Using SVMs to Classify Activities
Analysis

Tom Stuckey

17 Nov 2020

1 Setup Chunk

1.1 BE SURE TO SET "absolute_path_to_data_directory" variable !!!

```
absolute_path_to_data_directory <- "/Users/thomasstuckey/Google\ Drive/personal/Learning/Masters/2020\ C

knitr::opts_knit$set(root.dir = absolute_path_to_data_directory)
rm(list=ls())
library(roxygen2)
library(tidyverse)
library(e1071)
library(gridExtra)
library(kableExtra)
set.seed(888)
```

2 Load and Do Initial Data Prep

```
#' Load all the data for the specified file pattern
#'
#' @param file_path
#' @param col_to_bindA
#' @param col_to_bindB
load_the_data <-function(file_path, col_to_bindA, col_to_bindB){
  # Use OS-level file matching to get all the files that match the given file pattern
  the_data <- lapply(Sys.glob(file_path), read_csv, col_names = FALSE)
  the_data <- dplyr::bind_rows(the_data)
  new_colA <- rep(col_to_bindA, nrow(the_data))
  new_colB <- rep(col_to_bindB, nrow(the_data))
  the_data <- cbind(the_data, new_colA)
  the_data <- cbind(the_data, new_colB)
  names(the_data) <- c("time", "bending", "head_to_toe", "left_to_right", "sensor_id", "signal_strength
  return(the_data)
}

# Load the data
room1_men_data <- load_the_data("S1_Dataset/*M", "one", "male") #room one has four RFID sensors
room2_men_data <- load_the_data("S2_Dataset/*M", "two", "male") #room one has three RFID sensors
room1_wommen_data <- load_the_data("S1_Dataset/*F", "one", "female") #room one has four RFID sensors
room2_wommen_data <- load_the_data("S2_Dataset/*F", "two", "female") #room one has three RFID sensors
```

```r
# Combine the data
all_data <- dplyr::bind_rows(room1_men_data, room2_men_data, room1_wommen_data, room2_wommen_data)
rm(room1_men_data, room2_men_data, room1_wommen_data, room2_wommen_data) #free up some memory
all_data <- dplyr::as_tibble(all_data)

# Relocate the activity class to the end
all_data <- all_data %>% dplyr::relocate("activity_class", .after = last_col())
```

3 Do Stratified Sampling

Here we do stratified sampling so that we will have **num_folds** groups of data that all have the same stratification of activity_class a the original data. This enables to do rudimentary cross-validation between each stratified group.

```r
num_folds <- 5 #let's use five fold cross validation

#' Get Stratified Samples
#' @param my_data data handle
#' @param num_folds number of data folds
#' @return stratified sample
get_stratified_df <- function(my_data, num_folds){
  my_prop <-  num_folds / 100
  # group by the activity_class feature and add a temp column for the number of rows for each activity
  # then sample in the proportion that will give the the desired number of
  # stratified sub-datasets
  stratified_sample <- my_data %>%
    group_by(activity_class) %>%  mutate( num_rows = n() ) %>%
    dplyr::slice_sample(prop = my_prop, weight_by = num_rows, replace = FALSE) %>% ungroup
  stratified_sample <- stratified_sample %>% select( -num_rows ) #drop the tmp weighting row
  return(stratified_sample)
}

# Create and load the stratified data frames into a list
my_strat_dfs = list()
for (i in 1:num_folds){
  my_strat_dfs[[i]] <- get_stratified_df(all_data, num_folds = num_folds)
}

# Save one source row to demonstrate against at the end
sample_demonstration_row <- all_data %>% dplyr::slice_sample(n = 1)
```

