Machine Learning and Data Mining with Apache Spark

Benjamin Fovet, Maxime Gasque, François Horel, Héloïse Hourquebie,
Abderrahman Lahjouji, Anass Seddiki

{bfovet, mgasque, fhorel, hhourquebie, alahjouji, aseddiki}
@enseirb-matmeca.fr

Abstract. This paper deals with the concepts of ***machine learning*** and ***data mining*** social networks, which are increasingly useful for businesses to know the consumers' sentiment towards their brand. This project, intended for use by engineers at Orange France, focuses on the development of a micro-services architecture built around the open source cluster computing framework, **Apache Spark**. Thanks to is distributed model, Spark can process massive amounts of data stored in databases as well as real time data streamed from social networks. Data is then stored in a cluster database and exposed by an API (Application Programming Interface) for users to access them in real time.

1 Context

Big data [1] is a term used to designate data that is not only too voluminous to fit in a standard database, but is also too diverse and appears at such speed that it cannot be processed using mainstream systems.

Currently, Orange is a company already working on big data applications and is interested in using emerging big data technologies. More precisely, among those technologies available today, Apache Spark is the most promising one thanks to its processing performances that makes it well suited to machine learning algorithms.

1.1 Apache Spark

Apache Spark is a distributed and highly scalable system, providing the ability to develop applications using languages like Java, Scala (the language used to write Spark itself), Python and R. It was originally developed at the University of California, Berkeley and donated to the Apache Software Foundation in 2013.

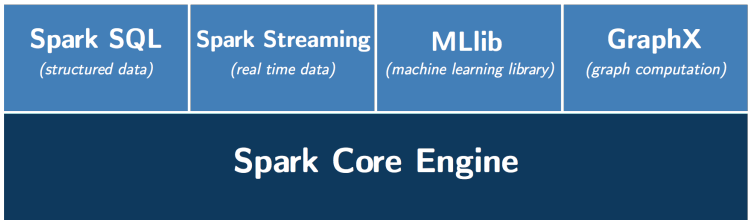

Figure 1: The Spark stack

Contrary to existing distributed computing software for processing very large data sets, such as Apache Hadoop, Spark is based on an in-memory programming model, instead of using the MapReduce [2] disk-based model, allowing it to be faster and more flexible. For instance, Spark can sort 100 TB of data three times faster and with ten times fewer machines than MapReduce [3].

Spark consists of four main interoperable components showed in figure 1. Spark Core is the foundation of the Spark project and contains features such as task scheduling, memory management and fault recovery. On top of it lies four modules, of which two are used:

- **Spark Streaming** leverages Spark Core capabilities to enable processing of live streams of data. It is used extensively in this project for fetching data from social networks.
- **MLlib** is a library containing machine learning algorithms that can be applied to compute statistical models from data.

1.2 Using Spark in real world scenarios

The goal of this project is to demonstrate Spark capabilities, effectiveness and usability. At first, requirements were not clearly defined as this project is mainly aimed at exploring what can be done with Spark, but they had to match certain expectations from Orange. These requirements, introduced in the next section, take the shape of use cases as well as Spark integration with other tools.

2 Developed use cases

2.1 Estimating the crowd in Orange stores

In the way Google shows rush hours of stores and businesses [4], the first use case consists of analyzing videos to deduce when customers are more likely to visit Orange stores, located by city or by department.
Having infrared sensors at the entrance of the stores to detect a person getting in or out was the first solution yet it was not chosen for many reasons (detection problems due to overlapping, high cost of connected sensors...). Having a terminal where users register each time they get to the store was another interesting idea to explore. This solution was also unimplementable since it is costly and necessarily needs the cooperation of the clients which is not always guaranteed.
Finally, the retained solution was installing a camera in the different stores and using video processing algorithms to detect people. This solution has to be discontinued due to different reasons that we will expose next.

2.1.1 Real time video processing

Detecting and tracking people on the real-time video stream of each store is developed with the OpenCV library HOG-based pedestrian detection. An algorithm to track a person over 5 successive frames is also implemented in order to count him only once if this person happens to appear multiple times. Consequently, the program's output is composed of videos with rectangles around the detected individuals and squares around their faces.
After multiple tests and changes, the detection quality could not be improved since many false negatives and false positives occured especially for face detection. The computation process could not be accelerated to get closer to a real time estimation since a GPU (Graphical Processing Unit) is required for this matter. These are the main problems that led to the cancellation of this use case.

