Project-AlGhammari-Schmidt

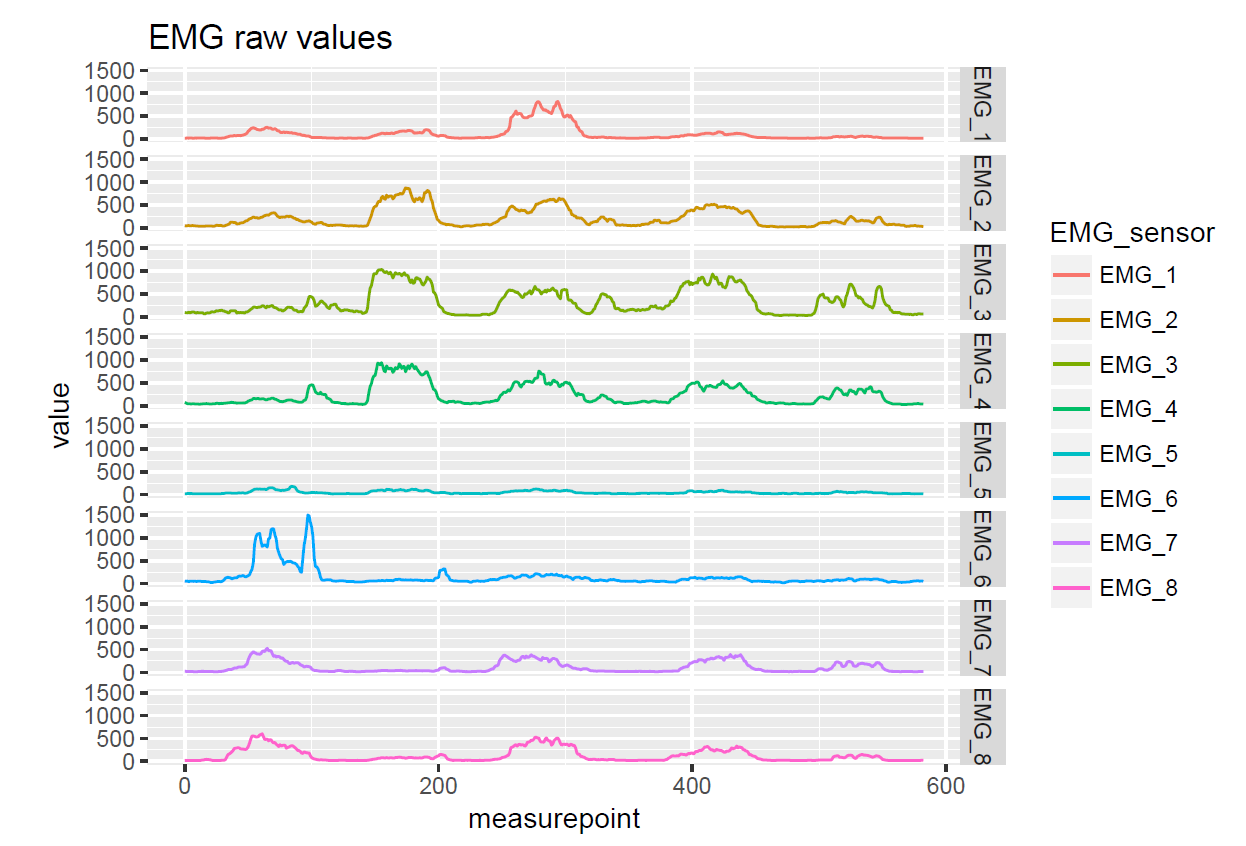
Bernd Schmidt , Osamah Al-Ghammari

January 8, 2017

require(plyr)  
require(corrplot)  
require(caret)  
require(doMC)  
registerDoMC(3)  
require(zoo)

# Acquiring data and data recording

The MYO armband offers 8 values (one for each sensor) of EMG data. Those values are not preprocessed and also not sorted according to the orientation of the armband. The gestures were performed in the following order: Fist - Spread Fingers - Wave Out - Wave In - Double Tap. According to the obtained raw data, the raw values are not very useable to detect gestures or movements.



