Homework 7

CS 498, Spring 2018, Xiaoming Ji

Problem 1

EM Topic models The UCI Machine Learning dataset repository hosts several datasets recording word counts for documents here. You will use the NIPS dataset. You will find (a) a table of word counts per document and (b) a vocabulary list for this dataset at the link. You must implement the multinomial mixture of topics model.

```
#Load data
header = readLines("docword.nips.txt", n = 2)
DocCount = as.integer(header[1])
VocabSize = as.integer(header[2])
df = read.table("docword.nips.txt", skip=3)

DocVectors = matrix(0, DocCount, VocabSize)
for (i in 1:(dim(df)[1])){
  DocVectors[df[i, 1],df[i, 2]] = df[i, 3]
}

VocabData = read.table("vocab.nips.txt", colClasses = "character")
```

- Cluster this to 30 topics, using a simple mixture of multinomial topic model.

We have,

$$\omega_{i,j} = \frac{\left[\prod_k p_{j,k}^{x_{i,k}}\right] \pi_j}{\sum_l \left[\prod_k p_{l,k}^{x_{i,k}}\right] \pi_l}$$

We transform this equation to logsumexp form to benefit calculation

$$\log(\omega_{i,j}) = \sum_k (x_{i,k} \log p_{j,k}) + \log \pi_j - LogSumExp_l(\sum_k (x_{i,k} \log p_{l,k}) + \log \pi_l)$$

```
library(matrixStats)
```

```
## Warning: package 'matrixStats' was built under R version 3.4.3
```

```
logDotExp = function(A, B) {
  max_a = max(A)
  max_b = max(B)
  exp_a = exp(A - max_a)
  exp_b = exp(B - max_b)

  C = log(exp_a %*% exp_b)

  return (C + max_a + max_b)
}

topic_EM = function(docs, vocab.size, topic.count, converge.threshold = 1e-7){
  #Initialize P and Pi
  doc_count = dim(docs)[1]
```

```
doc_length = rowSums(docs)
P  = matrix(0, topic.count, vocab.size)
pi = rep(0, topic.count)
W = matrix(0, doc_count, topic.count)
old_W  = W
log_W = W

#Randomly sample docs
i_doc = sample(1:doc_count, doc_count)

bat_size = round(doc_count / topic.count)
indexes = c(seq(from = 1, to = doc_count, by = bat_size), doc_count + 1)

for (j in 1:topic.count) {
  i_bat = i_doc[indexes[j]:(indexes[j + 1] - 1)]

  #count words and add smoothing
  words = colSums(docs[i_bat,])

  #Calculate P and Pi
  P[j,]  = words / sum(words)
  pi[j] = length(i_bat) / doc_count
}

#Iterately perform E and M steps
for (iter in 1:1000) {
  #P smoothing
  P = P + 1e-7
  P = P / rowSums(P)
  log_P = log(P)

  log_pi = log(pi)

  #E Step
  X_log_P = docs %*% t(log_P)
  for (i in 1:doc_count) {
    log_W[i, ] = X_log_P[i, ] + log_pi
    log_W[i, ] = log_W[i, ] - logSumExp(log_W[i, ])
  }

  W = exp(log_W)
  t_W = t(W)

  #M Step
  P = (t_W %*% docs) / c(t_W %*% doc_length)
  pi = colSums(W) / doc_count

  #check convergence
  d = max(abs(W - old_W))
  print(paste("Iteration=", iter, ", D=", d, sep=""))
  if(d < converge.threshold) break()

  old_W = W
```

```
  }

  return (list(P=P, pi=pi, iteration = iter))
}

log_topic_EM = function(docs, vocab.size, topic.count, converge.threshold = 1e-7){
  #Initialize P and Pi
  doc_count = dim(docs)[1]
  doc_length = rowSums(docs)
  P  = matrix(0, topic.count, vocab.size)
  pi = rep(0, topic.count)
  log_W = matrix(-10000, doc_count, topic.count)
  old_log_W = log_W
  #converge.threshold = abs(log(converge.threshold, 10))

  #Randomly sample docs
  i_doc = sample(1:doc_count, doc_count)

  bat_size = round(doc_count / topic.count)
  indexes = c(seq(from = 1, to = doc_count, by = bat_size), doc_count + 1)

  for (j in 1:topic.count) {
    i_bat = i_doc[indexes[j]:(indexes[j + 1] - 1)]

    #count words and add smoothing
    words = colSums(docs[i_bat,]) + (0.01 / topic.count) #smoothing

    #Calculate P and Pi
    P[j,]  = words / sum(words)
    pi[j] = length(i_bat) / doc_count
  }

  log_P = log(P)
  log_pi = log(pi)

  log_docs = log(docs)
  log_docs[log_docs < -10000] = -10000
  doc_count_zeros = matrix(0, doc_count, 1)

  for (iter in 1:10000) {
    #E Step
    X_log_P = docs %*% t(log_P)
    for (i in 1:doc_count) {
      log_W[i, ] = X_log_P[i, ] + log_pi
      log_W[i, ] = log_W[i, ] - logSumExp(log_W[i, ])
    }

    t_log_W = t(log_W)

    #M Step
    log_P = logDotExp(t_log_W, log_docs) - c(logDotExp(t_log_W, log(doc_length)))
    log_pi = logDotExp(t_log_W, doc_count_zeros) - log(doc_count)
```

```
    log_P[log_P < -10000] = -10000

    #check convergence
    d = max(abs(exp(log_W) - exp(old_log_W)))
    print(paste("Iteration=", iter, ", D=", d, sep=""))
    if(d < converge.threshold) break()

    old_log_W = log_W
  }

  return (list(P=exp(log_P), pi = exp(log_pi), iteration = iter))
}
```

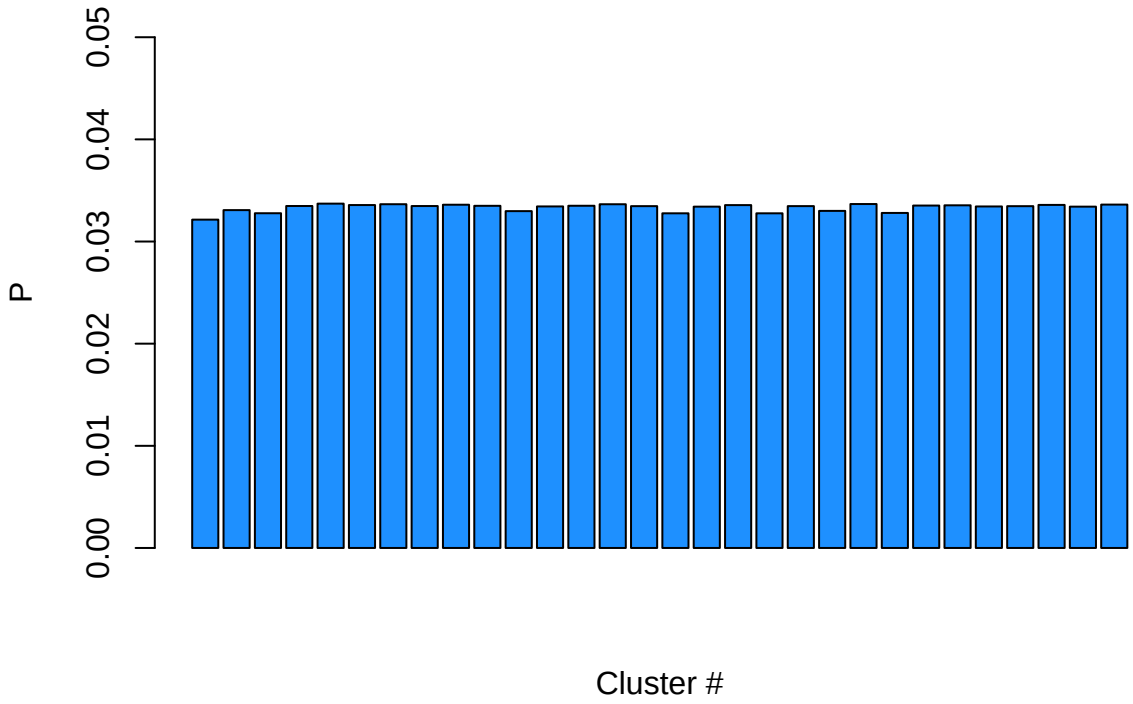
```
TopicCount = 30

set.seed(100000) #Only for homework, set seed for consistent output
r=log_topic_EM(DocVectors, VocabSize, TopicCount)
```

```
## [1] "Iteration=1, D=1"
## [1] "Iteration=2, D=0.0492107691689457"
## [1] "Iteration=3, D=0.00266766012869049"
## [1] "Iteration=4, D=0.000146479017300791"
## [1] "Iteration=5, D=1.02494542302267e-05"
## [1] "Iteration=6, D=8.23336358479754e-07"
## [1] "Iteration=7, D=6.65843795100152e-08"
```

```
#r=topic_EM(DocVectors, VocabSize, TopicCount)
```

