Assignment 5

Computational Intelligence, SS2017

Team Members		
Last name	First name	Matriculation Number
Alié	Félix	01644819
Sourmpis	Christos	01644560

Computational Intelligence - 05

Expectation Maximization and k-means Algorithm

Félix Alié, Christos Sourmpis

17.06.2017

Contents

Introduction

The objective of this assignment is to implement the algorithms of expectation maximization using the Gaussian Mixture Model and the K-means algorithm.

1 Expectation Maximization

1.1 Question 1

The first part is implemented in the code. The threshold comes up after some experimentation from question 2. We have make a small addition in the algorithm we have added a regularization option by adding a constant diagonal matrix of the order of 50, we chose this number after some trials and take into consideration the size of the rest points.

1.2 Question 2

After experimenting with different seeds and so setting different initial values for the means and the sigmas of the models we have the following results:

Figure 1: Scatter Plot for seed 233151758

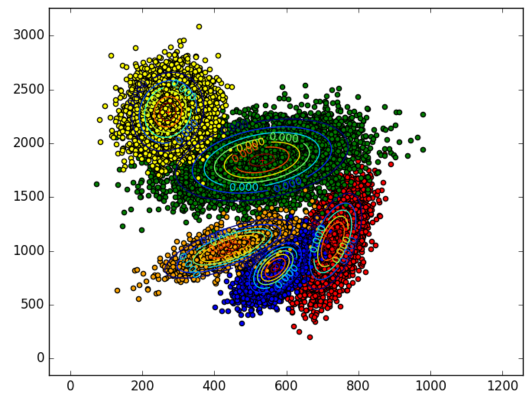


Figure 2: Scatter Plot for seed 1

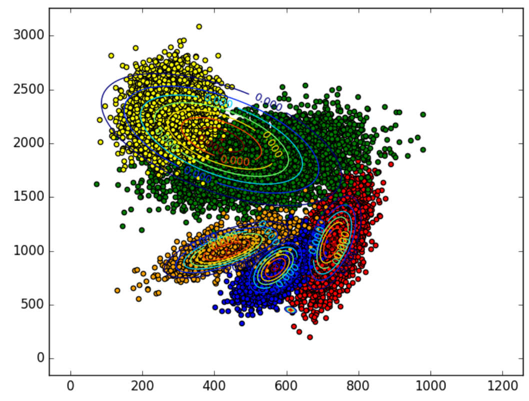


Figure 3: Scatter Plot for seed 233151758 with higher convergence

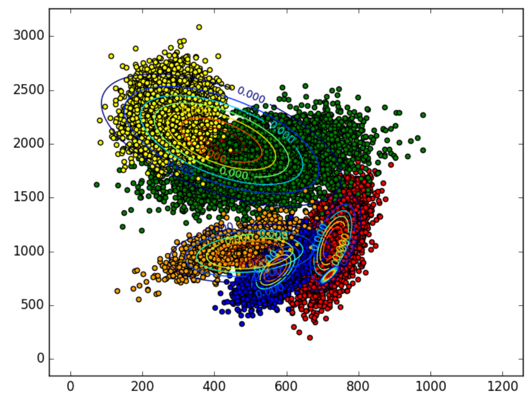


Figure 4: Scatter Plot for seed 233151758 with regularization

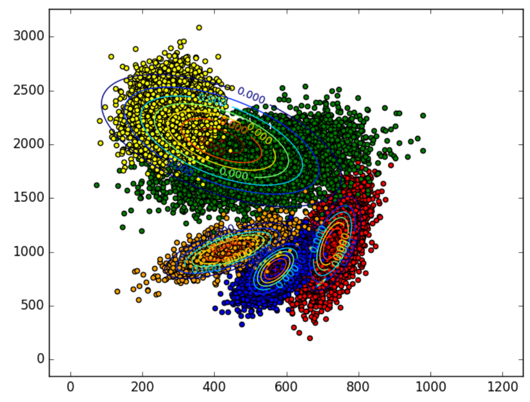


Figure 5: Scatter Plot for seed 233151758 with higher convergence and regularization

In the first case we see that three components have find exactly the right spot and classify the vowels correctly, the other two both are classifying wrong, one of them is modeling two vowels and the other is stuck in an irregular position, it is interesting that this point if we leave the algorithm to run, at some point it get stuck to a place and diminishes. Also we can see that with the addition of the regularization the final point that this model gets stuck is different. Finally, running the algorithm with seed number 1 we get the optimal results, meaning all the vowels are well classified.

