**Programming Assignment 3**

**Part A:**

1. Misclassified 71 emails which is 27 percent.
2. Train on different numbers of training examples:
   1. 50 – misclassified 132 which is 50 percent.
   2. 100 – misclassified 125 which is 48 percent.
   3. 400 – misclassified 117 which is 45 percent.
3. Code
4. #!/usr/bin/env python2
5. # -\*- coding: utf-8 -\*-
6. """
7. Created on Wed Oct 11 20:21:17 2017
9. @author: rditljtd
10. """
12. **import** numpy as np
13. **from** scipy **import** misc
14. **from** scipy **import** sparse as sps
15. **import** matplotlib.pyplot as plt
16. **from** math **import** \*
18. #find probability of email being spam given that it contains a particular word
19. #this equals the probability of seeing this word given that it is a spam \* probability of it being seen overall / probability of it being spam
21. #find probability of email being non-spam given that it contains a particular word
22. #this equls the probabiltiy of seeing this word givent that the email is non-spam \*
23. #probability of seeing this word overall
24. #/probability of email being non-spam
26. # Load the labels for the training set
27. train\_labels = np.loadtxt('train-labels-50.txt',dtype=int)
29. # Get the number of training examples from the number of labels
30. numTrainDocs = train\_labels.shape[0]
32. # This is how many words we have in our dictionary
33. numTokens = 2500
35. # Load the training set feature information
36. M = np.loadtxt('train-features-50.txt',dtype=int)
38. # Create matrix of training data
39. train\_matrix = sps.csr\_matrix((M[:,2], (M[:,0], M[:,1])),shape=(numTrainDocs,numTokens))
41. #train\_labels[i] = ith document label: spam or not spam
43. #tran\_matrix[i:] = ith document
45. #train\_matrix[i:, j] = jth word in ith document
47. #print train\_matrix[69]
49. **print** train\_matrix.shape[1]
51. spam\_prob = (sum(x == 1 **for** x **in** train\_labels))/(numTrainDocs\*(1.0))
53. nonspam\_prob = (sum(x == 0 **for** x **in** train\_labels))/(numTrainDocs\*(1.0))
55. **print** spam\_prob, nonspam\_prob
57. #create variable for sum of all non spam emails
58. num\_nonspam = (sum(x==0 **for** x **in** train\_labels))
59. #create variable for sum of all spam emails
60. num\_spam = (sum(x==1 **for** x **in** train\_labels))
62. #create array for probability all words in spam and non spam [spam\_prob, nonspam\_prob]
63. prob\_for\_each\_word = []

66. #add up the instances of this word [spam, nonspam]
67. sum\_for\_this\_word = [0, 0]
69. #create array for how many times each word was foud [[spam, nonspam]] i = word
70. found\_for\_all\_emails = []
72. #loop through each email
73. **for** j **in** range (0, train\_matrix.shape[0]):
75. #array of words in email [instances]
76. words\_in\_email = train\_matrix[j].toarray()[0]
77. #print train\_matrix[j],


81. #loop through each word in dictionary and see if it is in email
82. **for** k **in** range (0, len(words\_in\_email)):
84. **if** j == 0:
85. #create array for if this word was found [spam, nonspam]
86. found\_for\_all\_emails.append([0, 0])
88. #if email is non spam and there is more than zero instances of this word
89. **if** (train\_labels[j] == 0 **and** words\_in\_email[k] > 0):
91. #increment the sum for this word for non spam emails
92. found\_for\_all\_emails[k][1] += 1
94. #else if the email is spam and there is more than zero instances of this word
95. **elif** (train\_labels[j] == 1 **and** words\_in\_email[k] > 0):
97. #print words\_in\_email[k]
98. #increment the sum for this word for spam emails
99. found\_for\_all\_emails[k][0] += 1
101. #print found\_for\_all\_emails
102. #sum\_for\_this\_word = sum(found\_for\_all\_emails[:1])
103. #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 = []
108. #create array to hold the overall probability for each word [num] i = word
109. overall\_prob\_for\_each\_word = []
111. **for** i **in** range (0, numTokens):
113. prob\_for\_each\_word.append([0,0])
114. overall\_prob\_for\_each\_word.append(0)
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)))
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)))
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))

