EM Algorithm In Image Segmentation

Yuqing Zhang
Master of Data Science And Artificial Intelligence
University Of Waterloo
y3593zha@uwaterloo.ca

Abstract

In recent years, image segmentation has been significant in diverse fields, such as robot navigation, autonomous driving, and medical tumor diagnosis. Among all the segmentation models, unsupervised clustering models are of great interest because they are relatively easy to implement and do not need any training data.

In this project, we built two unsupervised models to do the image segmentation, the K-means algorithm and the Expectation-Maximization procedure (the Gaussian Mixture Model). Our result shows that the K-means algorithm works faster, but in general, Em is a better procedure because it can address two primary shortcomings of K-means, inflexible cluster shape and qualitative only estimate of a data point belonging to a cluster. We can use K-means to generate the initial Gaussian parameters for the input of the EM algorithm.

1 Introduction

Image segmentation has been of great significance in the realm of science. It can be applied in diverse fields, such as robot navigation, autonomous driving and medical tumor diagnosis [4]. Normally, image segmentation can be divided into two categories: divided using detected edges and divided by pixels [1].

In this project, we focused on the latter one - classify each pixel into several classes, thus forming a clustering problem. Since we did not divide each area manually beforehand, we did not have the labels of each pixel. Instead, we do the clustering based on the feature of each pixel (the RGB value). This problem can be attributed to an unsupervised problem [5].

Among the traditional machine learning methods, K-means is one of the most famous unsupervised models. In k-means, 'similar' data points are grouped together based on some distance measurement. We assign each data point into the nearest cluster.

However, K-means has some inevitable disadvantages, for example, it is sensible to the outliers. Also, if the clusters are overlapping or having a non-circular shape, K-means perform badly on them.

Gaussian mixture model (GMM) can be considered as an extension of K-means model, which clusters are models with Gaussian distributions. In order to describe the shape of each cluster, several parameters, such as mean and variance, are involved in this model. To fit the model, we can maximize the likelihood of the observed data using the expectation and maximization procedure (EM). In contrast to K-means, it will assign the data to each cluster by some soft probabilities.

In general, the data in the color image can be viewed as in the multivariate normal distribution. In the RGB image, the data has 3 channels, thus having 3 dimensions. In the project, we will try K-means and EM in the image and compare their performance.

The remainder of the article is organised as follows:

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

In section 2, we introduced K-means and EM models in more detail. In the Result Section, we compared the performance of two methods by some examples and plots. The last section will draw some conclusions.

2 Methodology

2.1 Notations

For convenience, we first define some frequently used notations in Table 1

Notation	Description
image_data	Dimension: height * width * n_features
data	Dimension: n_samples * n_features
n_samples	Number of samples in dataset (for an image, n_samples = number of pixels = height * width)
n_features	Dimension of a sample, i.e. channel of an image (for a RGB image, n_feature = 3)
n_clusters	Number of clusters into which the data is to be segmented

Table 1: Frequently used Notations

2.2 K-means

K-means is an unsupervised method that assigns each data point into the nearest group based on the distance (in most cases, Euclidean distance).

2.2.1 K-means: Content

Suppose we want to classify the data into K clusters. The number of clusters, K, must be set before the procedure. To do the clustering, the Euclidean distance between data and the cluster centroid needs to be minimized. Note this distance is not the physical distance in pixel, but the distance in pixel value.

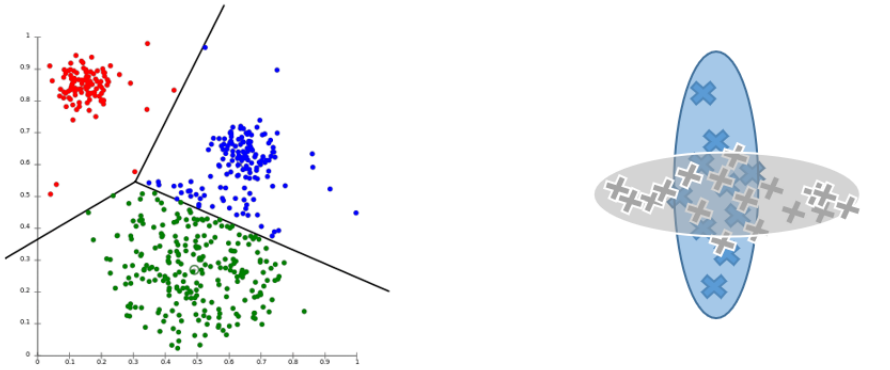
Suppose for a 3-channel RGB image. Denote the value of one data point as $x_i = \{x_{i1}, x_{i2}, x_{i3}\}$ and one cluster center as $c_j = \{c_{j1}, c_{j2}, c_{j3}\}$. Generally, each value should be an integer between 0 and 255. In some programming software, like R, it can be automatically replaced by the intensity between 0 and 1. The Euclidean distance between two points is $|| \ x_i - c_j \ || = \sqrt{\sum_{p=1}^{3}(x_{ip}) - c_{jp})^2}$.

