Inferring Latent User Attributes in Streams of Multimodal Social Data using Apache Spark

A thesis presented

By

Alessio Conese

Supervisor:
RUBÉN TOUS LIESA
JORDI TORRES VIÑALS

FIB - Facultat d'Informàtica de Barcelona
Barcelona, 2015
The work aims to discover and exploit the features of Apache Spark, which is open-source data analytics cluster computing framework that provides primitives for in-memory cluster computing that makes it well suited for this type of problems. For this purpose, initially, it is analyzed a multimodal social data such as community-contributed photos and videos (e.g. Instagram or Twitter). In this part the data are analyzed to obtain model used to understand a possible affiliation of the users to a precise social network group.

After, in order to analyze the behavior of Apache Spark in a real cluster, was built an implantation of Spark in the MareNostrum supercomputer. The software obtained from the first part is executed in the in the cluster to obtain a measurement of the models quality. In addition, this document describe the step to work with Spark, its advantages and disadvantages, and the implementation in a cluster, with a briefly analysis of its scalability and performance.
TABLE OF FIGURE

Figure 1-1. Increment of digital data in the last 10 years and his possible evolution...... 9
Figure 3-1. Histogram performance for logistic linear regression................................. 13
Figure 3-2. Spark timeline version with added features............................................. 14
Figure 3-3. Apache Spark stack.................................................................................. 15
Figure 3-4. Spark Streaming flow................................................................................ 17
Figure 3-5. Property Graph example .......................................................................... 18
Figure 3-6. Graph and table manipulation example.................................................... 18
Figure 3-7. Stack of MLbase ...................................................................................... 19
Figure 3-8. Simple architecture for a run application.................................................. 20
Figure 3-9. HDFS client operation schema ................................................................. 22
Figure 3-10. HDFS architecture................................................................................ 23
Figure 3-11. MapReduce overview ........................................................................... 24
Figure 3-12. Sequence File layout with records or blocks fields............................... 25
Figure 3-13. Structure of Shuffle and Sort phases.................................................... 27
Figure 3-14. Configuration stacks to run Spark over Hadoop.................................... 28
Figure 3-15. Possible types of dependencies among RDDs....................................... 30
Figure 3-16. Schema of the flow in the iterative phase .............................................. 32
Figure 3-17. Schema of the flow in the initialization phase....................................... 33
Figure 3-18. Final schema of the RDDs and its dependencies ................................... 34
Figure 3-19. SparkR data flow schema...................................................................... 35
Figure 3-20. New BlinkDB component for Spark stack........................................... 36
Figure 3-21. Yahoo chooses to work with Shark to provide Ad Analytics................. 37
Figure 3-22. Conviva network status control............................................................ 37
Figure 3-23. ClearStory process include in-memory computation done by Spark....... 38
Figure 4-1. Most used tags....................................................................................... 47
Figure 4-2. Example of one JSON object store in the file data sample..................... 48
Figure 4-3. Example of hyperplane that separate points in two different categories .... 51
Figure 4-4. Graph of the logistic function ................................................................ 53
Figure 4-5. Binary classification with Naive Bayes, the two categories created and the
decision boundary is shown in red.......................................................................... 54
Figure 5-1. MareNostrum supercomputer inside the glass box of Torre Girona chapel.55
Figure 5-2. Panoramic of MareNostrum racks: 7 Storage Server (green), 4 Network elements (red) and the Computation racks (blue) ................................................................. 58
Figure 5-3. Shared Disk environment .............................................................................. 59
Figure 5-4. LSF schema .................................................................................................. 60
Figure 5-5. Job lifecycle ................................................................................................. 62
Figure 5-6. Nodes distribution mode .............................................................................. 64
Figure 5-7. Example of a job configuration file for spark4mn command ................. 66
Figure 5-8. Example for setting a range of value in spark4mn_benchmark configuration file .................................................................................................................. 67
Figure 6-1. Histograms of the models accuracy ............................................................. 74
Figure 6-2. Comparison of models accuracy ................................................................. 75
Figure 6-3. Confusion matrices of Logistic Regression, Support Vector Machine and Naive Bayes .............................................................................................................. 75
Figure 6-4. Performance of Naive Bayes using 4 workers per node and different number of nodes ............................................................................................................. 77
Figure 6-5. Performance of K-means using 4 workers per node and different number of nodes .................................................................................................................. 77
Figure 6-6. Performance of Naive Bayes using 4 nodes and different number of workers per node ............................................................................................................. 78
Figure 6-7. Performance with 2 cores per worker ......................................................... 78
Figure 6-8. Performance with 4 cores per worker ......................................................... 79
Figure 6-9. Performance with 8 cores per worker ......................................................... 79
Figure 6-10. Histogram of performance using different partitions and number of workers per node ........................................................................................................... 80
# Table of Contents

Abstract ........................................................................................................................................ I
Table of Figure ........................................................................................................................ II
1. Introduction ............................................................................................................................... 7
   1.1 Motivation ............................................................................................................................. 7
   1.2 Objectives ............................................................................................................................ 8
2. Big Data ..................................................................................................................................... 9
   2.1 Increasing data ......................................................................................................................... 9
   2.2 About Big Data ....................................................................................................................... 9
   2.3 New Paradigms ....................................................................................................................... 10
      2.3.1 Volume ............................................................................................................................ 10
      2.3.2 Variety ............................................................................................................................. 10
      2.3.3 Velocity ........................................................................................................................... 11
      2.3.4 2 more V: Value and Veracity ......................................................................................... 11
   2.4 Relevance of Big Data and Application ................................................................................. 11
   2.5 Frameworks .......................................................................................................................... 12
3. Apache Spark ........................................................................................................................... 13
   3.1 What is Spark ......................................................................................................................... 13
      3.1.1 Spark Version and history .............................................................................................. 14
   3.2 Stack ..................................................................................................................................... 14
      3.2.1 Spark Core ...................................................................................................................... 15
      3.2.2 Spark SQL ....................................................................................................................... 16
      3.2.3 Spark Streaming .............................................................................................................. 16
      3.2.4 GraphX ........................................................................................................................... 17
      3.2.5 MLlib .............................................................................................................................. 18
      3.2.6 Cluster Managers ........................................................................................................... 19
   3.3 Spark and Hadoop ............................................................................................................... 21
      3.3.1 Hadoop .......................................................................................................................... 21
      3.3.2 Working together ........................................................................................................... 27
      3.3.3 Performance comparison ............................................................................................... 28
   3.4 Spark execution .................................................................................................................... 29
      3.4.1 RDD in deep ................................................................................................................... 29
      3.4.2 DAG (Directed Acyclic Graph) ....................................................................................... 30
7. Conclusion and Future Work ......................................................................................... 81
    7.1 Future Work .......................................................................................................... 82
1. **INTRODUCTION**

In both commercial and scientific fields, new data sources and instruments are producing rapidly increasing amounts of information in fact the past few years have seen a major change in computing systems, as growing data volumes require more and more applications to scale out to large data. More even the companies nowadays are interested in analyze a big amount of data to retrieve information of its interest. Furthermore, many other entities could have advantages in the data procession, for example a real time traffic analyses could adapt the public transport.

In order to inspect and figure out useful information the scientist moves his effort to discover and improve machine learning algorithms, which spend a lot of resource in term of time and computational power. Moreover, the statistical data analysis could be combined with the computing power offered by the cluster in order to increase the performance for complex computation.

The purposes of this introduction are to understand the motivation, for which this project was started, and second for helping the reading of the document, explaining the scope that the work yearns to achieve.

1.1 **Motivation**

In business, data is synonymous of evidence, objectivity, and hidden insight. This perception of data as the magic key that opens doors holding information to anything we’ve ever wanted to know makes us obsessed with the latest ways to get it.

The emergence of the Web and Online represents a fundamental shift as it has added important new dimensions to the production and dissemination of news and information, in the past there was a traditional one-way mass-media that generate information to consumer community, but in the last years the creation of social media are made from and individual users, that create and curate the information of their self and then share in public spaces, so the communication and generation of information become a many-to-many, the consumer community create a owns network to share everything. The Social Media data takes importance and becomes an important topics for the scientific community. Many platforms allows to manage these information, such as:

- Social media networks: Tweets, posts, favorites, sentiment, images
- Social search: keyword analysis and hashtag tracking
- Long-form publishing platforms: blogs, wikis, and social opinion sites such as Yelp
- Public multimedia content-sharing platforms: SlideShare, YouTube, Flickr, etc.

Analyzing social media data and producing genuine insights can be a complex undertaking, but the new cooperation between Big Data and Social Media converge in the world of what
happens after content is produced and collected. The use of social media creates and generates mountains of data, therefore producing big data and generating the need for big data analysis to synthesize the information and determine actions.

The increase interest in the Big Data and Social Media data in business and scientific research bring us to try to deal with new technologies for working with these concepts.

1.2 Objectives

The principal goal of this work can be expressed in two simple words Apache Spark; basically this framework help a developer to deal with big data. Our scope is to understand how Spark operates and use it to deal with Big Data; in order to discover the potential of Spark are taken into account some different function offered.

In the first part are exploited the APIs offered for implement different classifier basing on different algorithms. These classifier are created to work with Instagram metadata and to predict the possible users’ behaviors. To understand the quality of each classifier are used different methods, the mayor part provided by Spark itself, and other implemented for this specific scope.

In the second part instead, are analyzed the performances of Spark in a cluster, to reach this objective, a version of spark was implemented in a supercomputer that allows to run a spark job on many nodes. The main aspect studied is how the computation time change basing on the number of nodes used to run a job.

Analyzing the behaviors cited above we can create an overview of Spark performance and function, and we can understand if this framework is suitable for working with big data for the developers and if it can be used by the companies as support in their decision basing on information extracted in real time from big amount of data.
2. **Big Data**

In this chapter are introduced briefly the concepts of Big Data, to better understand the problems caused by rapid increase information produced and possible solutions.

2.1 **Increasing Data**

The production of information has exploded with the Internet advent, in fact in the present day every person is responsible of new data, in daily life: from phones and credit cards and televisions and computers; from the infrastructure of cities; from sensor-equipped buildings, trains, buses, planes, bridges, and factories. Everything contributes to produce information.

The digital data produced in the 80’s were 0.02 Exabyte; in those years, begin the digital age, and the information per-capita has doubled every 40 months, up to now, for example in the 2014 were produced 2.3 zettabytes per day.

![Figure 2-1. Increment of digital data in the last 10 years and his possible evolution](image)

2.2 **About Big Data**

The first definition of big data appears in 1997 in an article by NASA scientist, in which they describe a problem with visualization, they said “provides an interesting challenge for computer systems: data sets are generally quite large, taxing the capacities of main memory, local disk, and even remote disk. We call this the problem of big data. When data sets do not fit in main memory (in core), or when they do not fit even on local disk, the most common solution is to acquire more resources.” (1). After this first appear the big data remains left out for a while from the research world and it return only few years ago, until 2008 a group of Academic scientists coin in a paper a new use of this term “big-data computing” (2). Instead, Oxford English Dictionary define big data as “data of a very large size, typically to
the extent that its manipulation and management present significant logistical challenges.” It is possible to find many other definitions, all of them valid and interesting, and so is easy to understand the problems of this field, the difficult to handle with this huge amount of data.

The old system to store and analyze the data are not appropriate for this amount of data, the new challenge for the scientists and the companies is to find out new approach that allows to manage and store these data.

The big data studies precisely try to resolve these problems using new approaches and basing on different paradigms.

## 2.3 New Paradigms

Big data is difficult to work with using most relational database management systems and desktop statistics and visualization packages, requiring instead massively parallel software running on tens, hundreds, or even thousands of servers. Big data distances itself from the classic paradigms of old store and manipulation system, and it is based on 3 principles, the “3V” of Big Data.

### 2.3.1 Volume

The name ‘Big Data’ itself contains a term which is related to size and hence the characteristic. The volume of data created is outpacing the amount of currently usable data to such a degree that most organizations do not know what value is in their data, in fact the quantity of data that is generated is very important in this context. It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big Data or not. More sources of data with a larger size of data combine to increase the volume of data that has to be analyzed. This is a major issue for those looking to put that data to use instead of letting it just disappear.

### 2.3.2 Variety

In this new era, done of infinite connection and source of information, structure can no longer be imposed like in the past in order to keep control over the analysis. As new applications are introduced new data formats come to life. From excel tables and databases, data structure
has changed to lose its structure and to add hundreds of formats. Pure text, photo, audio, video, web, GPS data, sensor data, relational data bases, documents, SMS, pdf, flash, etc etc etc. One no longer has control over the input data format. Rarely does data present itself in a form perfectly ordered and ready for processing. A common theme in big data systems is that the source data is diverse, and doesn’t fall into neat relational structures, it take unstructured data and extract ordered meaning, for consumption either by humans or as a structured input to an application.

2.3.3 Velocity

The term ‘velocity’ in the context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development. Initially, companies analyzed data using a batch process. One takes a chunk of data, submits a job to the server and waits for delivery of the result. This approach can’t work with the inflow of data, and so the batch process breaks down. It’s not just the velocity of the incoming data, but now the data is now streaming into the server in real time, in a continuous fashion and the result is only useful if the delay is very short. The new technology has to deal with the rapidity of data creation and the system has to respond as fast as possible, the importance lies in the speed of the feedback loop, taking data from input through to decision.

The best practice to increase the velocity of system response is to avoid the storing of the data, because they are too much and too fast, and take decision in a real-time basing on a stream approach.

2.3.4 2 more V: Value and Veracity

Further, there two other V in the characteristic of big data:

- Veracity is refers to the untidiness or truthfulness of the data. With many different forms of big data, quality and accuracy are less but big data and analytics technology now allows us to work with these type of data, in fact often the data does not need to be perfect, but does need to be close enough to gain relevant insight.
- Value simply represents the business value to be derived from Big Data: the translation of both structured and unstructured data into business insights. A big data strategy gives businesses the capability to better analyze this data with a goal of accelerating profitable growth.

2.4 Relevance of Big Data and Application

The big data problem is not new, this problem has only been exaggerate as now individuals are generating so much more information and data than ever before as they go through their daily lives, are included information garnered from social media, data from internet-enabled
devices machine data, video and voice recordings, and the continued preservation and logging of structured and unstructured data.

Increasingly grows the importance of acquire and analyze information, for the companies is crucial to understand what the clients like, so big data is important because it will transform how we manage our enterprises, and adapt them to the needs of the people. In business, sometimes is need to make decisions based on incomplete information in a rapidly changing context, in this case the big data concept match perfectly to solve the companies requirements.