- Produce a graph showing, for each topic, the probability with which the topic is selected.

```
barplot(c(r$pi), xlab = "Cluster #", ylab = "P",
        col = "dodgerblue", ylim = c(0, 0.05))
```



- Produce a table showing, for each topic, the 10 words with the highest probability for that topic.

```
top_10 = matrix("", TopicCount, 10)
P = r$P

for (i in 1:TopicCount) {
  for (j in 1: 10) {
    top_word = which.max(P[i,])
    P[i, top_word] = 0
    top_10[i, j] = VocabData[top_word,]
  }
}

print(top_10)
```

```
##       [,1]      [,2]       [,3]       [,4]         [,5]         [,6]
##  [1,] "network" "learning" "input"    "function"   "neural"     "unit"
##  [2,] "network" "model"    "data"     "set"        "training"   "algorithm"
##  [3,] "model"   "network"  "learning" "function"   "training"   "set"
##  [4,] "network" "model"    "input"    "learning"   "neural"     "system"
##  [5,] "network" "function" "model"    "input"      "neuron"     "neural"
##  [6,] "network" "model"    "learning" "input"      "function"   "unit"
##  [7,] "network" "model"    "learning" "set"        "function"   "algorithm"
##  [8,] "network" "learning" "set"      "function"   "model"      "neural"
##  [9,] "network" "model"    "input"    "neural"     "function"   "algorithm"
## [10,] "network" "model"    "input"    "learning"   "function"   "unit"
## [11,] "network" "learning" "model"    "function"   "input"      "algorithm"
## [12,] "network" "function" "set"      "model"      "learning"   "data"
## [13,] "network" "model"    "neural"   "learning"   "system"     "algorithm"
## [14,] "network" "model"    "function" "input"      "learning"   "neural"
## [15,] "network" "model"    "learning" "input"      "system"     "neural"
## [16,] "network" "set"      "learning" "data"       "neural"     "function"
## [17,] "network" "model"    "function" "input"      "learning"   "data"
## [18,] "network" "learning" "model"    "function"   "algorithm"  "input"
## [19,] "network" "model"    "function" "algorithm"  "learning"   "set"
## [20,] "network" "input"    "model"    "function"   "learning"   "algorithm"
## [21,] "network" "model"    "learning" "function"   "neural"     "input"
## [22,] "network" "model"    "neural"   "input"      "training"   "set"
## [23,] "network" "learning" "model"    "function"   "neural"     "algorithm"
## [24,] "network" "learning" "model"    "function"   "algorithm"  "input"
## [25,] "network" "model"    "learning" "function"   "data"       "neural"
## [26,] "network" "model"    "function" "learning"   "input"      "neural"
## [27,] "model"   "network"  "learning" "input"      "system"     "data"
## [28,] "network" "model"    "learning" "function"   "input"      "data"
## [29,] "network" "learning" "model"    "function"   "system"     "algorithm"
## [30,] "network" "input"    "learning" "algorithm"  "model"      "set"
##       [,7]        [,8]        [,9]        [,10]
##  [1,] "model"     "set"       "system"    "output"
##  [2,] "learning"  "neural"    "input"     "function"
##  [3,] "error"     "input"     "unit"      "data"
##  [4,] "function"  "data"      "pattern"   "set"
##  [5,] "set"       "system"    "learning"  "algorithm"
##  [6,] "neural"    "system"    "algorithm" "set"
##  [7,] "input"     "data"      "unit"      "system"
##  [8,] "input"     "training"  "error"     "weight"
```

```
##  [9,] "unit"      "learning"  "system"    "set"
## [10,] "weight"    "system"    "neural"    "data"
## [11,] "system"    "neural"    "output"    "weight"
## [12,] "input"     "algorithm" "training"  "neural"
## [13,] "function"  "data"      "problem"   "input"
## [14,] "system"    "set"       "training"  "data"
## [15,] "neuron"    "set"       "function"  "algorithm"
## [16,] "training"  "system"    "error"     "model"
## [17,] "set"       "training"  "unit"      "algorithm"
## [18,] "neural"    "system"    "set"       "unit"
## [19,] "neural"    "input"     "result"    "data"
## [20,] "set"       "data"      "neural"    "unit"
## [21,] "output"    "system"    "set"       "algorithm"
## [22,] "unit"      "learning"  "output"    "system"
## [23,] "input"     "weight"    "set"       "data"
## [24,] "neural"    "data"      "system"    "weight"
## [25,] "input"     "system"    "neuron"    "set"
## [26,] "data"      "set"       "system"    "output"
## [27,] "algorithm" "method"    "training"  "function"
## [28,] "neural"    "unit"      "training"  "system"
## [29,] "input"     "neural"    "set"       "unit"
## [30,] "training"  "data"      "neural"    "function"
```

Note: I implemented 2 versions of topic EM (log_topic_EM does all EM steps on log space) and get similar results.

Problem 2

Image segmentation using EM You can segment an image using a clustering method - each segment is the cluster center to which a pixel belongs. In this exercise, you will represent an image pixel by its r, g, and b values (so use color images!). Use the EM algorithm applied to the mixture of normal distribution model lectured in class to cluster image pixels, then segment the image by mapping each pixel to the cluster center with the highest value of the posterior probability for that pixel. You must implement the EM algorithm yourself (rather than using a package). Test images are here, and you should display results for all three of them. Till then, use any color image you care to.

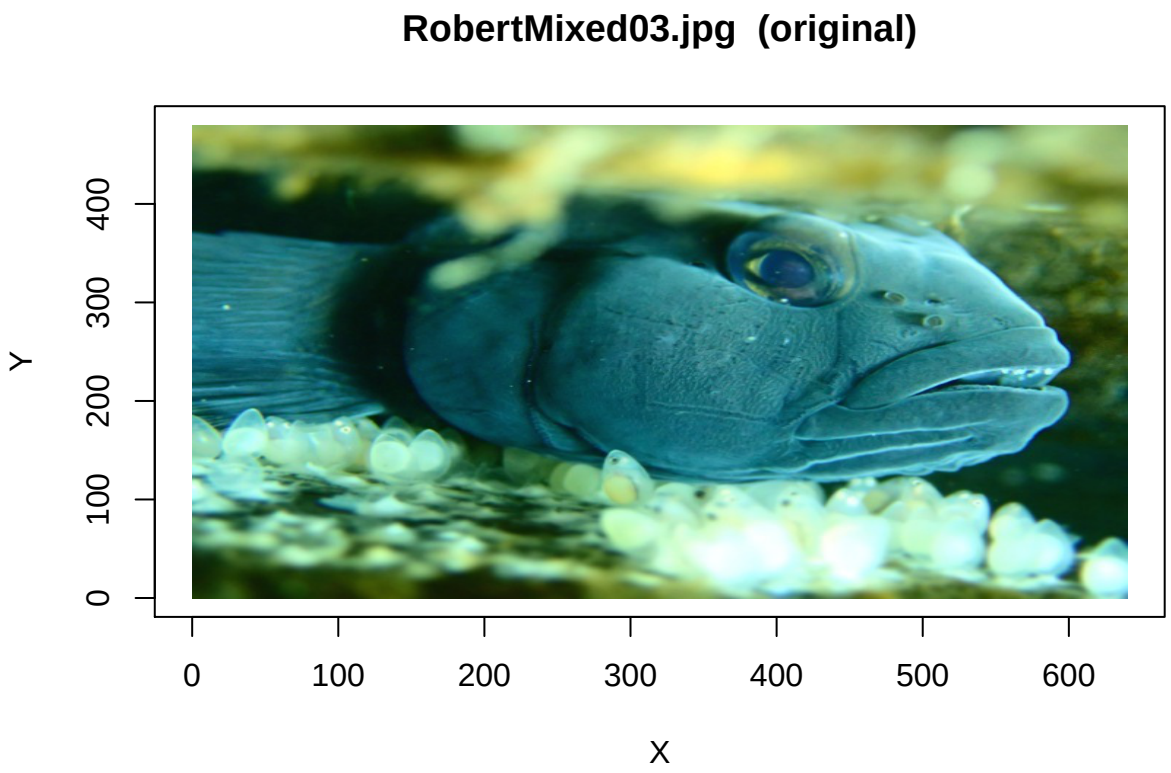
- Segment each of the test images to 10, 20, and 50 segments. You should display these segmented images as images, where each pixel's color is replaced with the mean color of the closest segment

