MASTER THESIS

Massive scale analytics with Stratosphere using R

Supervisor:
Prf. Dr. MARKL, Volker

Author:
LOPEZ PINO, Jose Luis

Advisors:
LEICH, Marcus
Dr. TZOUMAS, Kostas

August 10, 2014

Abstract

R is an extremely popular numerical computer environment, but scientific data processing frequently hits its memory limits. This work presents an innovative approach to overcome these limitations using the Stratosphere/Apache Flink big data platform.

The main contribution of this thesis is the innovative approach to tackle this problem by means of an R package and ready-to-use distributed algorithm. This package not only coordinates the execution of these algorithms but also facilitates data operations that were cumbersome with the already existing packages.

In addition to that, a comprehensive library that includes the most relevant algorithm after a study of the state-of-the-art data mining libraries is presented. In order to guide the design, development and evaluation of this solution, three scenarios are defined and presented, including common mining tasks like clustering, classification and text mining.

Finally, this work investigates the opportunities for parallelization of these complex data processing tasks and evaluates the performance of the solution. As presented in the conclusions outlined in Chapter 5, the tests conducted showed the competitiveness of the solution in terms of performance and scalability, without writing code that notably differs from the tantamount non-parallel programs.

Acknowledgements

I am very grateful to my advisors Marcus Leich and Kostas Tzoumas not only for their feedback but also for the trust and freedom that I have found to take the lead of this work. Their pragmatic view was a key ingredient for the success of this thesis.

This journey would not have been possible without the support of my dear classmates Janani Chakkaradhari, Prateek Gaur, Suryamita Harindrari, Silvia Julinda and Faisal Moeen. I would also like to thank the whole DIMA research group for their support and help from the head of the group Prof. Dr. Volker Markl to the last member. In particular, I'm very thankful to Robert Metzger, Stephan Ewen, Max Heimel and Ufuk Celebi.

Last but not least, I'm very obliged to Dr. Jose Quesada, Pere Ferrera and Andreu Estany, who generously helped me to understand the industry perspective on this problem and discussed with me the different approaches to tackle it.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Chapter 1

Introduction

This chapter contains the necessary **background information** for the understanding of the thesis' main core. Section 1.1 provides an overview of the general problem of widely applicable data analysis and emphasizes the demand for data analytics in large-scale and distributed systems. Section 1.2 gives a brief introduction to R as programming language and numerical computing environment. In Section 1.3, we recall the basic components of the MapReduce model and its contributions to parallel programming. Section 1.4 provides a synopsis of the Stratosphere project. Finally, in Section 1.5 we give a brief description of some other terms and definitions relevant to the thesis.

1.1 Data analysis to the masses

Short presentation of data analysis

Data analysis is usually a complex and iterative process compound of different and heterogeneous tasks like data cleaning, quality measurement, transformation, sampling, statistical techniques and exploratory analysis. Traditional data warehousing and traditional business intelligence are not able to deal with the problems and difficulties of answering all possible questions. Since they do not tap all the benefits of data, the demand for large, flexible and real-time answers is arising. As a consequence, the so-called **deep analytics**[DSB+10] (sophisticated statistical methods) are reaching the mainstream.

Concrete examples

For many years, the data analysis discipline has made sense of data from disparate data sources. Namely, credit card translations, communications, e-commerce, web analytics, spatial data, spatiotemporal data, personal information, and social network have been in the spotlight of the industry. Numerous areas field benefit from the analysis of data sources and enhance their gain. For instance, those data sources previously mentioned are currently being heavily used for fraud detection, recommendation systems, data driven marketing, traffic forecast, data-drive human resources, and trending markets detection.

New sources and new tools

In the era of massively parallel processing, **new data sources** are gaining significance. Some of these sources provide data that was not capitalized on for different reasons, mainly because of the size and the lack of formal structures. Recently, corporations are becoming aware of the value of these new sources like machine-generated and crowd-generated data. The explosion of new data sources and the popularization of data-driven decisions have increased the demand for most accessible data analysis tools and environments. For instance, user-friendly data pipelining tools[BCD$^+$09] and numerical computing environments like R, Matlab or SPSS.

BIG data analysis to the masses

Data analysis and information mining tools have to integrate and assimilate the new analysis techniques arising from the use of **vast amounts of data**. Big data[Zic13] brings new opportunities to the market but also presents unfamiliar challenges. It is necessary to tame the volume, velocity and variety of massive data.

1.2 R

What is R?

R[Wic14] is a domain-specific programming language for statistics strongly inspired by S. R follows the **functional programming** paradigm. In this paradigm, the results produced by a function depend exclusively on the inputs and not on the state of the program. R is not only a programming language but also a **numerical computing environment**. The R programming language in combination with the CRAN packages[Hor12b] has become a very popular toolkit for rich data analysis and modelling and, in particular, for data mining[ZC13] More than 3000 packages available in CRAN make it a very flexible and participative tool[Hor12a]. In addition, these packages include detailed and meaningful documentation that provides very valuable help and incentivizes the use of third-party code.

What can we do with R?

Statisticians, data scientists, data journalists and other professionals who make sense of data have embraced R because it is a very convenient tool for interactive data exploration. Admittedly, the large number of packages available cover a wide range of needs, from social sciences to web technologies. Putting the spotlight on **data analysis**, the repository comprehend packages for data integration, data preparation, data cleaning, data exploration, predictive models, evaluation, data visualization, text mining[MHF08, HNP05], etc.

Memory: the main drawback

The main **shortcoming** of R, is that it can only handle small to medium amounts of data. R has copy-on-modify semantics and hence it might frequently make copies of objects. The package *plyr* implements the split-apply-combine strategy (quite similar to the MapReduce model), avoiding this shortcoming by not making an extra copy of the data in the split step[Wic11]. This limitation differs by far from the position of the database research community, which agrees that nowadays tools should be able to handle very large volumes of data[AAA$^+$14]. We discuss some other options overcome this problem in the Subsection 4.2.