Big data roughly could be applied in all the area where there is need rapid decisions basing a fast input information flow, where have too much data to store effectively or compute efficiently using traditional methods, it has expanded to mean a situation where the logistics of storing, processing, or analyzing data have surpassed traditional operational abilities of organizations. Big data gives a structured approach to make decision, not basing only to human intuition, could be a power way to identify opportunity.

The big data could be used in different field, for instance in a biomedical experiments where different types of experiments generate different types of data that are very difficult to match up, or in other science where are produced many information. Big data is used by many famous companies, like Amazon or EBay, to extract relevant data from the behaviors of their users. Also, it has international and government application, such as smart city or health services.

2.5 Frameworks

For Big Data analyze there are several framework that can be used, such as GraphLab, Hadoop, Storm and many other; but in this work the only framework used is Spark, so and in next sections is explained in deep the structure and the features of it. Also it is presented an overview of Hadoop, because some concept relative to spark derive from Hadoop architecture.
3. **APACHE SPARK**

Apache Spark provides an attractive, attractive development API and permit data workers to rapidly iterate over data via machine learning and other data science techniques that require fast, in-memory data processing. Actually Spark is the most active project in the Big Data ecosystem, it was originally developed in 2009 in UC Berkeley’s AMPLab, and open sourced in 2010 and it is immediately become a top-level Apache project and interesting topic in all the big data communities, with over 200 contributors in 50+ organizations.

In this part are explained all the relevant components, the basic structure and the library that are included in Spark.

The first step is to understand in concrete what is spark, and why has been chosen for this project. After that, is described the stack the stack of the framework and each part of it. Finally, are illustrated the library include and the application in which Spark can used.

### 3.1 WHAT IS SPARK

Spark is s general-purpose open source computing framework specialize in big data analytics, it builds on top of Hadoop HDFS system, is totally compatible with it and is not tied to the map reduce platform. It offers different solution for the developers in three main languages Python, Java and Scala, but the last one is the more suitable for the framework, because the documentation and the implementation algorithm are principally done with this.

Apache Spark is platform implemented for cluster computing and it is designed to be fast. It use a own reformulation of MapReduce, to increase the support of the type in the computation, and adding new interesting features, like the possible to streaming processing and to submit queries in real time. Obviously, Spark was born for Big Data application, so the most important quality is the processing large data set very fast. Spark is designed in manner to allow an easy combination of different processing times and to cover a wide range of workloads that previously required separate distributed systems, like batch application, interactive queries, streaming and iterative algorithms.

Spark become very popular for its ease of use, is possible to use object-functional scripting and intuitive API, with a little effort is possible to write simple code in order to obtain a good result in data analysis, quite hard with previous framework.

Figure 3-1. Histogram performance for logistic linear regression
Hadoop is the standard for large scale data processing across nearly every industry and enterprise, it scales out computation and storage across cheap commodity servers and allows other applications to run on top of both of these. A bonus point for Apache is a totally integration with Hadoop, in fact it is built on top of Hadoop and is intention is to enhance, not replace, the Hadoop stack, it tries to making it as easy as possible for every Hadoop user to take advantage of its capabilities, such as better utilization of compute capacity, reduce unnecessary replicated reads and writes and the elimination of the need for data transfer in and out of the cluster, obtaining a performance 10-100x faster of Hadoop. Spark is not only compatible with Hadoop, but it can also runs with Mesos, standalone, or in the cloud, and it can access diverse data sources including HDFS, Cassandra, HBase, S3.

3.1.1 Spark Version and history

Spark started in 2009 as a research project in the UC Berkeley RAD Lab, later to become the AMPLab. In the begin they were working on Hadoop MapReduce, seeing that it was inefficient for iterative and interactive computing jobs, so they start to implement Spark on Hadoop, in the same year research papers were published about the project academic conferences. After one year the code was open sourced and a license was acquire. In 2011, was started the development for the high live component, such as Shark or Spark Streaming.

From the beginning, was a very popular project with an active participating from the community, who participation increase every day, in fact the creators of Spark decides to found the DataBricks society to commercialize it. The first version, the 1.0.0 was released May 2014, in about 7 months the project is arrived at the version 1.3.0, release only in March 2015.

Spark Roadmap

![Spark Roadmap](image)

*Figure 3-2. Spark timeline version with added features*

3.2 Stack

Spark tries to combine and integrate different component and so is stack is compose from different multiple closely-integrated components. The total integration of the components provide a closely interoperation, letting you combine them like libraries in a software project and giving some benefits from improvements at the lower layers. Every time that a new
component is included in the stack, Spark is able to provide a perfect interoperability and so every organization that uses Spark will immediately be able to try this it, reducing the cost of trying. Imaging to run a different program for each stack, without interoperability, you have start an execution 5-10 times, instead that only one, also this feature allows to save time and money.

The figure below shows the entire stack of Apache, it will now be describe in each part.

![Apache Spark stack diagram](image)

**3.2.1 Spark Core**

Spark core is the center of the framework, it contains the basic functionality of Spark, its API and it is an abstraction layer called Resilient Distributed Datasets, or RDDs. The core contains all the important functions to work with Spark, such as for task scheduling, memory management, fault recovery, interacting with storage systems, and more.

**3.2.1.1 RDDs overview**

RDDs are Spark’s principal programming abstraction, they represent a collection of items distributed across many compute nodes that can be manipulated in parallel. RDD contains also metadata, such as information on the location of their data partitions. It is formally a read-only, immutable distributed collection of objects. Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster. They could be created in different manner. The first method is by referencing datasets in external storage systems, the second is by applying coarse-grained transformation computed on other RDDs. All the sequences of transformations performed are, forming a directed acyclic graph (DAG, called “lineage” in Spark) which is used to calculate its data partitions from the original data. This DAG is used to regenerate an RDD after a failure, ensuring fault-tolerance within an application.
On a RDD is possible to execute two possible operations, the principal different is the way how Spark compute the result:

- Action that compute a new result based on the RDD itself, and returning it to the system or saving in external space. The concept of an action is to do something (e.g. count number of elements) with the dataset and return a final value to driver program.
- Transformation is every operation that give a new RDD for result. The transformation could be done on one RDD (e.g. filter, map, flatMap) or between more than one RDD (e.g. join, union, cogroup).

Usually RDD executes transformation in Lazy evaluation, means that the real execution of the transformation begin only when Spark find an action apply on it. To do this, Spark store only the metadata of the transformation, is possible to see an RDD like a as container of instructions on how to compute the data, and at the moment of the first action on it, it start the real execution of the transformation. The lazy evaluation can appear counter intuitive, but Spark uses this approach to reduce the number of passes it has to take over our data by grouping operations together.

3.2.2 Spark SQL

Spark SQL works on top of Spark core, it introduces a new data abstraction called SchemaRDD and similar to a normal RDD, but it provides support for structured and semi-structured data and can be used interacting with Spark via standard SQL. Each query lunched is transformed in a RDD operation, and it is applied to the SchemaRDD, so the developer can intermix use of SchemaRDD like a normal RDD, manipulating data with complex analytics through domain-specific language (Scala, Java and Python), or treat it as a database table giving to developers the power to integrate SQL commands in only single application.

Spark SQL permit to manipulate different type of source and join it with simple command, it supports many different types of data and can unified them through the SchemaRDD, such as Apache Hive tables, parquet files and JSON files. It also include a server mode that offer a standard connectivity through JDBC or ODBC to run your query.

3.2.3 Spark Streaming

Spark Streaming is general framework built on the Spark Core and it enables the processing live streams of data and real-time data processing, it gives provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data, implemented in a sequences of RDDs.

The streaming component provides API for manipulating data streams, and it is designed to have the same level of fault tolerance, throughput, and scalability. The Spark streaming approach differs from many existing streaming engines, because it uses a micro-batch, mean it captures input from a stream for a pre-defined interval and at the end a batch is created upon which data manipulation are executed. After all, each batch created is stored as a collection of RDD, and these are processed as explained in the previous section.
Spark Streaming is compatible with different types of data source, in fact one or more DStream can be created reading data from two principal categories of sources:

- Basic sources: Sources directly available in the StreamingContext API. Example: file systems, socket connections, and Akka actors.
- Advanced sources: Sources like Kafka, Flume, Kinesis, Twitter, etc. are available through extra utility classes.

In details, every input data, present in form of DStream, is associated with a Receiver object that store the source in the Spark’s memory, where can be processed and transformed with the Spark engine. In the processing phase can be used all the function present in the spark stack (machine learning, graph processing or whatever) and the result obtained can be pushed out to filesystems, databases, and live dashboards.

![Spark Streaming flow](image)

**Figure 3-4. Spark Streaming flow**

### 3.2.4 GraphX

Another component of Spark stack is GraphX, it is a library containing the Spark API and it has the scope to unify graph-parallel and data-parallel computation in one system with a single composable API. The new era of social network brings a different concept of data storage, indeed an inflexible structured data, actually are presented revolutionary representation of data in the system. One of this is exactly a graph representation, which is based on vertex, usually the data with its properties, and edge, a link that can also contain properties which represents a connection between the data.

The graph theory can be applied in a lot of fields, from protein modeling to social networks and targeted advertising and astrophysics, big graphs capture the structure in data and model them with a new free schema concept. The importance of this different representation of data is shown by the increase of new technologies schemaless (without a rigid schema), such as NoSQL database, in fact these database lost the classic model of relational databases.

The GraphX introduce the directed multigraph with properties in the Spark context and support it in the computation, with all the important operations usable in a normal graph (search subgraph, minimum spanning tree, etc.) for a complex data analysis. As DStream of Spark Streaming, GraphX owns its type of RDD, called Resilient Distributed Property Graph, which are partition and distribute graphs that permit a rapid conversion from graph to table, and vice versa. RDP graph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex. It has the ability to support parallel edges
simplifies modeling scenarios where there can be multiple relationships between the same vertices. Vertex and edges have each one own identifiers and GraphX does not impose any ordering constraints on these the identifiers.

As before, the data can be acquire directly in form of graph, all the computation can be done directly on the graph, or transforming it, and is possible to repeat this action each time that developer need it.

### 3.2.5 MLlib

MLlib is a Spark subproject providing primitive machine learning library consisting of common learning algorithms and utilities. Machine learning (ML) and statistical techniques are key to transforming big data into actionable knowledge.

In Big Data the principal main is to extract important value from different fast data inflows, in fact value from such Big Data is a growing concern, and machine learning techniques enable users to extract underlying structure and make predictions from large datasets. In Spark the distributed machine learning algorithms are provided by MLlib component, that

---

**Figure 3-5. Property Graph example**

**Figure 3-6. Graph and table manipulation example**
achieves high-quality algorithms, 100x faster than MapReduce, and also present some lower level ML primitives.

The main algorithms categories offered by MLlib are:

- Classification
- Regression
- Collaborative filtering
- Clustering
- Decomposition

MLlib works directly over Spark, and like other components, is totally integrated with the environment, so is possible to improve the data analysis using the spark components. MLlib was initially developed as part of the MLbase project, which is a platform that aims to make machine learning accessible to a broad audience of users and applicable to various data corpora, ranging from small to very large data sets. It consists of three components MLlib and two others:

- **MLI**: An experimental API for feature extraction and algorithm development that introduces high-level ML programming abstractions.
- **ML Optimizer**: Layer that aims to automating the task of ML pipeline construction. The optimizer solves a search problem over feature extractors and ML algorithms.

MLlib is the principal component used in this work, so it will be explained deeper in the following chapter.

### 3.2.5.1 Spark ML

Starting from the version 1.2 Spark introduces a new package called ML, which is an alpha component that aims to provide a uniform set of high-level APIs that help users create and tune practical machine learning pipelines. ML tries to make machine learning algorithms easier to combine multiple algorithms into a single pipeline, or workflow. Spark ML will not use in this work, for that will not describe in deep\(^1\).

### 3.2.6 Cluster Managers

The last component in the stack is the lower level of Spark stack, the cluster managers. A cluster manager is mostly used to dispatch work for the cluster to perform. Spark includes a simple cluster manager called Standalone Scheduler and also it is developed to work with a variety of cluster, such as Hadoop YARN and Apache Mesos.

---

\(^1\) for a more detailed description, please visit the official guide: ‘http://spark.apache.org/docs/latest/ml-guide.html’
3.2.6.1 How it works

When a user runs an application, Spark submits the work as independent sets of processes to a cluster, the coordinator of the cluster is the SparkContext object, which is initialized in the main program, called driver program. When the program is executed, SparkContext can connect to various types of cluster managers, such as those previously mentioned, which is used to allocate resources across applications. Once connected, the node in the cluster are acquired, and a node is created for each node, which is a process that runs computations and stores data for your application. Finally, the driver program sends code to the executor and the tasks that they have to run.

The figure shows the architecture of a simple cluster (only two nodes) when an application is executed. When an application is started, it gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads, in this manner, each application has a complete isolation from the others on both the scheduling and executor side, but this also denies to share data among the applications, so it is necessary to use an external storage system for the data sharing.

![Figure 3-8. Simple architecture for a run application](image)

3.2.6.2 Cluster manager supported

Spark has different cluster managers already supported, but it is possible to use its own cluster manager.

3.2.6.2.1 Apache Mesos

Apache Mesos is built with the same principles as the Linux kernel, but at a different level of abstraction. The Mesos kernel runs on every machine and provides APIs for resource management and scheduling across entire datacenter and cloud environments. It in Spark determines what machines handle what tasks. Work with Mesos gives the advantage to have
a dynamic partitioning between Spark and other frameworks and to scalable partitioning between multiple instances of Spark.

3.2.6.2.2 YARN
YARN (Yet Another Resource Negotiator) is a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications. It is one of the key features in second-generation Hadoop, basically it is an evolution version of MapReduce, called MapReduce 2.0 (MRv2). Working with YARN gives the advantages to improve reliability, security and easing the debugging process\(^2\).

3.2.6.2.3 Amazon EC2
Amazon EC2 is the central part of Amazon Web Service (AWS) cloud computing platform. In this case, Spark not really works with EC2, but is possible to run it in standalone cluster mode with a script that allows to launch, manage and shut down Spark on Amazon EC2.

3.2.6.2.4 Standalone
Standalone is the cluster manager already include in Spark that makes it easy to set up a cluster. It is possible to launch a standalone cluster either manually, by starting a master and workers by hand, or use our provided launch scripts.

3.3 Spark and Hadoop

In order to understand Spark, is necessary an explanation about his cousin Hadoop, in fact it works in Hadoop ecosystem and tries to improve his lake. It introduces interactive processing missing in Hadoop, which offers only batch processing, and it is compatible with any Hadoop Input/OutputFormat. Even if Hadoop MR is much more mature, Spark make easier the approach with Big Data world.

