Appendix

R Script

```r
############################################
# Brain Tumor Identification in MRI Images #
#   using Kmeans Clustering and XGBoost    #
############################################

# 1. Load libraries
# -----------------

library(imager)
library(xgboost)
library(caret)
library(pROC)

# 2. Setup files
# --------------

# 2.a. Set working directory
setwd("E:/MS ML Project/Data/")

# 2.b. Extract all files
orig_tumor_files = paste("./tumor/",
                         list.files(path = "./tumor"), sep = "")

orig_non_tumor_files = paste("./non_tumor/",
                             list.files(path = "./non_tumor"), sep = "")

# 3. Image augmentation to expand our data
# ----------------------------------------

# 3.a. Function to create 4 rotated versions of the image -- #
augment_image = function(file, target)
{
  # Load image
  img_original = load.image(file)

  # Flip image horizontally
  img_flip_horizontal = mirror(img_original, "x")

  # Flip image vertically
  img_flip_vertical = mirror(img_original, "y")

  # 180 deegrees rotation
  img_x_y_mirror = imrotate(img_original, 180)

  # Create file names
  original_file_name = strsplit(strsplit(file, "\\/")[[1]][3], "\\.")[[1]][1]

  img_original_file_name = paste(target,
                               original_file_name, "_OR", '.jpg', sep = "")
  img_flip_horizontal_file_name = paste(target,
                                      original_file_name, "_FH", '.jpg', sep = "")
  img_flip_vertical_file_name = paste(target,
                                    original_file_name, "_FV", '.jpg', sep = "")
  img_x_y_mirror_file_name = paste(target,
                                 original_file_name, "_XY", '.jpg', sep = "")

  # Save images
  save.image(img_original, img_original_file_name)
  save.image(img_flip_horizontal, img_flip_horizontal_file_name)
  save.image(img_flip_vertical, img_flip_vertical_file_name)
  save.image(img_x_y_mirror, img_x_y_mirror_file_name)
}

# 3.b. Augment tumor files
for(file_name in orig_tumor_files)
{
  augment_image(file_name, "./augmented/tumor/")
}

# 3.c. Augment non-tumor files
for(file_name in orig_non_tumor_files)
{
