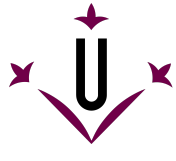
Clustering data - Probabilistic Algorithms

Ramon Béjar Torres

Data mining - Master on Computer Science



Universitat de Lleida

Probabilistic (soft) clustering with Expectation Maximization

We are now going to tackle the problem of clustering data as assigning clusters to points following a probabilistic model. In this case, we assume that there is a probabilistic generative model with two components:

1. There is a probability distribution among the k different clusters. This probability distribution dictates which cluster generates each point obtained from the source that generated our data set.
2. Once a cluster i has been selected, the value for each dimension of the data point comes from a particular Multivariate gaussian distribution with particular mean (d-dimensional) vector μ_i and covariance ($d \times d$) matrix Σ_i.

This generation model is called a **k-Gaussian mixture model**. So, in this model we have two different sets of parameters:

1. the probabilities that define the finite probability distribution among the k clusters ($P(C = i)$). It defines the probability that the cluster selected to generate a sample point $\overline{x}$ is the cluster i.
2. the μ_i and Σ_i parameters of the Multivariate gaussian distributions associated with the clusters, and that define the probabilities $P(\overline{x}|C = i)$

The conditional probability theorem tell us that the probability that a certain data point $\overline{x}$ is obtained in our data set can be expresed as:

$$P(\overline{x}) = \sum_i P(\overline{x}, C = i) = \sum_i P(\overline{x}|C = i)P(C = i)$$

where:

- $P(C = i)$ is the probability that the cluster picked to generate a sample data was i
- $P(\overline{x}|C = i)$ is the probability to obtain $\overline{x}$ if we assume that the cluster i (multivariate gaussian distribution with parameters μ_i and Σ_i) was the one picked to generate $\overline{x}$.

However, observe that in our clustering problem we do not know neither the cluster probabilities ($P(C = i)$) nor the gaussian parameters μ_i and Σ_i that allow us to compute $P(\overline{x}|C = i)$. So, the question is:

> *Can we learn estimates of these $P(C = i)$ and μ_i, Σ_i parameters from our data set ?*

Observe that in our data set we do not have any attributes that indicate which clusters generated each data point (unsupervised learning problem). So even if we assume a model where the clusters are indicated by a certain random variable, that variable **is hidden** in our data (we cannot directly observe its value in our data points).

How can we discover the most probable cluster for each point ($P(C = i|\overline{x})$)?

1. If we knew the μ_i, Σ_i and $P(C = i)$ parameters, we could learn the most likely values for the cluster indicator variables $P(C = i|\overline{x})$.
2. If we knew the $P(C = i|\overline{x})$ cluster indicator variable values, we could learn the most likely values for the parameters μ_i, Σ_i and $P(C = i)$ (the values for them that maximize the probabilities observed for our data points).

So, how can we break this deadlock situation ?

The EM algorithm for learning a k-Gaussian mixture model

Given the mutual dependence between the two sets of parameters we want to learn, the approach to learn a good estimation of them is to use an iterative algorithm that although it may not always learn the best possible model, in many cases learns a good enough model (one that predicts with good accuracy the probabilities of the observed data points).

This algorithm is the Expectation Maximization algorithm for a k-Gaussian mixture model.

As we are going to see, this algorithm actually behaves like a probabilistic version of the previous k-means algorithm, where instead of assigning an unique cluster to each point, we define a probability distribution over cluster selection and μ_i, $\sum_i$ parameters for each cluster that change in every iteration until no further changes to the parameters can give a better fit to the probabilites observed.

But with this k-Gaussian mixture model, a point is not crisply assigned to a cluster, although the algorithm computes the probability that a particular cluster generated one of our data points.

The EM algorithm for learning a k-Gaussian mixture model is the following one:

Initially, we generate random/pseudo-random values (or with some other method) for the parameters $P(C = i)$ and μ_i, Σ_i .

Then, we change the parameters iteratively **until they converge** repeating the two following steps:

1) In the E-step, we compute the expected values for the hidden indicator random variables $P(C = i|\overline{x})$. By bayes conditional probability rule these probabilities can be computed using our initial equation and current model parameters as:

$$P(C = i|\overline{x}) = \frac{P(C = i)P(\overline{x}|C = i)}{P(\overline{x})} = \frac{P(C = i)P(\overline{x}|C = i)}{\sum_j P(C = j)P(\overline{x}|C = j)}$$

Observe that $P(C = i)$ and $P(\overline{x}|C = i)$ are computed with our current model parameters, as $P(\overline{x}|C = i)$ is computed with the probability density function of a multivariate normal distribution with parameters μ_i, Σ_i.

