ONLINE DISTRIBUTED EIGENSPACE COMPUTATION

Akhtyamov, Timur
Skolkovo Institute of Science and Technology
Moscow, Russia
timur.akhtyamov@skoltech.ru

Nikolaeva, Anna
Skolkovo Institute of Science and Technology
Moscow, Russia
anna.nikolaeva@skoltech.ru

Mohamed, Samir
Skolkovo Institute of Science and Technology
Moscow, Russia
samir.mohamed@skoltech.ru

December 21, 2019

1 Abstract

We introduce an implementation to improve the performance of principal components analysis (PCA) computation for large datasets. The idea is to introduce two concepts for PCA computing. The first concepts of an online algorithm, the algorithm that is able to work with data that comes in a sequential manner. The second is the concept of the distributed algorithm, a kind of parallel algorithm that is designed to be run on hardware consisting of interconnected processors. Thus, we are getting the online distributed algorithm for PCA. As numerical results, we are going to show how distributed online algorithm decreases PCA computation time for one of example datasets and how it can potentially affect on ML model quality.

2 Introduction

Principal components analysis is a popular dimension reduction technique. It extracts a K dimensional subspace that accounts for most of the variation in the data, revealing its low-dimensional structure. Thus, it allows reducing the data dimension in machine learning tasks such as regression analysis, k-means clustering, etc. However, having a large dataset, it can take several hours to compute. Fortunately, it is possible to speed up computation buy disturbing it between nodes. Within the work, we decided to check the performance of one of the latest proposed algorithms, the so-called online distributed algorithm [1], on the popular dataset CIFR-10[2]. Our scrips are available on GitHub [1].

3 Algorithm

The online distributed approach is supposed that data are represented in a sequential manner on each node. Basically, we can consider two main steps. In the first step, the incoming batch of the data is further split into subsets that are going to be allocated in different computation nodes (slaves). Each node computes a low-rank approximation of the corresponding subset, followed by a local aggregation step to obtain an estimation of the principal eigenspaces of the current batch. Then, the final result is obtained by passing the intermediate batch results to the center (master) that aggregates across all batches.

[1] https://github.com/TimeEscaper/distributed_eigenspaces

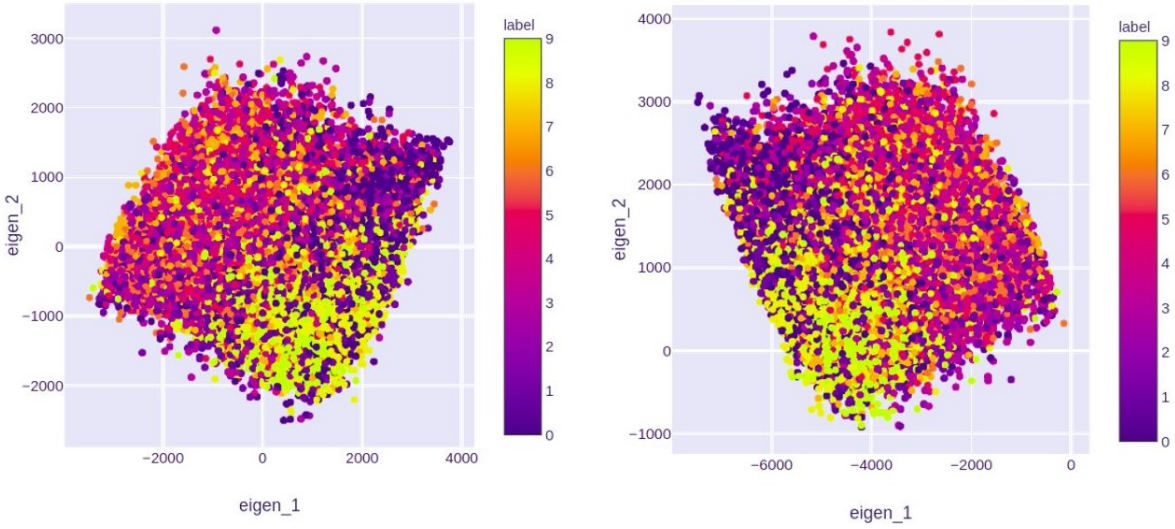
In order to identify K subspace, one can compute K leading eigenvectors $\{\tilde{v_j}(T)\}_{j=1}^{K}$ of the matrix $\hat{\Sigma} = N^{-1}\sum_{i=1}^{N} X_i X_i^T$, where $\{X_i(t)\}_{j=1}^{N}$ is data, represented in streaming samples, the following algorithm [1] allows doing that in online and distributed mode. Denoting $N = Tmn$ as the total number of samples, and $\{X_i^l(t)\}_{i\in[n]}$ as the batch of data that becomes available to a node $l \in [m]$, at time $t \in [T]$ (here $1 \leq t \leq T$ is time interval). In the first round of aggregation all local nodes contribute to obtain the approximation $\bar{\Sigma}(t)$ at time t. Then, the final estimate across all data batches, $\tilde{V}_K(T)$ is calculated by the fusion center. The communication cost for the local processors is divided by the number of time steps; so it is $O(NKd/(nT))$[1].