  augment_image(file_name, "./augmented/non_tumor/")
}

# 4. Create training/validation/test set
# --------------------------------------

# 4.a. Read augmented files
tumor_files = paste("./augmented/tumor/",
                    list.files(path = "./augmented/tumor"), sep = "")

non_tumor_files = paste("./augmented/non_tumor/",
                        list.files(path = "./augmented/non_tumor"), sep = "")

# 4.b. Create training set
set.seed(2)
train_Y_indices = sample(1:length(tumor_files),
                         floor(0.7*(length(tumor_files))))

train_Y_files = tumor_files[train_Y_indices]

train_N_indices = sample(1:length(non_tumor_files),
                         floor(0.7*(length(non_tumor_files))))
```

```r
train_N_files = non_tumor_files[train_N_indices]

val_test_tumor_files = tumor_files[-train_Y_indices]

val_test_non_tumor_files = non_tumor_files[-train_N_indices]

# 4.c. Create validation set
val_Y_indices = sample(1:length(val_test_tumor_files),
                       floor(0.5*(length(val_test_tumor_files))))

val_Y_files = val_test_tumor_files[val_Y_indices]

val_N_indices = sample(1:length(val_test_non_tumor_files),
                       floor(0.5*(length(val_test_non_tumor_files))))

val_N_files = val_test_non_tumor_files[val_N_indices]

# 4.d. Create test set
test_Y_files = val_test_tumor_files[-val_Y_indices]
test_N_files = val_test_non_tumor_files[-val_N_indices]

# 4.e. All files
train_files = c(train_Y_files, train_N_files)
val_files = c(val_Y_files, val_N_files)
test_files = c(test_Y_files, test_N_files)
all_files = c(train_files, val_files, test_files)

# 5. Identifying optimal number of clusters using elbow method
# ------------------------------------------------------------

# 5.a. Function to convert image to a vector
image_to_vec = function(file)
{
  # Load image
  im = load.image(file)

  # Convert to a greyscale intensity matrix
  mat = as.matrix(im[,,1,1])

  # Transpose and reverse to keep the aspect ratio intact
  mat = t(apply(mat, 1, rev))

  # Calculate aspect ratio
  asp_ratio = dim(mat)[2]/dim(mat)[1]

  # Convert to vector
  Vector = as.vector(mat)

  return(list(Vector, dim(mat)[2], dim(mat)[1], asp_ratio))
