Cloud Computing and Big Data Processing

Gaussian Mixture Model
Expectation - Maximation

Erick Escobar Gallardo

erick.emmanuel.escobar.gallardo@vub.be

January, 2021

1 Introduction

In the project assignment for the Big Data Processing Class, we implemented the clustering algorithm Gaussian Mixture Models - Expectation Maximization method, where we will analyze and determine the number of clusters in the dataset given the information that the raw numbers on it could belong to multiple gaussian distribution. We need to find the weights, means and standard deviation of this multi-modal dataset.

2 Tasks

2.1 Task 1

The algorithm is implemented in three sections: Initialization, Expectation and Maximization.

2.1.1 Initialization

The first part of the initialization process is to read the dataset, for this, we will map the inivital values of the dataset to double values, and we will be creating an RDD of the dataset points. This RDD has to be persisted in memory since we will use this RDD for latter calculations inside the main loop, lets remember that the map transformation is lazy and will not be executed until we find an action.

In the initialization phase, we will define the parameters of the input model, such as:

- K: Defines the number of clusters (normal distributions) to consider in our model.
- epsilon: Threshold value, used as one stop condition, when the difference between log likelihoods is below this value, the execution of the algorithm should end.
- maxCycles: Defines the maximum number of cycles to execute in the algorithm during the expectation-maximization step. This is used as a second stop condition, to ensure the algorithm comes to an end.
- numCyclesLnp: Optimization parameters, that defines the number of cycles that has to pass to calculate the new log likelihood. This parameters ensures a slightly faster execution in the Isabelle Cluster with bigger datasets, such as *Medium* and *Big* datasets.

During this steps, the initial weights, means and variances are also calculated.

2.1.2 Expectation

During this step we will first calculate the log likelihood value with the initial values for the mean, variances and weights, for K clusters. All the actions used in the expectation and maximization step are **Aggregates**. The use of Aggregates has the advantage of first aggregating elements in each partition and then aggregating results of all partition to get the final result, thus combining two operations in one action, a sequential and a combining operation. In this phase the gamma RDD is also calculated, to achieve this the following approach was taken, where we will map each element of the original input RDD (X) to an array of K+1 members, where the first K elements represent the gamma values or the likelihoods of a datapoint belonging to a single component divided by the sum of all likelihoods, the last K+1 element is the original datapoint. This approach help us to use only one action for the calculations of the new means and variances in the maximization step.

```
val gamma: RDD[Array[Double]] = X.map(r => {
   val tempArray: Array[Double] = Array.fill(K + 1)(0.0) // Temp Array of size K +1
   val divisor: Double = kSumLogLikelihood(K, r, datasetWeights, datasetVariance, datasetMean) // Sum of the
       logLikelihood for K values
   for (k <- 0 to K - 1) {
     tempArray(k) = (datasetWeights(k) * pdf(r, datasetVariance(k), datasetMean(k))) / divisor }
   tempArray(K) = r
   tempArray
 }).persist() //persist since the gamma RDD will be used to calculate the new weights,variances and means
```

2.1.3 Maximization

During the Maximization step we will calculate the new variances, weights and means. For these calculations, the Aggregate action was used, since we can use two operations in just 1 action, first we will perform a specific function for each RDD partition and then we will aggregate the results of each RDD partition into a single result.

```
for (k <- 0 to K - 1) {
    val sumLikelihoods = gamma.aggregate(0.0)((acc: Double, v: Array[Double]) => acc + v(k), _ + _) //
        Calculates the sum of the gamma values for k value
    datasetWeights(k) = sumLikelihoods / totalCount // update the weights
    datasetMean(k) = (gamma.aggregate(0.0)((acc: Double, v: Array[Double]) => acc + v(k) * v(K), _ + _)) /
        sumLikelihoods // update the means
    datasetVariance(k) = (gamma.aggregate(0.0)((acc: Double, v: Array[Double]) => acc + v(k) * pow((v(K) -
        datasetMean(k)), 2), _ + _)) / sumLikelihoods //update the variances
}
```

2.2 Task 2

In order to find the right number of clusters represented by the variable K, we run the dataset version *"dataset-mini"* for K values ranging from 2 to 5, and then we compared and analyzed the results of the joint probability, and we obtained the following results:

Table 1: Log Likelihood for K values - epsilon = 10000

K	Log Likelihood	Iterations	Time(s)	Means	Variances	Weights
2	- 7.343745E7	40	134.113	176.59602993248083	69.21587884184495	0.5894883485618796
				163.8034513977316	33.163600503938774	0.41051165143811863
3	- 7.343769E7	20	89.896	163.79830566482494	31.880612106407792	0.2422127834328912
				179.2492327215712	54.124122305956696	0.42682259417293283
				166.67298438903939	46.86085449104776	0.3309646223941759
4	- 7.343752E7	20	110.152	177.95760633228184	59.13448809476239	0.18059120116989427
				181.2132819797651	50.25891482818247	0.10541156643088735
				164.49030635972613	36.40187179402625	0.4854072819086218
				176.1240203202091	62.035793187838976	0.22858995049059527
5	- 7.344704E7	30	187.475	162.85406079370586	22.538429882338217	0.19765676090036813
				181.1896208646975	51.12216832270957	0.23728105885713627
				171.58329741530895	76.89905010114953	0.19026468295166135
				169.16525602832272	62.499026634697174	0.18741557682943508
				169.77097013666227	66.58093017664632	0.18738192046139038

We observed that we got the highest log likelihood with a value of k=2 (more positive value) in comparison with the other K values. We assume then that the data is drawn from two Gaussian distributions. Analyzing the final variables obtained running the algorithm using the dataset "dataset-medium" in the Isabelle Cluster, we realized that the data **comes from a normal distribution of heights between man and woman**. Results: Mean: (178.1664,164.2533) Variance: (65.3929,34.3434) Weights: (0.5559,0.4440). We compared this results with the results obtained in OurWorldInData [1] and the global distribution of heights of woman and man in cm is (man,woman): Means: (178.4,164.7) Variances: (131.04 , 101.64), and we have seen that our results are closer with the official results of OurWorldInData.

