CHAPTER 1: INTRODUCTION

In the telecommunication industry, customers are able to choose among multiple service providers and actively exercise their rights of switching from one service provider to another. In this fiercely competitive market, customers demand tailored products and better services at less prices, while service providers constantly focus on acquisitions as their business goals. Given the fact that the telecommunications industry experiences an average of 30-35 percent annual churn rate and it costs 5-10 times more to recruit a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition. For many incumbent operators, retaining high profitable customers is the number one business pain. Many telecommunications companies deploy retention strategies in synchronizing programs and processes to keep customers longer by providing them with tailored products and services. With retention strategies in place, many companies start to include churn reduction as one of their business goalsWe are awash in a flood of data today. In a broad range of application areas, data is being collected at unprecedented scale. Decisions that previously were based on guesswork, or on painstakingly constructed models of reality, can now be made based on the data itself. Such larger data analysis now drives nearly every aspect of our modern society, including mobile services, retail, manufacturing, financial services, life sciences, and physical sciences.

1.1CHURN

The subject of customer retention, loyalty, and churn is receiving attention in many industries. This is important in the customer lifetime value context. A company will have a sense of how much is really being lost because of the customer churn and the scale of the efforts that would be appropriate for retention campaign. The mass marketing approach cannot succeed in the diversity of consumer business today. Customer value analysis along with customer churn predictions will help marketing programs target more specific groups of customers.

Personal retail banking sector is characterized by customers who stays with a company very long time. Customers usually give their financial business to one company and they wont' switch the provider of their financial help very often. In the companys' perspective this produces a stabile environment for the customer relationship management. Although the continuous relationships with the customers the potential loss of revenue because of customer churn in this case can be huge.

People and devices are constantly generating data. While streaming a video, playing the latest game with friends, or making in-app purchases, user activity generates data about their needs and preferences, as well as the quality of their experiences. Even when they put their devices in their pockets, the network is generating location and other data that keeps services running and ready to use.

1.2 Big Data

Big Data is data that is too large, complex and dynamic for any conventional data tools to capture, store, manage and analyze.

The right use of Big Data allows analysts to spot trends and gives niche insights that help create value and innovation much faster than conventional methods.

Majorly there are three types of data:

- Structured (any relational DB that has some schema)
- Semi-structured (xml file, csv file)
- Unstructured (email,doc, post on social media,pdf)

Big Data can be better define by 3 V's , i.e, Volume , Variety , Velocity and Verasity.

- <u>Volume:</u> Scale of data
- <u>Variety:</u> Different forms of data
- <u>Velocity:</u> Analysis of streaming data

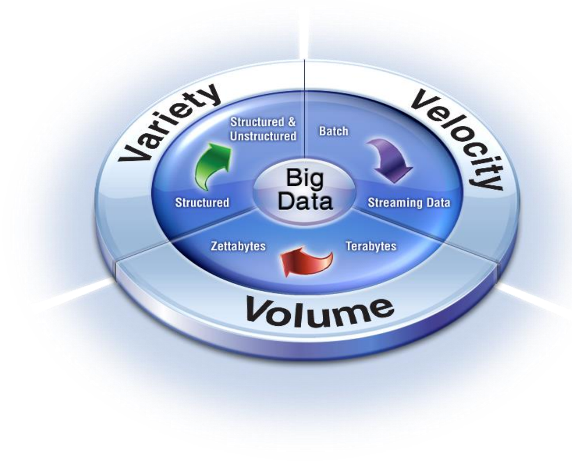


Figure 1.1 Types of data

1.3 Big Data Problem

The big data problem means that data is growing faster than computation speeds.We have many different sources of data and those sources of data are growing, web, mobile,scientific.

Storage is getting cheaper.This means people are saving more data and the size of storage is doubling every 18 months.So we can save even more data. But CPUs are not increasing in speed and we have many storage bottlenecks getting data in and out of this massive storage.

Some examples of big data include:

- Facebook's daily logs.They're 60 terabytes in size.Every day, they collect 60 terabytes worth of data.The 1,000 genomes project has 200 terabytes of human genome data.
- Google's web index is estimated to be largerthan 10 petabytes of data.

Now, driving all of these, the fact is that storage has really dropped in cost. But it can take three hours to read the data from the disk or write it to the disk.

The big data problem means that a single machine can no longer process or even hold all of the data that we want to analyze.

1.4 Big Data – Where is it ?

Big data surrounds us, although we may not immediately realize it. Part of the problem is that, except in unusual circumstances, most of us don't deal with large amounts of data in our everyday lives. Lacking this immediate experience, we often fail to understand both opportunities as well challenges presented by big data. Because of these characteristics, there are currently a number of issues and challenges in addressing these characteristics going forward.

1.5 Issues

We suggest there are three fundamental issue areas that need to be addressed in dealing with big data: storage issues, management issues, and processing issues. Each of these represents a large set of technical research problems in its own right.

- **Storage and Transport Issues**

The quantity of data has exploded each time we have invented a new storage medium. What is different about the most recent explosion – due largely to social media – is that there has been no new storage medium. Moreover, data is being created by everyone and everything (e.g., devices, etc) – not just, as heretofore, by professionals such as scientist, journalists, writers, etc.

Current disk technology limits are about 4 terabytes per disk. So, 1 exabyte would require 25,000 disks. Even if an exabyte of data could be processed on a single computer system, it would be unable to directly attach the requisite number of disks. Access to that data would overwhelm current communication networks. Assuming that a 1 gigabyte per second network has an effective sustainable transfer rate of 80%, the sustainable bandwidth is about 100 megabytes. Thus, transferring an exabyte would take about 2800 hours, if we assume that a sustained transfer could be maintained. It would take longer to transmit the data from a collection or storage point to a processing point than it would to actually process it.Two solutions manifest themselves. First, process the data "in place" and transmit only the resulting information. In other words, "bring the code to the data", vs. the traditional method of "bring the data to the code." Second, perform triage on the data and transmit only that data which is critical to downstream analysis. In either case, integrity and provenance metadata should be transmitted along with the actual data.

- **Management Issues**

Management will, perhaps, be the most difficult problem to address with big data. This problem first surfaced a decade ago in the UK eScience initiatives where data was distributed geographically and “owned” and “managed” by multiple entities. Resolving issues of access, metadata, utilization, updating, governance, and reference (in publications) have proven to be major stumbling blocks.Unlike the collection of data by manual methods, where rigorous protocols are often followed in order to ensure accuracy and validity, digital data collection is much more relaxed.

The richness of digital data representation prohibits a bespoke methodology for data collection. Data qualification often focuses more on missing data or outliers than trying to validate every item. Data is often very fine-grained such as click stream or metering data. Given the volume, it is impractical to validate every data item: new approaches to data qualification and validation are needed.

The sources of this data are varied - both temporally and spatially, by format, and by method of collection. Individuals contribute digital data in mediums comfortable to them: documents, drawings, pictures, sound and video recordings, models, software behaviors, user interface designs, etc – with or without adequate metadata describing what, when, where, who, why and how it was collected and its provenance.

Yet, all this data is readily available for inspection and analysis.Going forward, data and information provenance will become a critical issue. JASON has noted that “there is no universally accepted way to store raw data, reduced data, and the code and parameter choices that produced the data.” Further, they note: “We are unaware of any robust, open source, platform independent solution to this problem.” As far as we know, this remains true today. To summarize, there is no perfect big data management solution yet. This represents an important gap in the research literature on big data that needs to be filled.

- **Processing Issues**

Assume that an exabyte of data needs to be processed in its entirety. For simplicity, assume the data is chunked into blocks of 8 words, so 1 exabyte =1K petabytes. Assuming a processor expends 100 instructions on one block at 5 gigahertz, the time required for end-to-end processing would be 20 nanoseconds. To process 1K petabytes would require a total end-to-end processing time of roughly 635 years. Thus, effective processing of exabytes of data will require extensive parallel processing and new analytics algorithms in order to provide timely and actionable information.

CHAPTER 2: SOFTWARE REQUIREMENT SPECIFICATION

A Software Requirements Specification (SRS) - a requirements specification for a software system - is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation such as performance engineering requirements, quality standards, or design constraints.

2.1. Introduction

a. Purpose:

The main objective of this project is to do customer churn analysis of telecommunication company data set and design a model which will check it in future cases.In this project will do comparative study of decision tree and naïve bayes algorithm on data set which is performed using scala language on spark set up upon cloudera manager 5.

b. Scope:

With this analysis we will able to check which algorithm will be better for designing a model and checking accuracy so as to provide better throughput. Further scala language provide less overhead and spark provides high working efficieny.

c. Definitions:

Data:

According to English dictionary data is representation of facts or ideas in a formalized manner capable of being communicated or manipulated by some process.

Internet:

A global network made up of large number of individual networks connected through the use of public telephone lines and TCP/IP Protocols is called INTERNET.

Distributed Processing:

A technology which helps in computing the data which is stored at different places without clubbing them together.

Client:

Any entity using the services is called a CLIENT.

Server:

The server providing the services to the client is called a SERVER.

Cluster:

A group of computer connected together using any protocol is called as CLUSTER.

Churn:

In the telecommunications industry, the broad definition of churn is the action that a customer's telecommunications service is canceled. This includes both service-provider initiated churn and customer initiated churn. An example of service-provider initiated churn is a customer's account being closed because of payment default. Customer initiated churn is more complicated and the reasons behind vary. In this study, only customer initiated churn is considered and it is defined by a series of cancel reason codes. Examples of reason codes are: unacceptable call quality, more favorable competitor's pricing plan, misinformation given by sales, customer expectation not met, billing problem, moving, change in business, and so on.

High-Value Customers: Only customers who have received at least three monthly bills are considered in the study. High-value customers are these with monthly average revenue of $X or more for the last three months. If a customer's first invoice covers less than 30 days of

SUGI 27 Data Mining Techniques service, then the customer monthly revenue is prorated to a full month's revenue.

Granularity: This study examines customer churn at the account level.

Exclusions: This study does not distinguish international customers from domestic customers. However it is desirable to investigate international customer churn separately from domestic customer churn in the future. Also, this study does not include employee accounts, since churn for employee accounts is not of a problem or an interest for the company.

d. Abbreviations and Acronyms:

HDFS – Hadoop Distributed File System

MR – MapReduce

TCP/IP – Transmission Control Protocol/Internet Protocol

2.2. Requirement Analysis

- **Existing System:**
  - Every system possesses some issues with it, so is Big Data. There are various different problems with Big Data sometimes which makes it disastrous.
  - **Heterogeneity***:* We can analyze that there is lots of heterogeneous data present in the server. Processing of such data is a major problem as we have to convert that data in homogeneous form which is not an easy task.
  - **Scale***:* Since data volume is scaling faster than compute resources, and CPU speeds are constant. So, processing of this exponentially growing data is a big issue.
  - **Timeliness***:* The larger the data set to be processed, the longer it will take to analyze. The design of a system that effectively deals with size is likely also to result in a system that can process a given size of data set faster.
  - **Privacy***:* Lots of confidential data is present there on server, there is great public fear regarding the inappropriate use of such data, particularly through gathering and linking such data from different sources.

2.3.Functional requirements :

As an end user following requirements are there:

- o Manual: It provides a complete basic detail of hadoop and its usage. It provides a guide for how to use each tool of the application.
- o Frameworks: It saves time of user by directly providing connectivity to their own database with hadoop .
- o Lookups: Several lookups are there which helps user to find details of big data terms, protocols etc.

- Search functionality: It ease the task of finding a desired result from the whole cluster. Thus saves user time and resources.

2.4 Hardware and Software requirements:

The selection of hardware is very important in the existence and proper working of any software. In the selection of hardware, the size and the capacity requirements are also important.

- <u>For development purpose:</u>

  Hardware: Intel or AMD based Computer or Mac OS X Leopard 10.5.8 late. Software: Red Hat version 6.0 or later, VM Ware version 10 or later, python version 2 or 3, Hadoop version 2.0, PIG, HIVE, HBASE and JDBC driver.

- <u>For end users:</u>

  Hardware: AMD based Computer or Mac OS X Leopard 10.5.8 later

  Software: Red Hat version 6.0 or later

- **Design constraints:**
  - Intel or AMD based Computer or Mac OS X Leopard 10.5.8 later
  - 24x7 availability
  - Better component design to get better performance at peak time
  - The database used here is robust, reliable & fast. So users have to wait for the output very short time.
  - This application can be accessed from any type of platform
  - There is no case of redundancy in the database so it will take no extra memory space.

CHAPTER 3: RELATED WORK

In this CHAPTER, we shall discuss the similar existing systems which are currently available over the internet. The relative study of existing and proposed system shows the comparison between both present and future technologies. This chapter concludes the pros and cons of existing systems and what are the comparative points relative to our proposed system.

3.1 Symbolic Techniques

Symbolic techniques in supervised classification models make use of available lexical resources. In this data analysis Turney (2002) used bag-of-words approach. In this approach the document was treated as a collection of words where relationships between words are not considered important. To determine the overall sentiment, sentiments of every word are given a value and using aggregation functions, those values are combined. Where tuples are phrases having adjectives or adverbs which may be considered positive or negative, Turney (2002) found the polarity of a review was based on the average semantic orientation of tuples extracted from the review. WordNet which is a database consists of words and their relative synonyms were used by Kamps et al. (2004). in this study a distance metric was developed on WordNet and the semantic orientation of adjectives was determined from this metric. In their study, Balahur et al. (2012) introduced a conceptual representation of text, which stored the structure and the semantics of real events, in a system called EmotiNet,. EmotiNet was able to identify the emotional responses triggered by actions with the information it stored. The difficulty with using a Knowledge base approach however that is it requires of a large lexical database. This has become harder and harder to provide as the language of social networks is so trend dependent and changeable that lexicon datasets cannot keep up. Therefore Knowledge based approaches to data analysis are not as popular as they used to be.

3.2 Machine Learning Techniques

In contrast to Knowledge based approaches Machine Learning techniques are not depen dent on a lexicon dataset, instead the use of a training set and a test set in order to classify is employed. This allows the algorithm to remain dynamic in the face of ever changing social network language lexicons. In this methodology a classification model is developed using a training set, which tries to classify the input feature vectors into corresponding class labels. A test set is used to prove the model by predicting the class labels of unseen feature vectors as outlined in the training set. A number of machine learning techniques like Naive Bayes (NB) and Support Vector Machines (SVM) are used to classify reviews into either positive or negative orientation. Vinodhini, Chandrasekaran (2012). In their paper Domingosetal. (1997) found that Naïve Bayes works as a good classifier for certain problems as it results in highly dependent features. A new model which was based on Bayesian algorithm was introduced in 2012 by Zhen Niu et al. (2012). In this model weights of the classifier are adjusted by making use of representative feature(information that represents a class) and unique feature(information that helps in distinguishing classes). Using those weights, the researchers calculated the probability of each classification. This allowed for an improved Bayesian algorithm. Pak and Paroubek (2010) created a twitter corpus by using a Twitter API which automatically collected tweets from Twitter as well as annotating those using emoticons. Using that corpus, they built a sentiment classifier which used N-gram and POS-tags as features based on the multinomial Naive Bayes classifier. In combining various feature sets and classification techniques an Ensemble framework is created. Xia et al. (2011) used an ensemble framework for sentiment classification in their paper were two types of feature sets and three base classifiers were used to form the ensemble framework. Part-of-speech information and Word-relations were used to create two types of feature sets. .Three base classifiers Naive Bayes, Maximum Entropy and Support Vector Machines were also selected. Fixed combination, weighted combination and Meta-classifier combination ensemble methods for sentiment classification were applied and measured so as to obtain better accuracy . When the classifiers were measured separately Naive Bayes was found to be the most accurate.

