# Comparative analysis between sequencial and distributed version of the Twitter sentiment analysis in Hadoop framework

Francesco Bongini
bongini.francesco@gmail.com

## Abstract

*In the era of the Internet, social media has become an integral part of modern society. Social networks like Facebook, Instagram and Twitter get huge amount of data each day. Traditional systems are useful in working with data of small size, but they can't manage such a huge amount of data from social networks. Hadoop framework stems from the need to process large amounts of data. In this paper, I compare the sequential and distributed version of the Twitter sentiment analysis. The aim is not to provide a perfect classifier, rather we want to show the speedup reachable from the distributed version with Hadoop. From the experimental results we see that hadoop gets big benefits regarding the speed and the amount of data processed in the same quantity of time.*

## 1. Introduction

Twitter is one of the most famous social network in the world. Every day millions of tweets are being posted. Sentiment analysis allows us to know a person's behavior toward a particular topic. Sentiment analysis is extracting the perception of people towards a particular issue, brand and scheme from textual data. In this article, sentiment analysis is performed on tweets in order to classify whether a tweet is positive or negative. The dataset contains 1.6 millions of tweets with their relative sentiment, "0" for negative tweets and"1" for posititve tweets. The sentences are split into words. This step is also called tokenization. These tokenized words are taken as an input for identifying opinion words.
Sentimental analysis considers two phases: training and classification, performed both using LingPipe library. We consider two versions: the sequential version and the distribuited, which uses the MapReduce framework in the classification part. The whole code is written in Java. The main core components of Hadoop framework are MapReduce and HDFS. MapReduce is a programming model for processing larger data sets and HDFS is a Hadoop Distributed File System that stores data in the form of memory blocks and distributes them across clusters.

## 2. Training the model

LingPipe library allows to train the model very easily. In this phase, the algorithm learns from some examples how recognize negative and positive tweets. The approach is therefore supervised. The training phase is sequential because each tweet must be trained by a single model. In particular, parallel version would need a critical section to allow threads to access the shared memory. In order to guarantee the mutual exclusion, we lose the benefits of the parallelization.
The model is trained in a sample of tweets in order to avoid overtraining and long waiting times. Here is the pseudocode:

**Algorithm 1** Training the model

```
Inizialize classifier;
for each tweet do
    classifier.handle(tweet)
end for
saveModel()
```

The object tweet is defined as his sentiment (negative or positive) and the text. Each tweet read is used to train the model. The classifier is then stored in "model.txt" and it will be used to classify new tweets.

## 3. Classification

The classification phase consists in classifing new tweets, not always used in the training part. In particular, we consider the sequential and the distributed version.

### 3.1. Sequential version

Sequential classification is very easy. Here is the pseudocode:

**Algorithm 2** Sequential classification

```
model=LoadModel()
pos=0
neg=0
for each tweet do
    classified=classify(model, tweet).bestCategory()
    if classified==“0”
        neg=neg+1
    else
        pos=pos+1
    end if
end for
return pos, neg
```

The program uses the model trained before to classify each tweet. It returns the number of tweets classified as positive and negative .

### 3.2. Distributed version

Distributed version is performed with MapReduce framework. MapReduce is a processing technique and a program model for distributed computing based on Java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Here is the pseudocode of the map function:

**Algorithm 3** Map

```
model=addCacheFile(model)
classified=classify(model, tweet).bestCategory()
return classified, 1
```

In order to improve the performances, the model file is added to the distributed cache. In this way, the file becomes available to all map tasks. The type of the tweet text is Text while the value is IntWritable. These types are the Hadoop variant of Integer and String which have been optimized for serialization in the Hadoop environment. An integer would use the default Java Serialization which is very costly in Hadoop environment. The map function classifies the tweet and returns the sentiment classified and the value 1.
The output obtained is a data set of tuples (key/value pairs). The reduce function takes the output from a map and counts the number of positive and negative tweets classified.
Here is the pseudocode:

**Algorithm 4** Reduce

```
count=0
for each tuple with same key k do
    count=count+1
end for
return k, count
```

With multiple reducers, we need some way to determine the appropriate one to send a (key/value) pair outputted by a mapper. The partition phase takes place after the Map phase and before the Reduce phase. In this phase, all the values for each key returned from the mapper are grouped together, making sure that all the values of a single key go to the same reducer.
The distributed version is performed with Amazon EMR.