3.3.1 Hadoop

Hadoop is an open-source Java-based framework and it is the pioneer a fundamentally new way of storing and processing data. It enables distributed parallel processing of big data across inexpensive, industry-standard servers that store and process the data and can scale without limits. It designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance, in fact Hadoop core is divided in two principal parts, the storage part and processing part.

It is composed of 4 principal components:

- **Hadoop Common** – Is the part that contains the libraries and utilities used by other modules and contains the necessary Java ARchive (JAR) files and scripts needed to start Hadoop;

\(^2\) Extract from the article 'Extending Spark on YARN for Enterprise Hadoop’ http://hortonworks.com/blog/
• **Hadoop Distributed File System (HDFS)** – a distributed file-system that stores data on commodity machines, it is built using the Java language;
• **Hadoop YARN** – already described in the previous part;
• **Hadoop MapReduce** – a programming model for large scale data processing.

### 3.3.1.1 Hadoop distributed file system

Data in a Hadoop is split down into smaller blocks and distributed throughout the cluster, its goal is to use commonly available servers in a very large cluster, where each server has a set of inexpensive internal disk drives, in this way functions can be executed on smaller subsets of your larger data sets, and this provides the scalability that is needed for big data processing.

To minimize network congestion and increases the overall throughput of the system, the request by an application is executed near the data it operates on, this makes much more efficient the computation and it is especially true when the size of the data set is huge. Further, HDFS is designed to be highly fault-tolerant and deployed on low-cost hardware, basically a instance consist of hundreds or thousands of server machines, each one storing part of the file system’s data and with a non-trivial probability of failure, in case that a component fails the detection of faults is quick and an automatic recovery start in the HDFS. The HDFS usually manages 3 different replication of each data block to ensure reliability, availability and performance.

The architecture of an HDSF is composed in different part:

- **NameNode** – In a cluster there is a single NameNode, which stores the directory tree of all files in the file system, and records where in the cluster the file data is store, furthermore it executes file system namespace operations like opening, closing, and renaming files and directories. The file system namespace is maintained by the NameNode and all the changes made on namespace or its properties is recorded by it.

Unfortunately, the NameNode is Single Point of Failure in the cluster, in fact when it goes down, the whole system becomes offline, in any case is possible to activate another second NameNode to avoid that the whole system goes down.
- **DataNode** – There are more than one in the cluster, usually one per node. Each DataNode stores data in the HadoopFileSystem in a sequence of blocks of the same size (except for the last block). DataNode manages the data stored in its own node, block creation, replication, deletion, and is responsible for serving read and write requests from the file system’s clients. Different DataNode instances can communicate among themselves, in this way they can manage the replication of the data. When a DataNode goes up, it performs a handshake with the NameNode in order to verify the namespace ID and the software version of the DataNode itself.

- **HDFS Client** – A user access to a HDFS filesystems using a HDFS client through a library that exports the HDFS filesystem interface. The connection is done to a configurable TCP port on the NameNode machine, after it can access to the file systems, files and directories by paths in the namespace without know that file systems metadata and storage are on different servers, or the replicas of the blocks in the nodes.

When a user begin a read operation the HDFS client interrogate the NameNode, which answer with a sorted list (network topology list from the client) of DataNodes, each one containing a replica of the block. Now, the client can contact the DataNode directly to requests the desired block.

For a writing operation, as before, the client asks to the NameNode the list of DataNodes which have the first block desired. After, the client establishes a pipeline from node-to-node and sends the data, until the filling the first block. The client continue with a new DataNodes request for replicas of the next block and organizes other pipeline to transfer data. The process is repeated until the last block is transferred.

### 3.3.1.2 Hadoop MapReduce

MapReduce is the heart of Hadoop, it is a framework that implements the algorithm developed and maintained by the Apache Hadoop project, it help to write application to process a big amount of data with a parallel, distributed algorithm in a cluster. The first version of MapReduce for cluster was implemented by Google in 2004 using the Google File System for input and output, in the Hadoop version, MapReduce works with a Hadoop
Distributed File System (HDFS). MapReduce is based on the traditional paradigm “Divide and Conquer” – split big data to smaller data that can be processed by a machine and then collect the results together.

MapReduce consist in two main phases to process the data, Map phase and Reduce phase. The first, Map, takes a set of data and converts it into another set of data applying different operation, like filtering and sorting, the process is done splitting the input data-set into M, where M is the number or Mapper workers. The split function use to partition the data in M disjoint “bucket” is typically a hash function, when a bucket is filled with independent tuples (key/value pairs) usually of 64/128 MB, called chunk, and the chunk is written to disk, so could be applied the map tasks in a completely parallel manner. At the end is obtained a set of key/value pairs, where a value can be a list of any object, so the framework can sort the results the set, and the intermediate values are supplied to the second phase via an iterator. The second job, Reduce, takes the result from the previous task and combines those data, it merges together these values in R (where R is the number or Reducer workers) independent distinct files to form typically a smaller set of values with some operation. It is mandatory that none of reduce operation start until all mappers have finished. The basic schema of the process is:

- map(k1, v1) → list(k2, v2)
- reduce (k2, list(v2)) → list(v2)
In addition to Reducer and Mapper there are also Partitioner and Combiner. The first handles the division of the intermediate key space and assigns intermediate key-value pairs to reducers. Instead, Combiner work in isolation manner and it allows local aggregation of the tuples before the sort phase, and it is used essentially to save bandwidth.

In case that the number of tasks or jobs exceed the number of available machines in a cluster, in fact there could be occur that different users submits jobs, or a complex job could be split in various tasks. When this happens the scheduler has to take care of maintaining something similar to a queue of pending requests, in Hadoop there are three principals schedulers, FIFO scheduler, Fair scheduler, Capacity Scheduler and also it is offered the possibility to customize own scheduler.

The job execution time depends on the slowest map and reduce tasks, this is called straggler machine problem. It happens when a machine takes an unusually long time to complete a task, it will delay the whole job execution time and degrade the cluster throughput significantly. In order to improve the performance and solve the problem Google proposes and implements speculative execution for MapReduce, which consist to done the task in other backup machines so as to prevent a delay, with the hope that the backup one can finish faster.

### 3.3.1.3 Sequence File

Can happen that Hadoop has to deal with a lot of small files, this can deteriorate the performance, to solve the problem Hadoop uses Sequence File, which is a flat files containing persistent binary key-value pairs, as a container to store many small files.

![Sequence File layout with records or blocks fields](image)

A sequence file is divided in header and body. The header contain different information, such as:

- key/value class names
- version
- file format
- metadata of the file
- sync marker
The body instead is composed either by blocks, in case of a big file, or a record, which is the container of a small file. Each block/record contains a record length, a key length and a pair key-value, and every k byte there is a sync marker separator, that permit in MapReduce to seeking in a random point the file and then re-synchronizing input with record boundaries.

There are 3 different Sequence File formats, depending on whether compression and block compression are desired:

- Uncompressed key/value records
- Block compressed key/value records, in which both keys and values are collected in 'blocks' separately and compressed
- Record compressed key/value records, in which 'values' are compressed here.

### 3.3.1.4 Shuffle and Sort

Shuffle and Sort (SnS) happens at both Map and Reduce side, is magic part of the MapReduce framework, a good understanding allows optimizing both the execution time and the framework, SnS guarantees that the input to every reducer is sorted by key.

In the Map Side is produced an output, it is not simply written to the disk but it includes buffering writes and some presorting. Each map function writes the result in a circular memory buffer associated with, until the size reaches the default threshold. When map function exceeds the threshold (default of 80%) a background thread will start to flush the contents and write to disk, but before the thread first divides the data into partitions, each one corresponding to a different reducer, after they will be write to.

During this procedure the Map function can continue to write to the buffer, but every time that it reaches the buffer size it is blocked. A spill file is created each time that the memory buffer reaches the spill threshold. The spill file can be compressed or not, but is a good practice to use compression because makes it faster to write to disk, saves disk space, and reduces the amount of data. In the map task the disk there could be contains many spill files, but at the end of the task spill files are merged and sorted into a single partitioned output file.

In the Reduce Side are needed the map outputs, but the map tasks may finish at different times, so the reduce task has to wait until all outputs are created. This is known as the copy phase of the reduce task. The map outputs are copied to the reduce task JVM’s memory if they are small enough, otherwise, they are copied to disk and in case that a combiner is specified, it will be run during the merge to reduce the amount of data written to disk. When all the map outputs have been copied, the reduce task invokes a reduce function one for each key in the sorted output, so begin moves the real sort/merge phase which merges the map outputs, maintaining their sort ordering in different rounds. This final merge stage can use data from a mixture of in-memory and on-disk segments. The final output is written directly to the HDFS or other filesystem.
3.3.2 Working together

Spark starts with the same concept of being able to execute MapReduce jobs and essentially it becomes a complement of Hadoop, it makes easier and faster Big Data analysis and the combination of batch, streaming, and interactive analytics, it intended to enhance, not replace, the Hadoop stack, in fact it is able to read and write data to and from an HDFS, it creates an alternative processing engine for workloads that are highly iterative. Spark’s developers took most of the good parts from Hadoop, and improved or changed the bad.

Spark is thought making it as easy as possible for every Hadoop user, whether version is used Hadoop 1.x or Hadoop 2.0 (YARN), and to take advantage all its capabilities. To run Spark on Hadoop in not mandatory to have administration privilege, is always possible to use it, in three principal configuration: standalone, YARN, and SIMR (3).
a) Standalone deployment
With this configuration is possible allocate resources on a subset of machines or in all the machines within a Hadoop cluster and also Spark can share the cluster with Hadoop. Spark users can run jobs, read and write data directly on HDFS.

b) Hadoop Yarn deployment
Spark is totally compatible with YARN, so a user can run it and take all the advantages easily using the Hadoop stack. If is already present a Hadoop Yarn deploy is possible to run Spark over it, without the need to hold administration privilege or any pre-installation.

c) Spark In MapReduce (SIMR)
In case that YARN is not present, is possible to use another configuration in addition to standalone deployment. This option is SIMR that allows to run Spark jobs into MapReduce.

3.3.3 Performance comparison

For the sake of completeness, is given a little overview about the performance. The main difference between Spark and Hadoop is the method used during the execution, in fact Hadoop fits the data on disk, via HDFS, and instead Spark fits the data in RAM memory, as it could. This ensure better performance for Spark if the memory is enough to store the all the data. Spark won the Daytona GraySort contest in 2014, which consist in a benchmark test, done using a fix amount of data, 100 TB in this test, and without fit data in memory. The last record was done by Yahoo using a Hadoop MapReduce.
In the previous part was said that Spark is faster because it uses to store the data in the in-memory cache, but during this test, all the data was stored in the disk (HDFS). The benchmark for GraySort measure the performance of very large sort and not allows compression of input, output or temp files and no reduction of data along the pipeline.. The metric used in this test is terabyte/minute (4; 5).

<table>
<thead>
<tr>
<th></th>
<th>Hadoop MR Record</th>
<th>Spark Record</th>
<th>Spark 1 PB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Size</strong></td>
<td>102.5 TB</td>
<td>100 TB</td>
<td>1000 TB</td>
</tr>
<tr>
<td><strong>Elapsed Time</strong></td>
<td>72 mins</td>
<td>23 mins</td>
<td>234 mins</td>
</tr>
</tbody>
</table>
Analyzing the Table 3-1, the result is evident, Spark has better performance, using 10x fewer machines it completes benchmark test in 23 minutes, against the 72 minutes of Hadoop, so it is 3X times faster. The table shows that the ratio between sort rate and number of nodes, is more than 20X in favor of Spark. To push Spark they did a further test sorting 1 PB of data, and the result is that Spark can complete the job in less 4 hours using only 190 machines, beating the previous result of 16 hours on 3800 machines done with Hadoop MapReduce.

### 3.4 **Spark execution**

In this section is illustrated how Spark works in concrete, the first step to begin is to understand how an RDD is composed and its features.

#### 3.4.1 **RDD in deep**

In the section 3.2.1.1 above was already explained an overview on the RDD, now is analyzed in deep what it contains. It is composed of five parts (7) of information:

- The transformation details on which the RDD depends
- Partitioning scheme metadata
- Data placements metadata
- A set of dependencies on partitions in parent RDDs
- A set of partitions containing the data stored in the RDD

Is important to remember that the RDD is a read-only and immutable after his creation, so each operation (transformation or action) create a new RDD, which contains the five parts said before. So, every time that is created a new RDD, it depends on the partitions of their parent RDD. There are four different type of dependencies based on the type of transformation or action:
• **Narrow** → It is the faster because parent’s and child’s partitions are stored in the same node, this means that is not needed network communication.
  - One-to-one
• **Wide** → In this case the data dependencies from other RDDs might need network communication among different nodes, it generates more overhead and delay in the RDD generation.
  - One-to-many
  - Many-to-one
  - Many-to-many

Also, it is necessary to remember that the RDD is computed in *lazy* mode, so the data is evaluated only when an action operation is called by program or is execute data writing to external storage, thus a runtime can schedule tasks based on data locality to improve performance.

### 3.4.2 DAG (Directed Acyclic Graph)

In order to handle RDDs dependencies is created a DAG, this is one of the Spar keys. It is a new concept that generalize the MapReduce multi-stage execution model and it is the basement for Spark program execution. It is used to model the RDDs lineage, thus Spark can reach significant performance improvements.

In Hadoop MapReduce the sequence of action is always the same, first the data are read from HDFS, after is applied map and reduce, and finally the result is written back to HDFS. In this sequence of action Hadoop doesn’t have a global knowledge of what are happening during MR steps of different jobs, because each MR is isolated and independent from the others. This partial view could deteriorate the performance in case are used many iterative algorithms, because the job can operate many read and write on the filesystem. Spark in order to improve performance over Hadoop used a different approach for each job, it creates a DAG of map-reduce steps considering all the jobs, so it can optimize the globally performance.

Spark’s include many actions, such as reduce, count, collect, etc. An action triggers Spark to construct a DAG composed by multiple stages, they are connected through the shuffle...
operations for intermediate data shuffling and each stage contains many transformation that can be modeled as a pipeline. In every type of MR implementation, intermediate data shuffling is a major performance bottleneck, for this reason Spark tries to avoid the overhead using locale partitions (8).

A DAG concretely is composed by different chunks (called stages), each one representing a calculation that can be computed in on one node and, without the needing of network communication, and containing a list of consecutive RDDs (zero or more) with near dependencies. The narrows dependencies in a single stage are executed as pipeline. All the transformations executed in one chunk produce as output always one RDD.