4 Create Data Partitions in a One-vs-All Fashion and Rebundle

Aligned with the stratified sampling, we have five groups to analyze. Let's use filtering to facilitate this. Further, let's make this easier on ourselves by doing some one-vs-all relabeling and then by reorganizing back into a single dataframe we can easily sort for analysis.

```r
#' Utility to make the classes strictly positive/negative for the particular test
#' @param t_data
#' @param positive_activity
#' @return
make_pos_neg_only <- function(t_data, pos_act){
```

```
  act_pos <- t_data %>% dplyr::filter(activity_class == pos_act)
  act_neg <- t_data %>% dplyr::filter(activity_class != pos_act)

  # Make the subsets have 1 for the positive activity class and 0 for the negative activity class
  act_pos <- act_pos %>% dplyr::mutate(activity_class = 1)
  act_neg <- act_neg %>% dplyr::mutate(activity_class = 0)
  return_df <- rbind(act_pos, act_neg)
  return(return_df)
}

# create list of the unique activity classes
activities <- unique(all_data$activity_class)

# Make a new tibble with just the column headers
revised_data <- all_data[0,]

# Add some admin columns to facilitate filtering and processing later
revised_data <- revised_data %>% add_column("pos_activity", .before = 1) %>% add_column("strat_group",

# For each folds(stratified group) go through each activity and create
# a permutation data frame that activity is the positive activity and all other activities are the nega
# then add that data frame into an overall revised data frame that we can
# filter as needed
for (i in 1:num_folds){
  for (act in activities){
    tmp_df <- make_pos_neg_only(my_strat_dfs[[i]], act)
    tmp_df <- tmp_df %>% add_column("pos_activity" = act, .before = 1) %>% add_column("strat_group" = i
    revised_data <- rbind(revised_data, tmp_df)
  }
}

# We're done with the my_strat_df staging and the tmp_df so let's free up the memory
rm(all_data, my_strat_dfs, tmp_df)

# Make the activity class a factor before we move into classification efforts
revised_data$activity_class <- as.factor(revised_data$activity_class)
```

5 Build SVM models, Make Predictions, and Caclculate Accuracy

```
# Factor for training
train_factor <- 0.7
my_gamma <- 5 # determines how quickly the class boundaries dissipate further from the actual support v
my_cost <- 25 # cost of misclassifying; aka the regularization term in the LaGrange multiplier

act_enum <- c("Sitting on Bed", "Sitting on Chair", "Laying", "Walking")

#' Split Dataframe into train and test
#' @param t_train_factor
#' @param df
#'
#' @return vector of train_df and test_df
#' @export
```