Competitors

R is not one-of-a-kind, some other environments are also powerful tools for data analysis. In particular, **Python** is becoming popular for data-centric tasks thanks to libraries like *pandas*[McK12], *NumPy*[VDWCV11], *statsmodels*[SP10] and *sci-kit*[PVG$^+$11], although it is a general purpose language. More recently, a new performance-oriented domain-specific language called **Julia**[BKSE12] was presented.

Conclusion

In conclusion, most data practitioners prefer numerical computing environments because they provide a wide variety of out-of-the-box **statistical techniques**. Nevertheless, it is not all rosy and R is not suitable for handling vast amounts of data.

1.3 MapReduce and parallel programming

Shared nothing architectures allow clusters to easily scale out, however parallelizing a program among their nodes is a very challenging task. MapReduce aroused as a solution to parallelize computing in shared-nothing commodity clusters. In this section, we will introduce the MapReduce model and Hadoop, the free software project that was born as an open source implementation of this model.

1.3.1 The MapReduce programming model

What is MapReduce?

The MapReduce[DG08] programming model was introduced by Google in 2004. MapReduce was a completely different approach to big data analysis at the time and it has been proven effective in large cluster systems.

This model is based on two second-order functions that are coded by the user: map and reduce (described in Section 1.4). This model **hides the complexity** of some tasks related to fault-tolerance, data-distribution and balancing from the user. In consequence, this model facilitates programmers without any experience in parallel coding to write the **highly scalable programs** and hence process voluminous data sets. This high level of scalability is reached as a result of the decomposition of the problem into a big number of parallel tasks.

Drawbacks

However, MapReduce also has important **drawbacks**. Although the burden of programming parallel application is reduced, some common and very simple tasks do not fit well into the single workflow of this model. In consequence, they are complicated to write using these two second order operators. Moreover, many of the tasks are expensive to perform, the user code is difficult to debug, there is a lack of schema and indices and a lot of network bandwidth might be consumed[LLC+12]. The purpose of the different high level languages (Hive, Pig, etc.) is to address some of those shortcomings.

1.3.2 Hadoop

What is Hadoop?

Hadoop is a platform that runs on clusters of commodity hardware and currently has a three layers architecture: storage, resource management and data processing. The **data processing** layer is a open-source implementation of the Google-MapReduce model. The **storage** layer is the Hadoop Distributed File System (HDFS). It is designed for storing very large files with streaming data access patterns and it is derived from Google File System (GFS). Finally, the **resource management** layer was previously coupled with the MapReduce programming model, but it is now responsible for running jobs in parallel from different programming frameworks.[Vav13]

HDFS in a nutshell

HDFS is a **distributed file system**, it manages the storage across a network of machines. In order to reduce the risk of data loss, the data is redundantly replicated. This also improves data locality. The more copies of the files we store, the more probable that a task is located close to the data required. This is key to the performance of data-intensive computing tasks.

This section includes some text or images of my own work produced for the course Business Intelligence Seminar.

Under these circumstances, Hadoop has become a crucial industry standard for massively parallel data processing[RQRSD13, PRGK14]. Hadoop and some other Big Data platforms (Spark, Stratosphere, etc.) have opened up a new world of opportunities. They enable data-intensive computation tasks like network training[LLM10], natural language processing[Lin08] and clustering[ZMH09] to be performed on vast amounts of data.

Acceptance

1.4 The Stratosphere project

Stratosphere is a multi-purpose platform for **massively parallel computing** in general and Big Data Analytics in particular[ABE$^+$14]. It is designed to efficiently execute data-intensive tasks on clusters and specially iterative algorithms. Stratosphere presents a database-inspired approach to data-intensive computing, clearly shown in its query optimizer. Stratosphere is an actively developed open-source big data platform but also a research project in the Information Management field. Stratosphere is part of ongoing research programmes conducted in the areas of massively parallel data processing, streaming processing[LWK13], data-parallel iterative algorithms on large datasets[ETKM12], fault tolerance[SETM13], etc.

What is stratosphere?

Stratosphere is mainly an enhanced replacement of **Hadoop** runtime engine including top level abstractions and interfaces for writing massively parallel programs. Stratosphere is compatible with many Hadoop components or dependant to them. Namely, it is compatible with Hadoop input formats and it integrates with YARN[Vav13]. This means that Stratosphere and Hadoop are able to use the same input files and share the same resources in a cluster. Furthermore, Stratosphere uses HDFS (Hadoop Distributed Filesystem) for data storage. Stratosphere is not able currently to execute MapReduce jobs written for Hadoop. Nevertheless, it is extremely easy to rewrite them using Stratosphere's operators since they are a superset of Hadoop's operators. Complete compatibility with Hadoop's MapReduce is currently one of the nice-to-have requirements of the project.

Relation with Hadoop

Currently, one of the main goals of the project is to reduce the learning curve for its users. In consequence, new interfaces and top level abstraction have been introduced. Namely, it is possible to write PACT programs using either a Java or a Scala API. These interfaces avoid writing a big amount of boilerplate code for tasks like iterations, lowering the entry barrier to the platform. The main touchpoint between Stratosphere and this work is the Stratosphere client, which is the component in charge of submitting jobs

Relation of Stratosphere with the problem to the cluster. The Stratosphere client depends on several components like Hadoop, Jersey and Apache HttpComponents. The detailed dependency tree can be found in Appendix C.

1.4.1 PACT dataflow model

What is PACT?