2.1.2 Data transfer and processing

This part's purpose is to assure that the number of new persons detected after each frame is sent to a central unit where the data's corresponding store is identified using Spark's MapReduce functionality and aggregated to get the total number of persons per store hourly and then compute the mean time by city or by department. A front-end module was also planned to be implemented to visualize the results in real-time.

The work that has been done was mainly based on basic input/output text files for tests instead of the real Apache Kafka channel that was expected to exist between each store and the central module.
Since the problems that were faced in the first part the team from pursuing this use case, implementing the front-end or computing the mean times of affluence for cities and departments was abandoned.

2.2 Data mining social networks

2.2.1 Identifying useful sources

Social networks are a highly valuable source of information about consumers relationship towards a brand. For instance, Orange uses Facebook through two pages: Orange and Sosh and manages several Twitter accounts: @orange, @Orange_France, @Sosh_fr and @Orange_conseil. Moreover, consumers can access forums at `communaute.orange.fr`. These sources are used for fetching live data and feeding Spark with them.

2.2.2 Spark applications

As said in section 1.1, Spark applications can be written in several languages: Java, Python or Scala. Scala was initially chosen since Spark itself is developed in Scala and it is also less verbose than Java code. Furthermore, Java and Scala applications have the advantage of being compiled and packaged into an **Uber-JAR**. **JAR** (Java ARchive) is a package file format used to aggregate class files required for an application to run, and an Uber-JAR is a JAR that not only contains the application code but also embeds its dependencies as well. This way, only the Uber-JAR needs to be submitted to Spark for the application to be executed.
Applications are also separated by sources, allowing a more flexible development and release to production as well as a reduction of risks. Indeed, Facebook and Twitter have their own independent API services and having issues with one will not impact the other. This structure also leads to a better bug tracking management with having the bug-free application uninterrupted.

2.2.3 Sentiment analysis

Sentiment analysis refers to the use of natural language processing and text analysis to identify and extract subjective information in source materials. It aims at classifying the polarity of a given text — whether the expressed opinion is positive, negative, or neutral. Based on the source, sentiment analysis is applied to tweets on Twitter, posts on Facebook, and thread titles and messages on forums.

Dictionary-based approach The first approach consists in looking at words in isolation, giving positive points for positive words and negative points for negative words found in two separate dictionaries and then summing up these points.

The following workflow example is based on a real tweet from an Orange consumer.

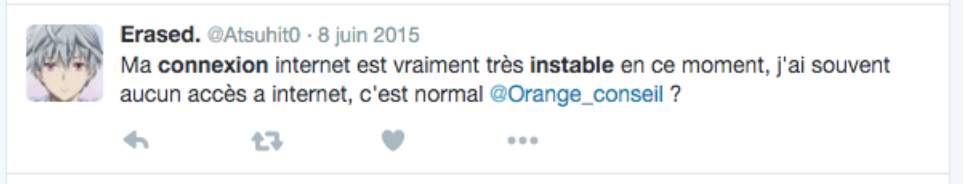

Figure 2: Tweet from a user to @Orange_conseil

The first step is to tokenize the text, translating a sentence into a list of independant words. This is done using the `FrenchTokenizer` class, part of the **Stanford CoreNLP** suite of Natural Language Processing tools [5].

Along with tokenizing a text, common words with no interesting meaning such as "le", "dans", "à" also known as "stop words" are filtered out to produce the following list of meaningful words.

connexion, internet, vraiment, instable, moment, souvent, aucun, accès, internet, normal

Figure 3: Words after tokenization

Each word in the above list is then looked up in the positive and negative words dictionaries. In the above tweet, the words *instable* and *aucun* can be found in the negative dictionary.
Finally, the sentiment for the whole tweet is given by computing the difference between the positive weight and the negative weight. A sentiment score equal to 0 is associated with "NEUTRAL", while positive scores are marked as "POSITIVE" and negative scores as "NEGATIVE". Consequently, the example tweet has a negative weight of 2 and a positive weight of 0, labeling it as negative.