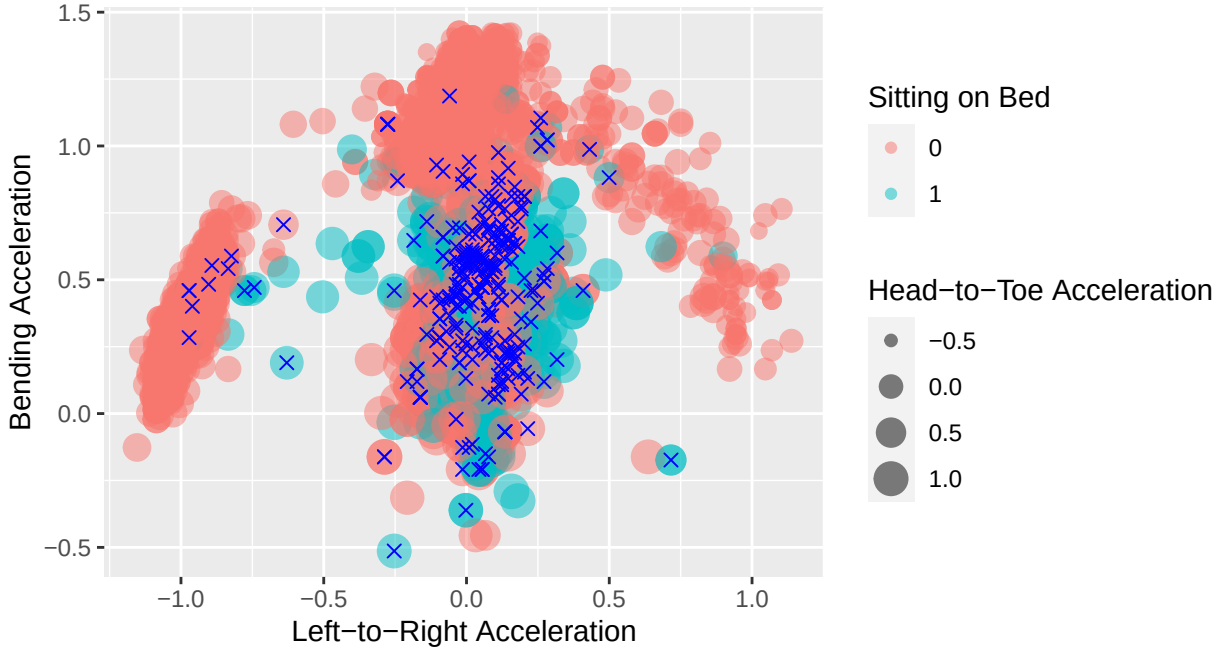
```r
split_data <- function(my_df, t_train_factor){
  # Split into Training and testing Data
  training_mask <- sample(seq_len(nrow(my_df)), size = t_train_factor * nrow(my_df))
  train_df <- my_df[training_mask,]
  test_df <- my_df[-training_mask,]
  t_dfs = list()
  t_dfs[[1]] <- train_df
  t_dfs[[2]] <- test_df
  return(t_dfs)
}

#' Compute Accuracy
#' @param t_predicitions
#' @param t_actual
#'
#' @return measure of accuracy
compute_accuracy <- function(t_pred, t_actual){
   # Do some cross tabulation with the 2-way table so we can read off the values of the confusion matri
   tmp_table <- table(t_pred, t_actual)

   true_neg <- tmp_table[1]
   false_neg <- tmp_table[2]
   false_pos <- tmp_table[3]
   true_pos <- tmp_table[4]

   # Accuracy = (TP+TN)/(P+N)
   return( (true_pos + true_neg) / (true_pos + false_neg + true_neg + false_pos ))
} # end compute_accuracy

#' Build a plot
#' @param t_predictions
#' @param test_df
#' @param t_accuracy
#' @param t_strat_group
#' @return the plot
build_plot <- function(test_df, t_act, t_pred, t_accuracy, t_strat_group){
  #Create a vector with just T/F for classifications to help with graphics
  gen_classification <- t_pred == test_df$activity_class

  # Create a subset of the test dataframe that were incorrectly classified
  error_df <- test_df[!gen_classification,]

  a_plot <- ggplot(data=test_df, aes(x = left_to_right, y = bending, size = head_to_toe, color = activi
  a_plot <- a_plot + geom_point(alpha = 0.5)
  a_plot <- a_plot + labs(title = paste0("Predictions for positive class:  ", t_act),
                          subtitle = paste0("\nAccuracy was: ",round(t_accuracy,5) * 100,"%",
                                            "\nBest stratification group was: ", t_strat_group),
                          size = 'Head-to-Toe Acceleration', color = t_act)+
    xlab("Left-to-Right Acceleration")+
    ylab("Bending Acceleration")

  # Put X's on the points that were misclassified in the test data
  a_plot <- a_plot + geom_point(data = error_df, aes(x = left_to_right, y = bending), shape=4, color='bl
```

```
    guides(col = guide_legend(order = 1))
  return(a_plot)
} # end build_plot

# Some initializations for results
my_plot_list <- list()
result_df <- tibble(activity=character(0), strat_group=numeric(0) , best=character(0))
final_model_per_act <- list()

# Loop through the activities on the outer-loop and the stratification groups on the inner loop
# This allows us to more easily find the best model for each activity across the stratified groups
for (act in activities){
  act_accuracy <- vector()
  act_predictions <- list()

  for (strat_group in 1:num_folds){
    # Filter the data for the stratification group and the positive activity
    t_data <- revised_data %>% dplyr::filter(strat_group == strat_group, pos_activity == act )
    train_test_df_list <- split_data(t_data, train_factor)
    train_df <- train_test_df_list[[1]]
    test_df <- train_test_df_list[[2]]

    #Train the model
    t_model <- e1071::svm(activity_class ~ bending + head_to_toe + left_to_right + signal_strength,
                          data = train_df,
                          kernel = 'radial', gamma = my_gamma, cost = my_cost, scale = FALSE)

    # Do predictions with the model
    tmp_predicitions <- predict(t_model, test_df)
    act_predictions[[strat_group]] <- list(test_df, tmp_predicitions, t_model) #store the test data, th

    # Compute accuracy
    tmp_accuracy <- compute_accuracy(t_pred = tmp_predicitions, t_actual = test_df$activity_class )
    act_accuracy <- c(act_accuracy, tmp_accuracy) # just add to the vector of accuracies
    result_df[nrow(result_df)+1,] <- list(act_enum[act], strat_group, paste0(round(tmp_accuracy,5) * 10
  } # end for strat_group loop

  # Find out which has the best accuracy for this activities across all the stratified groups, do predi
  index_of_max <- which.max(act_accuracy) # get the index of our most accurate model for the active act
  final_model_per_act[[act]] <- act_predictions[[index_of_max]][[3]]
  my_plot_list[[act]] <- build_plot(test_df = act_predictions[[index_of_max]][[1]], t_act = act_enum[ac
                                    t_pred = act_predictions[[index_of_max]][[2]],  t_accuracy = act_ac
} # end for activities loop

ml <- marrangeGrob(grobs = my_plot_list, nrow = 1, ncol = 1)
print(ml)
```