The procedure can be summarized as follows:

- initial K cluster centers randomly
  1. suppose k random cluster centers are $\mu_1, \mu_2, \ldots, \mu_k$
  2. The dimension of cluster centers matrix is n_cluster $\times$ n_features.
- Repeat the procedure until the cluster labels of the image do not change anymore or change only under a tolerance value or it has reached the maximum number of iteration that we have defined.
  1. Calculate the distance matrix of all the data points to each cluster centroid.
  2. According to the distance matrix, each data is assigned to a cluster whose centroid is nearest to the data. Normally we use Euclidean Distance here. Mathematically, for a data point i and a cluster j, we can assign the label $c^{(i)}$ to the data point as

$$c^{(i)} := \arg\min_j \left\| x^{(i)} - \mu_j \right\|^2$$

  3. Update the cluster centers by taking the average of all the data belonging to that specific center. Each cluster center can be formulated as



(a) what K-means can do

(b) what K-means can not do

Figure 1: K-means illustration

$$\mu_i := \frac{\sum_{i=1}^{m} 1\left\{c_{(i)} = j\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{c_{(i)} = j\right\}}$$

2.2.2 K-means: Disadvantages

Although very simple to implement, K-means has several drawbacks.

1. First, the class shape is inflexible. Consider classifying points in a 2-d plane. The boundary of each class by K-means will be a straight line, as shown in Figure 1 (a).
   If the shapes of clusters are overlapped and the centers of multiple groups are quite close, as shown in Figure 1 (b), then K-means can not separate them. Also, suppose a cluster with the shape of 'donut'. K-means cannot deal with these non-regular shapes.
2. The estimate of K-means is qualitative. It can only provide the information about which cluster the data point goes to, but provide nothing about probabilities associated with each class.
3. The result of K-means depends much on the initial points. If a bad initial state is chosen, the clustering result could be sub-optimal. To prevent the unfortunate result convergent to the local minima, it is recommended that use different initial points and repeat the procedure multiple times to choose a result that happened most frequently.

2.3 EM Procedure In Clustering

2.3.1 GMM Model

Gaussian Mixture Model (GMM) is highly reminiscent of K-means, doing the clustering almost the same way K-means does, except it shows a mixed representation of multiple Gaussian probability distributions.

In GMM, we have some parameters describing each cluster. For each cluster c, we have Mean μ_c, variance σ_c, "size" π_c. Then the joint probability distribution can be written as the weighted average of these individual components [3]

$$p(x) = \sum_c \pi_c \mathcal{N}\left(x; \mu_c, \sigma_c\right)$$

A more generative way to illustrate this idea is to introduce the missing variable Z (latent variable).

To draw a sample from $P(X)$ we first select one of the components with discrete probability π, so components with large π are selected more often as in k-means. We'll view this as the assignment of that sample to one of the components and denoted by Z.

$$p(z = c) = \pi_c$$

Given the component assignment $z = c$, we can draw a value from X from the corresponding Gaussian distribution.

$$p(x \mid z = c) = \mathcal{N}(x; \mu_c, \sigma_c)$$

So together these two distributions make a joint model over X and Z. Discarding the value of Z gives a sample from the marginal $p(x)$ defined above.

For a Multivariate Gaussian model, the distribution can be written as [3]

$$\mathcal{N}(\underline{x}; \underline{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp\left\{ -\frac{1}{2} (\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) \right\}$$

where all the parameters are in the vector form.

Recall that if we were given data from a multivariate Gaussian distribution, the maximum likelihood estimate for the model parameters is simply the first moment of the data. The covariance estimate is the centered second moment of the data.

$$\hat{\mu} = \frac{1}{m} \sum_i x^{(i)}$$
$$\hat{\Sigma} = \frac{1}{m} \sum_i \left(x^{(i)} - \hat{\mu} \right)^T \left(x^{(i)} - \hat{\mu} \right)$$

2.3.2 EM: Content

EM proceeds iteratively in two steps.

In the first *Expectation* step, for each data point, calculate the probability generated by each component in the model. In the second *Maximization* step, the model adjusts the parameters of data to maximize the probability of the model generating these parameters [5].

E-step treats the gaussian parameters (μ, Σ, π) fixed. For each data point i and each cluster c, we compute the *responsibility value* r_{ic}, the probability that data i belongs to cluster c. [2] First, we compute its probability under model component c, which is a weighted Gaussian. Then we normalize it to sum to one.

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i; \mu_{c'}, \Sigma_{c'})}$$

In the M-step, we start with assignment probabilities r_{ic}. Then update the parameters (μ, Σ, π). For each cluster c, we update the parameters using the weighted data points. It observed some fraction weight r_{ic} of data point i.

- Total responsibility allocated to cluster c: $m_c = \sum_i r_{ic}$
- Updated weight π is the fraction of total assigned to cluster c: $\pi_c = \frac{m_c}{m}$
- Updated mean is the weighted mean of assigned data: $\mu_c = \frac{1}{m_c} \sum_i r_{ic} x^{(i)}$
- Updated covariance is the weighted covariance of assigned data (here use the new weighed mean): $\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} \left(x^{(i)} - \mu_c \right)^T \left(x^{(i)} - \mu_c \right)$

Each step strictly increases the log-likelihood of the model.

$$\log p(\underline{X}) = \sum_i \log \left[\sum_c \pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c) \right]$$

In practice, convergence here won't typically be as abrupt as it was in k-means. One usually stops once the parameters or the log-likelihood objective have stopped changing very much.

The output of the EM model for each data point is the soft probabilities for each cluster. If we want to choose a single cluster for an output "answer", we can choose the cluster with the highest responsibility (r_{ic}). We will discuss the shortcomings of EM later associated with the example.

