Programming Assignment 3

Part A:

1. Misclassified 71 emails which is 27 percent.
2. Train on different numbers of training examples:
    a. 50 – misclassified 132 which is 50 percent.
    b. 100 – misclassified 125 which is 48 percent.
    c. 400 – misclassified 117 which is 45 percent.
3. Code

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 11 20:21:17 2017

@author: rditljtd
"""

import numpy as np
from scipy import misc
from scipy import sparse as sps
import matplotlib.pyplot as plt
from math import *

#find probability of email being spam given that it contains a particular word
#this equals the probability of seeing this word given that it is a spam * proba
bility of it being seen overall / probability of it being spam

#find probability of email being non-
spam given that it contains a particular word
#this equls the probabiltiy of seeing this word givent that the email is non-
spam *
    #probability of seeing this word overall
    #/probability of email being non-spam

# Load the labels for the training set
train_labels = np.loadtxt('train-labels-50.txt',dtype=int)

# Get the number of training examples from the number of labels
numTrainDocs = train_labels.shape[0]

# This is how many words we have in our dictionary
numTokens = 2500

# Load the training set feature information
M = np.loadtxt('train-features-50.txt',dtype=int)

# Create matrix of training data
train_matrix = sps.csr_matrix((M[:,2], (M[:,0], M[:,1])),shape=(numTrainDocs,num
Tokens))

#train_labels[i] = ith document label: spam or not spam

#tran_matrix[i:] = ith document

#train_matrix[i:, j] = jth word in ith document

```

```
#print train_matrix[69]

print train_matrix.shape[1]

spam_prob = (sum(x == 1 for x in train_labels))/(numTrainDocs*(1.0))

nonspam_prob = (sum(x == 0 for x in train_labels))/(numTrainDocs*(1.0))

print spam_prob, nonspam_prob

#create variable for sum of all non spam emails
num_nonspam = (sum(x==0 for x in train_labels))
#create variable for sum of all spam emails
num_spam = (sum(x==1 for x in train_labels))

#create array for probability all words in spam and non spam [spam_prob, nonspam_prob]
prob_for_each_word = []


#add up the instances of this word [spam, nonspam]
sum_for_this_word = [0, 0]

#create array for how many times each word was foud [[spam, nonspam]] i = word
found_for_all_emails = []

#loop through each email
for j in range (0, train_matrix.shape[0]):

    #array of words in email [instances]
    words_in_email = train_matrix[j].toarray()[0]
    #print train_matrix[j],



    #loop through each word in dictionary and see if it is in email
    for k in range (0, len(words_in_email)):

        if j == 0:
            #create array for if this word was found [spam, nonspam]
            found_for_all_emails.append([0, 0])

        #if email is non spam and there is more than zero instances of this word

        if (train_labels[j] == 0 and words_in_email[k] > 0):

            #increment the sum for this word for non spam emails
            found_for_all_emails[k][1] += 1

        #else if the email is spam and there is more than zero instances of this word
        elif (train_labels[j] == 1 and words_in_email[k] > 0):

            #print words_in_email[k]
            #increment the sum for this word for spam emails
            found_for_all_emails[k][0] += 1

       #print found_for_all_emails
       #sum_for_this_word = sum(found_for_all_emails[:1])
       #add the probability for this word to array

```

```
105.        #create an array to hold the probability for each word [spam, nonspam] i
     = word
106.        prob_for_each_word = []
107.
108.        #create array to hold the overall probability for each word [num] i = wo
    rd
109.        overall_prob_for_each_word = []
110.
111.        for i in range (0, numTokens):
112.
113.            prob_for_each_word.append([0,0])
114.            overall_prob_for_each_word.append(0)
115.
116.            #divide by the total number of spam emails
117.            prob_for_each_word[i][0] = (found_for_all_emails[i][0]/(num_spam*(1.
    0)))
118.
119.            #divide by the total number of nonspam emails
120.            prob_for_each_word[i][1] = (found_for_all_emails[i][1]/(num_nonspam*
    (1.0)))
121.
122.            overall_prob_for_each_word[i] = (found_for_all_emails[i][0] + found_
    for_all_emails[i][1])/((num_nonspam + num_spam)*(1.0))
123.
124.
125.        #print prob_for_each_word
126.        #print overall_prob_for_each_word
127.
128.
129.
130.        # Load the labels for the training set
131.        test_labels = np.loadtxt('test-labels.txt',dtype=int)
132.
133.        # Get the number of training examples from the number of labels
134.        numTestDocs = test_labels.shape[0]
135.
136.        # Load the training set feature information
137.        N = np.loadtxt('test-features.txt',dtype=int)
138.
139.        # Create matrix of training data
140.        test_matrix = sps.csr_matrix((N[:,2], (N[:,0], N[:,1])),shape=(numTestDo
    cs,numTokens))
141.
142.        prob_for_all_test_emails = []
143.
144.        #iterate through each email in the test set
145.        for i in range (0, (test_matrix.shape[0])):
146.
147.            #array of words in email [instances]
148.            words_in_test_email = test_matrix[i].toarray()[0]
149.            #print train_matrix[j],
150.
151.            prob_spam_given_all_words = 0
152.            prob_nonspam_given_all_words = 0
153.
154.            #loop through each word in dictionary and see if it is in email
155.            for k in range (0, len(words_in_test_email)):
156.
157.                prob_spam_given_word = 0
158.                prob_nonspam_given_word = 0
159.
```

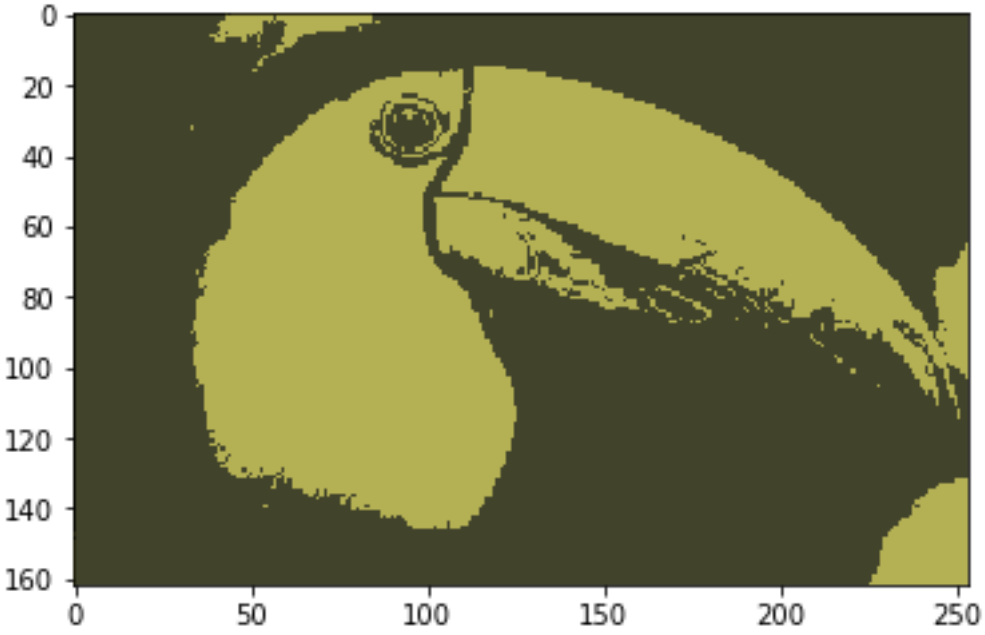
```
160.                #if email is non spam and there is more than zero instances of t
    his word
161.                if (words_in_test_email[k] > 0):
162.                    #prob_nonspam_given_word = float(((prob_for_each_word[k][1])
    *(1.0))*overall_prob_for_each_word[k])/((nonspam_prob)*1.0)
163.                    #prob_spam_given_word = float(((prob_for_each_word[k][0])*(1
    .0))*overall_prob_for_each_word[k])/((spam_prob)*1.0)
164.                    if (prob_for_each_word[k][1] > 0):
165.                        prob_nonspam_given_word = float(math.exp(math.log(prob_f
    or_each_word[k][1]))) + float(math.exp(math.log(overall_prob_for_each_word[k])))
     - float(math.exp(math.log(nonspam_prob)))
166.                    if (prob_for_each_word[k][0] > 0):
167.                        prob_spam_given_word = float(math.exp(math.log(prob_for_
    each_word[k][0]))) + float(math.exp(math.log(overall_prob_for_each_word[k]))) -
     float(math.exp(math.log(spam_prob)))
168.
169.                if (prob_spam_given_word > 0):
170.                    #print prob_spam_given_all_words,
171.                    #prob_spam_given_all_words += float((prob_spam_given_word*(1
    .0)))
172.                    #print prob_spam_given_all_words
173.                    prob_spam_given_all_words += prob_spam_given_word
174.
175.                if (prob_nonspam_given_word > 0):
176.                    #print prob_nonspam_given_all_words,
177.                    #prob_nonspam_given_all_words += float((prob_nonspam_given_w
    ord*(1.0)))
178.                    prob_nonspam_given_all_words += prob_nonspam_given_word
179.
180.            #prob_spam_given_all_words += float(math.exp(math.log(spam_prob)))
181.            #prob_nonspam_given_all_words += float(math.exp(math.log(nonspam_pro
    b)))
182.            prob_for_all_test_emails.append([prob_spam_given_all_words, prob_non
    spam_given_all_words])
183.
184.        print prob_for_all_test_emails
185.
186.        classify_each_test_email = []
187.        spam_count = 0
188.        nonspam_count = 0
189.        incorrect = []
190.        print len(prob_for_all_test_emails)
191.        for k in range(0, len(prob_for_all_test_emails)):
192.            if prob_for_all_test_emails[k][0] > prob_for_all_test_emails[k][1]:

193.                classify_each_test_email.append(1)
194.                spam_count += 1
195.                if(test_labels[k] != classify_each_test_email[k]):
196.                    incorrect.append([k, classify_each_test_email[k], test_label
    s[k]])
197.            elif prob_for_all_test_emails[k][1] > prob_for_all_test_emails[k][0]
    :
198.                classify_each_test_email.append(0)
199.                nonspam_count += 1
200.                if(test_labels[k] != classify_each_test_email[k]):
201.                    incorrect.append([k, classify_each_test_email[k], test_label
    s[k]])
202.            else:
203.                classify_each_test_email.append('unknown')
204.                incorrect.append([k, classify_each_test_email[k], test_labels[k]
    ])
```

```
205.
206.        print classify_each_test_email
207.        print "spam count: " + str(spam_count) + " spam percentage: " + str(floa
    t((spam_count*100)/(len(classify_each_test_email))))
208.        print "nonspam count: " + str(nonspam_count) + " nonspam percentage: " +
     str(float((nonspam_count*100)/(len(classify_each_test_email))))
209.        print "number misclassified: " + str(len(incorrect)) + " percentage misc
    lassified: " + str(float(((len(incorrect))*100)/(len(classify_each_test_email)))
    )
210.
211.
212.
213.
214.
215.        #def nBayes(train_matrix, train_labels):
216.        #    print 'called nBayes'
217.        #    #reset num_unique_words_in_spam ratio and wordsInSpam
218.        #
219.        #    #create variable for number of unique words in spam emails
220.        #    num_unique_words_in_spam = 0
221.        #
222.        #    #create variable for number of unique words in nonspam emails
223.        #    num_unique_words_in_nonspam = 0
224.        #
225.        #    #total number of words in spam emails
226.        #    totalWordsInSpam = 0
227.        #    totalWordsInNonSpam = 0
228.        #    wordsInSpam = {}
229.        #    wordsInNonSpam = {}
230.        #
231.        #    #loop through each email
232.        #    for i in range (0, train_matrix.shape[0]):
233.        #        #get array with words and # of instance of these words
234.        #        #two dim array with first being word id and second being # of i
    nstances
235.        #        wordsInEmail = train_matrix[i].toarray()[0]
236.        #        #print wordsInEmail
237.        #        #print "-------------"
238.        #
239.        #        #for each possible word in the email
240.        #        for j in range(0, len(wordsInEmail)):
241.        #
242.        #            if (wordsInEmail[j] == 0):
243.        #                continue
244.        #            #if the email is a spam email
245.        #            if train_labels[i] == 1:
246.        #
247.        #                #increment summation value for a unique word in this sp
    am email
248.        #                num_unique_words_in_spam = num_unique_words_in_spam+1
249.        #                #add the # of instances of this word to the number of w
    ords total in the email
250.        #                totalWordsInSpam = totalWordsInSpam + wordsInEmail[j]
251.        #
252.        #                #if this word has already been seen in a spam email
253.        #                if (j in wordsInSpam.keys()):
254.        #                    #add the # of instances of this word to the # of in
    stances of this word in all spam emails
255.        #                    wordsInSpam[j] += wordsInEmail[j]
256.        #
257.        #                #word has not been seen in a spam email yet
```

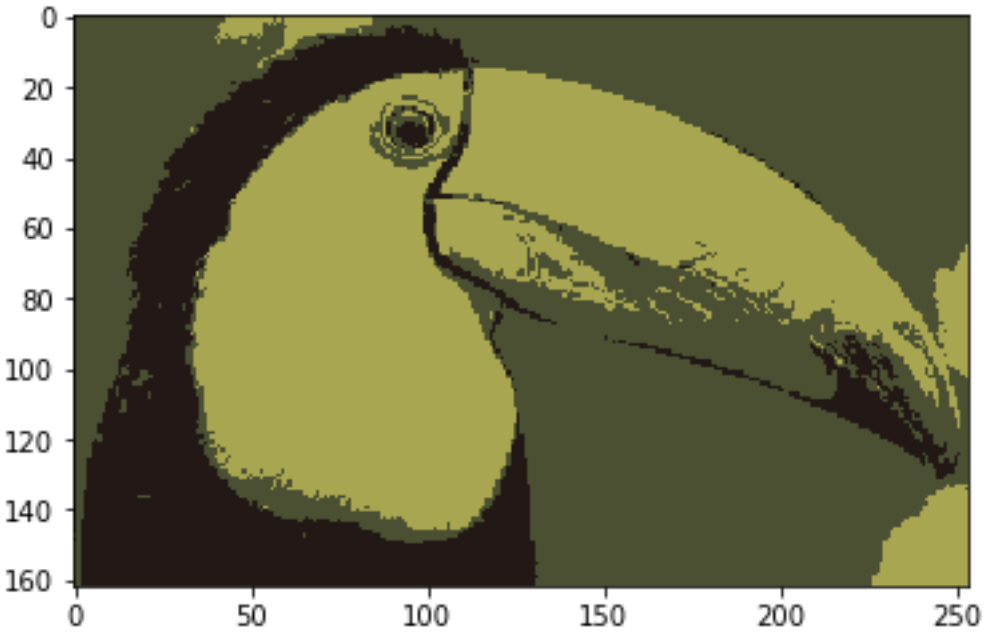
```
258.       #                else:
259.       #                    #set the # of instances of this word in a spam emai
    l to the # of instances in this email
260.       #                    wordsInSpam[j] = wordsInEmail[j]
261.       #
262.       #            #if email is non-spam
263.       #            if train_labels[i] == 0:
264.       #                #increment summation value for a unique word in this no
    n spam email
265.       #                num_unique_words_in_nonspam = num_unique_words_in_nonsp
    am+1
266.       #                #add the # of instances of this word to the number of w
    ords total in the email
267.       #                totalWordsInNonSpam = totalWordsInNonSpam + wordsInEmai
    l[j]
268.       #
269.       #                #if this word has already been seen in a non spam email