1.3 Question 3

In this question we study the effect of the number of the models to the sanity of our results. Concerning the initialization, we initialize the means and the covariances matrices randomly as we did before. Furthermore, we initialize the alpha parameter with equally weights for every model, i.e. $\alpha_m = \frac{1}{M}$. In order to see better the effect of the number of models we keep the same seeds.

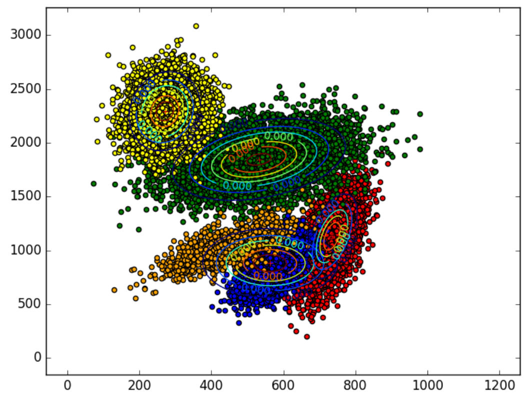


Figure 6: Scatter Plot for seed 1 with M = 4

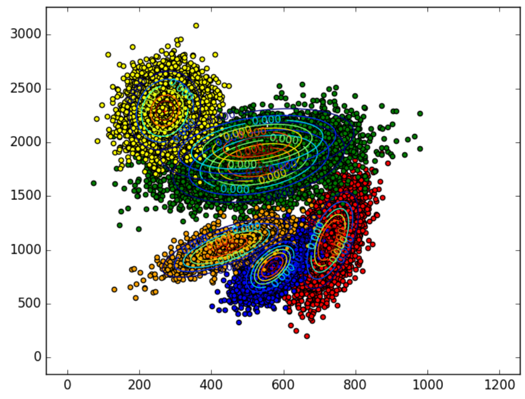


Figure 7: Scatter Plot for seed 1 with M = 6

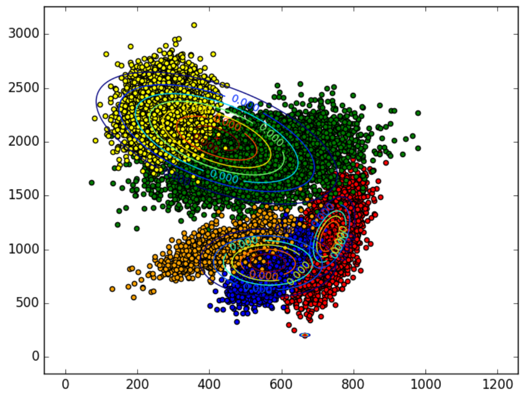


Figure 8: Scatter Plot for seed 233151758 with M = 4 and regularization

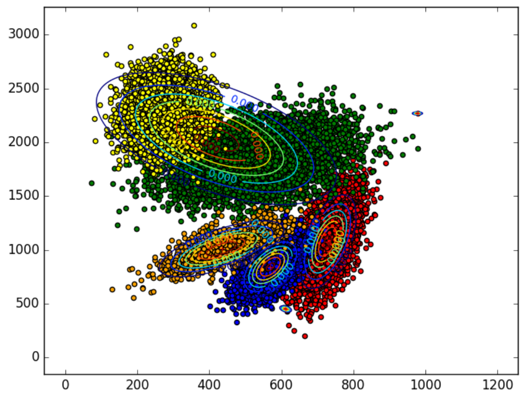


Figure 9: Scatter Plot for seed 233151758 with M = 4 and and regularization

At both cases we don't investigate the result of the order of the convergence and in the case of the seed 233151758 we use regularization, because other ways the algorithm doesn't converge. When we use the first seed with M = 4 we see that the algorithm works exactly as before but this time two classes are decribed by one model and when M = 6 one class is described by two models.
Concerning the seed 233151758, at both cases M = 4 and M = 6 we see that one point sticks in the wrong place and diminishes, especially for M = 6 there is an extra model that diminishes, and the rest of the models are trying to describe the rest classes the best they can. Especially for the case of M = 6 we get the exact the same results as M = 5, with the only difference that there is an extra model that is stuck to a point.
In case that we didn't have knowledge of the number of models that we should use, we think that the best solution would be to find a good seed, i.e. not diminishing to point models, and gradually increase the number of models till we see that two models are overlapping each other.