In this report, we construct a Twitter corpus using Twitter API, use R studio coding to prepossess the Twitter corpus, then using know ledge based methods we use an available lexical resources and apply it to the Twitter corpus. To compare the results from the knowledge based method to a machine learning technique we then use Breen’s classification models to the corpus which will split the corpus into positive and negative tweets as well as highlighting which tweets are classified. Breen’s Algorithm is used as it is often works well as a good first classifier in data analysis.

CHAPTER 4: PROPOSED WORK

In this CHAPTER, we shall discuss the algorithm which we have used to generate categorize the tweets.

4.1 Project Plan

Effective management of a software project depends on thoroughly planning the progress of the project. The project manager must anticipate problems which might arise and prepare tentative solution to those problems. A plan, drawn up at the start of the project should be used as driver for the project. This initial plan should be the best possible plan given the available information. It evolves as the project progresses and better information become available.

A planning is an iterative process which is only complete when the project itself is complete. As the project information become available during the project, the plan must be regularly revised. The overall goals of the business are an important factor which must be considered when formulating the project plan. As these change, changes to the project plan are necessary.

The planning process starts with an assessment of the constraints affecting the project. This is carried out in a conjunction with estimation of project parameters such as structure, size and distribution of functions. The progress milestones and deliverables are then defined. The process then enters the loop. A schedule for the project is drawn up and the activities defined in the schedule are initiated or given permission to continue. After sometime the progress is viewed and discrepancies noted. Because initial estimates of project parameters are tentative, the plan will always need to be modified.

4.2 MEET HADOOP!

Hadoop was created by Doug Cutting , the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

What’s up with the Names?

When naming software projects, Doug Cutting seems to have been inspired by his family.Lucene is his wife’s middle name, and her maternal grandmother’s first name. His son , as a toddler, used Nutch as the all-purpose word for meal and later named a yellow stuffed elephant Hadoop.

Doug said, “ I was looking for a name that wasn’t already a web domain and wasn’t trademarked, so I tried various words that were in my life but not used by anybody else. Kids are pretty good at making up words.”

What is Hadoop?

Apache Hadoop is a Framework that allows for the distributed processing of large datasets across clusters of commodity computers using a simple programming model.

It is an Open source Data management with scale out storage and distributed proceesing.

Hadoop Components

Hadoop designed and built on two independent protocols:

- HDFS (Hadoop Distributed Storage and Filesystem) – HDFS is a reliable distributed file system that provides high-throughput access to data.
- MapReduce(Computing) – MapReduce is a protocol for performing high performance distributed data processing using the divide and aggregate programming paradigm.

Hadoop has a master/slave architecture for both storage and processing.

Benefits of Using Hadoop

- Open Source
- Bright future for Job Seekers
- Low Cost
- Big Data handling
- Fast and Fasticious Working
- Handling various types of data

Who Are Using Hadoop?

Facebook , Amazon Web Services , Horton works , Ebay , Cisco , Google , Twitter , Cloudera , Vmware , IBM , Talend , Yahoo! , Microsoft , Oracle , RedHat , Wipro , Rackspace , Hadapt , EMC^2 , and many more..

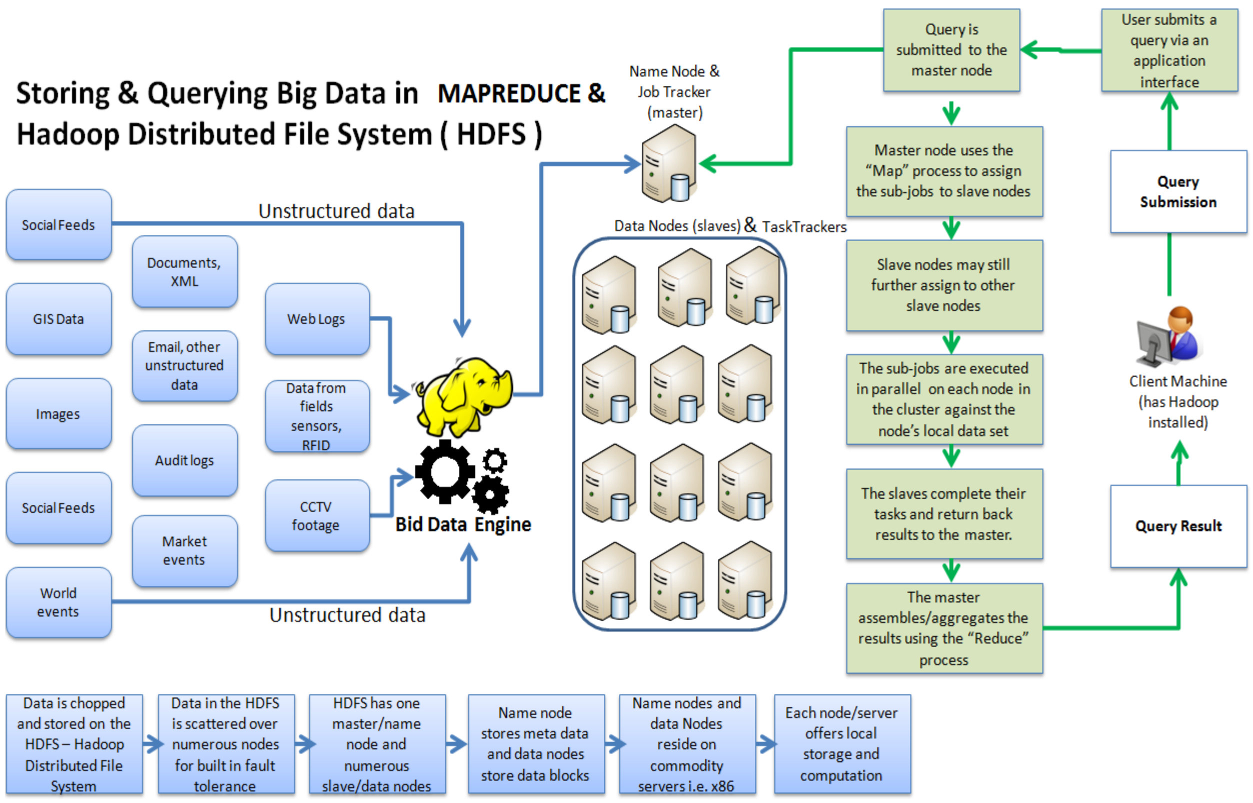

Storing & Querying Big Data in MAPREDUCE & Hadoop Distributed File System (HDFS)
Name Node & Job Tracker (master)
Query is submitted to the master node
User submits a query via an application interface
Master node uses the "Map" process to assign the sub-jobs to slave nodes
Query Submission
Data Nodes (slaves) & TaskTrackers
Slave nodes may still further assign to other slave nodes
The sub-jobs are executed in parallel on each node in the cluster against the node's local data set
Client Machine (has Hadoop installed)
The slaves complete their tasks and return back results to the master.
Query Result
The master assembles/aggregates the results using the "Reduce" process
Social Feeds
GIS Data
Images
Social Feeds
World events
Documents, XML
Email, other unstructured data
Audit logs
Market events
Web Logs
Data from fields sensors, RFID
CCTV footage
Unstructured data
Unstructured data
Bid Data Engine
Data is chopped and stored on the HDFS – Hadoop Distributed File System
Data in the HDFS is scattered over numerous nodes for built in fault tolerance
HDFS has one master/name node and numerous slave/data nodes
Name node stores meta data and data nodes store data blocks
Name nodes and data Nodes reside on commodity servers i.e. x86
Each node/server offers local storage and computation

Figure 3.1 Hadoop Architecture

HDFS Architecture

- <u>NAMENODE</u>
    - Master of the system
    - Maintains and manages the blocks which are present on the datanodes
    - Stores metadata and keeps track of overall file directory
- <u>DATANODE</u>
    - Slaves which are deployed on each machine and provide the actual storage
    - Responsible for serving read and write requests for the client
- <u>SECONDARY NAMENODE</u>

    - Used for checkpointing
    - Connects to namenode every hour and does housekeeping & backup of the metadata

MapReduce Architecture

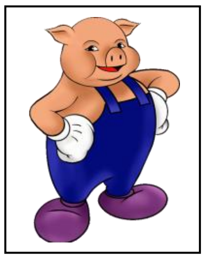
- <u>JOBTRACKER</u>
    - Master of the computing resources
    - Manage & monitor resources and tasks
    - Make tasktrackers to work and assign them jobs
- <u>TASKTRACKER</u>
    - Perform the mapper and reducer tasks
    - Completes the job and give out the results

4.3 OTHER FRAMEWORKS

PIG

Pig raises the level of abstraction for processing large datasets. Pig is made up of two pieces:

- The language used to express the data flows, called Pig Latin.
- The execution environment to run Pig latin programs.



There currently two environments or execution types :

- Local Mode – execution in single JVM
- MapReduce Mode – distributed execution on a Hadoop cluster

A Pig latin program is a series of operations, that are applied to the input data to produce output.

Under the covers, Pig turns the transformations into a series of MapReduce Jobs, which allows us to focus on the data rather than the nature of the execution.

HIVE

Hive was created to make it possible for analysts with strong SQL skills (but meager Java programming skills) to run queries on the huge volumes of data that Facebook stored in HDFS.

Hive grew from a need to manage and learn from the huge volumes of data that Facebook was producing every day from its burgeoning social network.

In normal use, Hive runs on your workstation and converts your SQL query into a series of jobs for execution on a Hadoop cluster. Hive organizes data into tables, which provide a means for attaching structure to data stored in HDFS. Metadata—such as table schemas—is stored in a database called the metastore.

SQOOP

Apache Sqoop is an open source tool that allows users to extract data from a structured data store into Hadoop for further processing. This processing can be done with MapReduce programs or other higher-level tools such as Hive.

It's even possible to use Sqoop to move data from a database into Hbase.

When the final results of an analytic pipeline are available, Sqoop can export these results back to the data store for consumption by other clients.

Sqoop connector is a modular component that uses this framework to enable Sqoop imports and exports. There is also a generic JDBC connector for connecting to any database that supports Java's JDBC protocol. Sqoop provides optimized MySQL, PostgreSQL, Oracle and Netezza connectors that use database-specific APIs to perform bulk transfers more efficiently.

HBASE

It is an unstructured database or NoSQL . There are three types of unstructured database , column oriented, combinational and key-value pair oriented.

Hbase is a distributed column-oriented database built on top of HDFS.

Hbase is the Hadoop application to use when you require real-time read/write random access to very large datasets.

Hbase depends on ZooKeeper, and by default it manages a ZooKeeper instance as the authority on cluster state, although Hbase can be configured to use an existing ZooKeeper cluster instead.

YARN

Apache YARN (Yet Another Resource Negotiator) is Hadoop’s cluster resource management system. YARN was introduced in Hadoop 2 to improve the MapReduce implementation. YARN provides its core services via two types of long-running daemon:

- a <u>resource manager</u> (one per cluster) to manage the use of resources across the cluster
- <u>node managers</u> running on all the nodes in the cluster to launch and monitor containers. A container executes an application-specific process with a constrained set of resources (memory, CPU, and so on).

To run an application on YARN, a client contacts the resource manager and asks it to run an application master process. The resource manager then finds a node manager that can launch the application master in a container. Precisely what the application master does once it is running depends on the application. It could simply run a computation in the container it is running in and return the result to the client.

ZOOKEEPER

It is Hadoop’s distributed coordination service, called ZooKeeper.ZooKeeper also has the following characteristics –

- ZooKeeper is simple
- ZooKeeper is expressive

- ZooKeeper is highly available
- ZooKeeper facilitates loosely coupled interactions
- ZooKeeper is a library
- ZooKeeper is highly performant

4.4 RED HAT:

Linux is an operating system that evolved from a kernel created by Linus Benedict Torvlads when he was the second year student of computer science at the University of Helsinki in 1991.

Linux is an operating system that can be downloaded free and “belongs” to an entire community of developers , not one corporate entity.

That’s why Linux is known as “Open Source” or “Free Software”, because there is nothing secret about this system.

What is Open Source?

It means Software and source code is available to all. The freedom to distribute source code and software to all.

The ability to modify and create derived works.

Linux Principles

- Everything is a file (including hardware)
- Small , single-purpose programs
- Ability to chain programs together to perform complex tasks
- Configuration data stored in text

RHEL 6.4

Red Hat Enterprise Linux (**RHEL**) is a Linux distribution developed by Red Hat and targeted toward the commercial market.

Red Hat Enterprise Linux 6 was forked from Fedora 12 and contains many back ported features from Fedora 13 and 14.

- Red Hat Enterprise Linux 6 (*Santiago*), 10 November 2010, uses Linux kernel 2.6.32-71
- 6.1, also termed Update 1, 19 May 2011 (kernel 2.6.32-131)
- 6.2, also termed Update 2, 6 December 2011 (kernel 2.6.32-220)
- 6.3, also termed Update 3, 20 June 2012 (kernel 2.6.32-279)
- 6.4, also termed Update 4, 21 February 2013 (kernel 2.6.32-358)
- 6.5, also termed Update 5, 21 November 2013 (kernel 2.6.32-431)
- 6.6, also termed Update 6, 13 October 2014 (kernel 2.6.32-504)
- 6.7, also termed Update 7, 22 July 2015 (kernel 2.6.32-573)

4.5 SCALA

Scala is a general purpose programming language. Scala has full support for functional programming and a very strong static type system. Designed to be concise,[8] many of Scala's design decisions were inspired by criticism of the shortcomings of Java.

Scala source code is intended to be compiled to Java bytecode, so that the resulting executable code runs on a Java virtual machine. Java libraries may be used directly in Scala code and vice versa (language interoperability). Like Java, Scala is object-oriented, and uses a curly-brace syntax reminiscent of the C programming language. Unlike Java, Scala has many features of functional programming languages like Scheme, Standard ML and Haskell, including currying, type inference, immutability, lazy evaluation, and pattern matching. It also has an advanced type system supporting algebraic data types, covariance and contravariance, higher-order types (but not higher-rank types), and anonymous types. Other features of Scala not present in Java include operator overloading, optional parameters, named parameters, raw strings, and no checked exceptions.

