НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Дисциплина: «Анализ данных»

Домашнее задание на тему:

«Лабораторная работа №13»

Выполнил: Осипов Лев,

студент группы 301ПИ (1).

Москва, 2015 г.

**СОДЕРЖАНИЕ**

**Практическая часть3**

**Задание 13**

**Задание 28**

**Задание 311**

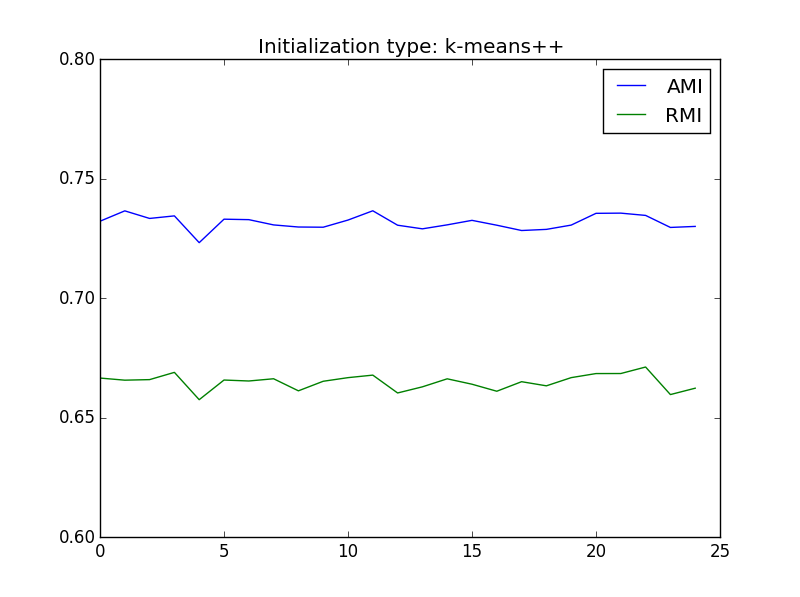
**Список литературы16**

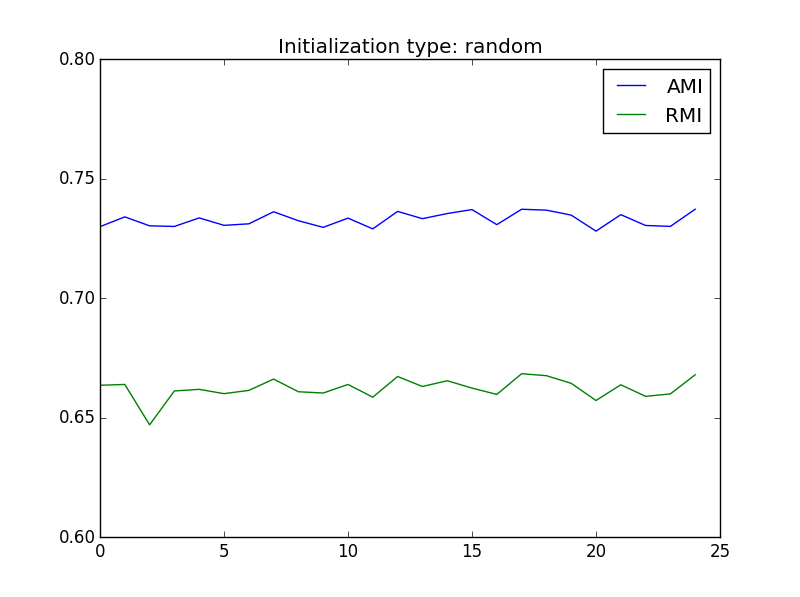
**Текст программы17**

**ПРАКТИЧЕСКАЯ ЧАСТЬ**

**ЗАДАНИЕ 1**

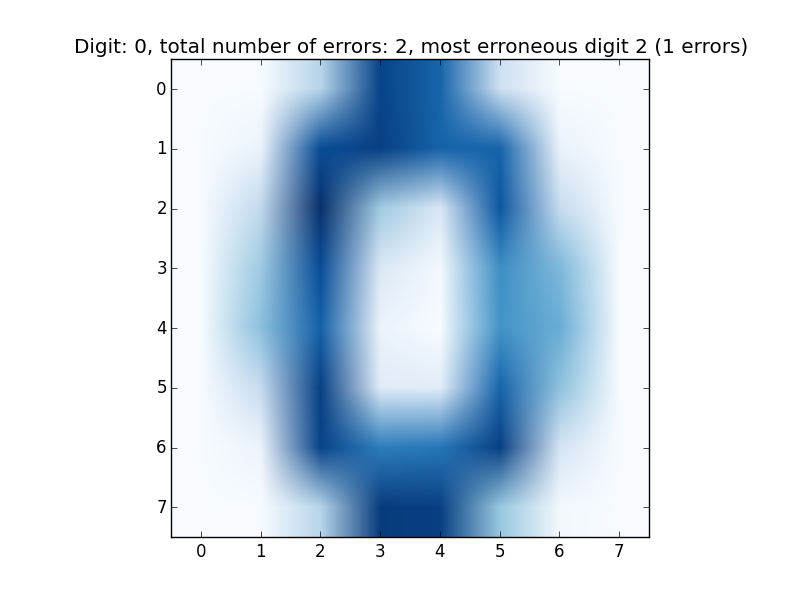