270.       #                if (j in wordsInNonSpam.keys()):
271.       #                    #add the # of instances of this word to the # of in
    stances of this word in all non spam emails
272.       #                    wordsInNonSpam[j] += wordsInEmail[j]
273.       #
274.       #                #word has not been seen in a non spam email
275.       #                else:
276.       #                    #set the # of instances of this word in non spam em
    ails to the # of instances in this email
277.       #                    wordsInNonSpam[j] = wordsInEmail[j]
278.       #
279.       #    #I think what actually needs to be done here is that for each uniqu
    e word in the dictionary, calculate the percentage of spam emails that contain t
    hat word
280.       #    #then calculate the # of non spam emails that contain that word. Mu
    ltiply the percentages for every word in
281.       #
282.       #
283.       #    #dictionary of words and # of instances in spam emails

284.       #    print wordsInSpam
285.       #    #dictionary of words and # of instances in non spam emails
286.       #    print wordsInNonSpam
287.       #    #of unique words in spam email
288.       #    print num_unique_words_in_spam
289.       #    #of unique words in non spam email
290.       #    print num_unique_words_in_nonspam
291.       #    #total # of words in spam emails
292.       #    print totalWordsInSpam
293.       #    #total # of words in non spam emails
294.       #    print totalWordsInNonSpam
295.       #
296.       #    #return # of unique words in spam + 1 / total # of words in spam em
    ails + # of words in dictionary,
297.       #    #return # of unique words in non spam +1 / total # of words in non
    spam emails + # of words in dictionary
298.       #    return (num_unique_words_in_spam+1)/((totalWordsInSpam+train_matrix
    .shape[1])*1.0), (num_unique_words_in_nonspam+1)/((totalWordsInNonSpam+train_mat
