Big Data Processing 2024-2025 Project Assignment

Implementing Expectation–Maximisation for Gaussian Mixture Models in Spark

Joeri De Koster (Joeri.De.Koster@vub.be), Jens Van der Plas (Jens.Van.der.Plas@vub.be)

As a practical assignment, students are required to complete the project assignment described in this document. As part of this project assignment students are required to implement an Expectation–Maximisation algorithm for classifying univariate multi-modal Gaussian Mixture Models in Spark. The implementation is required to work on our cluster configuration set up at the Software Languages Lab (Isabelle). The students are required to detail their findings in a report of up to 5 pages. The oral exam is an oral examination of the theory as well as an oral defence of the project. The project is to be completed individually by each student and submitted to Canvas.

Scenario

Prof. De Koster recently found a suspiciously looking folder on Isabelle named /data/secret/. This folder contains some files with raw numbers, but no description is given about their meaning. Closer inspection of the data seems to suggest that the data is drawn from multiple gaussian distributions. In order to unravel this mystery Prof. De Koster asks you to implement an expectation–maximisation algorithm for gaussian mixture models in Spark in order to find the weights, means and standard deviations of this multi-modal dataset. This might give us some insights in what these numbers represent.

The next section will give you some details on what gaussian mixture models are and how an expectation-maximisation algorithm works. The description in tasks 1 and 2 will give you some more information on how to apply this algorithm to our specific scenario.

Gaussian Mixture Models

A Gaussian Mixture Model is a probabilistic model for representing normally distributed subpopulations within an overall population. Similar to k-means, gaussian mixture models can be used to cluster unlabelled data. Unlike k-means, which are defined by a center and use euclidian distance as a metric, gaussian mixture models use a mixture of different gaussian distributions to model multimodal data. Each k-means cluster is represented by its center and data points belong to the cluster for which the center is closest. Gaussian mixture models however, assign a probability that a datapoint belongs to a certain cluster for each of the clusters. This means they are able to accommodate clusters of variable size much better than k-means. Figure 1 illustrates this phenomenon.

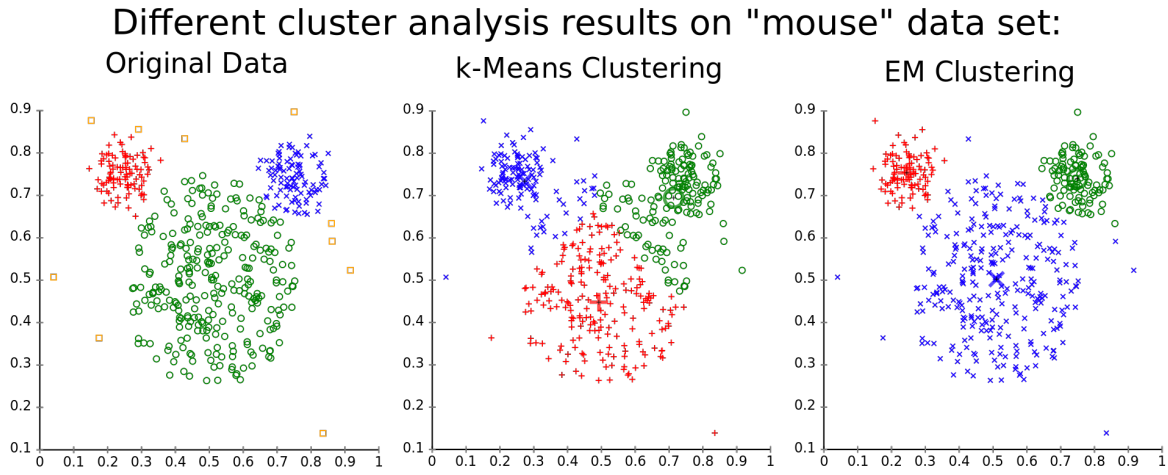

Figure 1

https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

For this assignment, we are going to restrict ourselves to one-dimensional (univariate) gaussian mixture models. A model with K components can be defined by K normal distributions, $N(\mu_k, \sigma_k)$, and their relative weight, ϕ_k. With the constraint that $\sum_{k=1}^{K} \phi_k = 1$.

The joint probability, $p(X)$, that all observations of a dataset, X, are drawn from a univariate gaussian mixture model is given by:

$$p(X) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \phi_k \mathcal{N}(x_n \mid \mu_k, \sigma_k) \qquad \textbf{(1)}$$

Where:

- $p(x_n)$ is the probability of observing x_n under the given gaussian mixture model.
- ϕ_k is the component weight of $N(\mu_k, \sigma_k)$, in other words, the probability that x_n is generated by $N(\mu_k, \sigma_k)$. For example, if 80% of the datapoints are drawn from $N(\mu_k, \sigma_k)$, then $\phi_k = 0.8$

- $\mathcal{N}(x_n \mid \mu_k, \sigma_k)$ is the probability of observing x_n, given a single gaussian $N(\mu_k, \sigma_k)$ with mean μ_k and covariance σ_k. This number can be calculated by the following formula:

$$\mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

A gaussian mixture model that perfectly fits the dataset maximises the value of $p(X)$. In the following section we will see how an expectation–maximisation algorithm can be used to find the parameters $(\phi_k, N(\mu_k, \sigma_k))$ from unlabelled data.

Expectation–maximisation algorithm

Expectation maximisation is an iterative method to find the maximum likelihood estimates for certain parameters in a statistical model. This can be applied to a number of different statistical models. However, in this section we will focus on its specific instantiation for univariate gaussian mixture models.