Scala is object-oriented:

Scala is a pure object-oriented language in the sense that every value is an object. Types and behavior of objects are described by classes and traits which will be explained in subsequent chapters.

Classes are extended by **subclassing** and a flexible **mixin-based composition** mechanism as a clean replacement for multiple inheritance.

Scala is functional:

Scala is also a functional language in the sense that every function is a value and because every value is an object so ultimately every function is an object.

Scala provides a lightweight syntax for defining **anonymous functions**, it supports **higher-order functions**, it allows functions to be **nested**, and supports **currying**. These concepts will be explained in subsequent chapters.

Scala is statically typed:

Scala, unlike some of the other statically typed languages, does not expect you to provide redundant type information. You don't have to specify a type in most cases, and you certainly don't have to repeat it.

Scala runs on the JVM:

Scala is compiled into Java Byte Code which is executed by the Java Virtual Machine (JVM). This means that Scala and Java have a common runtime platform. You can easily move from Java to Scala.

The Scala compiler compiles your Scala code into Java Byte Code, which can then be executed by the **scala** command. The **scala** command is similar to the**java** command, in that it executes your compiled Scala code.

Scala can Execute Java Code:

Scala enables you to use all the classes of the Java SDK's in Scala, and also your own, custom Java classes, or your favourite Java open source projects.

Scala vs Java:

Scala has a set of features, which differ from Java. Some of these are:

- All types are objects.
- Type inference.
- Nested Functions.
- Functions are objects.
- Domain specific language (DSL) support.
- Traits.
- Closures.
- Concurrency support inspired by Erlang.

Scala Web Frameworks:

Scala is being used everywhere and importantly in enterprise web applications. You can check few of the most popular Scala web frameworks:

- The Lift Framework
- The Play framework
- The Scala language can be installed on any UNIX-like or Windows system. Before you start installing Scala on your machine, you must have Java 1.5 or greater installed on your computer.
- **Installing Scala on Windows:**
- Step (1): JAVA Setup:
- First, you must set the JAVA_HOME environment variable and add the JDK's bin directory to your PATH variable. To verify if everything is fine, at command prompt, type **java -version** and press Enter. You should see something like the following:

```
• C:\>java -version

• java version "1.6.0_15"

• Java(TM) SE Runtime Environment (build 1.6.0_15-b03)

• Java HotSpot(TM) 64-Bit Server VM (build 14.1-b02, mixed mode)
```

```
C:\>
```

- Next, test to see that the Java compiler is installed. Type **javac -version**. You should see something like the following:

```
C:\>javac -version
javac 1.6.0_15
C:\>
```

Step (2): Scala Setup:

Next, you can download Scala from http://www.scala-lang.org/downloadsand install windows binaries. Finally, open a new command prompt and type**scala -version** and press Enter. You should see the following:

```
C:\>scala -version
Scala code runner version 2.9.0.1 -- Copyright 2002-2011, LAMP/EPFL
C:\>
```

Congratulations, you have installed Scala on your Windows machine. Next section will teach you how to install scala on your Mac OS X and Unix/Linux machines.

- **Installing Scala on Mac OS X and Linux**
- Step (1): JAVA Setup:
- Make sure you have got the Java JDK 1.5 or greater installed on your computer and set JAVA_HOME environment variable and add the JDK's bin directory to your PATH variable. To verify if everything is fine, at command prompt, type**java -version** and press Enter. You should see something like the following:

```
$java -version
java version "1.5.0_22"
```

```
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_22-b03)
Java HotSpot(TM) Server VM (build 1.5.0_22-b03, mixed mode)
$
```

Next, test to see that the Java compiler is installed. Type **javac -version**. You should see something like the following:

```
$javac -version
javac 1.5.0_22
javac: no source files
Usage: javac <options> <source files>
$
```

Step (2): Scala Setup:

Open your terminal and run the following commands as shown below

```
$ wget http://www.scala-lang.org/files/archive/scala-2.11.4.deb
$ sudo dpkg -i scala-2.11.4.deb
$ sudo apt-get update
$ sudo apt-get install scala
```

Finally, open a new command prompt and type **scala -version** and press Enter. You should see the following:

```
$scala -version
Scala code runner version 2.11.4 -- Copyright 2002-2013, LAMP/EPFL
```

Congratulations, you have installed Scala on your UNIX/Linux machine.

4.6 CLOUDERA MANAGER

Cloudera's open-source Apache Hadoop distribution, CDH (Cloudera Distribution Including Apache Hadoop), targets enterprise-class deployments of that technology. Cloudera says that more than 50% of its engineering output is donated upstream to the various Apache-licensed open source projects (Apache Hive, Apache Avro, Apache HBase, and so on) that combine to form the Hadoop platform

Cloudera Manager is the industry’s first and most sophisticated management application for Apache Hadoop. Cloudera Manager sets the standard for enterprise deployment by delivering granular visibility into and control over every part of the Hadoop cluster — empowering operators to improve performance, enhance quality of service, increase compliance and reduce administrative costs. With Cloudera Manager, you can easily deploy and centrally operate the complete Hadoop stack. The application automates the installation process, reducing deployment time from weeks to minutes; gives you a cluster-wide, real-time view of nodes and services running; provides a single, central console to enact configuration changes across your cluster; and incorporates a full range of reporting and diagnostic tools to help you optimize performance and utilization

4.6.1Terminology

To effectively use Cloudera Manager, you should first understand its terminology. The relationship between the terms is illustrated below and their definitions follow:

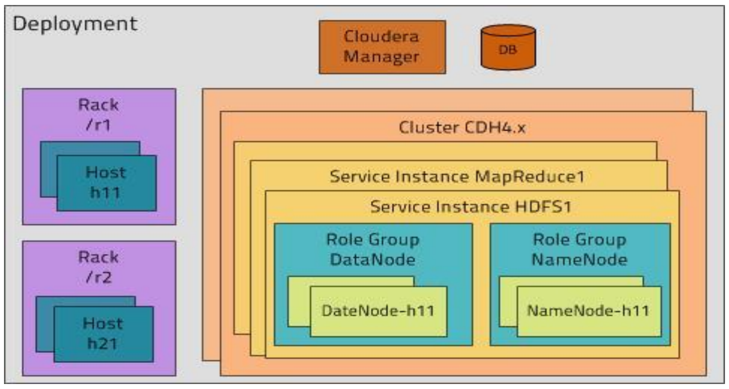


Fig 3 Cloudera Manager

Some of the terms, such as cluster and service, will be used without further explanation. Others, such as gateway and role group, are expanded upon in later sections.

- **deployment** - A configuration of Cloudera Manager and all the services and hosts it manages.
- **cluster** - A logical entity that contains a set of hosts, a single version of CDH installed on the hosts, and the service and role instances running on the hosts. A host may only belong to one cluster.
- **host** - A logical or physical machine that runs role instances.
- **rack** - A physical entity that contains a set of hosts typically served by the same switch.
- **service** - A category of functionality within the Hadoop ecosystem. Sometimes referred to as a service type. For example: HDFS, MapReduce.
- **service instance** - An instance of a service running on a cluster. A service instance spans many role instances. For example: HDFS-1 and MapReduce-1.
- **role** - A category of functionality within a service. For example, the HDFS service has the following roles: NameNode, SecondaryNameNode, DataNode, and Balancer. Sometimes referred to as a role type.
- **role instance** - An instance of a role, assigned to a host. It typically maps to a Unix process. For example: NameNode-h11 and DataNode-h11.
- **role group** - A set of configuration properties for a role and set of role instances.
- **host template** - A set of role groups. When a template is applied to a host, a role instance from each role group is created and assigned to that host.
- **gateway** - A role that designates a host that should receive a client configuration for a service when the host does not have any roles for that service running on it.

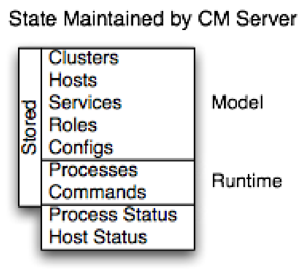
4.6.2Heartbeating

Heartbeats are a primary communication mechanism in Cloudera Manager. By default the Agents send heartbeats every 15 seconds to the Cloudera Manager Server. However, to reduce user latency the frequency is increased when state is changing.

During the heartbeat exchange the Agent notifies the Cloudera Manager Server the actions it is performing. In turn the Cloudera Manager Server responds with the actions the Agent should be performing. Both the Agent and the Cloudera Manager Server end up doing some reconciliation. For example, if you start a service via the UI, the Agent attempts to start the relevant processes; if a process fails to start, the server marks the start command as having failed.

4.6.3State Management

The Cloudera Manager Server maintains the state of the cluster. This state can be divided into two categories: "model" and "runtime", both of which are stored in the Cloudera Manager Server database.



Cloudera Manager models the Hadoop stack: its roles, configurations, and inter-dependencies. *Model state* captures what is supposed to run where, and with what configurations. That a cluster contains 17 hosts, each of which is supposed to run a DataNode, is model state. You interact with the model through the Cloudera Manager configuration screens (and operations like "Add Service").

Runtime state is what processes are running where, and what commands (for example, rebalance HDFS or execute a Backup/Disaster Recovery schedule or rolling restart or stop) are currently being executed. The runtime state includes the exact configuration files needed to run a process. In fact, when you press "Start" in Cloudera Manager Admin Console, the server gathers up all the configuration for the relevant services and roles, validates it, generates the configuration files, and stores them in the database.

When you update a configuration (for example, the Hue Server web port), you've updated the model state. However, if Hue is running while you do this, it's still using the old port. When this kind of mismatch occurs, the role is marked as having an "outdated configuration". To resynchronize, you restart the role (which triggers the configuration re-generation and process restart).

While Cloudera Manager models all of the reasonable configurations, inevitably, there are some corner cases that require special handling. To allow you to workaround, for example, some bug (or, perhaps, to explore unsupported options), Cloudera Manager supports a "safety valve" mechanism that lets you plug in directly to the configuration files. (The analogy stems from the idea that a safety valve "releases pressure" if the model doesn't hold up to a real-world use case.)

Configuration Management

Cloudera Manager defines configuration at several levels:

- The service level may define configurations that apply to the entire service instance, such as an HDFS service's default replication factor (dfs.replication).
- The **role group** level may define configurations that apply to the member roles, such as the DataNodes' handler count (dfs.datanode.handler.count). This can be set differently for different groups of DataNodes. For example, DataNodes running on more capable hardware may have more handlers.
- The role instance level may override configurations that it inherits from its role group. This should be used sparingly, because it easily leads to configuration divergence within the role group. One example usage is to temporarily enable debug logging in a specific role instance to troubleshoot an issue.
- Hosts have configurations related to monitoring, software management, and resource management.
- Cloudera Manager itself has configurations related to its own administrative operations.

4.6.4 Role Groups

It is possible to set configuration at the service instance (for example, HDFS-1), role (for example, all DataNodes), or role instance (for example, the DataNode on host17). An individual role inherits the configurations set at the service and role-type levels. Configurations made at the role level override those from the role-type level. While this approach offers flexibility when configuring clusters, it is tedious to configure subsets of roles in the same way.

Cloudera Manager supports role groups, a mechanism for assigning configurations to a group. The members of those groups then inherit those configurations. For example, in a cluster with heterogeneous hardware, a DataNode role group can be created for each host type and the DataNodes running on those hosts can be assigned to their corresponding role group. That makes it possible to set the configuration for all the DataNodes running on the same hardware by modifying the configuration of one role group.

In addition to making it easy to manage the configuration of subsets of roles, role groups also make it possible to maintain different configurations for experimentation or managing shared clusters for different users and/or workloads.

4.6.5Host Templates

In typical environments, sets of hosts have the same hardware and the same set of services running on them. A host template defines a set of role groups (at most one of each type) in a cluster and provides two main benefits:

- Adding new hosts to clusters easily - multiple hosts can have roles from different services created, configured, and started in a single operation.
- Altering the configuration of roles from different services on a set of hosts easily - which is useful for quickly switching the configuration of an entire cluster to accommodate different workloads or users.

4.6.6Server and Client Configuration

Administrators are sometimes surprised that modifying /etc/hadoop/conf and then restarting HDFS has no effect. That is because service instances started by Cloudera Manager do not read configurations from the default locations. To use HDFS as an example, when not managed by Cloudera Manager, there would usually be one HDFS configuration per host, located at /etc/hadoop/conf/hdfs-site.xml. Server-side daemons and clients running on the same host would all use that same configuration.

Cloudera Manager distinguishes between server and client configuration. In the case of HDFS, the file /etc/hadoop/conf/hdfs-site.xml contains only configuration relevant to an HDFS client. That is, by default, if you run a program that needs to communicate with Hadoop, it will get the addresses of the NameNode and JobTracker, and other important configurations, from that directory. A similar approach is taken for /etc/hbase/conf and /etc/hive/conf.

The HDFS server-side daemons (for example, NameNode and DataNode) obtain their configurations from a private per-process directory, under /var/run/cloudera-scm-agent/process/*unique-process-name*. Giving each process its own private execution and configuration environment allows us to control each process independently, which is crucial for some of the more esoteric configuration scenarios that show up. Here are the contents of an example 879-hdfs-NAMENODE process directory:

```
$ tree -a /var/run/cloudera-scm-Agent/process/879-hdfs-NAMENODE/

 /var/run/cloudera-scm-Agent/process/879-hdfs-NAMENODE/

  ├── cloudera_manager_Agent_fencer.py

  ├── cloudera_manager_Agent_fencer_secret_key.txt

  ├── cloudera-monitor.properties

  ├── core-site.xml
  ├── dfs_hosts_allow.txt
  ├── dfs_hosts_exclude.txt
  ├── event-filter-rules.json
  ├── hadoop-metrics2.properties
  ├── hdfs.keytab
  ├── hdfs-site.xml
  ├── log4j.properties
  ├── logs
  │   ├── stderr.log
  │   └── stdout.log
  ├── topology.map
  └── topology.py
```

There are several advantages to distinguishing between server and client configuration:

- Sensitive information in the server-side configuration, such as the password for the Hive metastore RDBMS, is not exposed to the clients.
- A service that depends on another service may deploy with customized configuration. For example, to get good HDFS read performance, Cloudera Impala needs a specialized version of the HDFS client configuration, which may be harmful to a generic client. This is achieved by separating the HDFS configuration for the Impala daemons (stored in the per-process directory mentioned above) from that of the generic client (/etc/hadoop/conf).
- Client configuration files are much smaller and more readable. This also avoids confusing non-administrator Hadoop users with irrelevant server-side properties.

4.6.7 Deploying Client Configurations and Gateways

A client configuration is a ZIP file that contain the relevant configuration files with the settings for a service. Each zip file contains the set of configuration files needed by the service. For example, the

MapReduce client configuration zip file contains copies of core-site.xml, hadoop-env.sh, hdfs-site.xml, log4j.properties, and mapred-site.xml. Cloudera Manager supports a "Client Configuration URLs" action to distribute the client configuration file to users outside the cluster.