PACT[AEH+11] is an extension of the MapReduce programming model also inspired by functional programming and, in consequence, Stratosphere is considered a large-scale data-processing system of the MapReduce family[SLF13, SG14]. The main impediments of the MapReduce model are the fixed dataflow and the limitation to only two simple second-order functions, while Stratosphere's flexible data flow system fits more appropriately graph analysis and machine learning problems[EST+13]. In addition, the PACT model includes new operators (called contracts) in order to perform those analyses easier and more efficiently:

- Map: it is applied in parallel to each key-value pair.
- Reduce: uses as input a key and a set of values related to this key and it might also produce a set of values, but commonly it emits only one or zero values as output.
- Cross: performs the Cartesian product over the input sets.
- CoGroup: groups all the pairs with the same key and process them with an user-defined function.
- Match: also matches key/value pairs from the input data, but pairs with the same key might be processed separately by the user function.

Evolution of the model

The new interface of the project, accepted into Apache Incubation[DPC+07] under the name of Apache Flink[Mar14], now offers to the user a new interface oriented to datasets and adds new operations that perform transformations to these datasets. For instance new operations like aggregations, filtering, distinct and union are now available, increasing the expressive power of the model.

1.5 Basic terms and definitions

During the development of this work, we have found many expected and unexpected terms that should be introduced to people who are not familiar with this terminology, in order to allow the reader to follow more easily the content of this document. In some other cases, we have included terms in this glossary to avoid ambiguity or misunderstanding.

- **KDD** stands for Knowledge Discovery from Data, Knowledge Discovery from Data Mining or, in some cases, Knowledge Discovery from Databases. Fayyad et al. [FPSS96a, FPSS96b, FPSS96c] summarized the KDD process[KM06] in nine steps: understanding the domain and the goals, creating the target source, cleaning and processing the source, data reduction and projection, choosing a data mining method, choosing the data mining algorithm, mining the data, interpretation of the patterns and using the discovered knowledge.

- A **shared-nothing architecture** is that one where the nodes share neither memory nor disk[Sto86]. Shared-nothing architecture are characterised not only by a high scalability in terms of number of nodes but also for a high availability and the possibility of creating clusters with nodes that are far away from each other. Since the there is no common central memory, the number of messages that the nodes have to exchange is bigger and also, because the disk space is not shared, involves challenges in terms of storage.

- **Data mining** consists in improving decisions by using historical data[Mit97] and, for this purpose, machine learning algorithms play a key role in many cases. **Machine learning** is a subfield of Artificial Intelligence that studies how computer can automatically learn knowledge, providing the technical basis for data mining[WF05].

- **Data frames** are lists of vectors and they are the most common way of manipulating data in R[Wic14]. The rows of the data frames are called **observations** and the columns are called **variables**, although we will normally refer to them as **instances** and **dimensions** respectively.

- In this work we understand as **distributed file system** any file system that stores the information in different nodes but behaves as a single one. This is independent from the fact that a distributed operating system is used, there is a single point of failure, all nodes keep replicates of the data or all nodes store all the data.

- A **cluster** is a group of nodes connected between them that can communicate although they do not share physical resources. We understand as a **node** of a cluster every machine either physical or virtual that has its own independent disk and processing unit. The nodes might or might not be physically located together.

- A **scenario**, as it is understood in software engineering, is a scene that illustrates the interplay between the user and a system or between different systems. Scenarios are typically represented by a path or by a number of algorithmic steps.

- In Hadoop and Stratosphere, a **job** is program written using the group of operators facilitated by the model and the corresponding arguments to execute this program. Each job is composed by one or more **tasks**, that are the minimum unit of work that can scheduled to run in a node.

- In this document, we will constantly mention the term analytics, and in this work we understand as analytics what Sudipto et al.[DSB+10] called *deep analytics*: sophisticated statistical methods like linear models, clustering or classification that frequently are used to extract knowledge from the data.

- **Scaling up** consists in increasing the resources of the machines, for instance increasing the processing power, storage capacity or the memory size. **Scaling out**, however, refers to increasing the number of nodes. Scaling out is, in normal conditions, more expensive and make programs that run on these architectures more complex. However it is convenient when the limit has been reached scaling-up or scaling-up becomes too expensive.

Chapter 2

A new approach for massive scale analytics using R

In the introductory chapter, we have explained the demand for data analytics in large-scale and distributed systems (Section 1.1). We have pointed out how R covers this demand for data analytics in small amounts of data but fails in large-scale level (Section 1.2). After that, we have disclosed the insights of the MapReduce model (Section 1.3) and the Stratosphere projects. Namely, we have seen the expressivity shortcomings of the MapReduce model and mentioned some efforts of the Stratosphere project to make the platform more accessible to data analysts. In this chapter, we are going to do an analysis of the requirements of our solution and present the goals behind it.

We know the problem, now we try to find a solution

2.1 Motivating problems

In this section we present some examples that underline the usefulness and highlight the advantages of our proposal. At the same time, our solution is driven by the goal of efficiently execute these examples on vast amounts of data. For this purpose, we present two scenarios related to the most typical data mining problems: clustering and classification. In addition to that, we also present a problem related to text mining and finding popular terms.

2.1.1 Clustering with K-Means

The first example that we are going to present is focused on a very well-known machine learning task: clustering. Clustering consists in **grouping** elements in accordance with the **similarity** between them. In this problem, the structure of the data is unknown and hence clustering is considered a problem of unsupervised learning. Since the given data is unlabelled, we used as a reward of the algorithm the distance to the cluster centre. The same criterion will be used to evaluate the solution after the clustering.

What is the problem that it solves?

