support..sec..  5.586744
## X.CV.Step.Length..cm..             X.CV.Step.Length..cm..  5.495061
## Cadence..steps.min..                 Cadence..steps.min..  5.412927
## Stride.Time..sec..                     Stride.Time..sec..  3.588017
## Mean.Step.Length..cm..             Mean.Step.Length..cm..  3.469735
## Mean.Stride.Length..cm..         Mean.Stride.Length..cm..  3.209986
## X.CV.Step.Time..sec..               X.CV.Step.Time..sec..  2.462443
## Gender                                             Gender  0.000000
```

```
gbm_predictions <- predict(gbm_model, newdata=cluster.data, n.trees=100)



# Predicting on the same dataset used for training as an example
predicted_classes <- predict(gbm_model, newdata = cluster.data, n.trees = gbm_model$n.trees, ty
pe = "response")



predictions_rounded <- round(predicted_classes)

actual_classes <- cluster.data$Condition  # Make sure this is the correct column for actual cla
ss labels
accuracy <- mean(predictions_rounded == actual_classes)

# Predict using the GBM model
predictions <- predict(gbm_model, newdata = cluster.data, n.trees = gbm_model$n.trees)

# Round predictions to nearest integer
predictions_rounded <- round(predictions)

# Assuming that Condition is a numeric column that represents integer classes
actual_values <- cluster.data$Condition

# Calculate accuracy
accuracy <- sum(predictions_rounded == actual_values) / length(actual_values)

accuracy
```

```
## [1] 0.2727273
```

XGBoost

```
# Ensure that 'Condition' is a factor and get its levels as numeric values
num_classes = length(unique(cluster.data$Condition))
cluster.data$Condition <- as.numeric(as.factor(cluster.data$Condition)) - 1

# Check the range of 'Condition' to make sure it's within [0, num_class)
if(min(cluster.data$Condition) < 0 || max(cluster.data$Condition) >= num_classes) {
  stop("Labels are not within the correct range.")
}

# Update the DMatrix
data_matrix <- xgb.DMatrix(data = as.matrix(cluster.data[, -which(names(cluster.data) == "Condi
tion")]),
                           label = cluster.data$Condition)

# Update the parameters (make sure num_class is set correctly)
params <- list(
  objective = "multi:softprob",
  eval_metric = "mlogloss",
  max_depth = 6,
  eta = 0.3,
  num_class = num_classes,  # This should be the number of unique classes
  nthread = 2
)

# Train the model
xgb_model <- xgb.train(params = params,
                       data = data_matrix,
                       nrounds = 100,
                       watchlist = list(train = data_matrix),
                       verbose = 0)

# Make predictions
pred_probs <- predict(xgb_model, data_matrix)
num_data <- nrow(cluster.data)

# Reshape the prediction probabilities and find the class with the maximum probability
pred_classes <- matrix(pred_probs, nrow = num_data, byrow = TRUE)
predicted_labels <- max.col(pred_classes) - 1  # Subtract 1 because max.col is 1-indexed

# Actual labels (make sure these are zero-indexed as well)
actual_labels <- cluster.data$Condition

# Compute accuracy
accuracy <- mean(predicted_labels == actual_labels)

accuracy
```

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

This is the accuracy with full dataset's condition compared with predicted condition

```
# Example of simple train-test split
set.seed(123)  # for reproducibility
train_indices <- sample(1:nrow(cluster.data), 0.8 * nrow(cluster.data))
test_indices <- setdiff(1:nrow(cluster.data), train_indices)

train_data <- cluster.data[train_indices, ]
test_data <- cluster.data[test_indices, ]

train_matrix <- xgb.DMatrix(data = as.matrix(train_data[, -which(names(train_data) == "Conditio
n")]),
                            label = train_data$Condition)
test_matrix <- xgb.DMatrix(data = as.matrix(test_data[, -which(names(test_data) == "Conditio
n")]),
                           label = test_data$Condition)

# Train and evaluate the model using the train and test datasets
# Make predictions on the test set
test_predictions_probs <- predict(xgb_model, test_matrix)
num_test_data <- nrow(test_data)

# Convert probabilities to class predictions
test_pred_classes <- matrix(test_predictions_probs, nrow = num_test_data, byrow = TRUE)
test_predicted_labels <- max.col(test_pred_classes) - 1

# Actual test labels
test_actual_labels <- test_data$Condition

# Calculate accuracy
test_accuracy <- mean(test_predicted_labels == test_actual_labels)

# Print the accuracy
test_accuracy
```

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

This is the test accuracy that I split dataset into train and test. After using the training dataset to build the model, I used the testing dataset to test the model, and the accuracy is still 1.

```
importance_matrix <- xgb.importance(feature_names = colnames(cluster.data[, -which(names(cluste
r.data) == "Condition")]), model = xgb_model)
print(importance_matrix)
```

```
##                            Feature       Gain      Cover  Frequency
##  1:         X.CV.Step.Time..sec.. 0.21707731 0.16007118 0.10784678
##  2:       X.CV.Stride.Time..sec.. 0.18180175 0.12001083 0.09669022
##  3:      X.CV.Stride.Length..cm.. 0.08168487 0.07498220 0.06656750
##  4:       Total.D..Support..sec.. 0.07773192 0.08041892 0.08181480
##  5:        X.CV.Step.Length..cm.. 0.07220324 0.08130739 0.08144292
##  6: X.CV.Total.D..Support..sec.. 0.05379688 0.07357945 0.07809595
##  7:                        ML.ACC 0.04793346 0.04477820 0.04946077
##  8:              Step.Time..sec.. 0.04447844 0.05749738 0.06693938
##  9:       X.CV.Stride.Width..cm.. 0.03996408 0.07006837 0.08107103
## 10:      Mean.Stride.Length..cm.. 0.03873092 0.03557810 0.04090740
## 11:                   X.CV.ML.ACC 0.03549733 0.04824104 0.06098922
## 12:           Velocity..cm..sec.. 0.02881613 0.05097826 0.06359241
## 13:          Cadence..steps.min.. 0.02426086 0.02566701 0.02900707
## 14:            Stride.Time..sec.. 0.02406618 0.02643541 0.02603198
## 15:            Stride.Width..cm.. 0.01687812 0.02873749 0.03941986
## 16:        Mean.Step.Length..cm.. 0.01507852 0.02164875 0.03012272
```

CV Step Time, CV Stride Time are the features contributed the most to XGBoost’s performance.