University of Groningen

Project Report

Group 32

Scalable Clustering

Authors:
Lennart Faber (S2500523), Rohit Malhotra (S3801120) &
Andreas Yelasis (S3917738)
March 31, 2019

1 Introduction

For this project, we chose the topic of clustering. More specifically, we will compare various clustering implementations on their scalability by using them to cluster a data set consisting of news articles. Doing research on this subject fits in the digital landscape of the present day: with the amount of real and fake news articles written online increasing at enormous rates, it can be useful to perform some form of automatic analysis on them in a scalable way. We will compare the scalability of prototype based clustering algorithms and a density based clustering algorithm.

2 Clustering Algorithms

We have evaluated two different clustering algorithms. A well known algorithm is the K-means algorithm. We have tested two implementations: one that comes with the machine learning library of Apache Spark and one that we have implemented ourselves. A lesser known, but also much used density based algorithm is the mean-shift algorithm, of which we made an implementation ourselves. We will discuss both algorithms and their implementations in the following sections.

2.1 K-Means [1]

The K-means algorithm classifies documents into K groups of similarity as follows:

- Initialise K points, called means, randomly
- Step 1: Classify each point to its closest mean
- Step 2: Update the mean's coordinates, which are the averages of items categorised to each K.
- Step 3: Repeat until convergence or until the maximum set amount of iterations has been reached.

In our own implementation, we aim for scalability by distributing step 2 and 3 of this algorithm over several workers. After initialising K prototypes at random points, the task of calculating distances to each data point from each of these prototypes will be mapped (which can be executed in parallel), and in the same mapping we will calculate the nearest prototype and emit the resulting prototype number and point as a key-value pairs. The result will be of the form `(closest prototype number, distance, point)`. Next is the reduction step to calculate the new means. We reduce by key (cluster number) of which the value is a tuple that consists of the sum of the feature vector and number of feature vectors in each key. This step is associative, meaning that there is no bottleneck in which the reduce waits for the mapping to finish. We repeat this

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

map and reduce until the maximum specified number of iterations has been reached.

2.2 Mean-shift [2]

Mean shift clustering is a probability density based clustering algorithm. The idea is to find an attraction basin for each point, with points in the same attraction basin ending up in the same clusters. The hyper parameters are window size and type of window, which can be Gaussian, circular etcera. Our parallel implementation of the algorithm is based on following steps:

- Initialisation: Initialise N windows with centres at N points. Every point gets its own window. N is number of points in the data set. Then for each window:
- Step 1: Calculate the mean of the points inside the window.
- Step 2: Shift the centre of window to the new mean.
- Step 3: Repeat step 1 and 2 until convergence.

In order to implement this in map reduce framework, we start by taking crossjoin of our dataframe with itself. By doing this, rows of new dataframe will represent a set of centre of each window and each point. Then we map these rows to calculate distance between centre of window and the point and filter out only those points whose distance is less than the window size, this is a map step which can be executed in parallel and it emits id of the point as key and point vector as value. Then we use reduce by key to calculate new centre of window which will be the mean of intermediate values with same key and this reduce step is done in same way as described above in K-means. These steps are done until specified number of iterations.

2.3 Data & Algorithm Evaluation Metric

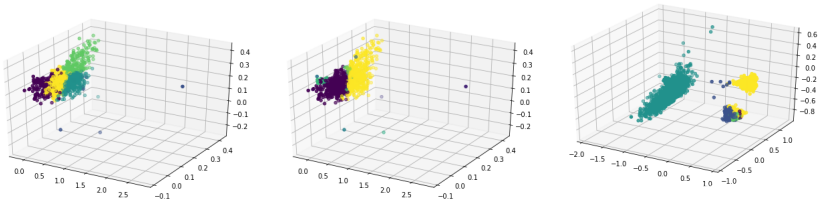
In our news article data set we have 6827 articles of variable length. As a preprocessing step we convert our data point, which is a text document, to a feature vector. This is done using the state of the art algorithm called Word2Vec. This is a neural network based algorithm that computes an N-dimensional vector representation of each word in the data, after which a document is represented as the sum of vectors of words in the document. We use the Spark ML-library implementation of Word2Vec to train our data and save the computed model in the HDFS.