1.4 Question 4

At this question we investigate the evolution of log likelihood over the number of iterations. We use as examples the experiments with seeds = 1, 233151758 M = 5 and without regularization. At both cases we see monotonically increment of the function, as we expected. It is quite interesting to see how the algorithm stuck in some local minima, but after some iterations it get unstuck till found its final position.

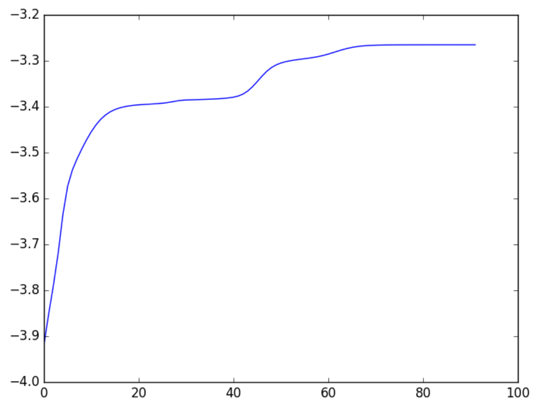


Figure 10: Log-likelihood function vs iteration number for seed 1 with M = 5

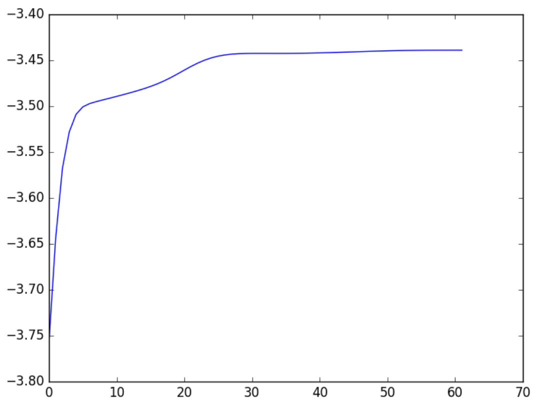


Figure 11: Log-likelihood function vs iteration number for seed 233151758 with M = 5

1.5 Question 5

Because in the previous examples we used diagonal matrices for the initialization of our algorithm, in order to show how effective this is we are going to do the opposite and run some examples with not diagonal matrices for the covariances matrices. Especially, we make random matrices but with higher values at the diagonal positions so that the $Sigma_0$ matrix to be positive semidefined. After running the examples from the previous questions we see that the performance of the algorithm has fell a lot especially for the seed number 1, which if we don't activate the regularization then converges too fast to the wrong place.