2.3 Task 3

Did you use RDDs, DataFrames or Datasets for your implementation? Why?

The entire implementation of the algorithm was done using RDDs. The decision of using RDDs is

[1]https://ourworldindata.org/human-height , the distrubution used is not globally representative since it does not include all world regions

due that we have a dataset of unstructured data consisting in just real numbers. As we have seen during the lectures, if we posses a lot of structure in our data, there will be more opportunities for the catalyst optimizer of Spark to optimize multiple operations, in our specific case, the data contains no structure, so the catalizer would perform minimum to no optimization to the operations realized to the dataset.

In which parts of your implementation have you used partitioning? Did it help for performance? If so, why?

Partitioning was not used for the implementation of the project. Spark has three types of partitioning: default, range and hash partitioning, the last two types only work for pair-RDDs, which is not our case, since we are using normal RDDs when we are reading the dataset. Also, there is no shuffling operation in our implementation, which means that data is not moved considerably between RDD partitions, making the use of pair RDDs and partitioning useless.

Have you persisted some of your intermediate results? Can you think of why persisting your data in memory may be helpful for this algorithm?

Persisting in memory was used in two specific phases of the algorithm.

- **X RDD:** At the start, when reading the dataset and transforming it to a RDD using the map transformation, we persist the resulting RDD, since we will use this RDD to calculate the log likelihood and to calculate another RDD called gamma. Persisting will allow further accesses to avoid reloading the dataset and applying the map function.

- **Gamma RDD:** During the expectation step of the algorithm, inside the iterations loop, we calculate a gamma RDD that will contain the likelihood of a datapoint belonging to a single component divided by the sum of all likelihoods, this for all each datapoint in the dataset and for each value of K. The gamma RDD will be used latter in each iteration during the maximization step to calculate the new variances, weights and means, so persisting it will avoid recalculating the gamma RDD for each action in the maximazation step. It is important to consider that the gamma RDD will change each iteration, so we will un-persist the gamma RDD at the end of each iteration, this to avoid the probability of getting an Out-Of-Memory associated error.

No persisting the intermediate results of X and Gamma results in a slower execution of the algorithm due to the necessity of having to recalculate and apply transformations for the RDDs during each action executing inside the iterations loop.

3 Results and Comments

This section will explain briefly the results obtained running the algorithm implementation in the Isabelle Cluster.

Table 2: Log Likelihood for K values - epsilon = 10000 - dataset: dataset-medium

K	Log Likelihood	Iterations	Time(min)	Means	Variances	Weights
2	- 3.672189E9	90	42.4	177.173798	65.345554	0.555566
				164.056919	34.359697	0.444433
	- 3.672364E9	70	37.6	176.602683	70.048355	0.582024
				164.010162	34.377740	0.417975

During the first testing sloth in Isabelle, many problems were encountered generating the corresponding .jar file and executing it in Isabelle, after a set of tests, the generation of the .jar file was successfully and it could run without no problems in Isabelle, but the time ran out so we couldn't perform any significant test on it. During the second slot, the algorithm implementation was tested twice in Isabelle as show in the table 2, the dataset used was: dataset_medium that contained 1000000000 elements. The first row shows the first test, where some mistakes where spotted in the code, specifically in the generation of the gamma RDD, where the denominator was inside the for loop of K values, and it was slowing down not significantly the generation of the RDD. The second modification done was to put a control condition to calculate the new log likelihood, this to save a few seconds in each iteration, since we will require more than 10 iterations to reach a local maxima, we then set to calculate the log likelihood each 10 iterations. These modifications showed an improvement of 12 % shown in the table 2. It was possible to observe that the garbage collection time consumed up to 30& time of each task, which gave us insight on possible improvements in memory management. Also in the next figure we observe that the job that takes most time represents the fist action realized on each RDD, where the action that took the highest time to complete is the first aggregate that executes and generates the gamma RDD.

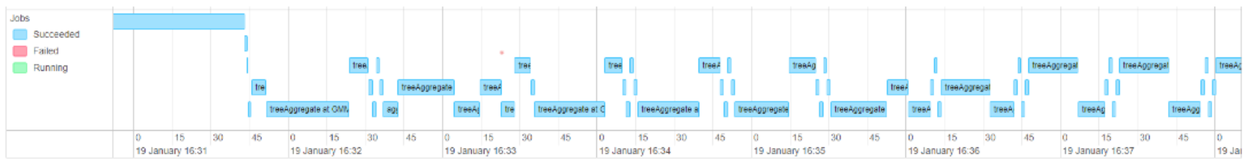

Figure 1: Spark Jobs - Dataset Medium - Epsilon: 10000