# Machine learning — Coursework 2

Michael Redman, CID: 00826863

March 2017

## Question 1

Gaussian mixture models and the K-means algorithm are both examples of clustering. Given an array of pixels each represented by a point in 3-dimensional colour space we can obtain a segregation of the image by assigning each pixel to one of k clusters — reducing the dimensionality of the data by representing each pixel by a label instead of a colour in $\mathbb{R}^3$.

In both methods this is done by assigning the points to clusters which are represented by a centroid (and in the Gaussian case a covariance matrix) based on which of these centroid is—in some sense—closest to the point to be classified. In the case of K-means this is typically defined as the euclidean distance to the centroid and in Gaussian mixture models by the cluster which, weighted by the estimated mixture proportion of the cluster, maximises the likelihood of the point.

The K-means algorithm simply consists of iteratively assigning each point to the closest cluster mean and then calculating the new mean of each cluster. However, ommited from this procedure, is what is our starting assignment of the points to the clusters? This is a matter of some study, but in this project we simply randomly assign the points to each cluster at the start. The class *KMeans* implements this K-means procedure for a given set of points and number of clusters.

In Gaussian mixture models we need a way of fitting our model to the data. Here we used the EM-algorithm, which consists of calculating the values of the parameters which maximise the conditional expectation of the log-likelihood where we condition on the previous values of our parameters. For a simple Gaussian mixture model this log-likelihood is linear in our parameters so we are able to find the optima seperately for each parameter. So, after assigning initial labels to each point, we begin by finding the conditional memebership probabilities of each point[1],

$$T_{j,i}^{(t)} = P(Z_i = j | X_i = x_i, \theta^{(t)}) = \frac{\tau_j^(t) f(x_i|\mu_j^{(t)}, \Sigma_j^{(t)})}{\sum_{l=1}^{k} \tau_l^(t) f(x_i|\mu_l^{(t)}, \Sigma_l^{(t)})} \tag{1}$$

where $Z_i$ is label which represent which cluster the $i$th point belongs to and $\tau_j$ is the proportion of points which belong to the $j$th cluster. Note that this is the probability which we will take expectations with respect to. Skipping over the calculation of this expectation, we obtain the following optima – which will serve as the new estimated of the parameters:

$$\tau_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} T_{j,i}^{(t)} \tag{2}$$

$$mu_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)} x_i}{\sum_{i=1}^{n} T_{j,i}^{(t)}} \tag{3}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{n} T_{j,i}^{(t)} (x_i - \mu_j^{(t+1)})(x_i - \mu_j^{(t+1)})^T}{\sum_{i=1}^{n} T_{j,i}^{(t)}}. \tag{4}$$

We see that our parameters can be calculated with relatively simple operations using the calculated values of $T$. Like before the inital values of the parameters could be a point of discussion, but in my implmentation I simply set the initial values of $\tau$ to all be equal, the values of $\mu$ to be $k$ randomly selected points in the dataset (sampled without

[1] Much of the notation used here is borrowed from the Wikipedia page on the EM-algorithm, generalized to the k-mixture case.