125. #print prob\_for\_each\_word
126. #print overall\_prob\_for\_each\_word

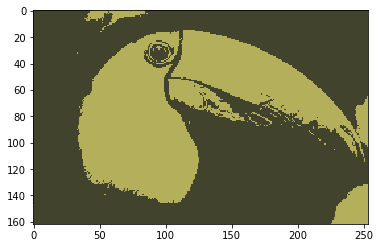

130. # Load the labels for the training set
131. test\_labels = np.loadtxt('test-labels.txt',dtype=int)
133. # Get the number of training examples from the number of labels
134. numTestDocs = test\_labels.shape[0]
136. # Load the training set feature information
137. N = np.loadtxt('test-features.txt',dtype=int)
139. # Create matrix of training data
140. test\_matrix = sps.csr\_matrix((N[:,2], (N[:,0], N[:,1])),shape=(numTestDocs,numTokens))
142. prob\_for\_all\_test\_emails = []
144. #iterate through each email in the test set
145. **for** i **in** range (0, (test\_matrix.shape[0])):
147. #array of words in email [instances]
148. words\_in\_test\_email = test\_matrix[i].toarray()[0]
149. #print train\_matrix[j],
151. prob\_spam\_given\_all\_words = 0
152. prob\_nonspam\_given\_all\_words = 0
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)):
157. prob\_spam\_given\_word = 0
158. prob\_nonspam\_given\_word = 0
160. #if email is non spam and there is more than zero instances of this 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\_for\_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)))
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
175. **if** (prob\_nonspam\_given\_word > 0):
176. #print prob\_nonspam\_given\_all\_words,
177. #prob\_nonspam\_given\_all\_words += float((prob\_nonspam\_given\_word\*(1.0)))
178. prob\_nonspam\_given\_all\_words += prob\_nonspam\_given\_word
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\_prob)))
182. prob\_for\_all\_test\_emails.append([prob\_spam\_given\_all\_words, prob\_nonspam\_given\_all\_words])
184. **print** prob\_for\_all\_test\_emails
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\_labels[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\_labels[k]])
202. **else**:
203. classify\_each\_test\_email.append('unknown')
204. incorrect.append([k, classify\_each\_test\_email[k], test\_labels[k]])
206. **print** classify\_each\_test\_email
207. **print** "spam count: " + str(spam\_count) + " spam percentage: " + str(float((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 misclassified: " + str(float(((len(incorrect))\*100)/(len(classify\_each\_test\_email))))



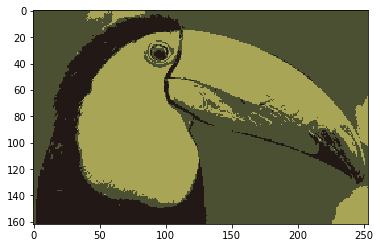

215. #def nBayes(train\_matrix, train\_labels):
216. #    print 'called nBayes'
217. #    #reset num\_unique\_words\_in\_spam ratio and wordsInSpam
218. #
219. #    #create variable for number of unique words in spam emails
220. #    num\_unique\_words\_in\_spam = 0
221. #
222. #    #create variable 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 instances
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 spam 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 words 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 instances 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 email 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 non spam email
265. #                num\_unique\_words\_in\_nonspam = num\_unique\_words\_in\_nonspam+1
266. #                #add the # of instances of this word to the number of words total in the email
267. #                totalWordsInNonSpam = totalWordsInNonSpam + wordsInEmail[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 instances 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 emails 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 unique word in the dictionary, calculate the percentage of spam emails that contain that word
280. #    #then calculate the # of non spam emails that contain that word. Multiply 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 emails + # 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\_matrix.shape[1])\*1.0)
299. #
300. #print nBayes(train\_matrix, train\_labels)