### 3.3. EMR

Amazon Elastic MapReduce (EMR) provides tools for big data processing and analysis, across a Hadoop cluster of virtual servers on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3). A cluster is a collection of nodes. A node is a process running on a virtual or physical machine. In the following analysis, each node has a r4.xlarge istance with four cores and 30.5 GB of RAM. The data sets are stored in Amazon S3. In the analysis I use a master node to manage from 2 to 6 slave nodes.

## 4. Speedup analysis

Speedup is an index that measures the relative performance of two systems processing the same problem. It is defined as:

$$S_p = t_s/t_p$$

where P is the number of processors, $t_s$ is the completion time of the sequential algorithm and $t_p$ is the completion time of distributed version. Here is some speedup types:

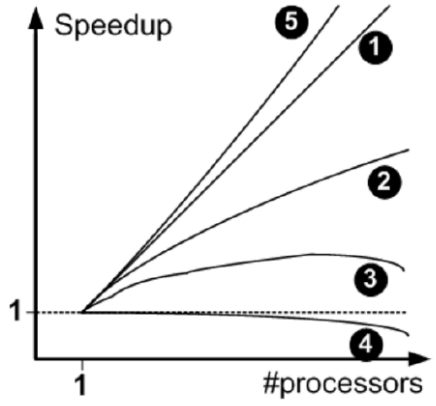


Figure 1. Speedup types

Types:
1. Linear speedup
2. Sub-linear speedup
3. Speedup with an optimal number of processors
4. No speedup
5. Super-linear speedup

In order to analyze the speedup, different examples are considerated with Amazon EMR. For the sequential version, I just used a r4.xlarge istance in Amazon EC2.

### 4.1. 1000 tweets

Hadoop is not a good solution for small data. In this first example the model classifies a sample of 1000 tweets. We can see that the sequential version is faster than the distributed one.

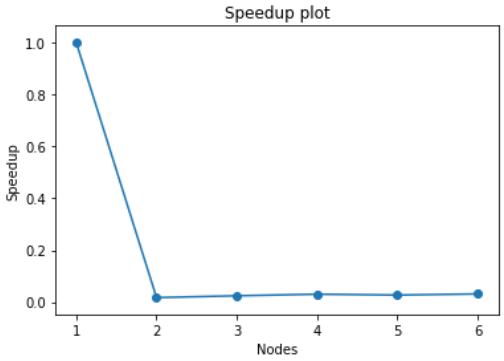


Figure 2. Speedup 1000 tweets

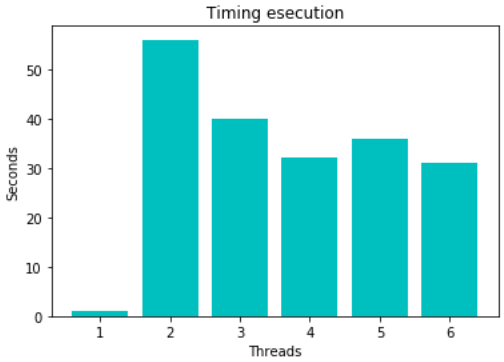


Figure 3. Barplot 1000 tweets

The reason is that there is always some initial delay while running MapReduce jobs, for example checking input/output paths, split creation, map creation, management of the tasks etc. In this case sequential version is preferable.

### 4.2. 1.6M tweets

We consider now the classification of the entire data set of 1.6 millions of tweets.

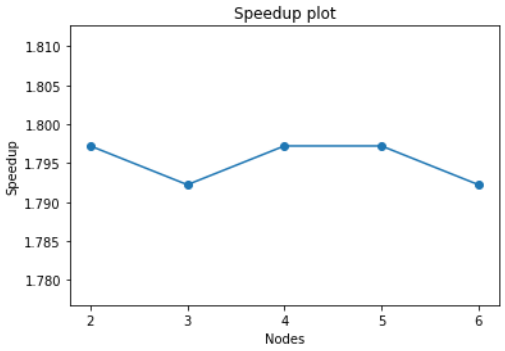


Figure 4. Speedup 1.6M tweets

As we can see Hadoop has not a good performance also this time. By default, in fact, EMR considers just 2 mappers if the input size is only one file. In addiction, time to manage and move data to the distributed environment is required.