}

# 5.b. Function to calculate within sum of square distance for k means clustering
perform_k_means = function(Vector, k)
```

```r
{
  # Set seed
  set.seed(1)

  # Perform k means
  kmc = kmeans(Vector,
               nstart = 50,
               centers = k,
               iter.max = 50)

  # Calculate total within sum of squared distance
  tot_within_ss = kmc$tot.withinss

  # Return the value
  return(tot_within_ss)
}

# 5.c. Create an elbow curve
max_k = 10
wss = matrix(0, nrow = length(train_files), ncol = max_k)
for(i in 1:length(train_files))
{
  vec = image_to_vec(train_files[i])[[1]]
  for(j in 1:max_k)
  {
    wss[i,j] = perform_k_means(vec, j)
  }

}

# 5.d. Average across all training samples
wss_avg = apply(wss, MARGIN = 2, mean)

# 5.e. Plot the curve
plot(1:max_k, wss_avg,
     type ="b",
     pch = 19,
     xlab ="Number of clusters K",
     ylab ="Total within-clusters sum of squares",
     main = "Elbow curve for identifying optimal no. of clusters")
abline(v = 3, col = 'green', lty = 2)

# 5.f. Observe the graph and identify the optimal no. of clusters
opt_k = 3

# 6. Visualizing the results of k-means clustering
# ------------------------------------------------

Y_image_index = 40
N_image_index = 159

# 6.a. Non-tumor
vec_all_N = image_to_vec(train_N_files[N_image_index])
```

```r
vec_N = vec_all_N[[1]]
height_N = vec_all_N[[2]]
width_N = vec_all_N[[3]]
asp_ratio_N = vec_all_N[[4]]

set.seed(1)
kmc_N = kmeans(vec_N,
               nstart = 50,
               centers = opt_k,
               iter.max = 50)

clusters_N = kmc_N$cluster
dim(clusters_N) = c(width_N, height_N)

# 6.b. Tumor
vec_all_Y = image_to_vec(train_Y_files[Y_image_index])
vec_Y = vec_all_Y[[1]]
height_Y = vec_all_Y[[2]]
width_Y = vec_all_Y[[3]]
asp_ratio_Y = vec_all_Y[[4]]

set.seed(1)
kmc_Y = kmeans(vec_Y,
               nstart = 50,
               centers = opt_k,
               iter.max = 50)

clusters_Y = kmc_Y$cluster
dim(clusters_Y) = c(width_Y, height_Y)

# 6.c. Plot

# Input images
par(mfrow = c(1,2), oma = c(0, 0, 2, 0)) #outer margins
plot(load.image(train_N_files[N_image_index]), main = 'Non-tumor')