This method is far from perfect since it gives a sentiment based on separate words instead of the context of a sentence. The sentiment analysis annotator from Stanford CoreNLP was tested before implementing this technique and although it yields more accurate results based on a large grammar treebank, it is only available for English text analysis.

Machine learning-based approach Thanks to Spark's Machine Learning library MLlib, classification algorithms can be applied to analyze tweets more accurately. In this approach, the Naive Bayes [6] algorithm, a probabilistic classifier based on Bayes' theorem, and the Random Forest [7] algorithm, an ensemble of decision trees for binary and multiclass classification, are implemented through two steps: train and predict.

Training the data requires having a set of tweets already classified by hand, with 1 meaning *positive* and 0 meaning *negative*. This data set is then loaded and 80% of the data is split into training data, used to train the classifier model, while 20% is split into testing data, used to assess the performance and the accuracy of the trained model. With these algorithms, 74% of the 1343 tweets is correctly classified. According to the computed results below, the Random Forest algorithm is slightly more accurate than the Naive Bayes algorithm yet it is less time efficient.

Algorithm	Naive Bayes	Random Forest
Accuracy	74.04%	77.39%
Training duration (ms)	623	14266
Testing duration (ms)	13	193

Predicting data classification is then possible by applying the previous trained model.

2.2.4 Specific computed indicators

Twitter For Twitter, the response time of a tweet addressed to after sales accounts such as @Sosh_fr and @Orange_conseil, the number of retweets for a tweet, the number of tweets per second as well as a ranking of trending hashtags are indicators computed from tweets data.

Facebook Similarly to Twitter, the number of Facebook posts collected every minute, the number of likes and comments per post are computed in the same manner.

Orange forums Scraping Orange forums web pages was done in Scala, for better integration with Spark, using `scala-scraper` [8] which provides a powerful library for scraping HTML. At first, each subforum's first page is scraped to fetch the threads' title, link, date and messages. Then the last thread's date is compared to the current system's time. If the two match, data is stored in Cassandra. This is done every minute for real time streaming. Following the same pattern, the number of unanswered topics can be computed.

3 Building a microservices architecture

Microservices pattern is an architectural style in which large complex software applications are broken down into a collection of small, independent, loosely coupled processes. This helps upgrading, modifying or replacing one service instead of taking down the entire system. As shown in the figure below, this project is composed of several services communicating with each other, which are presented in the next sections.

