Image detection

Emily Spano

2024-04-04

library(imager)  
library(caret)   
library(MLmetrics)  
library(pixmap)  
library(philentropy)

# Function to load images from a folder, extract grayscale values, rescale them, stretch into vectors, and stack  
load\_and\_process\_images <- function(folder\_path, image\_size = c(224, 224)) {  
 # List all PNG files in the folder  
 image\_files <- list.files(folder\_path, pattern = "\\.png$", full.names = TRUE)  
   
 # Load images, extract grayscale values, and rescale them  
 images <- lapply(image\_files, function(file) {  
 img <- load.image(file)  
 gray\_values <- grayscale(img)  
 gray\_values\_rescaled <- gray\_values \* 255 # Rescale grayscale values  
 return(gray\_values\_rescaled)  
 })  
   
 # Stretch each image into a vector and stack the vectors  
 image\_vectors <- lapply(images, function(x) as.vector(x))  
 image\_matrix <- matrix(unlist(image\_vectors), ncol = image\_size[1] \* image\_size[2], byrow = TRUE)  
   
 return(image\_matrix)  
}  
  
# Specify paths to the target and non-target emotion folders  
target\_folder <- "C:/Users/emily/Documents/Face data final exam AUD/Target" # 200 images  
nontarget\_folder <- "C:/Users/emily/Documents/Face data final exam AUD/Non-target" # 200 images  
  
# Load, preprocess, and stack images from target and non-target folders  
target\_images <- load\_and\_process\_images(target\_folder)

# Combine the images from both folders  
all\_images <- rbind(target\_images, nontarget\_images)  
  
# Checking image 5  
image5 <- matrix(all\_images[1,], nrow = 224, byrow = TRUE)  
image(image5, col = gray.colors(256),main="Face5")

A close up of a person's face

Description automatically generated

# Step 7: Apply Eigenface method

# Perform PCA on combined dataset  
PCAProcess <- prcomp(all\_images, center = TRUE)  
min(which(summary(PCAProcess)$importance[3,] >= .90)) # with 84 principal components we can explain 90% of the variance

## [1] 84

#plot the eigenfaces  
Eigenfaces =PCAProcess$rotation[,1:84]  
  
# Plot the first ten eigenfaces  
par(mfrow = c(2,5))  
par(oma = rep(2,4), mar = c(0, 0, 3, 0))

for (i in 1:10) {  
 # Reshape the eigenface into a matrix  
 eigenface\_matrix <- matrix(data = rev(Eigenfaces[, i]), nrow = 224, ncol = 224)  
   
 # Plot the eigenface  
 image(1:224, 1:224, eigenface\_matrix, col = gray.colors(256))  
}

A close-up of a graph

Description automatically generatedA graph of a circle

Description automatically generated with medium confidenceA graph of a person's face

Description automatically generated with medium confidenceA graph of a graph

Description automatically generated with medium confidenceA blurry image of a person's face

Description automatically generatedA blurry image of a person's face

Description automatically generatedA blurry image of a person's face

Description automatically generatedA close-up of a graph

Description automatically generatedA blurry image of a person's face

Description automatically generatedA blurry image of a face

Description automatically generated

#extract loadings  
Loadings <- all\_images%\*%Eigenfaces  
  
# Check the dimensions of eigenface loadings  
dim(Loadings)

## [1] 400 84

# Loadings have been saved in a csv file to be used later for the modelling

# Step 8 Extract two types of algorithmic image features

## The features selected are Hough lines and texture

# Set paths to your image folders for target and non-target emotions  
target\_emotion\_folder <- "C:/Users/emily/Documents/Face data final exam AUD/Target"  
non\_target\_emotion\_folder <- "C:/Users/emily/Documents/Face data final exam AUD/Non-target"  
  
  
# Function to load images from a folder  
load\_images <- function(folder\_path) {  
 # List all image files in the folder  
 image\_files <- list.files(folder\_path, full.names = TRUE, pattern = "\\.(png)$", recursive = TRUE)  
 images <- lapply(image\_files, function(path) load.image(path))  
 return(images)  
}  
  
# Load images, target and non-target emotion   
target\_images <- load\_images(target\_emotion\_folder)

non\_target\_images <- load\_images(non\_target\_emotion\_folder)

# Combine the images  
all\_images <- c(target\_images, non\_target\_images)  
  
plot(all\_images[[1]])

A person's face with a number of times

Description automatically generated

# Step 8: Extract algorithmic features

## Canny edges + Hough lines

# Canny edge detection  
edges <- lapply(all\_images, cannyEdges)  
plot(edges[[1]])

A black square with white lines

Description automatically generated

# Detect lines via Hough detection  
lines <- lapply(edges, hough\_line, ntheta = 800)  
plot(lines[[1]])

A graph of smoke and a number

Description automatically generated with medium confidence

# create dataframes for every image  
lines\_df <- lapply(edges, hough\_line, ntheta = 800, data.frame = TRUE)  
# Initialize a vector to store the count of important lines for each image  
important\_lines\_counts <- numeric(length(lines\_df))  
# Loop through each dataframe in the list  
for (i in seq\_along(lines\_df)) {  
 # Extract the current dataframe  
 current\_df <- lines\_df[[i]]  
 # extract the scores  
 scores <- current\_df$score  
 # Determine the .995th percentile threshold for scores  
 score\_threshold <- quantile(scores, .995)  
 # Count lines with score above or equal to the threshold  
 important\_lines\_counts[i] <- sum(scores >= score\_threshold)  
}