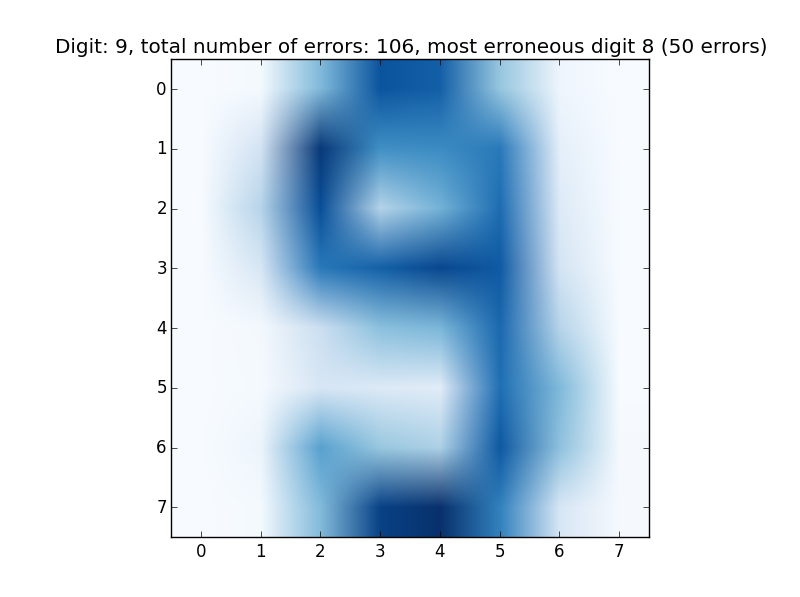
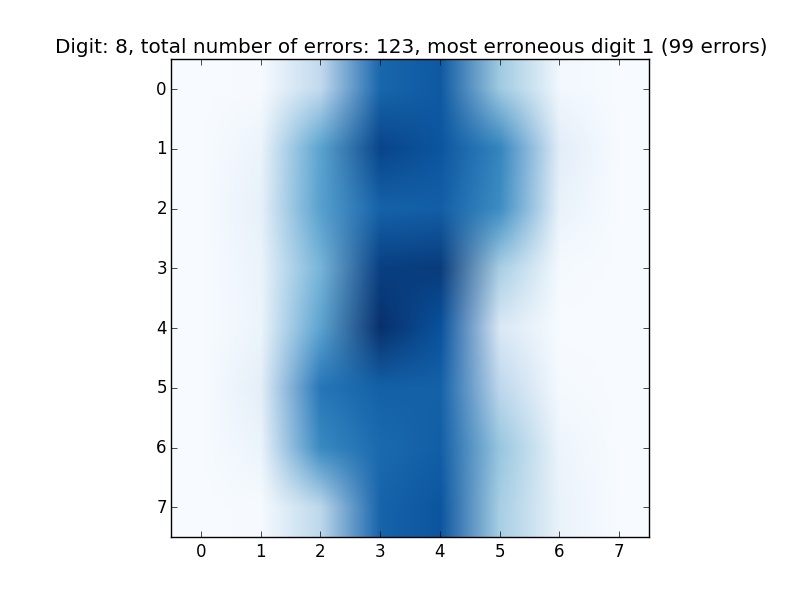
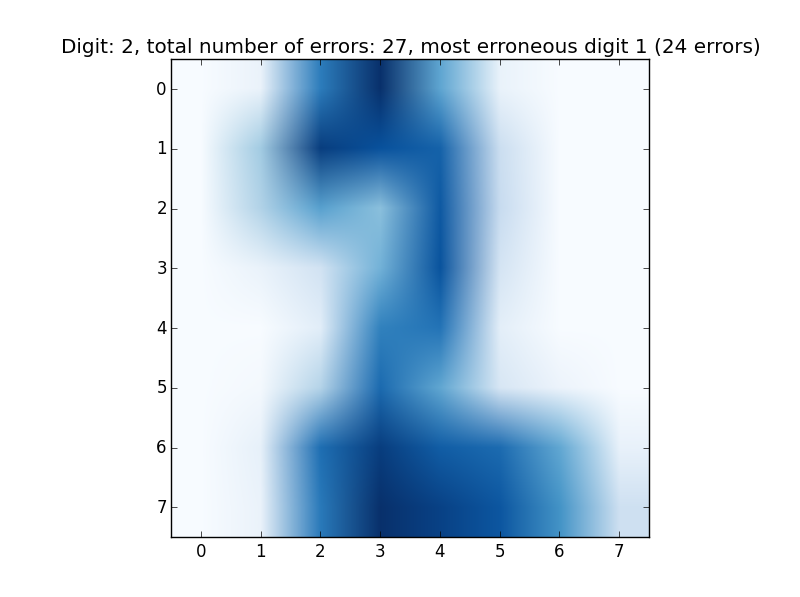
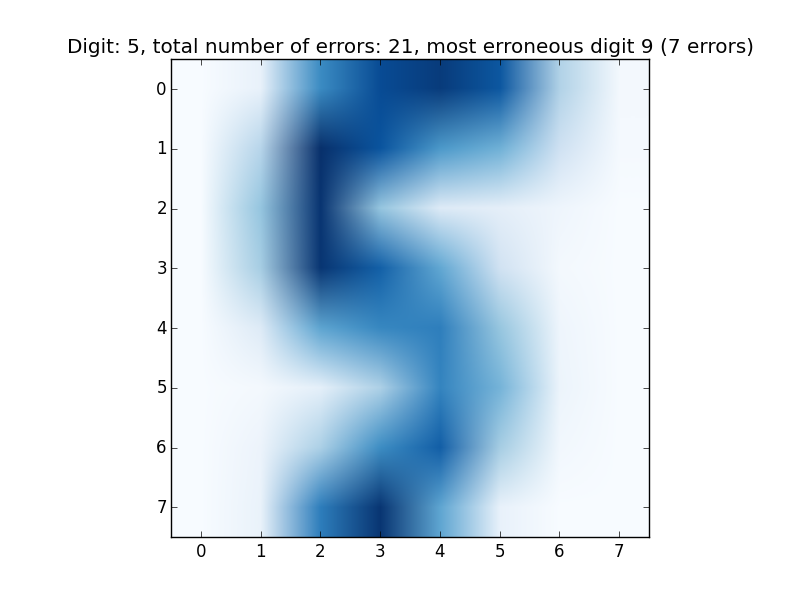
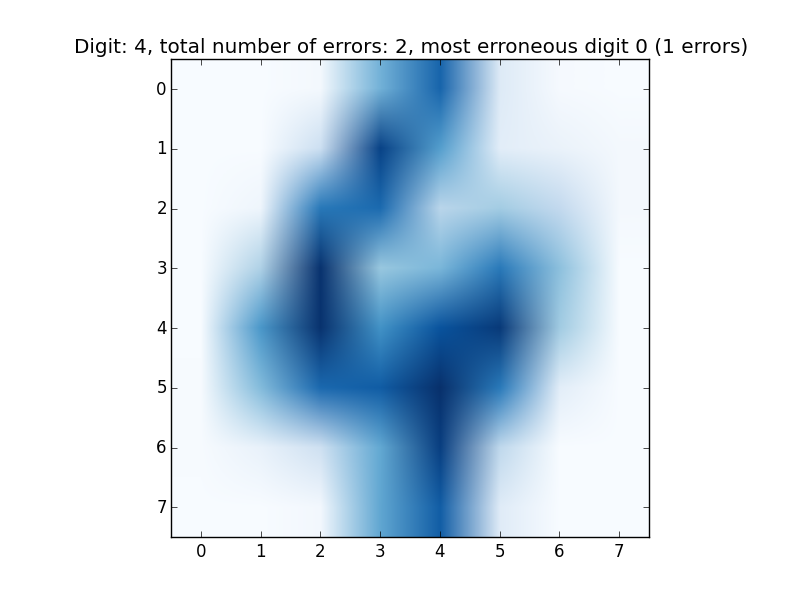
Для начала были посчитаны динамики двух метрик: Adjusted Mutual Information (AMI) и Adjusted Rand Index (RMI), при двух разных типах начального приближения (k-means++ и random), на 25 экспериментах:

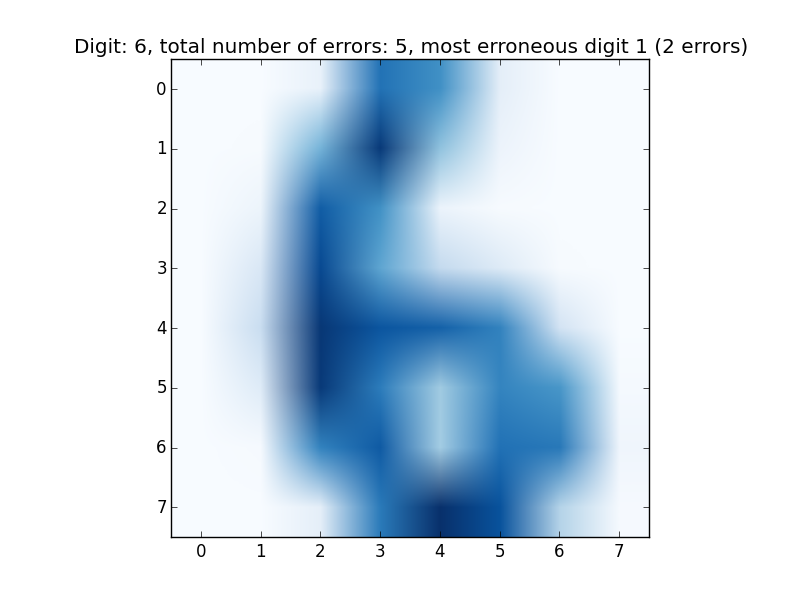
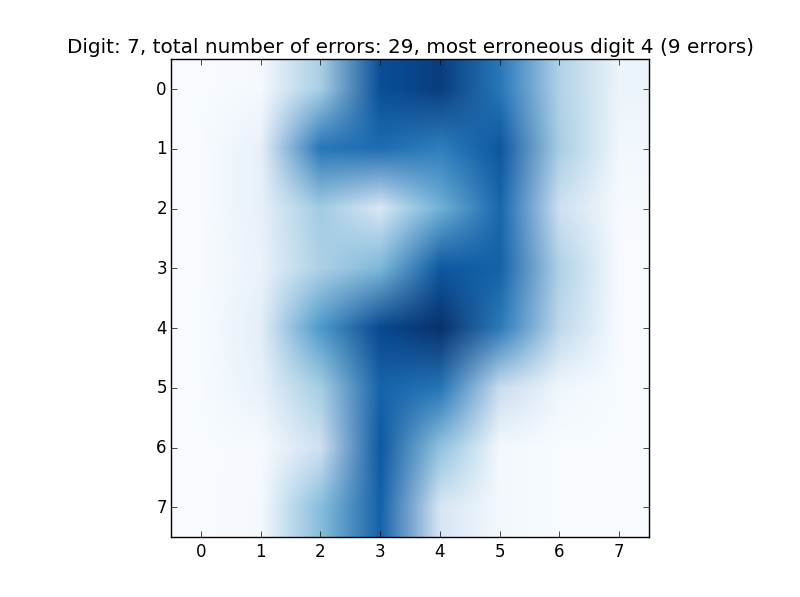
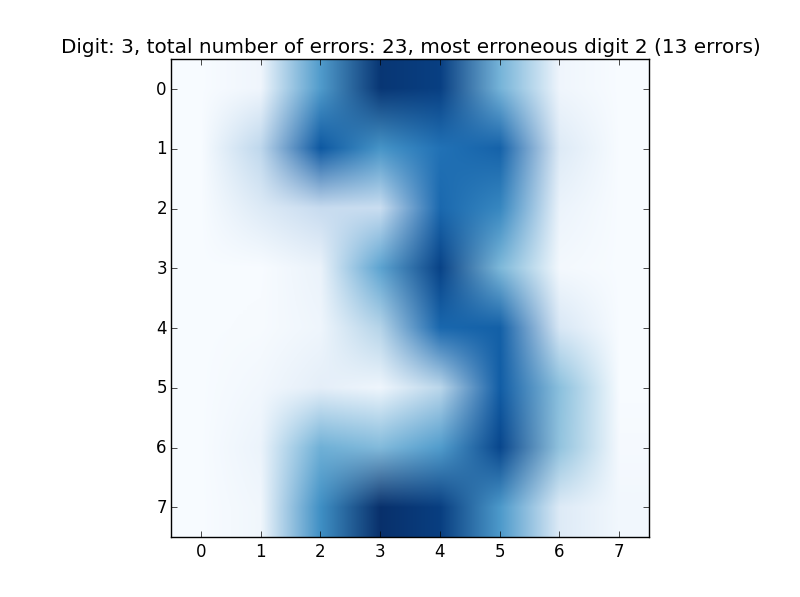
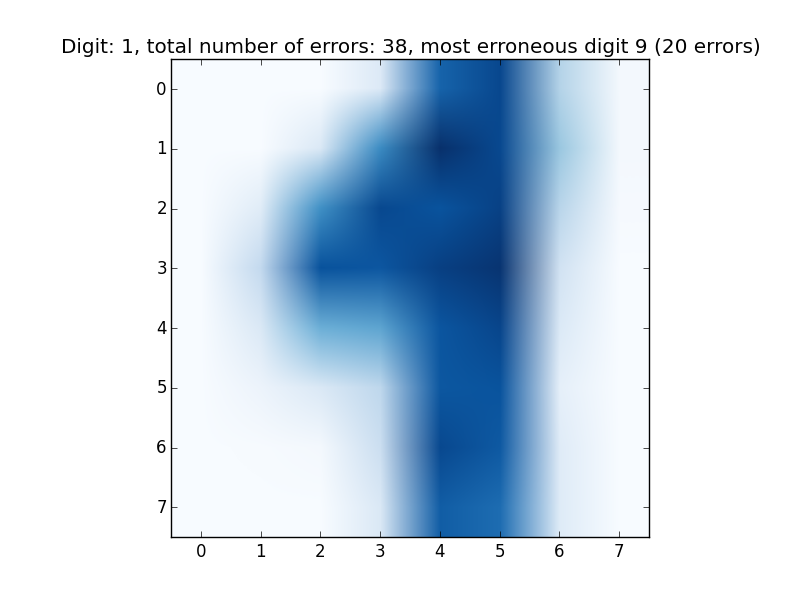




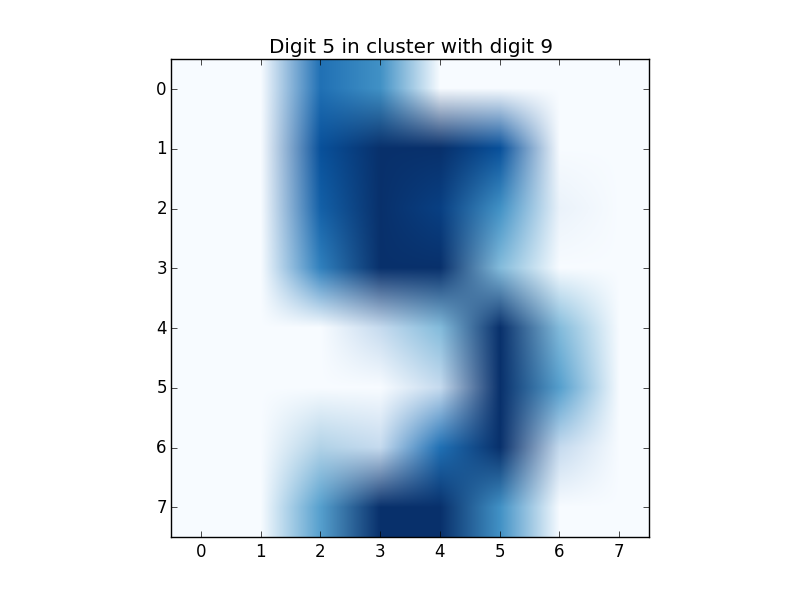
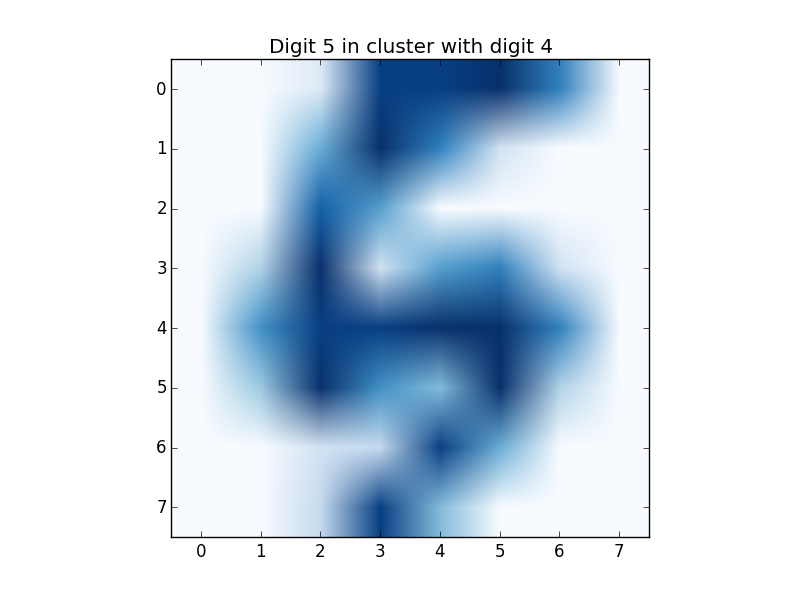
Как видно по графикам динамики, в целом метрики между собой согласуются неплохо.