Algorithm 1: Online distributed PCA

1. Initialize $\tilde{\Sigma}(0) = 0_{dxd}$;
2. **for** $t \leftarrow 1$ to T **do**
3. **Input:** online data $\{X_i^l(t)\}_{i\in[n],l\in[m]}$;
4. **for** $l \leftarrow 1$ **to** m **do**
5. Compute $\hat{V}_K^{(l)}(t)$; the K leading eigenvectors
6. of $\hat{\Sigma}^{(l)}(t) = n^{-1}\sum_{i=1}^{n} X_i^{(l)}(t) X_i^{(l)T}(t)$
7. Compute $\bar{V}_K(T)$; the K leading eigenvectors of
8. $\bar{\Sigma}(t) = m^{-1}\sum_{i=1}^{m} \hat{V}_K^{(l)}(t)\hat{V}_K^{(l)T}(t)$
9. Update $\tilde{\Sigma}(t) = \tilde{\Sigma}(t-1) + T^{-1}\bar{V}_K(t)\bar{V}_K^T(t)$;
10. Let $\{\tilde{v_j}(T)\}_{j=1}^{K}$ be the top K eigenvectors of $\tilde{\Sigma}(T)$;
11. **Output:** $\tilde{\Sigma}_K(T) = (\tilde{v}_1(T), ..., \tilde{v}_K(T)) \in \mathbb{R}^{dxK}$

Where both of lines 5, and 6 are for slaves and the rest of lines are for master side.

The proposed algorithm was implemented using python language. In order to check its correctness, we compared its output on the test data with results obtained by the scikit-learn library. The results are presented in figure [1].



(a) (b)

Figure 1. Eigenspace obtained from (a) our implementation and (b) using scikit-learn library, where the two figures is nearly the same and flippy horizontal to each other this is due the the difference in feature engineering in our implementation and scikit-learn library implementation

4 Setup of experiments

From the derived algorithm, we can see that it is possible to implement the system that can process data in a distributed way - it means that parts of the data can be processed simultaneously on multiple computational nodes. Under "computational node" we mean a process with its own context, and multiple nodes can be both on the same machine or

different machines. And since the obtained algorithm is online, our nodes can process data online - they can process smaller batches of data sequentially instead of processing one larger batch. This very actual for the clusters with machines with different computational power - if node processed its batch faster, it can ask for a new batch instead of keeping waiting and doing nothing. This way may dramatically increase the efficiency of the system.

Thus, we propose the following architecture of the system:

1. Single master node
    - 1.1 Splits dataset into batches and sends batches to slaves;
    - 1.2 Receives computed eigenvectors from slaves;
    - 1.3 Performs aggregation of computed eigenvectors and computes the final result.
2. Multiple slave nodes
    - 2.1 Receive batches from master;
    - 2.2 Compute eigenvectors and send them back to master.

A crucial part here plays how we organize communication between a master and a slaves (figure 2). In our project, we chose the MQTT (Message Queuing Telemetry Transport, figure 3) protocol that is widely used in distributed systems. MQTT is an asynchronous protocol that is based on publisher-subscriber architecture - publisher can publish a message on the channel that is actually a message queue, and subscribers of the channel will receive the message. The core component in MQTT protocol implementation is the message broker - a server that manages the channels and interacts with publishers and subscribers. A broker can manage a channel in multiple ways: it can broadcast the message to all subscribers, it can immediately assign a new message to one of the subscriber or it can assign a message to the first subscriber that becomes available (processed its previous message). The last approach is suitable for us.

There are many MQTT message brokers available - this protocol is supported by projects like RabbitMQ, Apache Kafka, Apache Spark and etc. We decided to use RabbitMQ, because this message broker is counted as lightweight and efficient, and it is easier to set up. For working with MQTT in Python we used Pika library.

Thus, in our system, we have two channels: the slaves' channel where single master publishes a message with batch to multiple slaves, and each new message automatically assigned to the first slave that becomes available, and the master channel where multiple slaves publish messages with computed data to a single master.