## Texture

# creating a matrix for Gabor filter  
all\_images\_matrix <- lapply(all\_images, function(x)  
 as.matrix(x[,,1,1]))  
# setting parameters   
lamdas <- seq(2, 10, by = 2)  
thetas <- seq(0, 180, by = 45)  
bws <- seq(1, 4, by = 1)  
# Initialize an empty list to hold feature matrices for each image  
image\_features\_list <- list()  
# Loop through each image  
for (i in 1:length(all\_images\_matrix)) {  
 image <- all\_images\_matrix[[i]]  
 # Initialize an empty matrix for this image's features: rows = combinations, columns = mean & sd  
 num\_combinations <- length(lamdas) \* length(thetas) \* length(bws)  
 features\_matrix <- matrix(nrow = num\_combinations, ncol = 2)  
 # 2 for mean and sd  
 row.names(features\_matrix) <- paste("Combination",  
 1:num\_combinations)  
 colnames(features\_matrix) <- c("Mean", "SD")  
 # Counter for filling in the features\_matrix  
 count <- 1  
 # Loop through each combination of parameters  
 for (lamda in lamdas) {  
 for (theta in thetas) {  
 for (bw in bws) {  
 # Apply Gabor filter  
 result <- gabor.filter(image, lamda = lamda,  
 theta = theta , bw = bw)  
   
 # Calculate mean and standard deviation  
 features\_matrix[count, "Mean"] <-  
 mean(result$filtered\_img)  
 features\_matrix[count, "SD"] <-  
 sd(result$filtered\_img)  
 count <- count + 1  
 }  
 }  
 }  
 # Append the features matrix to the list  
 image\_features\_list[[i]] <- features\_matrix  
}  
  
  
# Number of images  
num\_images <- length(image\_features\_list)  
# Total number of features per image (mean + sd for each combination)  
num\_features <- nrow(image\_features\_list[[1]]) \*  
 ncol(image\_features\_list[[1]])  
# Pre-allocate matrix to store flattened features for all images  
flattened\_features <- matrix(nrow = num\_images, ncol = num\_features)  
# Flatten each image's features into a single row  
for (i in 1:num\_images) {  
 # Stack mean and SD values side by side for each combination  
 flattened\_features[i, ] <- c(t(image\_features\_list[[i]]))  
}  
# Generate column names for the flattened features  
feature\_names <- c(sapply(1:(num\_features / 2), function(x)  
 paste("Combination", x, c("Mean", "SD"), sep = "\_")))  
# Convert to a dataframe for easier combination with labels dataframe  
flattened\_features\_df <- as.data.frame(flattened\_features)  
colnames(flattened\_features\_df) <- feature\_names  
  
  
# Set parameters for the Gabor filter  
lambda <- 8  
theta <- 10 # Initial theta  
bw <- 1.5  
phi <- 0  
asp <- 0.3  
  
# Initialize the filtered image  
filt\_img <- matrix(0, nrow = dim(all\_images\_matrix[[1]])[1], ncol = dim(all\_images\_matrix[[1]])[2])  
  
# Apply Gabor filter to the first image  
for (theta\_val in seq(10, 180, 10)) {  
 out <- gabor.filter(x = all\_images\_matrix[[1]], lamda = lambda, theta = theta\_val, bw = bw, phi = phi, asp = asp)  
 # Combine the filtered images  
 filt\_img <- out$filtered\_img + filt\_img  
}  
  
# Set up the plotting layout  
par(mfrow = c(1, 2))  
  
# Plot the original image  
plot(all\_images[[1]], axes = FALSE, main = "Original Image")  
  
# Plot the filtered image  
image(rot90c(filt\_img), col = gray(c(0:255)/255), asp = 1, axes = FALSE, main = "Filtered Image", useRaster = TRUE)

A collage of images of a person

Description automatically generated

# Step 9: Explain your choices

For this step I decided to extract Hough lines and texture. Hough lines are important for image detection, especially when it comes to emotions because they are able to detect the shape of the image through lines. In this analysis, facial features such as eyebrows, mouth or eyes which eventually convey the emotion play a significant role, thus identifying wide open eyes which might mean that someone is afraid or open mouth that someone is surprise is very important; as my target emotion was disgust, detecting the shape of a wrinkled nose or closed eyes might have been an essential detail.

Similarly, texture is concerned with color or intensity variations. Because images were all in grayscale, changes in intensity were the focus of this analysis. For example, a tongue out, wrinkled forehead or nose might show areas where the surface is more dense. Particularly the Gabor filter returns a matrix of color representing different intensities which show where the image is more or less lighter. Understanding differences in area intensity can be crucial to discriminate image details which are essential to interpret the image correctly.