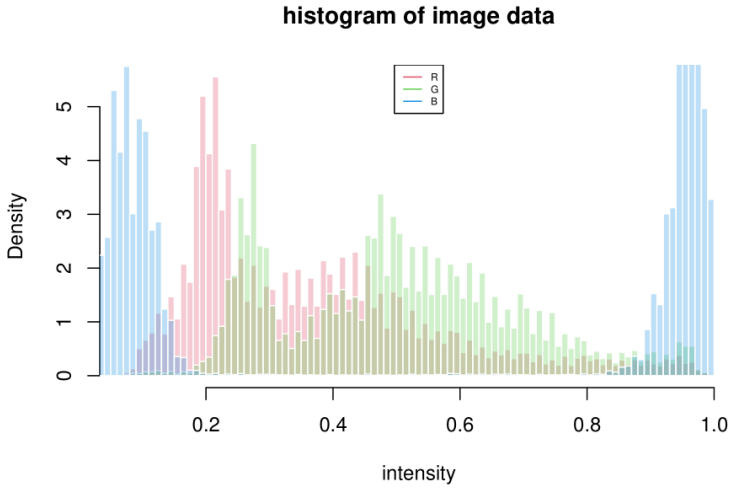
3 Results

In this project, we used image data as our input. Then we conducted two models (K-means and EM) on the data and compared their performance. Because both of the models are unsupervised, it is not possible to compare their accuracy or other quantitative metrics. Instead, we compared their convergence rate and the final output result.

3.1 Data

Take the following image as an example. The image contains blue sky, white clouds, and green grass. Our goal is to segment these three parts. The difficulty of the task is that the bright part of the grass may be interference to our model. It will confuse with the cloud part.

(a) original image data



(b) Histogram of image data: three histograms in one plot, each represents one channel

Figure 2: Image Data

The image size is height = 192, width = 254. It has RGB three channels. We reshaped the data to n_samples = 192*254, n_features = 3.

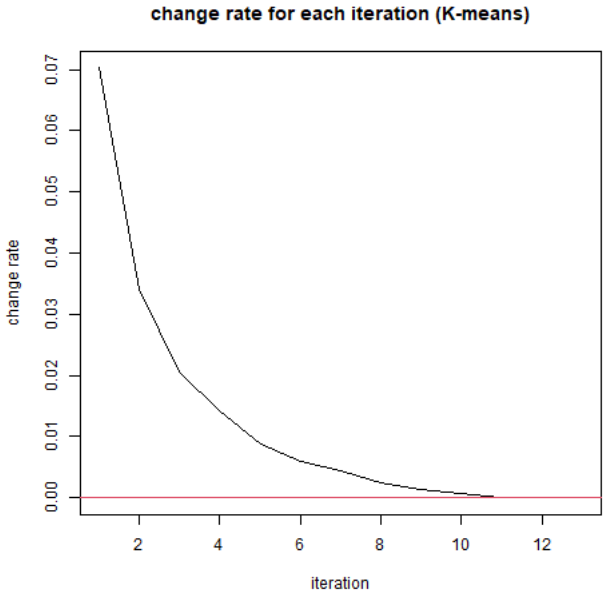


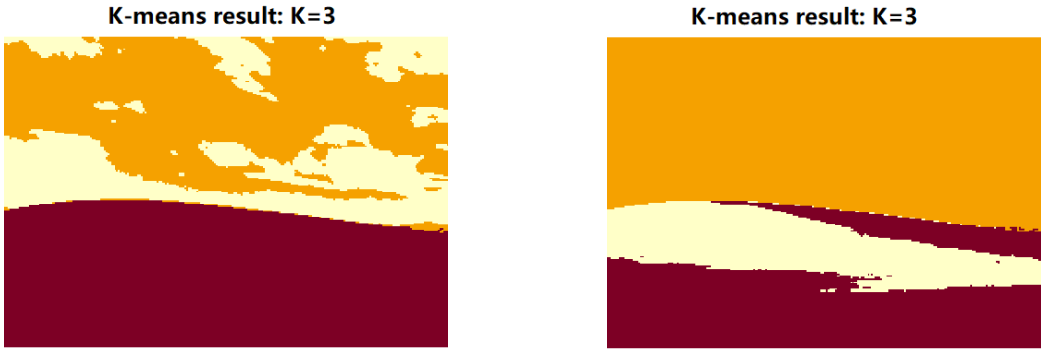
Figure 3: K-means convergence plot: x axis is the iteration; y axis is the label change rate of each iteration

To analyze the data, we plot the histogram for each channel. We can see some clear 'Multivariate Normal' patterns in each channel. Gaussian Mixture Model is worth trying.

3.2 K-Means Result

We implemented the K-means algorithm using R. We can see K-means convergences fast. It reached full convergence (the cluster labels do not change anymore) after 13 iterations (seen Figure 3). If we define the change tolerance value as 0.001 (that is, the label output of this iteration changes no more than 0.1 percent of the last iteration), then we only need 4 iterations to reach the convergence.

We can plot the label matrix to see the K-means segmentation result in Figure 4. Note that in K-means, initial cluster centers are defined randomly. This will largely affect our final segmentation output. Figure 4 plot two segmentation result with different initial points. The left one is close to our desired result, which classifies three groups of objects - grass, sky and cloud. In contrast, the right one was probably stuck in the sub-optimal result because of bad initial cluster center chosen.