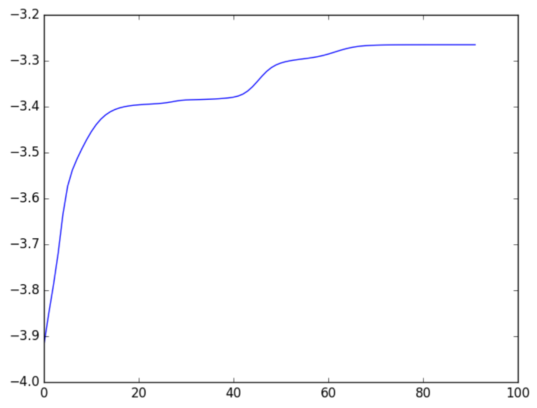


Figure 12: Log-likelihood function vs iteration number for seed 1 with M = 5

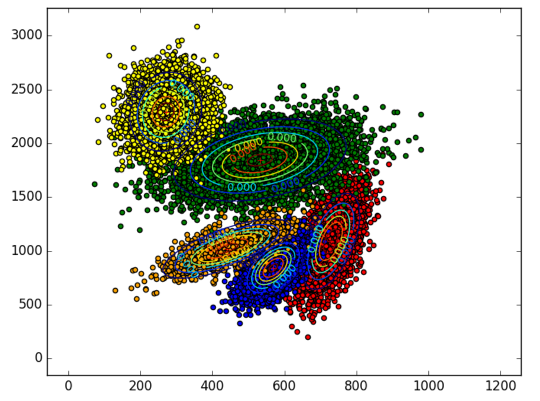


Figure 13: Scatter Plot for seed 1

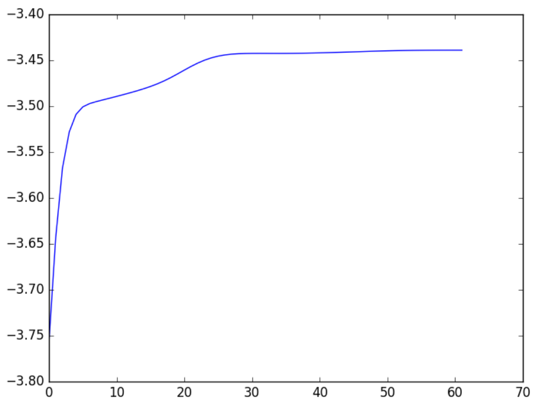


Figure 14: Log-likelihood function vs iteration number for seed 233151758 with M = 5

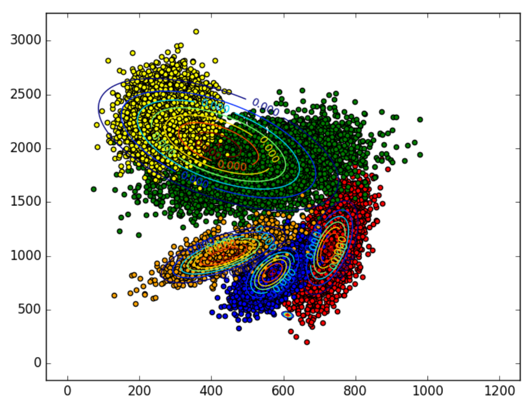


Figure 15: Scatter Plot for seed 233151758 with higher convergence

At this point would be important to mention that the results are heavily dependent by stochastic phenomena so slightly changes at the initialization could change drastically the result. But at most cases the initialization with diagonal identical matrices improves the performance of our algorithm.

1.6 Question 6

Using as examples the same experiments as question 4 after the soft-classification we get as results:

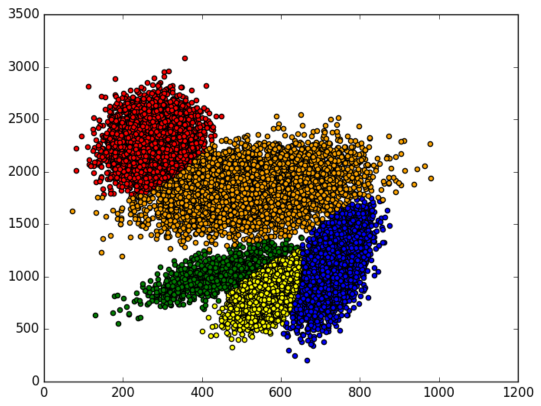


Figure 16: Scatter plot for the soft classified points from seed 1 with M = 5

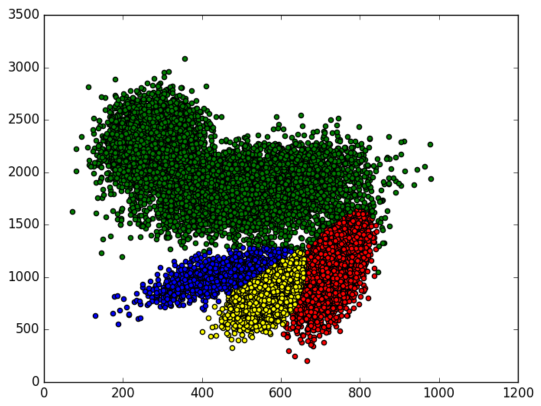


Figure 17: Scatter plot for the soft classified points from seed 233151758 with M = 5

We see that actually at both cases our data are very well classified and identical to the places that the Gaussian contour plots were defining.