As a starting point, we have selected a dataset that comes from the Kaggle Ford Challenge. The actual goal of this competition was to classify the dataset in order to detect whether the driver of the car was alerted or not. However, we use this dataset for illustration purpose of data processing on a **real dataset**. Kaggle is a platform for machine learning competition with the aim of promoting data analysis on real life cases. These competitions have resulted in significant research contributions[NSR11, HAMA$^+$11, BTH13]. Among the different variables available, we have selected two numeric variables (P7 and V2). We have selected numeric variables because they avoid the complexity of defining the distance between them. We have selected only two in order to plot the data observations as 2D data points.

The data used

The process here described is summarized in Algorithm 1. We assume that there is **no preprocessing** involved in this use case. Thus, after loading the data in the R environment the next step would be to cluster the data. We run the algorithm several times using different number of clusters in order to make comparison between them. Once the data is clustered, we sample the data to extract some points and **compute the distance** between them. The data sampled is the **same data** that we have clustered before because it is not necessary to make a distinction between training and testing data since the **Dunn index** is an internal evaluation method[RAAQ11, BP98]. The Dunn indices computed are shown in a bar plot in order to let the user decide which one is the one desired according to his criteria. Once the user has decided in favour of one of the cluster solutions, the data points are plot using a different colour for each cluster for **visual inspection** purposes. Finally, the cluster information is appended to the original file.

Description of the scenario

An example code to do this task in R in a single machine is available in Listing 2.1. Before the data is clustered, it is loaded into memory and stored in a **data frame**. We stick to the default implementation of k-means provided by the *stats* library, which is shipped as one of the core components of R. The Hartigan-Wong k-means[HW79] method is used, since

Solving this problem with R

it is the default method of the library. Once the data is clustered, the data is sampled and the distances between the points of the sample calculated. These computed distances are used as an input of the *cluster.stats* method of the *fpc* package, which stands for fix points clustering. After deciding the best solution to solve our problem, the data frame including the clusters is written back in **secondary memory**.

Benefits of our approach

R successfully covers these tasks for **small amounts of data**. Even for medium datasets, it is likely to consume all the memory available and crashes the interpreter when clustering or computing distances. The benefits of our approach to these kinds of problems using the Stratosphere platform are evident. Current interfaces available do not allow the user to integrate the use of the cluster with single machine tasks like data visualization and processing of small portions of data. Hence, with this example we illustrate the convenience and even need of an interactive interpreter that integrates the best of two worlds: the scalability of Stratosphere and the flexibility of R in data processing and data visualization.

2.1.2 Classification with Naive Bayes

What is the problem that it solves?

In this example, we are going to cover another very popular machine learning task: classification. Classification consists in predicting the **category** of an observation based on the categories and instances present in labelled training data. Since the training data is **labelled**, classification is considered a supervised learning task. The task is composed of two very differentiated parts. In the first part, the algorithm **learns** from the training dataset in order to recognise a pattern in the data and constructs a model as a result of it. In the second part, the category of unlabelled instances is **predicted** based on the constructed model. This second part is frequently called scoring, since linear methods assign to each unlabelled instance a score that represents the affinity of the instance to belong to one of the classes. Classification methods have been intensively used to solve problems like spam detection[BB08], medical diagnostics[Bra97a], text categorization[Seb02] and internet traffic inspection[NA08].

The data used

For this scenario, we have used a dataset which contains the votes of the U.S. House of Representatives Congressmen on 16 key votes[BL14]. This dataset has been selected due to it offering a good amount of categorical variables which, at first sight, appear to contribute to the classification independently to the presence or absence of other variables. We have chosen categor-

Listing 2.1: Clustering with K-Means using R

```
# 1 Read only two columns of the data
sm_mydataframe <- read.csv("fordTrain.csv")

# 2 K-Means to cluster
sm_clustered2 <- kmeans(sm_mydataframe, 2)
sm_clustered3 <- kmeans(sm_mydataframe, 3)
sm_clustered4 <- kmeans(sm_mydataframe, 4)


# 3 Sample data
sm_samplerows <- sample(nrow(sm_mydataframe),100)
sm_datasample <- sm_mydataframe[sm_samplerows,]
sm_sampleclustered2 <- sm_clustered2$cluster[sm_samplerows]
sm_sampleclustered3 <- sm_clustered3$cluster[sm_samplerows]
sm_sampleclustered4 <- sm_clustered4$cluster[sm_samplerows]


# 4 Compute the distances between points
sm_d <- dist(sm_datasample, method = "euclidean")

# 5 Evaluate the clustering
library(fpc)
sm_dunn <- c(
        cluster.stats(sm_d, sm_sampleclustered2)$dunn2,
        cluster.stats(sm_d, sm_sampleclustered3)$dunn2,
        cluster.stats(sm_d, sm_sampleclustered4)$dunn2
)

# 6 Plot dunn indexes
barplot(sm_dunn, names.arg=c(
        "k=2", "k=3", "k=4"
))

# 7 Decide which solution is the best one
sm_bestsample <- sm_sampleclustered2
sm_bestclustered <- sm_clustered2

# 8 Plot to visualize the best solution
plot(sm_datasample$P7, sm_datasample$V2, col=sm_bestsample)

# 9 Append cluster assignment
sm_finalframe <- data.frame(sm_mydataframe, sm_bestclustered$cluster)
write.csv(sm_finalframe, "fordTrainOUT.csv")
```

ical variables because quantitative attributes are usually discretized before applying Naive Bayes as some other classification methods[YW02, YW09]. The use of categorical variables hence avoids the prepossessing and loss of information due to the discretization. Due to the assumption of independence of the Naive Bayes method; if we had chosen variables with strong dependencies on each other, it would have led to poor classification performance.

Description of the scenario

The process here described is summarized in Algorithm 2. Initially, two files are read as input from the file system. The first one contains the training dataset and it is used to create a model using the Naive Bayes classifier. This process of training basically consists in counting the number of instances that belong to each class for every possible value of the variables. The second one is the unlabelled dataset and it is used to predict the category of its instances. This prediction is done in accordance with the probability model constructed based on the counting described before.