(a) Result A (b) Result B

Figure 4: K-Means Result: random initial centroids

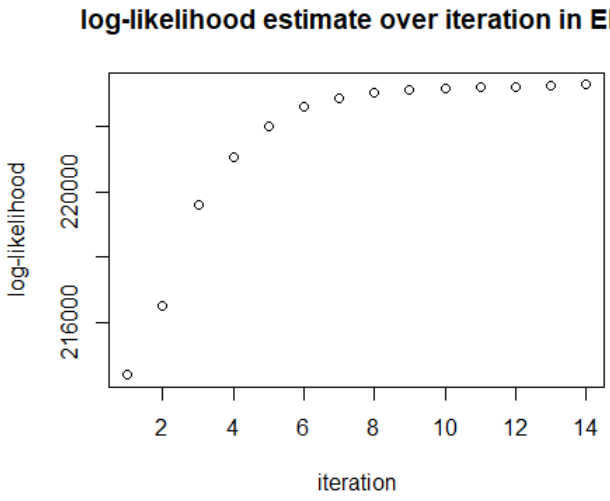


Figure 5: EM convergence plot: log likelihood over iteration

3.3 EM Result

For EM, the initial parameters can be generated randomly, or generated using the K-means results.

The dimensions of each parameter:

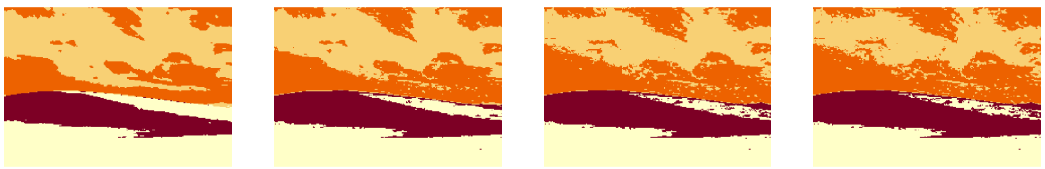
- μ - n_dimension * n_cluster
- Σ - n_dimension * n_dimension * n_cluster
- π - 1 * n_cluster

3.3.1 EM Result With K-means Initial Parameters

After conducting the K-means for several iterations, we can use the intermediate result to generate the Gaussian parameters.

We separate the data into k groups according to the labels. Then calculate the (column) mean and covariance matrix for each cluster. The updated weights π are the proportions of each label to total labels. Then we execute the EM algorithm as discussed in section 2.3. The log likelihood over each iteration is plotted in Figure 5. We can notice that the increase of log likelihood is gradually becoming slower with the evolution.

Here we use k = 4 to illustrate the performance between K-means and EM. The segmentation result for different iteration is in Figure 6. As the EM evolves, we can see more texture in the classification output. The classification boundary is more flexible.



(a) K means Initial (b) EM (4 iterations) (c) EM (8 iterations) (d) EM (12 iterations)

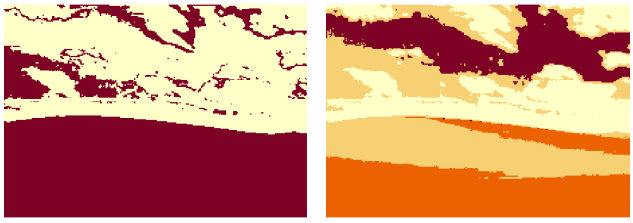
Figure 6: EM Result: K-means initial parameters

3.3.2 EM Result With Random Initial Parameters

When the initial parameters are generated randomly, the convergence rate is slower than the K-means initial method. It usually takes 20 iterations for convergence, compared to 12 using K-means initial parameters.

The result is easier to be stuck in the local minima, thus performing a bad segmentation. It tends to merge two or several clusters, i.e. one or more clusters has vanished and has no data points assignments.

Here are some results with the randiom initial parameters in Figure 7.

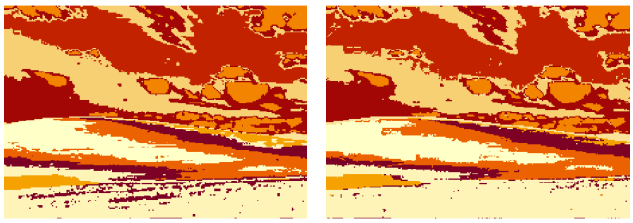


(a) vanishing clusters (b) not separated thoroughly

Figure 7: EM Result: random initial parameters

3.4 EM Result: When The Number Of Cluster Is Large

We also tried a larger cluster number. When $k = 10$, we can see there is not much difference between K means and EM result. The segmentation is very coarse. From the bottom of the image, we can see many small and discrete clusters. This may be since that Euclidean Distance is not the best metric for image segmentation [4].



(a) k means: k = 10 (b) EM: k = 10

Figure 8: Kmeans and EM Result: k = 10

3.5 Summary

From the above experiments, we can conclude that EM is better than K-means for its flexible classification boundary. But it still does not reach our expectation to separate the part of "sky" , "cloud" and "grass" clearly. It indicates pixel-based method is not competitive enough for the goal of object detection, compared to edge detection methods. There is still some limitation of the EM algorithm.

K-means convergences in less iterations, and for each iteration, it consumes less time. EM algorithm needs large computation resources when the is image size is large.

