From K-means to Expectation Maximization

Miguel de Benito

Universität Augsburg

October 19th, 2014

We demonstrate an implementation of K-means understood as a minimization problem, linking it to Expectation Maximization for a Gaussian mixture. We briefly discuss the initialization technique K-means++.

Table of contents

1. Introduction

K-means is one of the most popular clustering algorithms, mostly because it is simple and easy to implement.[1] One wishes to assign N data points to exactly K clusters determined by K centroids: starting with these in random positions, the algorithm assigns each datum x to its closest center, then recomputes all the centers as the centers of the new clusters. This is a form of Lloyd iteration.

Following [Bis06], it is possible to view this procedure as the minimization of a certain energy (in this context known as *distortion*) and this minimization can be performed iteratively just as described. This provides an entry point for a general optimization procedure called *expectation maximization*, used for the computation of maximum likelihood point estimators.

[1] It should be noted, tough, that it can perform very badly in many examples, which need not be deliberately constructed to make K-means fail. It is known that it can compute arbitrarily bad clusterings with high probability even if the centers are chosen uniformly from the data points ([AV07]).

In this document we provide interactive examples based on a simple PYTHON implementation of K-means. It is therefore intended for use inside TEXMACS. Please see the appendix for the setup required.

2. THE SETTING

We use the **data set** $\boldsymbol{x} = (x_1, \ldots, x_N)$, $x_i \in \mathbb{R}^d$ from the Old Faithful geyser, after normalization with the affine transformation

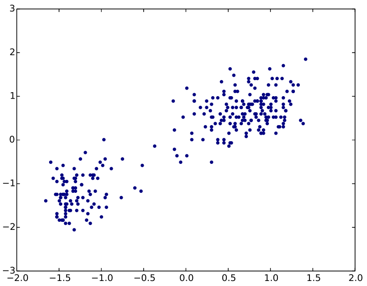
$$\tilde{x}_i := \frac{x_i - E(\boldsymbol{x})}{\sqrt{V(\boldsymbol{x})}},$$

where $E(\boldsymbol{x})$ and $V(\boldsymbol{x})$ denote the sample mean and variance. This trivially yields

$$E(\tilde{\boldsymbol{x}}) = 0 \quad \text{and} \quad V(\tilde{\boldsymbol{x}}) = E(\tilde{\boldsymbol{x}}^2) - \underbrace{E(\boldsymbol{x})^2}_{=0} = 1.$$

We read the data using `numpy.loadtxt` (which parses numbers, skips comments, etc.), then normalize and plot it with `matplotlib`:

```
>>> data = normalize(np.loadtxt("%s/old_faithful.data" % path, usecols=(1, 2)))
>>> ps_out(get_plot(data, cmap='jet'))
```



3. K-MEANS CLUSTERING

Problem 1. Assignment of each x_i to exactly one of K **clusters**, defined by K vectors $\mu_k \in \mathbb{R}^d$ (the centers of mass) such that the sum of the squares of the distances of each data point to its assigned vector μ_k is minimal.

NOTATION. *Let $r_{nk} \in \{0, 1\}$ for $n \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, K\}$ be such that $\sum_k r_{nk} = 1$ and denote the assignment of datum x_n to cluster k iff $r_{nk} = 1$.*

We want to minimize the energy (or **distortion measure**):

$$J_{\boldsymbol{x}}(\mu, r) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \, \|x_n - \mu_k\|^2.$$

Algorithm K-Means

1. Initialize μ_k to random values. (cf. K-means++)

2. Minimize the r_{nk} with respect to the μ_k.
3. Minimize the μ_k with respect to the r_{nk}.
4. Go to step 2 until convergence.

It is a trivial matter to differentiate wrt. each of the variables, set the result equal to zero and solve to find that the optima in each step are given by

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|x_n - \mu_j\|^2, \\ 0 & \text{otherwise,} \end{cases}$$

i.e. x_n is assigned to the closest cluster center, and

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}},$$

i.e. set the new center μ_k to be the center of all the points in class k.

