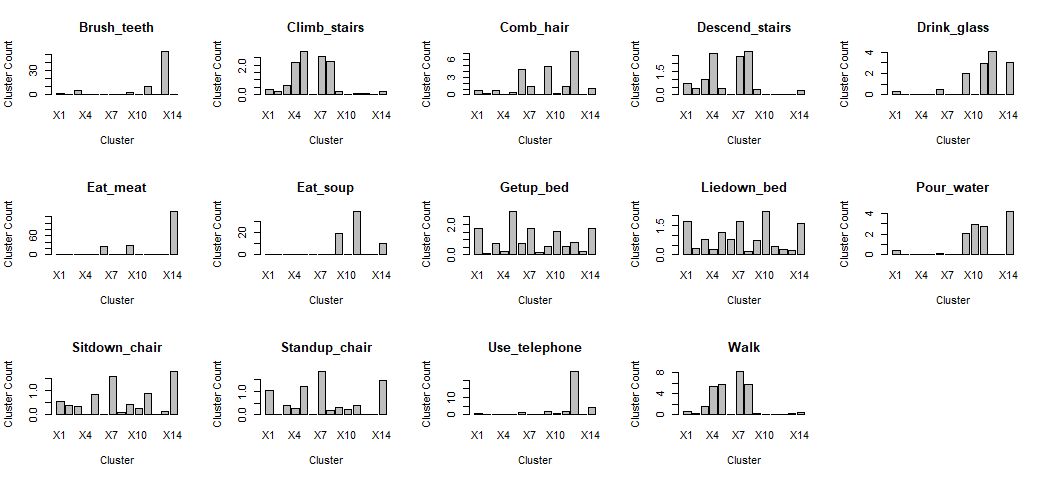
|  |  |  |  |
| --- | --- | --- | --- |
| Size of Fixed Length | Sample Overlap (%) | K | Accuracy |
| 32 | 0 | 14 | 0.7684211 |
| 32 | 0 | 28 | 0.7298246 |
| 32 | 0 | 140 | 0.7473684 |
| 64 | 0 | 14 | 0.7649123 |
| 64 | 0 | 28 | 0.7228070 |
| 64 | 0 | 140 | 0.7017544 |

I ran regular (not hierarchical) K-means



K-values is 14



Segmentation of Vector

d <- t(do.call(rbind,lapply(unlist(data), function(x) return(as.integer(unlist(strsplit(x, " ")))))))

lst <- lapply(c(1,which(1:ncol(d) %% seg\_size == 0) + 1), function(x) {

if (x + 31 <= ncol(d)) {

return(c(d[,x:(x+31)]))

} else {

return(NA)

}

})

K-means

segments <- do.call(rbind, lst)

clusters <- kmeans(segments, k)

Generating Histogram

output\_histogram <- function(cluster) {

setwd(og\_path)

data <- lapply(dir(og\_path, recursive = T), function(x) {

lines <- readLines(x)

d <- t(do.call(rbind,lapply(lines, function(y) return(as.integer(unlist(strsplit(y, " ")))))))

segments <- lapply(c(1,which(1:ncol(d) %% (ncol(cluster$centers)/3) == 0) + 1), function(y) {

if (y + 31 <= ncol(d)) {

return(c(d[,y:(y+31)]))

} else {

return(NA)

}

})

if (is.na(segments[[length(segments)]])) {

segments <- segments[-length(segments)]

}

predicts <- unlist(lapply(segments, function(y) {

distances <- apply(clusters$centers, 1, function(z) {

return(sqrt(sum((y - z)^2)))

})

return(which(distances == min(distances)))

}))

hist <- sapply(c(1:nrow(cluster$centers)), function(r) {

return(sum(predicts == r))

})

str <- strsplit(x, "/")[[1]][1]

index <- which(labels == str)

return(list(str, hist))

})

labs <- unlist(lapply(data, function(x) return(x[[1]])))

hist\_data <- do.call(rbind, lapply(data, function(x) return(x[[2]])))

return(data.frame(labs, hist\_data))

}

Classification

training <- as.h2o(x = hist\_data[unlist(splits),])

testing <- as.h2o(x = hist\_data[-unlist(splits),])

model <- h2o.randomForest(y = 1, training\_frame = training)

predictions <- predict(model, testing[,-1])

library(h2o)

#Gets all the data into a list of vectors, then combines it all into a vector.

#Each element of a vector is a string of three numbers separated by space". e.g.