page 1 of 4

Predictions for positive class: Sitting on Bed

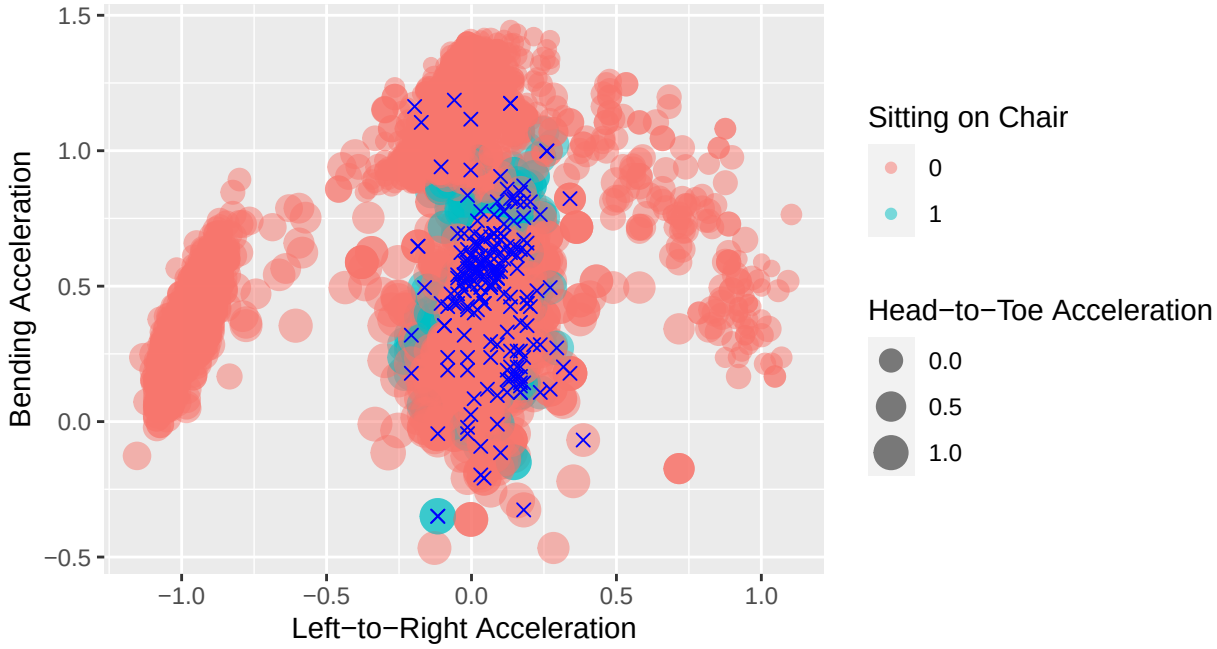
Accuracy was: 94.71%
Best stratification group was: 5