plot(load.image(train_Y_files[Y_image_index]), main = 'Tumor')
mtext('Input images', outer = T, cex = 1.5)

# Clustered images
image(clusters_N, col = c("green", "red", "blue"),
      asp = asp_ratio_N, main = 'Non-tumor', frame = F)

image(clusters_Y, col = c("green", "blue", "red"),
      asp = asp_ratio_Y, main = 'Non-tumor', frame = F)

mtext('Clustered images', outer = T, cex = 1.5)
par(mfrow = c(1,1))

# 7. Performing k-means using the optimal no. of clusters
# ----------------------------------------------------------

# 7.a. Initialize a dataframe to store cluster results
cluster_data = data.frame(matrix(0, nrow = length(all_files), ncol = 4*opt_k + 2))
```

```r
colnames(cluster_data) = c('data_set',
                           'c1_size', 'c2_size', 'c3_size',
                           'c1_intensity', 'c2_intensity', 'c3_intensity',
                           'c1_x_centroid', 'c2_x_centroid', 'c3_x_centroid',
                           'c1_y_centroid', 'c2_y_centroid', 'c3_y_centroid',
                           'tumor')

# 7.b. Function to perform k-means and store clustering output to a dataframe
kmeans_insert_data = function(cluster_data, files, data_set_name, tumor, curr_row)
{
  for(i in 1:length(files))
  {
    # Convert image to vector
    vec_all = image_to_vec(files[i])
    vec = vec_all[[1]]
    height = vec_all[[2]]
    width = vec_all[[3]]
    asp_ratio = vec_all[[4]]

    # Set seed
    set.seed(1)

    # Perform k means
    kmc = kmeans(vec,
                 nstart = 50,
                 centers = opt_k,
                 iter.max = 50)

    # Extract cluster information
    centers = kmc$centers
    ordered_centers = order(centers)
    centers = centers[ordered_centers] #Intensity
    size = kmc$size[ordered_centers] #size

    # Extract centroids
    clusters = kmc$cluster
    dim(clusters) = c(width, height)
    x_centroids = c()
    y_centroids = c()
    for(j in 1:opt_k)
    {
      coords = which(clusters == j, arr.ind = T)
      x_centroid = apply(coords, MARGIN = 2, mean)[1]
      y_centroid = apply(coords, MARGIN = 2, mean)[2]
      x_centroids = c(x_centroids, x_centroid)
      y_centroids = c(y_centroids, y_centroid)
    }