A simple example is given by consecutive map steps, with DAG is possible to prevent writing data back and forward after each reduce, in practice we can imagine to have jobs A and B that are independent of each other, and another job C needs the results from A and B to complete, so Spark model a DAG to understand the dependencies of the jobs and then it can take the decision to execute A and B in any order and forward the results to C, without store the data in the file system.

Also, Figure 3-12 shows an example of task, in this case the task A and B can be merged in one stage, like C, D and E, so we obtain two stages. Other two stages are B’ and F. When the program is execute the DAG is built and it present a list of the four stages and their dependencies. The scheduler decides the order to resolve the dependencies, in the Figure 3-11 it can choose between A-B and C-D-E. After their execution the job can continue after all the stages are completed and so the program can be terminated (9).

3.4.3 Execution

When a job is submitted first of all the Spark sees if enough memory is available, because is not required checkpointing to a DFS saved on HDD and data are stored in-memory (if the user not require persist HDD). When the execution start we can divide the data in two category, the static data and the intermediate data, these are separated in the initialization step by Spark and only the data that flows through the computation are saved (Figure 3-17).
In this first phase the behavior of Spark and Hadoop are equal, so the map and sort phase are executed in the same manner, instead after the sort in Apache spark are two steps. The first step creates a list containing the collaborating elements for one RDD, in this step the RDD created is static and is possible to perform multiples read on it. In the following step is created a dynamic RDD, which contains intermediate results that can be read only once time and after it is forgotten because in this step the created RDDs are immutable and they can’t be used another times for further storage (Figure 3-16).

Figure 3-16. Schema of the flow in the iterative phase
It is possible to see in the Figure 3-18 the execution schema of the example relative to the Figure 3-15. In the final flows we can see that are not used middle RDDs, in fact is not needed to rebuild the DAG, because we cached it in memory, so the performance are increased. The problem in this approach is the memory capacity, but the approach allows to perform the operations like reduce, join, map and sort can in pipeline, so in not require a global checkpoint anymore.
Spark tries to broaden one’s horizons adding new features and working with other software, in this manner it can be used in different fields, like statistic or simulation of system.

In this work are presented two particular features adding from the version 1.3.0, the first one is SparkR and the second is BlinkDB.
3.5.1 SparkR

R is a free software widely used for statistical computing and graphics, but usually its uses are limited to a single machine. Spark focuses its effort to enable large scale data analysis from R.

SparkR is an open source R package that uses rJava to call Java/Scala code, it could be seen as a lightweight frontend to use Spark from R. It allows users work with the R shell in order to lunch jobs interactively and manipulate the data in a Spark cluster. All the distributed operation are hided to the user, so he can use SparkR without knowing the implementation or the detail of the system. SparkR automatically serializes the necessary variables to execute a function on the cluster and in case of needed, computation module of R can be loaded in the cluster via a built-in function. Also, it is in project to integrate SparkR with Spark's MLlib machine learning library.

To allow the interaction between Spark and R is re-used the concept of RDD, but adapting it converting the object in a Distributed List of R, called 2RDDS, that permit to apply R function to each line of an RDD.

In the Figure 3-19 is shown a data flow example, the user can interact with the SparkContext from a R shell, lunching the job spark as usual divide the workload among the worker, each worker create a R session to execute its task. The result of each worker follow the normal procedure of Spark.

3.5.2 BlinkDB

BlinkDB is a massively parallel, approximate query engine for running interactive SQL queries on large volumes of data. In BlinkDB query are lunch over massive data and the user can have the ability to trade between accuracy and response time, ensuring that the result is still meaningful.

This new approach is necessary because data sets continue to grow, and could become impossible to analyze all the data in a limit time, so BlinkDB based is result not looking at all the data, but rather operating on statistical samples of the datasets, giving a response faster than normal database, but it needs to associate to each query an error bars.

BlinkDB is based on two principal point, the first one is a dynamic sample selection strategy that selects an appropriately sized sample based on a query’s accuracy or response time requirements, and the second is adaptive optimization framework that builds and maintains a set of multi-dimensional stratified samples from original data over time (10).
Spark helps BlinkDB to deal with a large dataset providing the API to work in a cluster and make faster the computation using its approach of in-memory MapReduce. This is possible adding over SparkSQL or Shark a new component to interact with BlinkDB.

### 3.6 Spark Real Use Cases

Spark evolves very fast over the time, always more it widens its application, and always more the companies are interested to deal with it. Many important companies starting to use Spark to face their needs. In this part are presented some interesting use cases of Spark in the real world.

#### 3.6.1 Yahoo!

Yahoo! tries to make the world’s daily habits inspiring and entertaining focusing to the transform the users’ experiences in a highly personalized and optimized experiences. But the bottleneck of this ‘dream’ is the Big Data, in fact create a personal dream for each user can become critical for their hundreds of millions of users.

Spark enter in the dream creation for the yahoo users, two projects are adopted by yahoo. The first one is for personalizing news pages for Web visitors, to reach this objective are used ML algorithms running on Spark, in order to figure out what individual users are interested in, furthermore it is used to to categorize news stories as they arise to figure out what types of users would be interested in reading them. The second project is running Spark to analytics for advertising. The projects require the processing power of the entire cluster and Yahoo decides to use Spark to improve performance of its iterative model training.

The reason that bring Yahoo to use Spark is simple, when a user want a personalization page, the system needs to react fast to understand what the user is doing and to answer on the events happening in the outside world. When a user look at Yahoo’s home page the system has to choose which items are going to show. In order to decide is necessary to learn something about each news item as it comes in to see what users may like it. And in the same time is necessary to learn something about users as they click around to figure out that they’re interest in a topic.

Yahoo address this problem using Spark, it wrote a Spark ML algorithm 120 lines of Scala that going to replace the previous code of ML algorithm for news personalization...
written in 15,000 lines of C++. This code is used to trains a model on a large, hundred million record data set, and in less than 30 minute the Scala ML algorithm was ready for business.

The second project is born with the needs of use existing BI tools to view and query their advertising analytic data collected in Hadoop. Yahoo uses Hive on Spark, called Shark, to allow interactive capability. The advantage of this is that Shark uses the standard Hive server API, so any tool that plugs into Hive, like Tableau, automatically works with Shark, and as a result they were able speed up the interactive and complex analytics and query (11).

3.6.2 Conviva

Conviva is one of the largest video companies on the internet, managing over 4 billion video streams per month, second only to YouTube. They has a dynamical approach to select and optimize sources, while the video is playing, to maximize quality; this could be extremely critical in terms of time, if a viewer has to wait much of 1 second buffering, he can decide to leave the platform.

To react before the buffer runs out, switching to a different stream, it requires pretty sophisticated behind-the-scenes technology to ensure a high quality of service. Conviva uses Spark Streaming to learn network conditions in real-time and so it can feeds directly the video player, to optimize the streams and guarantee the QoS avoiding dreaded screen buffering.

The process to analyze the network conditions starts with an overview of how data flows through the Conviva infrastructure. Conviva monitoring and optimization code embedded and collects non-personally identifiable information like buffering time and frame rate, all these data are sent to Conviva’s
backend infrastructure every few seconds. After, the data is replicated into two streams, the first is sent into Conviva custom live data processing stack. The other stream is stored into a Hadoop File System cluster. MapReduce jobs summarize this raw data and extract out the historical trending information shown in Pulse. The raw data written to HDFS, together with the summaries calculated by the MapReduce jobs, constitute the input to our Hive and Spark based offline reporting and ad-hoc analysis infrastructure, which is described next. Spark manipulate (filter and other operation) and group the data by video name and after with scala code it read from HDFS the data and executes the map-reduce job to produce the results (12).

Spark shows is power when are issued multiple queries on the same data stored in caching, for example can be submitted subsequent queries to find the video counts per country, state and city in addition to the video name.

3.6.3 ClearStory

ClearStory Data is a developer of data analytics software that brings next-generation Data Intelligence to everyone in order to accelerate the way businesses get answers across any number of data sources. It is specialized in data harmonization and helping users simplify data access to internal and external sources. ClearStory Data’s end-to-end solution includes an integrated platform and incredibly simple user application.

It was one of Databricks first customers and choose Spark to figure out a way to help business users to merge their internal data sources with external sources without requiring complex data modeling, an example of data can be the data generated by social media traffic and public data feeds. The normal practice of union the data and correctly preparing it for analysis is quite difficult in real time.

![Figure 3-23. ClearStory process include in-memory computation done by Spark](image)

The company needed Spark for an in-memory engine that could interface with the Hadoop Distributed File System, when users upload data into the ClearStory service, it's stored on a
HDFS and the result is the needs to process huge amounts data in real time. So, ClearStory is the part of front-end that sits atop of numerous data sources, this infrastructure is totally managed by ClearStory, and Spark work as a back-end. Spark carry out data inference and profiling, discovers relationships between different sources of data. The data is then blended, and notable overlaps and correlations exposed after processing in Apache Spark's core in-memory query-optimization engine. It can then give a harmonized data to users via the ClearStory application, so many users in the user same time explore company data or add data without carrying out any additional modelling (13).

Today, ClearStory can be considered one of the most visible success stories for Apache Spark, which is faster solution that has replaced the old MapReduce in the Hadoop stack.
4. IMPLEMENTATION

In this chapter will be introduce the principal instruments to deal with Apache Spark, the step necessary to manipulate and obtain the dataset that was used to create the application to test the model implemented with MLlib.

4.1 TOOLS

To begin with the Apache Spark the first step is to choose the language among the three principals offered by Spark: Java, Python and Scala.

Python is a dynamic programming language that offer a big community support and a wide standard library, it uses a simple and efficient approach to the object-oriented programming. Python is perfectly integrated with apache and it has its own environment for working (PySpark), it is useful to interact with other framework (e.g. lightning, thunder) but has lacks in the Machine Learning development, in fact at the begin of Spark the mayor part of the examples and algorithms implantation were done in Scala and Java. Since this work focusing to implement an application with the help of Machine Learning library of Spark, is not take in account the possibility to use Python, and is only use Scala language. During the implementation phase of my application Spark had important changes and improvements, in fact I started using the version 1.0.0 and finished using the version 1.2.1, in this version range many problems of Spark were fixed and interesting APIs and components were added.

Now is introduce a brief overview about Scala and its advantages, next how to create a project with Apache Spark.

4.1.1 Scala language

Scala, which name means “SCAlable Language” is a JVM language created to be a “better Java”, it started in 2001 and it was created to solve the needs of the professional developers. At the begin deal with Scala seems more difficult because is necessary to have competence in OOP and FP languages. Normal behavior of metaprogramming has to be reached using type-safe and mixin composition through traits. The difficult to begin with Scala is rewarded with efficiency, coherent, modular, and maintainable applications.

The important features are (14):

- Statically typed → It binds the type to a variable for the lifetime of that variable
- Mixed paradigm → Support for object-oriented programming (OOP), adding also the possibility to implementing classes with mixing composition using traits, and functional programming (FP), useful for concurrency applications.
- JVM and .NET → Scala is primarily a JVM language, it generates a JVM byte code and it is based on Java paradigms, but is under development a .NET version that can generate a .NET byte code.
- Elegant and flexible → The syntax of Scala uses a number of techniques to minimize unnecessary elements, it minimizes the declarations, the function creation code and the need for explicit type information.
- Sophisticated type system → Scala makes the Java system type more flexible.
- Scalable → Scala has the peculiarity that can scale starting from small interpreted scripts to large distributed applications.

In the Example 4.1 and Example 4.2 are shown Java code versus Scala code, in which is implemented the class “Order” with relative getting and setter method, is evident the compactness of Scala compared to Java, it makes programming less tedious and more enjoyable, but sometimes the readability and comprehensiveness could be lost.

In Scala, all the work is done by compiler that translate the entire code in java. It is incredibly smart and helps the developer to avoid to specify explicitly those things that the compiler can infer. For a deeper understanding and to a complete explanation refer to read (14).

4.1.2 SBT

SBT is an interactive open source tool, the name means Scala Build Tool, and it help Scala and Java developers to solve dependencies in their projects, it is similar to Java's Maven or
Ant. SBT has the complete support for continuous compilation of Scala code, and integrate many framework for Scala testing and with Scala interpreter for helping iteration and debugging, but in addition to Scala projects it provides support. The build descriptions are written in Scala using a DSL and the dependencies are solved and managed with Ivy. SBT offers two features, an incremental compiler, which will only compile what needs to be recompiled after changes, and an interactive shell that offer command to compile, run and test the code and it allows to modify build settings on the fly.

In practice builds projects with the same version of Scala used to run sbt itself. To use interactive sbt after the installation is enough write in a shell the command “sbt”, instead for the batch mode, specifying a space-separated list of sbt commands as arguments, such as “sbt compile”.

Comparing with Maven, sbt is totally effortless for simple project, it allows to build file in terms of a Scala DSL instead of XML, and so the result is a simplification of the dependencies file. In the Example 4-3 is possible to see the difference between a dependencies files of Maven and SBT for work with spark, is immediate to understand the easiness of sbt. To a complete guide of SBT see the website (15).

### 4.1.2.1 SBT, Eclipse and Apache Spark

```scala
name := "Simple Spark Project"
version := "1.0"
scalaVersion := "2.10.4"
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.2.0"
```

```xml
<project>

    <name>Simple Spark Project</name>
    <packaging>jar</packaging>
    <version>1.0</version>

    <dependencies>

        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.10</artifactId>
            <version>1.2.0</version>
        </dependency>

    </dependencies>

</project>
```

Example 4-3. "simple.sbt" file (above) and "pom.xml" file (bottom) to create a simple project and solve Apache Spark dependency
Due to its simplicity in my work I decided to use SBT against Maven. To start my project I choose to work with Scala IDE for Eclipse \(^3\) that provides many useful features for professional developers:

- Code Completion
- Semantic Highlight
- Error Markers
- Definition/Implicit Hyperlinking
- A integrate Scala Debugger
- Plugin for work with GitHub \(^4\)
- Plugins for work with SBT \(^5\)

To start with Apache Spark, SBT and Eclipse we have to do some steps. First of all we have to choose Apache Spark, the website offer different possibilities:

- Source Code → in this case after the download the source code has to be compiled, the easiest way is use sbt. Changing the current position in the directory of Apache Spark and running the command “sbt assembly”, the source code is automatically compiled with all the necessary dependencies.
- Pre-built for Hadoop X → it is sufficient to download and unzip the package for have a working version of spark that is developed on the “X” version of Hadoop.
- Pre-built for MapR X → this version is for working on MapReduce mode presented also in previous chapter.
- Pre-built for CDH4 X → CDH4 is Cloudera's Distribution Including Apache Hadoop is own version of Cloudera to work with a cluster.