Check:
https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Density_function (https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Density_function)

2) In the M-step, we compute new parameter values for $P(C = i)$ and μ_i, Σ_i that maximize the likelihood of our data points, given the expected values for the hidden variables computed in the previous step:

$$\mu_i = \frac{1}{p_i} \sum_{\overline{x}} \overline{x} P(C = i|\overline{x})$$

$$\Sigma_i = \frac{1}{p_i} \sum_{\overline{x}} P(C = i|\overline{x})(\overline{x} - \mu_i)(\overline{x} - \mu_i)^T$$

$$P(C = i) = p_i / N$$

where $p_i = \sum_{\overline{x}} P(C = i | \overline{x})$ and N is the number of data points in the data set.

k-Gaussian Mixture with EM algorithm vs K-means

E-step:

	$given:$	find for each point $\overline{x}:$
EM	$P(C=i)$	$P(C=i\|\overline{x})$
	μ_i, Σ_i	
$k-means$	cluster centers i	Closest center for $\overline{x}$

M-step:

	$given:$	find best (MLE) for cluster i :
EM	$P(C=i\|\overline{x})$	$P(C=i)$
		μ_i, Σ_i
$k-means$	Cluster assigned to $\overline{x}$	cluster center for i

Parallel implementation in the Map-Reduce framework

The implementation in a Map-Reduce framework of this algorithm follows a similar scheme to the one described for the k-means algorithm, as both algorithms perform a similar work inside of the two steps they execute in every iteration:

1. In the E-step, for each point $\overline{x}$ we have to compute the probabilites $P(C = i|\overline{x})$ associated with the hidden variable. This computation is local to each data point (for each point $\overline{x}$ we compute its set of k $P(C = i|\overline{x})$ values using only the current parameters of the model. So, this step is totally independent for each data point, and thus it can be implemented as a pure Map operation.

1. In the M-step, for each cluster i we compute new parameters $P(C = i)$ and μ_i, Σ_i collecting the values $P(C = i|\overline{x})$ computed for all the points $\overline{x}$:
    A. First, we can compute μ_i using a Map-Reduce operation (with all the $\overline{x}P(C = i|\overline{x})$ values)
    B. Then, Σ_i is computed with a second Map-Reduce operation, with Map we compute the $d \times d$ matrix $(\overline{x} - \mu_i)(\overline{x} - \mu_i)^T$ for each point $\overline{x}$ and cluster i, and then all such matrices with the same i value are added up in a Reduce operation to finally compute Σ_i.

Then, once the algorithm converges (so no better parameters can be found), we end up with a model that although is locally good, it may not necessarilly be the best global model. That is, better models could be found if we were changing at the same time both the hidden variables distributions and the clusters parameters. But there is no good method to perform such global search through the space of all parameter combinations.

With the resulting model, observe that we can use the final parameters $P(C = i|\overline{x})$ to make a soft prediction of the cluster that generated each data point $\overline{x}$.

For a more detailed discussion of the implementation of the distributed version of this algorithm in different distributed frameworks, read the paper:

H. Cui, J. Wei and W Dai. Parallel Implementation of Expectation-Maximization for Fast Convergence. URL: https://users.ece.cmu.edu/~hengganc/archive/report/final.pdf (https://users.ece.cmu.edu/~hengganc/archive/report/final.pdf)

It is worth noticing that EM based algorithms are also used for predicting values of other families of random variables in problems where we assume that our data has been generated (or can be modelled) with certain random variables where some of their parameters are unknown, but we want to learn the most probable parameters. As an another example of the use of EM based algorithms for big data applications, check for example the following paper:

> *Timothy Hunter, Teodor Mihai Moldovan, Matei Zaharia, Samy Merzgui, Justin Ma, Michael J. Franklin, Pieter Abbeel, Alexandre M. Bayen: Scaling the mobile millennium system in the cloud. SoCC 2011: 28 URL: https://cs.stanford.edu/~matei/papers/2011/socc_mobile_millennium.pdf (https://cs.stanford.edu/~matei/papers/2011/socc_mobile_millennium.pdf)*

Probabilistic clustering with the countries data

We are going to test the learning of a k-Gaussian mixture model with an implementation of the EM algorithm available in spark. Let's try it first with our data set of countries.