2 K-means algorithm

We implement the k-mean algorithm and make it run with the same random seed for the generation of μ as the EM algorithm. We then plot the cumulative distance and the computed clusters:

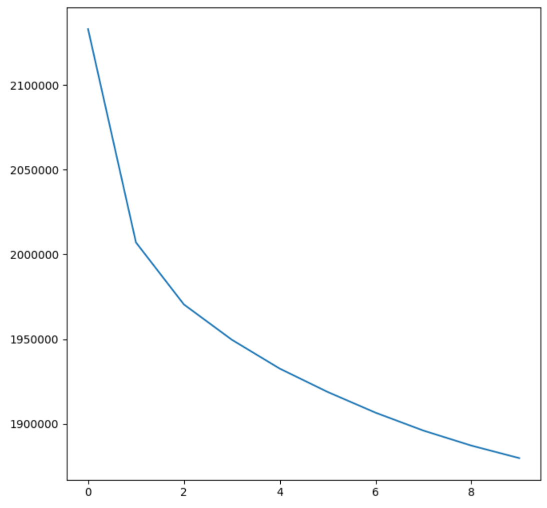
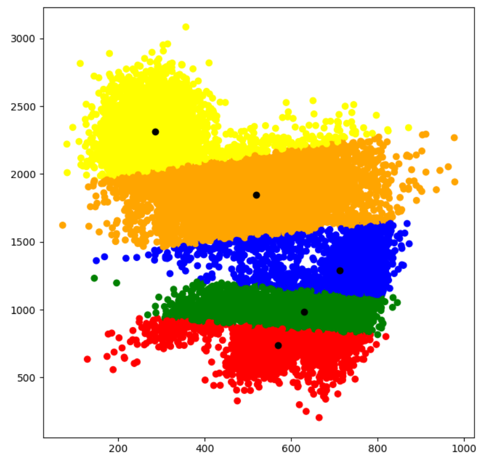
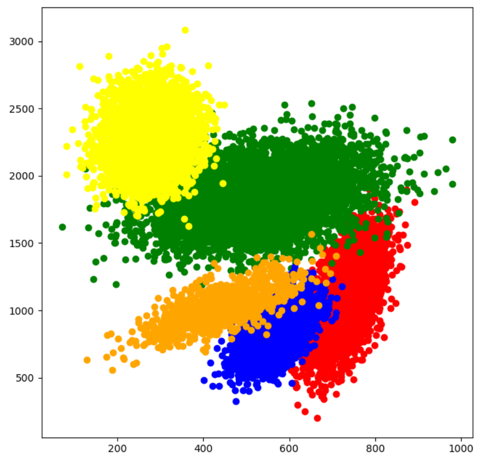


Figure 18: Cumulative distance over iterations

As we can see, the cumulative distance (which means the sum of the distances between each point and it's closest center) is reducing over time, which is normal, since we're trying to find the best centers in order to minimize this distance.



(a) Resulting clusters



(b) True Clusters

Figure 19: Kmeans clusters vs true clusters

Note: Each black point is the center of a cluster.

We then obtain these clusters. As we can see, they are not really accurate. All the clusters are polygons, separating the clusters with straight lines, although they are supposed to be ellipsoids, like the ones we can find with a smooth classification like EM(1.6). This can be explained by the fact that kmeans divide the space in k strict zones, like on the example bellow (fictionnal example):

