Distributed Systems
Parellel K-means

by: Andriy Kusyy

Introduction

K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells [1]

Sequential k-means

Let us have a training set X = $\{x^1, x^2, ..., x^m\} \in R^m$ of m training examples.

1. Initiate cluster centroids $\mu^1, \mu^2, ..., \mu^k \in R^m$ randomly.
2. Repeat until convergence:
    a. For each learning example, $x^{(i)}$, find the nearest centroid $c^{(i)} = \arg\min_j \| x^{(i)} - \mu^{(j)} \|$
    b. For each cluster, set its centroid to the point that is the average of all learning examples belonging to this cluster

$$\mu^{(j)} = sum(x^{(i)}\{c^{(i)} == j\}) / sum(1, \{c^{(i)} == j\})$$ [2]

In this algorithm k is the number of clusters we want to break into the study sample. Each cluster is characterized by μ-centroid. Thus, we repeat two steps in a loop: first assign each study example to a cluster with the nearest centroid, and the second one - define centroid as the mean value of all points in this cluster. Generally, centroids are initialized before the first cycle in a random manner. However, k-means converges to a local optimums, some of which obviously are not solutions to the problem (imagine two centroids, the first of which was initialized in the center of the entire sample, and the second - very far from all points).

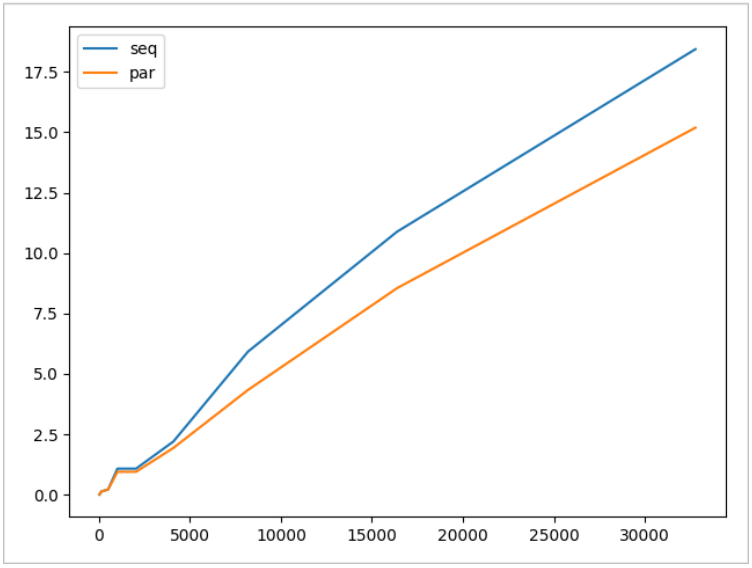
Therefore, in practice, a good solution is to initialize each centroid in the coordinates of random points from the training sample. can constantly "reassign" points from the cluster to the cluster on each loop. The deviation function will then remain minimal. You can try to come up with such a placement of points on your own

Parallel k-means

One of the most popular approaches to parallelization of k-means is based on a MapReduce paradigm and is dealing with the most complex from the computational point of view part of the sequential k-means algorithm - determining the closest cluster for each data point.[3][4]

1. Initiate cluster centroids $\mu^1, \mu^2, ..., \mu^k \in R^m$ randomly.
2. Split training data into chunks(number of chunks is equal to number of cores)
3. Repeat until convergence:
    a. ***(MAP) For each chunk:*** For each learning example, $x^{(i)}$ in a chunk, find the nearest centroid:
        $$c^{(i)} = \arg\min_j \| x^{(i)} - \mu^{(j)} \|$$
    b. ***(REDUCE)*** Combine results of a map step into a single array
    c. For each cluster, set its centroid to the point that is the average of all learning examples belonging to this cluster
        $$\mu^{(j)} = sum(x^{(i)}\{c^{(i)} == j\}) / sum(1, \{c^{(i)} == j\})$$

My implementation[5] was done using Python 3. For the sake of performance, I decided not to split the Map and Reduce functions into separate, so they are combined in one. On the random set after a few thousand points distributed algorithm is showing constant speedup from 9% to 19%. Taking into account that this is Python implementation with threads for parallelization emulation - such gain is showing good qualities of an algorithm. In the original paper, the speedup gained was around 25%



References

[1] *https://en.wikipedia.org/wiki/K-means_clustering*

[2] *cs229.stanford.edu/notes/cs229-notes7a.pdf*

[3]https://pdfs.semanticscholar.org/8c79/29fd589ac40421ac296daf45fa

[4]https://www.cs.ucsb.edu/~veronika/MAE/parallelkmeansmapreduce_zhao.pdf

[5] https://github.com/andriyka/parallel-distributed-k-means-algorithm