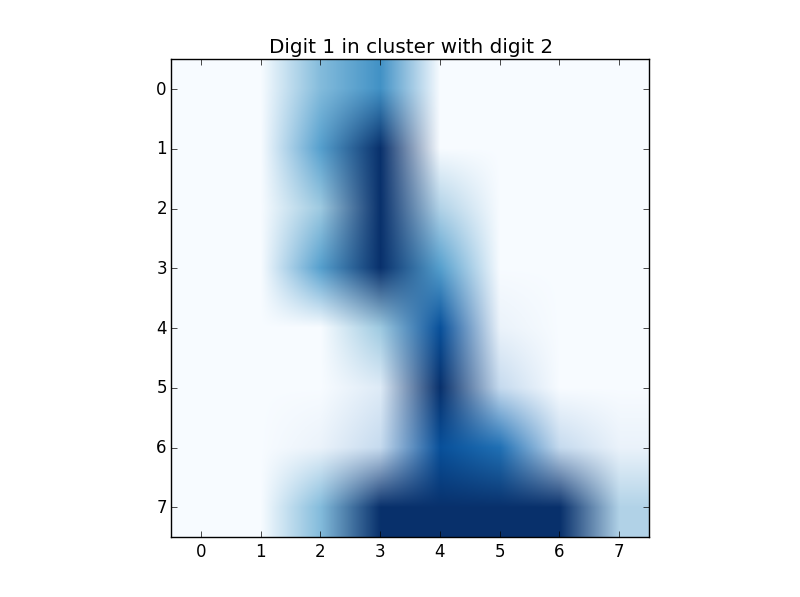
Далее, была проведена очередная кластеризация. При этом было определено соответствие меток и кластеров, было посчитано общее количество ошибочных объектов в каждом кластере, также было посчитано, какая метка в каждом кластере наиболее ошибочна. Данные были визуализированы:





Также были высчитаны и визуализированы примеры ошибок:

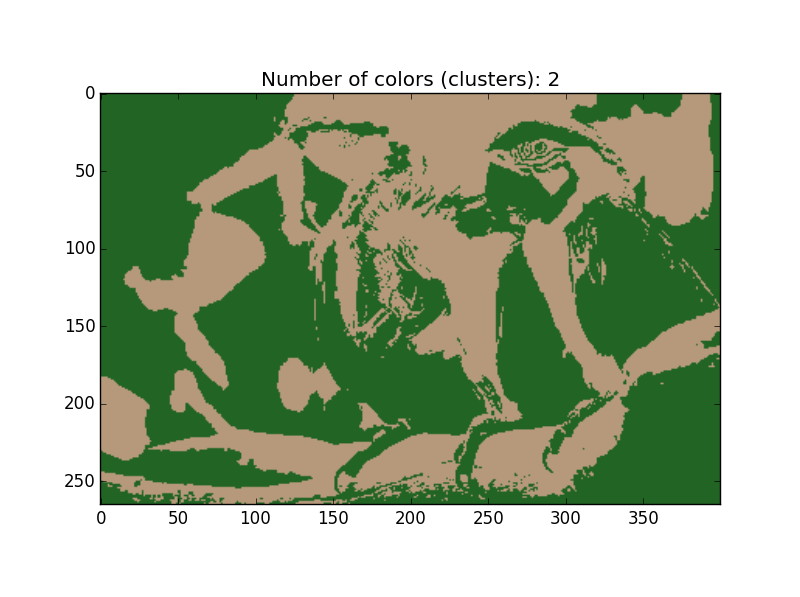


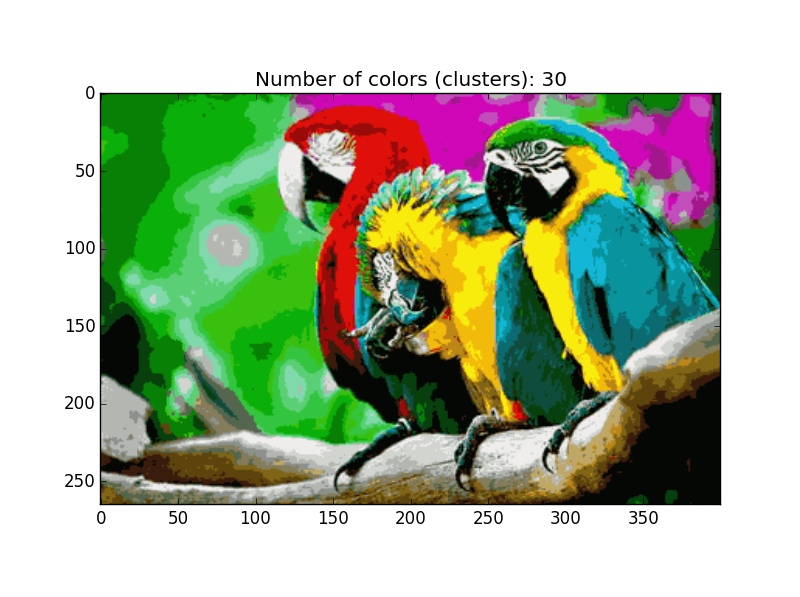
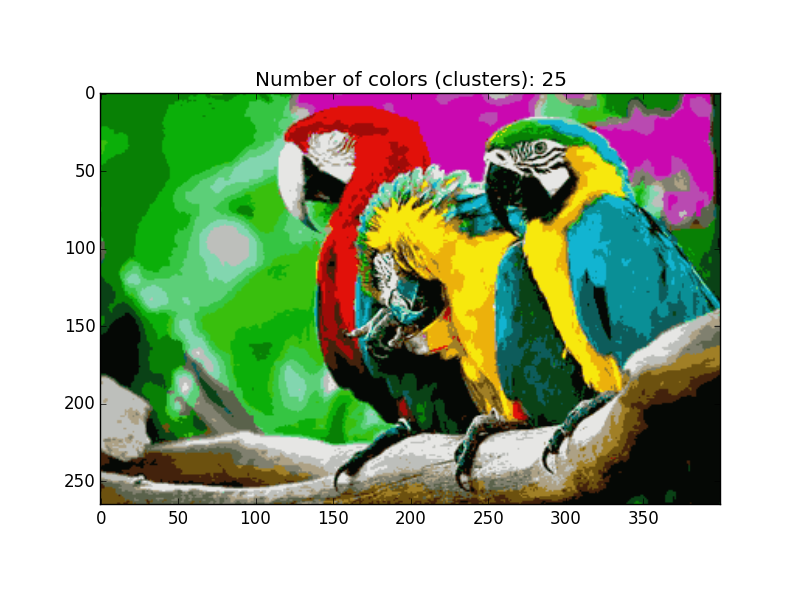
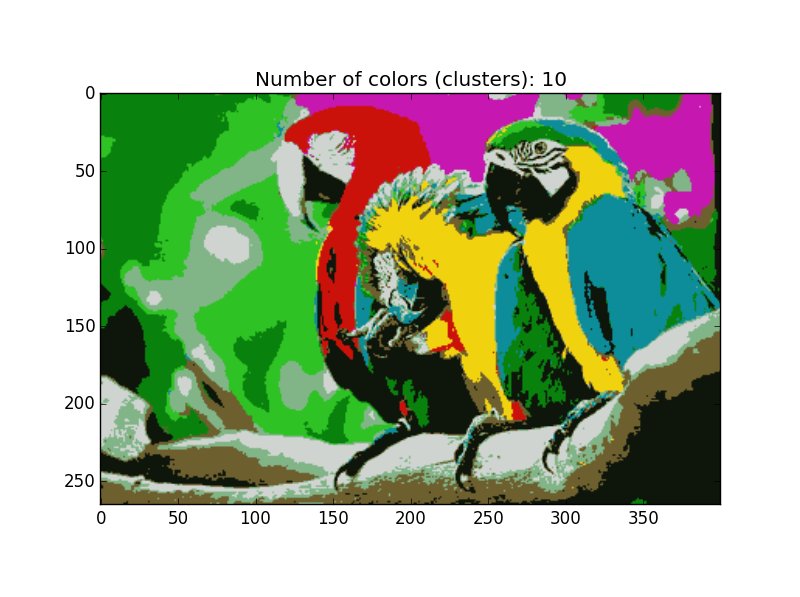
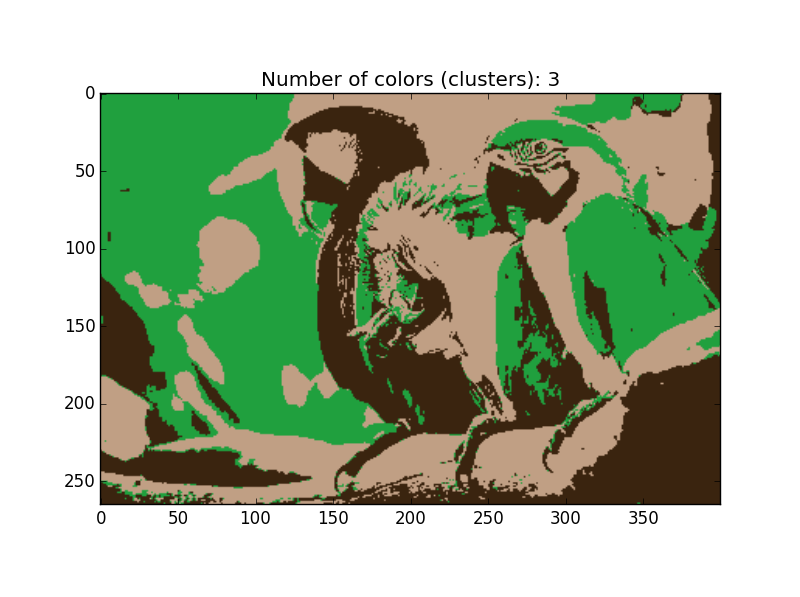
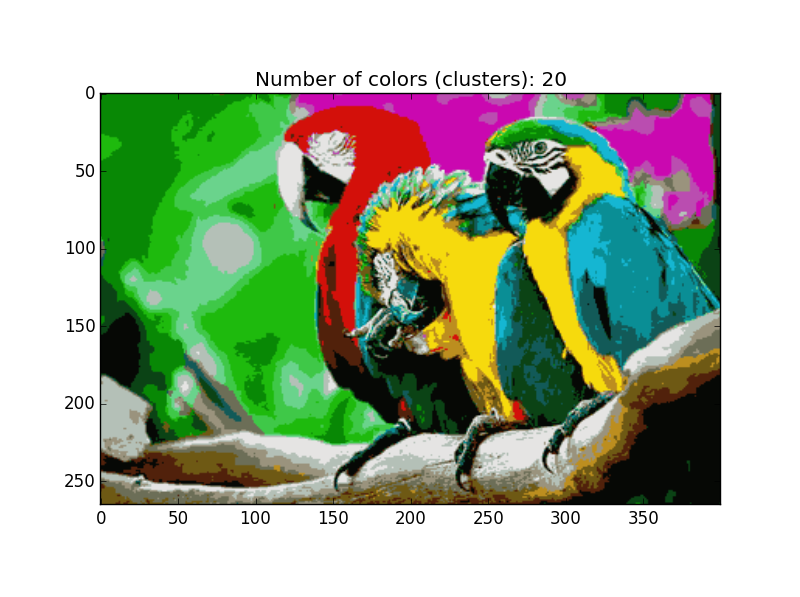


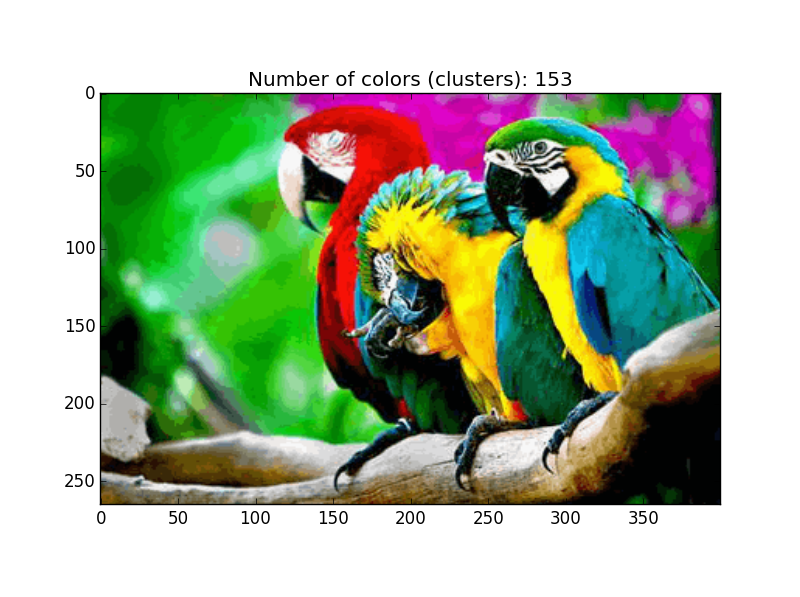
В целом можно сказать, что на основе большинства результатов кластеры выделились верно и охватили все метки цифр. Но заметны и крупные недочеты, например, с восьмеркой. Видно, что ввиду ее строения для нее построился достаточно нечеткий центр кластера, из-за которого возникло множество ошибок, особенно с единицей, которая зачастую была отнесена к этому кластеру.

Из-за «рукописных» цифр зачастую возникали весьма закономерные ошибки, как, например, на картинке с примером ошибки, где пятерка была ошибочно воспринята за четверку. Здесь уже нельзя винить алгоритм, так как на вид мог бы ошибиться даже человек.

**ЗАДАНИЕ 2**

Были произведены и визуализированы кластеризации для картинки с попугаями с различным количеством кластеров (в данном случае – цветов):