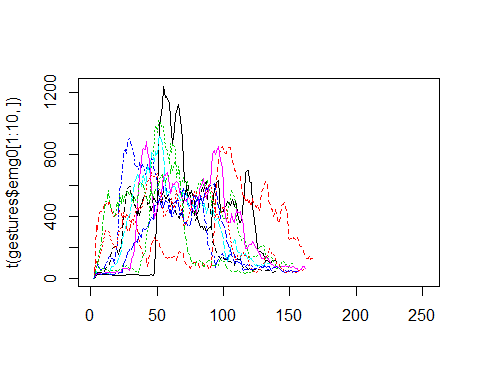
image

The following code allows to read the recorded unprocessed data from the file provided by the MYO Armband.

filedir <- '../dat'  
  
filenames <- list.files(filedir, full.names = T, pattern="\*.csv")  
  
gestures <- list()  
gestures$emg0 <- ldply(filenames[1], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg1 <- ldply(filenames[2], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg2 <- ldply(filenames[3], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg3 <- ldply(filenames[4], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg4 <- ldply(filenames[5], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg5 <- ldply(filenames[6], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg6 <- ldply(filenames[7], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))  
gestures$emg7 <- ldply(filenames[8], read.table, sep=',', fill = T, col.names = c('gesture', 'person', 'sample', paste('emg', 1:250, sep='')))

The following plot shows the un-usable raw values that need to be processed.

matplot(t(gestures$emg0[1:10,]), type='l')



# Data Preprocessing

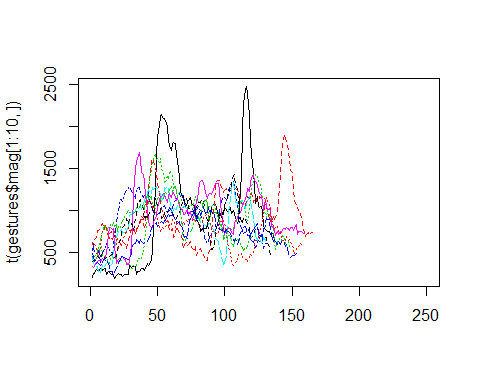
## Calculating the variance threshhold

To detect the start and the end of each gesture, the variance over a specific window is calculated. If the value is greater than the threshold, the signal will be cut. For this purpose we conducted the following steps:

### Magnitude over all 8 EMG values

Initially, We will utilize the magnitude of all eight EMG values. The following plot shows the 10 first magnitude values before processing.

gestures$mag <- as.data.frame(mapply(function(emg0, emg1, emg2, emg3, emg4, emg5, emg6, emg7) {  
 m <- sqrt(emg0^2 + emg1^2 + emg2^2 + emg3^2 + emg4^2 + emg5^2 + emg6^2 + emg7^2)  
 m  
}, gestures$emg0[,4:253], gestures$emg1[,4:253], gestures$emg2[,4:253], gestures$emg3[,4:253], gestures$emg4[,4:253], gestures$emg5[,4:253], gestures$emg6[,4:253], gestures$emg7[,4:253]))  
matplot(t(gestures$mag[1:10,]), type='l')



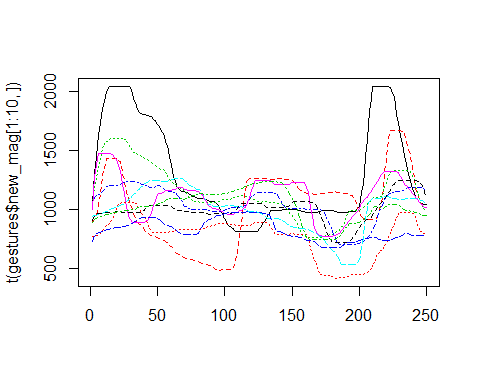
### Calculate variance using the 25% quantile