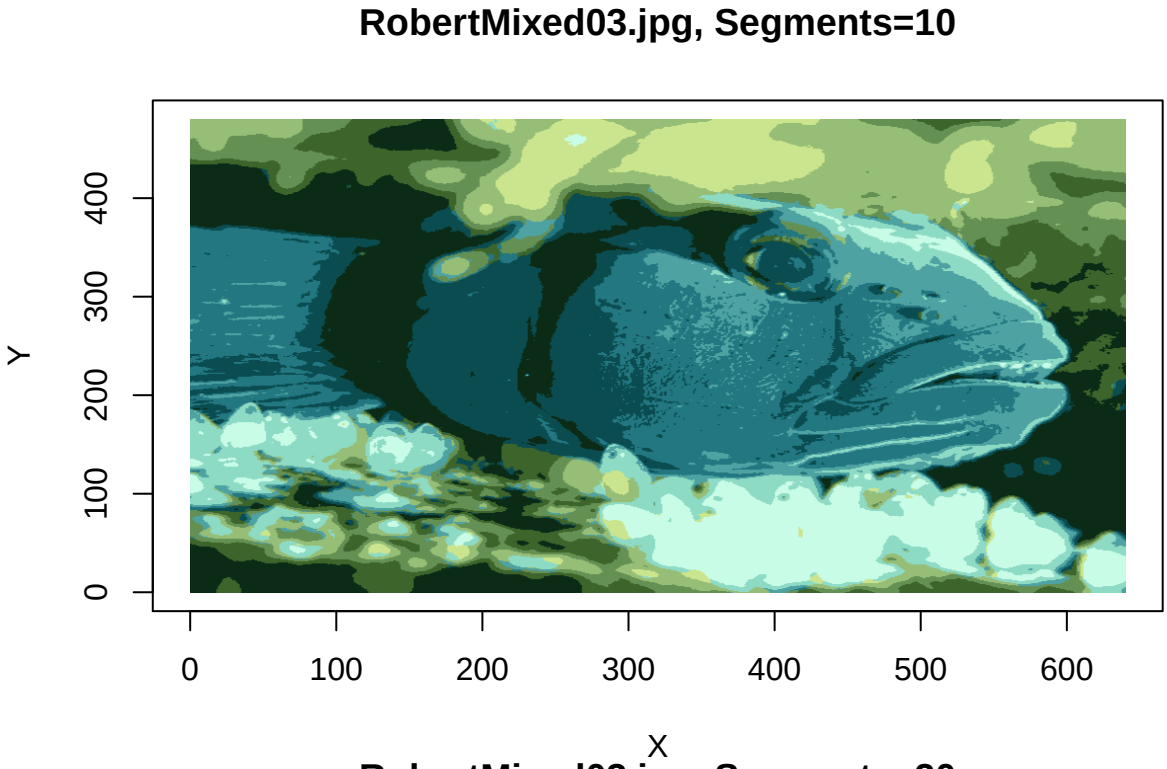
```
library(jpeg)

GMM_EM = function(X, cluster.count, converge.threshold = 1e-4) {
  N = dim(X)[1]
  W = matrix(0, N, cluster.count)
  #Take kmean as the initial value for U and Pi
  kmeans_cluster = kmeans(X, cluster.count)

  U = kmeans_cluster$centers
  pi = kmeans_cluster$size / N

  old_W = W
  log_W = W

  for (iter in 1:10000) {
```

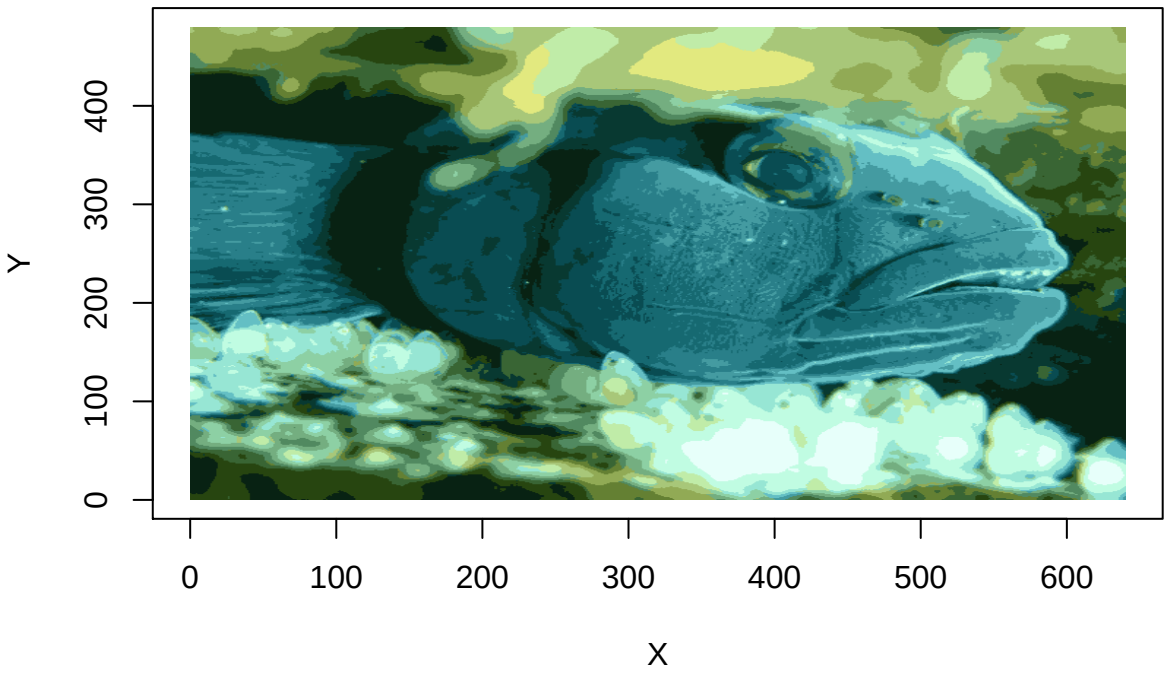
```
    log_pi = log(pi)
    #E Step
    for (j in 1:cluster.count) {
      D = t(X) - U[j,]
      log_W[,j] = (-0.5 * colSums(D * D)) + log_pi[j]
    }
    log_W = log_W - rowLogSumExps(log_W)

    W = exp(log_W)

    #M Step
    w_sum = colSums(W)
    U = crossprod(W, X) / w_sum
    pi = w_sum / N

    #Check convergence
    #I use mean and small convergence threshold for this model
    d = mean(abs(W - old_W))
    if(interactive()) print(paste("Iteration=", iter, ", D=", d, sep=""))
    if(d < converge.threshold) break()

    old_W = W
  }

  return (list(W = W, centers = U, pi=pi, iteration = iter))
}

segment_image = function(file.name, cluster.count) {
  img = readJPEG(file.name)
  old_dim = dim(img)
  dim(img) = c(dim(img)[1] * dim(img)[2], 3)

  r = GMM_EM(img * 255, cluster.count)
  r$centers = r$centers / 255

  for (i in 1:dim(r$W)[1]){
    img[i,] = r$centers[which.max(r$W[i,]),]
  }
  dim(img) = old_dim

  return (list(img = img, centers = r$centers, pi=r$pi))
}

plot_image = function(file.name) {
    img = readJPEG(file.name)
    plot(c(0, dim(img)[2]), c(0, dim(img)[1]), type="n",
         main=paste(name, " (original)"),
         xlab = "X", ylab = "Y")
    rasterImage(img, 0, 0, dim(img)[2], dim(img)[1])
}
```