Cloudera Manager can deploy client configurations within the cluster; each applicable service has a "Deploy Client Configuration" action. This action does not necessarily deploy the client configuration to the entire cluster; it only deploys the client configuration to all the hosts that this service has been assigned to. For example, suppose a cluster has 10 nodes, and a MapReduce service is running on hosts 1-9. When you use Cloudera Manager to deploy the MapReduce client configuration, host 10 will not get a client configuration, because the MapReduce service has no role assigned to it. This design is intentional to avoid deploying conflicting client configurations from multiple services.

To deploy a client configuration to a host that does not have a role assigned to it you use a gateway. A **gateway** is a marker to convey that a service includes a particular host. Unlike all other roles it has no associated process. In the preceding example, to deploy the MapReduce client configuration to host 10, you assign a MapReduce gateway role to that host.

Process Management

In a non-Cloudera Manager managed cluster, you most likely start a role instance using an init script, for example, service hadoop-hdfs-datanode start. Cloudera Manager does not use init scripts for the daemons it manages; in a Cloudera Manager managed cluster, starting and stopping services using init scripts will not work.

In a Cloudera Manager managed cluster you can only start or stop services via Cloudera Manager. Cloudera Manager uses an open source supervisor called **supervisord**that takes care of redirecting log files, notifying of process failure, setting the effective user ID of the calling process to the right user, and so forth. Cloudera Manager supports automatically restarting a crashed process. It will also flag a role instance with a bad state if it crashes repeatedly right after start up.

It is worth noting that stopping Cloudera Manager and the Cloudera Manager Agents will not bring down your cluster; any running instances will keep running.

One of the Agent's main responsibilities is to start and stop processes. When the Agent detects a new process from the heartbeat, the Agent creates a directory for it in/var/run/cloudera-scm-agent and unpacks the configuration. These actions reflect an important point: a Cloudera Manager process never travels alone. In other words, a process is more than just the arguments to exec()—it also includes configuration files, directories that need to be created, and other information.

The Agent itself is started by init.d at start-up. It, in turn, contacts the server and figures out what processes should be running. The Agent is monitored as part of Cloudera Manager's host monitoring: if the Agent stops heartbeating, the host will be marked as having bad health.

4.6.8 Software Distribution Management

A major function of Cloudera Manager is to distribute and activate software in your cluster. Cloudera Manager supports two software distribution formats: packages and parcels.

A **package** is a distribution format that contains compiled code and meta-information such as a package description, version, and dependencies. Package management systems evaluate this meta-information to allow package searches, perform upgrades to a newer version, and ensure that all dependencies of a package are fulfilled. Cloudera Manager uses the native "system package manager" for each supported OS.

A **parcel** is a binary distribution format containing the program files, along with additional metadata used by Cloudera Manager. There are a few notable differences between parcels and packages:

- Parcels are self-contained and installed in a versioned directory. This means that multiple versions of a given parcel can be installed "side-by-side". You can then designate one of these installed versions as the "active" one. With traditional packages, only one package can be installed at a time so there's no distinction between what's "installed" and what's "active".
- Parcels can be installed at any location in the filesystem.

Advantages of Parcels

As a consequence of their unique properties, parcels offer a number of advantages over packages:

- **CDH is distributed as a single object** - In contrast to having a separate package for each part of CDH, when using parcels there is just a single object to install. This is especially useful when managing a cluster that isn't connected to the Internet.

- **Internal consistency** - All CDH components are matched so there isn't a danger of different parts coming from different versions of CDH.
- **Installation outside of /usr** - In some environments, Hadoop administrators do not have privileges to install system packages. In the past, these administrators had to fall back to CDH tarballs, which deprived them of a lot of infrastructure that packages provide. With parcels, administrators can install to /opt or anywhere else without having to step through all the additional manual steps of regular tarballs.

Resources

Cloudera Manager Server:

- 5 GB on the partition hosting /var.
- 500 MB on the partition hosting /usr.
- For parcels, the space required will depend on the number of parcels you download to the Cloudera Manager server, and distribute to agent nodes. You can download multiple parcels of the same product, of different versions and builds.
  Even if you are managing multiple clusters, there will be only one parcel of any given product/version/build/distro downloaded on the Cloudera Manager server — not one per cluster.

  In the local parcel repo on the Cloudera Manager server the approximate sizes of the various parcels (as of Cloudera Manager 4.6) are as follows:

- CDH4.3 — slightly over 700MB per parcel.
- Impala — approximately 200MB per parcel.
- Cloudera Search (Solr) — approximately 400MB per parcel.
- **Cloudera Manager Agent** Each unpacked parcel will require about three times the space of the downloaded parcel on the Cloudera Manager Server.
- **RAM** - 4 GB is appropriate for most cases, and is required when using Oracle databases. 2GB may be sufficient for non-Oracle deployments involving fewer than 100 hosts.

4.7 APACHE SPARK

Apache Spark is an open source cluster computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation that has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.It was developed in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

4.7.1 SPARK CORE

Spark Core is the foundation of the overall project. It provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface (for Java, Python, Scala, and R) centered on the RDD abstraction. This interface mirrors a functional/higher-order model of programming: a "driver" program invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster. These operations, and additional ones such as joins, take RDDs as input and produce new RDDs. RDDs are immutable and their operations are lazy; fault-tolerance is achieved by keeping track of the "lineage" of each RDD, the sequence of operations produced it, so that it can be reconstructed in the case of data loss. RDDs can contain any type of Python, Java, or Scala objects.

Aside from the RDD-oriented functional style of programming, Spark provides two restricted forms of shared variables: *broadcast variables* reference read-only data that needs to be available on all nodes, while *accumulators* can be used to program reductions in an imperative style.

CHAPTER 5: EXPERIMENTAL SETUP & IMPLEMENTATION

Spark SQL is a component on top of Spark Core that introduces a new data abstraction called DataFrames, which provides support for structured and semi-structured data. Spark SQL provides a domain-specific language to manipulate DataFrames in Scala, Java, or Python. It also provides SQL language support, with command-line interfaces and ODBC/JDBC server

5.1 SPARK STREAMING

Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics. It ingests data in mini-batches and performs RDD transformations on those mini-batches of data. This design enables the same set of application code written for batch analytics to be used in streaming analytics, on a single engine.

5.2 SPARK MLLIB

Spark MLlib is a distributed machine learning framework on top of Spark Core that, due in large part of the distributed memory-based Spark architecture, is as much as nine times as fast as the disk-based implementation used by Apache Mahout (according to benchmarks done by the MLlib developers against the Alternating Least Squares (ALS) implementations, and before Mahout itself gained a Spark interface), and scales better than Vowpal Wabbit. Many common machine learning and statistical algorithms have been implemented and are shipped with MLlib which simplifies large scale machine learning pipelines, including:

- summary statistics, correlations, stratified sampling, hypothesis testing, random data generation
- classification and regression: support vector machine, logistic regression, linear regression, decision trees, naive Bayes classification
- collaborative filtering techniques including alternating least squares (ALS)
- cluster analysis methods including k-mean, and Latent Dirichlet Allocation (LDA)

- dimensionality reduction techniques such as singular value decomposition (SVD), and principal component analysis (PCA)
- feature extraction and transformation functions
- optimization algorithms such as stochastic gradient descent, limited-memory BFGS (L-BFGS)

5.3 MLLIB ALGORITHMS FOR CHURN ANALYSIS :

5.3.1 DECISION TREE(MLLIB):

Decision trees and their ensembles are popular methods for the machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions. Tree ensemble algorithms such as random forests and boosting are among the top performers for classification and regression tasks.

spark.mllib supports decision trees for binary and multiclass classification and for regression, using both continuous and categorical features. The implementation partitions data by rows, allowing distributed training with millions of instances.

Ensembles of trees (Random Forests and Gradient-Boosted Trees) are described in the Ensembles guide.

BASIC ALGORITHM:

The decision tree is a greedy algorithm that performs a recursive binary partitioning of the feature space. The tree predicts the same label for each bottommost (leaf) partition. Each partition is chosen greedily by selecting the *best split* from a set of possible splits, in order to maximize the information gain at a tree node.

Continuous features

For small datasets in single-machine implementations, the split candidates for each continuous feature are typically the unique values for the feature. Some implementations sort the feature values and then use the ordered unique values as split candidates for faster tree calculations.

Sorting feature values is expensive for large distributed datasets. This implementation computes an approximate set of split candidates by performing a quantile calculation over a sampled fraction of the data. The ordered splits create “bins” and the maximum number of such bins can be specified using the maxBins parameter.

Note that the number of bins cannot be greater than the number of instances NN (a rare scenario since the default maxBinsvalue is 32). The tree algorithm automatically reduces the number of bins if the condition is not satisfied.

Categorical features

For a categorical feature with MM possible values (categories), one could come up with 2M−1−12M−1−1 split candidates. For binary (0/1) classification and regression, we can reduce the number of split candidates to M−1M−1 by ordering the categorical feature values by the average label. (See Section 9.2.4 in Elements of Statistical Machine Learning for details.) For example, for a binary classification problem with one categorical feature with three categories A, B and C whose corresponding proportions of label 1 are 0.2, 0.6 and 0.4, the categorical features are ordered as A, C, B. The two split candidates are A | C, B and A , C | B where | denotes the split.

In multiclass classification, all 2M−1−12M−1−1 possible splits are used whenever possible. When 2M−1−12M−1−1 is greater than themaxBins parameter, we use a (heuristic) method similar to the method used for binary classification and regression. The MMcategorical feature values are ordered by impurity, and the resulting M−1M−1 split candidates are considered.

Stopping rule

The recursive tree construction is stopped at a node when one of the following conditions is met:

1. The node depth is equal to the maxDepth training parameter.
2. No split candidate leads to an information gain greater than minInfoGain.
3. No split candidate produces child nodes which each have at least minInstancesPerNode training instances.

Usage tips

We include a few guidelines for using decision trees by discussing the various parameters. The parameters are listed below roughly in order of descending importance. New users should mainly consider the “Problem specification parameters” section and the maxDepth parameter.

Problem specification parameters

These parameters describe the problem you want to solve and your dataset. They should be specified and do not require tuning.

- **algo**: Classification or Regression
- **numClasses**: Number of classes (for Classification only)
- **categoricalFeaturesInfo**: Specifies which features are categorical and how many categorical values each of those features can take. This is given as a map from feature indices to feature arity (number of categories). Any features not in this map are treated as continuous.
    - E.g., Map(0 -> 2, 4 -> 10) specifies that feature 0 is binary (taking values 0 or 1) and that feature 4 has 10 categories (values {0, 1, ..., 9}). Note that feature indices are 0-based: features 0 and 4 are the 1st and 5th elements of an instance's feature vector.
    - Note that you do not have to specify categoricalFeaturesInfo. The algorithm will still run and may get reasonable results. However, performance should be better if categorical features are properly designated.

Stopping criteria

These parameters determine when the tree stops building (adding new nodes). When tuning these parameters, be careful to validate on held-out test data to avoid overfitting.

- **maxDepth**: Maximum depth of a tree. Deeper trees are more expressive (potentially allowing higher accuracy), but they are also more costly to train and are more likely to overfit.
- **minInstancesPerNode**: For a node to be split further, each of its children must receive at least this number of training instances. This is commonly used with RandomForest since those are often trained deeper than individual trees.
- **minInfoGain**: For a node to be split further, the split must improve at least this much (in terms of information gain

5.3.2 NAÏVE BAYES(MLLIB):

Naive Bayes is a simple multiclass classification algorithm with the assumption of independence between every pair of features. Naive Bayes can be trained very efficiently. Within a single pass to the training data, it computes the conditional probability distribution of each feature given label, and then it applies Bayes' theorem to compute the conditional probability distribution of label given an observation and use it for prediction.

Spark.mllib supports multinomial naive Bayes and Bernoulli naive Bayes. These models are typically used for document classification. Within that context, each observation is a document and each feature represents a term whose value is the frequency of the term (in multinomial naive Bayes) or a zero or one indicating whether the term was found in the document (in Bernoulli naive Bayes). Feature values must be nonnegative. The model type is selected with an optional parameter "multinomial" or "bernoulli" with "multinomial" as the default. Additive smoothing can be used by setting the parameter λλ(default to 1.01.0). For document classification, the input feature vectors are usually sparse, and sparse vectors should be supplied as input to take advantage of sparsity. Since the training data is only used once, it is not necessary to cache it.

5.3.3 Project Module Description:

SINGLE NODE CLUSTER:

The steps involved in setting up a single node Hadoop cluster are as follow:

1. Download the Hadoop Software, the hadoop.tar.gz file using the ftp://hadoop.apche.org URL, and ensure that the software is installed on every node of the cluster. Installing the Hadoop Software on all the nodes require unpacking of the software, the hadoop.apache.org URL, on the nodes.
2. Create the keys on local machine such that ssh, required by Hadoop, does not need password. Use following command to create key on local machine:

```
$ ssh-keygen -t rsa -P " "
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

3. Modify the environment parameters in the hadoop-env.sh file. Use the following command to change the environment parameter:

Export JAVA_HOME=/path/to/jdk_home_dir

4. Modify the configuration parameters in files given below as shown below.

Do the following changes to the configuration files under hadoop/conf

core-site.xml

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>TEMPORARY-DIR-FOR-HADOOPDATASTORE</value>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
</property>
</configuration>
```

mapred-site.xml

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
</property></configuration>
```

hdfs-site.xml

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

4. Format the hadoop file system. From hadoop directory run the following:

```
bin/hadoop namenode –format
```

MULTI NODE CLUSTER:

The initial steps are same as Single Node Cluster.The steps involved in setting up a multi node Hadoop cluster are as follow:

5. Download the Hadoop Software, the hadoop.tar.gz file using the ftp://hadoop.apche.org URL, and ensure that the software is installed on every node of the cluster. Installing the Hadoop Software on all the nodes require unpacking of the software, the hadoop.apache.org URL, on the nodes.
6. Create the keys on local machine such that ssh, required by Hadoop, does not need password. Use following command to create key on local machine:

```
$ ssh-keygen -t rsa -P " "
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

7. Modify the environment parameters in the hadoop-env.sh file. Use the following command to change the environment parameter:

```
Export JAVA_HOME=/path/to/jdk_home_dir
```

8. Modify the configuration parameters in files given below as shown below.
   Do the following changes to the configuration files under hadoop/conf

core-site.xml

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>TEMPORARY-DIR-FOR-HADOOPDATASTORE</value>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://192.168.0.15:5900</value>
</property>
</configuration>
```

mapred-site.xml

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>l92.168.0.20:5901</value>
</property>
</configuration>
```

hdfs-site.xml<configuration>

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```

9. Format the hadoop file system. From hadoop directory run the following

```
bin/hadoop namenode –format
```

to install spark on standalone:-

installing java:-

```
yum update
yum install git
yum install java-1.6.0-openjdk
```

Installing scala

Download the latest version of Scala by visit the following link Download http://www.scala-lang.org/download/

type the following command.

```
$ tar xvf scala-2.11.6.tgz
```

- Use the following commands for moving the Scala software files, to respective directory (/usr/local/scala).