#[1] "22 49 35" "22 49 35" "22 52 35"

og\_path <- getwd()

data <- lapply(dir(getwd()), function(x) {

# print(paste0(og\_path, "/", x))

setwd(paste0(og\_path, "/", x))

lines <- lapply(dir(getwd(), ".txt"), function(y) {

return(readLines(y))

})

vec <- unlist(lines)

return(vec)

})

labels <- unique(unlist(lapply(dir(og\_path, recursive = T), function(x) return(strsplit(x, "/")[[1]][1]))))

run\_kmeans <- function(k, seg\_size) {

d <- t(do.call(rbind,lapply(unlist(data), function(x) return(as.integer(unlist(strsplit(x, " ")))))))

lst <- lapply(c(1,which(1:ncol(d) %% seg\_size == 0) + 1), function(x) {

if (x + 31 <= ncol(d)) {

return(c(d[,x:(x+31)]))

} else {

return(NA)

}

})

if (is.na(lst[[length(lst)]])) {

lst <- lst[-length(lst)]

}

segments <- do.call(rbind, lst)

clusters <- kmeans(segments, k)

return(clusters)

}

output\_histogram <- function(cluster) {

setwd(og\_path)

data <- lapply(dir(og\_path, recursive = T), function(x) {

lines <- readLines(x)

d <- t(do.call(rbind,lapply(lines, function(y) return(as.integer(unlist(strsplit(y, " ")))))))

segments <- lapply(c(1,which(1:ncol(d) %% (ncol(cluster$centers)/3) == 0) + 1), function(y) {

if (y + 31 <= ncol(d)) {

return(c(d[,y:(y+31)]))

} else {

return(NA)

}

})

if (is.na(segments[[length(segments)]])) {

segments <- segments[-length(segments)]

}

predicts <- unlist(lapply(segments, function(y) {

distances <- apply(clusters$centers, 1, function(z) {

return(sqrt(sum((y - z)^2)))

})

return(which(distances == min(distances)))

}))

hist <- sapply(c(1:nrow(cluster$centers)), function(r) {

return(sum(predicts == r))

})

str <- strsplit(x, "/")[[1]][1]

index <- which(labels == str)

return(list(str, hist))

})

labs <- unlist(lapply(data, function(x) return(x[[1]])))

hist\_data <- do.call(rbind, lapply(data, function(x) return(x[[2]])))

return(data.frame(labs, hist\_data))

}

hist <- output\_histogram(clusters1)

# hist\_labels <- sapply(hist[,1], function(x) return(labels[[x]]))

# hist <- hist[,-1]

#Takes a matrix of histograms (i.e. obtained from runnuing output\_histogram()),

#which returns a matrix with 839 rows (each row corresponds to one of the files)

#and splits the rows into training and test sets. Performs 3-fold cross-validation

#and outputs a confusion table for the classes

validation <- function(hist\_data) {

splits <- lapply(labels, function(x) {

occur <- grep(x, dir(og\_path, recursive = T))

return(sample(occur, as.integer(2\*length(occur)/3)))

})

#h2o.init()

training <- as.h2o(x = hist\_data[unlist(splits),])

testing <- as.h2o(x = hist\_data[-unlist(splits),])

model <- h2o.randomForest(y = 1, training\_frame = training)

predictions <- predict(model, testing[,-1])

# print(as.vector(predictions[,1]) == "Brush\_teeth")

# print(testing[,1])

#Creates Confusion Matrix

rows <- lapply(c(1:14), function(x) {

x\_indices <- which(as.vector(predictions[,1]) == labels[x])

entries <- unlist(lapply(c(1:14), function(y) {

y\_indices <- which(as.vector(testing[,1]) == labels[y])

return(length(intersect(x\_indices,y\_indices)))

}))

return(entries)

})

confusion\_matrix <- do.call(rbind, rows)

row.names(confusion\_matrix) <- labels

colnames(confusion\_matrix) <- labels

return(list(acc = sum(diag(confusion\_matrix))/sum(c(confusion\_matrix)),

conf\_mat = confusion\_matrix))

}

clusters1 <- run\_kmeans(14, 32)

clusters2 <- run\_kmeans(28, 32)

clusters3 <- run\_kmeans(140, 32)

clusters4 <- run\_kmeans(14, 64)

clusters5 <- run\_kmeans(28, 64)

clusters6 <- run\_kmeans(140, 64)

hist1 <- output\_histogram(clusters1)

hist2 <- output\_histogram(clusters2)

hist3 <- output\_histogram(clusters3)

hist4 <- output\_histogram(clusters4)

hist5 <- output\_histogram(clusters5)

hist6 <- output\_histogram(clusters6)

val1 <- validation(hist1)

val2 <- validation(hist2)

val3 <- validation(hist3)

val4 <- validation(hist4)

val5 <- validation(hist5)

val6 <- validation(hist6)

# > sapply(c(1:6), function(x) return(eval(parse(text=paste0("val",x,"$acc")))))

# [1] 0.7684211 0.7298246 0.7473684 0.7649123 0.7228070 0.7017544

#So val1 has the best accuracy.

#Calculating mean quantized vectors, for each action

quantized\_action <- lapply(labels, function(x) {

occur <- grep(x, hist1[,1])

df <- hist1[occur,-1]

return(colMeans(df))

})

par(mfrow=c(3,5))

mean\_quantized\_barplots <- lapply(c(1:14), function(x) {

barplot(quantized\_action[[x]], main = labels[x],

xlab = "Cluster", ylab = "Cluster Count")

})