WORLDQUANT UNIVERSITY

Compiled Content Module 4

MScFE 650

Machine Learning in Finance

Revised: 09/07/2019

Table of Contents

Module 4: Clustering Algorithms

This module introduces you to *k*-means clustering, Gaussian Mixture Models, and the Expectation Maximization algorithm. Thereafter, it explores certain applications in finance, including regime detection and the Hierarchical Risk Parity algorithm used in portfolio optimization.

Unit 1: *K*-means Clustering

Introduction

In our discussion of classification, we assumed fully observed data – i.e. each observation came with a class label. In many situations class labels are not available and need to be inferred from the data itself. This problem is often referred to as "clustering", or "unsupervised learning." The more general problem is to infer *any* missing information from the observed data. Throughout our discussion, the fundamental assumption is that, should the missing data somehow become available, the training can be done with ease.

A general tool for dealing with partially observed data is the **Expectation Maximization** (EM) algorithm. The basic idea is very simple. Since we can proceed with the training (estimation of the parameter values) for the fully observed data, we estimate the missing values by calculating an expectation based on the current estimate of the parameters. Once we have values for the missing data, we proceed to maximize a likelihood to get an updated estimate for the parameters. These are then used to re-estimate the missing data, and so on.

The simplest example of the EM algorithm is *k*-means clustering, the starting point of our discussion.

K-means clustering

We have N observations, each observation belonging to one of k classes, but we are not given any class information. Our task is to assign each observation to an appropriate class, more commonly referred to in this setting as *clusters*. It is important to note that we know, or guess, the value of k, – i.e. we assume that we know the number of clusters.

The intuition behind the *k*-means algorithm is straightforward and is based on the fact that if we know the cluster labels of each observation, then it is easy to calculate the mean of each cluster. On the other hand, if the mean of each cluster is known, then it is easy to assign each observation to a cluster. This allows the following procedure to be followed: choose a cluster mean for each cluster. For the time being, we choose these initial cluster means by selecting k random observations. Next, we assign each observation to the cluster mean closest to it[1]. Once all the

[1] It is possible to do a more sophisticated assignment.

observations have been assigned to a specific cluster, we use the observations belonging to each cluster to *update* the cluster mean. With updated cluster means available, one can re-assign the data values, until convergence.

Notes:

1. It is common practice to rescale the data before the clustering algorithm is applied, by, for example, whitening the data. This ensures that the data is normalized so that the distance measures have some objective meaning. Note, however, that by doing this one runs the risk of rescaling distances when they are actually indicative of between-cluster separation.

2. The description above uses a *hard assignment* of each data point to a specific class. This does not take into account that, at least for some points, there may be considerable ambiguity as to which class they belong. In such cases, it may be better to use a *soft assignment* where data points are assigned to clusters in such a way as to reflect this uncertainty.

3. The *k*-means algorithm is an optimization procedure, and as is commonly the case with optimization procedures, it is possible to end up in (bad) local optima. It is quite possible that you will experience this as you experiment with the accompanying Jupyter notebook. Since the optimum is determined by the initial choice of the parameters the initialization is of paramount importance.

4. The *k*-means algorithm is initialized by assigning *k* initial cluster means. As pointed out above, the initial choice is important because it determines the final clustering. One possibility is to select *k* random samples from the data and use these as the initial cluster means. This does not work well in practice since the choice of initial clusters may be heavily biased. In our experience a binary split procedure works better. It starts by selecting two initial cluster means at random. The data points are then assigned to these clusters in the normal way. The scatter of each cluster is computed and the cluster with the largest scatter is split by choosing two random samples from it. The algorithm is now iterated a few times, starting with these three initial cluster means. Then the cluster with the largest scatter is split again, and the procedure is repeated until the desired number of clusters is obtained. The algorithm is then iterated until convergence.