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv scala-2.11.6 /usr/local/scala
# exit
```

- Set PATH for Scala

```
$ export PATH = $PATH:/usr/local/scala/bin
```

- Verifying Scala Installation

```
$scala -version
Scala code runner version 2.11.6 -- Copyright 2002-2013, LAMP/EPFL
```

- Download the latest version of Spark by visiting the following link Download

    https://spark.apache.org/downloads.html

    extracting files:-

```
$ tar xvf spark-1.3.1-bin-hadoop2.6.tgz
```

    moving spark software files:-

```
$ su –
Password:

# cd /home/Hadoop/Downloads/
# mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark
# exit
```

- Setting up the environment for Spark

  Add the following line to ~/.bashrc file. It means adding the location, where the spark software file are located to the PATH variable.

```
export PATH = $PATH:/usr/local/spark/bin
```

Use the following command for sourcing the ~/.bashrc file.

```
$ source ~/.bashrc
```

Step 7: Verifying the Spark Installation

Write the following command for opening Spark shell.

```
$spark-shell
```

5.4 Customer churn analysis in telecommunication sector :

5.4.1 LITERATURE REVIEW_: Marketing research literature has noted that 'customer churn' is a term used in the cellular mobile telecom service industry to indicate the customer movement from one provider to another, and 'churn management' is a term that describes an operator's process to retain profitable customers. Churn is also called attrition and often used to indicate a customer leaving the service of one company in favor of another company. In essence, effective customer management presumes an ability to forecast the customer decision to shift from one service provider to another.

5.4.2 PROBLEM DESCRIPTION : Given a set of data related to a customer we need to predict whether the customer will churn or not.

5.4.3 DATASET USED : The data set contains 20 variables worth of information about 3,333 customers, along with an indication of whether or not that customer churned (left the company). The variables are as follows-

1.) State: categorical variable, for the 50 states and the district of Columbia
2.) Account length: integer-valued variable for how long account has been active

3.) Area code: categorical variable

4.) Phone number: essentially a surrogate key for customer identification

5.) International Plan: dichotomous categorical having yes or no value

6.) Voice Mail Plan: dichotomous categorical variable having yes or no value

7.) Number of voice mail messages: integer-valued variable

8.) Total day minutes: continuous variable for number of minutes customer has used the service in day.

9.) Total day calls: integer-valued variable.

10.) Total day calls: integer-valued variable.

11.)Total day charge: continuous variable based on foregoing two variables

12.)Total evening minutes: continuous variable for minutes customer has used the service during the

13.)Total evening calls: integer-valued variable

14.)Total evening charge: continuous variable based on previous two variables.

15.)Total night minutes: continuous variable for storing minutes the customer has used the service

16.)Total night calls: integer-valued variable

17.)Total night charge: continuous variable based on foregoing two variables

18.)Total international minutes: continuous variable for minutes customer has used service to make

19.)Total international calls: integer-valued variable

20.)Total international charge: continuous variable based on foregoing two variables.

21.)CHURN : This Boolean shows whether the customer churned or not ('1' for churned and '0' for a customer who continued with the services) .

5.4.4 METHODOLOGY USED :

I have used a SUPERVISED learning methodology wherein the label for each input record of training set is known to us, using which we will generate a model for prediction.

5.4.5 Feature Extraction : In this step the features of the dataset which are most likely to effect the decision of the customer to churn or not are extracted/identified. In our dataset we will use all the features except for STATE, ACCOUNT LENGTH, AREA CODE, PHONE NUMBER, EVENING CALLS, INTL CALLS, and NIGHT CALLS.

5.4.6 Model Construction :

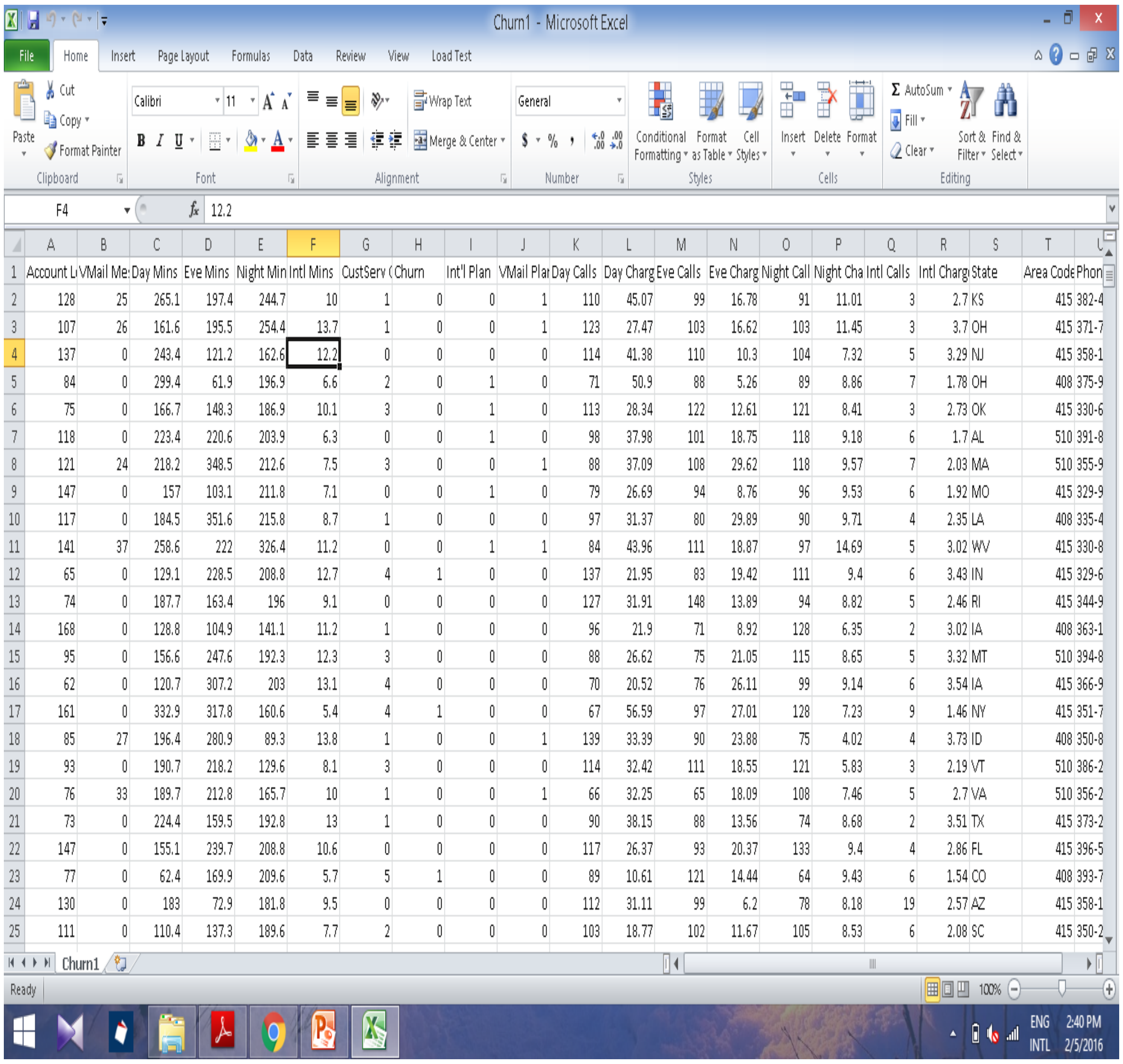
I used DECISION TREE ALGORITHM and NAÏVE BAYES for generation of the prediction model for this problem. The Decision tree algorithm uses the 'divide and conquer' method to construct a model based on a tree structure. Nodes in the tree represent features, with branches representing possible values connecting the features. A leaf representing the class terminates a series of nodes and branches. Initially, the method starts to search an attribute with best information gain at root node and divide the tree into sub-trees. Similarly, each sub-tree is further separated recursively following the same rule.

The partitioning stops if the leaf node is reached or there is no information gain or the depth of the tree equals the predefined maxDepth parameter. Once the tree is created, rules can be obtained by traversing each branch of the tree.

5.4.7 Framework Used :

I used Apache Spark for the implementation of the model and used SCALA as the implementation language. Apache Spark has an Inbuilt library / API called MLLIB which has most of the popular machine learning algorithms implemented. We just need to tweak with the parameters that these API's need inorder to function as we want them to. In our case decision tree algorithm has an inbuilt function called **trainClassifier()**, which takes number of classes(int) , impurity(string), num of bins(int), description of categorical variables(Map()), and training set in the form of a **LabeledPoint()** , as its parameters. A LabeledPoint is a collection type in Spark which takes the label(double) associated with the features in each record and a Vector(), which is a collection of all the features on which we think the label depends.

5.5 DATASET USED

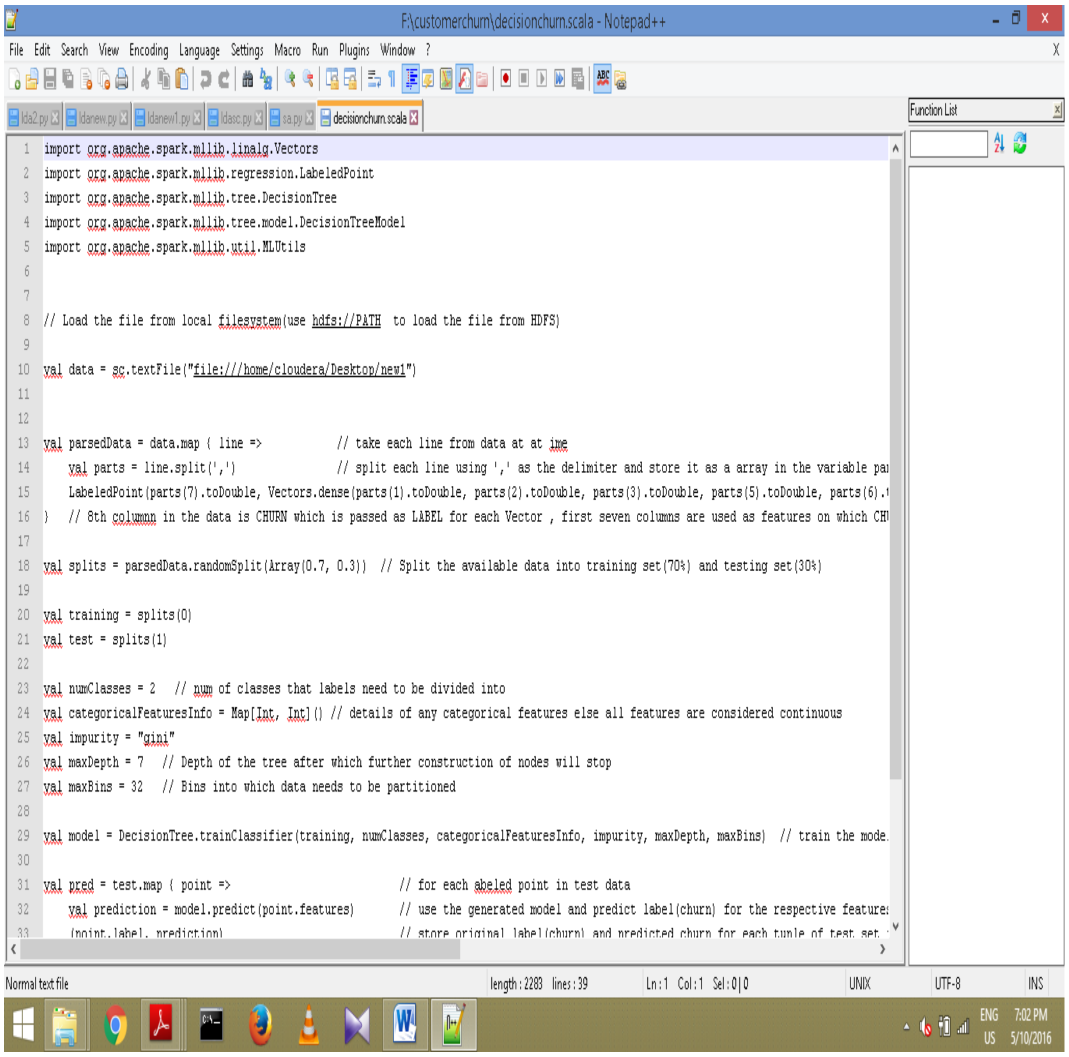


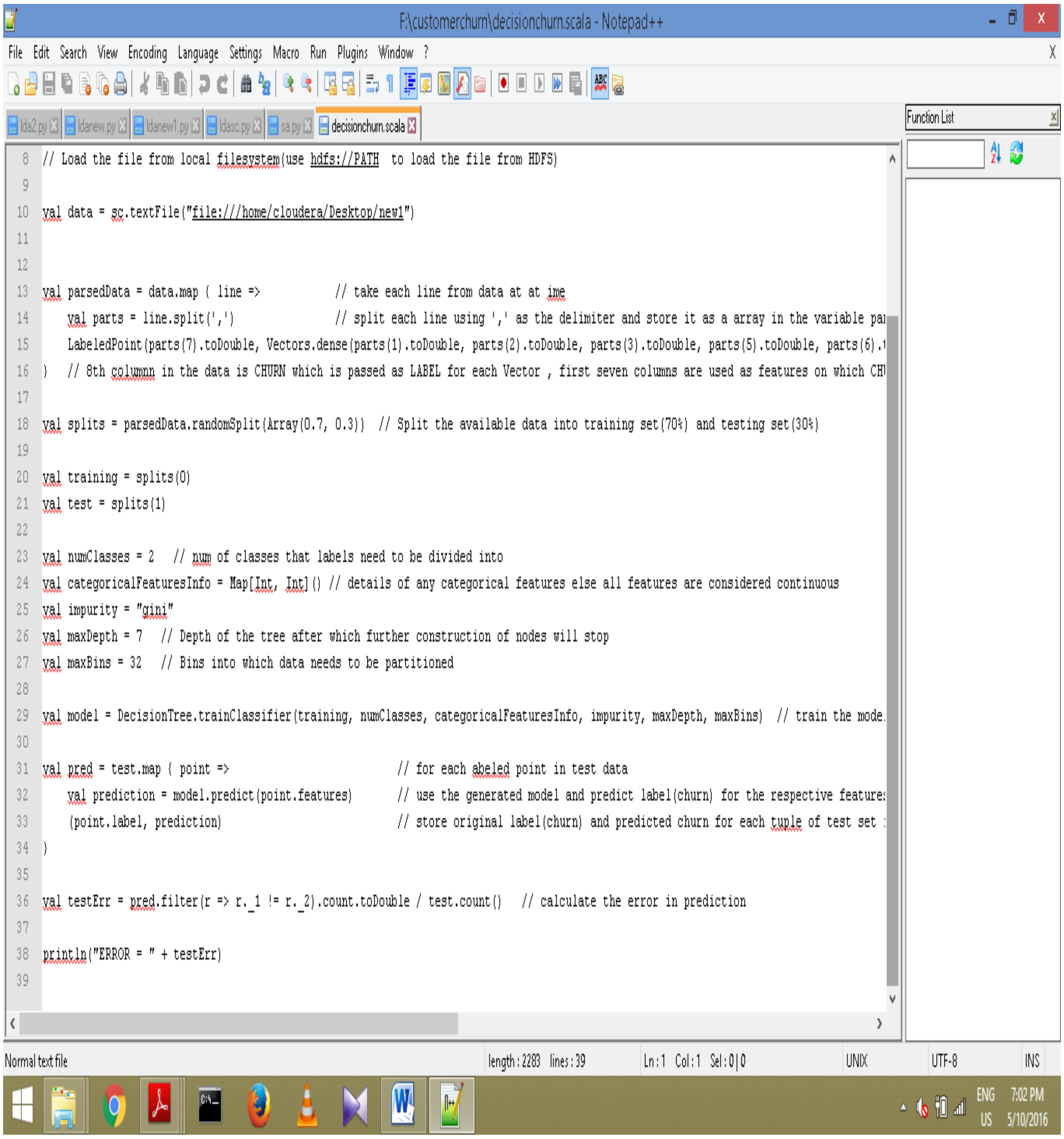
F4 12.2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Account L	VMail Me	Day Mins	Eve Mins	Night Min	Intl Mins	CustServ (	Churn	Int'l Plan	VMail Plar	Day Calls	Day Charg	Eve Calls	Eve Charg	Night Call	Night Cha	Intl Calls	Intl Charg	State	Area Code	Phon
2	128	25	265.1	197.4	244.7	10	1	0	0	1	110	45.07	99	16.78	91	11.01	3	2.7	KS	415	382-4
3	107	26	161.6	195.5	254.4	13.7	1	0	0	1	123	27.47	103	16.62	103	11.45	3	3.7	OH	415	371-7
4	137	0	243.4	121.2	162.6	12.2	0	0	0	0	114	41.38	110	10.3	104	7.32	5	3.29	NJ	415	358-1
5	84	0	299.4	61.9	196.9	6.6	2	0	1	0	71	50.9	88	5.26	89	8.86	7	1.78	OH	408	375-9
6	75	0	166.7	148.3	186.9	10.1	3	0	1	0	113	28.34	122	12.61	121	8.41	3	2.73	OK	415	330-6
7	118	0	223.4	220.6	203.9	6.3	0	0	1	0	98	37.98	101	18.75	118	9.18	6	1.7	AL	510	391-8
8	121	24	218.2	348.5	212.6	7.5	3	0	0	1	88	37.09	108	29.62	118	9.57	7	2.03	MA	510	355-9
9	147	0	157	103.1	211.8	7.1	0	0	1	0	79	26.69	94	8.76	96	9.53	6	1.92	MO	415	329-9
10	117	0	184.5	351.6	215.8	8.7	1	0	0	0	97	31.37	80	29.89	90	9.71	4	2.35	LA	408	335-4
11	141	37	258.6	222	326.4	11.2	0	0	1	1	84	43.96	111	18.87	97	14.69	5	3.02	WV	415	330-8
12	65	0	129.1	228.5	208.8	12.7	4	1	0	0	137	21.95	83	19.42	111	9.4	6	3.43	IN	415	329-6
13	74	0	187.7	163.4	196	9.1	0	0	0	0	127	31.91	148	13.89	94	8.82	5	2.46	RI	415	344-9
14	168	0	128.8	104.9	141.1	11.2	1	0	0	0	96	21.9	71	8.92	128	6.35	2	3.02	IA	408	363-1
15	95	0	156.6	247.6	192.3	12.3	3	0	0	0	88	26.62	75	21.05	115	8.65	5	3.32	MT	510	394-8
16	62	0	120.7	307.2	203	13.1	4	0	0	0	70	20.52	76	26.11	99	9.14	6	3.54	IA	415	366-9
17	161	0	332.9	317.8	160.6	5.4	4	1	0	0	67	56.59	97	27.01	128	7.23	9	1.46	NY	415	351-7
18	85	27	196.4	280.9	89.3	13.8	1	0	0	1	139	33.39	90	23.88	75	4.02	4	3.73	ID	408	350-8
19	93	0	190.7	218.2	129.6	8.1	3	0	0	0	114	32.42	111	18.55	121	5.83	3	2.19	VT	510	386-2
20	76	33	189.7	212.8	165.7	10	1	0	0	1	66	32.25	65	18.09	108	7.46	5	2.7	VA	510	356-2
21	73	0	224.4	159.5	192.8	13	1	0	0	0	90	38.15	88	13.56	74	8.68	2	3.51	TX	415	373-2
22	147	0	155.1	239.7	208.8	10.6	0	0	0	0	117	26.37	93	20.37	133	9.4	4	2.86	FL	415	396-5
23	77	0	62.4	169.9	209.6	5.7	5	1	0	0	89	10.61	121	14.44	64	9.43	6	1.54	CO	408	393-7
24	130	0	183	72.9	181.8	9.5	0	0	0	0	112	31.11	99	6.2	78	8.18	19	2.57	AZ	415	358-1
25	111	0	110.4	137.3	189.6	7.7	2	0	0	0	103	18.77	102	11.67	105	8.53	6	2.08	SC	415	350-2

Fig 4 Dataset Used

5.6.1 CODE : Code for churn analysis using decision tree algorithm in scala(MLLIB)

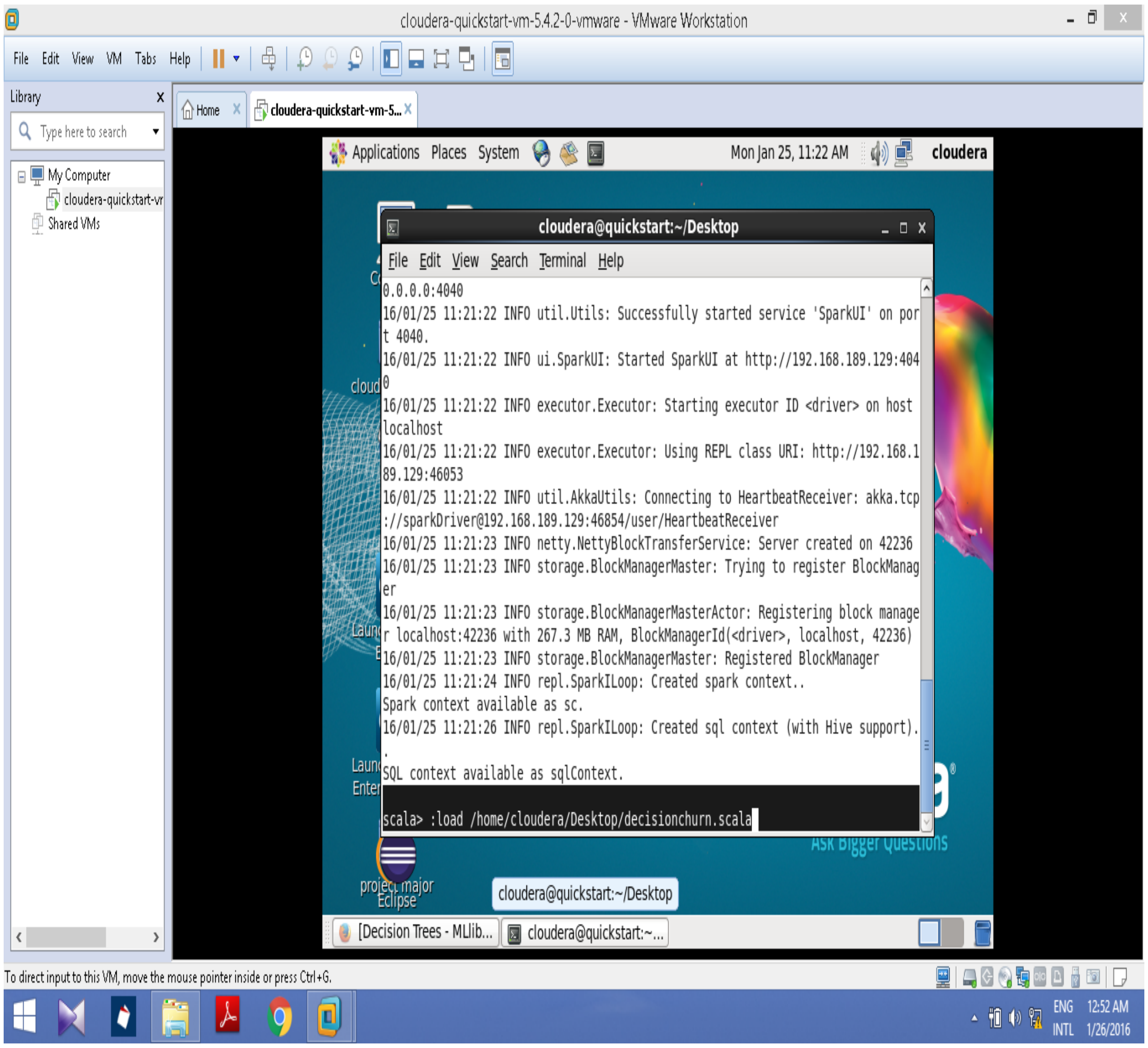