I chose to use the source code, in this manner I could avoid to load all the library of spark, and I selected only the principal package necessary to my work. The principal packages that are needed to have a basic version of spark. The following packages are included in my project from begin:

- log4j → is the logger used to write in the console of apache spark, this logger can be set for choosing which type of message will be shown depending to the output mode (console in interactive mode, shell in script mode or in a IDE console)
- spark-core → contains all the fundamental APIs to use spark, from the creation of the Spark Context to all the functions for create and manipulate a RDD
- scalap → is the package that it is necessary to use scala language with spark.

Working with Eclipse and Spark are not immediate as it can be seen, in fact to deal with them some instructions have to be followed. First, the structure for a complete project has a rigid schema and can’t be changed. The schema is the same used by Maven and can be of two type. For a simple project (usually one

\(^3\) The software can be downloaded at [http://scala-ide.org/](http://scala-ide.org/)

\(^4\) The project is available at [http://eclipse.org/egit/](http://eclipse.org/egit/)

\(^5\) The project is available at [https://github.com/typesafehub/sbteclipse/](https://github.com/typesafehub/sbteclipse/)
file of code) is enough to create a simple sbt file, and put it in the same directory of the scala (or java project) as shown in the Example 4-4.

```sbt
simple.sbt
project/
  simple.scala
```

*Example 4-4. Structure for a simple project*

Instead for a more complex projects, containing different source code files and test code files, the structure of the directories has to be a little more complex and it is shown in the Example 4-5.

```sbt
simple.sbt
src/
  main/
    resources/
      <files to include in main jar here>
    scala/
      <main Scala sources>
  java/
    <main Java sources>
  test/
    resources
      <files to include in test jar here>
    scala/
      <test Scala sources>
  java/
    <test Java sources>
```

*Example 4-5. Structure for a complex project*

After this step is possible to create the necessary files and compile the project with the command “sbt compile”. To integrate the project in eclipse are needed few expedients, first of all we have to create a folder called “project” inside the root of your project and inside generate a new file for sbt called “plugins.sbt” in the project directory and add this line of code to this file:

```scala
addSbtPlugin("com.typesafe.sbteclipse" % "sbteclipse-plugin" % "2.5.0")
```

To create the project that has to be imported in eclipse is sufficient use the command “sbt eclipse” in the directory of the project. SBT automatically creates the folders .classpath and .project necessary to import the project into Eclipse. Summarize your root folder has to contains a src directory, a simple.sbt file, a ./project/plugins.sbt file and the folder necessary to Eclipse.
4.2 **THE DATASET**

In this work are used two principals dataset. In the first part of my studies I used a littler dataset deriving by the World Cup 2014.

This dataset consist in users of Instagram that share picture during the world cup and who used the tag “worldcup” and similar tags. At begin the data set was composed by 500 user images, of each pictures was download the relative JSON file, in this manner was possible to extrapolate all the tags used in the picture and other information. Unfortunately Instagram not release many information about the user, but analyzing the metadata, other information can be retrieved, such as the city, using the coordinates, the language, or the moment when the picture was published.

In this case to tries the capabilities of Spark all the 500 images and all the 500 json files, were stored in two Hadoop Sequence File, this was obtained with a shell script that initially generate a tarball of the files, obtained the tarball file it can be manipualted with a java program to generate a Hadoop sequence file.

To work with HSF spark offers simple APIs that allow to read, write and manipulate them. When a HSF is read in Spark, it created immediately and RDD to store all the information about the data, we already sew, that in a RDD are not stored the data, but only the step to obtain the data. Applying to an RDD a map function, obtained from HSF reading, is possible to work directly with each file, in fact every “line” contains a file, in our case a json file or an image.

This dataset was used principally to deal with Hadoop Sequence File and to understand how to integrate this type of file in Apache Spark. Also was used to start with the basic APIs of spark and to try simple function of MLlib and Spark SQL.

Finished to work with Hadoop sequence file part, another dataset was used, because in this new process is necessary a greater set of data.

In the second part the dataset is composed by two different typologies. The first typology, positive type, was taken a group of user that belong to a certain group of Instagram, we can call them group A.

The second typology, negative type, contains users that should not belong to the first group, so we can suppose that this group B, is equal to a ‘not A’.

In order to obtain the first group ‘A’ were selected some tags belonging to the group of interest, and after were taken the users that uses these tags. In this way was created a database of the users that follow a given topic and for each one were downloaded image and relative metadata that they shared.

Instead, create a negative data sample was a tortuous process, in fact is impossible a priori to said that a user not belong to a group, a given user can not follow a topic, in the present, but he can become a follower and maturate interest in this topic in the future. So, the negative

---

6 For a complete guide see [http://personals.ac.upc.edu/rtous/howto_spark_opencv.xhtml](http://personals.ac.upc.edu/rtous/howto_spark_opencv.xhtml)
a data sample was created catching that in the present not belong to group A, users were selected changing some characteristics, such as city, following and used tags.

In order to understand how the database is composed, the similarities and diversities of two samples, are shown some statistics.
First, the characteristics of the final database are shown in the following table:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of ‘positive’ users</strong></td>
<td>1.073</td>
</tr>
<tr>
<td><strong>Number of ‘negative’ users</strong></td>
<td>3.694</td>
</tr>
<tr>
<td><strong>Total numbers of tags data sample</strong></td>
<td>1.827.097</td>
</tr>
<tr>
<td><strong>Total numbers of tags of ‘positive’ data sample</strong></td>
<td>22.7254</td>
</tr>
<tr>
<td><strong>Total numbers of tags of ‘negative’ data sample</strong></td>
<td>1.599.843</td>
</tr>
<tr>
<td><strong>Total different tags in database</strong></td>
<td>176.057</td>
</tr>
</tbody>
</table>

*Table 4-1. Summary of database characteristic*

In the Figure 4-1 is represented two histograms of the most used tags in each data sample, the first one is negative, and the second one the positive. Is possible to see that some tags are common for both data sample, for this reason we cannot ensure that all the element of negative data sample B not belong instead to positive data sample A.
4.3 Algorithms

Figure 4.1. Most used tags
In this section will be explained the methods used to convert and normalize the data and the algorithms used to create the followers classifiers.

The first step to obtain a compatible input is to extract from a set of files the relevant data. From all the files were taken the tag field, which contains a vector of words used to comment the relate picture, and were stored in a single file. The position of each picture also was taken and converted and merged in the tag vector. So, with this method were created the two data sample file, positive and negative. The files contain a json objects, one for each follower, the structure of one json object is shown in the Figure 4-2.

When the files are obtained, three method to convert json data to numeric values usable from machine learning algorithms.

4.3.1 TF-IDF

Term frequency-inverse document frequency (TF-IDF) is used in information retrieval and it is a feature vectorization method widely used in text mining useful to calculate the frequency of terms in a document and to correct the value based on the importance of a term in a document. Formally, giving a term $t$, a document by $d$, and the collection of document denoted by $D$ we can said that: “Term Frequency $TF(t,d)$ is the number of times that term $t$ appears in document $d$, while Document Frequency $DF(t,D)$ is the number of documents that contains term $t$”\(^7\). A correction factor is adopted because the only use of term frequency can easily to over-estimate the importance of some terms that appear more frequently than others words but giving less information about the document, for example some terms that fall in this category are “a”, “the”, and “of”. The idea that introduce this correction is that if a term is repeated many times in a collection of document, is probable that not bring relevant information for the document. So, the inverse document frequency is a numerical measure of how much information a term provides:

$$IDF(t, D) = \log |D| + 1DF(t, D) + 1$$

where $|D|$ is number of document present in the collection.

The TF-IDF measure is simply the product of TF and IDF:

---

\(^7\) https://spark.apache.org/docs/1.2.0/mllib-feature-extraction.html
For an example of TF-IDF we can consider a document containing 1000 words wherein the term ‘car’ appears 4 times. It is simple to calculate the term frequency for ‘car’, in fact it is \( \frac{4}{1000} = 0.004 \).

Now, assume we have a collection of 100 million documents and the term cat appears in one thousands of these, so we can calculate the inverse document frequency as \( \log(100,000,000 / 1,000) = 40 \). The final measure as we said before, is simply the product of both, so the TF-IDF: \( 0.004 \times 40 = 0.16 \).

In my project the TF-IDF was applied to both data samples and each one gave as result a RDD of vectors, one for each user. Inside a vector is stored the result of TF-IDF for the relate user in form of float number.

### 4.3.2 Feature Hashing and bigrams

Talking about text classification, we can introduce some concepts, the first is the ‘Unigrams’ that is considered a set of single words, the second, most important in our case, is ‘Bigrams’ that introduce the concept of relation between two words which appear next to each other in the text, and finally the ‘Trigram’ that it is just the next extension of that concept. Bigrams are commonly used for statistical analysis of text in many applications, then can be useful for a classification algorithm, in fact it tends to really improve performance helping the classifier to determine which class has a higher probability for a better classifications.

A definition of bigram could be “2-gram is a contiguous sequence of 2 items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application”.

If we consider a bigram which has a letter as single item, we can calculate that the total possible pairs constructible in a natural language, depurate the string from special characters, are \( 26^2 = 676 \).

In this work to implement the calculation of bigram frequency was used the Hashing Trick, or formally ‘Feature hashing’. It is a fast way to featurize a vector useful to preserve sparsity and introduces no additional overhead to store projection matrices.

The concept is very simple, to each element, in this case a bigram, is applied a hash function, and the hash values obtained is used as indices directly, rather than looking the indices up in an associative array.

In Scala the hashing trick could implemented in easy way, to each data sample is applied a map function that takes the tags vectors and then apply the function to convert the tags in bigrams, after with all the bigrams is create an array with hashing trick to count the occurrences and finally the frequency.

We can take as example the sentences below and see the example in the Table 4-2:
"The sun is in the sky"

<table>
<thead>
<tr>
<th>Bigram</th>
<th>Hash Value/Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>th</td>
<td>5</td>
</tr>
<tr>
<td>he</td>
<td>6</td>
</tr>
<tr>
<td>su</td>
<td>4</td>
</tr>
<tr>
<td>un</td>
<td>1</td>
</tr>
<tr>
<td>is</td>
<td>3</td>
</tr>
<tr>
<td>sk</td>
<td>2</td>
</tr>
<tr>
<td>ky</td>
<td>7</td>
</tr>
<tr>
<td>in</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Count</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.2</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>Tot</td>
<td>10</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4-2. Example of bigram and hashing trick combination

### 4.3.3 Counting Frequency

Another algorithm used for compute the frequency of the tags is a simple frequency calculator. It can be seen as a TF-IDF without the value document frequency as correction factor.

The purpose to use this simple algorithm is based on the idea that in a document compose of only tags, is more difficult to find useless term, in fact thinking about the use of picture and tags that a user do.

Supposing the no presence of irrelevant terms (e.g. ‘the’, ‘of’…), that in the tags are not often used, apply the TF-IDF can negative influence relevant tags. Analyzing for example the Figure 4-1, the most used terms are not particle words and they seem to carry important information to the text.

To avoid a collision problem and to under-estimate the size of a hash map, due to a big dictionary generated by the huge variety of tags, were used two array, the first one is create to associate a tag with a corresponding index as value, from 0 to number of different tags, and the second for the count, where the index is get by the first array, that contains the total number of times that a single tag appear in the data sample. A clarification is given by the example below.

Using the series of tags

"summer sky summer sun sky sun sky sky summer"

We can create the following arrays:
<table>
<thead>
<tr>
<th>First Array</th>
<th>summer 3</th>
<th>Second Array</th>
<th>1 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sun 1</td>
<td></td>
<td>2 3</td>
</tr>
<tr>
<td></td>
<td>sky 2</td>
<td></td>
<td>3 4</td>
</tr>
</tbody>
</table>

Table 4.3. Creation example of counting arrays

These algorithms are used to extract the information from a data sample, commonly this technique are part of information retrieval (IR) processes. These technique are used to obtaining information resources relevant to an information need from a collection of information resources. The relevant information are used in our case as input to create and training the classifier model.

In order to train the model were used three principal algorithms that will be explained in the following section. The first two algorithms are part of binary classification method, instead the third is multiclass classification algorithm.

Knowing that our goal is to find a model to predict if a user can belong, or in this case follow, a group or a brand, so immediately we can understand the need to use a binary classification in fact these types of algorithms have the scope to divide items into two categories: positive and negative, follower or not follower, in the group or out of the group. Actually in MLlib are implemented two principal linear methods for binary classification: linear Support Vector Machines (SVMs) and logistic regression. These algorithms need an RDD of LabeledPoint as input. A LabelPoint is a simple vector, of floating point, with a label put at begin, a training label could be denoted as either +1 (positive) or −1 (negative).

4.3.4 SVM – Support Vector Machine

SVM is a relatively new type of learning algorithm, introduced by Vapnik and Corinna Cortes in 1993 and published in 1995. The idea is to take a set of training examples, in our case the two data sample, negative and positive, and with an SVM training algorithm builds a model that can predict for a new entry one category or the other. An SVM model is represented by a hyper-plane that divides set of points in space, in order to improve the classification the separate categories are divided by a clear gap wide as possible.

The SVM can construct one or a set of hyperplanes 1 to infinite-dimensional space, which can be used for classification. Larger is the distance from the hyperplane to the nearest training data point of any class better is the quality of the model created. This distance is called functional margin, so larger

Figure 4-3. Example of hyperplane that separate points in two different categories
is the margin, and lower is the generalization error of the classifier. Formally, given some training data set $D$, a set of $n$ points of the form:

$$D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$$

where the point $x_i$ belong to the class $y_i$ that could be either 1 or $-1$; $x_i$ is a vector belong to a p-dimensional space. The scope is to figure out the maximum-margin hyperplane that can divide the points having $y_i = 1$ from those that have $y_i = -1$.

The definition of a hyperplan is:

$$\begin{align*}
    x_i \cdot w - b &\geq +1 \quad \text{when} \ y_i = +1 \\
    x_i \cdot w - b &\leq +1 \quad \text{when} \ y_i = -1
\end{align*}$$

From this equation is possible to calculate the two hyperplanes $H_1$ and $H_2$, the SVM algorithm tries to maximize the distance between them, so the problem is basically a maximization problem that can be resolve with linear methods.

MLlib include APIs to training the model with the SVM algorithm, but only for linear hyperplans, in fact SVM can also work with nonlinear classification, using for example polynomial equations.

With MLlib the SVM model is trained using a part of RDD of LabeledPoint as training set, all the calculus and communication between workers necessary to find the model in distributed system are done automatically by spark.

4.3.5 Logistic Regression

The name logistic regression comes from the fact that the data curve is compressed by using a logistic transformation, to minimize the effect of extreme values.