# Combining loadings and algorithmic features for modelling

# Reload the loadings from step 7 and add to teh dataframe  
eigenvector\_loadings <- read.csv("eigenface\_loadings.csv") # Adjust file path  
  
# Create a vector for disgust  
disgust <- rep(c(1, 0), c(200, 200))  
  
  
# Create a dataframe with the important features for modelling  
feature\_df <- data.frame(Important\_Lines\_Count = important\_lines\_counts,  
 Gabor\_features = flattened\_features\_df,  
 Eigenvector\_Loadings <- eigenvector\_loadings,  
 Disgust = disgust,  
 stringsAsFactors = FALSE)  
  
  
# The dataframe is saved as a CSV file for modelling

final\_df <- read.csv("feature\_df2.csv")  
  
# Shuffling the feature data frame  
set.seed(123)  
  
# Generate a vector of shuffled row indices  
shuffled\_df <- sample(nrow(final\_df))  
  
# Shuffle the rows of the data frame  
features <- final\_df[shuffled\_df, ]  
# Creating a dataset with Disgust emotion and loadings  
features\_loadings <- features[,170:286]  
# Creating a dataset with Disgust emotion and algorithmic features  
features\_algorithmic <- cbind(features$Disgust, features[1:169])

# Step 10: Classification models to predict disgust

## For this analysis three classification models have been chosen: SVM with radial kernel, logistic regression and random forest.

# Split all 3 dataset into training and validation samples  
train\_indices <- createDataPartition(y = features$Disgust, p = 0.8, list = FALSE)  
  
# Create training and validation datasets  
train\_data <- features[train\_indices,]  
test\_data <- features[-train\_indices,]  
  
train\_loadings <- features\_loadings[train\_indices, ]  
test\_loadings <- features\_loadings[-train\_indices, ]  
  
train\_algorithmic <- features\_algorithmic[train\_indices,]  
test\_algorithmic <- features\_algorithmic[-train\_indices,]

## First model: SVM

### SVM all features

# Train an SVM model   
svm\_allfeatures <- svm(Disgust ~ ., data = train\_data,   
 type = 'C-classification', kernel = 'radial')   
summary(svm\_allfeatures)

##   
## Call:  
## svm(formula = Disgust ~ ., data = train\_data, type = "C-classification",   
## kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 305  
##   
## ( 151 154 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

# Predict on train and test set   
prediction\_svm\_allfeatures\_train <- predict(svm\_allfeatures, train\_data)   
prediction\_svm\_allfeatures\_test <- predict(svm\_allfeatures, test\_data)   
  
  
# Evaluate predictions with kernel on train data  
accuracy\_svm\_allfeatures <- Accuracy(prediction\_svm\_allfeatures\_train,   
 train\_data$Disgust)   
  
recall\_svm\_allfeatures <- Recall(prediction\_svm\_allfeatures\_train,   
 train\_data$Disgust)   
  
f1\_svm\_allfeatures <- F1\_Score(prediction\_svm\_allfeatures\_train,   
 train\_data$Disgust)   
  
# Evaluate predictions with kernel on test data  
accuracy\_svm\_allfeatures\_test <- Accuracy(prediction\_svm\_allfeatures\_test,   
 test\_data$Disgust)   
  
recall\_svm\_allfeatures\_test <- Recall(prediction\_svm\_allfeatures\_test,   
 test\_data$Disgust)   
  
f1\_svm\_allfeatures\_test <- F1\_Score(prediction\_svm\_allfeatures\_test,   
 test\_data$Disgust)   
  
# resut train   
print(accuracy\_svm\_allfeatures)

## [1] 0.84375

print(recall\_svm\_allfeatures)

## [1] 0.8571429

print(f1\_svm\_allfeatures)

## [1] 0.8407643

# result test  
print(accuracy\_svm\_allfeatures\_test)

## [1] 0.575

print(recall\_svm\_allfeatures\_test)

## [1] 0.5789474

print(f1\_svm\_allfeatures\_test)

## [1] 0.5641026

### SVM loadings

# Train an SVM model   
svm\_loadings <- svm(Disgust ~ ., data = train\_loadings,   
 type = 'C-classification', kernel = 'radial')   
summary(svm\_loadings)

##   
## Call:  
## svm(formula = Disgust ~ ., data = train\_loadings, type = "C-classification",   
## kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 311  
##   
## ( 152 159 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

# Predict on test set   
pred\_loadings <- predict(svm\_loadings, train\_loadings)   
pred\_loadings\_test <- predict(svm\_loadings, test\_loadings)   
  
# Evaluate predictions with kernel on train  
accuracy\_svm\_loadings <- Accuracy(pred\_loadings,   
 train\_loadings$Disgust)   
  
recall\_svm\_loadings <- Recall(pred\_loadings,   
 train\_loadings$Disgust)   
  
f1\_svm\_loadings <- F1\_Score(pred\_loadings,   
 train\_loadings$Disgust)  
  
# Evaluate predictions with kernel on test  
accuracy\_svm\_loadings\_test <- Accuracy(pred\_loadings\_test,   
 test\_loadings$Disgust)   
  
recall\_svm\_loadings\_test <- Recall(pred\_loadings\_test,   
 test\_loadings$Disgust)   
  
f1\_svm\_loadings\_test <- F1\_Score(pred\_loadings\_test,   
 test\_loadings$Disgust)   
  
  
  
# result train  
print(accuracy\_svm\_loadings)

## [1] 0.94375

print(recall\_svm\_loadings)

## [1] 0.9671053

print(f1\_svm\_loadings)