Data still needs more definition. Therefore, we need to calculate variance using the 25% quantile to define the necessary peaks for each gesture and discard all values below this quanitle.

gestures$q25 <- as.data.frame(apply(gestures$mag, 1 ,function(m) {  
 q25 <- quantile(m, na.rm = T)[[3]]  
 q25  
}))

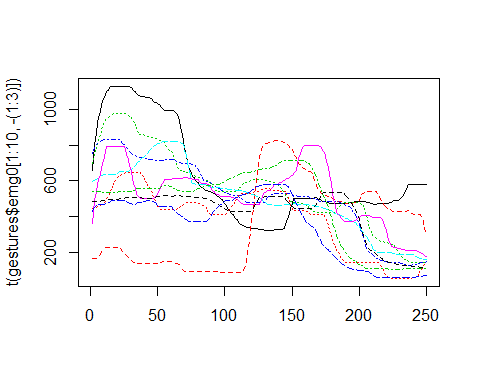
### Stripping the data for all values below this quanitle

stepwidth <- 1/250  
gestures$new\_mag <- data.frame(matrix(NA, ncol = 250))  
gestures$data <- data.frame(matrix(NA, ncol = 2003))  
for (i in 1:(dim(gestures$mag)[1])) {  
 # 1 remove NAs  
 mag <- gestures$mag[i,]  
 mag[is.na(mag)] <- 0  
 # 2 pre filter  
 mag <- rollapply(data = mag[!is.na(mag)], width = 10, FUN = median)  
 # 3 cut  
 start <- which.max(mag > gestures$q25[i,])  
 stop <- length(mag) - which.max(rev(mag) > gestures$q25[i,])  
 mag <- mag[start:stop]  
 start <- start + 3  
 stop <- stop + 3  
 emg0 <- gestures$emg0[i,start:stop]  
 emg1 <- gestures$emg1[i,start:stop]  
 emg2 <- gestures$emg2[i,start:stop]  
 emg3 <- gestures$emg3[i,start:stop]  
 emg4 <- gestures$emg4[i,start:stop]  
 emg5 <- gestures$emg5[i,start:stop]  
 emg6 <- gestures$emg6[i,start:stop]  
 emg7 <- gestures$emg7[i,start:stop]  
 # remove NAs and apply runmed  
 emg0 <- rollapply(data = emg0[!is.na(emg0)], width = 10, FUN = median)  
 emg1 <- rollapply(data = emg1[!is.na(emg1)], width = 10, FUN = median)  
 emg2 <- rollapply(data = emg2[!is.na(emg2)], width = 10, FUN = median)  
 emg3 <- rollapply(data = emg3[!is.na(emg3)], width = 10, FUN = median)  
 emg4 <- rollapply(data = emg4[!is.na(emg4)], width = 10, FUN = median)  
 emg5 <- rollapply(data = emg5[!is.na(emg5)], width = 10, FUN = median)  
 emg6 <- rollapply(data = emg6[!is.na(emg6)], width = 10, FUN = median)  
 emg7 <- rollapply(data = emg7[!is.na(emg7)], width = 10, FUN = median)  
 # 4 approx  
 mag\_approx <- approx(x = seq(0,1,1/(length(mag)-1)), y = mag, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
   
 emg0\_approx <- approx(x = seq(0,1,1/(length(emg0)-1)), y = emg0, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg1\_approx <- approx(x = seq(0,1,1/(length(emg1)-1)), y = emg1, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg2\_approx <- approx(x = seq(0,1,1/(length(emg2)-1)), y = emg2, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg3\_approx <- approx(x = seq(0,1,1/(length(emg3)-1)), y = emg3, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg4\_approx <- approx(x = seq(0,1,1/(length(emg4)-1)), y = emg4, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg5\_approx <- approx(x = seq(0,1,1/(length(emg5)-1)), y = emg5, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg6\_approx <- approx(x = seq(0,1,1/(length(emg6)-1)), y = emg6, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 emg7\_approx <- approx(x = seq(0,1,1/(length(emg7)-1)), y = emg7, xout = seq(0,1,stepwidth), method = 'linear')$y[1:250]  
 # 5 filter  
   