    rix.shape[1])*1.0)
299.       #
300.       #print nBayes(train_matrix, train_labels)
```

Part B:

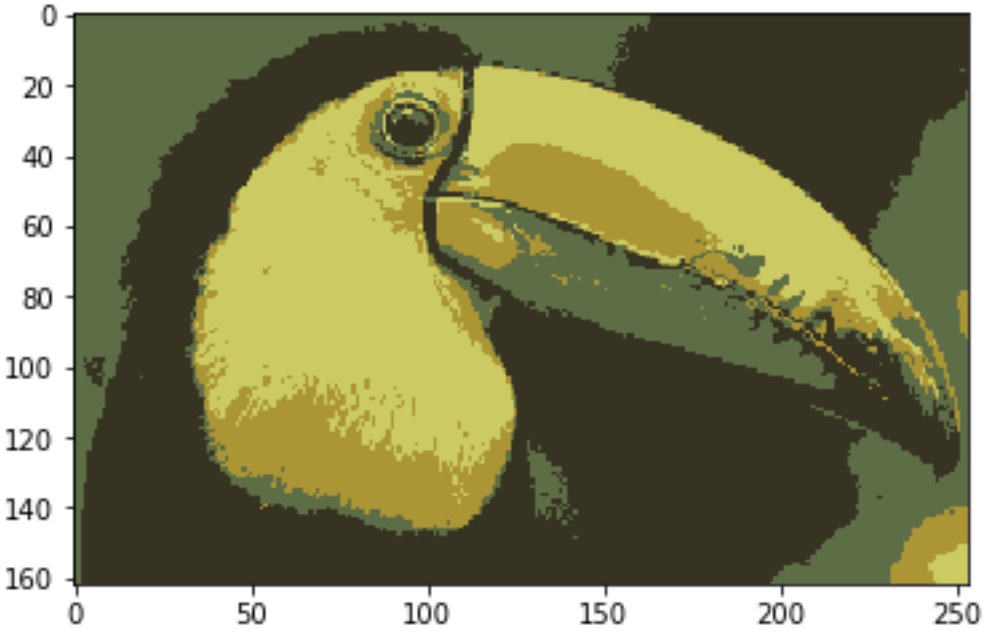
1. Images for k = 2..15
    a. 2



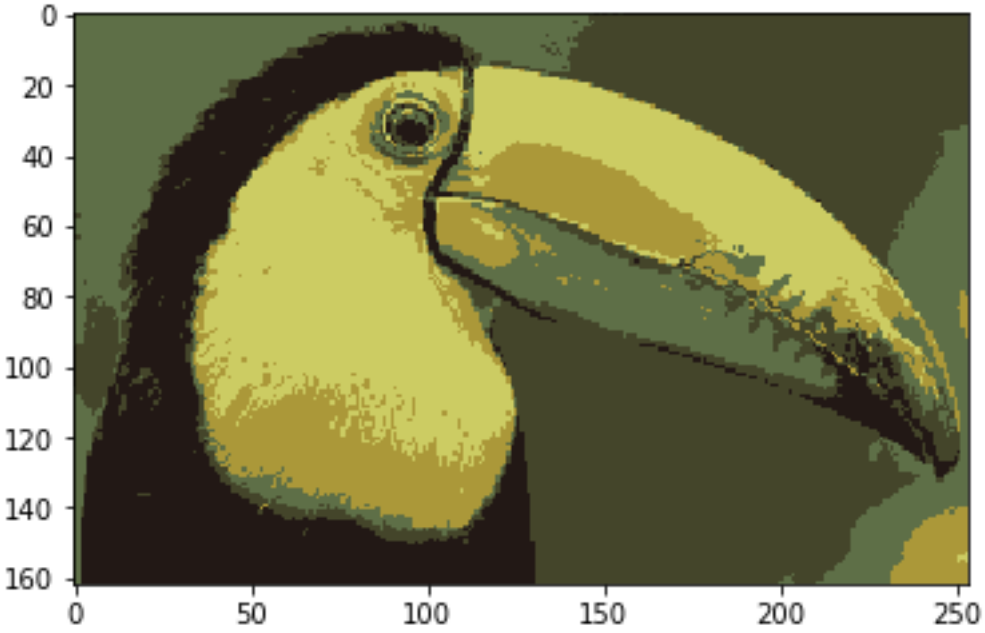
    b. 3



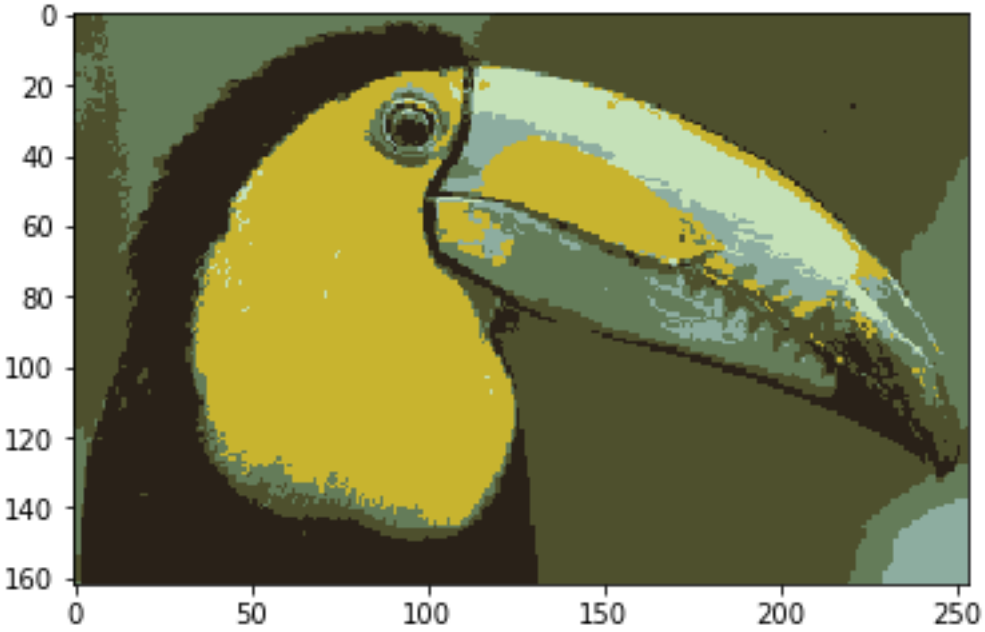
    c. 4


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

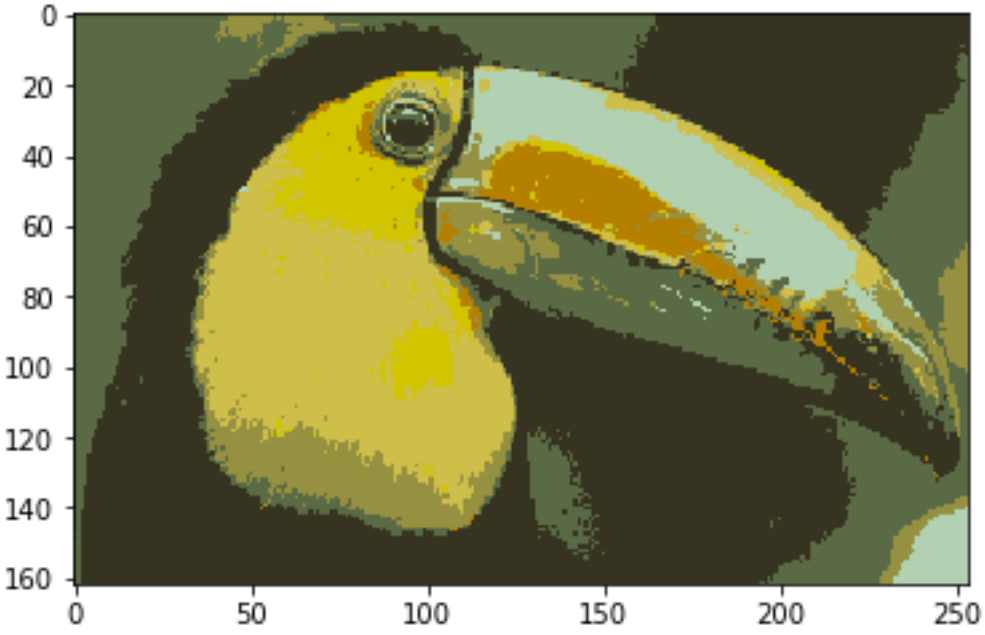
d. 5


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

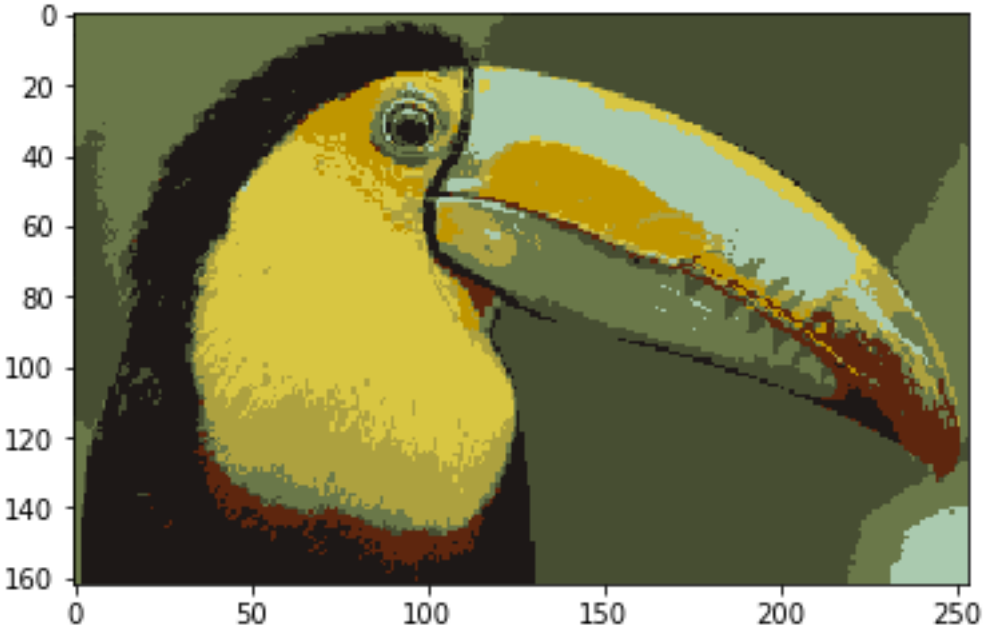
e. 6


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

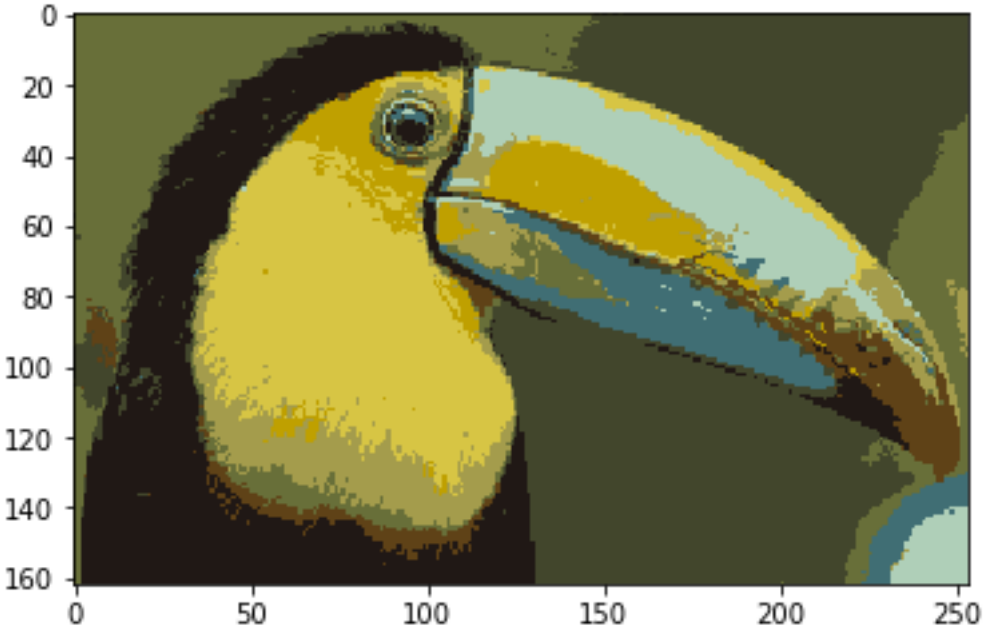
f. 7


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

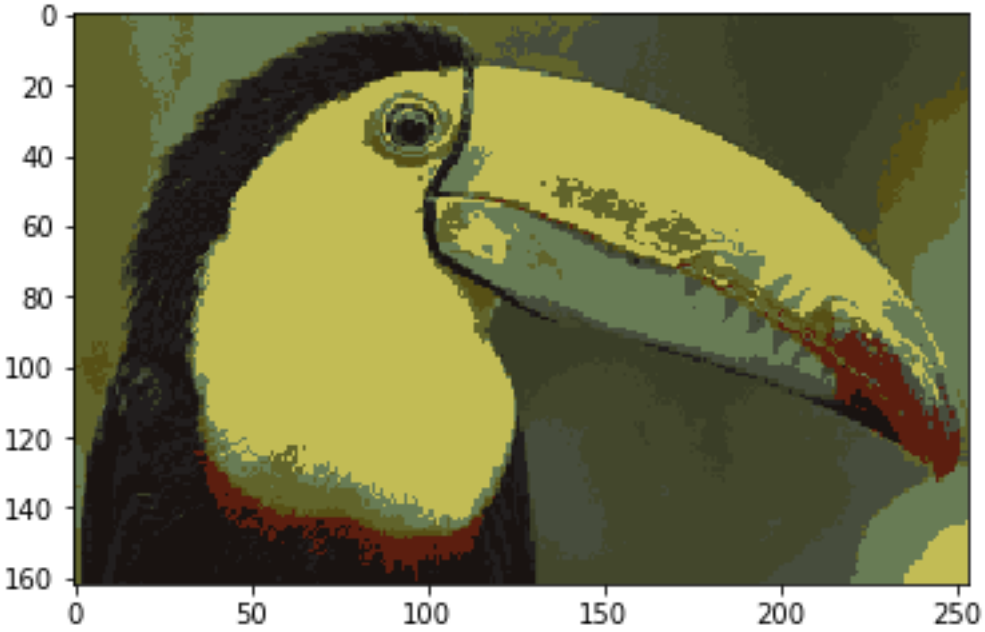
g. 8


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

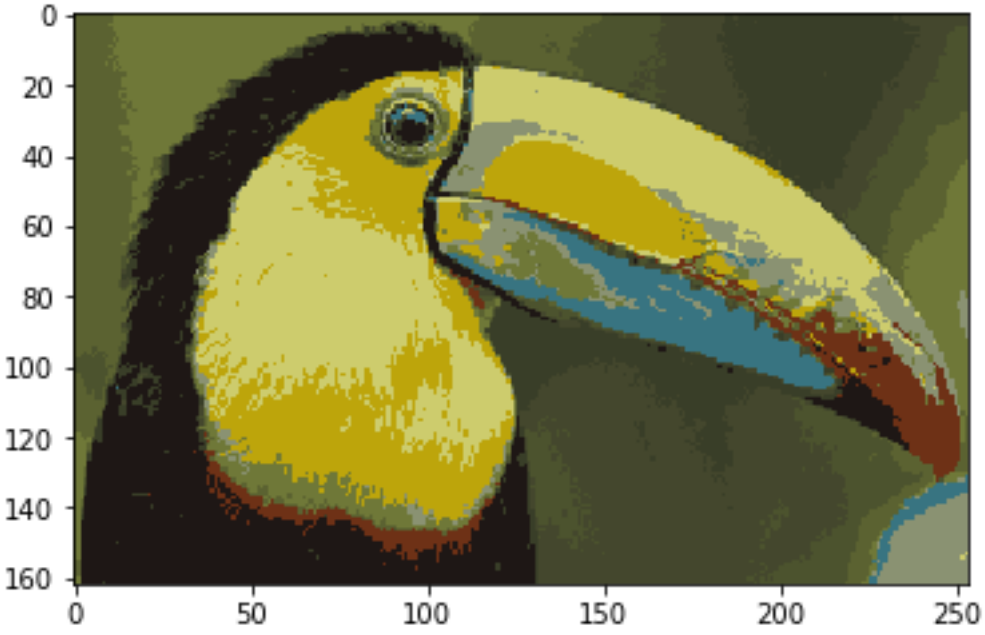
h. 9


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

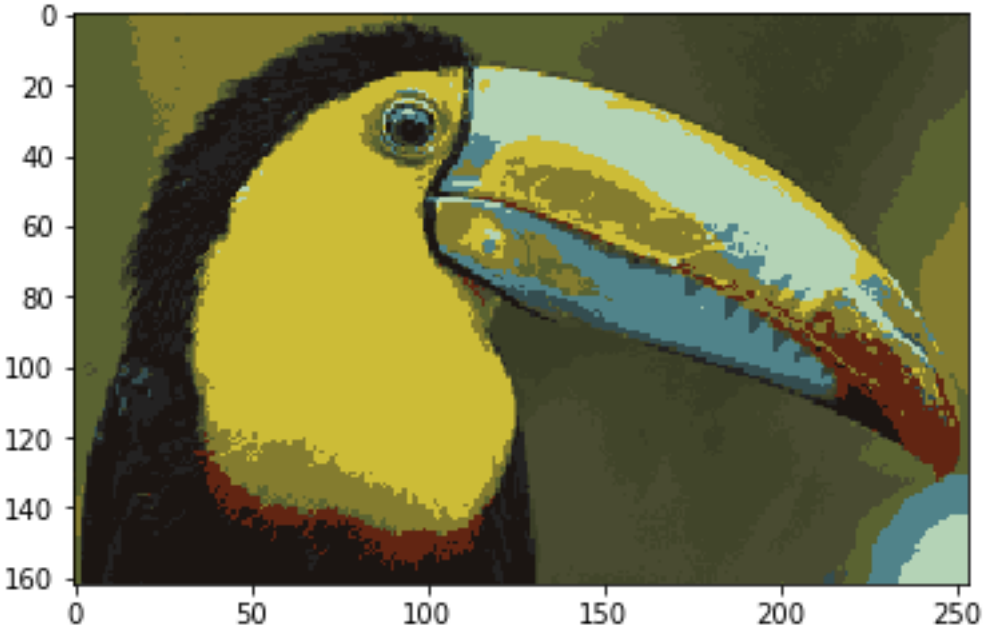
i. 10


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