## [1] 0.9423077

# result test  
print(accuracy\_svm\_loadings\_test)

## [1] 0.6125

print(recall\_svm\_loadings\_test)

## [1] 0.6153846

print(f1\_svm\_loadings\_test)

## [1] 0.6075949

### SVM algorithmic

# Train an SVM model   
svm\_algorithmic <- svm(`features$Disgust` ~ ., data = train\_algorithmic,   
 type = 'C-classification', kernel = 'radial')   
summary(svm\_algorithmic)

##   
## Call:  
## svm(formula = `features$Disgust` ~ ., data = train\_algorithmic, type = "C-classification",   
## kernel = "radial")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
##   
## Number of Support Vectors: 294  
##   
## ( 146 148 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## 0 1

# Predict on test set   
pred\_algorithmic <- predict(svm\_algorithmic, train\_algorithmic)   
pred\_algorithmic\_test <- predict(svm\_algorithmic, test\_algorithmic)   
  
  
# Evaluate predictions with kernel train  
accuracy\_svm\_algorithmic\_train <- Accuracy(pred\_algorithmic,   
 train\_algorithmic$`features$Disgust`)   
  
recall\_svm\_algorithmic\_train <- Recall(pred\_algorithmic,   
 train\_algorithmic$`features$Disgust`)   
  
f1\_svm\_algorithmic\_train <- F1\_Score(pred\_algorithmic,   
 train\_algorithmic$`features$Disgust`)   
  
  
# Evaluate predictions with kernel test  
accuracy\_svm\_algorithmic <- Accuracy(pred\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
recall\_svm\_algorithmic <- Recall(pred\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
f1\_svm\_algorithmic <- F1\_Score(pred\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
  
# result train  
print(accuracy\_svm\_algorithmic\_train)

## [1] 0.66875

print(recall\_svm\_algorithmic\_train)

## [1] 0.6626506

print(f1\_svm\_algorithmic\_train)

## [1] 0.6748466

# result test   
print(accuracy\_svm\_algorithmic)

## [1] 0.6

print(recall\_svm\_algorithmic)

## [1] 0.6111111

print(f1\_svm\_algorithmic)

## [1] 0.5789474

## Second model: logistic regression

### Logistic regression all features

# Train regression model  
reg\_allfeatures <- glm(train\_data$Disgust ~ ., data = train\_data[,-286], family = binomial)

# Print model summary  
summary(reg\_allfeatures)

# The output has been shortened for readability

##   
## Call:  
## glm(formula = train\_data$Disgust ~ ., family = binomial, data = train\_data[,   
## -286])  
##   
## Coefficients: (38 not defined because of singularities)  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) 6.835e+77 1.219e+08 5.605e+69 <2e-16  
## Important\_Lines\_Count -9.779e+73 3.955e+04 -2.473e+69 <2e-16  
## Gabor\_features.Combination\_1\_Mean 1.587e+71 6.256e+04 2.536e+66 <2e-16  
## Gabor\_features.Combination\_1\_SD -4.322e+72 6.684e+04 -6.466e+67 <2e-16  
## Gabor\_features.Combination\_2\_Mean -3.007e+73 5.374e+06 -5.596e+66 <2e-16  
## Gabor\_features.Combination\_2\_SD -4.552e+73 2.297e+06 -1.981e+67 <2e-16  
   
## (Intercept) \*\*\*  
## Important\_Lines\_Count \*\*\*  
## Gabor\_features.Combination\_1\_Mean \*\*\*  
## Gabor\_features.Combination\_1\_SD \*\*\*  
## Gabor\_features.Combination\_2\_Mean \*\*\*  
## Gabor\_features.Combination\_2\_SD \*\*\*  
## Gabor\_features.Combination\_3\_Mean \*\*\*  
## Gabor\_features.Combination\_3\_SD \*\*\*

# Predict on train set   
pred\_reg\_\_allfeatures\_probabilities\_train <- predict(reg\_allfeatures, newdata = train\_data[, -286], type = "response")  
  
# Predict on test set   
pred\_reg\_\_allfeatures\_probabilities\_test <- predict(reg\_allfeatures, newdata = test\_data[, -286], type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_reg\_allfeatures\_train <- ifelse(pred\_reg\_\_allfeatures\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_reg\_allfeatures\_test <- ifelse(pred\_reg\_\_allfeatures\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_reg\_allfeatures\_train <- Accuracy(pred\_reg\_allfeatures\_train,   
 train\_data$Disgust)   
# Accuracy on test set  
accuracy\_reg\_allfeatures\_test <- Accuracy(pred\_reg\_allfeatures\_test,   
 test\_data$Disgust)   
# Recall on train set  
recall\_reg\_allfeatures\_train <- Recall(pred\_reg\_allfeatures\_train,   
 train\_data$Disgust)   
  
# Recall on test set  
recall\_reg\_allfeatures\_test <- Recall(pred\_reg\_allfeatures\_test,   
 test\_data$Disgust)   
# F1 score on train set  
f1\_reg\_allfeatures\_train <- F1\_Score(pred\_reg\_allfeatures\_train,   
 train\_data$Disgust)   
# F1 score on test set  
f1\_reg\_allfeatures\_test <- F1\_Score(pred\_reg\_allfeatures\_test,   
 test\_data$Disgust)   
  
