Project 1: kmeans and EM Algorithms

Sam Beckmann

February 22, 2017

1 Introduction

Both the kmeans and EM algorithms are intended for unsupervised clustering, that is, assigning n-dimensional points to k clusters, with no information about the points other than their coordinates. Both algorithms work in iterative steps, at each step assigning points to clusters then updating the cluster parameters based on those assignments. These steps are repeated until convergence has been reached.

The difference between the two algorithms relies primarily on how points are classified. The kmeans algorithm uses discrete classifications, that is, each point is assigned to only one cluster. EM, on the other hand, returns a weight vector, or probability distribution, for each point at each step. For the sake of this project, I consider the final assignment of a point to be the cluster with the maximum weight for the point once the algorithm has reached convergence.

2 Testing Procedure

Direct comparison between the two algorithms was done using 19 datasets divided among 4 distinct test cases. Each test case was run 10 times, with results shown being the average among the individual trials. The main metric used for evaluating the algorithms was pairwise accuracy. That is, the probability that a pair of points that were placed into the same category by the algorithm were generated by the same cluster in the dataset. This method has the unique property that the value remains largely independent of the the number of clusters the algorithm divides the dataset in, as it only considers pairs which are clustered together by the algorithm.

The first test case varies the number of distributions and the spacing between them, while the second and third cases only vary the standard deviation of the distributions, with different more spacing between distributions in the second case than in the third. Finally, the fourth test case varied the number of points while keeping all other parameters the same, to see how the algorithms reacted with more or less data.

In addition to the univariate datasets, I also performed some real-world examples with image compression and the kmeans algorithm. The results of these experiments are reviewed in section 3.2.

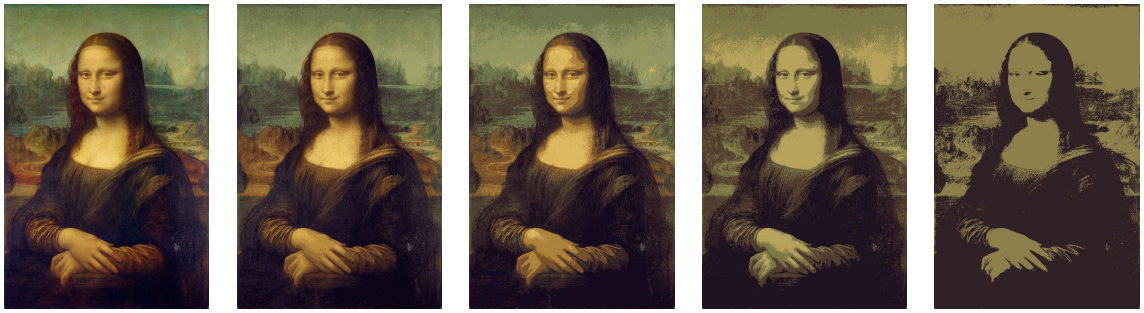
3 Results

3.1 Univariate Datasets

For the first test case, there were 1000 points per source and each source has a standard deviation of 1. The number of sources was pulled from the set $\{3, 5, 10\}$, and the the spacing between them pulled from the set $\{\frac{1}{2}\sigma, \sigma, \frac{3}{2}\sigma, 2\sigma\}$. For each of these 12 possibilities, 4 tests were run, varying the number of clusters used in the algorithms from the set $\{2, 3, 6, 8\}$.

In general, the algorithms performed worse when more sources were present, and better with higher numbers of clusters. Also expectedly, both algorithms improved when the spacing between clusters was higher, achieving pairwise accuracy as high as 0.66 in some cases.

The EM algorithm did consistently worse than the kmeans algorithm, and this effect was exaggerated with a high number of sources. I found the EM algorithm to consistently place means close to the center of the entire distribution of points. I do not know if this behavior is the result



(a) Uncompressed (b) 16 colors (c) 8 colors (d) 4 colors (e) 2 colors

Figure 1: Image compression with kmeans

of the algorithm, or a bug in my code, but I have have found nothing that indicates the algorithm is not performing as intended.

In the second test case, kmeans did much better clustering points with a lower standard deviation. It had an accuracy of 0.79 for the 1σ dataset, which dropped to 0.35 for 3σ. EM did much worse, performing with an accuracy of 0.20 regardless of the standard deviation.

When the sigmas where sampled randomly, as was done in the third test case, kmeans did poorly, with an accuracy of only 0.35, though it beat EM, which scored 0.20 again.

For the fourth and final test case, varying the number of data points has seemingly no effect on the accuracy of kmeans, but improved the accuracy of EM. The kmeans algorithm was still more effective at clustering than the best-performing EM, however.

3.2 Real World Example: Image Compression

In addition to the generated Gaussian data, I tested the kmeans algorithm on image compression. Image clustering can be treated as a 3-variable compression problem, where each pixel represents a data point, and a cluster center as a color that all pixels assigned to that color will be reduced to. As assigning points to n colors requires $\log_2 n$ bits, and a standard 8-bit rgb image has 16 bits per pixel, compressing an image to 16 colors will result in the image using approximately ⅛ the space of the uncompressed version.[1]

The results of using kmeans on the image of the Mona Lisa are shown in Figure 1. With 16 colors, kmeans is quite effective at retaining accuracy. Slight artifacts are visible on the hands, and some details of cracks in the paint have been lost, but the overall image still highly resembles the original, despite being only ⅛ in size. With 8 colors, significant artifacting is seen. With only 4 colors, all pretense of retaining color is gone, only monochromatic shades remain. Finally, the 2-color compression lacks most detail, only providing the vaguest outline of the original image.

4 Conclusion

To summarize, kmeans performed remarkably well at clustering points effectively, as seen both quantitatively in its pairwise accuracy scores for the univariate data, and qualitatively with the image compression. EM, however, performed much worse, often clustering its means close to each other. I am uncertain at this time if these results are the product of a bug I have not been able to root out, or the process of the algorithm itself. On trivially-sized data sets, EM was able to correctly classify the points without any difficulty.

A Code

All of my code is available at https://github.com/samvbeckmann/machine-learning/tree/master/clustering. The implementations of both algorithms are written in python.

[1] Of course, the colors need be stored too, taking 24 bits per color, but we can consider this trivial when compared to the space of the image.