replacement) and the inital covariance matricies to be diagonal with a large variance to prevent undesirable local optima.
The Gaussian mixture model is implemented in the class GaussianMixture, which is well documented. The calculations of the matrix $T$, which we call expectation in the class, and the inner product used in the updating of $Sigma$ are both very computationally intensive so were rewritten in Fortran to obtain workable performance. This code, included in the appendix, will need to be compiled with **f2py** (including LAPACK flags) in order to be importable by Python — an example makefile is included alongside the code.
Both the KMeans and Gaussian mixture model algorithms are iterative and need criteria for when to stop, for this project I manually inspected the values of the parameters between iterations until it was clear that the values had stabalized.
So we begin with the image data, flatten the array and apply the two methods for varying values of $k$. The resulting segregations of the image are reshape and diplayed in figure x. We see that both of the methods produce a desirable classification for most values of $k$. The Gaussian mixture model produces clean demarcations around the cells which are "well filled", while the K-means algorithm produces less clean lines but is free from the noise of the Gaussian mixture where points clearly not a part of a cell have been spurriosly assigned to the same class as the cells — presumably because one of the Gaussian mixtures has settled on a covariance matrix with larger variance values compared to the other mixtures (something which is not possible in K-means).
As we are going to use a Gaussian mixture model again for the counting of clusters we will use the K-means segregation with $k = 3$ as the leptokurtic nature of the GMM segregation will handicap the fitting of a mixture of Normal distributions by overestimating the covariances.
We start by calculating which cluster has the fewest members, this will be the cluster containing the points that are determined to be within the disc-shaped section of the cells, and find the coordiantes of each of the points in this cluster — these will be our covariates. We now split our points randomly into train and test portions. Fitting a Gaussian mixture model for a range of $k$ values to the train set we calculate the log-likelihood of the test set under the fitted model. This is first done at a coarse level, in increments of 10, and then at the individual level around the optima of the log-likelihood. The results of the individual level optimization can be seen in fig y.
We see that we have optimal $k$ value of X, this is what we take as our estimate the number of cells in the image. Taking the values of the $\mu_j$ for the $k = X$ case and overlaying over the points obtained via segregation we obtain fig z. It's clear from this image that the EM-alogrithm has not been able to find the cluster in a few cases, and in others the irregular shapes of the discs has meant multiple normals fit better than a single normal with larger covariance.
Overall, these two effects have counterbalanced each other and the whole procedure has worked quite well. The segregation procedure certainly worked excellently for both methods and the counting stage worked as well as can be expected. The EM-algorithm's tendency to become stuck in local optima should be expected for so many mixture components. Perhaps a less naive method would be to place explicit priors on the hyper-parameters, such as a hierachical prior on the covariance matricies to provide shrinkage or a non-local prior to penalize small distances between means of the mixtures. Also a method of fitting the models which is less likely fall victim to the complex geometry of Gaussian mixture models, such as Hamiltonian Monte Carlo, would probably obtain more accurate results.

# Question 2

Say we wish to fit the model

$$y_i = f(\mathbf{x}_i) + \epsilon_i \tag{5}$$

where our dependent variable is modeled as coming from an underlying function of the covariates with noise. If our function can be decomposed into a finite linear sum of our

covariates, transformed by some kernal function(s), then we are doing linear regression:

$$f(x) = \sum_{m=1}^{M} \theta_m \phi_m(x) = \theta^T \phi(x). \tag{6}$$

The frequentist way of estimating the $\theta_m$ would be to find the values that maximise the likelihood of the data. However, viewing this model from a generative perspective, this modal estimate often gives non-typical values. Using a Bayesian approach we can place a prior over the values of $\theta$, which allows us to incorporate any additional information we believe about the structure of the model. Not only does this provide shrinkage, but the explicit computation of a posterior distribution enables us to work with expectations of model statistics. This gives both more robust estimatimation of the $\theta_m$ and allows us to calculate their marginal variances.
Say we place an independent Gaussian prior over each of the weights

$$p(\theta) \sim N(0, \alpha I_M) \tag{7}$$

then we can calculate the posterior as follows

$$p(\theta|X, y, \alpha) \propto p(y|\theta, X, \alpha) \cdot p(\theta). \tag{8}$$

This formulation can also be viewed from the perspective of a Gaussian process. The prior placed on the $\theta_m$ can allows us a direct interpretation of the covariance between two points

$$\begin{aligned} K(x, x^*) &= E[f(x), f(x^*)] && (9)\\ &= \phi(x)^T \Sigma_\theta \phi(x^*) && (10)\\ &= \alpha \phi(x)^T I_M \phi(x^*) && (11) \end{aligned}$$

In fact Gaussian process can be viewed a limiting case of Bayesian linear regression as every kernal function can be decomposed into an infinite sum over the eigendecomposistion of the basis functions

$$K(x, x^*) = \sum_{i=1}^{\infty} \lambda_i \varphi_i(x) \varphi_i(x^*) \tag{12}$$

we recognize that this summation form could be found for Gaussian noise linear regression by diagonalizing the covariance matrix.
Now we turn to the climate data. Plotting the data we see two things:

1. At the macro level the CO2 data is positively correlated with the previous values,
2. There is a yearly seasonal oscillating trend.

We use these two facts to construct an appropriate kernal function to represent the correlation between time points. The geenral trend is best captured using a squared exponential kernal function and the seasonal trend is captured using a sinusoidal kernal function. We combine these two to make our mixed kernal function

$$K(x, x^*) = \alpha_1 \exp\left(-0.5 * \frac{(point1 - point2)^2}{lambda^2}\right) + \alpha_2 \exp\left(-2 \frac{\sin^2(point1 - point2)}{period}^2\right) \tag{13}$$