 # 6 append to one line  
 gestures$new\_mag[i,] <- as.data.frame(t(mag\_approx))  
 gestures$emg0[i,4:253] <- as.data.frame(t(emg0\_approx))  
 gestures$emg1[i,4:253] <- as.data.frame(t(emg1\_approx))  
 gestures$emg2[i,4:253] <- as.data.frame(t(emg2\_approx))  
 gestures$emg3[i,4:253] <- as.data.frame(t(emg3\_approx))  
 gestures$emg4[i,4:253] <- as.data.frame(t(emg4\_approx))  
 gestures$emg5[i,4:253] <- as.data.frame(t(emg5\_approx))  
 gestures$emg6[i,4:253] <- as.data.frame(t(emg6\_approx))  
 gestures$emg7[i,4:253] <- as.data.frame(t(emg7\_approx))  
   
 gestures$data[i,1:253] <- gestures$emg0[i,]  
 gestures$data[i,254:503] <- gestures$emg1[i,4:253]  
 gestures$data[i,504:753] <- gestures$emg2[i,4:253]  
 gestures$data[i,754:1003] <- gestures$emg3[i,4:253]  
 gestures$data[i,1004:1253] <- gestures$emg4[i,4:253]  
 gestures$data[i,1254:1503] <- gestures$emg5[i,4:253]  
 gestures$data[i,1504:1753] <- gestures$emg6[i,4:253]  
 gestures$data[i,1754:2003] <- gestures$emg7[i,4:253]  
}  
matplot(t(gestures$new\_mag[1:10,]), type='l')



The plot above shows the new magnitude for the first 10 values after processing the whole data. Whereas, the following plot shows the usable data values.

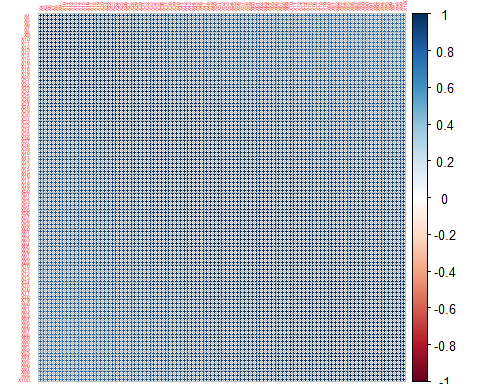
matplot(t(gestures$emg0[1:10,-(1:3)]), type='l')



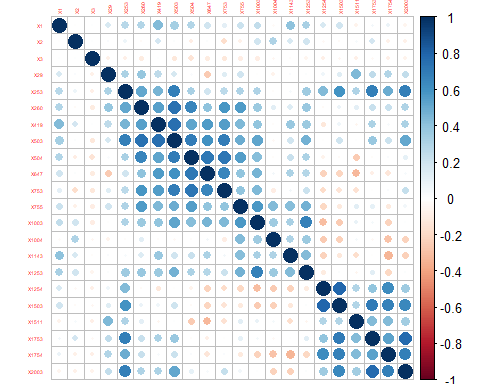
## Data Validation and Optimization

### Feature Correlation

# feature correlation as plot  
corrplot(cor(gestures$data[,4:100]), tl.cex = 0.3) # addgrid.col = NA



# remove correlated variable using ?findCorrelation  
foundCorIndexes <- findCorrelation(cor(gestures$data))  
#foundCorIndexes  
corrplot(cor(gestures$data[,-foundCorIndexes]), tl.cex = 0.3)



# remove the features from the data  
gestures$data <- gestures$data[,-foundCorIndexes]

## Data Partitioning

# split into training and test data  
set.seed(1704)  
indexes\_train <- createDataPartition(gestures$emg0[,1], p=0.75, list = F)  
indexes\_test <- (1:nrow(gestures$data))[-indexes\_train]  
  
training <- gestures$data[indexes\_train,]  
training\_gest <- gestures$emg0[indexes\_train,1]  
testing <- gestures$data[indexes\_test,]  
testing\_gest <- gestures$emg0[indexes\_test,1]

## Feature Selection

We tried to use feature selection, but it outputted less samples which resulted with poor training and testing accuracies. So, we had to skip feature selection.