  
# Print results  
# Train set  
print(accuracy\_reg\_allfeatures\_train)

## [1] 0.509375

print(recall\_reg\_allfeatures\_train)

## [1] 0.5428571

print(f1\_reg\_allfeatures\_train)

## [1] 0.1948718

# Test set  
print(accuracy\_reg\_allfeatures\_test)

## [1] 0.525

print(recall\_reg\_allfeatures\_test)

## [1] 0.6

print(f1\_reg\_allfeatures\_test)

## [1] 0.24

### Logistic regression loadings

# Train regression model  
reg\_loadings <- glm(train\_loadings$Disgust ~ ., data = train\_loadings[,-286], family = binomial)

# Print model summary  
summary(reg\_loadings)

# The output has been shortened for readability

##   
## Call:  
## glm(formula = train\_loadings$Disgust ~ ., family = binomial,   
## data = train\_loadings[, -286])  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 3.595e+00 2.480e+00 1.450 0.147124   
## Gabor\_features.Combination\_85\_Mean -3.335e-04 2.136e-04 -1.561 0.118497   
## Gabor\_features.Combination\_85\_SD -7.519e-05 8.651e-05 -0.869 0.384770   
## Gabor\_features.Combination\_86\_Mean 1.417e-03 1.806e-03 0.785 0.432663   
## Gabor\_features.Combination\_86\_SD 6.132e-04 8.667e-04 0.707 0.479261   
## Gabor\_features.Combination\_87\_Mean -3.430e-03 1.608e-02 -0.213 0.831139   
## Gabor\_features.Combination\_87\_SD -6.792e-03 5.796e-03 -1.172 0.241264   
## Gabor\_features.Combination\_88\_Mean 2.991e-03 2.288e-02 0.131 0.895998   
## Gabor\_features.Combination\_88\_SD 1.362e-02 7.752e-03 1.756 0.079005

# Predict on train set   
pred\_reg\_\_loadings\_probabilities\_train <- predict(reg\_loadings, newdata = train\_loadings[, -286], type = "response")  
  
# Predict on test set   
pred\_reg\_\_loadings\_probabilities\_test <- predict(reg\_loadings, newdata = test\_loadings[, -286], type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_reg\_loadings\_train <- ifelse(pred\_reg\_\_loadings\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_reg\_loadings\_test <- ifelse(pred\_reg\_\_loadings\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_reg\_loadings\_train <- Accuracy(pred\_reg\_loadings\_train,   
 train\_loadings$Disgust)   
# Accuracy on test set  
accuracy\_reg\_loadings\_test <- Accuracy(pred\_reg\_loadings\_test,   
 test\_loadings$Disgust)   
# Recall on train set  
recall\_reg\_loadings\_train <- Recall(pred\_reg\_loadings\_train,   
 train\_loadings$Disgust)   
  
# Recall on test set  
recall\_reg\_loadings\_test <- Recall(pred\_reg\_loadings\_test,   
 test\_loadings$Disgust)   
# F1 score on train set  
f1\_reg\_loadings\_train <- F1\_Score(pred\_reg\_loadings\_train,   
 train\_loadings$Disgust)   
# F1 score on test set  
f1\_reg\_loadings\_test <- F1\_Score(pred\_reg\_loadings\_test,   
 test\_loadings$Disgust)   
  
  
# Print results  
# Train set  
print(accuracy\_reg\_loadings\_train)

## [1] 0.9125

print(recall\_reg\_loadings\_train)

## [1] 0.902439

print(f1\_reg\_loadings\_train)

## [1] 0.9135802

# Test set  
print(accuracy\_reg\_loadings\_test)

## [1] 0.5875

print(recall\_reg\_loadings\_test)

## [1] 0.5897436

print(f1\_reg\_loadings\_test)

## [1] 0.5822785

### Logistic regression algorithmic features

# Train regression model  
reg\_algorithmic <- glm(train\_algorithmic$`features$Disgust` ~ ., data = train\_algorithmic[,-1], family = binomial)

# Print model summary  
summary(reg\_algorithmic)

# The output has been shortened for readability

##   
## Call:  
## glm(formula = train\_algorithmic$`features$Disgust` ~ ., family = binomial,   
## data = train\_algorithmic[, -1])  
##   
## Coefficients: (32 not defined because of singularities)  
## Estimate Std. Error z value Pr(>|z|)  
## (Intercept) 3.991e+19 7.634e+07 5.228e+11 <2e-16  
## Important\_Lines\_Count -9.750e+15 2.471e+04 -3.945e+11 <2e-16  
## Gabor\_features.Combination\_1\_Mean -1.360e+16 3.895e+04 -3.493e+11 <2e-16  
## Gabor\_features.Combination\_1\_SD 2.053e+16 4.278e+04 4.799e+11 <2e-16  
## (Intercept) \*\*\*  
## Important\_Lines\_Count \*\*\*  
## Gabor\_features.Combination\_1\_Mean \*\*\*  
## Gabor\_features.Combination\_1\_SD \*\*\*  
## Gabor\_features.Combination\_2\_Mean \*\*\*  
## Gabor\_features.Combination\_2\_SD \*\*\*  
## Gabor\_features.Combination\_3\_Mean \*\*\*  
## Gabor\_features.Combination\_3\_SD \*\*\*  
## Gabor\_features.Combination\_4\_Mean \*\*\*  
## Gabor\_features.Combination\_4\_SD \*\*\*