Logistic regression is used to model the relationship between a dependent and one or more independent variables and estimates the probability of an event occurring, in fact it wants to predict from a knowledge of relevant independent variables is not a precise numerical value of a dependent variable, but rather the probability that it is 1 event rather than 0 event. In others words it is used to predict a binary response from a binary predictor, basing the outcome of a categorical dependent variables, on one or more predictor variables. The term logistic regression is strictly used to refer to problems in which the dependent variable is binary, instead the problem has more categories the model is referred to multinomial logistic regression or ordered logistic regression.

Logistic regression treats the same set of problems a using similar techniques, the first assumes a logistic function and the second a standard normal distribution function.
Logistic regression can be seen as a special case of generalized linear model and thus analogous to linear regression. In this algorithm, the result predicted by the model is usually coded as "0" or "1", true or false, and in our case follower or not follower.

Essentially what it does is to create a model that “draws a line” through the data, and says if this data point falls on one side, assign it a label 0, or if it falls on the other side of the line, give it a label 1.

In Logistic Regression the effect of each input on the output is measured, and the various inputs are weighted in the finished model, for do this it is necessary that logistic regression use a logistic function to create continues criterion, formally in logistic regression, the dependent variable is in fact a logit, which is a log of odds, that has the formula:

\[
\text{logit}(P) = \ln\left(\frac{P}{1-P}\right)
\]

Where \( P \) is the probability of a “1” event and it is equal to:

\[
P = \frac{e^{\alpha + \beta x}}{1 + e^{\alpha + \beta x}}
\]

\( \alpha \) and \( \beta \) are the parameters of the model.

In MLlib, as SVM, the input format expected in the Logistic Regression algorithm implementation in Spark is a LabeledPoint array, in which the first element is the class, that could be 0 or 1, and all elements after the first elements are the features, which must a float datatype.

4.3.6 Naive Bayes

Naive Bayes is different from the previous algorithms because is a multiclass classification algorithm and is necessary to assume the independence between every pair of features. The peculiarity of Naïve Bayes is that is very efficiently to train and it also give a high quality prediction.

It is simple technique, or better a family of algorithms, used in our case to construct a classifiers. This group of algorithms is based on common principle:

- The value of a particular feature is independent of the value of any other feature
- Each of the features is considered to contribute independently to the probability of one category
Some of these algorithms belonging to naive Bayes classifiers can be trained very efficiently in a supervised learning setting, to improve the quality of the classifier can be chosen different parameter estimation, for example in MLlib is possible to modify the s ‘additive smoothing’ parameter $\lambda$. Naive Bayes has the advantage that is enough a small amount of training data to estimate the parameters necessary for classification, and despite this and its design it works well in many complex real-world situations and can often outperform more sophisticated classification methods.

Formally (16), given a set of variables $X = \{x_1, x_2, x_\ldots, x_d\}$, the algorithm aims to construct the posterior probability for the event $C_j$ among a set of possible outcomes $C = \{c_1, c_2, c_\ldots, c_d\}$, so $X$ is the predictors and $C$ is the set of categorical levels present in the dependent variable. The Bayes’ rule claims that

$$p(C_j | x_1, x_2, x_\ldots, x_d) \propto p(x_1, x_2, x_\ldots, x_d | C_j)p(C_j)$$

Where $p(C_j | x_1, x_2, x_\ldots, x_d)$the probability that X is belongs to $C_j$, but assuming the independency of the variables and using the conditional probabilities we can obtain:

$$p(X | C_j) \propto p(C_j) \prod_{k=1}^d p(x_k | C_j)$$

Using this formula we can label a new case $X$ with a class level $C_j$.\(^9\)

Like the previous algorithm also here, Naïve Bayes takes an RDD of LabeledPoint to create a model which can be used for evaluation and prediction. In this work the Naïve Bayes, even if is a multiclass classifier, is used as a binary classification, so the model predict only if a new income belongs a positive or negative category.

---

\(^8\) For more information [http://en.wikipedia.org/wiki/Lidstone_smoothing](http://en.wikipedia.org/wiki/Lidstone_smoothing)

\(^9\) An example with explanation could be find at [http://sebastianraschka.com/Articles/2014_naive_bayes_1.html](http://sebastianraschka.com/Articles/2014_naive_bayes_1.html)
5. **IMPLEMENTATION IN A CLUSTER**

This chapter is divided in two sections, the first one contains the architecture and the features of the cluster used to work, and in the second part is explained some step to do for implement Spark in the cluster.

5.1 **BSC**

The Barcelona Supercomputing Center - BSC - is center located in Barcelona, Catalonia, Spain, it is a public research center that hosts different supercomputers, the principal one is MareNostrum, a 63.8 TFLOPS PowerPC-based supercomputer ranked 465th in the world, and a newer one called MareNostrum II that is a 103.2 TFLOPS supercomputer using Xeon E5649 processors.

Geographically it is located at the Polytechnic University of Catalonia (UPC) in a former chapel named Torre Girona, the MareNostrum supercomputer is contained inside an enormous glass box inside this chapel. The maintenance is divided among Spanish Ministry of Education and Science (51%), the Government of Catalonia (37%) and the UPC (12%). Professor Mateo Valero is its main administrator.

MareNostrum is one of the most powerful supercomputers in Europe. The mission of BSCS is to investigate, develop and manage information technology in order to facilitate scientific progress. The main areas of specialization of BSC are Computer Sciences, Life Sciences, Earth Sciences and Computational Applications in Science and Engineering. The iteration among the difference discipline allows cooperation and multidisciplinary loop can be set up. The structure that result from this complementary improve the research work and the way to service society.

Supercomputing is a powerful tool that aims to solve the most complex problems in science, engineering and business relevant areas. Nowadays, supercomputers are used for modeling and simulating complex behaviors, processing large information sets, solving complex computational problems and designing complex systems, they are used to a wide range of computationally intensive tasks in various fields, such as quantum mechanics, weather forecasting, climate research, oil and gas exploration, molecular modeling and physical simulations and also they have been essential in the field of cryptanalysis.
BSC can contribute to all levels of the problem solution as said before, from the enabling supercomputing infrastructure to a wide range of problem specific skills like modeling and optimization in a wide range of application areas.

In BSC the principal supercomputer is MareNostrum III, which comes from and update after the last agreement was signed between the Spanish government and IBM in August 2012.

5.1.1 History

BSC derive from the tradition of the well-known European Center for Parallelism of Barcelona CEPBA (17), which belongs to the Technical University of Catalonia UPC. CEPBA was a research institution, development and innovation center on efficient computing technologies both for academia and industry.

In 1991 CEPBA started its activities, get together the experience and needs from different UPC departments. These departments were:

- Signal Theory and Communications (TSC),
- Strength of Materials and Structural Engineering (RMEE),
- Computer Systems and Languages (LSI),
- Nuclear Physics and Engineering (FEN) and
- Applied Physics (FA)

From 1995 to 2000 CEPBA coordinated the service activities with CESCA (Supercomputing Center of Catalonia) and in 2000 it signed an agreement with IBM to launch the CEPBA-IBM Research Institute for joining research on topics related with Deep Computing and Architecture, and supporting local research in other areas of science and engineering.

In 2004 the Ministry of Education begin to support the idea of actual BSC advancing the initiative of creating a National Supercomputing Center in Barcelona with UPC and Generalitat de Catalunya, the Catalan Government.

In one year the idea of BSC become official constituted and started its activities. So in 2005 the MareNostrum was built and become part of the most powerful machine in Europe. In less than one year the MareNostrum improve is calculation capacity of two times, and it was confirmed another time against the most powerful supercomputing in Europe.

5.1.2 MareNostrum

Marenostrum is an IBM System X iDataplex based on Intel Sandy Bridge EP processors at 2,6 GHz (eight-core), 2 GB/core (32 GByte/node) and around 500 GByte of local disk. Currently the supercomputer consists on 48,896 Intel Sandy Bridge processors in 3,056 nodes, and 84 Xeon Phi 5110P in 42 nodes (not used in this work), with more than 104.6 TB of main memory and 2 PB of GPFS disk storage. All compute nodes are interconnected through an Infiniband FDR10 network, with a non-blocking fat tree network topology and a peak performance of 1,1 Petaflops. At June 2013, MareNostrum was positioned at the 29th place in the TOP500 list of fastest supercomputers in the world (18).
5.1.2.1 Architecture

The total amount of racks in MareNostrum is 44 (19) and they can be distinguished by the different function in the supercomputer, the categories are:

- 36 Computation racks, in particular it contains 4 Network elements racks are dedicated to network elements which allow to interconnect the different nodes connected to the Myrinet network.
  In this group of racks, each one has a total of 1,344 cores and 2,688 GB of memory for a total of 48,448 Intel SandyBridge cores.
- 1 Operation rack used to manage the system
- 7 Storage Server racks used to storage 20 server, each one provide a total capacity of 280 TB external storage and in total they have 560 disks of 512GB used to work with working with GPFS. The 20 storage nodes has two nodes p615 in charge of the disk requests, a controller type FASTT100 and one unit EXP100.

Each compute rack (20) is BM iDataPlex Compute and contains:

- 84 IBM dx360 M4 compute nodes
- 4 Mellanox 36-port Managed FDR10 IB Switches
- 2 BNT RackSwitch G8052F (Management Network)
- 2 BNT RackSwitch G8052F (GPFS Network)
- 4 Power Distribution Units

The compute nodes are based on Intel Xeon (R) technology and it is composed of:

- Two 8-core Intel Xeon processors E5-2670 at 2.6 GHz, 20 MB cache memory, with a peak performance of 332.8 Gflops per node.
- Eight 4 GB DIMM’s, 1.5V DDR3 @ 1600 MHz. Having 32 GB per node and 2 GB per core. In terms of memory bandwidth, this is the best configuration for memory access for Intel E5-26xx EP processors (4 memory channels per chip).
- Local hard drive: IBM 500 GB 7.2K 6Gbps NL SATA 3.5.
- MPI network card: Mellanox ConnectX-3 Dual Port QDR/FDR10 Mezz Card.
- 2 Gigabit Ethernet network cards (management network and GPFS).

The interconnection networks offer two possibilities; the first one is Infiniband FDR10, high bandwidth network used by parallel applications communications, and the other is Gigabit Ethernet, 10GbitEthernet network used by the GPFS Filesystem.

The operating system used is Linux SuSe Distribution 11 SP3.
5.1.2.1.1 File Systems

In MareNostrum the File System has different typology based on the use that is necessary, the principal types of storage inside the nodes are three:

- Root filesystem
- GPFS filesystems
- Local hard drive

5.1.2.1.2 Root Filesystem

One of the server contains the operating system, which is stored in the root filesystem, and is mounted in form of NFS. A Network File System is a client/server application that allows remote hosts to mount all or a portion of file systems over a network and interact with those file systems as though they are mounted locally.

The root filesystem must be contains all the adequate elements to boot, restore, recover, and/or repair the system, but only data from the operating system has to reside in this filesystem, no other types of data can be stored.

The root filesystem has to be mounted from very small media, in fact keeping it small on servers in networked systems minimizes the amount of lost space for areas of un-shareable files, and this can also allows workstations with smaller local hard drives.

In the case of a cluster for the communication between client and server is used the TCP/IP protocol.

5.1.2.1.3 GPFS Filesystem
IBM General Parallel File System (GPFS) is a scalable high-performance file management infrastructure, it is a high-performance enterprise file management, that help to simply adding storage to optimizing data management for shared-disk file system and it provides fast, reliable data access from all nodes of the cluster to a global filesystem.

GPFS is used on many of the largest supercomputers in the world, it is implemented also in MareNostrum 10 in order to allow the user to work with parallel applications and then to simultaneous access on one or more files, the access is permitted from any node that has the GPFS file system mounted and each file can be accessed with different privilege basing on the needs (21).

GPFS provides high level of control over all file system operations and, in addition, can read or write large blocks of data in a single I/O operation, so the overhead is minimized. In case of MareNostrum a daily incremental backup is performed for the principal paths of the system, in this case the pat are ‘/gpfs/home’ and ‘/gpfs/projects’.

MareNostrum allows access to different GPFS filesystems to all the machine from all nodes, the shared paths are:

- /apps: In this path reside all the applications and libraries that have already been installed on the machine. In this folder is also presented the Apache Spark implemented in MareNostrum.
- /gpfs/home: In this part of filesystem are stored all the user directories, this means that when user access to the cluster via SSH the home default directory is in this path. Every user will have their own home directory to store own developed sources and their personal data. This directory is recommended to run the job of the users.
- /gpfs/projects: This directory is used to give an additional support to a user group, each group of users has in this directory other space in order to store data that needs to be shared between the users of the same group or project.
- /gpfs/scratch: The space in this directory is used to store temporary files of your jobs during their execution.

5.1.2.1.4 Local Hard Drive

10 For a complete guide about GPFS and MareNostrum refer to http://www.bsc.es/media/513.pdf and http://www.bsc.es/media/512.pdf
Every node has an internal hard drive and this space is mounted over /scratch/tmp directory. This space is used as a local scratch space to store temporary files during executions of one of your jobs for a total of about 500 GB.

5.1.2.2 LSF

MareNostrum used LSF utility to run jobs, it is used to support batch processing in the cluster.

The Platform LSF stand for Load Sharing Facility, it is a job scheduling and monitoring software system developed and maintained by Platform Computing. It helps to balance the workload on a central computational servers, further it gives you access to the software and hardware you need to get your work done regardless of where you are logged in. LSF takes care of that batch management; based on the job specifications LSF will start execution of jobs when there are enough system resources available for the job to complete, when the job is submit it automatically balances load and allocates resources, and provides access to those resources (22).

LSF allows to use a network of heterogeneous computers as a single system, so there is not the need to rewrite or change a program to take advantage of LSF.

![Cluster](image)

*Figure 5-4. LSF schema.*

It can run batch jobs automatically when required resources become available, or when systems are
lightly loaded. It maintains full control over the jobs, including the ability to suspend and resume the jobs based on load conditions.

Furthermore, it allows to transparently run and use software that is not available on your local host and it help to manage the easily the access and the control of the resources such as:

- who can submit jobs and which hosts they can use
- how many jobs specific users or user groups can run simultaneously
- time windows during which each host can run load shared jobs
- load conditions under which specific hosts can accept jobs or have to suspend some jobs
- resource limits for jobs submitted to specific queues

5.1.2.3 Run a job

To LSF provides a resource management framework that takes your job requirements, finds the best resources to run the job, and monitors its progress, the jobs submitted behave just like jobs run on the local host. Even jobs with complicated terminal controls behave transparently to the user as if they were run locally and in according to host load and site policies.