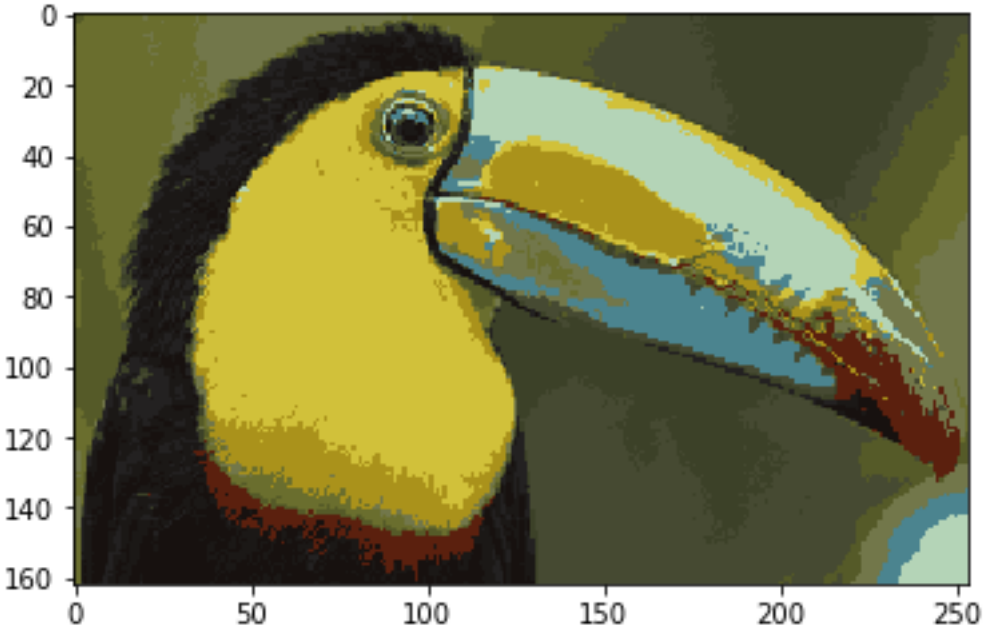
j. 11


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

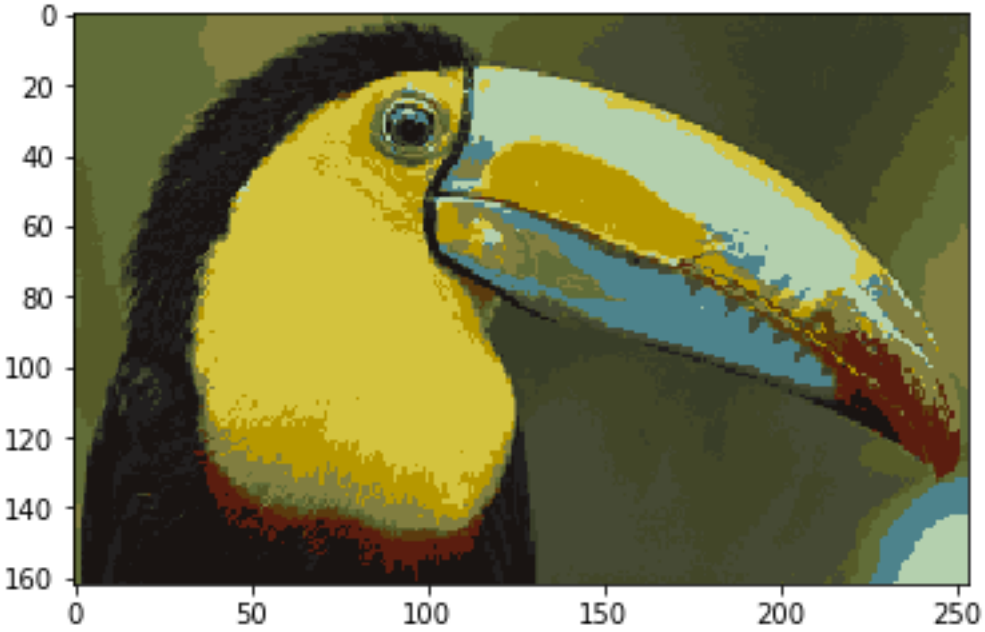
k. 12


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

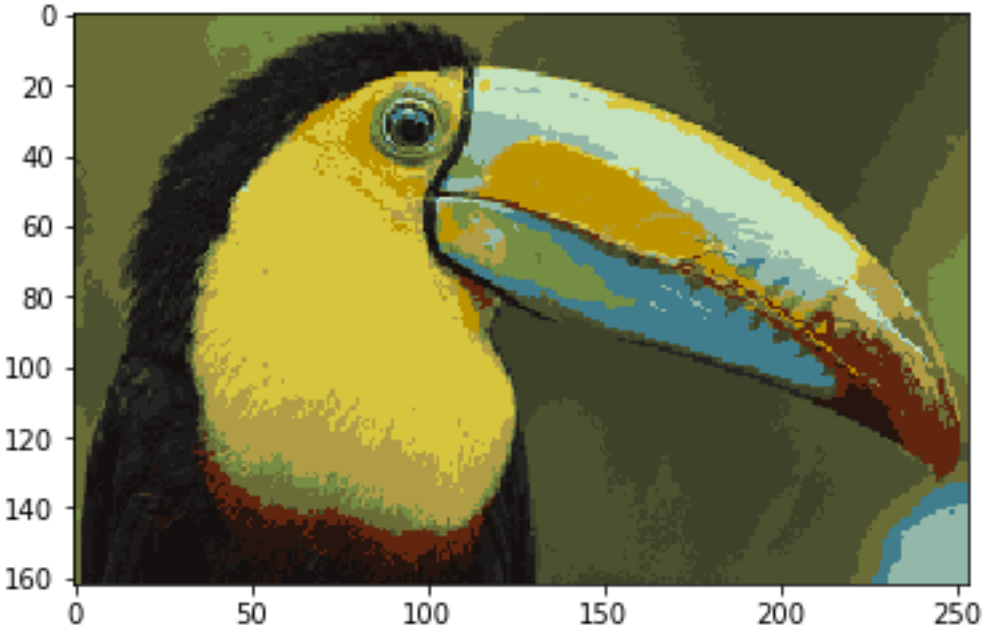
l. 13


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

m. 14

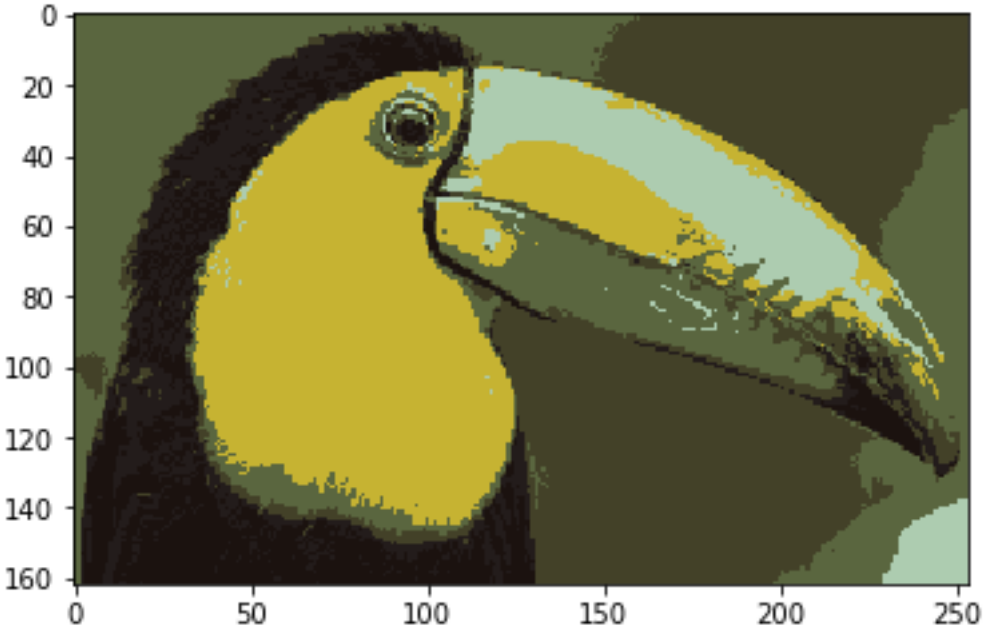


n. 15

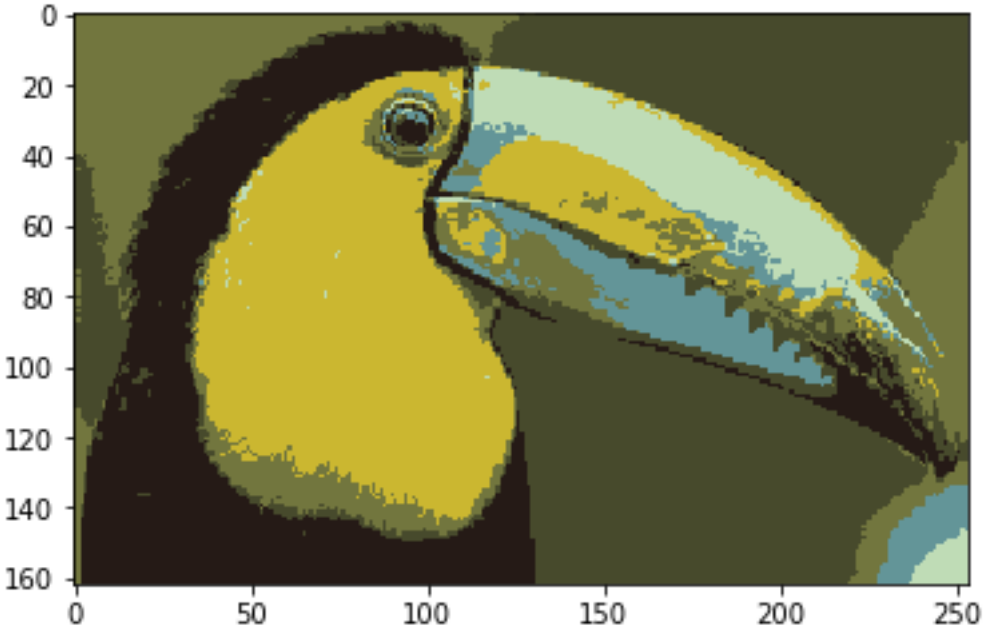


Overall the images improve as k increases. Depending on the centroid clusters, the image improves variably, but as a whole, they improve noticeably in the images above. We can see this as we scroll down through them.

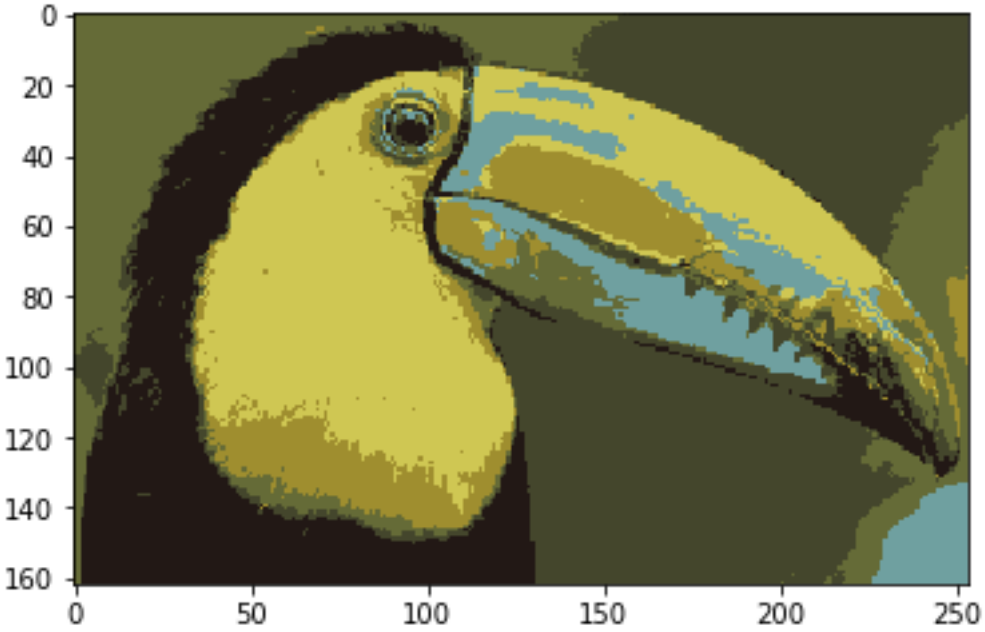
2. k = 6 examples
    a.


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

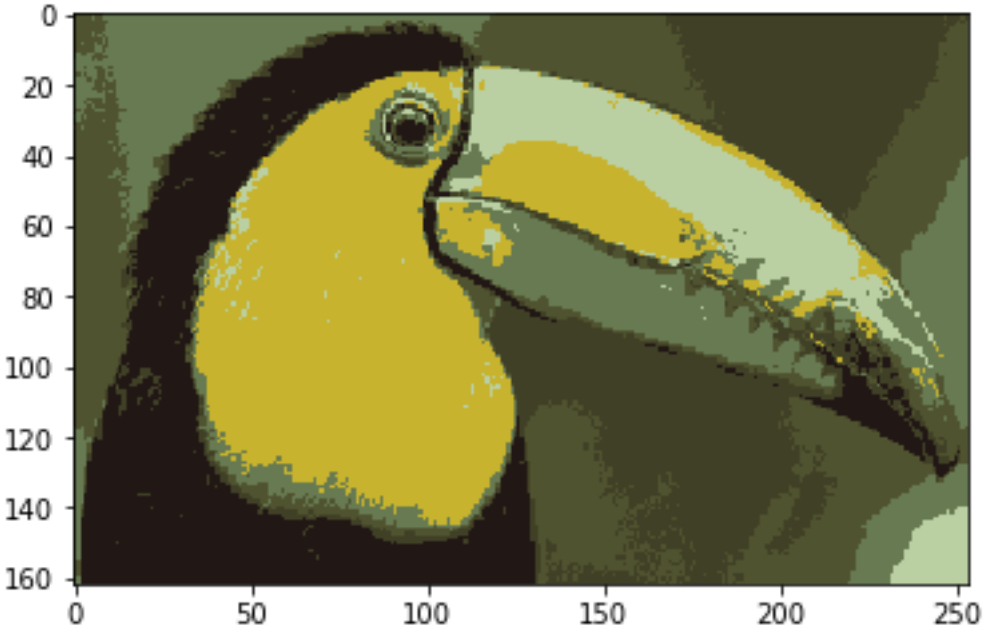
b.


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

c.

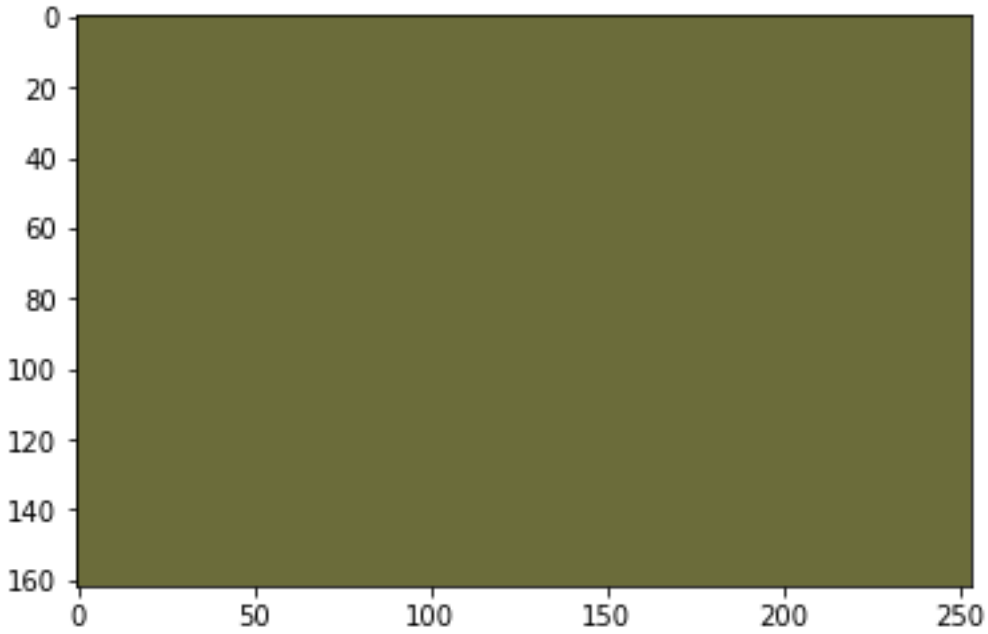


d.



We can see by looking at these images that the centroid clusters are different between each of them. They are close in most of them, but if we examine closer we can see pixels with slightly different RGB values and centroids.

3. k = 1
    a.



It takes one iteration to get the average color for the image, which is returned as the only RGB value. Since there is only one centroid, it becomes the average for all the points immediately, and all the points fit to it since there is no other centroid.

4. Code

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 18 22:01:20 2017

@author: rditljtd
"""
import numpy as np
from scipy import misc
from scipy import sparse as sps
import matplotlib.pyplot as plt
from collections import namedtuple
from math import sqrt
import random
try:
    import Image
except ImportError:
    from PIL import Image

#make array of RGB values for each pixel in image
A = misc.imread('b_small.tiff', mode='RGB')

plt.imshow(A)

num_of_centroids = 16

#instatiate array of random 16 pixels (used for cluster centroids)
random_16 = []