Benefits of our approach

We propose the training of the dataset using the R environment and the prediction or scoring using the data cluster facilities as a solution to this problem. As we have mentioned before in Section 1.2, many researchers use R due to the cutting-edge functionalities, including mining models from the data. Very frequently, the size of the labelled samples is usually limited[CFB08]. Furthermore, new techniques are able to learn a new category using a very limited number of examples in some fields like computer vision[FFFP07, FFFP03]. In consequence, a scenario where the training data fits in main memory, but it is necessary to store the unlabelled data in a cluster seems very probable. Under this scenario, our solution would be very beneficial.

2.1.3 Finding frequent terms

Overview

In this subsection, we present an scenario with the goal of finding relevant in large corpus and see how they are semantically related. The algorithm is very similar to a word count including a step for removing irrelevant words (stop words) and another step for getting rid of those words that are not relevant (frequent) enough. Finally, we will find similarities among those popular words. The different steps of the scenario can be found in Algorithm 3.

Implementation in R

As it can be observed in Listing 2.3, the implementation in R is quite straight forward using the *qdap* package[LS08]. This package has been used for social media mining[HD14] and other text mining applications[JBB14].

Algorithm 1: Clustering with K-Means

```
  Input: inputFilePath, outputFilePath
1 Read $dataPoints$ from $inputFilepath$.
2 $clusters \leftarrow$ cluster $dataPoints$ using different number of clusters.
3 $dataSample \leftarrow$ take a sample of $dataPoints$.
4 $d \leftarrow$ Euclidean distance matrix of $dataSample$.
5 $dunn \leftarrow$ Compute the Dunn indices of $clusters$ using $d$.
6 Plot the different Dunn indices as a bar chart.
7 $best \leftarrow$ choose the solution with maximum Dunn index.
8 Plot $dataSample$ visualizing each cluster of $best$ in a different colour.
9 $outputFilePath \leftarrow$ append $best$ to $dataPoints$.
```

Algorithm 2: Classification with Naive Bayes

```
  Input: inputTraining, inputUnlabelled
  Output: predictions
1 Read $trainingInstances$ from $inputTraining$.
2 Read $unlabelledInstances$ from $inputUnlabelled$.
3 $model \leftarrow$ create a Naive Bayes model based on $trainingInstances$.
4 $predictions \leftarrow$ predict the classes for $unlabelledInstances$ using
  $model$.
```

Listing 2.2: Classification with Naive Bayes using R

```
# Required library
install.packages('e1071')
library(e1071)

# Load the required datasets
dataset_train <- read.csv("train.csv")
dataset_classify <- read.csv("unlabelled.csv")

# Create the model
model <- naiveBayes(dataset_train, dataset_classify)

# Predict the class
prediction <- predict(model, dataset_classify)

# Write to disk
finalframe <- data.frame(dataset_classify, prediction)
write.csv(finalframe, "output.csv")
```

This package can handle the selection of most frequent terms and stop-wording in a single step and hence we will expectedly reproduce the same behaviour with a PACT program. In this scenario, we have opted for writing a more idiomatic code in R making use of the vectorization and the *apply* function.

Alternatives for semantic relatedness

In our case, we are going to use synonyms lookup offered by the *qdap*, giving consistency to the solution, since we are using the same package to find the most frequent terms. Nevertheless, there are many ways of finding synonyms or similar terms, from traditional dictionaries to lexical databases like WordNet[Mil95, MBF$^+$90, Fel98]. Moreover, many different approaches can be used to measure the similarity between two text strings, which can be grouped into term-based and character-based[GF13]. Techniques for semantic relatedness have been proven effective for instance for querying electronic health records[PPM$^+$11].

Benefits of this scenario

Social media have opened up new opportunities for text analysis and opinion mining. Among other purposes, finding frequent terms is a helpful technique for problems like frequent term distribution and dataset profiling[DRSG04]. However, the significant amount of data generated in social media clearly hits the limits of already existing text mining tools[MBR12]. On contrary to those approaches[MAEA05] that give an approximate solution to the Top K problem using data streams, we propose here a exact batch-oriented solution.

Algorithm 3: Most frequent terms

```
Input: inputFile
1 $allWords$ ← Read all the words from $inputFile$.
2 $interestingWords$ ← Remove all the stop words from $allWords$.
3 $frequentWords$ ← Choose the most frequent words in
  $interestingWords$.
4 foreach $word$ in $frequentWords$ do
5     $synonyms$ ← Find all the synonyms of $word$.
6     Find the intersection between $synonyms$ and $words$
```

2.2 The complexity of writing massively parallel programs

MR/Stratosphere doesn't fit the needs of the majority

There is no doubt that the MapReduce and the consequent programming models are a major step forward in parallel programming. As we have mentioned before, they have provided a higher level of abstraction and hence greatly facilitated the implementation of data-intensive tasks. Nevertheless, writing parallel programs is a cumbersome and onerous process that is not appealing for the majority of the data practitioners, as we have mentioned before. In this section we are going to expand this idea, including examples of parallel programs and explaining its implications. Furthermore, the Beckman report[AAA$^+$14], which discuss the state of database research, remarks the need of single tools that facilitates the end-to-end processing and understanding of the data, from a small amount of data up to very large volumes.

Evidences of the problem

First, it is necessary to put the spotlight on the targeted users of the solution. Data scientists is an ambiguous term that covers many different job positions and roles. A survey[HMV13] conducted by Harris et al. showed that the largest group of respondents was called Data Researchers and they are the most skilled group in Statistics, including data manipulation with R. At the same time, this group was the least skilled group in Big Data systems and implementation of machine learning algorithm. This is no wonder, since their programming skills are also inferior. Similarly, Kandel et al.[KPHH12] conducted an interview study and also detected that those user who perform most of their analyses with the R package were not able to write code that runs at scale. Moreover, the article also highlights the importance of using a the same environment to facilitate data visualization and analysis at the same time.