    x_centroids = x_centroids[ordered_centers] #centroids
    y_centroids = y_centroids[ordered_centers] #centroids

    # Insert data
    cluster_data[curr_row,1] = data_set_name
    cluster_data[curr_row,2:(opt_k+1)] = size
    cluster_data[curr_row,(opt_k+2):(2*opt_k+1)] = centers
    cluster_data[curr_row,(2*opt_k+2):(3*opt_k+1)] = x_centroids
    cluster_data[curr_row,(3*opt_k+2):(4*opt_k+1)] = y_centroids
    cluster_data[curr_row,4*opt_k+2] = tumor

    # Increment row
    curr_row = curr_row + 1
  }
  return(list(cluster_data, curr_row))
}

# 7.c. Use the above function to create clustering output dataframe

# 7.c.1 Insert training data

# 7.c.1.a Non-tumor
curr_row = 1
cluster_output = kmeans_insert_data(cluster_data, train_N_files, 'train', 0, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.c.1.b Tumor
cluster_output = kmeans_insert_data(cluster_data, train_Y_files, 'train', 1, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.c.2 Insert validation data

# 7.c.2.a Non-tumor
cluster_output = kmeans_insert_data(cluster_data, val_N_files, 'val', 0, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.c.2.b Tumor
cluster_output = kmeans_insert_data(cluster_data, val_Y_files, 'val', 1, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.c.3 Insert test data

# 7.c.3.a Non-tumor
cluster_output = kmeans_insert_data(cluster_data, test_N_files, 'test', 0, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.c.3.b Tumor
cluster_output = kmeans_insert_data(cluster_data, test_Y_files, 'test', 1, curr_row)
cluster_data = cluster_output[[1]]
curr_row = cluster_output[[2]]

# 7.d View data
knitr::kable(cluster_data[1:5, 1:5])
```

```r
knitr::kable(cluster_data[1:5, 6:10])
knitr::kable(cluster_data[1:5, 11:14])

# 8. Cross Validation for XGBoost
# -------------------------------

# 8.1. Create data matrix for XGBoost
X_train = data.matrix(cluster_data[cluster_data$data_set == 'train', c(-1, -14)])
X_val = data.matrix(cluster_data[cluster_data$data_set == 'val', c(-1, -14)])
X_test = data.matrix(cluster_data[cluster_data$data_set == 'test', c(-1, -14)])
Y_train = data.matrix(cluster_data[cluster_data$data_set == 'train', 14])