#sbfRes <- sbf(x = training, y = training\_gest, sbfControl = sbfControl(functions = rfSBF, method = 'repeatedcv', repeats = 5)) # more repeats are better  
#sbfRes  
#sbfRes$optVariables  
#gestures\_opt$data <- gestures\_opt$data[,sbfRes$optVariables]

# Model Training

models <- list()

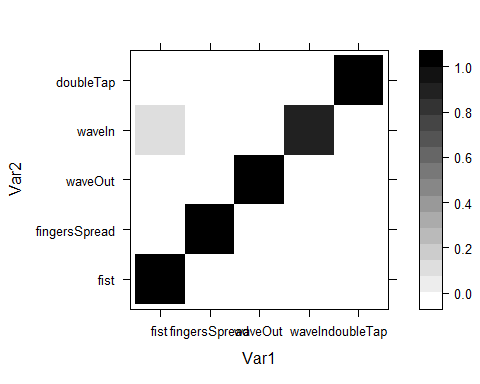
trControl <- trainControl(  
 method = 'repeatedcv', # none, cv, repeatedcv, LOOCV, ...  
 number = 10, # nr of CV partitions  
 repeats = 20, # nr of partitioning repetitions  
 returnData = F,   
 # classProbs = T, # enable computation of class probabilities?  
 # summaryFunction = twoClassSummary, # use when classifying two classes   
 returnResamp = 'final', # return CV partition results for best model  
 allowParallel = T  
)  
  
#trControl <- trainControl(  
# method = 'LOOCV',  
# preProcOptions = list(thresh = 0.9),  
# returnResamp = 'final',  
# returnData = F,  
# savePredictions = T,  
# allowParallel = T  
# )

## KNN

models$knn <- train(training,  
 factor(training\_gest),  
 method = 'knn',  
 preProcess = c('center', 'scale', 'pca'),  
 metric = 'Kappa',  
 trControl = trControl  
 )  
models$knn

## k-Nearest Neighbors   
##   
## Pre-processing: centered (22), scaled (22), principal component  
## signal extraction (22)   
## Resampling: Cross-Validated (10 fold, repeated 20 times)   
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.9476667 0.9345833  
## 7 0.9323333 0.9154167  
## 9 0.9100000 0.8875000  
##   
## Kappa was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

predicted <- predict(models$knn, newdata = testing)  
  
# to ensure, that also when one level is not predicted, the results can be displayed  
u = union(predicted, testing\_gest)  
t = table(factor(predicted, u), factor(testing\_gest, u))  
conf <- confusionMatrix(t)  
  
levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100:0/100))



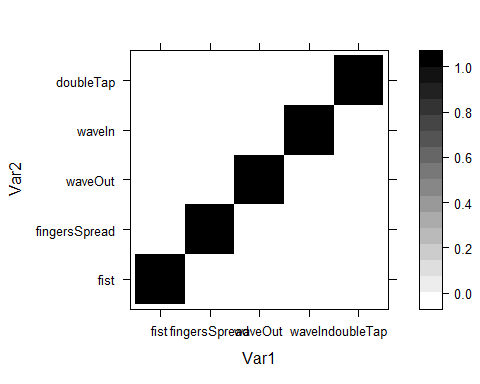
## LDA

To compare the results, now a *lda* model with the same parameters is trained.

models$lda <- train(training,  
 factor(training\_gest),  
 method = 'lda',  
 preProcess = c('center', 'scale', 'pca'),  
 metric = 'Kappa',  
 trControl = trControl  
 )  
models$lda