Conclusions of these results

These results show the big data skills gap in professionals that work with statistical computing environments. This gap evidences that neither the proposed high level languages nor the numerous solutions (see Chapter 4) for bringing large-scale analytics to the mass audience (and the the R environment in particular) were resounding successful. As stated by Markl[Mar14], it is impossible to educate the ever-increasing number of data scientist in all the required skills to extract knowledge from big amounts of data. Hence, these results uphold the idea that the entry barrier to parallel computing should be lowered for these users.

Although Stratosphere offers a more expressive interface, writing a parallel program is still not a trivial job. The implementation of an iterative KMeans algorithm described in Figure 1 shows that it is necessary to define several operations with their different input and output structures. In this first example, we can observe how Stratosphere benefits from the new features added (which were described in Section 1.4) like dataset broadcasting and the additional operator GroupBy, but it is not always the case.

Explained with an example

Figure 2 shows a simple program that outputs the most popular terms found in the input document. This is a simple case as we do not have to broadcast any dataset to all the nodes and after the tokenizing the data structure reminds the same. However, it is necessary to three additional classes (two for *flatmaps* and one for the *reducegroup*), resulting in extremely verbose code although the logic enclosed in this code is not very complex.

Even simpler tasks

In order to illustrate how the code looks, we have chosen two very different examples. First, at it can be observed in Listing 2.4 how Stratosphere makes easy some data transformations like simple aggregations (sum, minimum and maximum). However, it is necessary to define a method and manipulate a collector in order to perform some other simple tasks like keeping the most frequent element (Listing 2.5).

Some examples with code

2.3 Our approach and the KDD process

In this section, we would like to define the tasks within the KDD process (defined in 1.5) that we are going to cover at higher level of detail. We have mentioned that R has became a very popular numerical computing environment for scientific data processing. Nowadays, collecting and analysing data is an essential part of the scientific method but also of the day-to-day tasks of many companies.

Why this section?

The target dataset can be created in R using a classical JDBC connector to the database (*RJDBC*) or in a variety of ways to query heterogeneous data sources. For instance, the *rattle* package[Wil11, Wil09] allows the user to load data from CSV files or any ODBC connector. SQL has became a de-facto standard to filter only the data of our interest or to take a sample of the data. It is also possible to use SQL to query a CSV file or an R data frame with the *sqldf* package. Structured documents can also be queried with *RXQuery*. It is possible to compute aggregations either using the SQL language or once the data is loaded in the R environment with the *aggregate* function. In conclusion, although we need some basic functionalities to load data into data frames and to sample data, query languages already cover this

R and data extraction

Listing 2.3: Frequent terms using R

```
install.packages("qdap")

# Load the data
fileName <- 'inputCorpus.txt'
allwords = readChar(fileName, file.info(fileName)$size)

# Select the most frequent elements
library(qdap)
frequency <- freq_terms(allwords, top=1000)
words <- frequency[,"WORD"]

# Find coincidences based on synonyms
lapply(words, function(x) intersect( synonyms(x, return.list=FALSE), words)
     )
```

Listing 2.4: Aggregation with Stratosphere

```
DataSet<Tuple2<String, Integer>> aggregated = inputDataset.groupBy(1).sum
    (0)
```

Listing 2.5: Keep frequent items with Stratosphere

```
public void reduce(Iterator<Tuple2<String, Integer>> in,  Collector<Tuple2<
    String, Integer>> out) throws Exception {

        int minimum = 0;

        while( in.hasNext()){

                Tuple2<String, Integer> currentTuple = in.next();

                if( currentTuple.f1 > minimum ){

                        minimum = addElement( currentTuple );

                }


        }

        for( Tuple2<String, Integer> tuple : mostFrequent){
                out.collect(tuple);
        }
}
```