```scala
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils


// Load the file from local filesystem(use hdfs://PATH  to load the file from HDFS)

val data = sc.textFile("file:///home/cloudera/Desktop/new1")


val parsedData = data.map { line =>           // take each line from data at at ime
    val parts = line.split(',')               // split each line using ',' as the delimiter and store it as a array in the variable pa
    LabeledPoint(parts(7).toDouble, Vectors.dense(parts(1).toDouble, parts(2).toDouble, parts(3).toDouble, parts(5).toDouble, parts(6).
}   // 8th columnn in the data is CHURN which is passed as LABEL for each Vector , first seven columns are used as features on which CH

val splits = parsedData.randomSplit(Array(0.7, 0.3))  // Split the available data into training set(70%) and testing set(30%)

val training = splits(0)
val test = splits(1)

val numClasses = 2   // num of classes that labels need to be divided into
val categoricalFeaturesInfo = Map[Int, Int]() // details of any categorical features else all features are considered continuous
val impurity = "gini"
val maxDepth = 7   // Depth of the tree after which further construction of nodes will stop
val maxBins = 32   // Bins into which data needs to be partitioned

val model = DecisionTree.trainClassifier(training, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins)  // train the mode

val pred = test.map { point =>                      // for each abeled point in test data
    val prediction = model.predict(point.features)      // use the generated model and predict label(churn) for the respective feature
    (point.label, prediction)                         // store original label(churn) and predicted churn for each tuple of test set
```



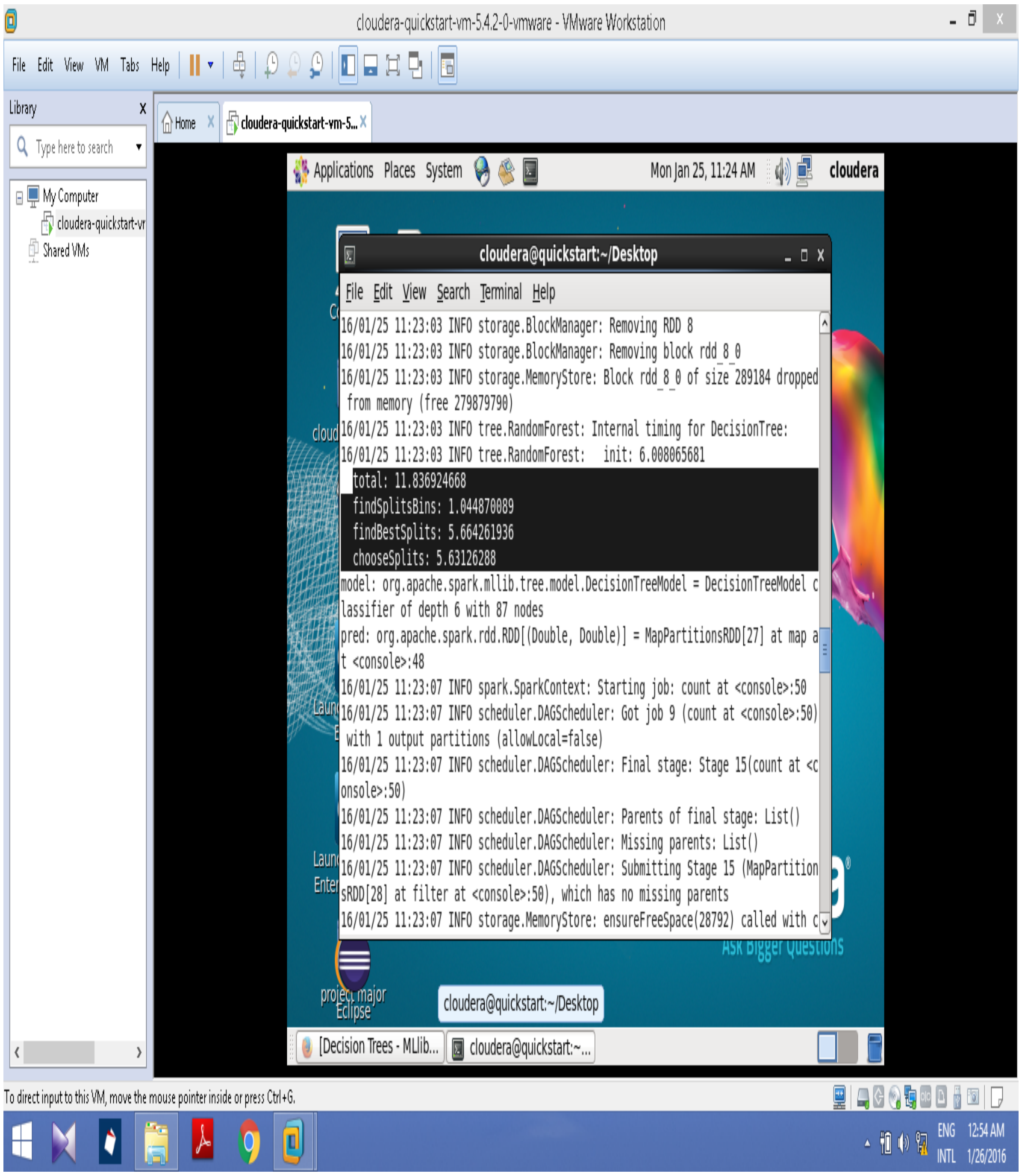
```
// Load the file from local filesystem(use hdfs://PATH  to load the file from HDFS)

val data = sc.textFile("file:///home/cloudera/Desktop/new1")


val parsedData = data.map { line =>           // take each line from data at at ime
    val parts = line.split(',')               // split each line using ',' as the delimiter and store it as a array in the variable pa
    LabeledPoint(parts(7).toDouble, Vectors.dense(parts(1).toDouble, parts(2).toDouble, parts(3).toDouble, parts(5).toDouble, parts(6).
}   // 8th columnn in the data is CHURN which is passed as LABEL for each Vector , first seven columns are used as features on which CH