**Part B:**

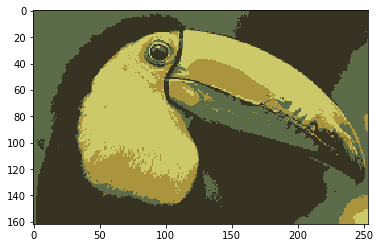
1. Images for k = 2..15
   1. 2



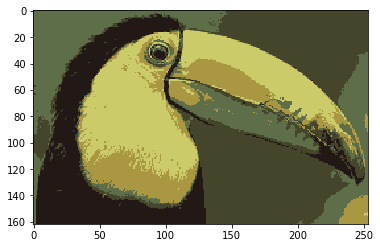
* 1. 3



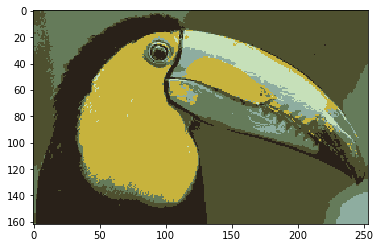
* 1. 4



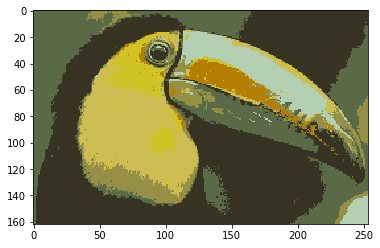
* 1. 5



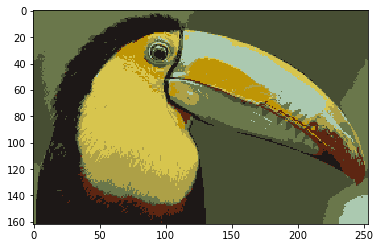
* 1. 6



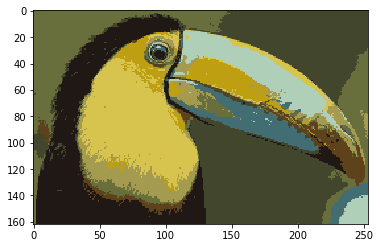
* 1. 7



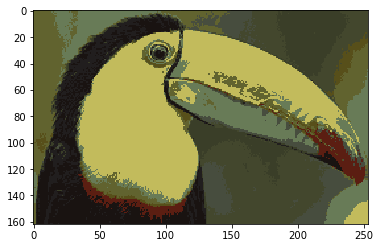
* 1. 8



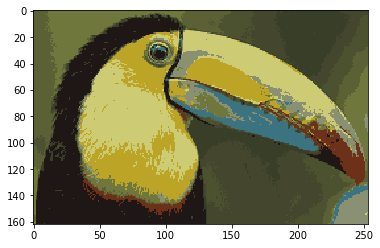
* 1. 9



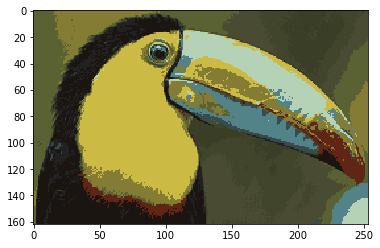
* 1. 10



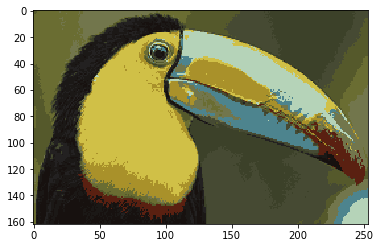
* 1. 11



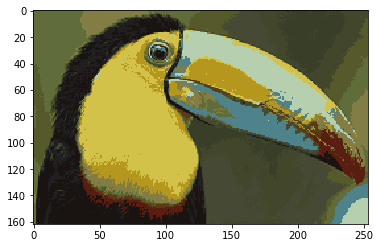
* 1. 12



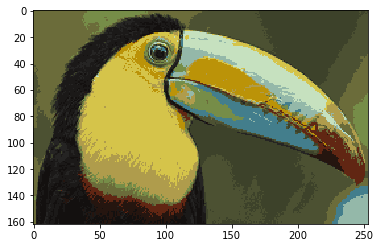
* 1. 13



* 1. 14

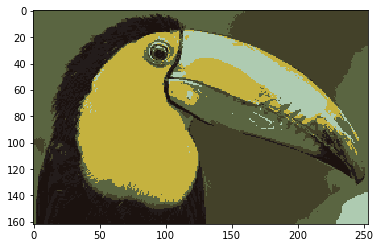


* 1. 15

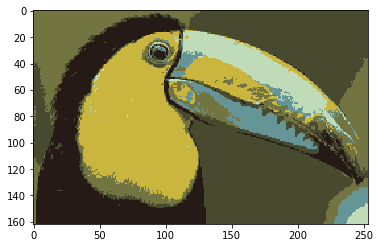


Overall the images improve as k increases. Depending on the centroid clusters, the image improves variably, but as a whole, they improve noticeably in the images above. We can see this as we scroll down through them.

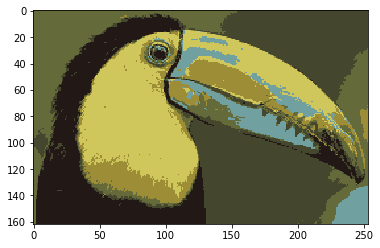
1. k = 6 examples



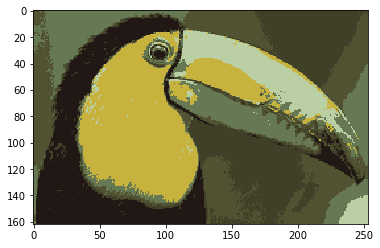






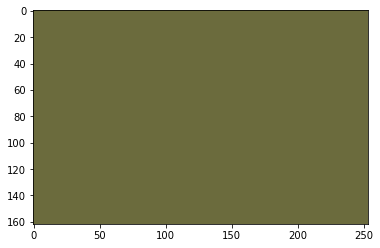






We can see by looking at these images that the centroid clusters are different between each of them. They are close in most of them, but if we examine closer we can see pixels with slightly different RGB values and centroids.

1. k = 1



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.

1. Code
2. #!/usr/bin/env python2
3. # -\*- coding: utf-8 -\*-
4. """
5. Created on Wed Oct 18 22:01:20 2017
7. @author: rditljtd
8. """
9. **import** numpy as np
10. **from** scipy **import** misc
11. **from** scipy **import** sparse as sps
12. **import** matplotlib.pyplot as plt
13. **from** collections **import** namedtuple
14. **from** math **import** sqrt
15. **import** random
16. **try**:
17. **import** Image
18. **except** ImportError:
19. **from** PIL **import** Image
21. #make array of RGB values for each pixel in image
22. A = misc.imread('b\_small.tiff', mode='RGB')
24. plt.imshow(A)
26. num\_of\_centroids = 16
28. #instatiate array of random 16 pixels (used for cluster centroids)
29. random\_16 = []
31. #loop 16 times to actually choose the random 16 pixels for cluster centroids
32. **for** i **in** range (0, num\_of\_centroids):
34. #generate random x-value
35. random\_x = random.randint(0, 253)
36. #generate random y-value
37. random\_y = random.randint(0, 161)
39. #add random pixel to array of 16 random centroids
40. random\_16.append(A[random\_y][random\_x])
42. #set number of iterations
43. iterations = 50