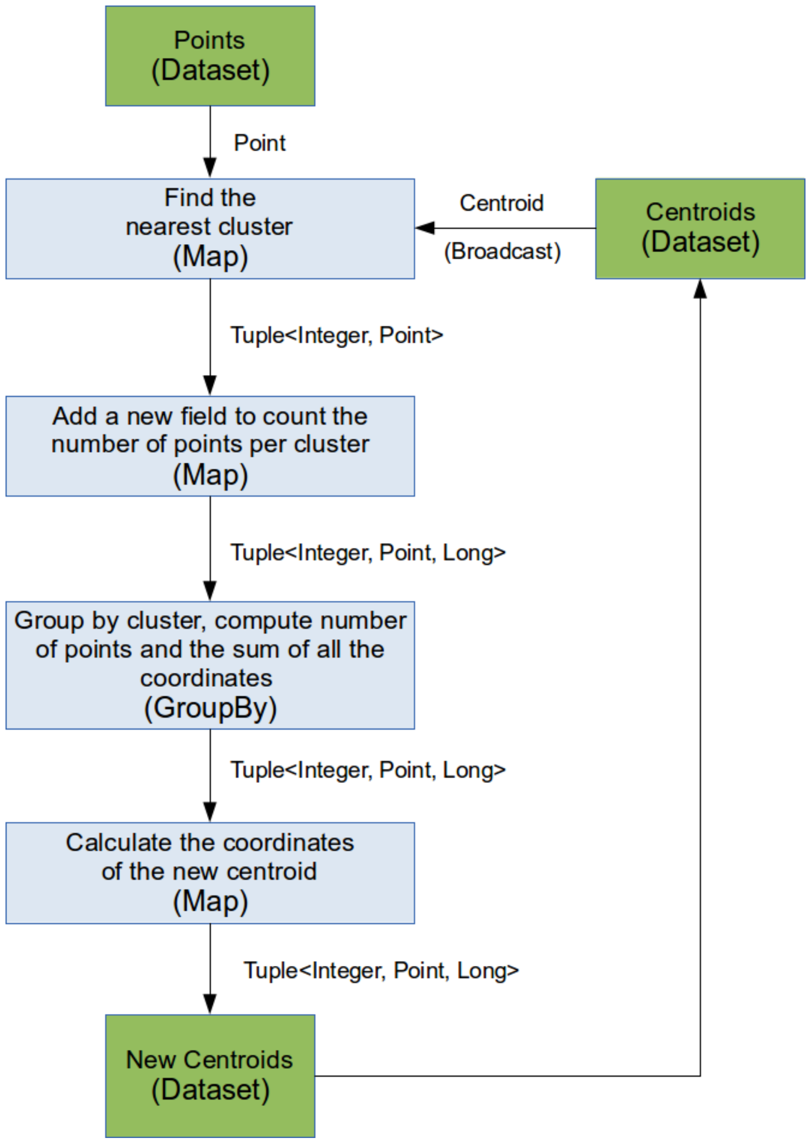
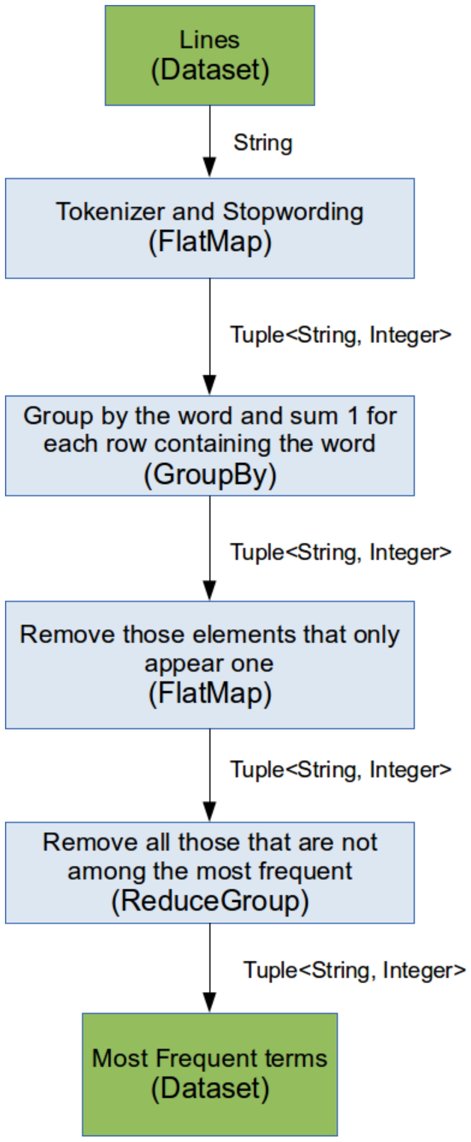

Points
(Dataset)
Point
Find the
nearest cluster
(Map)
Centroid
(Broadcast)
Centroids
(Dataset)
Tuple<Integer, Point>
Add a new field to count the
number of points per cluster
(Map)
Tuple<Integer, Point, Long>
Group by cluster, compute number
of points and the sum of all the
coordinates
(GroupBy)
Tuple<Integer, Point, Long>
Calculate the coordinates
of the new centroid
(Map)
Tuple<Integer, Point, Long>
New Centroids
(Dataset)

Figure 1: Iterative KMeans for Stratosphere data flow



Lines
(Dataset)
String
Tokenizer and Stopwording
(FlatMap)
Tuple<String, Integer>
Group by the word and sum 1 for each row containing the word
(GroupBy)
Tuple<String, Integer>
Remove those elements that only appear one
(FlatMap)
Tuple<String, Integer>
Remove all those that are not among the most frequent
(ReduceGroup)
Tuple<String, Integer>
Most Frequent terms
(Dataset)

Figure 2: Frequent Terms for Stratosphere data flow

part of the KDD process. We will cover in Subsection 4.2.3 some packages already available to perform these tasks in big amounts of data.

Difficult steps

About the pre-processing and transformation of data, there are a variety of tasks that might be performed by the users. In general, the data processing tasks take more time than the data mining step[Pal07] and in this step we could include tasks like normalization of values, combination or matrices, PCA, data cleaning, inconsistent or malformed data, data reshaping[Wic07], etc. Most of these operations are usually easy to parallelize but there are a lot of different needs that they have to cover, which makes these steps the most difficult to tackle in our approach. Similarly, there are many options to do the interpretation and evaluation of the result, but in this case usually the amount of data we have to work with is not that big or can be reduced just taking a sample of the data to work around this problem.

Data mining

The data mining step consists in the application of algorithms to discover patterns in the data. This step is usually the most difficult part to parallelize due to the complexity of the algorithms and where we believe our approach can have a very good competitive advantage. Not without a reason, a lot of effort has been put in paralellizing different data mining algorithms and offering them in libraries.

Data visualization

Finally, data visualization might be useful in different steps of the process but these tasks cannot be distributed among different nodes. R can however cover this tasks with packages like *tabplot*[TdJDN13], oriented to the visualization of large dataset by presenting how distributed are the instances of the whole dataset using one dimension, facilitating data discovery and exploration. It is also possible to sample the data in order to speed up the process. Some other packages explore similar problems like representing large amounts of data with a high number of dimensions[OW11, WO11].

2.4 Design goals and conclusion

Link with the previous sections

In summary, so far we have studied a few cases that motivate the conception of a different solution and analysed the two main ideas behind our approach. As we have discussed before, we have shown that our targeted users do not want to develop hand-coded parallel programs and evidences support that this is the largest group of users among the roles described as data scientists. In this section, we will present the main design goals of our solution which aims to satiate the appetite for large-scale data processing.