4 Discussion

From the above experiments, we can conclude that EM is better than K-means for its flexible classification boundary. But it still does not reach our expectation to separate the part of "sky", "cloud" and "grass" clearly. There is still some limitation of the EM algorithm.

4.1 Conclusion

In this project, we built two unsupervised models to do the image segmentation, the K-means algorithm and the Expectation-Maximization procedure (the Gaussian Mixture Model).

K-means algorithm works faster, but the results are not so good. EM runs in more time because it needs to compute some complex matrix on the whole set of data in each cluster on each iteration.

In general, Em is a better procedure because it can address two primary shortcomings of K-means, inflexible class shape and qualitative only estimate of a data point belonging to a cluster. We can use K-means to generate the initial Gaussian parameters for the input of the EM algorithm.

4.2 Limitation

In the example we stated above, the segmentation result of EM still has a lot to improve. The algorithm failed to separate the "cloud" to other parts correctly. Although EM is an improvement of K-means, admittedly, there are still some limitations of EM:

1. The final output depends largely on the initial parameters.
2. The convergence rate is relatively slow, as it needs a complex parameter estimation process.
3. The phenomenon of vanishing cluster is easy to occur. Some clusters may merge together during the evolution.
4. When the number of clusters increases, the segmentation result is very coarse. There are so many small and discrete clusters. This may indicate that Euclidean Distance is not the best measurement for image segmentation.

4.3 Future Work

Due to the limitation of time and computational resources, our project only used several small RGB image as examples and implemented two relatively simple procedures. For more general conclusions, there is much we can do in the future.

1. Change the input to the grey-scale image and see the result. It has only one channel, so it should be easier to implement.
2. Edge-detection methods can be tried to do the image segmentation and compare the performance. For the objective of object detection, edge-detection methods may be better than pixel-based method because the former one tend to classify each group clearly.
3. We can try some Graph Partitioning approaches to do the image segmentation, such as the Normalized Cut [6], and compare the result with the unsupervised learning models.
4. We can try other machine learning techniques (including supervised ones) to compare with these methods, such as SVM or deep learning models.
5. We can optimize the performance of image segmentation with some image preprocessing steps, such as add some reasonable noise to the data or blur the data.
6. We can try different distance measurement instead of the euclidean distance to calculate the *similarities*. Besides, we can calculate the *dissimilarities*. between each cluster in order to separate them more clearly.

References

[1] Digital image processing techniques: Michael p. ekstrom, ed. academic press, new york, 1984, xiii + 372 pp. *Computer vision, graphics, and image processing*, 29(3):394–394, 1985.

[2] Hani El Assaad, Allou Samé, Gérard Govaert, and Patrice Aknin. A variational expectation–maximization algorithm for temporal data clustering. *Computational statistics data analysis*, 103:206–228, 2016.

[3] Daniel Povey, Lukáš Burget, Mohit Agarwal, Pinar Akyazi, Feng Kai, Arnab Ghoshal, Ondřej Glembek, Nagendra Goel, Martin Karafiát, Ariya Rastrow, Richard C Rose, Petr Schwarz, and Samuel Thomas. The subspace gaussian mixture model—a structured model for speech recognition. *Computer speech language*, 25(2):404–439, 2011.

[4] Suman Tatiraju and Avi Mehta. Image segmentation using k-means clustering, em and normalized cuts. *Department of EECS*, 1:1–7, 2008.

[5] T. Yamazaki. Introduction of em algorithm into color image segmentation. 1998.

[6] Jiandong Yin, Hongzan Sun, Jiawen Yang, and Qiyong Guo. Automated detection of the arterial input function using normalized cut clustering to determine cerebral perfusion by dynamic susceptibility contrast-magnetic resonance imaging. *Journal of magnetic resonance imaging*, 41(4):1071–1078, 2015.

Appendix

Yuqing Zhang

2021/8/12

```
library(jpeg)
library(png)
library(ForeCA)
```

```
## This is 'ForeCA' version 0.2.7. See https://github.com/gmgeorg/ForeCA for latest updates and citatio
## May the ForeC be with you.
```

```
library(mvtnorm)
library('plot.matrix')
library(ramify) # matrix
```

```
##
## Attaching package: 'ramify'
```

```
## The following object is masked from 'package:graphics':
##
##     clip
```

```
# change the img directory here
img_dir <- "windows.png"
# change the output name here
name <- "c4.png"

n_cluster <- 4
max_iteration <- 30
err <- 0.001
```

read img

```
readimg <- function(img_dir){
  img_type = strsplit(img_dir, split = ".",
                      fixed = TRUE)[[1]][2]
  if (img_type == "png"){
   img = readPNG(img_dir, native = FALSE)
  }else {
    img = readJPEG(img_dir, native = FALSE)}
  return(img)
}

img = readimg(img_dir)
dim(img)
```

```
## [1] 192 254   4
```

```
h <- dim(img)[1]
w <- dim(img)[2]
channel <- dim(img)[3]

n_samples <- h * w
# reshape to n_samples * channel
img_data <- matrix(img, n_samples, channel)
if (channel == 4) {
  img_data <- img_data[,1:3]
  channel <- 3}
n_dimension <- channel
labels <- numeric(n_samples)
```