We use visualisation as an evaluation metric of the quality of clustering of our algorithms. First we convert our N-dimensional feature vector to a 3 dimensional feature vector using Principal Component Analysis, after which we plot 1300 samples points with different colors representing the clusters. For

[2] https://en.wikipedia.org/wiki/Mean_shift

K-means we specified the number of clusters as 5 and for mean-shift we saw that after 14 iterations the number of clusters were not changing anymore (with window size of 0.12). Plots are shown in figure 1. As we can see there is mainly one cluster. With the given window size, the results of means shift show two different clusters, where as in K-means we had specified number of clusters before-hand so the result shows 5 clusters. In conclusion, for our sample, where we did not know the distribution of the data we can apply mean shift clustering to determine the number of clusters after which K-means clustering can be applied to cluster all data points if after visualisation we find that we have circular clusters, and otherwise we can use some other algorithm like GMM. In order to find number of clusters, sampling should be done in a uniform way so that we can cover maximum variation in the distribution. In our sample, we have not covered the full distribution, so the outliers are shown as clusters themselves.

Figure 1c shows different clusters found by the K-means algorithm. We can see the algorithm was able to separate the green cluster quite well.



(a) K-means (b) Mean Shift (c) Result of K-means on all data points.

Figure 1: (a) shows result for K-means clustering for 1300 sample points with k=5. (b) shows result for mean shift clustering of the same points. We can clearly see with the given window size all outliers are considered as different clusters.

3 Technologies and Architecture

3.1 Apache Spark [3]

We chose to use Apache Spark as our framework for parallel computing. Apache Spark is an open-source, distributed, general-purpose cluster-computing framework owned by the Apache Software Foundation. Spark provides an interface for programming entire clusters with capabilities of fault tolerance and data parallelism. Apache Spark is the successor of HADOOP, which also used MapReduce, but in a less efficient way. HADOOP saves tasks to the hard disk while

[3] https://spark.apache.org/

Apache Spark performs in-memory computation, making it (supposedly) 100 times faster.

Other reasons for going with Apache Spark is that it offers an interactive API and extra features on top of its core. Some of these are Spark SQL, Spark Streaming and Spark GraphX, each of which could be used in future projects. A final reason for working with Apache Spark is that it allows for coding in Python, a language that we prefer over a more difficult-to-learn language like Scala.

3.2 MongoDB [4]

We choose MongoDB as our data source for batch processing and as sink for updating prototypes in case of data streams. We choose MongoDB because it is a distributed No-SQL database, which is best suited for our data that contains a variable number of attributes. Another reason is that Spark provides official support for MongoDB as data source via the Spark MongoDB connector.

3.3 Kafka [5]

For implementing streaming data processing we needed to implement a producer-consumer system, where the task of the producer is to put the documents into a Kafka queue. We used the Kafka queue because Spark officially provides packages to use Kafka as a streaming data source. We used direct stream, which maps every Kafka partition to an RDD partition and checks for updates on the stream periodically. Also, Kafka is highly scalable without any downtime and provides a very high throughput.

3.4 HDFS [6]

We used the Hadoop Distributed File System (HDFS) to store our Word2Vec model trained on batch data. To process streaming data, this model is loaded from HDFS to get produce the vector representation of the new document, which is then passed on to the clustering algorithm. We chose this setup to make the model accessible to all nodes, because in cluster mode deployment we do not know on which machine the process will be deployed.

3.5 Architecture and Data Flow

Figure 2 shows the architecture and data flow of our implementation.