### 4.3. 1.6M tweets in 5 input splits

In order to improve the performances, original data set is divided in 5 input data. Each split becomes input of each mapper and will be stored in the slave data node. The number of input data determines the number of mappers.

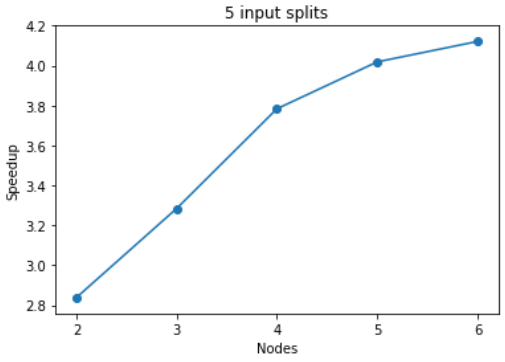


Figure 5. Speedup 1.6M tweets

Finally we note good improvements. The reason is that for each input split Hadoop creates one map task to process records in that input split. That is how parallelism is achieved in Hadoop framework.

### 4.4. 1.6M tweets in 10 input splits

Let's consider 10 input splits in the original data set.

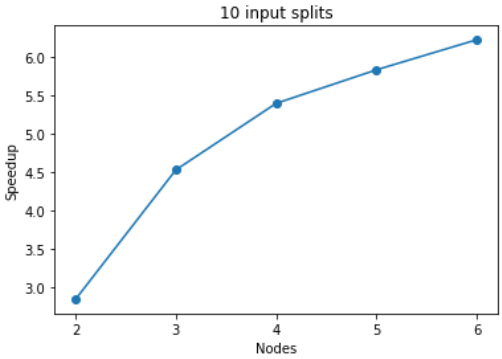


Figure 6. Speedup 1.6M tweets

With 6 nodes we achieve 6 of speedup. There is the same trend of the previous example but with speedup higher.

### 4.5. 4.8M tweets in 10 input splits

Hadoop works well with huge amount of data. In order to reduce the impact of the initialization time, the data set is tripled. The new data set contains 4.8 millions tweets.

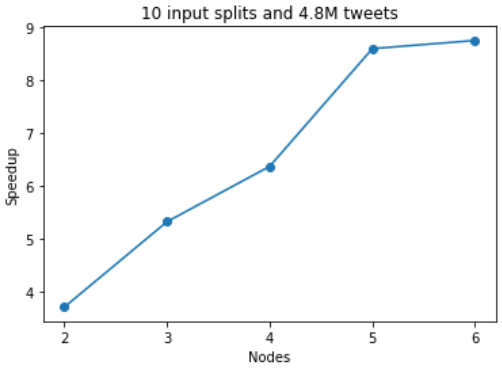


Figure 7. Speedup 4.8M tweets

In fact, we can see that using 6 nodes, Hadoop allows us to obtain a speedup of almost 9, rather than 6 in the previous example. This example dimostrates us the importance of choosing the number of maps.

### 4.6. 4.8M tweets in 20 input splits

In this last example, we consider 20 input splits. As we can see, we obtain still better performances only using more then 5 nodes.

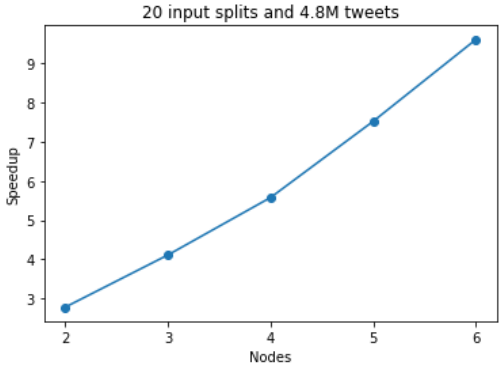


Figure 8. Speedup 4.8M tweets

The trend is linear. Increasing the number of nodes we expect an increasement of the speedup. The program seems to be scalable.

## 5. Conclusions

In this paper I compared the sequential and distributed version of the Twitter sentimental analysis with Hadoop framework. For data set big enough, distributed version is preferible for the time spent and the amount of data processable in the same time. In some cases it achieved a speed-up of 9 or more with 6 nodes. Hadoop has dimostrated to be a valid solution for processing of huge amount of data.