## Linear Discriminant Analysis   
##   
## Pre-processing: centered (22), scaled (22), principal component  
## signal extraction (22)   
## Resampling: Cross-Validated (10 fold, repeated 20 times)   
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.995 0.99375

predicted <- predict(models$lda, newdata = testing)  
  
# to ensure, that also when one level is not predicted, the results can be displayed  
u = union(predicted, testing\_gest)  
t = table(factor(predicted, u), factor(testing\_gest, u))  
conf <- confusionMatrix(t)  
  
levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100:0/100))

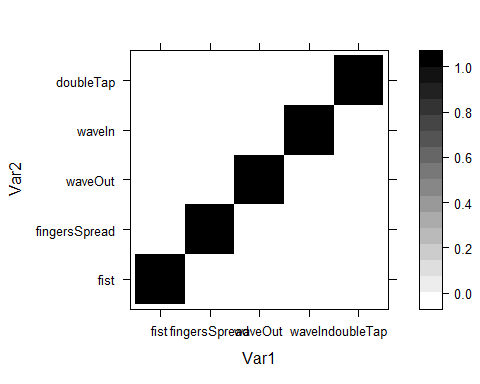


## LDA2

models$lda2 <- train(training,  
 factor(training\_gest),  
 method = 'lda2',  
 preProcess = c('center', 'scale', 'pca'),  
 metric = 'Kappa',  
 trControl = trControl  
 )  
models$lda2

## Linear Discriminant Analysis   
##   
## Pre-processing: centered (22), scaled (22), principal component  
## signal extraction (22)   
## Resampling: Cross-Validated (10 fold, repeated 20 times)   
## Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...   
## Resampling results across tuning parameters:  
##   
## dimen Accuracy Kappa   
## 1 0.9720000 0.9650000  
## 2 0.9946667 0.9933333  
## 3 0.9943333 0.9929167  
## 4 0.9953333 0.9941667  
##   
## Kappa was used to select the optimal model using the largest value.  
## The final value used for the model was dimen = 4.

predicted <- predict(models$lda2, newdata = testing)  
  
# to ensure, that also when one level is not predicted, the results can be displayed  
u = union(predicted, testing$pers)  
t = table(factor(predicted, u), factor(testing\_gest, u))  
conf <- confusionMatrix(t)  
  
levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100:0/100))



## SVM

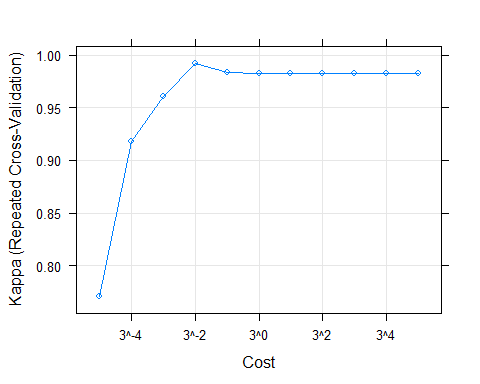
train\_model <- function(method, tuneGrid=NULL) {  
 train(x = training, # in real life apps only use train data here!  
 y = training\_gest, # in real life apps only use train data here!  
 method = method,   
 metric = 'Kappa',   
 tuneGrid = tuneGrid,  
 trControl = trControl  
 )  
}  
models$svmLinear <- train\_model('svmLinear', tuneGrid = expand.grid(C=3\*\*(-5:5)))

##   
## Attaching package: 'kernlab'

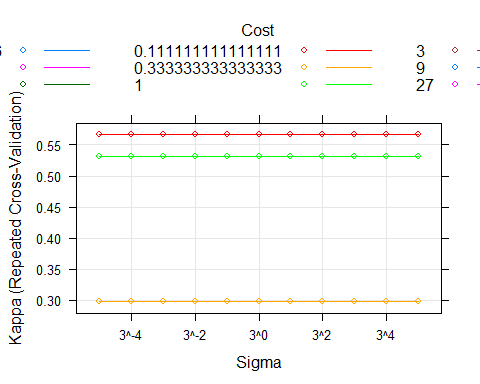
## The following object is masked from 'package:ggplot2':  
##   
## alpha

models$svmRadial <- train\_model('svmRadial', tuneGrid = expand.grid(C=3\*\*(-5:5), sigma=3\*\*(-5:5)))

