Distributed Mixture of Gaussian Clustering

Hongzhe Cheng, Yuzhou Wang

1. **Project Link**

   The following github link will be used for the code base, reports for this project.
   https://github.com/EricChengWOW/DistributedGMM

2. **Summary**

We are planning to implement the traditional Machine Learning clustering algorithm - Mixture of Gaussian in a distributed way. The potential parallelization in the problem includes data parallel for machine learning models, and SIMD processing for each distributed node.

3. **Background**

Clustering is a traditional important Machine Learning problem that focuses on grouping similar data points (samples, objects, etc) such that samples within the same cluster are more identical and with the similar features or characteristics.

The Gaussian Mixture Model uses the EM concepts for a general 2-step algorithm. The expectation step represents a soft assignment for each data point to a cluster. The function is defined as a likelihood function for the current clustering status.

In this step, the contribution of each data point could be perfectly parallelized, but we would need a global synchronization to gather all intermediate parameters and create the whole likelihood function.

The maximization step involves the typical Machine Learning concept of MLE (Maximum Likelihood Estimation). Thus, given the function result of the E step, we will simply use the derivation of each GMM variable to gain the cluster parameters (mean, variance and mixture proportion for each sub-distribution) for the next iteration.

From the distributed Perspective, this step either needs to be done sequentially with future parameter distribution to every node (parameter server analogy). Or we need all nodes to have the function, and compute the same next round simultaneously (ring communication analogy).

There are several benefits of clustering algorithms compared with classical clustering algorithms. Firstly, clustering algorithms provide a summary of the pattern of training data. One can get a visualization of the data's overall patterns by inspecting the data points near the clusters. Secondly, clustering algorithms are unsupervised. Training a clustering algorithm doesn't require labeled data. In most cases, large size labeled data are costly to acquire. Clustering algorithms overcome this issue. Lastly, clustering algorithms are generative methods. Given the clusters, one can generate a sample based on the model. Other models like logistic regression or decision trees.

Compared to kNN, GMM has several benefits. One is that GMMs can model complex data distributions with multiple peaks, while kNN assumes a simple Euclidean distance metric between data points. This makes GMMs more suitable for data with complex distributions, where there may be multiple overlapping clusters or subgroups. Another benefit of using GMMs is that they can handle missing data and incomplete datasets. GMMs use an iterative approach to estimate the parameters of the underlying distribution, which allows them to make predictions even when there are missing data points. In contrast, kNN requires complete datasets and cannot handle missing data.

The algorithm is as follows:

Prediction model: Mixture of K Gaussians distributions

Observed data: $D = \{x_1, x_2, ..., x_n\}$

Probability of a given data: $P(x) = \sum_{i=1}^{K} P(x|y = i)P(y = i)$

Model's parameters: $\theta = \{\mu_1, ..., \mu_K, \Sigma_1, ..., \Sigma_K, \pi_1, ..., \pi_K\}$,

where $P(x|y = i) = N(\mu_i, \Sigma_i), P(y = i) = \pi_i$.

Goal: find $argmax_\theta log(P(D|\theta))$

Assume we are using m workers.

Distribute D among m workers. $D_m = \{x^m_1, ..., x^m_{n^m}\}$.

E-step:

On each worker, calculate $R_{i,j,m}^{t-1} = P(y_j^m = i|x_j^m, \theta^{t-1}), s_i^m = \sum_{j=1}^{n_m} R_{i,j,m}^{t-1}$ for $i = 1, ..., K, j = 1, ..., n^m$. Send S_i^m to the server.

On server, calculate $R_i = \sum_{m=1}^{M} s_i^m$, and sent it to each worker.

On each worker, calculate $w_{i,j}^m = R_{i,j,m}^{t-1}/R_i$.

M-step:

On each worker, calculate $\mu_{i,m}^{t-1} = \sum_{j=1}^{n_m} w_{i,j}^m x_j^m$, and send it to the server.

On server, calculate $\mu_i^t = \sum_{m=1}^{M} \mu_{i,m}^{t-1}$. This is parameter μ on round t.

On server, calculate $\pi_i^t = R_i / \left(\sum_{m=1}^{M} n_m \right)$ and sent to workers. This is parameter π on round t.

On each worker, calculate $\Sigma_{i,m}^{t-1} = \sum_{j=1}^{n_m} w_{i,j}^m (x_j^m - \mu_i^t)^2$, and send it to the server.

On server, calculate $\Sigma_i^t = \sum_{m=1}^{M} \Sigma_{i,m}^{t-1}$. This is the parameter Σ on round t.

Now, return to step 1 for new iteration, until reaches a certain number of iterations.s

4. **Challenge**

   Workload:

   - Large scale dataset that can not be stored in a single computing resource
   - Complex computation required with huge amount of parameters in the EM algorithm

   Constraints:

   - In order to mimic the situation for multiple computing nodes or servers that each contains a subset of the data, we will use the message passing model which creates communication overhead
   - The challenge on the sequential portion inherent in the algorithm that could harm the speedup for the system.

5. **Resources**

We will start the project from scratch on the GHC machine with MPI to first create a valid model that behaves correctly. Then we will move to PSC with more available threads to measure the scalability and robustness of our algorithm.

We do not need to refer to papers as the algorithm is already clear, but we do need to find benchmark datasets that we can use for clustering. In order to utilize the system on real world data, we might also need to encode or transform the data such that it could be processed by our system.

6. **Goals and Deliverables**

   PLAN TO ACHIEVE:

   - Create the correct baseline GMM EM algorithm for clustering
   - Parallelize the algorithm with MPI to separate nodes/threads.
   - Benchmark the performance, and create data visualizations

   HOPE TO ACHIEVE:

- Try to implement ring communication version for comparison with the parameter server
- Add SIMD parallelization to each node for further utilization of the computing resource

Demo:

- The demo would contain the image illustration of how our algorithm performs and successfully cluster the dataset
- The demo would presents the speedup graphs we have been creating for the labs in this course

Analysis:

- We hope to see the speedup for parallelizing the GMM clustering problem with high count of nodes
- We hope to identify the bottleneck of the system for the potential non-linear speedup that we could have
- We hope to measure the detailed work runtime, and use that to further confirm the Amdhal's Law

7. **Platform**

We would like to use the GHC machine for development and PSC machine for final performance benchmarking since we want to test the scalability of our algorithm to a larger scale.

We would develop the algorithm in C++ with access to MPI so that we can better mimic the behavior of multiple computing/storage nodes in the real world large scale situation.

8. **Schedule**

- **4/3 - 4/9:** Get more detailed information about all the computation process, the parallel algorithm, obtain the dataset, and starting to implement the baseline algorithm
- **4/10 - 4/16:** Finish implementing the baseline GMM algorithm, come up with data visualization for how the algorithm clusters
- **4/17 - 4/23:** Start to implement the distributed algorithm with parameter server
- **4/24 - 4/30:** Finish implementing the parameter server version *Try ring communication, SIMD if time allows
- **5/1 - 5/5:** Benchmark the performance, and create final analysis *Try ring communication, SIMD if time allows