Y_val = data.matrix(cluster_data[cluster_data$data_set == 'val', 14])
Y_test = data.matrix(cluster_data[cluster_data$data_set == 'test', 14])

# 8.2. Create xgb matrix
train_data = xgb.DMatrix(X_train, label = Y_train)
val_data = xgb.DMatrix(X_val, label = Y_val)

# 8.3 Define model parameters
parameters = list(eta = 0.01,
                  max_depth = 6,
                  objective = "binary:logistic",
                  seed = 1)

# 8.4 Train model
set.seed(1)
model_xgb_cv = xgb.train(data = train_data,
                         params = parameters,
                         watchlist = list(eval = val_data, train = train_data),
                         nrounds = 2500,
                         verbose = 0)

# 8.5 Plot validation error vs. no. of iterations
plot(model_xgb_cv$evaluation_log$iter,
     model_xgb_cv$evaluation_log$eval_error,
     type = "l",
     main = "Validation error vs No. of iterations",
     xlab = "No. of iterations",
     ylab = "Validation error")

# 8.6 Find optimum no. of iterations
opt_iterations = model_xgb_cv$evaluation_log$iter[
  which(model_xgb_cv$evaluation_log$eval_error ==
          min(model_xgb_cv$evaluation_log$eval_error))[1]]

print(paste('Optimal no. of iterations:', opt_iterations))

# 9. Train final model using hyperparameters obtained from cross validation
# -------------------------------------------------------------------------

# 9.1 Append training and validation set
X_train_val = data.matrix(
  cluster_data[cluster_data$data_set %in% c('train','val'), c(-1, -14)])
```

```r
Y_train_val = data.matrix(
  cluster_data[cluster_data$data_set %in% c('train','val'), 14])

# 9.2. Create xgb matrix
train_val_data = xgb.DMatrix(X_train_val, label = Y_train_val)

# 9.3 Train model
set.seed(1)
model_xgb = xgb.train(data = train_val_data,
                      params = parameters,
                      watchlist = list(train = train_val_data),
                      nrounds = opt_iterations,
                      verbose = 0)

# 9.4 Plot validation error vs. no. of iterations
plot(model_xgb$evaluation_log$iter,
     model_xgb$evaluation_log$train_error,
     type = "l",
     main = "Training + validation error vs No. of iterations",