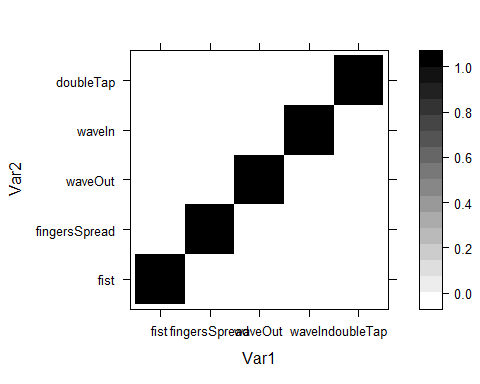
print(plot(models$svmLinear, scales=list(x=list(log=3))))



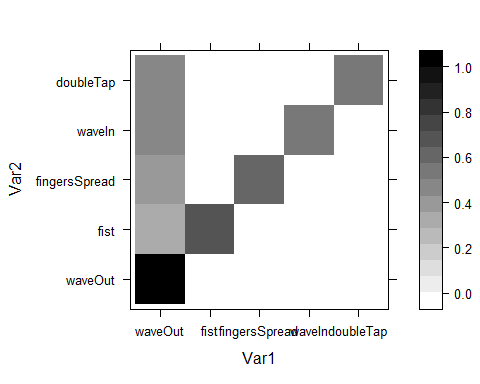
print(plot(models$svmRadial, scales=list(x=list(log=3))))



predicted <- predict(models$svmLinear, newdata = testing)  
  
# to ensure, that also when one level is not predicted, the results can be displayed  
u = union(predicted, testing$pers)  
t = table(factor(predicted, u), factor(testing\_gest, u))  
conf <- confusionMatrix(t)  
  
levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100:0/100))



predicted <- predict(models$svmRadial, newdata = testing)  
  
# to ensure, that also when one level is not predicted, the results can be displayed  
u = union(predicted, testing$pers)  
t = table(factor(predicted, u), factor(testing\_gest, u))  
conf <- confusionMatrix(t)  
  
levelplot(sweep(conf$table, MARGIN = 2, STATS = colSums(conf$table), FUN = `/`), col.regions = gray(100:0/100))



## Result Comparison

# save models to file to ensure that the results were not lost  
saveRDS(object = models, file = "project\_models.RDS")  
results <- resamples(models)  
summary(results)

##   
## Call:  
## summary.resamples(object = results)  
##   
## Models: knn, lda, lda2, svmLinear, svmRadial   
## Number of resamples: 200   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## knn 0.7333333 0.9333333 0.9333333 0.9476667 1.0000000 1.0000000 0  
## lda 0.9333333 1.0000000 1.0000000 0.9950000 1.0000000 1.0000000 0  
## lda2 0.9333333 1.0000000 1.0000000 0.9953333 1.0000000 1.0000000 0  
## svmLinear 0.8666667 1.0000000 1.0000000 0.9940000 1.0000000 1.0000000 0  
## svmRadial 0.2666667 0.5333333 0.6666667 0.6530000 0.7333333 0.9333333 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max.  
## knn 0.66666667 0.9166667 0.9166667 0.9345833 1.0000000 1.0000000  
## lda 0.91666667 1.0000000 1.0000000 0.9937500 1.0000000 1.0000000  
## lda2 0.91666667 1.0000000 1.0000000 0.9941667 1.0000000 1.0000000  
## svmLinear 0.83333333 1.0000000 1.0000000 0.9925000 1.0000000 1.0000000  
## svmRadial 0.08333333 0.4166667 0.5833333 0.5662500 0.6666667 0.9166667  
## NA's  
## knn 0  
## lda 0  
## lda2 0  
## svmLinear 0  
## svmRadial 0

bwplot(results)