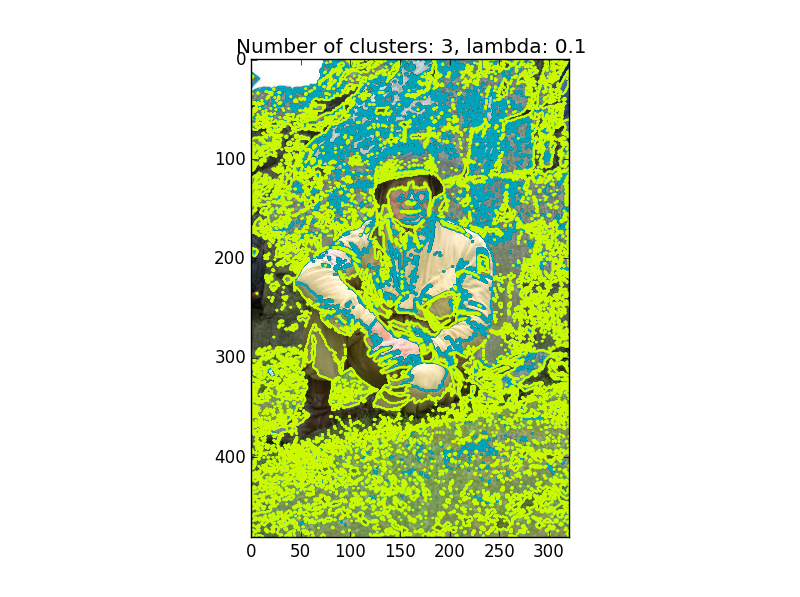


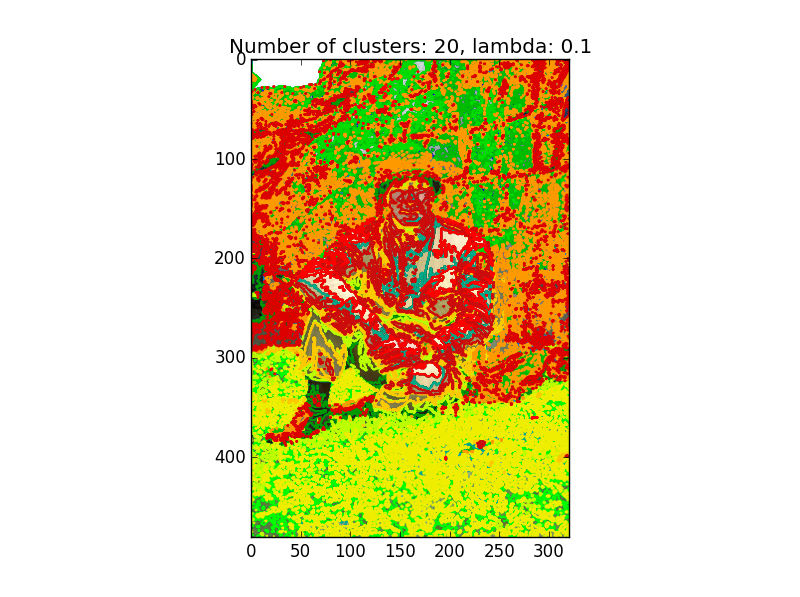


Более-менее приемлемым изображение становится, начиная с 20 кластеров (цветов). С увеличением количества цветов, что очевидно, качество изображения улучшается, что иллюстрирует пример с количеством кластеров, равным 153. Но все же искажение можно заметить.

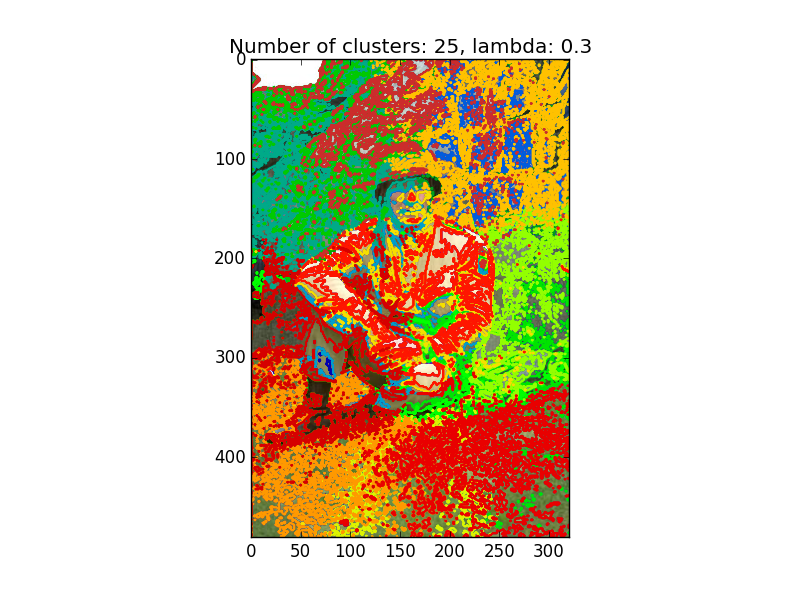
**ЗАДАНИЕ 3**

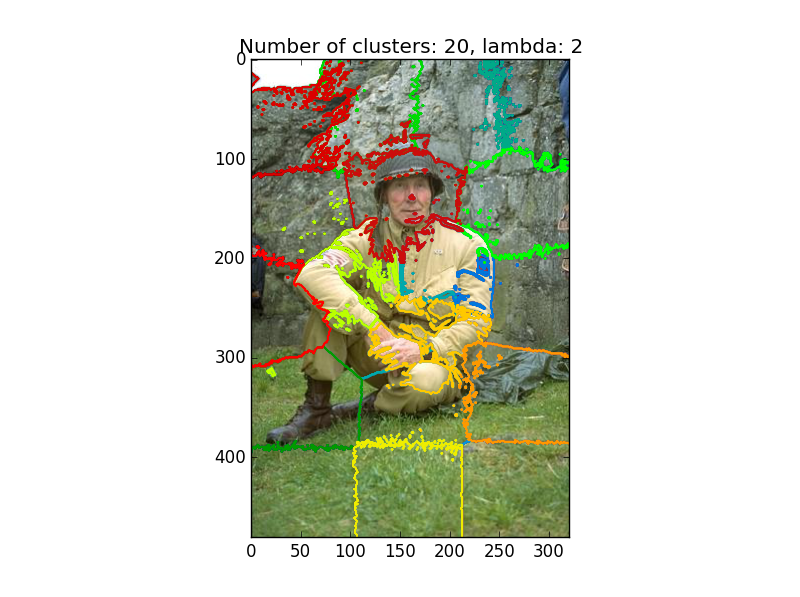
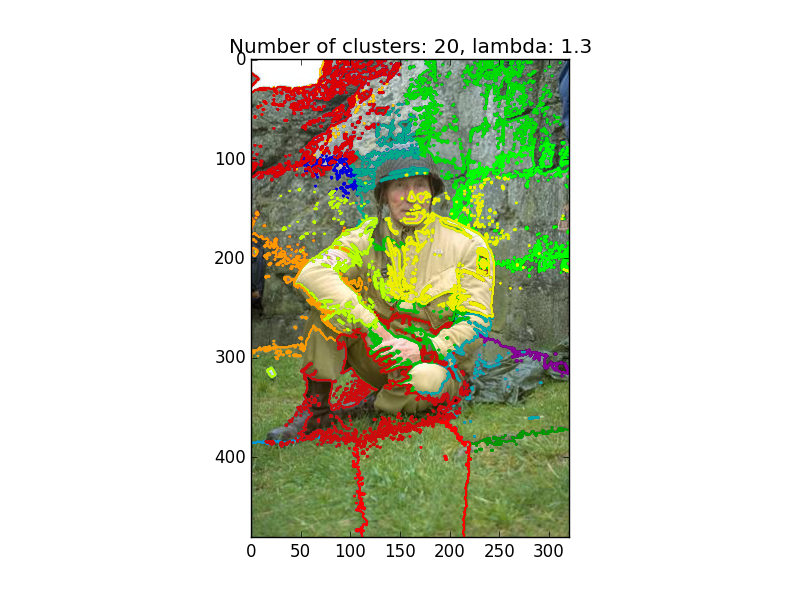
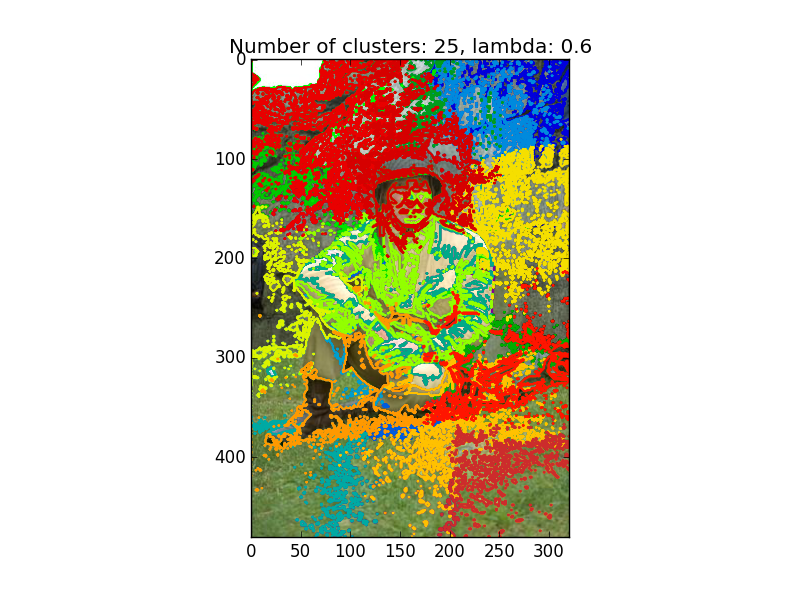
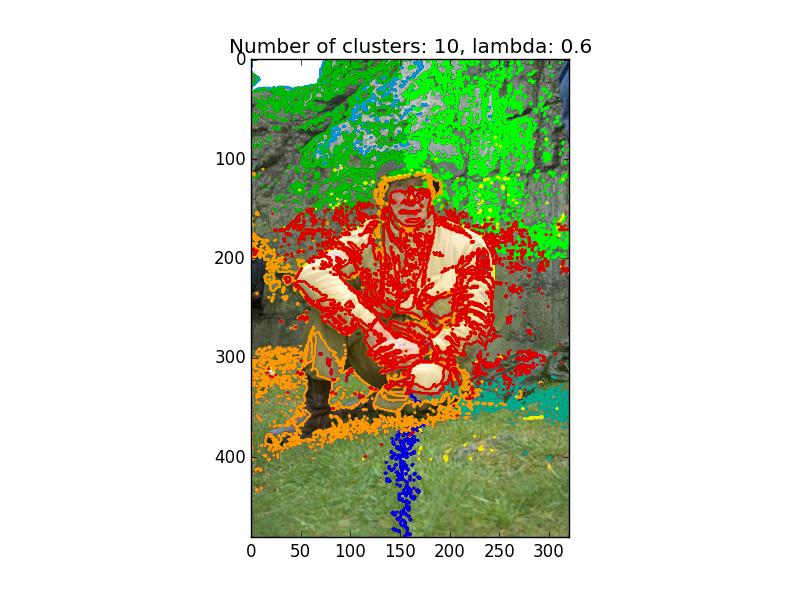
Были произведены и визуализированы сегментации (на основе кластеризации) с различными количествами кластеров и взятого параметра лямбда:

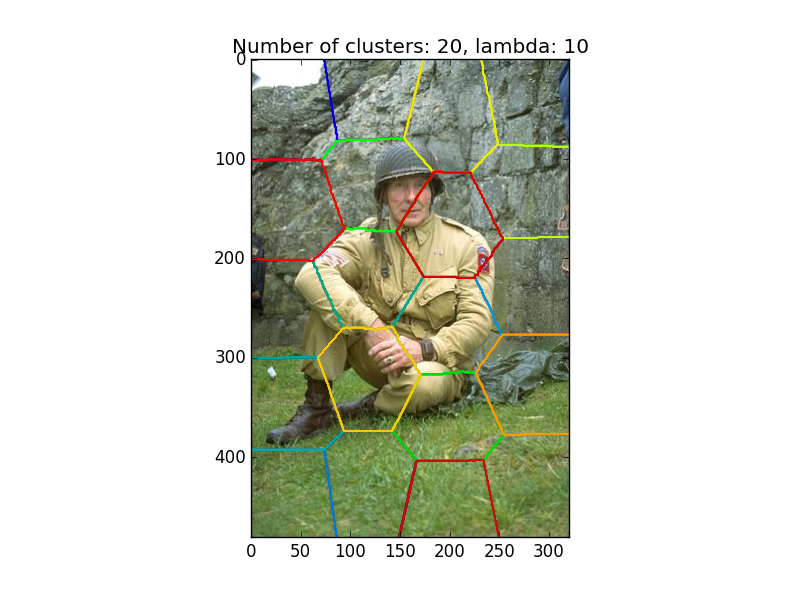


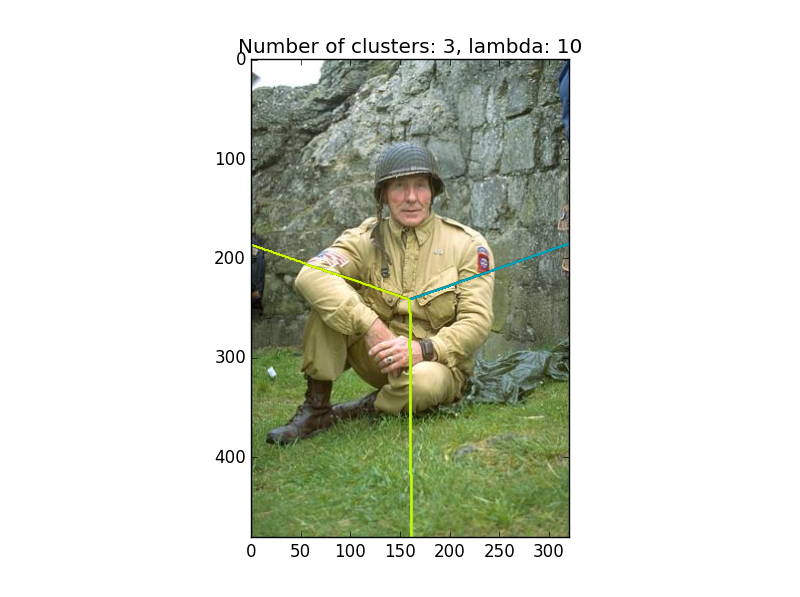








****

****

Количество кластеров, очевидно, влияет на количество выделенных сегментов. Что касается лямбда, при совсем малых значениях контур достаточно сложно воспринимать, а при высоких значениях контуры имеют вид диаграммы Вороного и малоинформативны на взгляд. Поэтому для наглядности и информативности лучше выбирать средние значения.

**СПИСОК ЛИТЕРАТУРЫ**

1. **Анализ данных (Программная инженерия)** –   
   http://wiki.cs.hse.ru/Анализ\_данных\_(Программная\_инженерия)

**ТЕКСТ ПРОГРАММЫ**

\_\_author\_\_ = 'Lev Osipov'  
  
**import** numpy **as** np  
**import** random  
**from** sklearn.datasets **import** load\_digits  
**from** sklearn.cluster **import** KMeans  
**from** sklearn.metrics **import** adjusted\_mutual\_info\_score **as** mutual\_score  
**from** sklearn.metrics **import** adjusted\_rand\_score **as** rand\_score  
**import** matplotlib.pyplot **as** plt  
**from** skimage.io **import** imread  
  
  
**def swap**(digs, i1, i2):  
 temp = digs.data[i1]  
 digs.data[i1] = digs.data[i2]  
 digs.data[i2] = temp  
  
 temp = digs.images[i1]  
 digs.images[i1] = digs.images[i2]  
 digs.images[i2] = temp  
  
 temp = digs.target[i1]  
 digs.target[i1] = digs.target[i2]  
 digs.target[i2] = temp  
  
  
**def shuffle\_digits**(digs):  
 indices = np.arange(len(digs))  
 random.shuffle(indices)  
 **for** i **in** xrange(len(indices)):  
 swap(digs, i, indices[i])  
  