     xlab = "No. of iterations",
     ylab = "Training + validation error")

# 10. Model performance
# ---------------------

# 10.1. Training + validation results

# 10.1.a Predictions
pred = predict(model_xgb, X_train_val)
pred_labels = as.factor(as.numeric(pred >= 0.5))
Y_train_val = as.factor(Y_train_val)

# 10.1.b Confusion matrix
confusion_matrix = confusionMatrix(pred_labels, Y_train_val)
print("Confusion matrix for training + validation data:")
print(confusion_matrix$table)

# 10.1.c Accuracy, sensitivity, and specificity
train_val_accuracy = confusion_matrix$overall['Accuracy']
train_val_sensitivity = confusion_matrix$byClass['Sensitivity']
train_val_specificity = confusion_matrix$byClass['Specificity']
print(paste('Training + validation accuracy:', train_val_accuracy))
print(paste('Training + validation sensitivity:', train_val_sensitivity))
print(paste('Training + validation specificity:', train_val_specificity))

# 10.1.d ROC
train_val_roc = roc(response = Y_train_val, predictor = pred)
plot(train_val_roc, main = "ROC Curve for training + validation set")

# 10.1.e AUC
train_val_AUC = train_val_roc$auc
print(paste('Training + validation set AUC:', train_val_AUC))
```

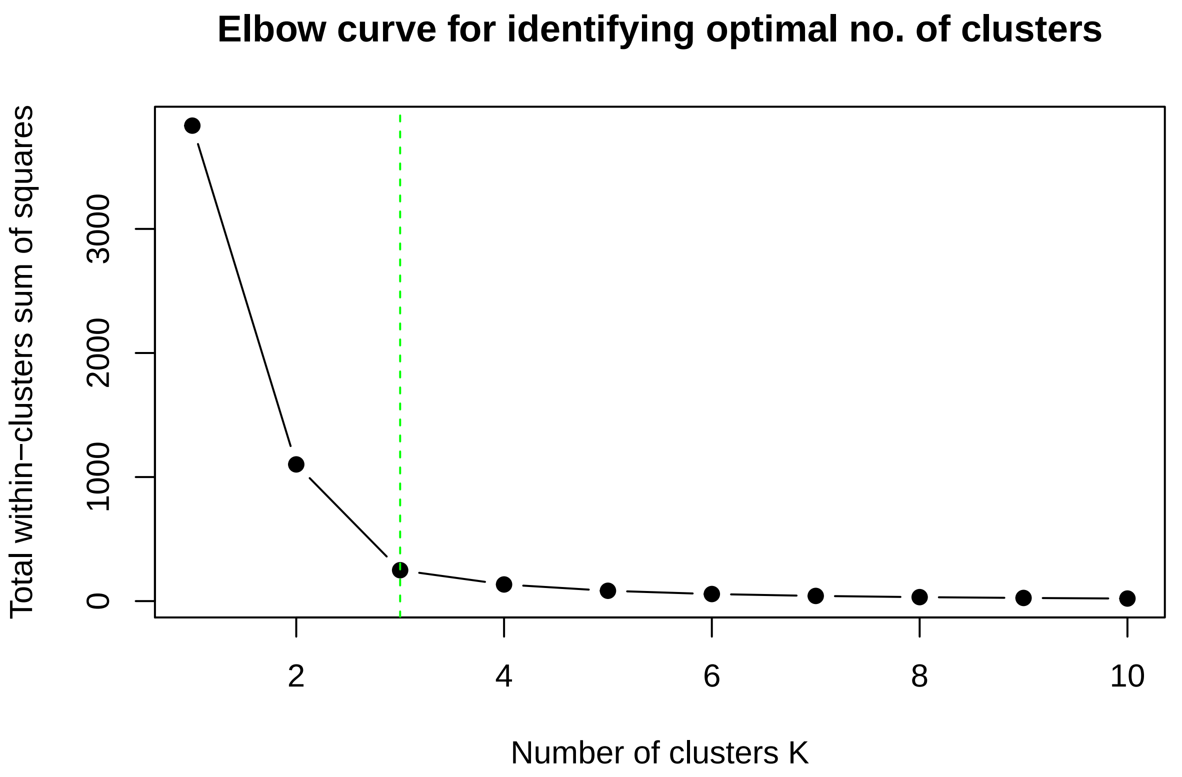
```r
# 10.2. Test results

# 10.2.a Predictions
pred = predict(model_xgb, X_test)
pred_labels = as.factor(as.numeric(pred >= 0.5))
Y_test = as.factor(Y_test)

# 10.2.b Confusion matrix
confusion_matrix = confusionMatrix(pred_labels, Y_test)
print("Confusion matrix for test data:")
print(confusion_matrix$table)

# 10.2.c Accuracy
test_accuracy = confusion_matrix$overall['Accuracy']
test_sensitivity = confusion_matrix$byClass['Sensitivity']
test_specificity = confusion_matrix$byClass['Specificity']
print(paste('Test accuracy:', test_accuracy))
print(paste('Test sensitivity:', test_sensitivity))
print(paste('Test specificity:', test_specificity))

# 10.2.d ROC
test_roc = roc(response = Y_test, predictor = pred)
plot(test_roc, main = "ROC Curve for test set")

# 10.2.e AUC
test_AUC = test_roc$auc
print(paste('Test AUC:', test_AUC))
```

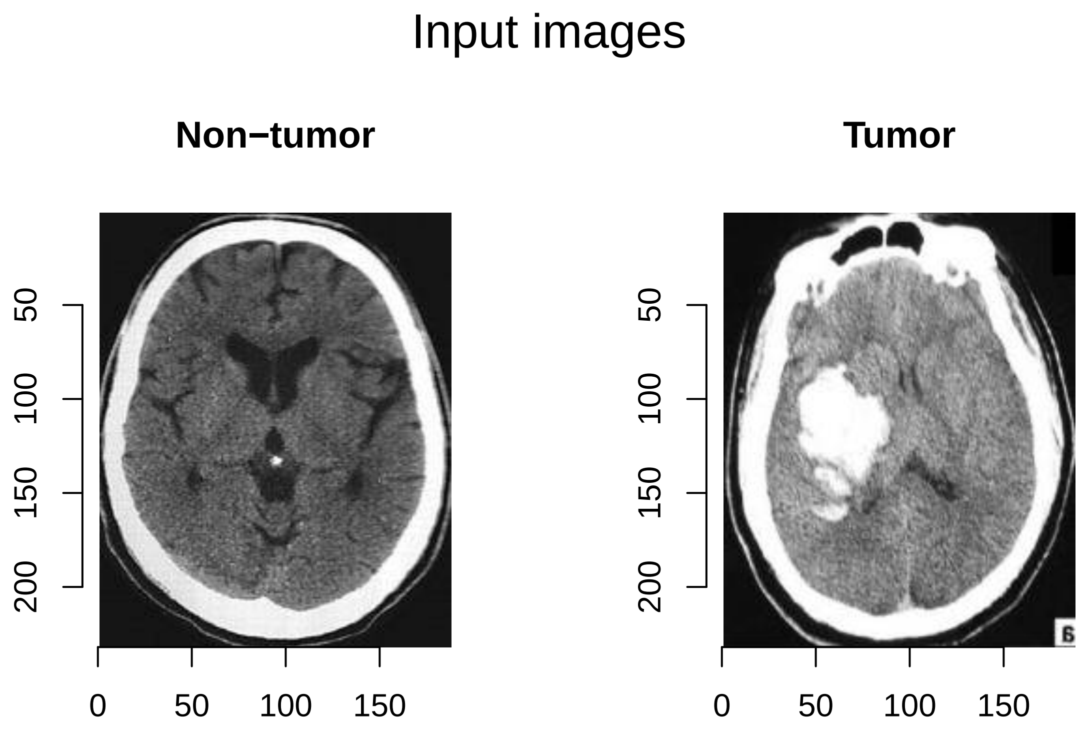