# Predict on train set   
pred\_reg\_algorithmic\_probabilities\_train <- predict(reg\_algorithmic, newdata = train\_algorithmic[, -1], type = "response")  
  
# Predict on test set   
pred\_reg\_algorithmic\_probabilities\_test <- predict(reg\_algorithmic, newdata = test\_algorithmic[, -1], type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_reg\_algorithmic\_train <- ifelse(pred\_reg\_algorithmic\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_reg\_algorithmic\_test <- ifelse(pred\_reg\_algorithmic\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_reg\_algorithmic\_train <- Accuracy(pred\_reg\_algorithmic\_train,   
 train\_algorithmic$`features$Disgust`)   
# Accuracy on test set  
accuracy\_reg\_algorithmic\_test <- Accuracy(pred\_reg\_loadings\_test,   
 test\_algorithmic$`features$Disgust`)   
# Recall on train set  
recall\_reg\_algorithmic\_train <- Recall(pred\_reg\_loadings\_train,   
 train\_algorithmic$`features$Disgust`)   
  
# Recall on test set  
recall\_reg\_algorithmic\_test <- Recall(pred\_reg\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
# F1 score on train set  
f1\_reg\_algorithmic\_train <- F1\_Score(pred\_reg\_algorithmic\_train,   
 train\_algorithmic$`features$Disgust`)   
# F1 score on test set  
f1\_reg\_algorithmic\_test <- F1\_Score(pred\_reg\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
  
# Print results  
# Train set  
print(accuracy\_reg\_loadings\_train)

## [1] 0.9125

print(recall\_reg\_loadings\_train)

## [1] 0.902439

print(f1\_reg\_loadings\_train)

## [1] 0.9135802

# Test set  
print(accuracy\_reg\_loadings\_test)

## [1] 0.5875

print(recall\_reg\_loadings\_test)

## [1] 0.5897436

print(f1\_reg\_loadings\_test)

## [1] 0.5822785

## Third model: Random Forest

### Random Forest model on all features

library(randomForest)

rf\_model <- randomForest(Disgust ~ ., data = train\_data, ntree = 100, importance = TRUE)

# Print model summary  
print(rf\_model)

##   
## Call:  
## randomForest(formula = Disgust ~ ., data = train\_data, ntree = 100, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 95  
##   
## Mean of squared residuals: 0.2403985  
## % Var explained: 3.84

# Make predictions on the same dataset  
pred\_rf\_allfeatures\_probabilities\_train <- predict(rf\_model, train\_data, type = "response")  
  
# Make predictions on the same dataset  
pred\_rf\_allfeatures\_probabilities\_test <- predict(rf\_model, test\_data, type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_rf\_allfeatures\_train <- ifelse(pred\_rf\_allfeatures\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_rf\_allfeatures\_test <- ifelse(pred\_rf\_allfeatures\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_rf\_allfeatures\_train <- Accuracy(pred\_rf\_allfeatures\_train,   
 train\_data$Disgust)   
# Accuracy on test set  
accuracy\_rf\_allfeatures\_test <- Accuracy(pred\_rf\_allfeatures\_test,   
 test\_data$Disgust)   
  
# Recall on train set  
recall\_rf\_allfeatures\_train <- Recall(pred\_rf\_allfeatures\_train,   
 train\_data$Disgust)   
# Recall on test set  
recall\_rf\_allfeatures\_test <- Recall(pred\_rf\_allfeatures\_test,   
 test\_data$Disgust)   
  
# F1 score on train set  
f1\_rf\_allfeatures\_train <- F1\_Score(pred\_rf\_allfeatures\_train,   
 train\_data$Disgust)   
# F1 score on test set  
f1\_rf\_allfeatures\_test <- F1\_Score(pred\_rf\_allfeatures\_test,   
 test\_data$Disgust)   
  
  
# Print results  
# Train set  
print(accuracy\_rf\_allfeatures\_train)

## [1] 1

print(recall\_rf\_allfeatures\_train)

## [1] 1

print(f1\_rf\_allfeatures\_train)

## [1] 1

# Test set  
print(accuracy\_rf\_allfeatures\_test)

## [1] 0.625

print(recall\_rf\_allfeatures\_test)

## [1] 0.6315789

print(f1\_rf\_allfeatures\_test)

## [1] 0.6153846

### Random Forest model on loadings

rf\_model\_loadings <- randomForest(Disgust ~ ., data = train\_loadings, ntree = 100, importance = TRUE)

# Print model summary  
print(rf\_model\_loadings)

##   
## Call:  
## randomForest(formula = Disgust ~ ., data = train\_loadings, ntree = 100, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 38  
##   
## Mean of squared residuals: 0.2440759  
## % Var explained: 2.37