[4] https://www.mongodb.com/
[5] https://kafka.apache.org/
[6] https://hadoop.apache.org/

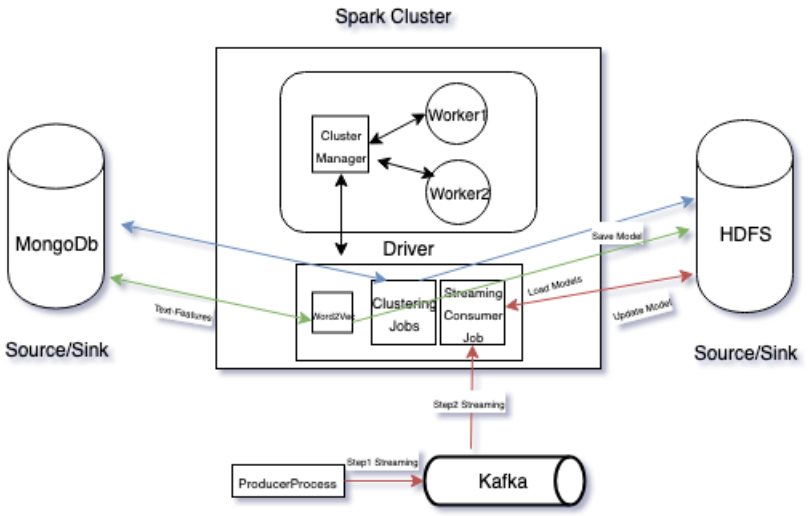


Figure 2: This figure shows the architecture of our cluster and flow of data. Red arrows show flow of data in case of streaming data. Green arrows show flow of data for pre-processing(text to feature). Blue arrows show flow of data in clustering job.

4 Infrastructure

In order to test the scalability of our algorithms, we used the Google Cloud[7] platform. We formed clusters by taking "n2-standard-2" machines of their Compute Engine service. All the components of our architecture are running as Docker[8] containers. We chose Docker Swarm [9] as our orchestrator, because we find it simple to use with Docker containers as we can use docker-compose files to directly deploy our services to the stack.

As we are using Python, Spark does not support the "cluster" deployment mode, which would allow our driver process to reside on any worker inside the cluster resulting in less network overheads. To overcome this we used a separate driver container on the cluster which is in the same network, and we use this to submit our jobs.

Other containers are used for the Spark standalone cluster manager and its workers. We assigned 1 CPU core to each worker, with 3GB of memory. Another container runs Kafka, and another one runs Zookeeper (used by Kafka). One container is running our MongoDB. We also have an HDFS cluster running, which makes as many datanode containers as number of machines in the swarm cluster, and one container running the namenode. In order to visualise our clustering results we are running one container containing Jupyter and Spark.

[7]https://cloud.google.com/
[8]https://www.docker.com/
[9]https://docs.docker.com/engine/swarm/

Algorithm	Timing (1 worker)	Timing (2 workers)	Timing (3 workers)	Timing (4 workers)
K-means (Ours)	58s	58s	57s	1.1min
K-means (SparkML)	1.6min	1.4min	1.2min	1.3min
MeanShift	2.7min	2min	1.3min	1.4min

Table 1: Change in computation time with increase in number of workers for different implementations of K-means and Mean Shift. Timing for Mean Shift is for data sample of size 300.

The containers are distributed among 6 machines.

5 Results

Table 1 shows change in computation time for the K-means and mean shift algorithms after increasing the number of workers. It can be noted that there is an increase in time for every algorithm after adding a fourth worker. The reason for this is that in our cluster we were only capable of hosting 6 machines, so when we added a new container it had to share resources with another worker container. This effect verifies the fact that a slow machines affects performance more than a dead machine. However, adding up to three workers did decrease computation time, so we can state that our algorithms are horizontally scalable. Our implementation of K-means has better computation time than SparkML's K-means which could be due to the fact that SparkML's implementation requires additional pre-processing of feature vectors.

6 Discussion

Our implementation of mean shift clustering is of $O(N^3)$ and given a dataset with N points we require to form a data set of n^2 rows. This means clustering in a real big data environment will require cluster with many nodes if the mean shift algorithm is used. We can further optimise the algorithm by following standard optimisation steps. We had also started to work on our own algorithm[10], which was a combination of mean shift and graph based clustering. We will further work on that and implement mean shift clustering with standard optimisation techniques.

To conclude, we can say that for planet scale use-cases we can use the optimised means shift algorithm on a data sample to find the required number of clusters after which we can use this number as a parameter for the K-means algorithm to cluster the entire data set.

[10]We have not verified if it will be totaly new.