Implementation. The code is in the class `Kmeans`. The assignment of points to classes will actually be implemented with a single array

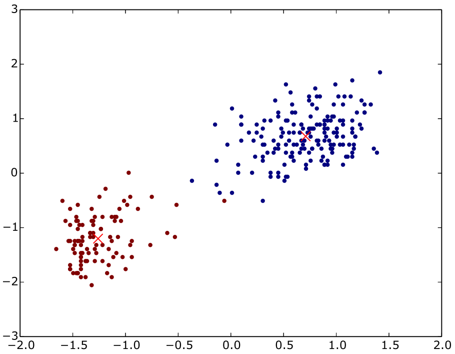
$$r[n] = k \Leftrightarrow r_{nk} = 1.$$

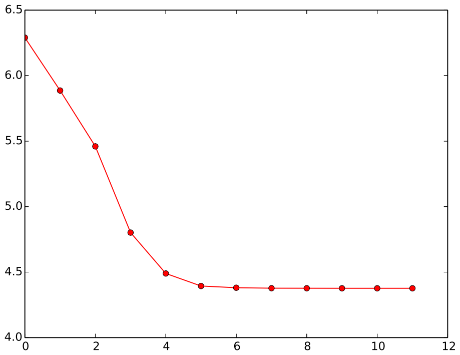
Step 2 of the algorithm is in `KMeans.minimize_assignments()`, step 3 in `KMeans.minimize_prototypes()`. Notice that sometimes the algorithm sets a center of a cluster to be a data point: $\mu_k = x_n$ for some n, k and a division by zero happens. One should avoid this...

Tests. Searching for two clusters in the dataset yields a J with a unique minimum. Starting from $K = 4$ upwards, the problem with local minima of gradient descent-ish methods like K-means becomes apparent. We plot the cluster assignment and the decrease in the logarithm of the objective function J after each of the stages described above:

```
>>> km = KMeans(2, data)
>>> km.restart()
    J = km.run()
    ps_out (get_plot (data, km.MU, km.R))
    pl.clf()
    pl.plot(np.log(J), marker='o', color='r')
    ps_out(pl)
```

```
   Iterations = 5, Distortion = 79.575959
```





4. Expectation maximization for Gaussian mixtures

We now introduce a statistical model on the data. This allows us to perform inference, thus answering questions like how likely it is that some data point belongs to a given class, or how large is the spread of a cluster.

We assume that $\boldsymbol{x} = (x_1, \ldots, x_N), x_i \in \mathbb{R}^d$ has been sampled from a convex combination of K independent Gaussian random variables of densities $\mathcal{N}(\cdot|\mu_k, S_k)$ with means $\mu_k \in \mathbb{R}^d$ and covariance matrices $S_k \in \mathbb{R}^{d \times d}_{\text{sym}}$. That is: letting the coefficients of the combination be $\pi_k \in [0, 1], \sum_k \pi_k = 1$, the probability density from which we assume the data is sampled is:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, S_k), \quad x \in \mathbb{R}^d.$$

Our goal is to find out which data points are more likely to come from each of these Gaussians, i.e. determine π_k, as well as to determine the parameters μ_k, S_k. To this end we introduce the discrete **latent random variable**

$$Z \in \{e_1, \ldots, e_K\}, e_i \text{ standard basis vectors of } \mathbb{R}^K,$$

and its "unobserved" realizations $\boldsymbol{z} = \{z_1, \ldots, z_N\}$, which are to be interpreted as $z_n^k = 1$ iff x_n "belongs to group k". This is to say that the mixing coefficients for the Gaussians are given by $p(z^k = 1) = \pi_k$. Our method is the maximization of the marginal log likelihood $\log p(\boldsymbol{x}|\mu, \Sigma, \pi)$, but ignorance of the actual values $\boldsymbol{z}$ leads us (after some work, see my notes) to maximize the expected value of $\log p(\boldsymbol{x}, \boldsymbol{z}|\theta)$ under the posterior of $\boldsymbol{Z}|\boldsymbol{X} = \boldsymbol{x}$:

$$E_{\boldsymbol{Z}|\boldsymbol{x}}[\log p(\boldsymbol{x}, \boldsymbol{z}|\theta)] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} [\log \pi_k + \log \mathcal{N}(x_n|\mu_k, \Sigma_k)],$$

wrt. the parameters μ_k, Σ_k, π_k to obtain new values $\mu_k^{\text{new}}, \Sigma_k^{\text{new}}, \pi_k^{\text{new}}$ at each step. This maximization (direct, by differentiation) requires us to compute the posterior probabilities

$$\gamma_{nk} = p(z_n^k = 1|x_n) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, S_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, S_j)},$$

which we call **responsibilities**.

Notice that the latent variables don't appear explicitly in the computations other than in the responsibilities, which we use via Bayes' law to compute better approximations of the parameters.

Warning 1. As it stands, the procedure will occasionally fail: exactly as with K-means if one of the means μ_k coincides at some step with a data point x_n, the contribution of $\mathcal{N}(x_n|\mu_k, S_k)$ to the log likelihood will be $\mathcal{O}((\det S_k)^{-1/2})$ so it would suffice to send the eigenvalues of S_k to 0 in order to achieve unbounded log likelihood (imagine for example the case of spherical covariance $S_k = \sigma_k \operatorname{Id}_d$ for some scalar σ_k, then $\mathcal{N}(x_n|\mu_k, S_k) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{\sigma_k}} \rightarrow \infty$). See the discussion in my notes and [Bis06, Chapter 9] for more on this problem.

One solution is to add some heuristics, like resetting the mean to some random value if it has been assigned to some x_n, but this isn't particularly elegant.

A better one is to *regularize* the log likelihood with a parameter penalizing vanishing covariances. Or in a better justified approach, one adds the prior of the parameters to the likelihood, and performs *maximum a posteriori* estimation.

Algorithm EMG

1. Initialize μ_k, S_k, π_k. Compute log likelihood.
2. **E-step:** Evaluate the **responsibilities** with the "old" parameter values

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, S_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, S_j)},$$

   and the total responsibility of each Gaussian:

$$R_k = \sum_{n=1}^{N} \gamma_{nk}$$

3. **M-step:** Compute the "new" parameter values:

$$\begin{aligned}
\mu_k^{\text{new}} &= \frac{1}{R_k} \sum_{n=1}^{N} \gamma_{nk} x_n, \\
S_k^{\text{new}} &= \frac{1}{R_k} \sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k^{\text{new}}) \otimes (x_n - \mu_k^{\text{new}}), \\
\pi_k^{\text{new}} &= \frac{R_k}{N}.
\end{aligned}$$

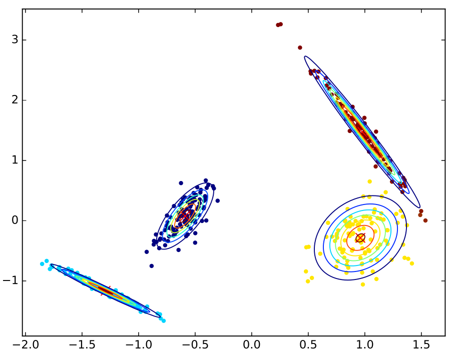
4. Evaluate the log likelihood with the new parameters. Repeat if the stop criterion is not satisfied.

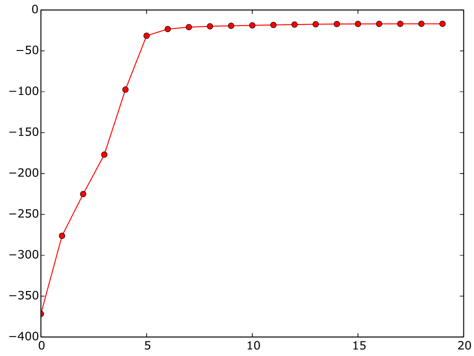
The implementation is in the class `EMGauss`.