In [7]:
```
from pyspark.mllib.clustering import GaussianMixture
```

In [9]:
```
gmc = GaussianMixture.train( countriesRDD , 3, maxIterations=20)
bestcclustersRDD = gmc.predictSoft(countriesRDD)

for i,countrypred in enumerate(bestcclustersRDD.collect()):
    print ( countriesdata[i][0], " soft: ",countrypred , \
    " most probable: ", countrypred.index(max(countrypred)))

print ("\n")
```

```
Afghanistan  soft:  array('d', [0.9999999999996638, 1.680814077342096e-13, 1.
680814077342096e-13])  most probable:  0
Armenia  soft:  array('d', [3.904695943883008e-14, 0.00015816183353148798, 0.
9998418381664295])  most probable:  2
India  soft:  array('d', [2.2919835913478193e-14, 0.9999986047543992, 1.39524
55779348648e-06])  most probable:  1
Iran  soft:  array('d', [5.677016599833733e-15, 0.9999951209368437, 4.8790631
5055438e-06])  most probable:  1
Iraq  soft:  array('d', [0.9999999892503094, 1.074952250294818e-08, 1.6808140
772726895e-13])  most probable:  0
Yemen  soft:  array('d', [0.9999038372059142, 9.616279391772611e-05, 1.680894
7543223407e-13])  most probable:  0
Israel  soft:  array('d', [1.2465136513832529e-14, 0.9999994778845184, 5.2211
54692621626e-07])  most probable:  1
Italy  soft:  array('d', [8.609980685138532e-15, 2.3493667968656236e-10, 0.99
999999997650547])  most probable:  2
Germany  soft:  array('d', [3.234994976195858e-14, 5.416481204497942e-14, 0.9
999999999999134])  most probable:  2
Denmark  soft:  array('d', [7.10915289314594e-15, 1.015613675143266e-09, 0.99
99999989843792])  most probable:  2
France  soft:  array('d', [1.0622384617552475e-14, 2.6182168483984205e-05, 0.
9999738178315054])  most probable:  2
Spain  soft:  array('d', [9.189833997001492e-15, 2.221761764856341e-06, 0.999
997778238226])  most probable:  2
Austria  soft:  array('d', [7.11173161454085e-15, 1.3123156615554472e-07, 0.9
```

In [10]:
```
# output parameters of model
#
#  weights[i] = probability of selecting cluster i in k-Gaussian mixture model
#  gmc.gaussians[i].mu = mean vector for i cluster
#  gmc.gaussians[i].sigma.toArray() = covariance matrix for i cluster
for i in range(3):
    print ("weight = ", gmc.weights[i], "mu = ", gmc.gaussians[i].mu,
        "sigma = ", gmc.gaussians[i].sigma.toArray())
```

```
weight =  0.14285256537431182 mu =  [33.36677453753062,8.000054482818726] sigm
a =  [[14.06897657 14.8136246 ]
 [14.8136246  18.4471651 ]]
weight =  0.4754587004936496 mu =  [17.71407736172193,6.129427140733065] sigma
=  [[5.94638146 0.0988983 ]
 [0.0988983  0.56491959]]
weight =  0.3816887341320386 mu =  [10.384219009222933,9.606497197354345] sigm
a =  [[ 2.83668015 -0.74263449]
 [-0.74263449  0.89034546]]
```

Probabilistic clustering with the user-movies data set

Next, let's check the probabilistic model obtained with our users and movies data set

In [8]:

```
# Build the model (cluster the data) with expectation maximization
# for our usersandmovies data set
gmm = GaussianMixture.train( usersandmoviesRDD, 3, maxIterations=20)

# Let's show the soft prediction for each data point
bestclustersRDD = gmm.predictSoft(usersandmoviesRDD)
for user in bestclustersRDD.collect():
    print (user,\
    " most probable: ", user.index(max(user)))

print ("\n")
```

```
array('d', [1.2920150255400061e-11, 1.2920150255400061e-11, 0.999999999974159
7])  most probable:  2
array('d', [1.2920150255400061e-11, 1.2920150255400061e-11, 0.999999999974159
7])  most probable:  2
array('d', [1.2920150255400039e-11, 1.2920150255400039e-11, 0.999999999974159
7])  most probable:  2
array('d', [1.2920150255400061e-11, 1.2920150255470102e-11, 0.999999999974159
7])  most probable:  2
array('d', [4.718711285790615e-12, 0.9799910055803017, 0.020008994414979654])
most probable:  1
array('d', [4.727342028721405e-12, 0.9817834548172617, 0.018216545178010985])
most probable:  1
array('d', [4.287434503846263e-12, 0.8904226167505394, 0.10957738324517319])
most probable:  1
array('d', [4.798648986065309e-12, 0.9965926204135449, 0.003407379581656468])
most probable:  1
array('d', [1.0, 5.510387703321201e-33, 1.0524175752111612e-28])  most probabl
e:  0
array('d', [1.0, 5.510387703321201e-33, 1.0524175752111612e-28])  most probabl
e:  0
```

As you can see, picking the most probable cluster for each data point we obtain **in many executions** the same user clusters we obtained with the most typical executions of k-means.

However, observe that the probability distributions obtained over the three clusters give us more information, as they can be used to measure how similar may be two particular clusters, when considered as possible clusters for a data point.