Unit 2: Gaussian Mixture Models

In practice, one often encounters situations where the data cannot be accurately represented by a single Gaussian probability density function (PDF). This is certainly the case for all multi-modal datasets. One possibility is to represent the data by a mixture of k Gaussians,

$$p(x|\theta) = \sum_{j=1}^{k} \pi_j \mathcal{N}\left(x|\mu_j, \Sigma_j\right), \tag{1}$$

where $\mathcal{N}\left(x|\mu_j, \Sigma_j\right)$ is a Gaussian distribution with means and covariance, μ_j and Σ_j, respectively, and $\pi_j \geq 0$ with $\sum_{j=1}^{k} \pi_j = 1$. The parameters θ that need to be estimated are the class means and covariances, μ_j and Σ_j respectively, as well as the "class probabilities" or mixture coefficients, π_j. These should be estimated from the data, $x_n,\ n = 1, \ldots, N$, but there is a complication – we don't know how the data values x_n contribute towards the different mixture components. In fact, this is exactly the same as the situation encountered with the k-means algorithm of the previous set of notes. In this case, the mixture components are the equivalent of the different clusters of the k-means algorithm.

If we know the mixture component to which each data point belongs, the learning problem is easy. We simply calculate sample estimates of the means and covariances from the data belonging to each mixture component. The data points assigned to each mixture component determines the mixture coefficients. The first task is, therefore, to assign an observation to the different mixture components, i.e. we want to calculate the mixture component probability $P\left(z_j = 1|x\right)$. This might remind you of the classification problem. In this case, however, we do not have class labels for training.

However, defining $\gamma\left(z_j\right) = P\left(z_j = 1|x\right)$, it follows from Bayes' theorem that:

$$\gamma\left(z_j\right) = \frac{P\left(z_j = 1\right)p\left(\boldsymbol{x}|z_j = 1\right)}{\sum_{i=1}^{k} P(z_i = 1)p(\boldsymbol{x}|z_i = 1)}$$

$$= \frac{\pi_j \mathcal{N}\left(x|\mu_j, \Sigma_j\right)}{\sum_{i=1}^{k} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i)}. \tag{2}$$

Note that this is a soft assignment where x is not assigned to a specific component, but its assignment is distributed over all the components. Here π_j is the prior probability of the j^{th}

mixture component, $z_j = 1$ i.e. $P(z_j = 1) = \pi_j$, *before* any observations, and $\gamma(z_j)$ is the posterior probability once we have observed x. Alternatively, $\gamma(z_j)$ can be viewed as the *responsibility* that the j^{th} component takes for "explaining" the observation x. Said in a different way, $\gamma(z_j)$is the probability of x belonging to the j^{th} mixture component.

In summary, provided we know the mixture parameters, π_j, μ_j and Σ_j the responsibilities allow one to make soft assignments of data to the different components. Thus, we also need to estimate these mixture parameters. For this we use the likelihood.

Example 1

Gaussian Mixture Models (GMM's) are generative models – i.e. we can draw samples from them. Consider a three-component GMM. We first find the mixture component by sampling the mixture component probabilities. Then we draw a sample from the corresponding Gaussian density function. The result is shown in Figure 6 (below). In (a) the samples, as drawn from the different components are shown. (This corresponds to fully observed data where each data point is assigned a "cluster"). In (b) the data is shown without its mixture components; this is the way it is presented to us. In (c) the responsibility that each component takes for every data point is indicated with different color shades.

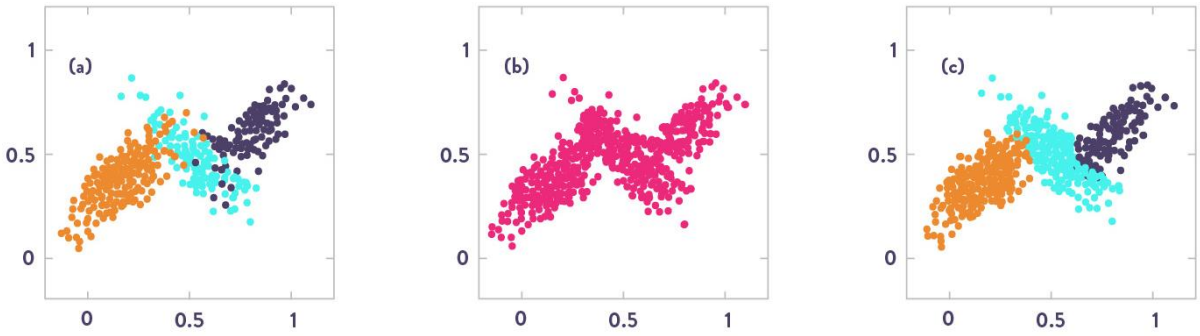

Figure 6: Generating samples from a mixture of three Gaussians

(Adapted from Bishop (2006)).