page 2 of 4

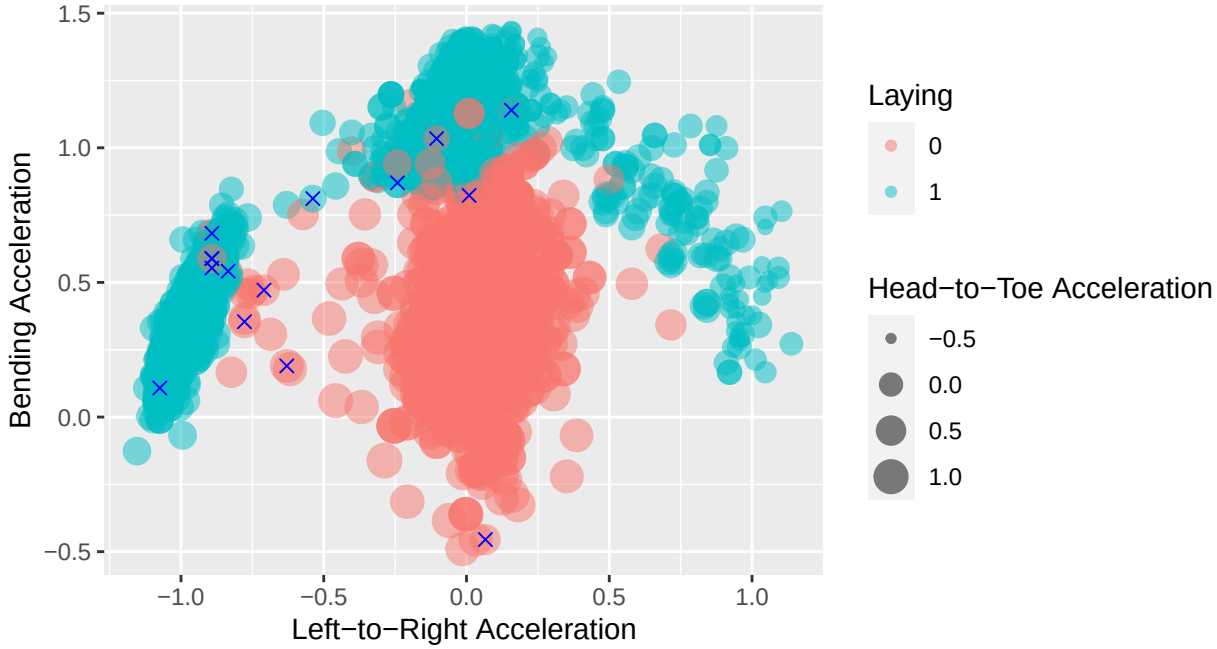
Predictions for positive class: Sitting on Chair