initial kmeans

```
n_samples <- h * w
# reshape to n_samples * channel
img_data <- matrix(img, n_samples, channel)
if (channel == 4) img_data <- img_data[,1:3]
labels <- numeric(n_samples)

# initial centroids
centroids_idx <- sample(1:n_samples, n_cluster)
centroids <- img_data[centroids_idx,]

# initial distance
distance <- matrix(0, n_samples, n_cluster)

calculate_distance <- function(data, centroids){
  distance <- matrix(0, n_samples, n_cluster)
  for (i in 1:n_samples){
    for (j in 1:n_cluster){
      distance[i,j] <- dist(rbind(data[i,],
                                  centroids[j,]))
    }
  }
  return(distance)
}

update_centroids <- function(data, label){
  for (j in 1:n_cluster){
    temp = data[label ==j,]
    count = nrow(temp)
    centroids[j,] = apply(temp, MARGIN = 2, FUN=sum)/count
  }
  centroids
}
```

run kmeans

```
for (iter in 1:max_iteration){
  distance = calculate_distance(img_data, centroids)
```

```
  labels_old = labels
  labels = argmin(distance)
  print(paste0("iter: ", iter, " change rate:",
               1-mean(labels == labels_old)))
  if (mean(labels == labels_old) > 1- err){
    print(paste0('Converged at iteration: ',iter))
    break
  }
  centroids = update_centroids(img_data, labels)
}
```

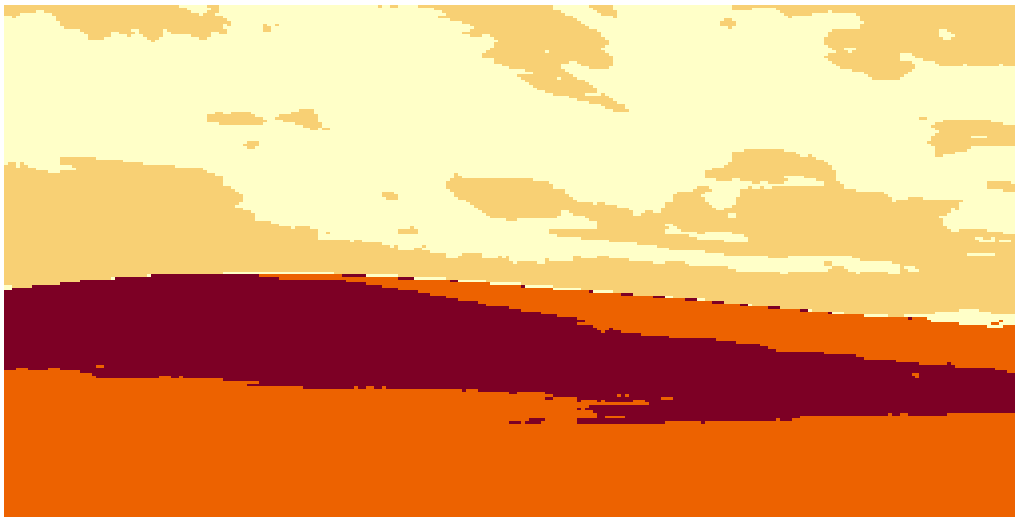
```
## [1] "iter: 1 change rate:1"
## [1] "iter: 2 change rate:0.128916502624672"
## [1] "iter: 3 change rate:0.0536417322834646"
## [1] "iter: 4 change rate:0.0284817913385826"
## [1] "iter: 5 change rate:0.0168963254593176"
## [1] "iter: 6 change rate:0.0107037401574803"
## [1] "iter: 7 change rate:0.00707431102362199"
## [1] "iter: 8 change rate:0.00531085958005251"
## [1] "iter: 9 change rate:0.00311679790026242"
## [1] "iter: 10 change rate:0.001701935695538"
## [1] "iter: 11 change rate:0.000943241469816281"
## [1] "Converged at iteration: 11"
```

```
m <- matrix(labels, h, w)
rotate <- function(x) t(apply(x, 2, rev))
png(filename=paste0("results/kmeans", name))
par(mar = c(5,0,5,0))
image(rotate(m), useRaster=TRUE, axes=FALSE)
dev.off()
```

```
## pdf
##   2
```

```
image(rotate(m), useRaster=TRUE, axes=FALSE)
title(main = "k means result")
```