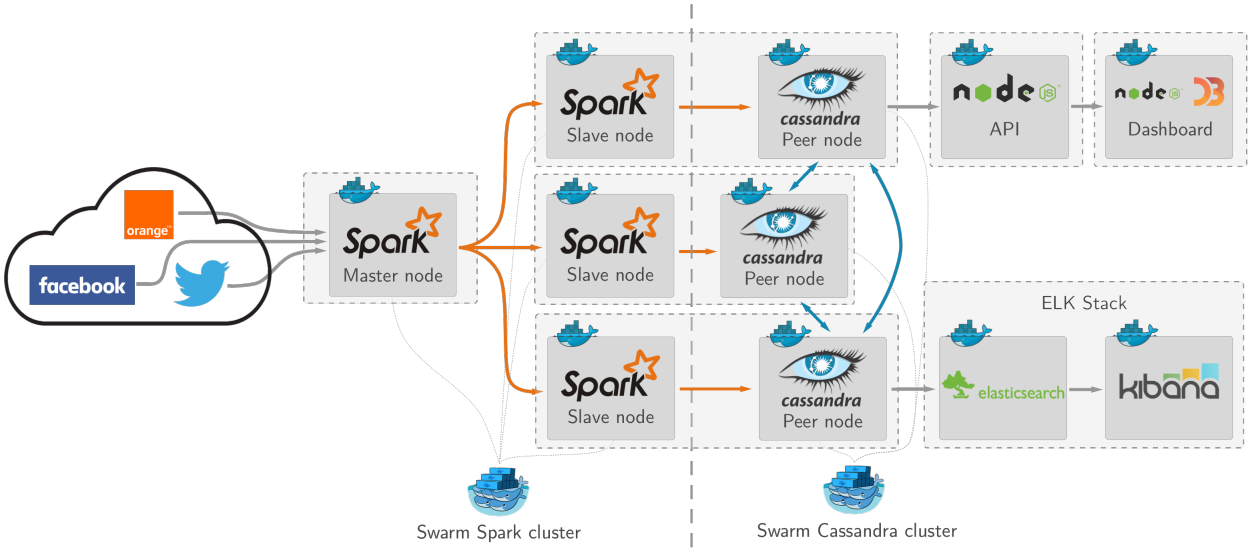

Figure 4: Deployed infrastructure

3.1 Docker

Docker is the ideal technology to facilitate the deployment of each application. It automates the deployment of applications inside containers, isolated from the host and other containers by leveraging features of the Linux kernel. Docker containers wrap up a piece of software in a complete filesystem with its dependencies – ensuring that it will run consistently across all environments. In terms of virtualization, Docker uses the features of the host OS (Operating System) instead of requiring an hypervisor and a guest OS like a virtual machine.
The Spark cluster is deployed with Docker Compose [9] and both clusters are orchestrated with Docker Swarm [10].

3.2 Spark cluster

Spark follows the master-slave model and it is composed of a master node which schedules and shares jobs between one or more workers.

3.3 Cassandra cluster

Apache Cassandra is a NoSQL open-source distributed database system. It is designed to handle large amount of data replicated across servers in a cluster. Each node is installed on the same machine as the Spark workers in order to optimize access to the database and minimize network latency.

Parameters Ensuring reliability and fault tolerance on multiple nodes is possible by setting the replication factor according to the number of nodes in the cluster. For instance, three nodes (A, B, C) are deployed for this project.
With a replication factor of 3, the QUORUM level (rounded down to a whole number) is (replication factor/2) + 1 = 2 which means the cluster can tolerate 1 node down. Besides, each node holds 100% of the data. This allows a client to write data to A and read the same exact replica from B.
Furthermore, consistency is configured to use the QUORUM level with the following CQL (Cassandra Query Language) [11] command:

```
CONSISTENCY QUORUM;
```

With a replication factor of 3, this ensures that 2 nodes are always written and 2 nodes are always read.

3.4 API

A RESTful API, developed with NodeJS, is connected to Cassandra to make data available for visualization. Its documentation is written with Swagger [12] for a better visual representation.

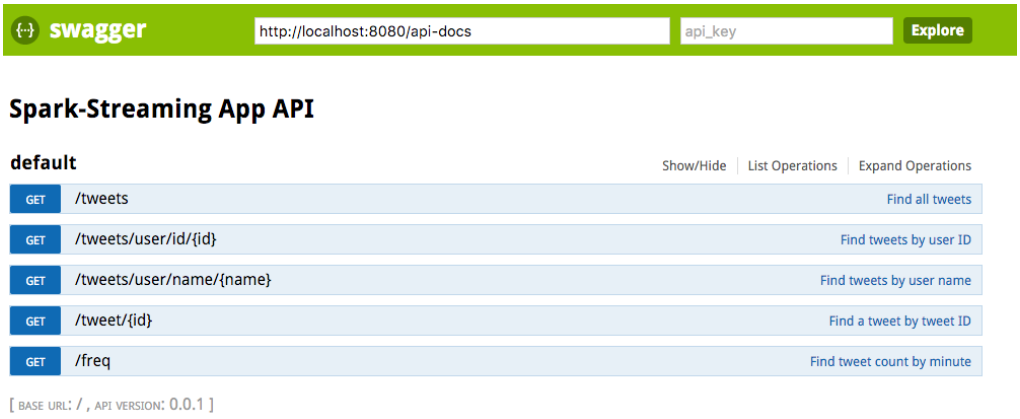

Figure 5: API documentation available at `{API_IP_ADDRESS}:8080/docs`

The NodeJS driver for Cassandra [13] is implemented for streaming data stored in the database to the API. Along with this API, another NodeJS web server has been developed to serve a web dashboard, using D3.js [14] for charts. When a client loads the page, data is pushed from the API to the dashboard using Socket.io [15].