It is important to note that, for mixture models, the likelihood can have a singularity – i.e. it can become infinite. To see how this can happen, consider a mixture model where all the components have covariances of the form $\sigma_k^2 I$. Suppose one of the mixture components has its mean, say μ_j, exactly equal to one of the data points, i.e. $\mu_j = x_n$. The contribution of this data point to the likelihood is of the form,

$$\mathcal{N}\left(x_n \middle| x_n, \sigma_j^2 I\right) = \frac{1}{\sqrt{2\pi\sigma_j^2}}.$$

This goes to infinity as $\sigma_j \to 0$. That is to say, this particular component collapses onto a specific data point, sending the likelihood to infinity – i.e. we are in the process of maximizing the likelihood after all. There is always a possibility that this can happen and one has to take steps in order to avoid this situation[2]. Also note that this danger does not exist for single Gaussians.

In the next set of notes we show how we use the log-likelihood in order to estimate the parameters, given the responsibilities.

[2] Maybe it was not such a good idea after all to get rid of the priors above!

Unit 3: The Expectation Maximum (EM) Algorithm for Gaussian Mixture Models

In the previous section, we derived an expression for allocating the data values to the different mixture components. Now we show how that expression can be used to estimate the parameters of the Gaussian Mixture Model (GMM). One can readily derive the expressions by maximizing the log-likelihood. For those students who are interested in the details, we recommend you consult Bishop (2006). Here, we give the very intuitive expressions. Note that the expressions are very similar to the well-known estimates for the mean and covariances. The only difference is that the data values are weighed with their class assignment probabilities. The means are given by,

$$\mu_j = \frac{1}{N_j}\sum_{n=1}^{N}\gamma(z_{nj})x_n\,, \tag{3}$$

where

$$N_j = \sum_{n=1}^{N}\gamma(z_{nj}).$$

The expressions for the covariance matrices are similarly given by,

$$\Sigma_j = \frac{1}{N_j}\sum_{n=1}^{N}\gamma(z_{nj})(x_n-\mu_j)(x_n-\mu_j)^T.$$

Again, all the data points contribute towards the j^{th} mixture component, weighted with their responsibilities. Finally, the mixture coefficients are given by,

$$\pi_j = \frac{N_j}{N},$$

which is the average responsibility for the j^{th} mixture component.

Note that these equations are not solvable in closed form since the responsibility depends in a complicated way on the parameters. Their form however, does suggest the following algorithm: **Expectation Maximization** (EM) algorithm for GMMs.

1 Initialize the different coefficients, π, μ, and Σ. One possibility is to use the k-means algorithm.

2 **E step.** Evaluate the responsibilities using the current parameter estimates,

$$\gamma(z_{nj}) = \frac{\pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}{\sum_{i=1}^{k} \pi_i \mathcal{N}(x_n|\mu_i, \Sigma_i)}.$$

3 **M step.** Re-estimate the parameters using the current responsibilities,

$$\mu_j^{\text{new}} = \frac{1}{N_j} \sum_{n=1}^{N} \gamma(z_{nj}) x_n$$

$$\Sigma_j^{\text{new}} = \frac{1}{N_j} \sum_{n=1}^{N} \gamma(z_{nj})(x_n - \mu_j)(x_n - \mu_j)^T$$

$$\pi_j^{\text{new}} = \frac{N_j}{N}$$

Where

$$N_j = \sum_{n=1}^{N} \gamma(z_{nj}).$$

4 Evaluate the log-likelihood,

$$\ln p\,(X|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left\{ \sum_{i=1}^{k} \pi_i \mathcal{N}(x_n|\mu_i, \Sigma_i) \right\},$$

check for convergence of either the parameters, or the log-likelihood. It can be shown that the log-likelihood should increase monotonically, and this can be used a useful debugging tool – If it does not increase, something has to be wrong in the code.

5 Keep iterating from step two until convergence.

Bibliography

References

Bishop, C.M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer.

Collaborative Review Task

The collaborative review task is designed to test your ability to apply and analyze the knowledge you have learned in the module.

Instructions

- Work through the notebook, answer all questions, and do all problems.
- You are allowed to consult the internet, as well as your peers on the forum
- Your answers and solutions to the problems should be added to this notebook.
- Submit your final work as an html file or as two html files in one zipped file.
- Note that Python (version 3.6.4) has been used to calculate the solutions.

Task

Download this notebook to complete this week's collaborative review task. This notebook contains both the questions as well as providing you with space to answer the questions.