Machine Learning E16 - Handin 4
Representative-based Clustering Algorithms

Group 22
Mark Medum Bundgaard, Morten Jensen, Martin Sand Nielsen

December 19, 2016, Aarhus University

K-Means clustering vs Gaussian Mixture-model EM

During the assignment we have implemented the K-Means and GMM-EM, in order to part the data into clusters. The following plot show the result of a run with both the EM and the K-means algorithm. The main difference between k-means and GMM-EM, is the fact that K-Means work only with hard distances, and GMM work with Gaussian distributions. The effect of this, is that in K-Means, you are always associated with whatever mean is closest to you. So all points are clustered by small "spheres" around the centers found via K-Means. That means outliers are often placed in the wrong clusters. On the other hand, the clusters from the GMM-EM algorithm doesnt have to be convex. Because of that, GMM-EM is better at clustering outliers. There is also the point that GMM-EM gives probabilities, and therefore you could use the percentages to see what the second most probable cluster would be and such. In 4 above we see the actual classification of the data, where we have chosen two dimensions. Here in 5 we see three different runs of the k-means algorithm. We see how different the results are. This is because k-means is relient on which starting conditions you initiate it with. What points it chooses when starting, for the first centers. In 6 we see three different runs of the em algorithm. Again we see the results differ quite a bit. We even misclassify some datapoint in the blue group, which would not be possible with the kmeans algorithm, since the blue points are fairly isolated. This also reflects the weakness that the em algorithm is dependent on a good choice of starting means.

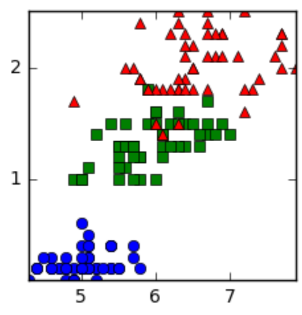

Figure 1: *Original classification of datapoints.*

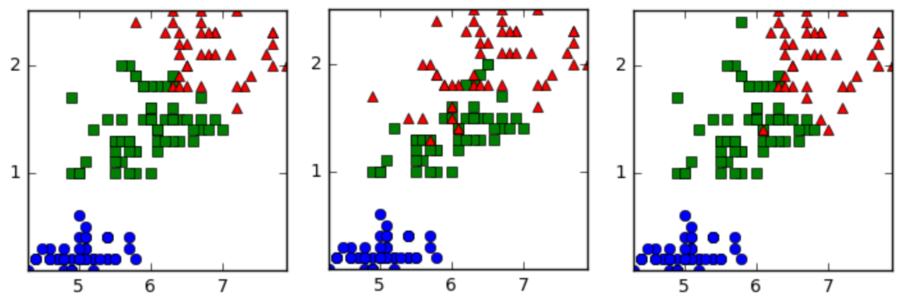

Figure 2: *K-means clustering of datapoints.*

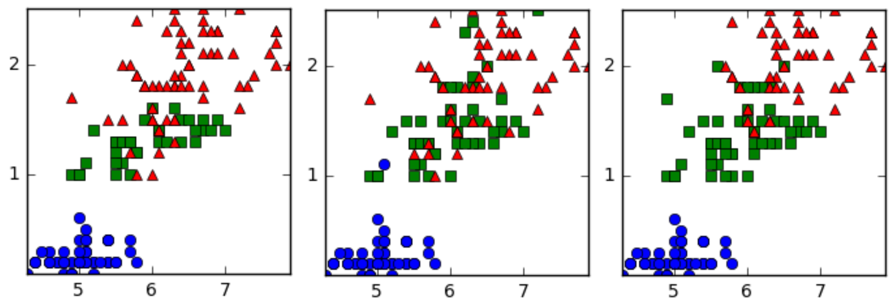

Figure 3: *GMM-EM clustering of datapoints.*

Status of work

Both K-means and Gaussian Mixture Expectation Maximization has been implemented in Python. Some effort has been done into improving performance, but can certainly be improved further. To test to algorithms, they have been used on the SKLearn Iris data set of four-dimensional data point with three labels. Iterations of clustering for several number of parameter k has been performed with both from random initialization, and GM-EM again but initialized from a K-means instead.

The two clustering measures, F_1-score and Silhouette, has been implemented and used for evaluating the clustering algorithms and optimizing the k parameter.

Finally the implemented clustering has been used to perform color depth compression of two photographs.

K-Means clustering vs Gaussian Mixture-model EM

During the assignment we have implemented the K-Means and GMM-EM, in order to part the data into clusters. The following plot show the result of a run with both the EM and the K-means algorithm. The main difference between k-means and GMM-EM, is the fact that K-Means work only with hard distances, and GMM work with Gaussian distributions. The effect of this, is that in K-Means, you are always associated with whatever mean is closest to you. So all points are clustered by small "spheres" around the centers found via K-Means. That means outliers are often placed in the wrong clusters. On the other hand, the clusters from the GMM-EM algorithm doesnt have to be convex. Because of that, GMM-EM is better at clustering outliers. There is also the point that GMM-EM gives probabilities, and therefore you could use the percentages to see what the second most probable cluster would be and such.