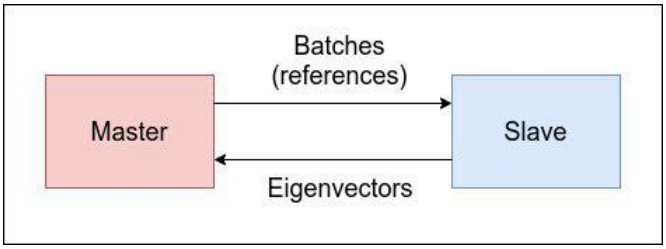


Figure 2. Master-slave architecture where the master interact with a slave through a channel(queue) and the slave interact with the master through another channel

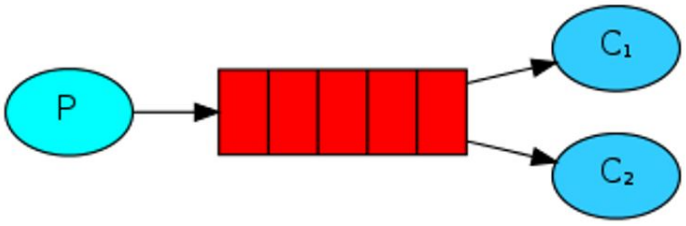


Figure 3. Work queue illustration such that producer P write data on the red queue and the consumers C_i read the data from the queue(en-queue)

Note that in master to slave messages we send not the batch itself but the "reference" to the batch - we assume that nodes have local copies of a dataset on machines. This dramatically decreases message size and helps to avoid huge delays introduced by networking. Also, in a potential real system, the distributed file system like Ceph can be used to share the dataset between nodes.

In our project, we used one desktop PC and one laptop as a computational base for our system.

5 The experiments

We performed two types of experiments that show how the system can speed up computations and when it can become less efficient.

In the first type of experiments, nodes are run on a single machine (desktop PC). Here we performed computations with a different number of nodes and different batch sizes, and obtained the following results (figure 4).

Figure 4. Slaves at one machine

We can make the following conclusion: 1. After some point, the increment of nodes does not speed up computations. This happens because all nodes are the processes on one machine, and the operating system can not run all the processes truly parallel (especially if we have more processes than a number of processor cores). 2. Bigger batch size performed much faster than smaller. The dataset that we used is not the most huge and complicated one, and because of that process one large batch still becomes faster than to compute multiple smaller batches because of network delays and specificity of the operating system.

In the next experiments, we used two machines - desktop PC and laptop. Now multiple nodes can be run on different machines. We obtained the following result (figure 5).

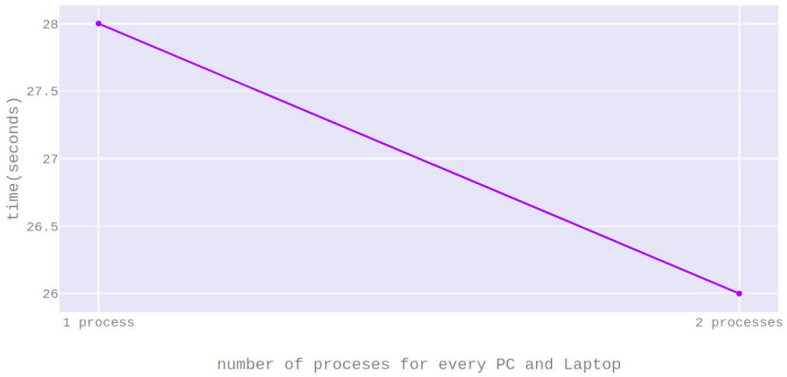


Figure 5. Processes on different machines with batch size: 200

We can see that for the same batch size computation became faster than running two nodes on a single machine. This happens because now nodes work truly parallel. And increment of processes is affected by the same things as in the case of a single machine.

6 Conclusion

We have implemented the online distributed algorithm for the estimation of principal eigenspaces. The correctness was checked by comparing our result with the result obtained from the python library. Then we have organized series of experiments approving the effectiveness of the proposed algorithm. Compute PCA can be done quickly on large scale datasets using the distributed system.

7 Acknowledgment

The general responsibilities of each team member were the following:
Samir: Implementation of the core algorithm,
Timur: Building the network,
Anna: Organization of multiprocessing.

References

[1] D.Tarzanagh, M.Faradonbeh, G.Michailidis Online Distributed Estimation of Principal Eigenspaces. 2019.

[2] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.