```

```
#loop 16 times to actually choose the random 16 pixels for cluster centroids
for i in range (0, num_of_centroids):

    #generate random x-value
    random_x = random.randint(0, 253)
    #generate random y-value
    random_y = random.randint(0, 161)

    #add random pixel to array of 16 random centroids
    random_16.append(A[random_y][random_x])

#set number of iterations
iterations = 50


#print out the centroids
#print random_16

#print out the last pixel in the image
print A[161][253]

#create an array for all pixels that is equal to the array of RGB values for the
 image
all_pixels = A

#create an array for associating pixels to centroids
pixels_to_centroids = []

#loop the number of iteraions
for a in range(0, iterations):
    print a+1,

    #loop through all y-values
    for i in range (0, len(A)):

        #loop through all x-values
        for j in range (0, len(A[0])):

            #print out this pixel's RGB values
            #print A[i][j]

            #set this_pixel variable equal to this pixel's RGB values
            this_pixel = A[i][j]

            #set distance = to maximum distance possible
            distance = 3*253

            #create a variable for closest centroid
            pixel_to_centroid = [0, 0, 0, [0, 0, 0]]

            #loop through the cluster centroids
            for k in range(0, len(random_16)):

                #set this_distance equal to 0
                this_distance = 0

                #loop through the red, green, blue values for this centroid
                for l in range (0, len(random_16[0])):

                    #calculate the distance between the r, g, or b value for thi
s centroid and this pixel
```

```
                    this_distance += abs(int(random_16[k][l]) -
    int(this_pixel[l]))

                #if the distance between the RGB values for this centroid is les
    s than all other centroids
                if (this_distance < distance):

                    #set the distance to the minimum distance calculated
                    distance = this_distance

                            #set the closest centroid
                            pixel_to_centroid = [int(k), int(i), int(j), A[i][j]
    ]

                    #set this pixel's centroid to closest
                    pixels_to_centroids.append(pixel_to_centroid)

                            #all_pixels[i][j] = this_pixel


            #create variable for number of values associated with each centroid

            num_assoc_16 = []

            #create variable for sum of values associated with each centroid
            sum_16 = []

            #create variable for new centroids
            average_16 = []

            #loop through centroids
            for i in range (0, len(random_16)):
                num_assoc_16.append(0)
                sum_16.append([0, 0, 0])

                #loop through each pixel
                for j in range(0, len(pixels_to_centroids)):

                    #if this pixel is associated with this cluster
                    if (pixels_to_centroids[j][0] == i):

                        #number of points associated with this centroid
                        num_assoc_16[i] += 1
                        #sum up the R values
                        sum_16[i][0] += int(pixels_to_centroids[j][3][0])
                        #sum up the G values
                        sum_16[i][1] += int(pixels_to_centroids[j][3][1])
                        #sum up the B values
                        sum_16[i][2] += int(pixels_to_centroids[j][3][2])

            for i in range (0, len(sum_16)):
                average_16.append([sum_16[i][0] / num_assoc_16[i],
                                   sum_16[i][1] / num_assoc_16[i],
                                   sum_16[i][2] / num_assoc_16[i]])

            #set centroids as the averages
            if (np.array_equal(np.array(random_16), np.array(average_16))):
                break
            random_16 = average_16

        B = A
```

```
150.
151.       #loop through all pixels
152.       for a in range (0, len(pixels_to_centroids)):
153.
154.           i=pixels_to_centroids[a][1]
155.           j=pixels_to_centroids[a][2]
156.
157.           k=pixels_to_centroids[a][0]
158.           B[i][j] = random_16[k]
159.
160.       plt.imshow(B)
161.       plt.savefig('kmeans-' + str(iterations) + '.png')
162.
163.       #loop through y-values
164.       #for i in range (0, len(all_pixels)):
165.
166.           #loop through x-values
167.           #for j in range (0, len(all_pixels[0])):
168.
169.               #loop through centroids
170.               #for k in range (0, len(random_16)):
171.
172.                   #if this pixel is closest to this centroid
173.                   #if all_pixels[i][j] = random_16[k]
174.
```