data_set	c1_size	c2_size	c3_size	c1_intensity
train	17607	20955	5054	0.0348607
train	16905	19751	6960	0.0655778
train	13714	23038	6864	0.0463897
train	10641	27462	5513	0.0217019
train	14514	18560	10542	0.0409230

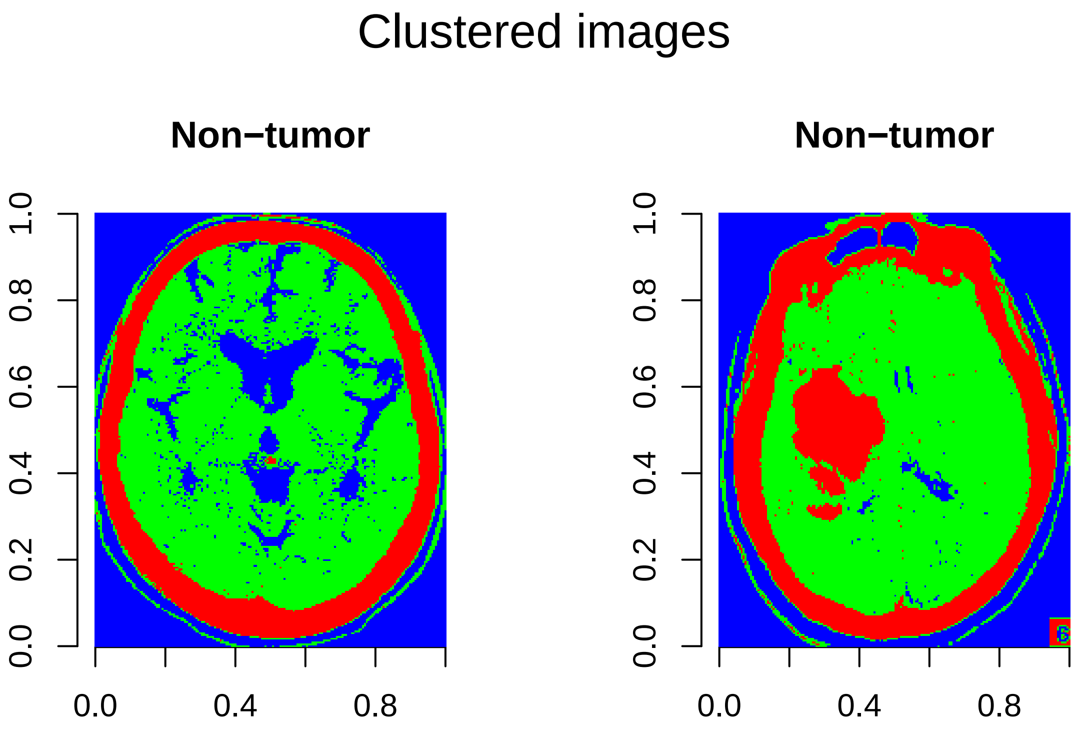
c2_intensity	c3_intensity	c1_x_centroid	c2_x_centroid	c3_x_centroid
0.3471991	0.9332255	95.55342	93.76683	93.87000
0.3442183	0.9425208	98.41083	92.93332	89.44698
0.3662888	0.9529098	94.07255	94.85298	94.16929
0.4386409	0.9407041	90.59261	95.80537	95.53945
0.3548802	0.9439750	92.02039	95.83109	95.57039

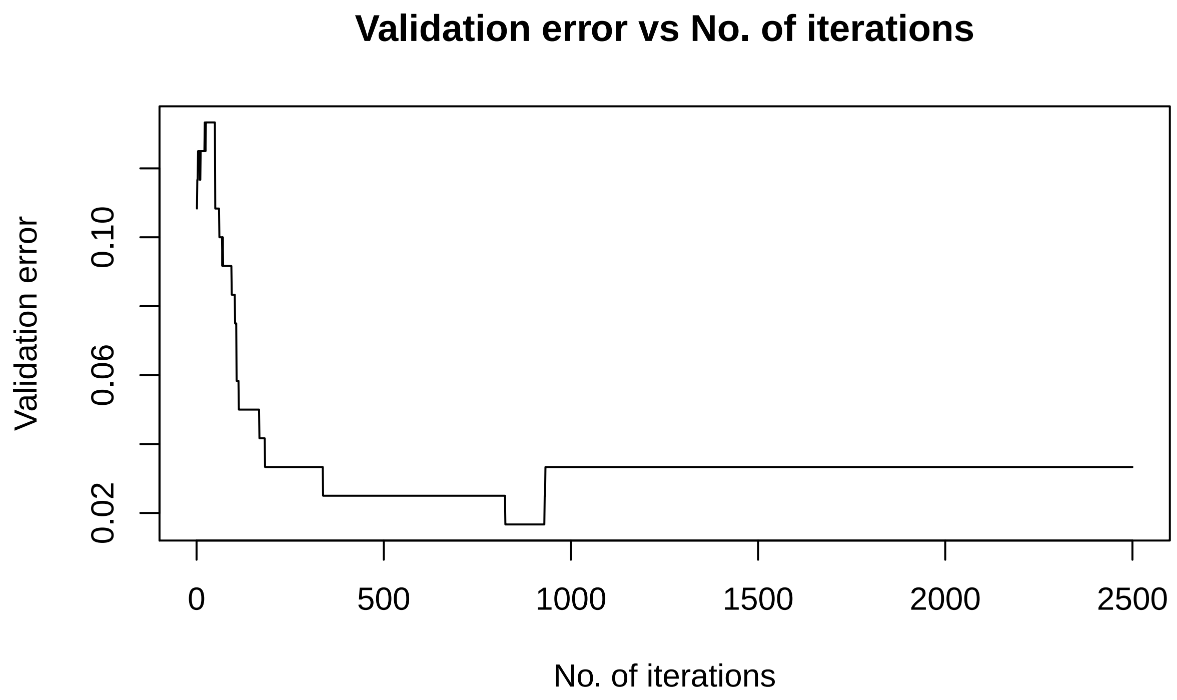
c1_y_centroid	c2_y_centroid	c3_y_centroid	tumor
108.5343	122.3085	120.1672	0
104.9151	123.3934	125.0761	0
114.7050	116.6865	119.4605	0
109.5807	117.9369	122.6980	0
112.0284	121.7142	113.4764	0

```
## [1] "Optimal no. of iterations: 825"
## [1] "Confusion matrix for training + validation data:"
##           Reference
## Prediction   0   1
##          0 340   0
##          1   0 340
## [1] "Training + validation accuracy: 1"
## [1] "Training + validation sensitivity: 1"
## [1] "Training + validation specificity: 1"
## [1] "Training + validation set AUC: 1"
## [1] "Confusion matrix for test data:"
##           Reference
## Prediction  0  1
##          0 59  0
##          1  1 60
## [1] "Test accuracy: 0.991666666666667"
## [1] "Test sensitivity: 0.983333333333333"
## [1] "Test specificity: 1"
## [1] "Test AUC: 1"
```

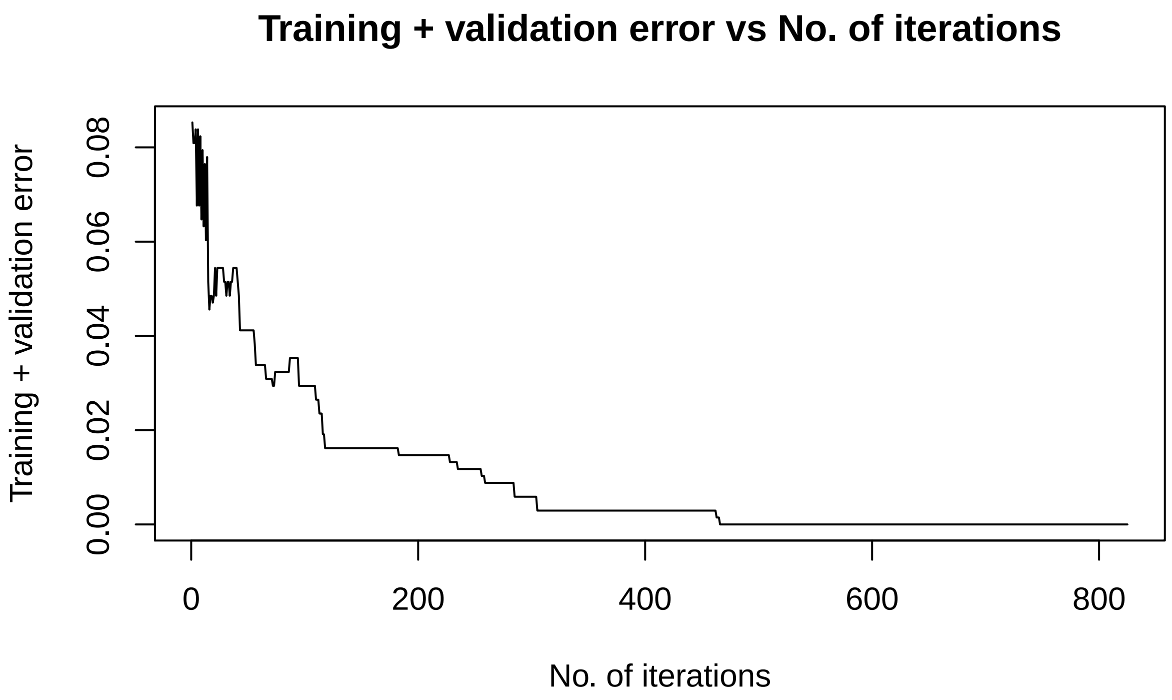


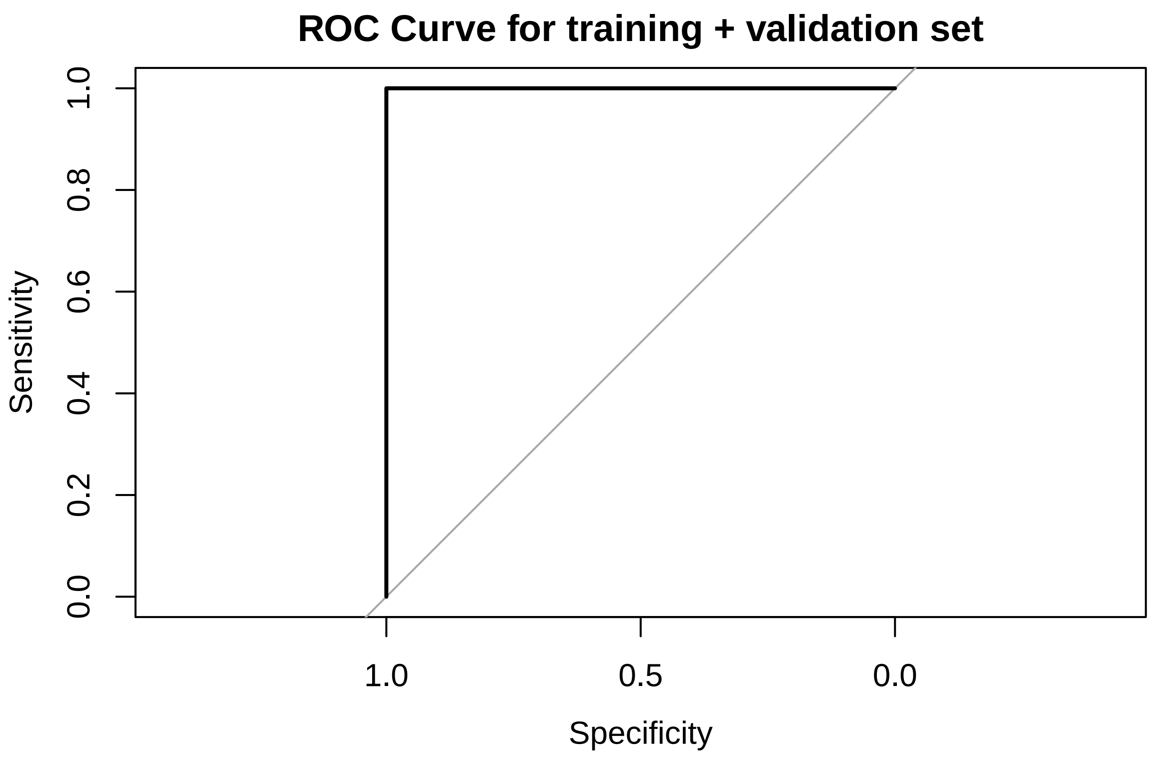

Input images
Non−tumor
Tumor
50
100
150
200
0
50
100
150

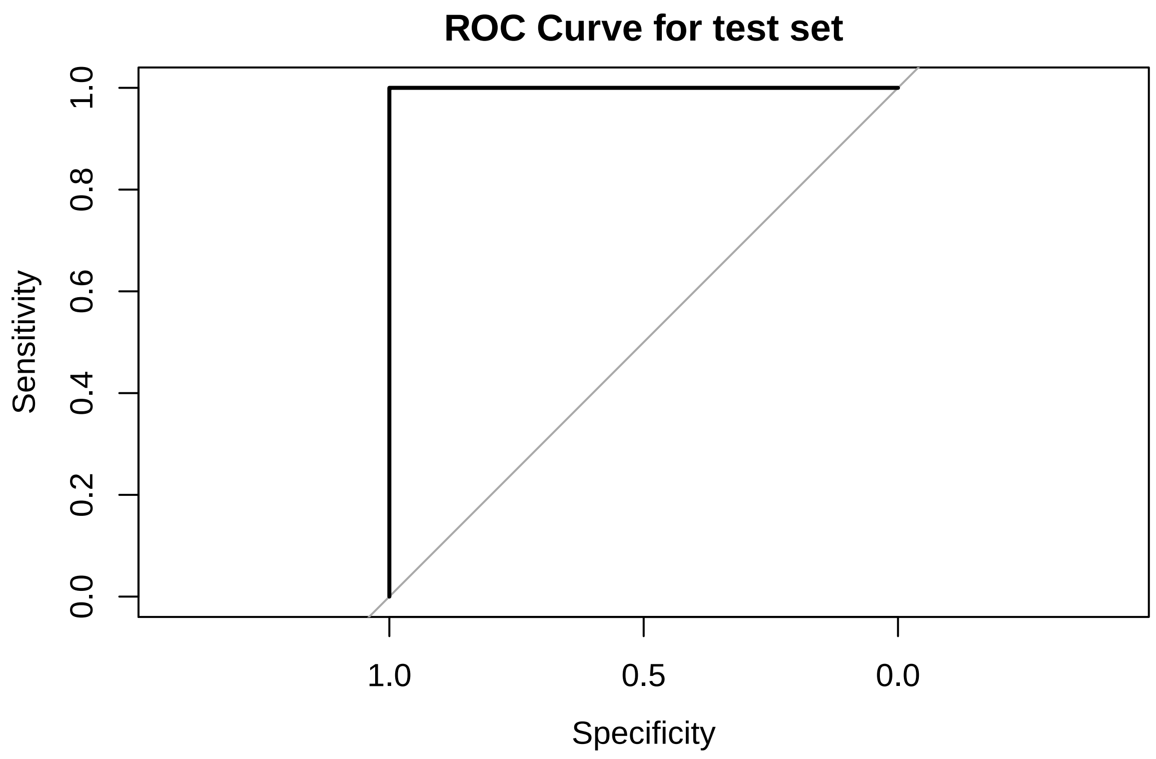

Clustered images
Non−tumor
Non−tumor
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.4
0.8



Validation error vs No. of iterations
Validation error
0.02
0.06
0.10
0
500
1000
1500
2000
2500
No. of iterations


Training + validation error vs No. of iterations
Training + validation error
0.00
0.02
0.04
0.06
0.08
0
200
400
600
800
No. of iterations


ROC Curve for training + validation set
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
Specificity
1.0
0.5
0.0


ROC Curve for test set
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
Specificity
1.0
0.5
0.0