Accuracy was: 96.112%
Best stratification group was: 5



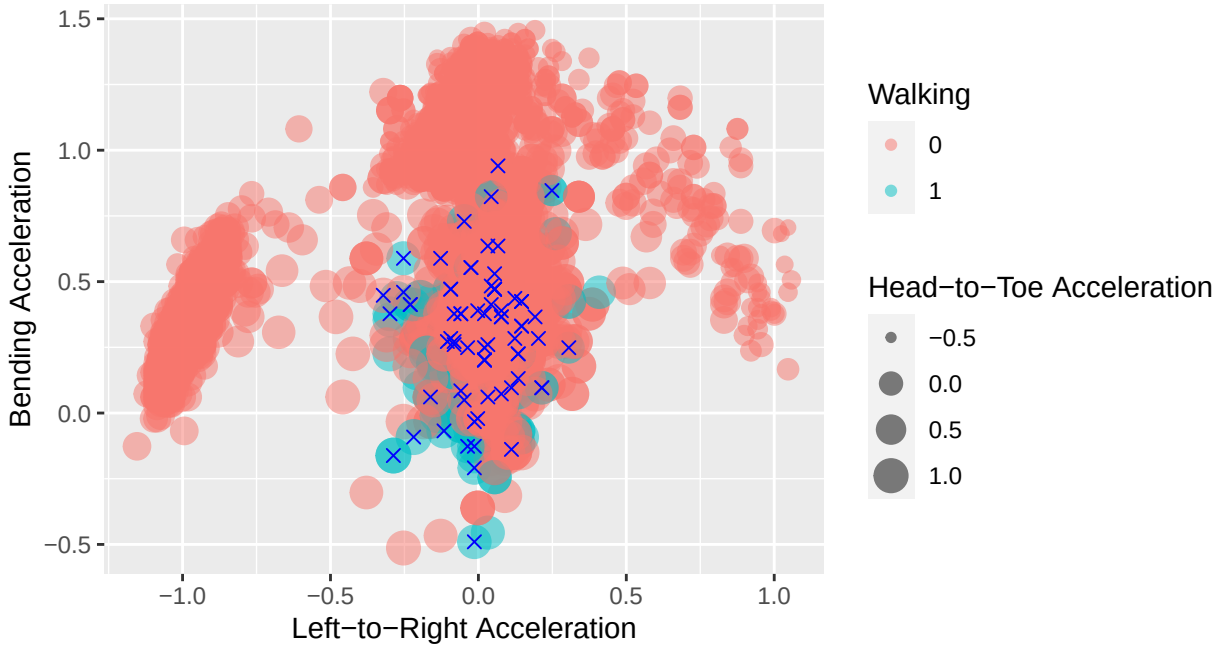
Predictions for positive class: Laying

Accuracy was: 99.734%
Best stratification group was: 2



Predictions for positive class: Walking

Accuracy was: 98.722%
Best stratification group was: 4



```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

```
all_results <- knitr::kable(result_df, align = 'c', format ='simple')
print(all_results)
```

```
##
##
##      activity        strat_group     best
## ------------------  -------------  ---------
##   Sitting on Bed          1         94.532%
##   Sitting on Bed          2         94.426%
##   Sitting on Bed          3         94.142%
##   Sitting on Bed          4         94.568%
##   Sitting on Bed          5         94.71%
##       Laying              1         99.627%
##       Laying              2         99.734%
##       Laying              3         99.645%
##       Laying              4         99.663%
##       Laying              5         99.645%
##      Walking              1         98.296%
##      Walking              2         98.42%
##      Walking              3         98.242%
##      Walking              4         98.722%
##      Walking              5         98.385%
##  Sitting on Chair         1         96.023%
##  Sitting on Chair         2         96.059%
##  Sitting on Chair         3         95.651%
##  Sitting on Chair         4         95.899%
##  Sitting on Chair         5         96.112%
```

6 Example of how this could be used on in source data

```
# Run it through all the models in a nested fashion
cat("For actual:", act_enum[sample_demonstration_row$activity_class], "with attributes:\n")
```

```
## For actual: Laying with attributes:
```

```
sample_row <- knitr::kable(sample_demonstration_row, align = 'c', format ='simple')
print(sample_demonstration_row)
```

```
## # A tibble: 1 x 11
##    time bending head_to_toe left_to_right sensor_id signal_strength phase
##   <dbl>   <dbl>       <dbl>         <dbl>     <dbl>           <dbl> <dbl>
## 1  136.   0.249       0.331         -1.02         2             -49  2.49
## # ... with 4 more variables: frequency <dbl>, location <chr>, gender <chr>,
```

```
## #   activity_class <dbl>
```

```
for (act in activities){
    a_prediction <- predict(final_model_per_act[[act]], sample_demonstration_row)
    if (a_prediction == 1){
      cat("\t predicted WAS", act_enum[act],"\n")
    }else{
      cat("\t predicted WAS NOT", act_enum[act],"\n")
    }
}
```

```
##   predicted WAS NOT Sitting on Bed
##   predicted WAS Laying
##   predicted WAS NOT Walking
##   predicted WAS NOT Sitting on Chair
```