Synthetic tests. We sample multivariate normals and try to fit their parameters with `EMGauss`:

```
>>> n=400
    k=4
    # Generate data
    means, covs, syndata = gen_data (n, k)
    # Run EMG and plot result
    emg = EMGauss (syndata, k, maxiter=20)
    LL = emg.run()
    ps_out (emg.get_plot (cmap='jet'))
    # Plot log p(x|π, μ, S)
    pl.clf()
    pl.plot (LL, marker='o', color='r')
    ps_out (pl)
```

```
Iterations = 20, log likelihood = -16.887090
```

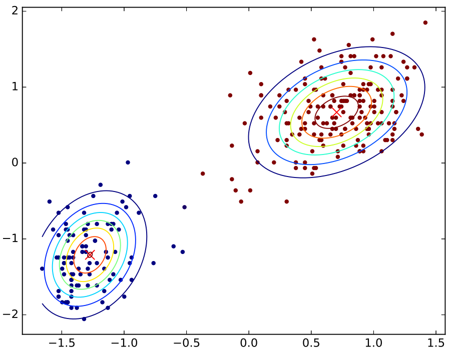


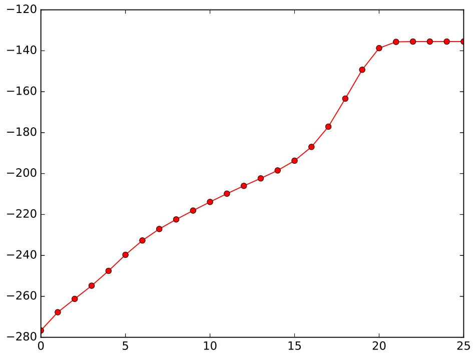


Old faithful data. Applied to this data, convergence is somewhat slower:

```
>>> emg = EMGauss (data, k=2, maxiter=50)
    #emg.restart()
    LL = emg.run()
    ps_out (emg.get_plot ())
    pl.clf()
    pl.plot (LL, marker='o', color='r')
    ps_out (pl)
```

```
Iterations = 26, log likelihood = -135.509415629
```





5. K-MEANS++

K-means++ is an initialization technique for the means which provides improved convergence rates. With this method [AV07] proves that the expected distortion measure is (tightly) bounded above by roughly $\log(k)$ times the optimal distortion:

$$\mathbb{E}(J) \leqslant 8 \log(k+2) J_{\text{opt}}.$$

Algorithm K-means++ ("D^2 seeding" [AV07])

1a. Pick μ_1 uniformly at random from the data $\boldsymbol{x}$.

1b. For each $k = \{1, \ldots, n\}$ set $\mu_k = x$ with probability $\frac{D(x)}{\sum_{i=1}^{n} D(x_i)^2}$, where $D(x)$ is the distance from x to the closest μ_i, with $i < k$.

2-4. Proceed as in the standard K-means.

The bottom line here is that, if one is doing vanilla K-means, then one should be doing K-means++.

Appendix. TeXmacs setup

In order to use this file you need:

- A working installation of Python, NumPy and Matplotlib.
- A recent version ($\geqslant$1.99.3) of TeXmacs if you want syntax highlighting, autocompletion, etc. in the Python sessions below
- In case you don't have this (hidden) preference set, execute the following line:

```
Scheme] (set-preference "editor:verbatim:tabstop" 4)
```

Here's the Python init code. You should change the path to the data set and to `emkmeans.py`.

```
>>> import numpy as np
    import matplotlib.pyplot as pl
    import sys
    sys.path.extend(['/path/to/km2em'])
    from km2em import *
    path = "PATH TO YOUR DATASET"
```

All of the functions and the two classes that we will be using below are defined inside `kmeans.py`.

6. References

Most of this document is heavily based on [Bis06] and [Mur12]. For a deeper treatment of the questions in this note you should read [Bis06, Chapter 9]. The sources are in Bitbucket.

[AV07] David Arthur and Sergei Vassilvitskii. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035. Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, 1 edition, aug 2006.

[Mur12] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, aug 2012.