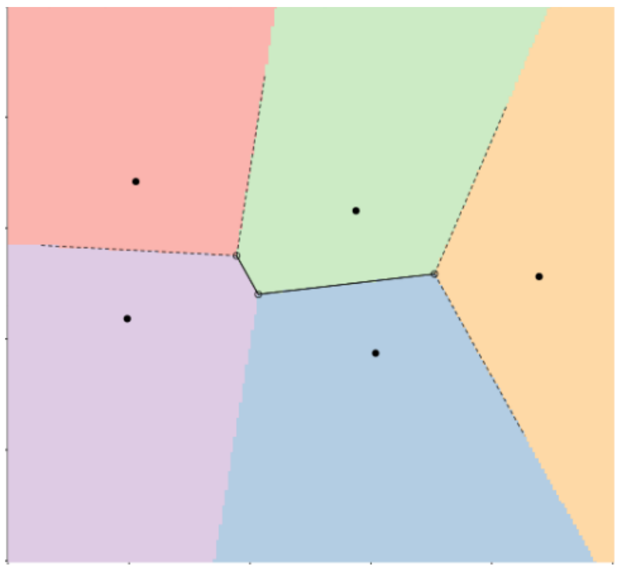
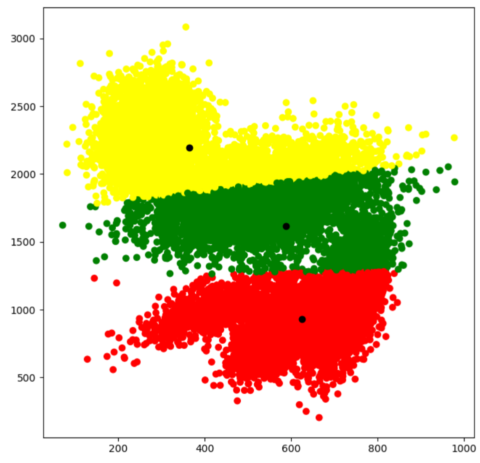


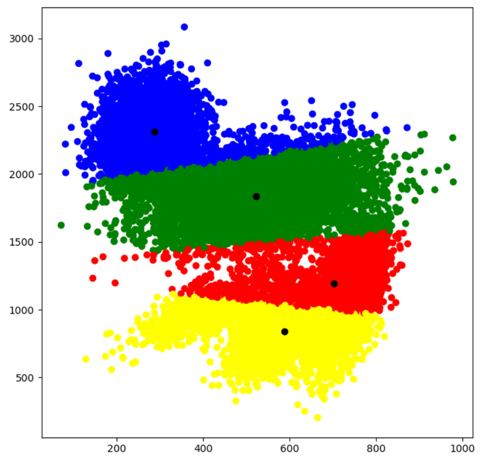
Figure 20: Resulting clusters

That's why when clusters are overlapping, the k-means algorithm performs really poorly and create that kind of trapezoidal clusters. Actually, the clusters that are not overlapping too much (yellow and green on figure 19b, are almost well computed by the kmeans. The real problem is the three others.

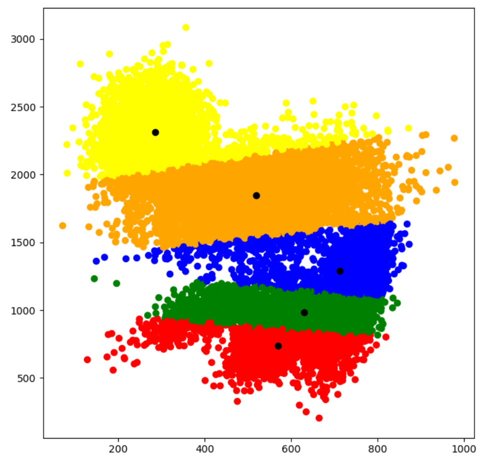
As asked we can try with a different number of clusters as well:



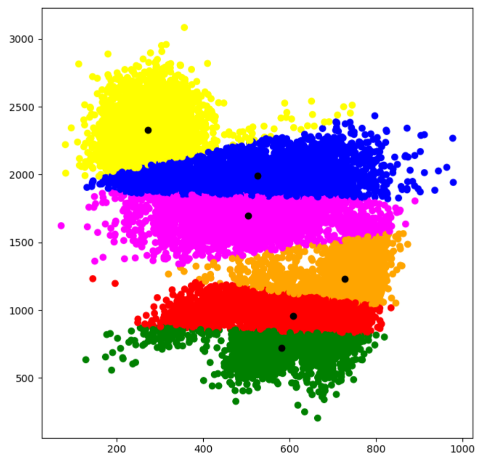
(a) $k = 3$



(b) $k = 4$



(c) $k = 5$



(d) $k = 6$

Figure 21: Different clusters

3 Combining the two methods

Initializing randomly first the mu_0 of K-means and then taking the result of K-means, i.e. the mean values, and putting them as initial condition for Expectation Maximization we see that almost every time we take perfect results(plots like the ones from seed number 1).