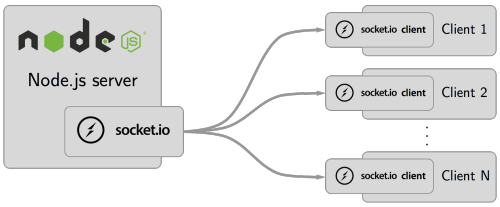

Figure 6: Real time communication between clients and the server with Socket.io

3.5 ELK stack

In parallel, the Elasticsearch - Logstash - Kibana stack is tested as a turnkey solution to replace Cassandra, the API and the web dashboard. Each of the three parts of the ELK stack has a very specific role :

- Elasticsearch manages the database and the requests to it
- Logstash sends data to Elasticsearch from a data file (`csv`, `json`, ...). In this project, Logstash is not used as Spark enables data to be directly saved in Elasticsearch
- Kibana is the visualization tool, for creating diagrams and graphs. It is a web application, in which the dashboards can directly be created using visual programming

The architecture used for integrating ELK tools in the project is shown on Figure 4.
Once Elasticsearch has been filled with data processed by Spark, Kibana has access to this data and graphs interpreting those information can be created.

4 Organization and Management

4.1 Methodology

Agile software methodology was an efficient way of organizing this project where objectives were not accurate and stable at the beginning. The development was split in sprints (or iteration) of 3 weeks (See appendices).
The clients were actively involved in the process and meetings were planned every week when possible to demonstrate and review new features. The progress of the project could be followed thanks to weekly progress reports available at `bfovet.vvv.enseirb-matmeca.fr/reporting_projet`.
A team charter was first written to define the organization and the values shared by the team during this project.

4.2 Tools

The project was managed on Trello. For each sprint, three boards (To do, In progress, Done) were used to sort tasks. Progress and meeting reports, resources, tutorials as weel as management documents were shared on Google Drive. GitHub was used for source code hosting and issue tracking. Each service/use case has its own repository, available at `https://github.com/t3g7`. Applications were unit tested with Travis CI, a continuous integration service used to build and test projects hosted at GitHub. Each time a team member pushed code to GitHub, a test build was triggered on Travis CI. After a successful test on Travis CI, the resulting built Docker image was uploaded to the Docker Hub, a cloud hosted service for Docker image repositories.

4.3 Issues

Early in the project, data privacy issues were faced. A use case about analyzing customers logs was considered yet data had to be anonymized in the first place. Also, because access to Orange data was restricted, it was decided to focus on alternative sources of data: social networks and open source data sets.
Moreover, many technical issues had to be solved or bypassed. At first, Scala was a new language with a particular syntax to be assimilated. Installing and configuring Apache Spark took several hours to be complete. Since a microservices architecture was preferable to be dealt with, many different parts hat to be developed and then integrated together.
The sprint involving video processing had to be abandoned due to an overtime risk. Its integration with Spark was indeed too complex since it involved machine learning on images as well as streaming large amounts of data in real time.

5 Conclusion

The project consists of applications and services developed around Apache Spark, itself being a useful piece of software for processing large amounts of data from various aggregated sources in real time, whether they are databases or online APIs. Development of such applications is relatively simple thanks to a rich ecosystem, including many libraries, bindings for different popular languages and an active community.
Though not completed, it would have been possible with enough allocated time to compute stores rush hours with video processing by combining OpenCV and Spark. Objectives of this use case were defined as identifying overcrowded stores and predicting popular times to shorten customers waiting time.

Finally, applications for data mining social networks - Twitter, Facebook and Orange forums - can be tools to help community managers to focus on unanswered questions of customers on after-sales accounts and offer a global point of view on trends and sentiment towards the brand from uncorrelated data.

While Spark continues to be heavily developed and maintained, next generation software for big data processing are being released. Apache Flink [16] is still in a beta state (0.10.1 - 27 November 2015) and it is solely focused on streaming, making it equivalent to the Spark Streaming module.

6 References

[1] *Big data now, 2014 edition.* O'Reilly Media, 2015

[2] `https://wiki.apache.org/hadoop/MapReduce`

[3] "Spark wins Daytona Gray Sort 100TB Benchmark." `https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark`. 5 November 2014

[4] Business popular times on Google. `https://support.google.com/business/answer/6263531?hl=en`

[5] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60. `http://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf`

[6] Naive Bayes, `https://spark.apache.org/docs/latest/mllib-naive-bayes`

[7] Random Forest, `https://spark.apache.org/docs/latest/mllib-ensembles#random-forests`

[8] Rui Gonçalves, scala-scraper on GitHub, `https://github.com/ruippeixotog/scala-scraper`