val splits = parsedData.randomSplit(Array(0.7, 0.3))  // Split the available data into training set(70%) and testing set(30%)

val training = splits(0)
val test = splits(1)

val numClasses = 2   // num of classes that labels need to be divided into
val categoricalFeaturesInfo = Map[Int, Int]() // details of any categorical features else all features are considered continuous
val impurity = "gini"
val maxDepth = 7   // Depth of the tree after which further construction of nodes will stop
val maxBins = 32   // Bins into which data needs to be partitioned

val model = DecisionTree.trainClassifier(training, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins)  // train the mode

val pred = test.map { point =>                        // for each abeled point in test data
    val prediction = model.predict(point.features)    // use the generated model and predict label(churn) for the respective feature
    (point.label, prediction)                         // store original label(churn) and predicted churn for each tuple of test set
}

val testErr = pred.filter(r => r._1 != r._2).count.toDouble / test.count()   // calculate the error in prediction

println("ERROR = " + testErr)
```

Fig 5 decision tree scala code

6.2.3 EXECUTION(SPARK_SHELL) : Execution of algorithm in spark-shell

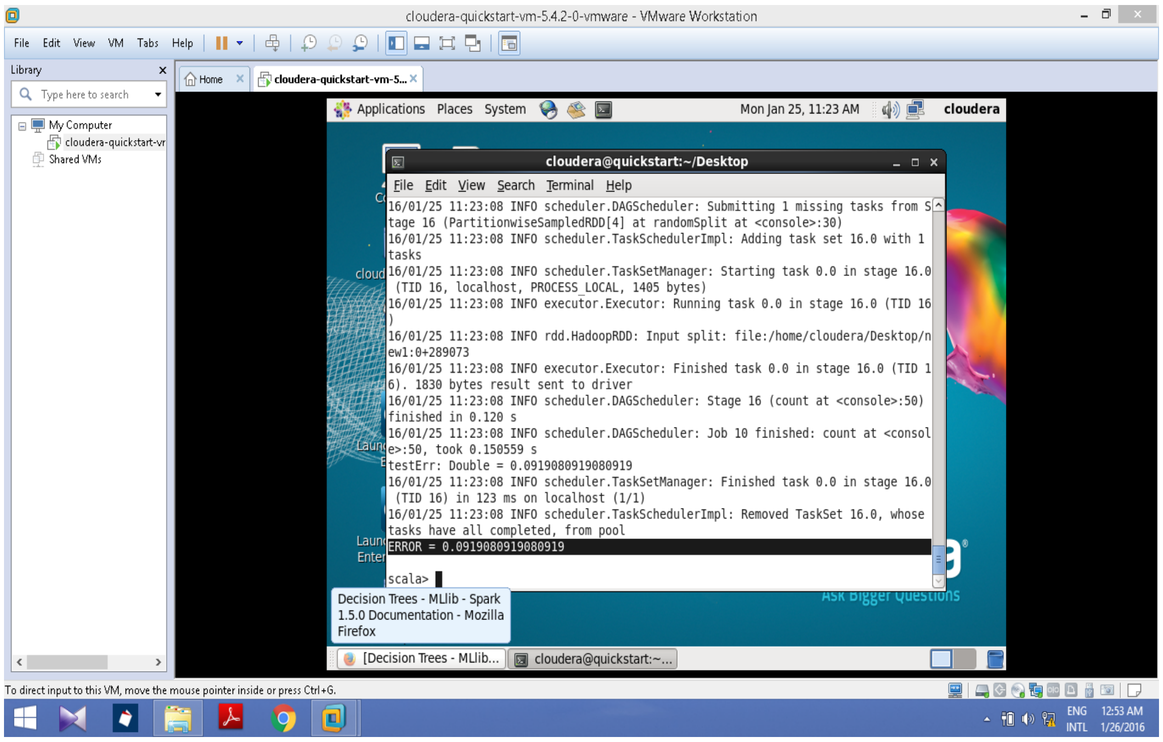

cloudera-quickstart-vm-5.4.2-0-vmware - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
cloudera-quickstart-vr
Shared VMs
Home
cloudera-quickstart-vm-5...
Applications Places System
Mon Jan 25, 11:22 AM
cloudera
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
Ask Bigger Questions
project major
Eclipse
[Decision Trees - MLlib...
cloudera@quickstart:~...
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
ENG INTL
12:52 AM
1/26/2016

```
0.0.0.0:4040
16/01/25 11:21:22 INFO util.Utils: Successfully started service 'SparkUI' on por
t 4040.
16/01/25 11:21:22 INFO ui.SparkUI: Started SparkUI at http://192.168.189.129:404
0
16/01/25 11:21:22 INFO executor.Executor: Starting executor ID <driver> on host
localhost
16/01/25 11:21:22 INFO executor.Executor: Using REPL class URI: http://192.168.1
89.129:46053
16/01/25 11:21:22 INFO util.AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp
://sparkDriver@192.168.189.129:46854/user/HeartbeatReceiver
16/01/25 11:21:23 INFO netty.NettyBlockTransferService: Server created on 42236
16/01/25 11:21:23 INFO storage.BlockManagerMaster: Trying to register BlockManag
er
16/01/25 11:21:23 INFO storage.BlockManagerMasterActor: Registering block manage
r localhost:42236 with 267.3 MB RAM, BlockManagerId(<driver>, localhost, 42236)
16/01/25 11:21:23 INFO storage.BlockManagerMaster: Registered BlockManager
16/01/25 11:21:24 INFO repl.SparkILoop: Created spark context..
Spark context available as sc.
16/01/25 11:21:26 INFO repl.SparkILoop: Created sql context (with Hive support).
.
SQL context available as sqlContext.

scala> :load /home/cloudera/Desktop/decisionchurn.scala
```


cloudera-quickstart-vm-5.4.2-0-vmware - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
cloudera-quickstart-vr
Shared VMs
Home
cloudera-quickstart-vm-5...
Applications Places System
Mon Jan 25, 11:24 AM
cloudera
cloudera@quickstart:~/Desktop
File Edit View Search Terminal Help
Ask Bigger Questions
project major
Eclipse
cloudera@quickstart:~/Desktop
[Decision Trees - MLlib...
cloudera@quickstart:~...
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
ENG
INTL
12:54 AM
1/26/2016

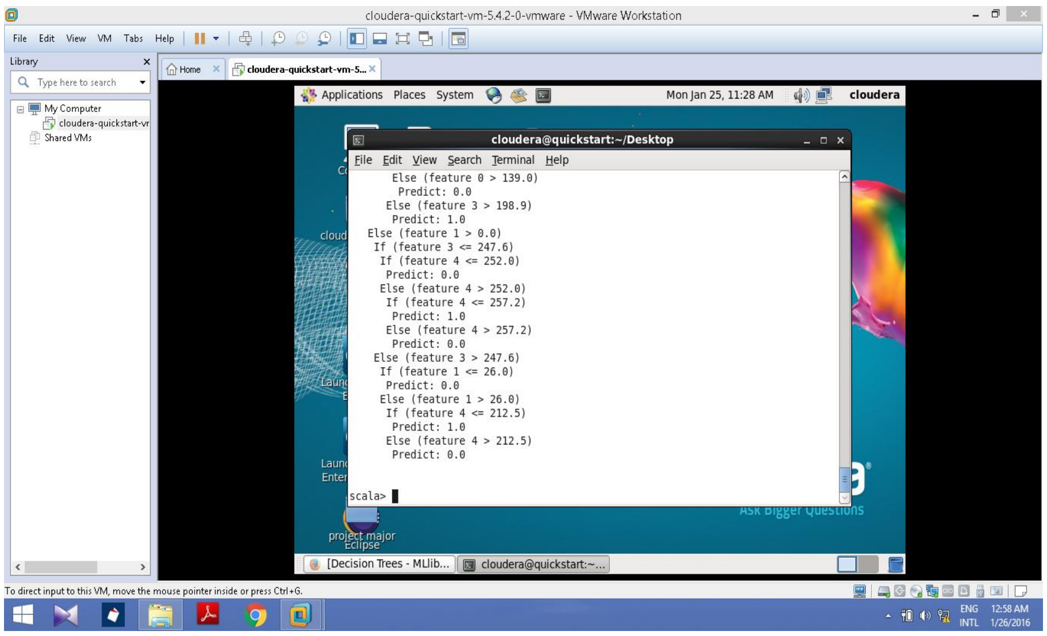
```
16/01/25 11:23:03 INFO storage.BlockManager: Removing RDD 8
16/01/25 11:23:03 INFO storage.BlockManager: Removing block rdd_8_0
16/01/25 11:23:03 INFO storage.MemoryStore: Block rdd_8_0 of size 289184 dropped
 from memory (free 279879790)
16/01/25 11:23:03 INFO tree.RandomForest: Internal timing for DecisionTree:
16/01/25 11:23:03 INFO tree.RandomForest:   init: 6.008065681
  total: 11.836924668
  findSplitsBins: 1.044870089
  findBestSplits: 5.664261936
  chooseSplits: 5.63126288
model: org.apache.spark.mllib.tree.model.DecisionTreeModel = DecisionTreeModel c
lassifier of depth 6 with 87 nodes
pred: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[27] at map a
t <console>:48
16/01/25 11:23:07 INFO spark.SparkContext: Starting job: count at <console>:50
16/01/25 11:23:07 INFO scheduler.DAGScheduler: Got job 9 (count at <console>:50)
 with 1 output partitions (allowLocal=false)
16/01/25 11:23:07 INFO scheduler.DAGScheduler: Final stage: Stage 15(count at <c
onsole>:50)
16/01/25 11:23:07 INFO scheduler.DAGScheduler: Parents of final stage: List()
16/01/25 11:23:07 INFO scheduler.DAGScheduler: Missing parents: List()
16/01/25 11:23:07 INFO scheduler.DAGScheduler: Submitting Stage 15 (MapPartition
sRDD[28] at filter at <console>:50), which has no missing parents
16/01/25 11:23:07 INFO storage.MemoryStore: ensureFreeSpace(28792) called with c
```

6.2.4 ACCURACY :



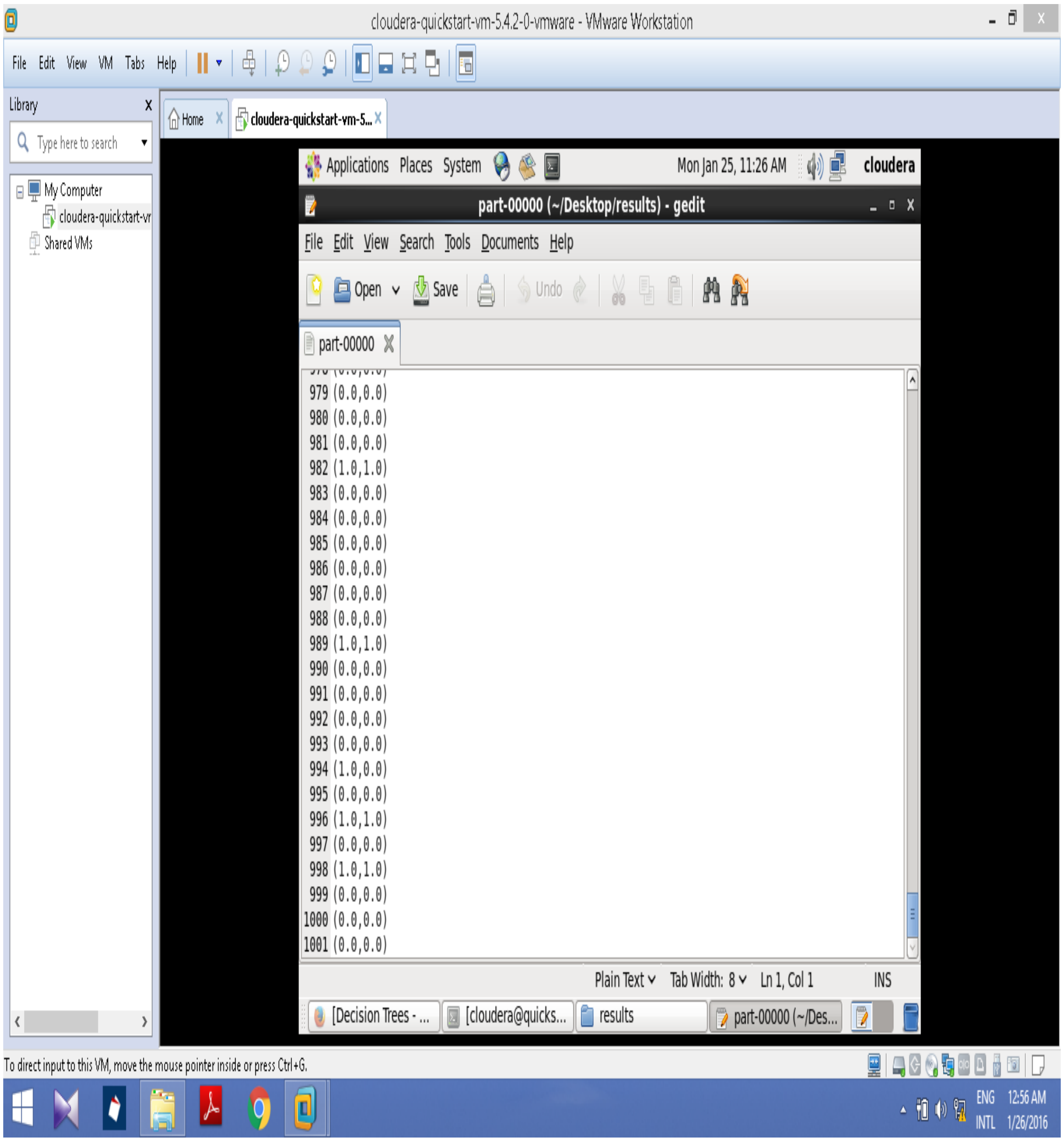
```
16/01/25 11:23:08 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from S
tage 16 (PartitionwiseSampledRDD[4] at randomSplit at <console>:30)
16/01/25 11:23:08 INFO scheduler.TaskSchedulerImpl: Adding task set 16.0 with 1
tasks
16/01/25 11:23:08 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 16.0
 (TID 16, localhost, PROCESS_LOCAL, 1405 bytes)
16/01/25 11:23:08 INFO executor.Executor: Running task 0.0 in stage 16.0 (TID 16
)
16/01/25 11:23:08 INFO rdd.HadoopRDD: Input split: file:/home/cloudera/Desktop/n
ew1:0+289073
16/01/25 11:23:08 INFO executor.Executor: Finished task 0.0 in stage 16.0 (TID 1
6). 1830 bytes result sent to driver
16/01/25 11:23:08 INFO scheduler.DAGScheduler: Stage 16 (count at <console>:50)
finished in 0.120 s
16/01/25 11:23:08 INFO scheduler.DAGScheduler: Job 10 finished: count at <consol
e>:50, took 0.150559 s
testErr: Double = 0.0919080919080919
16/01/25 11:23:08 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 16.0
 (TID 16) in 123 ms on localhost (1/1)
16/01/25 11:23:08 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 16.0, whose
tasks have all completed, from pool
ERROR = 0.0919080919080919