k means result



initial parameter

```
random_init <- function(img_data, labels, n_cluster){
  mean_matrix <- matrix(runif(n_cluster * n_dimension),
                        n_cluster, n_dimension)
  cov_para <- runif(n_cluster)
  cov_matrix <- array(dim = c(n_dimension, n_dimension, n_cluster))
  pi_list <- runif(n_cluster)
  pi_list <- pi_list/sum(pi_list)
  for (j in 1:n_cluster){
    cov_matrix[,,j] <- cov_para[j] * diag(n_dimension)
  }
  return(list(mean_matrix, cov_matrix, pi_list))
  }

kmeans_init <- function(img_data, labels, n_cluster){
  n_dimension <- dim(img_data)[2]
  cov_matrix <- array(dim = c(n_dimension, n_dimension, n_cluster))
  pi_list <- numeric(n_cluster)
  mean_matrix <- matrix(0, n_cluster, n_dimension)
  for (j in 1:n_cluster){
    mean_matrix[j,] <- apply(img_data[labels == j,], 2, mean )
    cov_matrix[,,j] <- cov(img_data[labels == j,])
    pi_list[j] <- mean(labels == j)
  }
```

```
  return(list(mean_matrix, cov_matrix, pi_list))
  }



#### initial parameter
# chose whether random or kmeans

# result_temp <- random_init(img_data, labels, n_cluster)
result_temp <- kmeans_init(img_data, labels, n_cluster)
mean_matrix <- result_temp[[1]]
cov_matrix <- result_temp[[2]]
pi_list <- result_temp[[3]]

# update responsibilities
update_responsibilities <- function(img_data, mean_matrix, cov_matrix, pi_list){
  # output matrix: n_samples * n_cluster
  prob_matrix <- matrix(0, n_samples, n_cluster)
    for (i in 1:n_samples) {
      for (j in 1:n_cluster){
        prob_matrix[i,j] <- pi_list[j] *
          dmvnorm(img_data[i,], mean = mean_matrix[j,],
                  sigma = cov_matrix[,,j])
      }
    }
  # normalize
  temp_sum <- apply(prob_matrix, 1, sum)
  r_matrix <- prob_matrix/temp_sum
  r_matrix[is.na(r_matrix)] <- 1/n_cluster
  return(list(prob_matrix, r_matrix))
}
```

```
update_pi <- function(m_c){
  # output vector:  n_cluster
  return(m_c/n_samples)
}

update_means <- function(img_data, r_matrix, m_c){
# output matrix: n_cluster * n_dimension
t(t(img_data) %*% r_matrix) / m_c}

update_cov <- function(img_data, r_matrix, m_c){
  # output matrix: n_dimension * n_dimension * n_cluster
  cov_matrix <- array(dim = c(n_dimension, n_dimension, n_cluster))

  for (j in 1:n_cluster){
    temp_j = matrix(0, n_dimension, n_dimension)
    for (i in 1:n_samples){
        temp_j <- temp_j + r_matrix[i,j] * ((img_data[i,] - mean_matrix[j,]) %*% t(img_data[i,] - mean_
      }
    cov_matrix[,,j] = temp_j / m_c[j]
  }
  return(cov_matrix)
  }
```

run EM

```
# EM
label <- labels
count <- 0
log_lik <- 2
log_lik_old <- 1

log_lik_list <- numeric(max_iteration)
while (count < max_iteration &
       abs(log_lik/log_lik_old) - 1> err){

  label_old <- label
  log_lik_old <- log_lik
  count <- count + 1
  print(paste0("count: ",count))
  resp_result <- update_responsibilities(
    img_data, mean_matrix,
    cov_matrix, pi_list)
  r_matrix <- resp_result[[2]]
  prob_matrix <- resp_result[[1]]
  log_lik <- sum(log(apply(prob_matrix, 1, sum)))
  log_lik_list[count] <- log_lik

  print(paste0("log_likelihood: ",log_lik))
  m_c <- apply(r_matrix, 2, sum)
  pi_list <- update_pi(m_c)
  mean_matrix <- update_means(img_data, r_matrix, m_c)
  cov_matrix <- update_cov(img_data, r_matrix, m_c)
  label <- apply(r_matrix, 1, which.max)

  print(mean(label == label_old))

  for (j in 1:n_cluster){
    print(paste(j, sum(label == j), sep = ":"))
  }

  if (count %% 2 == 0){
  m <- matrix(label, h, w)
  png(filename=paste0("results/em_c4_", count, ".png"))
  par(mar = c(5,0,5,0))
  image(rotate(m), useRaster=TRUE, axes=FALSE)
  dev.off()}
}
```

```
## [1] "count: 1"
## [1] "log_likelihood: 214405.965095081"
## [1] 0.9870612
## [1] "1:16424"
## [1] "2:10912"
## [1] "3:13117"
## [1] "4:8315"
## [1] "count: 2"
## [1] "log_likelihood: 216648.742660355"
```

```
## [1] 0.980561
## [1] "1:16041"
## [1] "2:11275"
## [1] "3:12817"
## [1] "4:8635"
```

```
## [1] "count: 3"
## [1] "log_likelihood: 219668.665279032"
## [1] 0.9753527
## [1] "1:15266"
## [1] "2:12000"
## [1] "3:12642"
## [1] "4:8860"
## [1] "count: 4"
## [1] "log_likelihood: 221134.124709051"
## [1] 0.9820784
## [1] "1:14722"
## [1] "2:12501"
## [1] "3:12500"
## [1] "4:9045"
```