[9] Docker Compose, `https://docs.docker.com/compose`

[10] Docker Swarm, `https://docs.docker.com/swarm`

[11] CQL, `http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c`

[12] Swagger, `http://swagger.io`

[13] DataStax, nodejs-driver on GitHub, `https://github.com/datastax/nodejs-driver`

[14] Socket.io, `http://socket.io`

[15] D3.js - Data-Driven Documents, `http://d3js.org`

[16] Apache Flink, `https://flink.apache.org`

7 Appendices

7.1 Data visualization dashboard

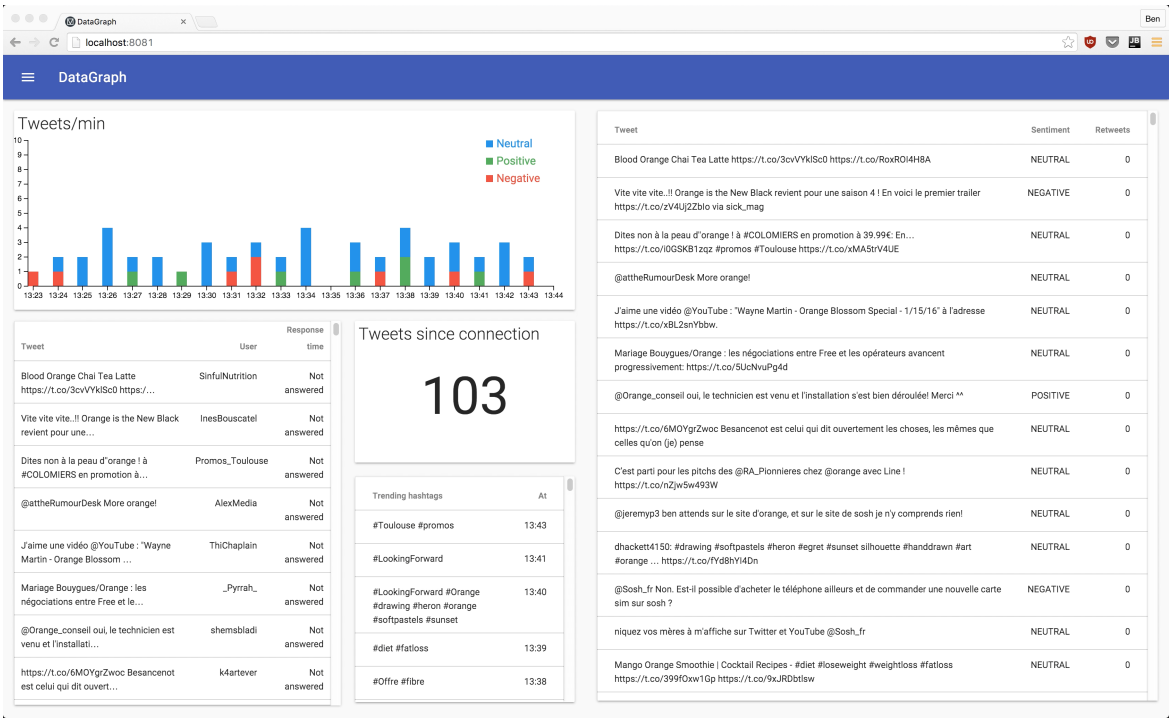

Figure 7: Data visualization dashboard