scala>
```

6.2.5TRAINEDMODEL



```
      Else (feature 0 > 139.0)
       Predict: 0.0
     Else (feature 3 > 198.9)
      Predict: 1.0
   Else (feature 1 > 0.0)
    If (feature 3 <= 247.6)
     If (feature 4 <= 252.0)
      Predict: 0.0
     Else (feature 4 > 252.0)
      If (feature 4 <= 257.2)
       Predict: 1.0
      Else (feature 4 > 257.2)
       Predict: 0.0
    Else (feature 3 > 247.6)
     If (feature 1 <= 26.0)
      Predict: 0.0
     Else (feature 1 > 26.0)
      If (feature 4 <= 212.5)
       Predict: 1.0
      Else (feature 4 > 212.5)
       Predict: 0.0

scala>
```

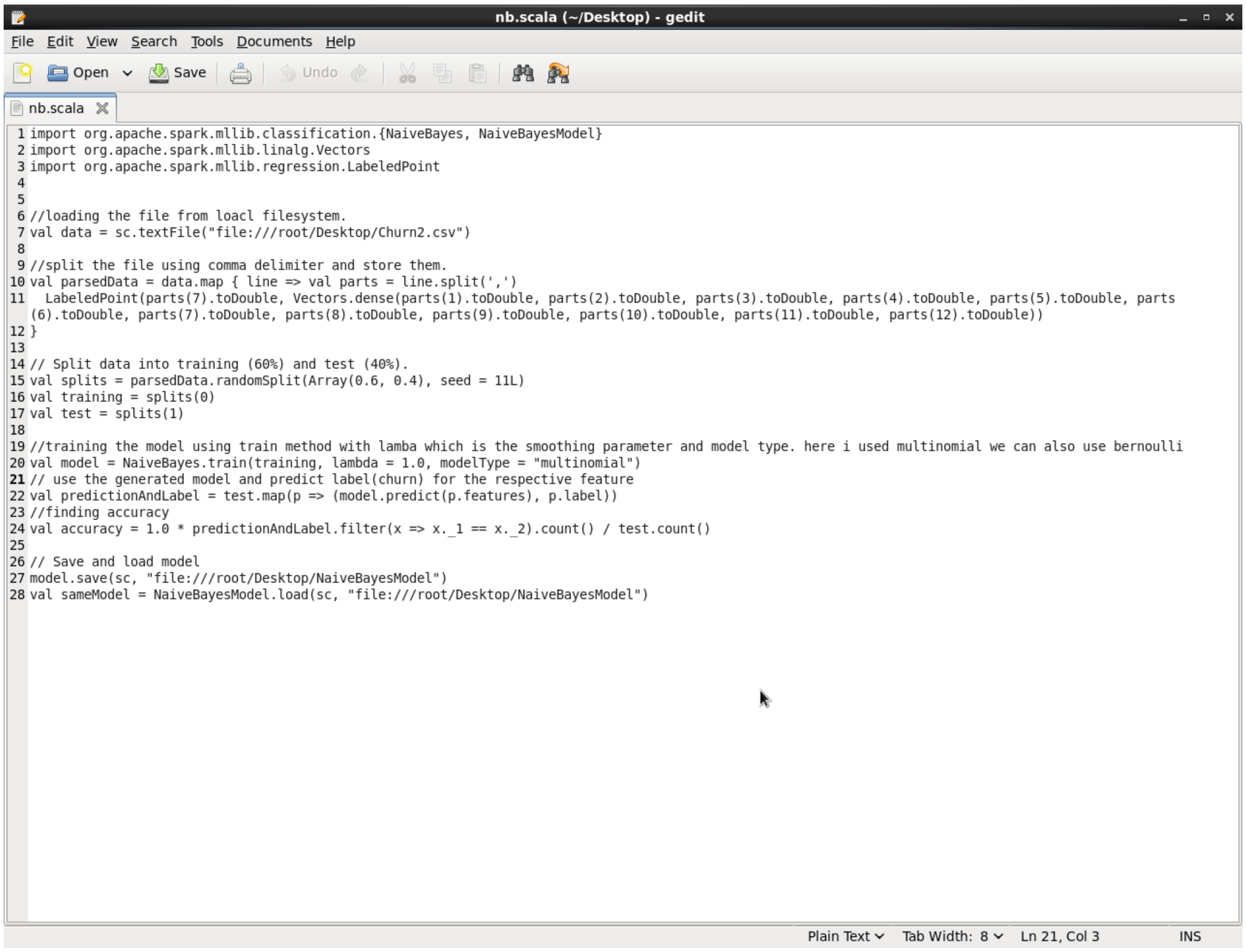
6.2.6 RESULTS :



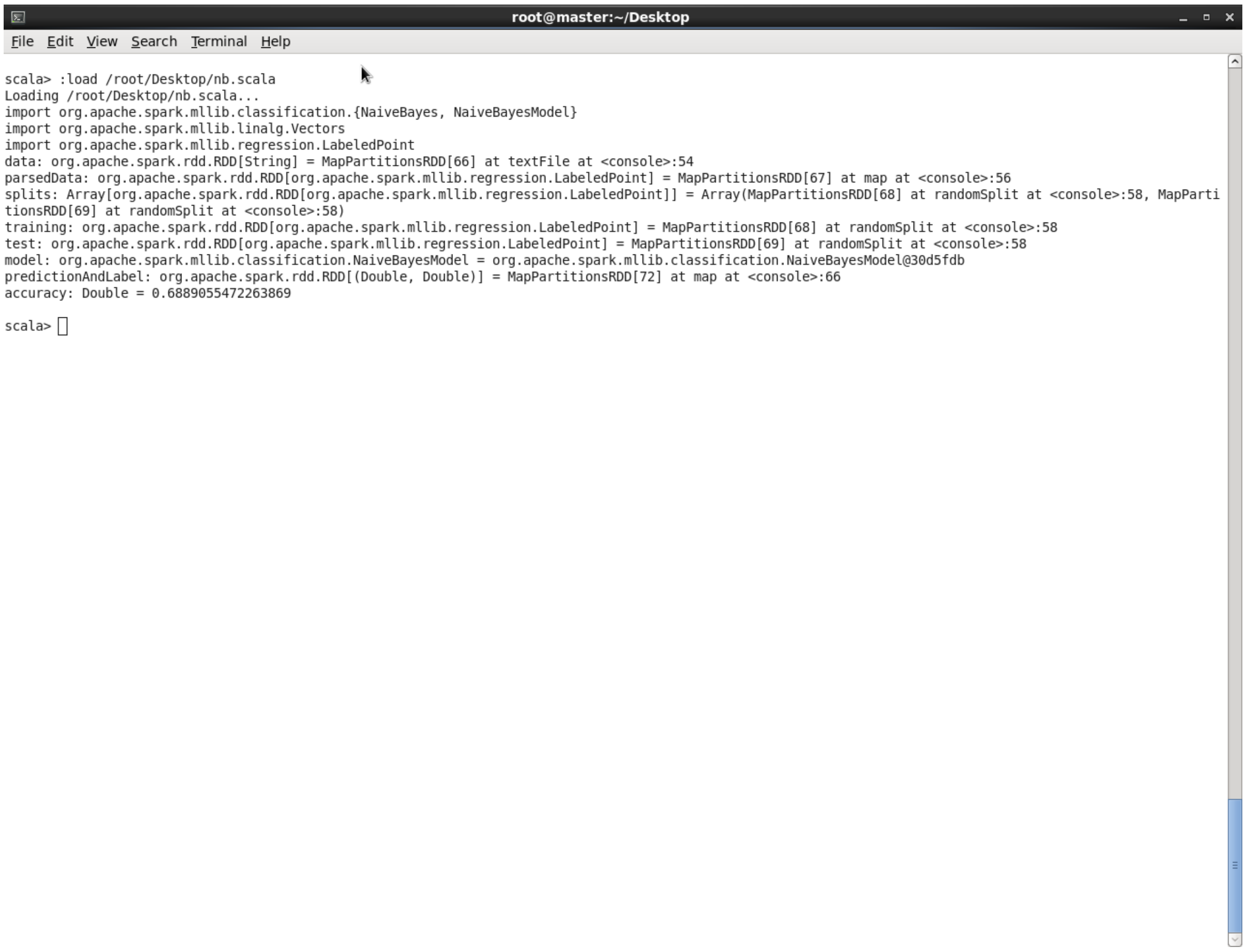
```
979 (0.0,0.0)
980 (0.0,0.0)
981 (0.0,0.0)
982 (1.0,1.0)
983 (0.0,0.0)
984 (0.0,0.0)
985 (0.0,0.0)
986 (0.0,0.0)
987 (0.0,0.0)
988 (0.0,0.0)
989 (1.0,1.0)
990 (0.0,0.0)
991 (0.0,0.0)
992 (0.0,0.0)
993 (0.0,0.0)
994 (1.0,0.0)
995 (0.0,0.0)
996 (1.0,1.0)
997 (0.0,0.0)
998 (1.0,1.0)
999 (0.0,0.0)
1000 (0.0,0.0)
1001 (0.0,0.0)
```

cloudera-quickstart-vm-5.4.2-0-vmware - VMware Workstation
part-00000 (~/Desktop/results) - gedit
Mon Jan 25, 11:26 AM
cloudera
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
[Decision Trees - ...
[cloudera@quicks...
results
part-00000 (~/Des...
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

NAÏVE BAYES SCREENSHOTS


nb.scala (~/Desktop) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
nb.scala
Plain Text Tab Width: 8 Ln 21, Col 3 INS

```
import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint


//loading the file from loacl filesystem.
val data = sc.textFile("file:///root/Desktop/Churn2.csv")

//split the file using comma delimiter and store them.
val parsedData = data.map { line => val parts = line.split(',')
  LabeledPoint(parts(7).toDouble, Vectors.dense(parts(1).toDouble, parts(2).toDouble, parts(3).toDouble, parts(4).toDouble, parts(5).toDouble, parts(6).toDouble, parts(7).toDouble, parts(8).toDouble, parts(9).toDouble, parts(10).toDouble, parts(11).toDouble, parts(12).toDouble))
}

// Split data into training (60%) and test (40%).
val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)
val training = splits(0)
val test = splits(1)

//training the model using train method with lamba which is the smoothing parameter and model type. here i used multinomial we can also use bernoulli
val model = NaiveBayes.train(training, lambda = 1.0, modelType = "multinomial")
// use the generated model and predict label(churn) for the respective feature
val predictionAndLabel = test.map(p => (model.predict(p.features), p.label))
//finding accuracy
val accuracy = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / test.count()

// Save and load model
model.save(sc, "file:///root/Desktop/NaiveBayesModel")
val sameModel = NaiveBayesModel.load(sc, "file:///root/Desktop/NaiveBayesModel")
```



root@master:~/Desktop

File Edit View Search Terminal Help

```
scala> :load /root/Desktop/nb.scala
Loading /root/Desktop/nb.scala...
import org.apache.spark.mllib.classification.{NaiveBayes, NaiveBayesModel}
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
data: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[66] at textFile at <console>:54
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[67] at map at <console>:56
splits: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint]] = Array(MapPartitionsRDD[68] at randomSplit at <console>:58, MapParti
tionsRDD[69] at randomSplit at <console>:58)
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[68] at randomSplit at <console>:58
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[69] at randomSplit at <console>:58
model: org.apache.spark.mllib.classification.NaiveBayesModel = org.apache.spark.mllib.classification.NaiveBayesModel@30d5fdb
predictionAndLabel: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[72] at map at <console>:66
accuracy: Double = 0.6889055472263869

scala> 
```

CHAPTER 7: RESULT ANALYSIS

I used both Naïve Bayes and Decision tree algorithm on the customer churn dataset and in this comparative study it came out that decision tree model has better accuracy then naïve bayes and is more suitable in this to train the dataset.

In Naïve bayes the accuracy comes out to be of 68% while in decision tree the accuracy comes out to be of 89-90%.With this result we can see clearly the decision tree is better approach for our analysis.

CHAPTER 8: CONCLUSION AND FUTURE WORK

In the future work we are focusing on deep data analysis of text, in different areas of application.In near future would be working on analyzing same dataset using different algorithms .such as NLP . The findings of this study will help telecommunications companies understand customer churn risk and customer churn hazard over the time of customer tenure. Overall, this study is helpful in customizing marketing communications and customer treatment programs to optimally time their marketing intervention efforts.

REFERENCES

[1]. http://www.google.com/

[2]. Bo Pang, Lillian Lee, "Opinion Mining and Data analysis".

[3]. http://www.cloudera.com/hadoop-training-thinking-at-scale

[4]. http://developer.yahoo.com/hadoop/tutorial/module1.html

[5]. http://hadoop.apache.org/core/docs/current/api/

[6]. O'reilly, Hadoop: The Definitive Guide by Tom White

[7]. Murphy Choy, Michelle L.F. Cheong, Ma Nang Laik, and Koo Ping Shung, "A data analysis of Singapore Presidential Election 2011 using Twitter data with census correction", Journal CoRR, Vol. abs/1108.5520, 2011

[8]. Dr. Tariq Mahmood, Tasmiyah Iqbal, Farnaz Amin, Wajeeta Lohanna, Atika Mustafa "Mining Twitter Big Data to Predict 2013 Pakistan Election Winner" Department of Computer Science National University of Computer & Emerging Sciences Karachi, Pakistan.

[9]. http://www.ijcsit.com/docs/Volume%205/vol5issue03/ijcsit2014050393.pdf

[10]. http://www.edureka.co/blog/sentiment-analysis-methodology/

[11]. https://www.r-project.org/about.html

[12]. Jason Venner, *Pro Hadoop*, Apress

[13]. Tom White, *Hadoop: The Definitive Guide*, O'REILLY

[14]. Chuck Lam, Hadoop in Action, MANNING

[15]. "Hadoop Tutorial from Yahoo!", Module 7: Managing a Hadoop Cluster

[16]. http://developer.yahoo.com/hadoop/tutorial/module7.html#machines

[17]. K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop distributed file system", in poc. The 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010

[18]. M.M. Anwar, M.F. Zafar, Z. Ahmed. A proposed Preventive Information Security System. IEEE International Conference on Electrical Engineering, April, 2007.

[19]. MacDonald, Neil, 2012, Information Security is Becoming a Big Data Analytic Problem, Gartner, (23 March 2012), DOI= http://www.gartner.com/id=1960615

[20]. Larry Barrett, "Big data analytics: the enterprise's next great security weapon?" February 2014.

[21].http://www.edupristine.com/courses/big-data-hadoop-program/bigdata-hadoop-course/