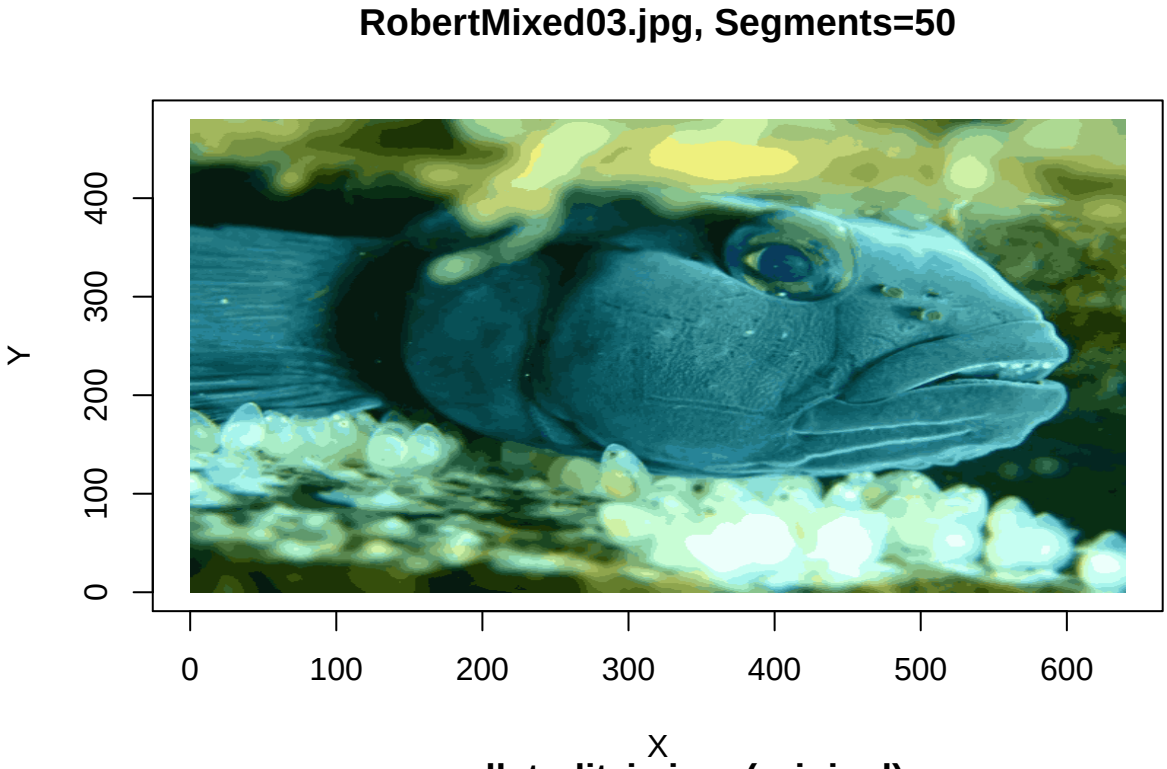
```
image_names = c("RobertMixed03.jpg", "smallstrelitzia.jpg", "smallsunset.jpg")
clusters = c(10, 20, 50)
```

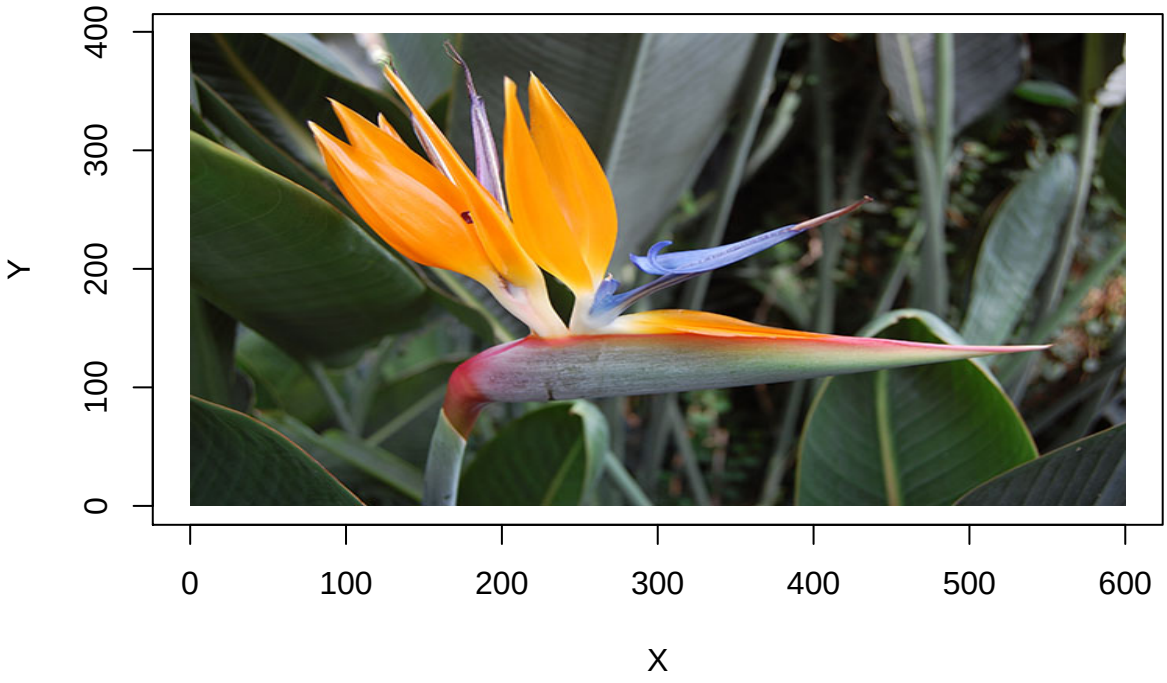
```
for (name in image_names) {
  plot_image(name)
  for (c in clusters) {
    r = segment_image(name, c)
    plot(c(0, dim(r$img)[2]), c(0, dim(r$img)[1]), type="n",
         main=paste(name, ", Segments=",c, sep=""),
         xlab = "X", ylab = "Y")
    rasterImage(r$img, 0, 0, dim(r$img)[2], dim(r$img)[1])
  }
}
```