In 4 above we see the actual classification of the data, where we have chosen two dimensions. Here in 5 we see three different runs of the k-means algorithm. We see how different the results are. This is because k-means is relient on which starting conditions you initiate it with. What points it chooses when starting, for the first centers.

In 6 we see three different runs of the em algorithm. Again we see the results differ quite a bit. We even misclassify some datapoint in the blue group, which would not be possible with the kmeans algorithm, since the blue points are fairly isolated. This also reflects the weakness that the em algorithm is dependent on a good choice of starting means.

Silhouette measures & F1 scores

To evaluate clustering performance, some sort of measure is useful. The F_1 score is based on simply evaluating the ratios of correctly clustered data points. The F_1 score is defined as,

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where precision is the ratio of selected elements which are relevant, while recall is the ratio of relevant elements being selected. The F_1 score is thus bound to the interval between 0 and 1, where 1 denotes a perfect clustering.

F_1 scores for various clusterings performed with both K-means and EM-GM and a combination, where K-means is used to initialize GM-EM centers, can be seen in Figure 7 and 9. To illustrate the variating results for some for some configurations all clustering has been performed 20times and averaged. The standard deviation is marked for each such average. The clusters are assigned labels by voting of its assigned data points (supervised).

The Silhouette is another clustering performance measure, which relies on a measure of similarity of points clustered. This allows for some evaluation of unsupervised clusterings. For Silhouette a similarity measure between two data points has to be chosen. For this exercise a measure of Euclidean distance has been used as such. Silhouette is calculated for each data point, and can then be averaged for each cluster, which again can be averaged to evaluate the clustering. For the i'th data point,

$$\text{silhouette}(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}},$$

where the dissimilarity $a(i)$ is the average distance to all other data points in its cluster, and the dissimilarity $b(i)$ is the average distance to all data points in its nearest neighbouring cluster. This is fairly many computations when working with large data sets. The measure is bounded to the interval between -1 and 1. A plot of Silhouette measures for various number of clusters k, can

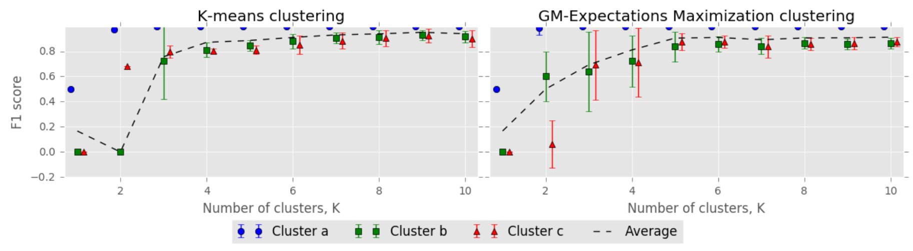

Figure 4:

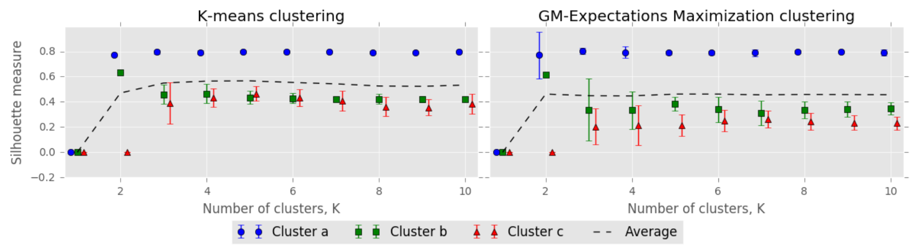

Figure 5:

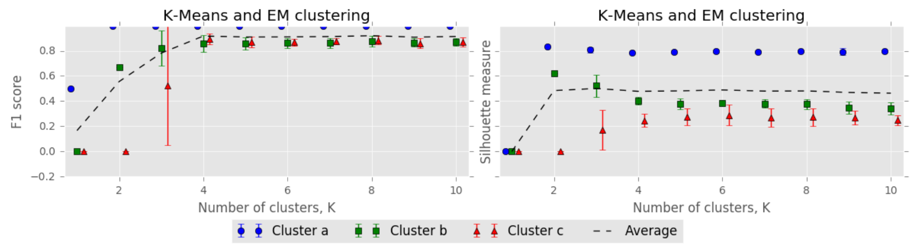

Figure 6:

be seen in Figure 8. Naturally only positive silhouette measures is found for our clusters, since both clustering methods use distance for descissions.