46. #print out the centroids
47. #print random\_16
49. #print out the last pixel in the image
50. **print** A[161][253]
52. #create an array for all pixels that is equal to the array of RGB values for the image
53. all\_pixels = A
55. #create an array for associating pixels to centroids
56. pixels\_to\_centroids = []
58. #loop the number of iteraions
59. **for** a **in** range(0, iterations):
60. **print** a+1,
62. #loop through all y-values
63. **for** i **in** range (0, len(A)):
65. #loop through all x-values
66. **for** j **in** range (0, len(A[0])):
68. #print out this pixel's RGB values
69. #print A[i][j]
71. #set this\_pixel variable equal to this pixel's RGB values
72. this\_pixel = A[i][j]
74. #set distance = to maximum distance possible
75. distance = 3\*253
77. #create a variable for closest centroid
78. pixel\_to\_centroid = [0, 0, 0, [0, 0, 0]]
80. #loop through the cluster centroids
81. **for** k **in** range(0, len(random\_16)):
83. #set this\_distance equal to 0
84. this\_distance = 0
86. #loop through the red, green, blue values for this centroid
87. **for** l **in** range (0, len(random\_16[0])):
89. #calculate the distance between the r, g, or b value for this centroid and this pixel
90. this\_distance += abs(int(random\_16[k][l]) - int(this\_pixel[l]))
92. #if the distance between the RGB values for this centroid is less than all other centroids
93. **if** (this\_distance < distance):
95. #set the distance to the minimum distance calculated
96. distance = this\_distance
98. #set the closest centroid
99. pixel\_to\_centroid = [int(k), int(i), int(j), A[i][j]]
101. #set this pixel's centroid to closest
102. pixels\_to\_centroids.append(pixel\_to\_centroid)
104. #all\_pixels[i][j] = this\_pixel

107. #create variable for number of values associated with each centroid
108. num\_assoc\_16 = []
110. #create variable for sum of values associated with each centroid
111. sum\_16 = []
113. #create variable for new centroids
114. average\_16 = []
116. #loop through centroids
117. **for** i **in** range (0, len(random\_16)):
118. num\_assoc\_16.append(0)
119. sum\_16.append([0, 0, 0])
121. #loop through each pixel
122. **for** j **in** range(0, len(pixels\_to\_centroids)):
124. #if this pixel is associated with this cluster
125. **if** (pixels\_to\_centroids[j][0] == i):
127. #number of points associated with this centroid
128. num\_assoc\_16[i] += 1
129. #sum up the R values
130. sum\_16[i][0] += int(pixels\_to\_centroids[j][3][0])
131. #sum up the G values
132. sum\_16[i][1] += int(pixels\_to\_centroids[j][3][1])
133. #sum up the B values
134. sum\_16[i][2] += int(pixels\_to\_centroids[j][3][2])
136. **for** i **in** range (0, len(sum\_16)):
137. average\_16.append([sum\_16[i][0] / num\_assoc\_16[i],
138. sum\_16[i][1] / num\_assoc\_16[i],
139. sum\_16[i][2] / num\_assoc\_16[i]])
141. #set centroids as the averages
142. **if** (np.array\_equal(np.array(random\_16), np.array(average\_16))):
143. **break**
144. random\_16 = average\_16
146. B = A
148. #loop through all pixels
149. **for** a **in** range (0, len(pixels\_to\_centroids)):
151. i=pixels\_to\_centroids[a][1]
152. j=pixels\_to\_centroids[a][2]
154. k=pixels\_to\_centroids[a][0]
155. B[i][j] = random\_16[k]
157. plt.imshow(B)
158. plt.savefig('kmeans-' + str(iterations) + '.png')
160. #loop through y-values
161. #for i in range (0, len(all\_pixels)):
163. #loop through x-values
164. #for j in range (0, len(all\_pixels[0])):
166. #loop through centroids
167. #for k in range (0, len(random\_16)):
169. #if this pixel is closest to this centroid
170. #if all\_pixels[i][j] = random\_16[k]