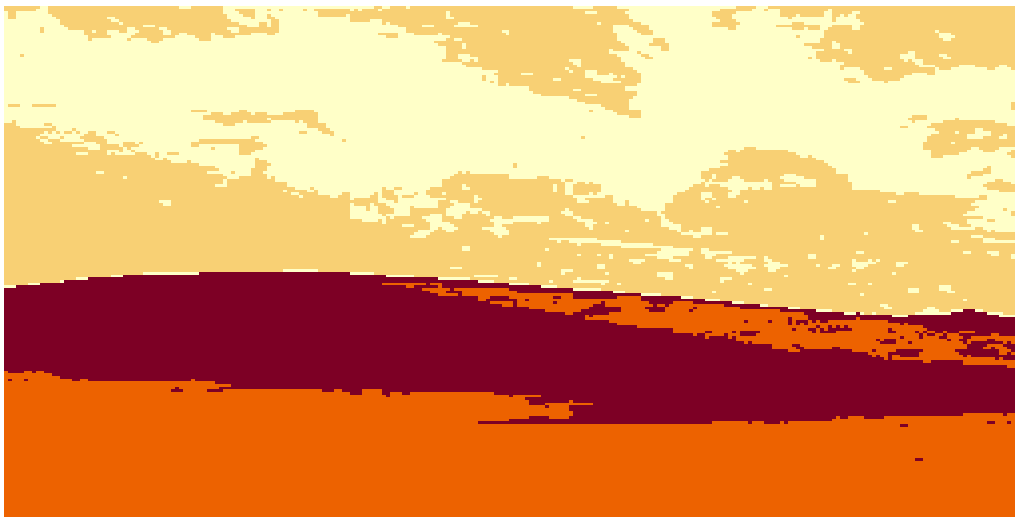
```
## [1] "count: 5"
## [1] "log_likelihood: 222046.039565183"
## [1] 0.9862
## [1] "1:14241"
## [1] "2:12941"
## [1] "3:12402"
## [1] "4:9184"
## [1] "count: 6"
## [1] "log_likelihood: 222634.698533407"
## [1] 0.9889477
## [1] "1:13881"
## [1] "2:13275"
## [1] "3:12313"
## [1] "4:9299"
```

```
## [1] "count: 7"
## [1] "log_likelihood: 222907.35432972"
## [1] 0.9901165
## [1] "1:13556"
## [1] "2:13583"
## [1] "3:12228"
## [1] "4:9401"
## [1] "count: 8"
## [1] "log_likelihood: 223056.586610585"
## [1] 0.9940945
## [1] "1:13345"
## [1] "2:13781"
## [1] "3:12191"
## [1] "4:9451"
```

```
m <- matrix(label, h, w)
rotate <- function(x) t(apply(x, 2, rev))

# par(mar = c(5,0,5,0))
image(rotate(m), useRaster=TRUE, axes=FALSE)
```

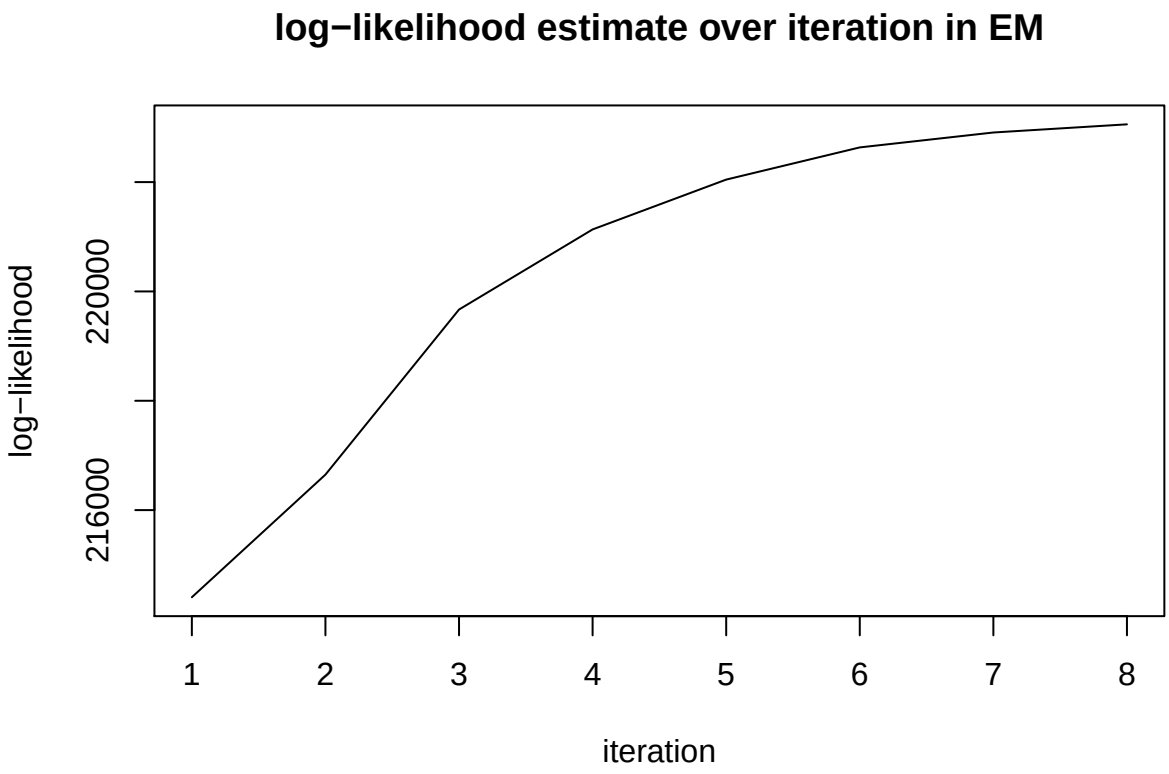
```
title(main = "EM result")
```




```
log_lik_list[1:count]
```

```
## [1] 214406.0 216648.7 219668.7 221134.1 222046.0 222634.7 222907.4 223056.6
```

```
plot(1:count, log_lik_list[1:count], type = 'l',
     xlab = "iteration",
     ylab = "log-likelihood",
     main = "log-likelihood estimate over iteration in EM")
```




```
png(filename=paste0("plot-", name))
plot(1:count, log_lik_list[1:count], type = 'l')
dev.off()
```

```
## pdf
##   2
```