# Make predictions on the train dataset  
pred\_rf\_loadings\_probabilities\_train <- predict(rf\_model\_loadings, train\_loadings, type = "response")  
# Make predictions on the test dataset  
pred\_rf\_loadings\_probabilities\_test <- predict(rf\_model\_loadings, test\_loadings, type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_rf\_loadings\_train <- ifelse(pred\_rf\_loadings\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_rf\_loadings\_test <- ifelse(pred\_rf\_loadings\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_rf\_loadings\_train <- Accuracy(pred\_rf\_loadings\_train,   
 train\_loadings$Disgust)   
# Accuracy on test set  
accuracy\_rf\_loadings\_test <- Accuracy(pred\_rf\_loadings\_test,   
 test\_loadings$Disgust)   
  
# Recall on train set  
recall\_rf\_loadings\_train <- Recall(pred\_rf\_loadings\_train,   
 train\_loadings$Disgust)   
# Recall on test set  
recall\_rf\_loadings\_test <- Recall(pred\_rf\_loadings\_test,   
 test\_loadings$Disgust)   
  
# F1 score on train set  
f1\_rf\_loadings\_train <- F1\_Score(pred\_rf\_loadings\_train,   
 train\_loadings$Disgust)   
# F1 score on test set  
f1\_rf\_loadings\_test <- F1\_Score(pred\_rf\_loadings\_test,   
 test\_loadings$Disgust)   
  
  
# Print results  
# Train set  
print(accuracy\_rf\_loadings\_train)

## [1] 1

print(recall\_rf\_loadings\_train)

## [1] 1

print(f1\_rf\_loadings\_train)

## [1] 1

# Test set  
print(accuracy\_rf\_loadings\_test)

## [1] 0.55

print(recall\_rf\_loadings\_test)

## [1] 0.5526316

print(f1\_rf\_loadings\_test)

## [1] 0.5384615

### Fit a Random Forest model on algorithmic features

rf\_model\_algorithmic <- randomForest(train\_algorithmic$`features$Disgust` ~ ., data = train\_algorithmic, ntree = 100, importance = TRUE)

# Print model summary  
print(rf\_model\_algorithmic)

##   
## Call:  
## randomForest(formula = train\_algorithmic$`features$Disgust` ~ ., data = train\_algorithmic, ntree = 100, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 56  
##   
## Mean of squared residuals: 0.2548484  
## % Var explained: -1.94

# Make predictions on the same dataset  
pred\_rf\_algorithmic\_probabilities\_train <- predict(rf\_model\_algorithmic, train\_algorithmic, type = "response")  
  
# Make predictions on the same dataset  
pred\_rf\_algorithmic\_probabilities\_test <- predict(rf\_model\_algorithmic, test\_algorithmic, type = "response")  
  
# Convert predicted probabilities for train set to predicted class labels  
pred\_rf\_algorithmic\_train <- ifelse(pred\_rf\_algorithmic\_probabilities\_train > 0.5, 1, 0)  
  
# Convert predicted probabilities for test set to predicted class labels  
pred\_rf\_algorithmic\_test <- ifelse(pred\_rf\_algorithmic\_probabilities\_test > 0.5, 1, 0)  
  
  
# Accuracy on train set  
accuracy\_rf\_algorithmic\_train <- Accuracy(pred\_rf\_algorithmic\_train,   
 train\_algorithmic$`features$Disgust`)   
# Accuracy on test set  
accuracy\_rf\_algorithmic\_test <- Accuracy(pred\_rf\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
# Recall on train set  
recall\_rf\_algorithmic\_train <- Recall(pred\_rf\_algorithmic\_train,   
 train\_algorithmic$`features$Disgust`)   
# Recall on test set  
recall\_rf\_algorithmic\_test <- Recall(pred\_rf\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
# F1 score on train set  
f1\_rf\_algorithmic\_train <- F1\_Score(pred\_rf\_algorithmic\_train,   
 train\_algorithmic$`features$Disgust`)   
# F1 score on test set  
f1\_rf\_algorithmic\_test <- F1\_Score(pred\_rf\_algorithmic\_test,   
 test\_algorithmic$`features$Disgust`)   
  
  
# Print results  
# Train set  
print(accuracy\_rf\_algorithmic\_train)

## [1] 1

print(recall\_rf\_algorithmic\_train)

## [1] 1

print(f1\_rf\_algorithmic\_train)

## [1] 1

# Test set  
print(accuracy\_rf\_algorithmic\_test)

## [1] 0.575

print(recall\_rf\_algorithmic\_test)

## [1] 0.575

print(f1\_rf\_algorithmic\_test)

## [1] 0.575

# Result Tables

# Results SVM  
allfeatures\_svm <- c(accuracy\_svm\_allfeatures, recall\_svm\_allfeatures, f1\_svm\_allfeatures, accuracy\_svm\_allfeatures\_test, recall\_svm\_allfeatures\_test, f1\_svm\_allfeatures\_test)  
  
loadings\_svm <- c(accuracy\_svm\_loadings, recall\_svm\_loadings, f1\_svm\_loadings,  
 accuracy\_svm\_loadings\_test, recall\_svm\_loadings\_test, f1\_svm\_loadings\_test)  
  
algorithmic\_svm <- c(accuracy\_svm\_algorithmic\_train, recall\_svm\_algorithmic\_train, f1\_svm\_algorithmic\_train, accuracy\_svm\_algorithmic, recall\_svm\_algorithmic, f1\_svm\_algorithmic)  
  