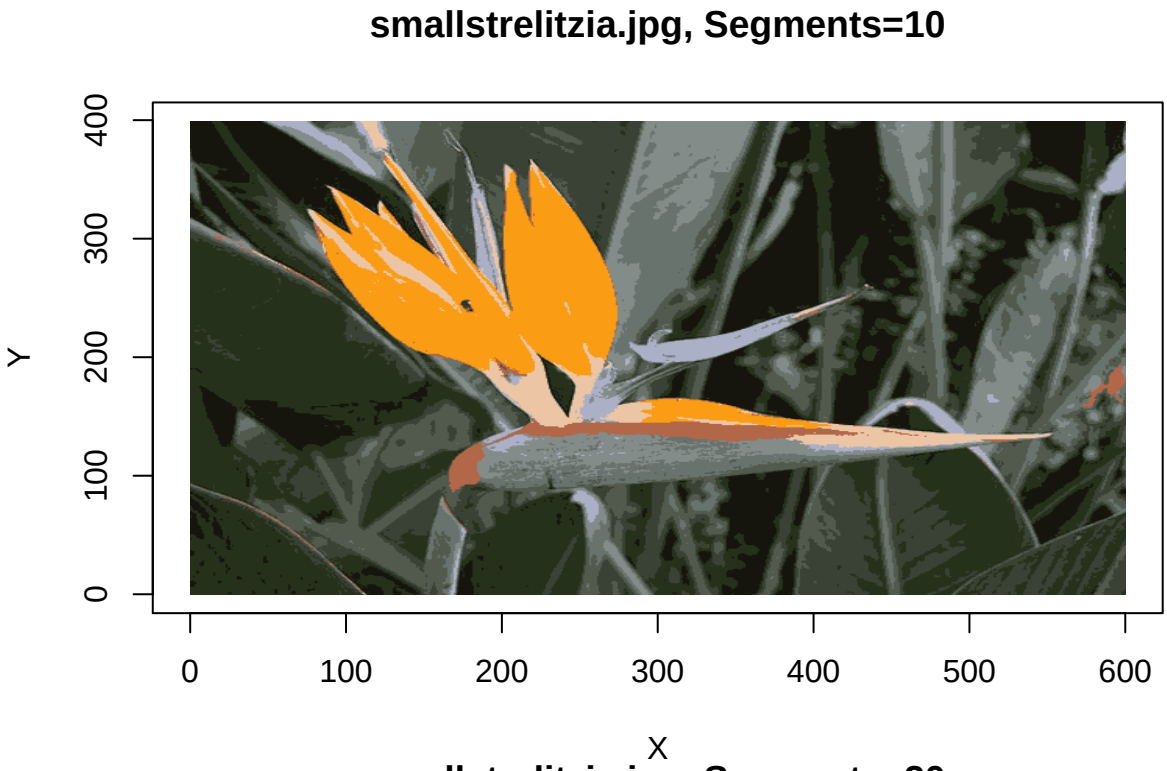


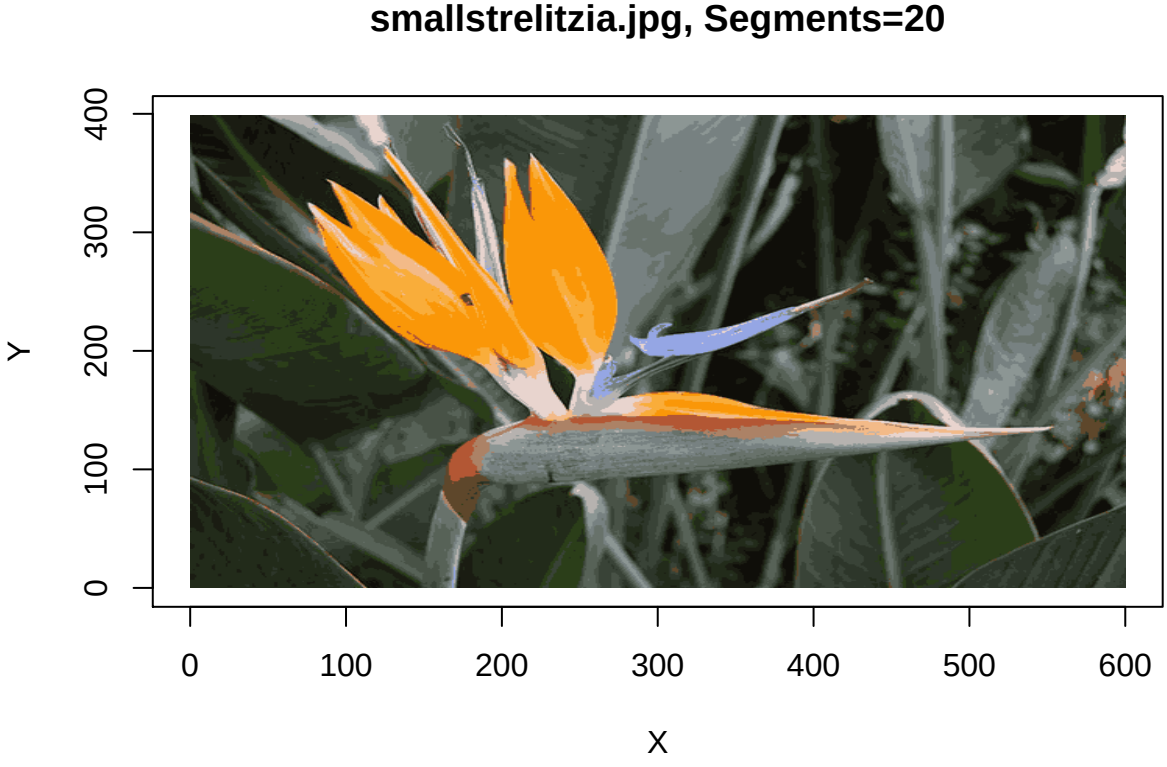

RobertMixed03.jpg, Segments=10
Y
0
100
200
300
400
0
100
200
300
400
500
600
X

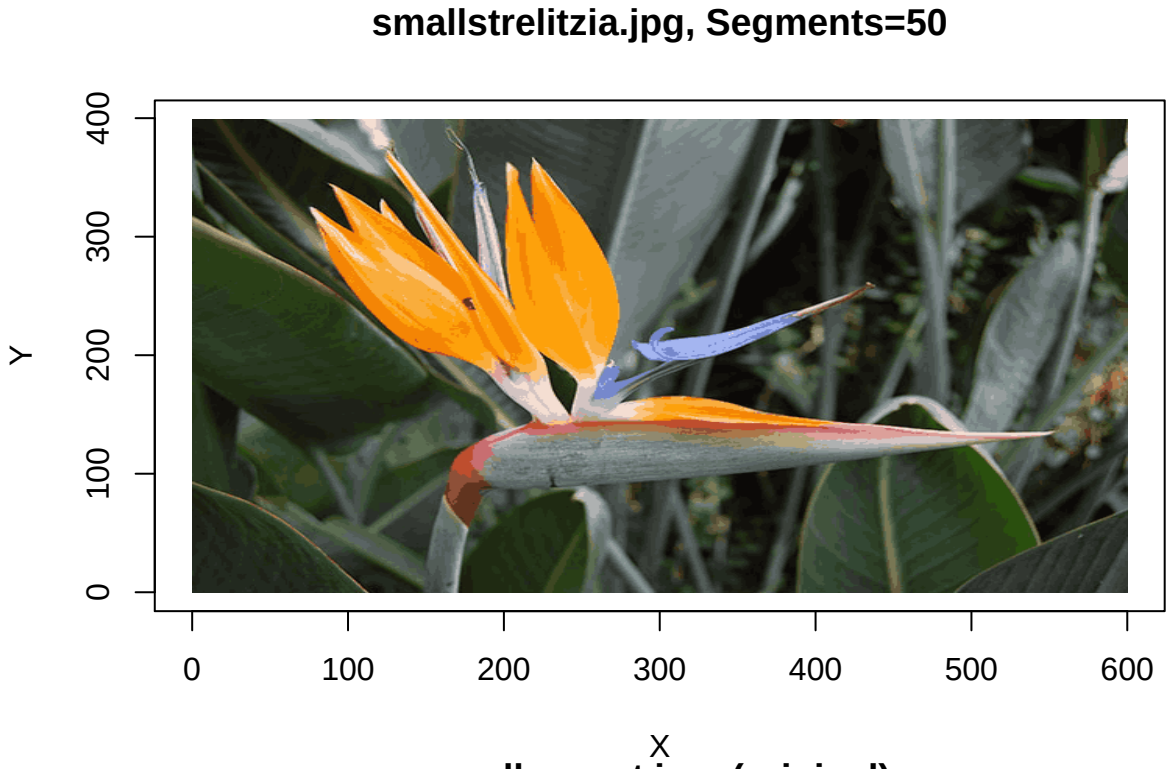

RobertMixed03.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X

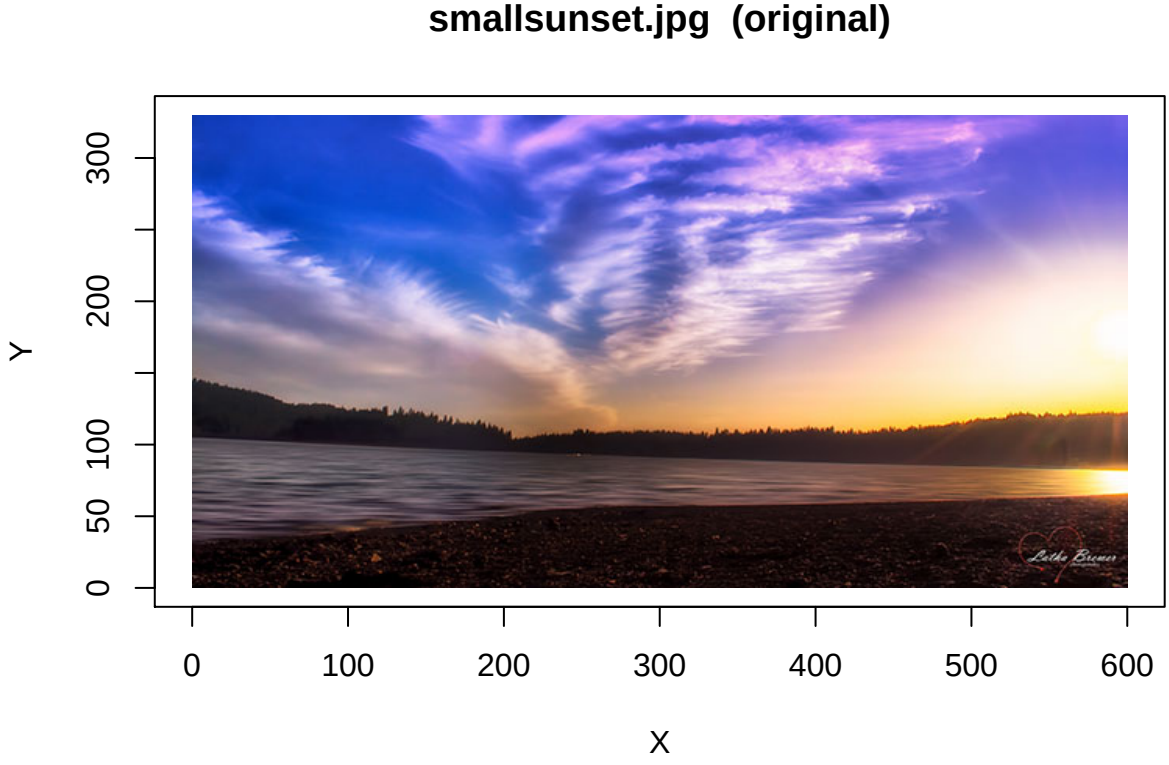

RobertMixed03.jpg, Segments=50
Y
0
100
200
300
400
0
100
200
300
400
500
600
X

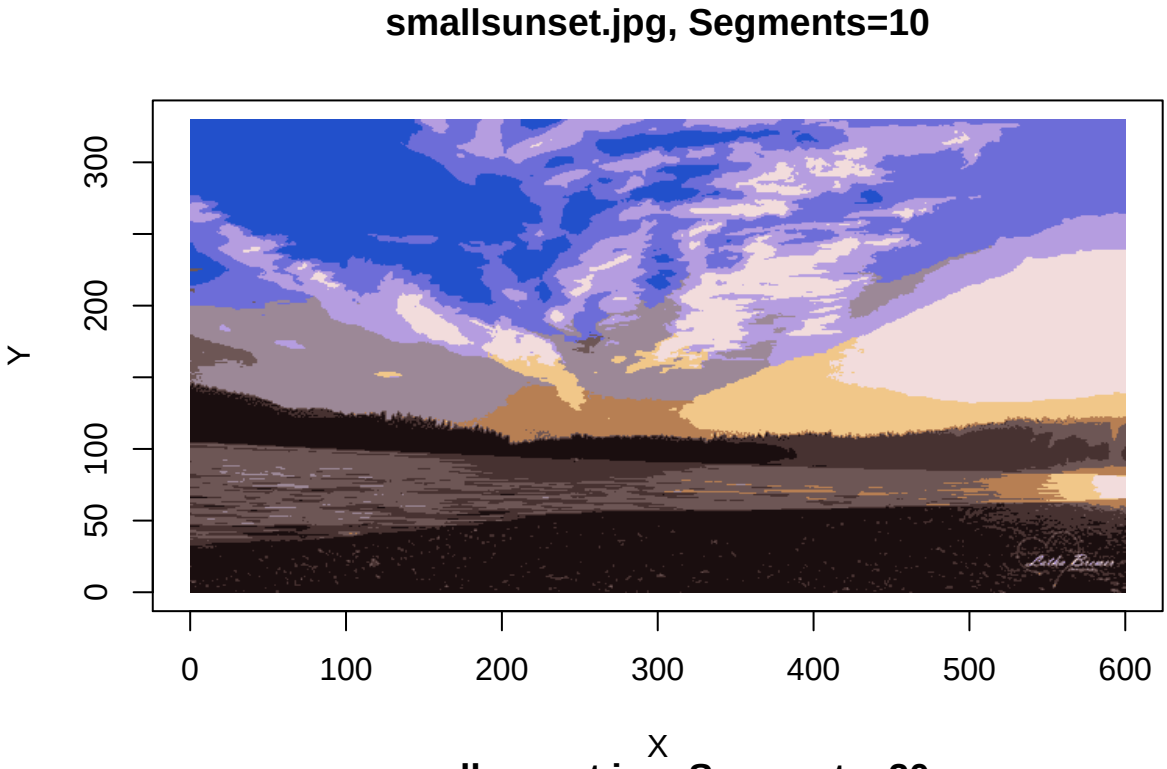

smallstrelitzia.jpg (original)
Y
0
100
200
300
400
0
100
200
300
400
500
600
X

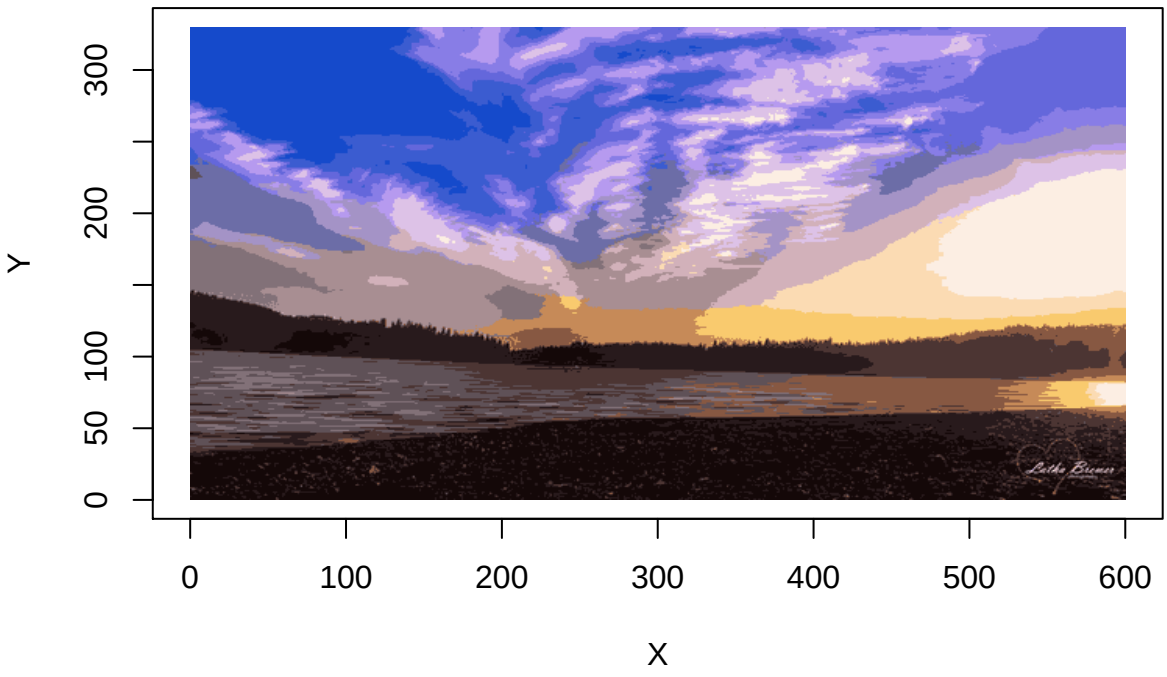

smallstrelitzia.jpg, Segments=10
Y
0
100
200
300
400
0
100
200
300
400
500
600
X

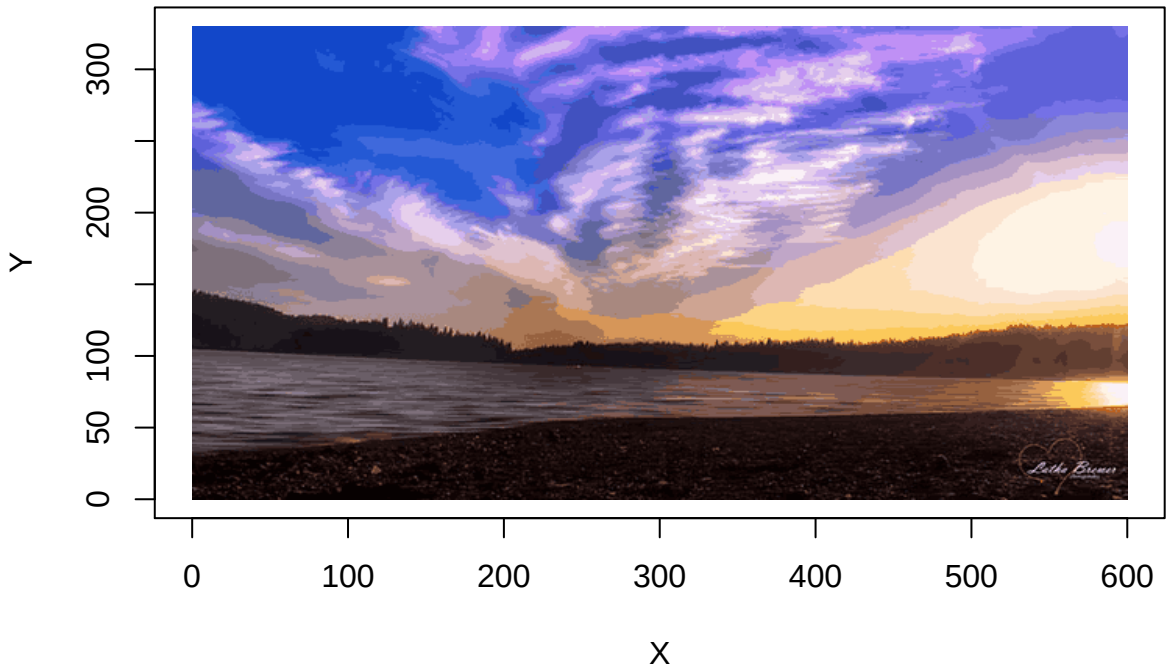

smallstrelitzia.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X


smallstrelitzia.jpg, Segments=50
Y
0
100
200
300
400
0
100
200
300
400
500
600
X


smallsunset.jpg (original)
Y
0
50
100
200
300
0
100
200
300
400
500
600
X

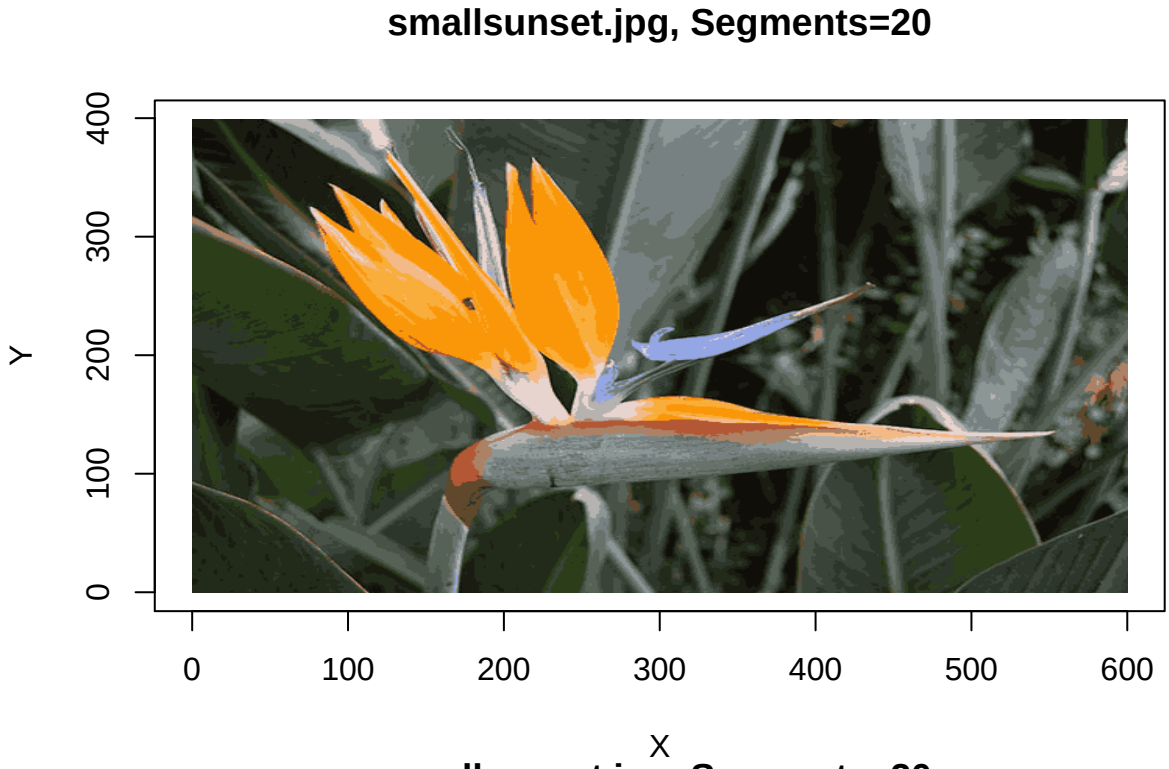

smallsunset.jpg, Segments=10
Y
0
50
100
200
300
0
100
200
300
400
500
600
X

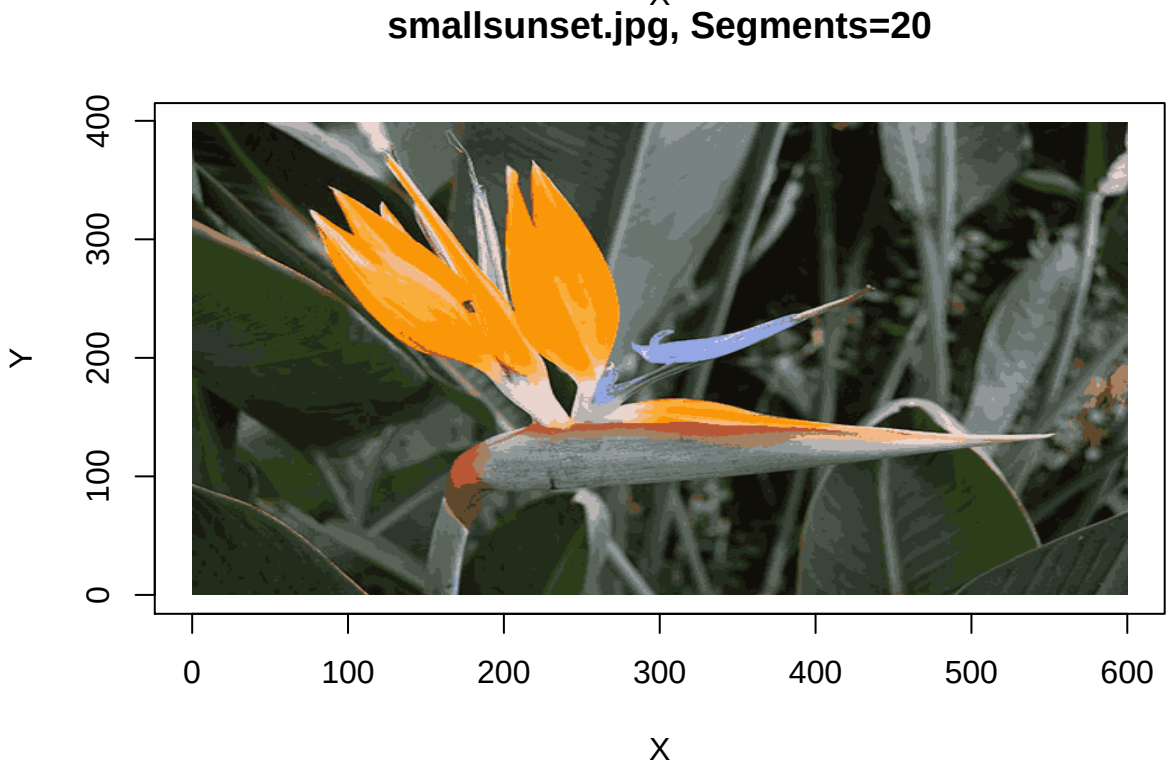

smallsunset.jpg, Segments=20
Y
0
50
100
200
300
0
100
200
300
400
500
600
X

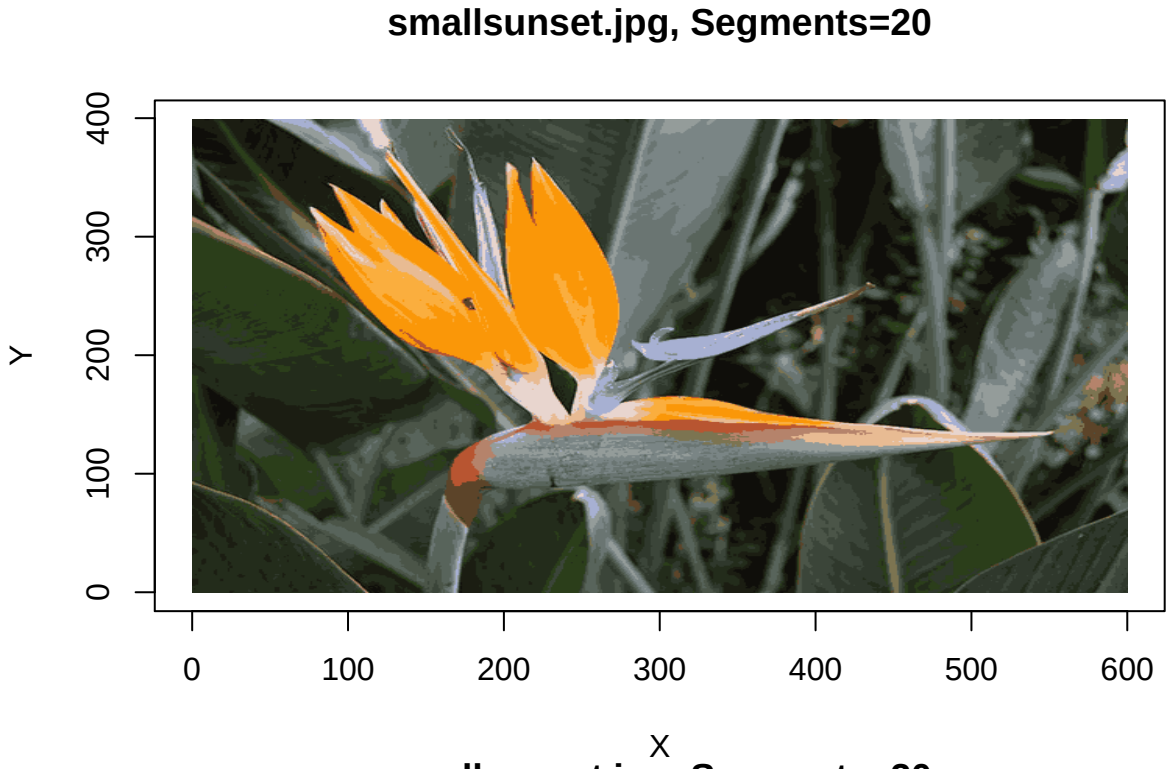


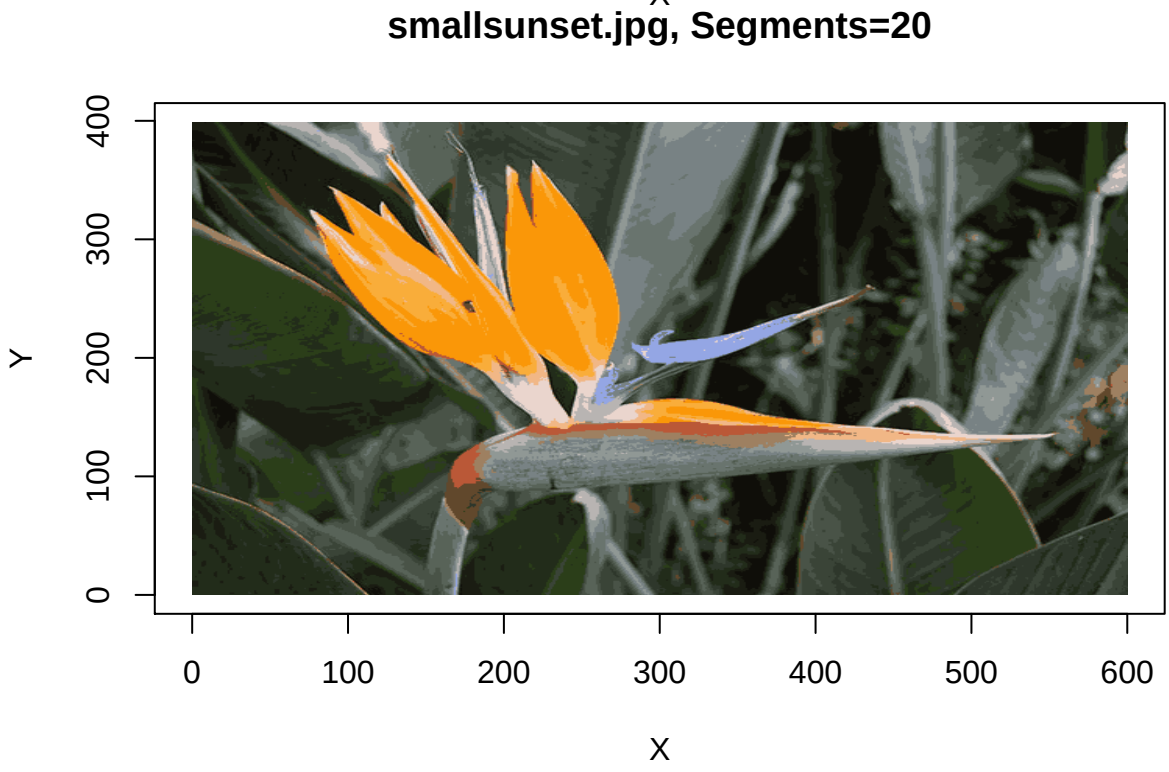