Job in a LSF is a program or command that is submitted to a queue for scheduling and execution in an LSF cluster. A job can be complex problems, simulation scenarios, and extensive calculations, anything that needs compute power.

All LSF jobs run in queues, even interactive programs. A queue is associated with one or more servers, and has various limits defined, such as the number of jobs that can run on it at a given time and runtime limits for jobs that run on it.

A job can be submitted from an LSF client or server with the `bsub` command, in this case is used the default queue as shown below:

```
bsub my_job
Job <1234> is submitted to default queue <normal>
```

It is also possible to specify a different queue using instead the command `bsub -q name_queue`. Jobs are held in a queue waiting to be scheduled and have the PEND state. The job stands in the PEND state until all conditions for its execution are met, and all the resources that it needs are free. When the resources are free and the policies choose by the user are respected, the job is execute and enter in RUN state until the end. Normally, if not
to start, no other jobs can have access to the same resource. This allows to run a job without interruption given by absence of resources.

In MareNostrum are present different queues, is possible to see them and the relative facilities using the command ‘bsc_queues’.

Each queue has some information to help the work of scheduler and dispatcher, the attributes associated are:

- Priority
- Name
- Queue limits (restrictions on hosts, number of jobs, users, groups, or processors)
- Standard UNIX and Linux limits: memory, swap, process, CPU
- Scheduling policies
- Administrators
- Run conditions
- Load-sharing threshold conditions

When a job is submitted the LSF system uses built-in and configured resources to track resource availability and usage. Jobs are scheduled according to the resources available on individual hosts. To choose the best queue LSF considers the requirements of the job and automatically chooses a suitable queue from a list of candidate default queues according to:

- User access restriction
- Host restriction
- Queue status
- Job’s requested resources
- Exclusive execution restriction

Jobs submitted through the LSF system will have the resources they use monitored while they are running. This information is used to enforce resource limits and load thresholds as well as fair share scheduling.
LSF allows multiple scheduling policies in the same cluster based on the needs of each host. The common scheduling policies used also on MareNostrum are:\footnote{Please refer to glossary for the definition of the queues.}:

- First-come, first-served (FCFS) scheduling
- Service level agreement (SLA) scheduling
- Fairshare scheduling
- Preemption
- Backfill

### 5.2 Spark and MareNostrum

Spark in MareNostrum was implemented avoiding the use of a pre-implented cluster manager, such as YARN. In our case was created an own cluster manager in order to simplify and automate the use of Spark over the IBM LSF Platform used in the BSC supercomputer. To reach a complete control of the Spark in the HPC cluster was created a special framework named Spark4MN that also allows a cooperation with other different cluster manager, the version of Spark in the cluster each part of it was compiled to make compatible with MareNostrum 64 bits, the principal component are Hadoop 2.6.0 and Spark 1.3.0, furthermore is used Java 1.8.0 u40 that not needs to be re-compiled.

The framework is used to help a user to work in a cluster without knowing the real structure of the supercomputer, basically it is composed by different Bash scripts that provide a series of command to work with it. The implementation was done to make the framework modularized, so add a new distributed file systems (DFSs) will be easy.

Spark4MN gives to the user the three principals commands to the developers to work with their applications work with Big Data:

1. `spark4mn`
2. `spark4mn_benchmark`
3. `spark4mn_plot`

These three commands will be analyzed better in the section Error! Reference source not found.

Spark4MN take in charge to monitor the application before and during its execution, until the end, where are written a series of files to help the developers in the analysis of the results.

#### 5.2.1 Setup

Every time that an application is executed in the HPC cluster in MareNostrum is created a job that is managed by the framework. To run a job is necessary to create a dedicated configuration file in order to manage the resource needs by the application, so the user can
avoid to interact directly with Spark, indeed every action pass through spark4MN because it is designed to manage and work a supercomputer environment.

When a job is executed through Spark4MN, when the part of the cluster configuration is end, the job enter in the proper queue until an exclusive number of cluster’s nodes are available for it. The execution has to follow several step, the first is to understand if a DFS cluster has to be set up, this is reach collection all the information given by the host. The second step is to initialize the system, so Spark4MN tries to balance the configuration distributing the hardware resources homogenous as possible and deciding how to split the nodes between the DFS and Spark.

Calling M the number of nodes belonging to the DFS and N the number of nodes belonging to Spark and remembering that one node is always necessary for manage the job script submitted with Spark, four nodes distribution are possible (Figure 5-6):

- Exclusive mode
- Shared mode with M=N
- Shared mode with M<N
- Shared Mode with M>N

![Nodes distribution mode](image)
Finally, Spark4MN can distribute across the nodes previously configured the data and start the services required, also if a DFS is requested, its master service is executed and all the DFS worker services are launched and connected to the master while Spark4MN waits that the interconnection is ready. Now, the nodes configuration for DFS is finished and Spark4MN can execute the Spark configuration. This step is similar to DFS configuration, in fact the steps to follow are: start master, which corresponds to the standalone Spark manager, wait for master to be ready, and start workers, that are worker services.

Spark4MN waits until the cluster is completely set up and ready and start to executed sequentially the user’s application via spark-submit, each application is submitted with one call of spark-submit. To improve the performance Spark4MN allows to use the output of one application as input of the next application, so it creates an application pipeline that can help users running related applications.

When all the application submitted by the user are executed and completed, Spark4MN can turn off all the component associated with each job and create a gather all the metrics files in order the create one single metric file.

### 5.2.2 Commands

The principal command provided by the framework is spark4mn; it is used to deploy all the Big Data cluster’s services, and to execute all the user’s applications. To use the command is enough to pass as argument one or more configuration files, the configuration files are read through a cycle and for each configuration file read, a job is submitted into the cluster putting it into a proper queue, in our case in the bsc queue.

In addition to spark4mn command, the framework provides the command spark4mn_benchmark. This command can be used to run multiple jobs changing automatically the configuration file and monitoring during the execution each job. To allow the lunch of many jobs, spark4mn_benchmark is implement on the top of spark4mn, so when a user wants to run the same applications with different geometries, he has just to use one configuration file for spark4mn_benchmark that takes care to generated the correct configuration values and pass them to spark4mn. Like spark4mn, also spark4mn_benchmark offer to use more than one configuration file, this can be useful to run different applications, each one with different geometries. Moreover, spark4mn_benchmark provides at the end of all the jobs submitted a set of logs and metrics files that contain also important information about their application’s scalability. An interesting feature is the possible to set a ”restart” flag that can be used to complete a previous execution adding only the jobs with the missing geometries, this is reached looking at the already generated metric files and submitting the jobs missing.

The last command provided by Spark4MN is spark4mn_plot that collects all the metrics files generated using spark4mn_benchmark command, calculates statistics about the application performance and writes them within gnuplot-compliant data files.

### 5.2.3 Configuration file and logs
The configuration file, shown in Figure 5-7, is composed by different settable parameters, only few parameters are mandatory, almost of them are optional and are used to instruct Spark4MN about the geometries and optional component. The main parameter to lunch a job is the domain program, and it has to be set writing the path of the JAR file of the program, plus an entry point, that is the class of the Java/Scala program that the user wants to submit. Other parameters are used to modify the geometries of the job, it is possible to set the number of nodes, the number of workers per node and the number of cores per worker, each one of these values, if not set, have a default value calculate by Spark4MN. In case that sum of the resources request is more than the cluster’s hardware the resource, a warning is raised by Spark4MN.

It is possible also to change the network interface, this can be useful to test the behaviors of different cluster architectures, or to simulate an environment in which the DFS and Spark are in different networks.

Another set of parameters established the characteristics of CPU affinity that will be applied to the worker services in order to change the pinning modes. The four possible options are:

- Core → this mode pin one worker per core, so if in the configuration file is set the parameter "number of cores per worker" it is misleading. In order to decide how many threads or subprocesses they will be able spawn, Spark wants to know the number of CPUs. In fact is possible to use less physical cores than requested(#coresworker * #workersnode) if one chooses core pinning, because each worker has all its threads and subprocesses inside one single core.

```sh
### JOB PARAMETERS: ###
JOB_NAME="[STRING]"
WORKING_DIR="[PATH]"
WALLCLOCK=[INT] # Minutes.
NEAR_NODES="[BOOL]"
DOMAIN_JAR_1[1-9]+="[PATH]"
DOMAIN_ENTRY_POINT_1[1-9]+="[STRING]"
[...]

### SPARK_PARAMETERS: ###
# SPARK_VERSION="[STRING]"
# SPARK_NNODES=[INT]
# SPARK_NWORKERS_PER_NODE=[INT]
# SPARK_NCORES_PER_WORKER=[INT]
# SPARK_WORKER_MEM_SIZE=[INT] # MiB.
# SPARK_WORKER_AFFINITY="[CORE | SOCKET | NODE | NONE]"
# SPARK_NETWORK_INTERFACE="[IB | ETH]"

### DFS PARAMETERS: ###
# DFS_MODULE="[HDFS | NONE]"
# DFS_SHARE_NODES="[BOOL]"
# DFS_NNODES=[INT]
[...]
```

*Figure 5-7. Example of a job configuration file for spark4mn command*
• **Socket** → it assigns each worker to a socket, and the requested number of cores belongs to it;
• **Node** → the pinning is fixed, but a worker can have CPUs from more than one socket. The set of CPUs assigned to each worker, no matter the pinning mode, is calculated taking into account the workload balance; pinning to each socket, and core the same amount of workers (± 1);
• **None binding** → it is the kernel’s default scheduling.

In the configuration file of spark4mn_benchmark are present all the parameters of the configuration file of spark4mn, but for many of them is possible to set a range of values that the user wants to use for his applications.

As shown in the Figure 5-8 to set a range is enough to choose the start and end values, the operator and the step used to calculate the values inside the range. Setting these values benchmark program takes care to call a spark4mn program one per each value of the list of numbers calculated.

```
# SPARK_START_NNODES=[INT]
# SPARK_END_NNODES=[INT]
# SPARK_NNODES_STEP_OPERATOR="[+ | *]"
# SPARK_NNODES_STEP=[INT]
```

*Figure 5-8. Example for setting a range of value in spark4nn_benchmark configuration file*

The number of jobs submitted can increase dramatically if are set different parameters, in fact if we decide to create one range for the number of nodes (e.g. [16 32 64 128 256]), and for the number of worker (e.g. [1 2 4 8 16]), the program will submit all the possible combination of the values in the ranges (in this case will have 25 jobs).

When a job ends, a set of logs files will be created; in case that spark4mn_benchmark is used will be generated one set of logs files per each job submitted. The principals files generated are:

• **Metrics file** → in this file are collected all the information about the time that the master and all workers related spent for the job;
• **Domain file and workers files** → here are saved the output (info, warnings and user output, errors) of the program generated by the master and the workers;
• **Error file** → in this file is stored the errors outputs generated by spark4mn program, so is possible to see if an error occur in the cluster during the set up or the execution of the program. In this file are not present error generated by the user application (e.g. errors due to logic of the program);
• **Out file** → here are stored different information about the job. The principals information are: a portion of code used to submit the job in the LSF (using *bsub* command), submission information (id_job, name_job, user, queue, directories), list
of worker used by the job and a summary of resource usage (CPU time, max memory, average memory, total requested memory, delta memory, max processes, max threads).
6. EVALUATION

The late part of the development has seen a validation and performance testing phase of the model and the implementation in the MareNostrum.

The first part will consist in the models quality evaluation, in this part will be explained the different algorithms used to estimate the correctness of the models and error. Moreover, will be shown the influence of different parameters configurable for each model.

Subsequently is analyzed how Spark work in a real cluster, in this case in MareNostrum; to test phase in this case are done on the scalability of Spark, changing the number of nodes used.

6.1 MODELS EVALUATION

6.1.1 Methodology

The experiments are executed in a local machine, precisely in a virtual machine using Oracle VirtualBox. The operating system used is Xubuntu 14.04 LTS (64bit), for the VM is reserved a RAM of 4096 MB and 50GB of hard disk.

To reach acceptable results the final values are obtained calculating the average of 10 experiments. The experiment is done splitting the dataset in two parts, one for the training of the model, and the other one for the testing. The split is done randomly using a floating number as seed at run time. For document classification, the input feature vectors are usually sparse, and sparse vectors should be supplied as input to take advantage of sparsity.

6.1.1.1 Parameters

For each model can be considerate different parameters, that can be changed in order to obtain a better estimator. MLlib use an optimizer object to changing the values of the parameters, so each model has different values that can be used to improve the estimator. The binary classifiers, SVM and Logistic Regression, has an optimizer that allows us to change the following parameters:

- Number of iteration → is the maximum number of iteration that the algorithm has to do
- Regularization parameter → is fixed regularization parameter \( \lambda \geq 0 \) and settable with regParam method. It is used to defines the trade-off between the two goals of minimizing the loss and minimizing model complexity
Updater → this regularization methods is to prevent overfitting by penalizing models with extreme parameter values. In this work are used three common variants: zero, L₁ and L₂ regularization.

Instead for Naïve Bayes is possible to modify only the parameter \( \lambda \), that is the ‘additive smoothing’, a technique used to smooth categorical data, it is used to incorporate a small-sample correction, in all probability estimates such that no probability is ever set to be exactly zero.

6.1.1.2 Metrics

The three principal models are evaluated with three metrics, the first one is the Mean Squared Error, after, the second one is the Accuracy and finally is calculated the confusion matrix. These three metrics are calculate for each model and for each parameters changed in the model.

6.1.1.2.1 Mean Squared Error

The mean squared error in static is one of the most important criterion used to evaluate the performance of a model predictor or an estimator. It is useful to relay the concepts of bias and precision in statistical estimation. In statistic the mean squared error is frequently used to assess the risk of an estimator, that is, how large are on average the losses generated by the estimation errors committed when employing the estimator in question, in practice it measures the average of the squares of the "errors", that is, the difference between the estimator and what is estimated.

Let \( \hat{X} \) be a vector of n predictions and \( X \) the vector of true values, the formula to obtain the MSE is:

\[
\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{X}_i)^2
\]

There is also another possible mathematical definition for the MSE that uses the concepts of targets and estimator, but is not relevant for our scope.

As we can see, the MSE could be seen like the average of “distance” between the real value and the prediction of the value. Considering that in our case each point is represented by a vector of LabeledPoint (store in a RDD) the MSE is calculated the label of each vector, so if the value is correct estimated the ‘distance’ is 0 (label 0.0 prediction 0.0 and label 1.0 prediction 1.0), instead if the estimation is wrong, the ‘distance’ is equal to 1 (label 0.0 prediction 1.0 and label 1.0 prediction 0.0). Doing the average of these distance we can obtain our MSE with LabeledPoint vectors.