Both measures indicate that $k = 4$ is the most suitable for the data set. Here the average measures is high, and the results stable (low deviation between iterations). So it can be concluded that one of the three labels seems to be best described by two subclusters(when using K-means), or sum of two Gaussian distributions (when using GM-EM). Naturally only bad results occur with $k < 3 =$ number of labels.

Image color compression with K-means

The individual pixels in an normal RGB image holds information about the intensity of the three color channels. The color depth is given by the possible color combinations, which is usually 24 bit; 8 bit for each color channel. In a normal picture far from all possibilities af colors are present, so much of the color depth is unused. By limiting the number of possible colors to a small

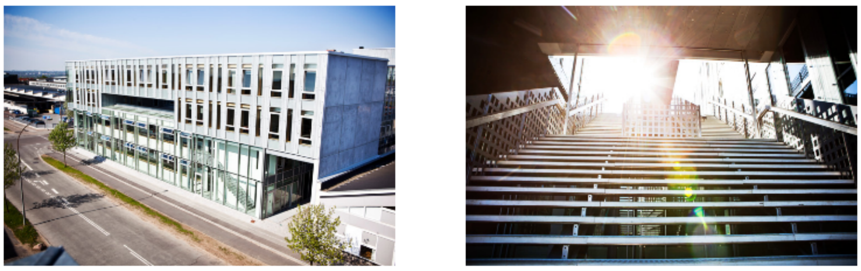

Figure 7: *Uncompressed* nygaard_facade.jpg *and* nygaard_stairs.jpg *images.*

sub set of the full color depth, and thus let each pixel refer to a color in this color set, disk usage can be decreased significantly. The color compression would then be a algorithm deciding a color set and transform each pixel as to refer these. Likewise a decompression algorithm would then do the reverse transformation by looking up each pixels reference color. So the space savings comes at two additional computational costs.

The space savings would be closely related to the size of a color set. And how few bits required for each pixel to index the color map. To further reduce the size, certain very closely related colors, could be said to yield same color, since small color variations can be almost impossible to notice to the human eye. This would add a quality loss to the compression, but often at a great reduction in color space. To create a color set, where similar colors are mapped together, K-means and GM-EM has been tried on the two test images shown in Figure 10.

A reasonable depth would be some 2^n, where n is the number of bits necessary for each pixel to reference the color map. In Figure 11 images are compressed to $2^4 = 16$ colors using both K-means and GM-EM. Likewise in Figure 12 images are compressed to $2^8 = 256$ colors using both K-means and GM-EM. The space savings relative to a raw 24bit bitmap approximate

$$\text{saving ratio} \approx \frac{3 \cdot 8\,\text{bit} - n\text{bit}}{3 \cdot 8\,\text{bit}},$$

for large images, where the additional space needed for the color map is negligible. For the 16 and 256 color maps, the savings are 83% and 67%.

Image quality loss acceptance limit will always be subject to personal taste and dependent on purpose. Compression of images can be done in various ways, and is done most image formats used for photographs. Color depth compression with ie. K-means to determine optimal color map of a given size, is very computational demanding, and probably not very useful. A fixed colormap could decrease image quality, depending on how well the pixel colors matches the color map.

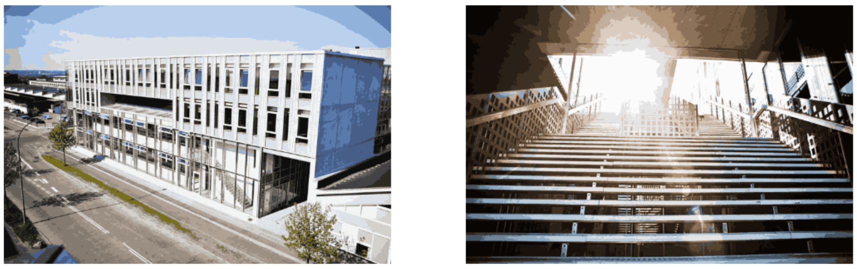

Figure 8: *Images with color depth compressed to a set of 16 colors (4bit) with the K-means algorithm.*

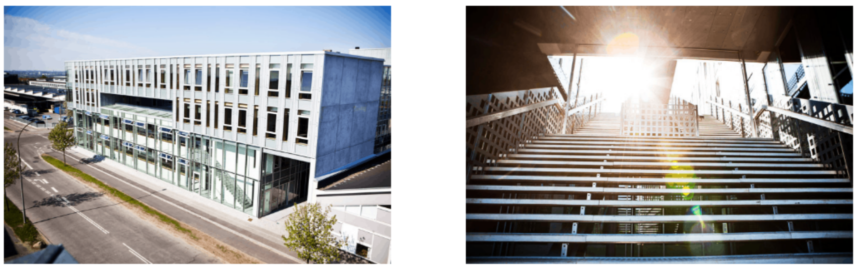

Figure 9: *Images with color depth compressed to a set of 256 colors (8bit) with the K-means algorithm.*