- We will identify one special test image. You should segment this to 20 segments using five different start points, and display the result for each case. Is there much variation in the result?

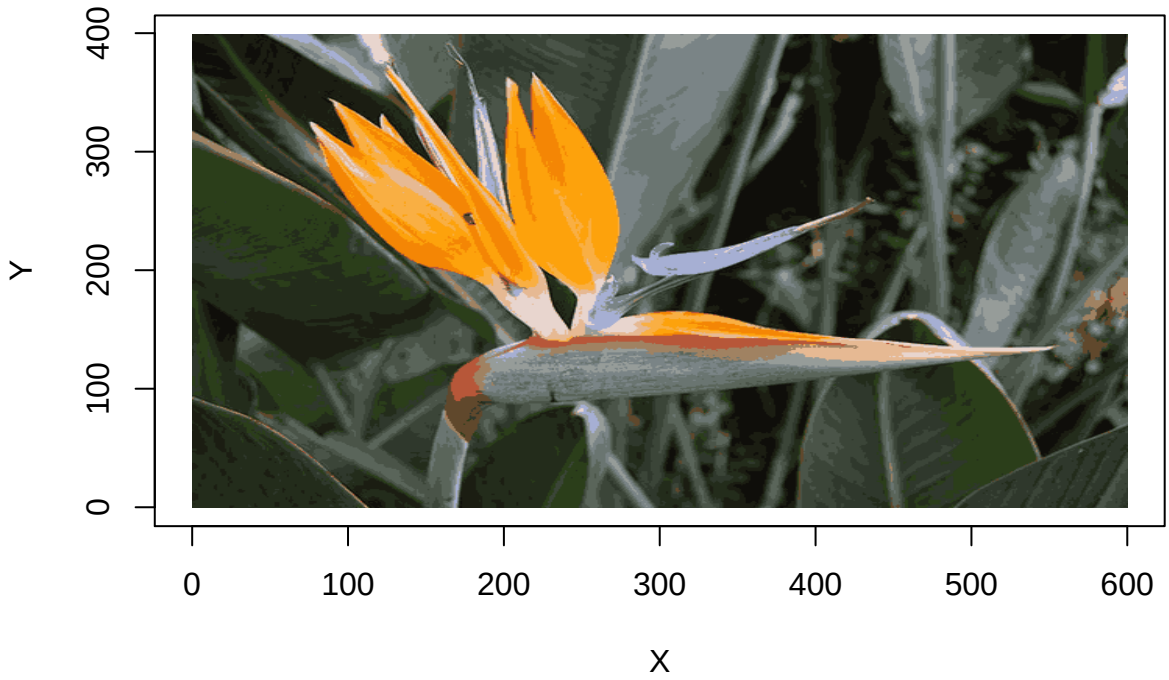
```
test_image = "smallstrelitzia.jpg"
cluster = 20

results = list()

for (i in 1:5) {
  #make different start points for k-means
  set.seed(i * 197208)
  r = segment_image(test_image, cluster)
  plot(c(0, dim(r$img)[2]), c(0, dim(r$img)[1]), type="n",
         main=paste(name, ", Segments=", cluster, sep=""),
         xlab = "X", ylab = "Y")
  rasterImage(r$img, 0, 0, dim(r$img)[2], dim(r$img)[1])
  results[[i]] = list(centers = r$centers, pi = r$pi)
}
```


smallsunset.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X


smallsunset.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X


smallsunset.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X


smallsunset.jpg, Segments=20
Y
0
100
200
300
400
0
100
200
300
400
500
600
X



From the output image, I do see some small variations (for example: shadow on leaf) but the differences are not as obvious as the ones produced by different segments. The output of new centers and π can tell the exact differences as illustrated below.