The value of $p(X)$, given by equation (1), is a value between 0 and 1. In order to find the maximum of that value, let us first take the logarithm of both sides of the equation to get more manageable numbers.

$$\ln p(X) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \phi_k \mathcal{N}(x_n \mid \mu_k, \sigma_k) \qquad \textbf{(2)}$$

If we differentiate this equation with respect to the mean and covariance and then equate it to zero, then we will be able to find the optimal values for the parameters. However, because we are dealing with not just one, but many Gaussians, things will get a bit complicated.

Expectation maximisation to the rescue!

We are not going into details of the math behind this algorithm. We simply describe the three different steps that need to be taken in order to estimate the different parameter of the model.

Initialisation step

- Randomly assign sample data points, x_n, from the dataset to the mean estimates, μ_k
- Set all component variance to the dataset variance, σ^2, given by:

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - \bar{x})^2$$

Where $\bar{x}$ is the dataset mean.

- Set all component weights to a uniform distribution $\phi_1, \ldots, \phi_K = \frac{1}{K}$

Expectation step

For each datapoint, calculate the likelihood, γ_{nk}, that it belongs to each of the gaussian components of the current gaussian mixture model. This is given by the likelihood of a

datapoint belonging to a single component divided by the sum of all likelihoods. $\forall n, k$ calculate:

$$\gamma_{nk} = \frac{\phi_k \mathcal{N}\left(x_n \mid \mu_k, \sigma_k\right)}{\sum_{k'=1}^{K} \phi_{k'} \mathcal{N}\left(x_n \mid \mu_{k'}, \sigma_{k'}\right)}$$

Maximisation step

Using the values for γ_{nk}, calculated in the expectation step, update each of the parameters of the model in the following order:

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

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}}$$

$$\sigma_k^2 = \frac{\sum_{n=1}^{N} \gamma_{nk} \left(x_n - \mu_k\right)^2}{\sum_{n=1}^{N} \gamma_{nk}}$$

The Expectation and Maximisation (EM) steps can be repeated until the likelihood given by formula (2) converges to a (local) optimum. Convergence can be measured by taking the delta of the values between iterations until it falls below some ϵ threshold.

In all of the above formulas, the value for K is fixed. When the number of components is not known a priori, it is typical to guess the number of components and fit that model to the data using the EM algorithm. This is done for many different values of K. Usually, the model with the best trade-off between fit and number of components (simpler models have fewer components) is kept.

Task 1: Implement EM

You are asked to implement an Expectation Maximisation algorithm for an unlabelled dataset of floating point numbers and a given value for K. Pseudocode for the algorithm is given below. Please note that any skeleton code that is included in this document serves a purely illustrative purpose and you are not forced to follow it. For example, all of the code assumes an implementation using RDDs while you are also allowed to use Dataframes or Datasets.

```
type GMM = (Array[Float], Array[Float], Array[Float]);

def EM(X: RDD[Float], K: Int): GMM = {
  // Initialisation Step
  val μ⃗: Array[Float] = sample(X, K);
  val x̄: Float = mean(X);
  val σ²: Float = variance(X, x̄);
  val σ⃗²: Array[Float] = Array.fill(K)(σ²);
  val φ⃗: Array[Float] = Array.fill(K)(1 / K);
  var ln p(X): Float = logLikelihood(X, φ⃗, μ⃗, σ⃗²);

  do {
    // Expectation Step
    val γ⃗: RDD[Array[Float]] = gamma(X, φ⃗, μ⃗, σ⃗²);

    // Maximisation Step
    for ( k <- 0 to K-1) {
      φ⃗(k) = updateWeight(γ⃗, k);
      μ⃗(k) = updateMean(γ⃗, X, k);
      σ⃗²(k) = updateVariance(γ⃗, X, μ⃗(k), k);
    }

    // Measure Convergence
    val ln p(X)′ = ln p(X);
    ln p(X) = logLikelihood(X, φ⃗, μ⃗, σ⃗²);

  } while ((ln p(X) - ln p(X)′) > ϵ);

  return (φ⃗, μ⃗, σ⃗²)
}
```

Task 2: Find the GMM for our secret dataset

Once you have finished the implementation of the algorithm you are asked to run your code on the secret dataset. We only guarantee that the values were drawn from multiple gaussian distributions. The amount of distributions, their relative weight, mean and standard deviation are not given. In order to find the optimal values you will need to run the EM algorithm for multiple values of K and report on your findings. Once you have found the optimal parameters, can you guess what the dataset represents?

Task 3: Benchmarks

You are asked to run some benchmarks on your local machine as well as on Isabelle. Please report on the dataset used, total runtime and number of iterations of your algorithm. Analyse and report on your results in terms of the performance techniques we've seen in class that are specific to Spark. What techniques did you apply? Can you explain the performance results in terms of Spark's underlying execution mechanics?

Task 4: Answering Questions

Please include a section in your report in which you answer the following questions:

- Have you persisted some of your intermediate results? Can you think of why persisting your data in memory may be helpful for this algorithm?
- In which parts of your implementation have you used partitioning? Did it help for performance? If so, why?
- Did you use RDDs, DataFrames or Datasets for your implementation? Why?

Practical Info

- Students are required to implement the assignment individually. Collaboration between students is not allowed.
- The report should include an explanation of the design decisions made for implementing this assignment, a section about the obtained results from running the experiments in terms of performance and a section about the questions detailed in task 3.
- The deliverables for this project include a single scala file with your code and a pdf with the report (max 5 pages). These can be submitted as a single zip file through Canvas.
- The oral exam is an oral defence of your project in combination with an examination of the theory.
- For running your experiments on a local machine you will be provided with a smaller sample dataset.

- Once you have finalised your implementation you are required to run some experiments on Isabelle with the full dataset. Each student will get two 2h slots in order to run their experiments. Jens Van der Plas (Jens.Van.der.Plas@vub.be) will contact you about reserving a slot.