7.2 Project organization

7.2.1 Project planning

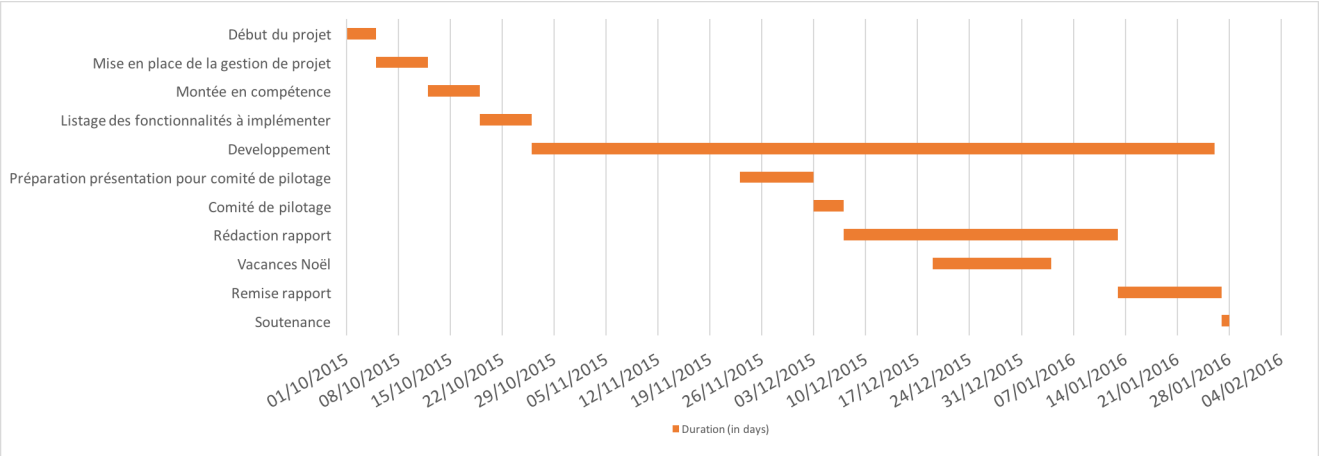

Figure 8: Gantt diagram

7.2.2 Sprints

Sprint	Task
0	Learning Big Data and Spark Finding the use cases to develop
1	Deploying Apache Spark Setting up real-time video processing Exploring the Twitter API
2	Twitter mining with Spark Real-time video processing with Spark
3	Twitter analysis Facebook mining & analysis Forums mining & analysis Data visualization

7.2.3 Burndown chart

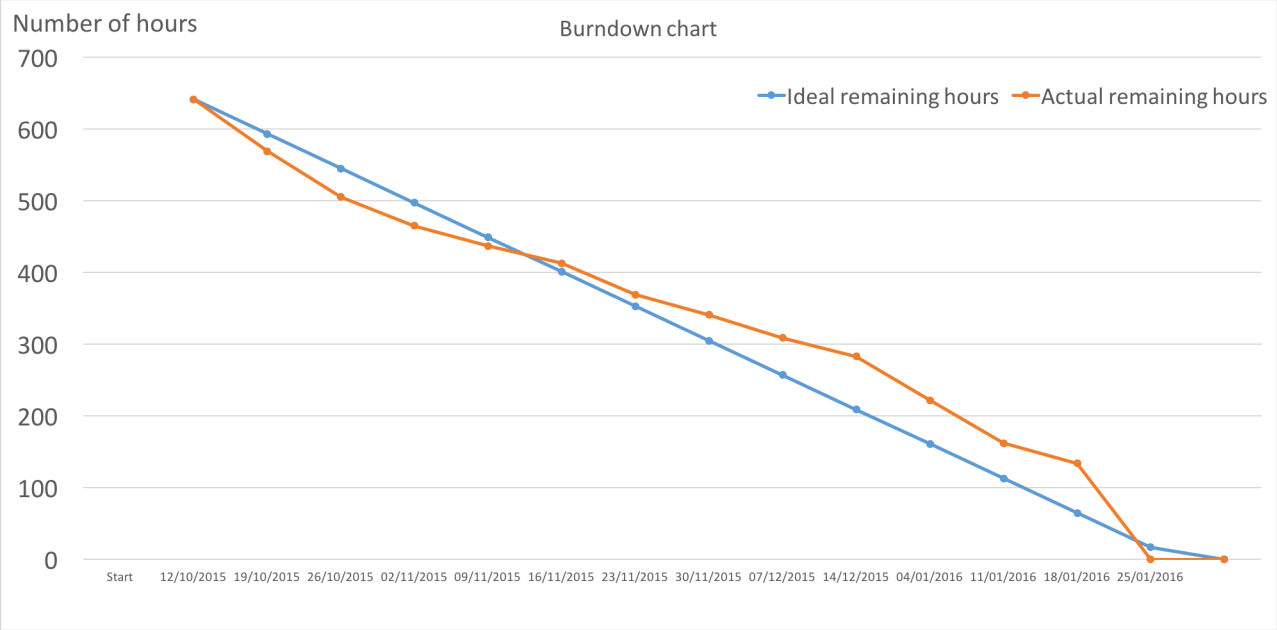

Figure 9: Burndown chart