```
results[[1]]
```

```
## $centers
##             [,1]       [,2]       [,3]
##  [1,] 0.97916576 0.59033135 0.03028964
##  [2,] 0.97791719 0.67882375 0.22612119
##  [3,] 0.16972380 0.24281113 0.09943070
##  [4,] 0.95359392 0.73121381 0.53232859
##  [5,] 0.22550176 0.26300382 0.21436583
##  [6,] 0.90866245 0.83149883 0.80620454
##  [7,] 0.05801294 0.05332337 0.03716187
##  [8,] 0.28139316 0.32236806 0.27513329
##  [9,] 0.40757694 0.44754785 0.41804425
## [10,] 0.71362073 0.69662430 0.68474693
## [11,] 0.55076126 0.58911544 0.58203080
## [12,] 0.70380689 0.34005671 0.20518212
## [13,] 0.46507298 0.50502888 0.50654290
## [14,] 0.10270030 0.11035031 0.07084855
## [15,] 0.17766924 0.21299097 0.16633603
## [16,] 0.13328836 0.17048972 0.10123918
## [17,] 0.67302898 0.51091846 0.39874140
## [18,] 0.34152723 0.38490920 0.33983022
## [19,] 0.58315901 0.65173130 0.89504569
## [20,] 0.37254471 0.28234654 0.17283859
##
## $pi
```

```
##  [1] 0.058098538 0.020842793 0.075016121 0.011834046 0.092663075
##  [6] 0.013419172 0.065737781 0.077736889 0.058027349 0.016107241
## [11] 0.032181147 0.008898994 0.063063968 0.086228636 0.089432651
## [16] 0.124075429 0.013170593 0.073098311 0.007271530 0.013095735
```