# Results regressions  
allfeatures\_reg <- c(accuracy\_reg\_allfeatures\_train, recall\_reg\_allfeatures\_train, f1\_reg\_allfeatures\_train, accuracy\_reg\_allfeatures\_test, recall\_reg\_allfeatures\_test, f1\_reg\_allfeatures\_test)  
  
loadings\_reg <- c(accuracy\_reg\_loadings\_train, recall\_reg\_loadings\_train, f1\_reg\_loadings\_train,  
 accuracy\_reg\_loadings\_test, recall\_reg\_loadings\_test, f1\_reg\_loadings\_test)  
  
algorithmic\_reg <- c(accuracy\_reg\_algorithmic\_train, recall\_reg\_algorithmic\_train, f1\_reg\_algorithmic\_train, accuracy\_reg\_algorithmic\_test, recall\_reg\_algorithmic\_test, f1\_reg\_algorithmic\_test)  
  
# Results Random Forests  
allfeatures\_rf <- c(accuracy\_rf\_allfeatures\_train, recall\_rf\_allfeatures\_train, f1\_rf\_allfeatures\_train, accuracy\_rf\_allfeatures\_test, recall\_rf\_allfeatures\_test, f1\_rf\_allfeatures\_test)  
  
loadings\_rf <- c(accuracy\_rf\_loadings\_train, recall\_rf\_loadings\_train, f1\_rf\_loadings\_train,  
 accuracy\_rf\_loadings\_test, recall\_rf\_loadings\_test, f1\_rf\_loadings\_test)  
  
algorithmic\_rf <- c(accuracy\_rf\_algorithmic\_train, recall\_rf\_algorithmic\_train, f1\_rf\_algorithmic\_train, accuracy\_rf\_algorithmic\_test, recall\_rf\_algorithmic\_test, f1\_rf\_algorithmic\_test)  
  
  
# Storing results to data frames for each classifier  
svm\_results <- data.frame(allfeatures\_svm, loadings\_svm, algorithmic\_svm)  
regression\_results <- data.frame(allfeatures\_reg, loadings\_reg, algorithmic\_reg)  
rforest\_results <- data.frame(allfeatures\_rf, loadings\_rf, algorithmic\_rf)  
  
# Rounding the results  
svm\_results <- round(svm\_results, digits = 2)  
regression\_results <- round(regression\_results, digits = 2)  
rforest\_results <- round(rforest\_results, digits = 2)  
  
# Labeling rows  
rowlabels <- c("Accuracy\_train", "Recall\_train", "F1-score\_train", "Accuracy\_test", "Recall\_test", "F1-score\_test")  
  
row.names(svm\_results) <- rowlabels  
row.names(regression\_results) <- rowlabels  
row.names(rforest\_results) <- rowlabels  
  
# Combining all results  
results <- cbind(svm\_results, regression\_results, rforest\_results)  
print(results)

## allfeatures\_svm loadings\_svm algorithmic\_svm allfeatures\_reg  
## Accuracy\_train 0.84 0.94 0.67 0.51  
## Recall\_train 0.86 0.97 0.66 0.54  
## F1-score\_train 0.84 0.94 0.67 0.19  
## Accuracy\_test 0.58 0.61 0.60 0.52  
## Recall\_test 0.58 0.62 0.61 0.60  
## F1-score\_test 0.56 0.61 0.58 0.24  
## loadings\_reg algorithmic\_reg allfeatures\_rf loadings\_rf  
## Accuracy\_train 0.91 0.47 1.00 1.00  
## Recall\_train 0.90 0.90 1.00 1.00  
## F1-score\_train 0.91 0.62 1.00 1.00  
## Accuracy\_test 0.59 0.59 0.62 0.55  
## Recall\_test 0.59 0.50 0.63 0.55  
## F1-score\_test 0.58 0.64 0.62 0.54  
## algorithmic\_rf  
## Accuracy\_train 1.00  
## Recall\_train 1.00  
## F1-score\_train 1.00  
## Accuracy\_test 0.58  
## Recall\_test 0.58  
## F1-score\_test 0.58

## This code has been partly generated with chatGPT

# Step 11: Discussing results

What stands out from the summary table above is that when the models are tested on the train set, they tend to overfit, whereas when they are tested on the test set the accuracy is pretty low (0.56- 0.63). All the models however show very similar results for all combinations, particularly, SVM provides better results when all features and only the loadings are considered, the logistic regression model perform nicely only on the loadings dataset whereas random forests performs slightly better when all features are analyzed.

Some possible reasons behind these result can be pinpointed in the data set. As a matter of fact, the quality of the images is overall pretty low, some of them present numbers or text which might create some noise. Moreover, the variation of these picture is very high (in some images, characters show hands, or they are turned on the side) and due to lack of computational power I could only select 400 images which might not be enough for an accurate image detection analysis.

Furthermore, the amount of features to analyze is also very high which might also create some noise. However, when taking a subset of the main data set, results are similar which might suggest that either the amount of feature (285) is not a problem or the feature extracted are not relevant enough to describe the image correctly. To confirm the latter is the number of support vector machines used for the SVM which are around 300, meaning that it is very difficult for the model to find some discriminant features.

In sum, it can be suggested that by increasing the quality of pictures, by using a larger data set and by using different models or even a deep learning approach, the accuracy might increase by a lot.