  
**def errors\_analyzing**(digs, kmns, n\_cl):  
 # Array for defining connection between labels and clusters  
 cluster\_digits = np.zeros(n\_cl)  
 # Array with numbers of errors and most erroneous number  
 errors = np.zeros(shape=(n\_cl, 3))  
 # Array with error examples (for each cluster)  
 err\_examples = np.zeros(n\_cl)  
 # Defining cluster digits and counting errors  
 **for** i **in** xrange(n\_cl):  
 # Array for true labels  
 temp = np.zeros(n\_cl)  
 **for** j **in** xrange(len(kmns.labels\_)):  
 # Counting number of "votes"  
 **if** kmns.labels\_[j] == i:  
 temp[digs.target[j]] += 1  
 # Defining the digit  
 index = np.argmax(temp, axis=0)  
 cluster\_digits[i] = index  
 errors\_number = 0  
 # Counting errors  
 max\_errors = 0  
 max\_error\_ind = 0  
 **for** k **in** xrange(len(temp)):  
 **if** k != index:  
 errors\_number += temp[k]  
 **if** max\_errors < temp[k]:  
 max\_errors = temp[k]  
 max\_error\_ind = k  
 errors[i, 0] = errors\_number  
 errors[i, 1] = max\_errors  
 errors[i, 2] = max\_error\_ind  
 # Finding example  
 **for** b **in** xrange(len(kmns.labels\_)):  
 **if** kmns.labels\_[b] == float(i) **and** cluster\_digits[i] != digs.target[b]:  
 err\_examples[i] = b  
 **break  
 return** cluster\_digits, errors, err\_examples  
  
  
**def visualization**(digs, kmns, cl\_digs, errs, err\_exs):  
 # Showing cluster centres  
 **for** i **in** xrange(len(kmns.cluster\_centers\_)):  
 Z = kmns.cluster\_centers\_[i, :]  
 Z = Z.reshape([8, 8])  
 plt.title("Digit: " + str(int(cl\_digs[i])) + ", total number of errors: " + str(int(errs[i, 0])) +  
 ", most erroneous digit " + str(int(errs[i, 2])) + " (" +  
 str(int(errs[i, 1])) + " errors)")  
 plt.imshow(Z, cmap='Blues')  
 plt.show()  
  
 # Showing errors  
 **for** i **in** xrange(len(err\_exs)):  
 Z = digs.data[err\_exs[i]]  
 Z = Z.reshape([8, 8])  
 plt.title("Digit " + str(int(digs.target[err\_exs[i]])) +  
 " in cluster with digit " + str(int(cl\_digs[kmns.labels\_[err\_exs[i]]])))  
 plt.imshow(Z, cmap='Blues')  
 plt.show()  
  
  
**def task1**():  
  
 # Data loading  
 digits = load\_digits()  
 cluster\_numbers = 10  
  
 # Testing different methods of initialization in clustering  
 n\_tests = 25  
 **for** init\_type **in** ('k-means++', 'random'):  
 results1 = np.zeros(n\_tests)  
 results2 = np.zeros(n\_tests)  
 **for** i **in** xrange(n\_tests):  
 # Data shuffling  
 shuffle\_digits(digits)  
  
 kMeans = KMeans(n\_clusters=cluster\_numbers, init=init\_type)  
 kMeans.fit(digits.data)  
 predict = kMeans.predict(digits.data)  
 results1[i] = mutual\_score(digits.target, predict)  
 results2[i] = rand\_score(digits.target, predict)  
  
 plt.title("Initialization type: " + init\_type)  
 plt.plot(results1, label='AMI')  
 plt.plot(results2, label='RMI')  
 plt.ylim(0.6, 0.8)  
 plt.legend()  
 plt.show()  
  
 # Took k-means++  
 shuffle\_digits(digits)  
 kMeans = KMeans(n\_clusters=cluster\_numbers)  
 kMeans.fit(digits.data)  
  
 # Defining clusters and finding errors  
 cluster\_digits, errors, err\_examples = errors\_analyzing(digits, kMeans, cluster\_numbers)  
 visualization(digits, kMeans, cluster\_digits, errors, err\_examples)  
  
  
**def cluster\_parrots**(im, n\_cl, sh0, sh1, sh2):  
 # Clustering  
 kMeans = KMeans(n\_clusters=n\_cl)  
 kMeans.fit(im)  
 new\_image = np.zeros(shape=(sh0, sh1, sh2))  
 # Filling new image  
 **for** i **in** xrange(len(kMeans.labels\_)):  
 ind1 = i / sh1  
 ind2 = i % sh1  
 new\_image[ind1, ind2] = kMeans.cluster\_centers\_[kMeans.labels\_[i]]  
  
 # Showing  
 plt.title("Number of colors (clusters): " + str(n\_cl))  
 plt.imshow(new\_image)  
 plt.show()  
  
  
**def task2**():  
 # Data reading  
 image = imread("parrots.jpg")  
 image = np.array(image, dtype=np.float64) / 255  
 shape0 = image.shape[0]  
 shape1 = image.shape[1]  
 shape2 = image.shape[2]  
 image = image.reshape(shape0 \* shape1, shape2)  
 # Clustering tests  
 cluster\_parrots(image, 153, shape0, shape1, shape2)  
 cluster\_parrots(image, 3, shape0, shape1, shape2)  
 cluster\_parrots(image, 10, shape0, shape1, shape2)  
 cluster\_parrots(image, 20, shape0, shape1, shape2)  
 cluster\_parrots(image, 25, shape0, shape1, shape2)  
 cluster\_parrots(image, 30, shape0, shape1, shape2)  
 cluster\_parrots(image, 50, shape0, shape1, shape2)  
  
  
**def lambda\_injection**(im, sh0, sh1, sh2, lam):  
 data = np.zeros(shape=(sh0, sh1, sh2 + 2))  
 **for** i **in** xrange(sh0):  
 **for** j **in** xrange(sh1):  
 **for** k **in** xrange(sh2):  
 data[i][j][k] = im[i][j][k]  
 data[i][j][sh2] = lam \* i  
 data[i][j][sh2 + 1] = lam \* j  
 data = data.reshape(sh0 \* sh1, sh2 + 2)  
 **return** data  
  
  
**def cluster\_segments**(im, lam, n\_cl):  
 shape0 = im.shape[0]  
 shape1 = im.shape[1]  
 shape2 = im.shape[2]  
 # Injecting lambdas  
 data = lambda\_injection(im, shape0, shape1, shape2, lam)  
 # Clustering  
 kMeans = KMeans(n\_clusters=n\_cl)  
 kMeans.fit(data)  
 # Showing  
 plt.title("Number of clusters: " + str(n\_cl) + ", lambda: " + str(lam))  
 plt.imshow(im)  
 labels = kMeans.labels\_  
 labels = labels.reshape(shape0, shape1)  
 **for** i **in** xrange(n\_cl):  
 plt.contour(labels == i, contours=1, colors=[plt.cm.spectral(i / float(n\_cl)), ])  
 plt.show()  
  
  
**def task3**():  
 # Data reading  
 image = imread("grass.jpg")  
 # Clustering segments  
 cluster\_segments(image, 0.1, 3)  
 cluster\_segments(image, 0.1, 20)  
 cluster\_segments(image, 0.3, 10)  
 cluster\_segments(image, 0.3, 25)  
 cluster\_segments(image, 0.6, 10)  
 cluster\_segments(image, 0.6, 25)  
 cluster\_segments(image, 1.3, 20)  
 cluster\_segments(image, 2, 20)  
 cluster\_segments(image, 10, 20)  
 cluster\_segments(image, 10, 3)  
  
  
task1()  
task2()  
task3()