```
results[[5]]
```

```
## $centers
##             [,1]       [,2]       [,3]
##  [1,] 0.90272640 0.72528345 0.57798833
##  [2,] 0.71779478 0.34276363 0.22488813
##  [3,] 0.10476023 0.11412591 0.07223308
##  [4,] 0.47669241 0.51939306 0.52567441
##  [5,] 0.91086606 0.83430951 0.80702723
##  [6,] 0.62566430 0.50856332 0.38592583
##  [7,] 0.97580088 0.68645788 0.26352831
##  [8,] 0.99141679 0.63386262 0.04815002
##  [9,] 0.18351454 0.21905071 0.17296262
## [10,] 0.16951399 0.24180368 0.10074617
## [11,] 0.95622913 0.52405860 0.01810259
## [12,] 0.29011117 0.33128386 0.28327553
## [13,] 0.65192794 0.68721867 0.82555013
## [14,] 0.13529587 0.17245953 0.10443088
## [15,] 0.58736907 0.61037300 0.60102880
## [16,] 0.35194235 0.39464522 0.35158070
## [17,] 0.23240469 0.27026747 0.22139837
## [18,] 0.37805429 0.28314006 0.17338845
## [19,] 0.05931597 0.05490302 0.03824535
## [20,] 0.41729662 0.45979099 0.44125109
##
## $pi
##  [1] 0.014903504 0.008870284 0.088750329 0.057251470 0.012727746
##  [6] 0.014346567 0.018632510 0.039860623 0.093317529 0.078045563
## [11] 0.022696074 0.078372452 0.014062398 0.124212459 0.028720611
## [16] 0.073839808 0.090199917 0.012846920 0.069619319 0.058723917
```