---

12 The regularizer values used in MLlib are in [https://spark.apache.org/docs/1.2.0/mllib-linear-methods.html#regularizers](https://spark.apache.org/docs/1.2.0/mllib-linear-methods.html#regularizers)
6.1.1.2.2 Accuracy

Accuracy is a simple measure of how close measurements are to the "correct" value, and is a stronger statement than precision, as it includes both random and systematic errors. To assess accuracy the true result must already be known. Error is the difference between a measurement and the true value of the measuring. The error not take into account the mistakes.

Also for calculate the accuracy are used the vectors of LabeledPoint, in this case are counted the number of correct prediction, where the label predicted is equal to the real label, and it is divided by the number of total prediction done, the same result can be also obtained through the formula:

\[
Accuracy \approx 1 - MSE
\]

This measure can be extrapolate from the confusion matrix also, in fact the formula of accuracy is:

\[
\text{accuracy} = \frac{n^\circ \text{true positives} + n^\circ \text{true negatives}}{\text{total } n^\circ \text{of points/predictions}}
\]

6.1.1.2.3 Confusion matrix

A confusion matrix, also known as a table of confusion in predictive analysis, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. It contains information about actual and predicted classifications done by a classification system.

The confusion matrix shows how the predictions are made by the model. The rows correspond to the known class of the data, i.e. the labels in the data. The columns correspond to the predictions made by the model. The value of each of element in the matrix is the number of predictions made with the class corresponding to the column for examples with the correct value as represented by the row. Thus, the diagonal elements show the number of correct classifications made for each class, and the off-diagonal elements show the errors made.

An example of confusion matrix is shown below:

<table>
<thead>
<tr>
<th></th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Negative</td>
<td>Positive</td>
</tr>
<tr>
<td>Real</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>Positive</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>
Each entire in the confusion matrix has the following meaning:

- \(a\) is the number of correct predictions that an instance is negative,
- \(b\) is the number of incorrect predictions that an instance is positive,
- \(c\) is the number of incorrect predictions that an instance negative, and
- \(d\) is the number of correct predictions that an instance is positive.

As said before from the confusion matrix can be extract the accuracy, but also can be used to calculate the Precision, proportion of the predicted positive cases that were correct, and the Recall or true positive rate, that is a measure of the ability of a prediction model to select instances of a certain class from a data set, the formulas are:

\[
P = \frac{d}{b + d} \quad RC = \frac{d}{c + d}
\]

These measures will not be used because they are not of real interest in our case.

In addition, from the confusion matrix is possible to create a ROC graph that is a plot with the false positive rate on the X axis and the true positive rate on the Y axis. The two axis have both the range \([0-1]\). The two most meaningful points are (0,1) and (1,0), respectively the classifier that is incorrect for all classifications and a perfect classifier.

### 6.1.2 Results

In this section will be examined the result of each model, basing on the metrics and parameters explained before. The test for each model was executed 10 times using the second data sample, and the average values was taken to construct a summary table. An example is shown in Table 6-1, as we can see, the table contains all the possible combination of the parameters used, and the average result of the metric, in this case MSE.

<table>
<thead>
<tr>
<th>Updater</th>
<th>L2</th>
<th>L1</th>
<th>Simple</th>
</tr>
</thead>
<tbody>
<tr>
<td>N* Iteration</td>
<td>50</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>0.1</td>
<td>0,1635</td>
<td>0,1571</td>
<td>0,1577</td>
</tr>
<tr>
<td>0.5</td>
<td>0,1728</td>
<td>0,1838</td>
<td>0,1670</td>
</tr>
<tr>
<td>1</td>
<td>0,1983</td>
<td>0,1797</td>
<td>0,1780</td>
</tr>
<tr>
<td>2.5</td>
<td>0,1861</td>
<td>0,1820</td>
<td>0,1832</td>
</tr>
<tr>
<td>5</td>
<td>0,1936</td>
<td>0,2278</td>
<td>0,1959</td>
</tr>
<tr>
<td>10</td>
<td>0,8174</td>
<td>0,7258</td>
<td>0,1948</td>
</tr>
<tr>
<td>40</td>
<td>0,8174</td>
<td>0,8174</td>
<td>0,8174</td>
</tr>
<tr>
<td>80</td>
<td>0,8174</td>
<td>0,8174</td>
<td>0,8174</td>
</tr>
<tr>
<td>100</td>
<td>0,8174</td>
<td>0,8174</td>
<td>0,8174</td>
</tr>
</tbody>
</table>

*Table 6-1. Summary table of MSE of the Logistic Regression tests*

The summary tables were created 3 for each model, the three table are related to the MSE, Accuracy and ROC value. Using these tables, we can extract some information about the
quality of the models and which one is the best to predict if a user of Instagram can belong to a group of followers or not. The Figure 6-1 shows the histograms of the models accuracy, comparing for each model the methods used to retrieval the information.
Figure 6-1. Histograms of the models accuracy
The Figure 6-2 shows a histogram used to compare the accuracy of the models, the accuracy is the same of the previous charts, so the method of information retrieval use can be seen in the previous graphs.

![Comparison Different Classifiers](image)

*Figure 6-2. Comparison of models accuracy*

For completeness in the Figure 6-3 are shown the confusion matrices related to the three best estimator, one for each algorithm. The confusion matrices are calculated testing the model on a training set obtained with a random split from the data sample. The matrix present in the columns the values predicted, and in the rows the real values.

\[
\begin{pmatrix}
1384 & 223 \\
225 & 193
\end{pmatrix}
\begin{pmatrix}
1967 & 143 \\
249 & 223
\end{pmatrix}
\begin{pmatrix}
1583 & 186 \\
351 & 204
\end{pmatrix}
\]

*Figure 6-3. Confusion matrices of Logistic Regression, Support Vector Machine and Naive Bayes*

### 6.2 Spark on MareNostrum

#### 6.2.1 Methodology

We have selected three representative benchmarks including sort-by-key, K-means, and Naive Bayes, which was inspired by the composition of HiBench (the Hadoop Benchmark Suite). The three selected workloads are typical Spark workloads that are also part of HiBench. The operation used to test Spark on MareNostrum are:

- Sort-by-key → is a critical operation used by many applications and it helps revealing the pattern of shuffle operations. It consists of three stages.
  - The first is computation stage, where each task generates a set of key-value pairs in memory.
  - The next stage, Spark schedules ShuffleMapTasks to partition the intermediate data and store
them into the file systems. In the last stage, fetching tasks shuffle intermediate data over the
network. Across such data processing pipeline, the intermediate data size is equal to the input
size.

- K-means → is a clustering algorithm for knowledge discovery and data mining. Spark’s
  MLlib includes a parallelized variant of the k-means++ method called K-means++.
- Naive-Bayes → we already explained this algorithm.

To obtain acceptable result we have run the experiment 5 times, so each test was executed 5
times, and was calculate the average value and discarding outliers with high deviation. In
total we submitted more than 1 million jobs to MareNostrum, the results are described in the
following section.

6.2.1.1 Parameters

There are many different parameters in MLlib that can be tweaked in both the Spark
framework and within the selected benchmarks.

Starting with k-means we have run different configurations changing the parameters: num-
partitions, num-examples, num-features, nb-lambda and per-negative.
Sort-by-key was run changing the values of: num-partitions, reduce-tasks, num-records,
unique-keys, key-length, uniquevalues, value-length and persistent-type.
Also for Naive-Bayes are changed the values of num-partitions, but optional smoothing
parameter was left with the default values (1.0).

Initially we performed tests with varying values for all the parameters. In order to minimize
the number of possible combinations, we defined default values for parameters showing
independent and irrelevant behaviors. Instead, for some parameters showing an independent
but relevant behavior we defined isolated experiments and used default values for the rest of
the tests, is the case of network interface that was used InfiniBand for all the general tests.

6.2.2 Results

We performed experiments with sequences of increasing input data sizes from $10^6$ to $10^7$
data points. These input data sizes were not really an issue for runs on GPFS, other than to
inflated the benchmark run times, but large data sizes were an issue when running in node
local memory.
The input data sizes ranged anywhere from 500 MB to 250GB, depending on the benchmark. We did scaling tests for each of the benchmarks, going from 8 nodes to 128 nodes. The Figure 6-4 and Figure 6-5 show the performance of the three benchmarks with different input data sizes and different amount of nodes.

**Figure 6-4.** Performance of Naive Bayes using 4 workers per node and different number of nodes

**Figure 6-5.** Performance of K-means using 4 workers per node and different number of nodes
Figure 6-6 shows the performance of Naive Bayes with a fix number of nodes and with different number of worker per nodes.

Each MareNostrum node has 16 cores, we performed experiments with different distribution of the workers within each node. The Error! Reference source not found., Figure 6-8 and Figure 6-9 show some possible distributions and related performance.

The last analysis done is on the influence of the number of partitions and the correlation with the number of workers per node, result of the performance are collected in a histogram shows in the Figure 6-10.
Figure 6-8. Performance with 4 cores per worker

Figure 6-9. Performance with 8 cores per worker
Figure 6-10. Histogram of performance using different partitions and number of workers per node.
7. CONCLUSION AND FUTURE WORK

The scope of this work three algorithms were implemented using Spark frameworks in order to create a classifier to predict if a user can belong to a certain group of followers, so the main goal of this thesis was to show that Spark framework is suitable for classification algorithms. The goal was fulfilled by implementing three different algorithms and showing quality of the classifiers.

The three algorithms implemented were: Support Vector Machine, Logistic Regression and Naïve Bayes. Algorithms were tested using three different method to convert data into a RDDs containing the same data in a form usable by MLlib algorithms. The quality taken by each algorithm with different parameters was calculated with three metrics: MSE, accuracy, confusion matrix. The number of initials data tags are more than 1.800.000 divide in two data sample, negative and positive. For the first experiment each algorithm was tested with all the possible combination of the parameters settable. In the binary classifier the parameters taken into account are the number of iteration that the algorithm has to do, the regularization parameter and the regularization method, instead remembering that Naïve Bayes is a multiclass classifier, in this case is possible to set only the value of λ (additive smoothing).

The results show that the best performance are reached with the binary classifiers SVM and Logistic Regression, instead the Naïve Bayes has worst performance probably due to its simplicity and the inability to change many parameters to improve the model. In the case of the two binary classifier the best results are obtained with the combination of the simple frequency counter, low value of regularization parameter and $L_2$ as regularization method. TF-IDF can be worse than the counting frequency because introduce a correction factor in the calculus of the frequency, and in tags case the correction value can hide due importance to relevant tags, only for the massive use of them. Usually $L_1$ regularization is preferred performs feature selection within the learning algorithm producing sparse models, but in this case $L_2$ works better because $L_1$ can be not differentiable for these algorithms. Finally, the regularization parameter has not theoretical value, but it is can be found trough experiments, in our case we discovered that for algorithms a RP lower than 0.5 improve the overall quality of the models. Regarding the number or iteration we can said that each algorithms with its parameters has unknowable value a priori, but each algorithm has to be tested with different number of iteration values in order to create a good classifier.

All the three models created can be used to have a good prediction of a user, also Naïve Bayes, that has the worst performance, can predict if a user can belong to a followers group with less of 30% of error.

The second scope we reached in this thesis, to explore the feasibility and convenience of deploying a state-of-the-art instantiation of the emerging Apache Spark over a real-world, petascale, HPC setup. The deployed Apache Spark 1.3.0 was successfully done the MareNostrum supercomputer. To work with MareNostrum, we have designed and developed a Spark4MN framework in order to automate the usage of Spark over an IBM LSF-based environment and the hardware particularities of MareNostrum. Spark4MN has proven to be convenient tool for running Spark-based applications on an HPC setup and is being currently used at BSC for executing Big Data workloads. The framework is also
extremely useful for benchmarking the Spark deployment, providing tools to run multiple instance of the same application with different configuration for each job.

In order to test Spark4MN framework a data input were generated in a range with sizes from 100,000 to 400,000. Different tests were done changing the number of nodes, the number of workers per node and, the number of cores per worker and at the end also the number of partitions.

Three algorithms were used to test Spark: sort-by-key, K-means and Naïve Bayes. Beginning with the difference of the algorithms, we can see that the faster is Naïve Bayes, in fact as we said before it also is the simplest of the three algorithms. It is used to the performance tests of Spark in order to obtain the time spent to run the algorithm with different cluster configurations. The first parameter changed is the number of nodes used in the cluster, from 4 to 32. We can see that in this case that increase the participants node Spark need less computation time, this is visible mostly increasing the data size, bigger is the data size, and better is the scalability. Instead changing only the number of worker per node we can see that in initially the performance not improving, but changing also the number of total nodes used in the cluster, the overall performance increase and the computation time considerably decrease. The others tests are centered to find this equilibrium point exploring different distribution of the cores. All the results shows that in the all the possible distribution, the time spent is the same, so the distribution of the cores not affect the computation time.

Finally, we tested the influence of the number of partitions, this parameter can be changed trough the configuration file or also during running time. To explore the relevance of this parameters were taken into account different combination changing the number of workers per node. Results show that the number of partitions is crucial to improve the performance, we can see a big difference between the use of 100 and 1,000 partitions, and in fact this parameter said how to cut the dataset in the cluster to create a parallel collections. RDD is the key of Spark, and is based of concept of distributing and parallelize the work, knowing that Spark will run one task for each partition of the cluster, and that each RDD is divided in different partitions, we can understand how this number can strongly influence the performance.

7.1 **Future Work**

To improve the classifiers we are working on a better pre-process algorithm in order to normalize the data and take off the possible outliers’ users or tags. The pre-processing consist in the find a possible correlation among the tags, so to fold a group of tags in only one word. Another improvement in working is to use OpenCV framework (available in Scala and Java) to manipulate the images, in order to extract some important information, such as writes, brands, faces and objects. These information can be added to the data sample in form of tags, in this way all the pictures can be used to enrich the details of each user and to create a more fitting profile.

The project is MareNostrum is still on going, the test using the cluster are trying to figure out the impact of storage (GPFS vs. HDFS) and the impact of network. A further feature of Spark4MN will be the extension of the framework to work with other workload managers,
with the addition of new job headers that state all the job’s configuration. The addition of other distributed file systems like Apache HBase, or Apache Cassandra is also possible, because each DFS is packed as a code module. We also plan to port Spark4MN to a compiled language (C++) to improve its efficiency. Using Spark4MN, MareNostrum, OpenCV and the classifiers will be possible to do create a stream application cl of the meta-data and pictures and so obtaining a real-time classifiers to analyze the big data generated by Instagram traffic.
REFERENCES