We will provide a library that offers functions very similar to those already existing in R, mimicking the interface as much as possible, but processes the data in the cluster. For this purpose, we also provide the means to load the answer in R or a sample of the output, in case it is very big. We do it this way for the reasons explained before: first, writing parallel programs is a **cumbersome** and onerous process that is not appealing for the majority of the data analysts and, second, most of the complex tasks performed with data to extract knowledge from it can be covered with a **library**.

Overview of the idea

In this world of tradeoffs, **accessibility** is very often a neglected consideration. We aim for big data platforms to reach a wider audience by making it a top priority and our approach to it is offering an interface that is as similar as possible to the well-known R packages. This results in a mechanism that allows the user to effortlessly execute the ready-to-use algorithm and operations with the data.

Easiness is a must

The design of a library that covers different data processing tasks is one of the most significant parts of this work. In consequence, this library will cover not only machine learning and data mining tasks but also file management. The design should also be realistic. It is impossible to implement a very exhaustive library that cover a huge number of cases. The library should be a far-reaching but prudent. In other words, we are aware that is not feasible to include any possible algorithm in a distributed library. The algorithm will be selected based on the previous experiences in creating data processing libraries, putting the focus on those for distributed computing. Furthermore, the functionalities selected should be suitable for large-scale computation and fit ensemble scenarios of processing in R and in the cluster as those presented in Section 2.1. Last but not least, the functionalities should be categorized to do not deliver a hotchpotch to the user.

Details about the library

The purpose of the resulting software is executing some of the tasks already available in different R packages but in a distributed system using the Stratosphere big analytics platform. With this software, we do not intend to reinvent the wheel and thus it will be implemented on top of already existing packages that bring functionalities for interaction with HDFS or any other reusable component. We will put the focus on bringing to the user new functionalities to interact with both the Stratosphere execution engine and the HDFS distributed file system of the cluster.

What the software should do?

Finally, the software should be accessible for the analysts. First of all, they should be able to perform analysis in the cluster within the R envi-

Easy for the users

ronment as they would do any other task. This also means that installing libraries or any other software in the main node or in the slave nodes of the cluster should be avoided if possible. As we have mentioned (see Section 3.3), R packages are the de-facto system for software distribution among the R community and our solution will preferentially follow this lead.

In summary, the goals presented in this chapter can be summarized in the following points:

- Efficient execution in large amounts of data.
- Easiness for the user and accesibility.
- Designing a realistic but far-reaching library.
- Avoiding writing parallel programs.
- Facilitating the transfer of the data between the cluster and the R environment.

Chapter 3

Developing the solution

The purpose of this chapter

In the second chapter, we have presented the main ideas behind our approach. In this chapter, we investigate the possibility of bringing our idea to life. In summary, we previously proposed the creation of a library that covers the most popular tasks in data analysis and manipulation and allows working with traditional R packages and large-scale analytics in the cluster at the same time. For this purpose, we had defined the scenarios that motivate our solution (Subsection 2.1), the tasks targeted (Subsection 2.3) and the design goals of the solution (Subsection 2.4). Since developing the whole system would be biting more than we can chew by far, our concrete goal in this chapter is to prove that this approach can lead to a solution that reaches the comfort zone of those users that use R for data processing purposes focusing on those scenarios.

Structure of the chapter

In the first section (3.1), we will architect the software that will be implemented in order to provide a solution for the problems discussed before. In the section 3.2, we will define the library that will cover data mining tasks as we argued in section 2.3. In the third chapter (3.3), we will discuss about the resulting software distributed as a package. Finally, we will provide some examples of how to solve the problems presented to motivate our solution (Section 2.1) in Section 3.4.

3.1 Architecting the solution

Why an architecture?

In the previous chapter we described what is the approach that we want to follow to solve this problem and in this chapter we will give details on how we decided to solve it. The first step to the implementation of the solution is architecting a feasible solution to achieve the proposed goals. As it can be observed in Figure 3, several components take part in the architecture, and hence a good planning becomes essential.

What do we mean by architecture?

Architecting the solution consists in deciding the components of the system that are going to take part, defining how they are going to interact and finally sketching a diagram to represent this interaction and to make easier the communication of the idea. Since one of our goals is reutilizing already existing software, interaction with third-party software is also a critical part of the architecture.

Interaction with the cluster

Figure 4 shows that the communication with the cluster has two touchpoints: the Stratosphere client that is necessary to execute the jobs and the HDFS system. It doesn't make sense to reimplement neither a Stratosphere client nor a library to interact with the HDFS system, when the Stratosphere java client and the *rhdfs* R package can be reused. Besides the saved time and resources, reusing these software pieces also assures that the code will be compatible with new releases of the Stratosphere system and the HDFS file system as long as the interface does not change without any effort. More details on how these components technically interact will be revealed in Section 3.3.

Interaction with the library

In our design, the R package is responsible for storing the library, that is a compilation of Stratosphere programs in JAR executables. We believe that this is the right method because the interface offered in these programs are strongly coupled with the code that wraps their functionality in R, so a small change in the interface of the program has to be reproduced in the R package. The main problem of this approach is that the programs should be compatible with the version of Stratosphere installed on the cluster nodes. Initially, this problem is easy to work around since there are not many version of the Stratosphere system and hence it is reasonable to include a JAR file compiled for each of them. This system could be improved with component that could automatically generate an R wrapper for any Stratosphere program, but it is not necessary at this stage.

3.2 Library

Connection with the previous chapter

Previously we have discussed that we want to cover the data mining algorithms plus some other tasks with our solution (in Section 2.3) and the goals that we will have for this library (in Section 2.4). In this section, we are going to use this as a starting point of the library we want to design. As

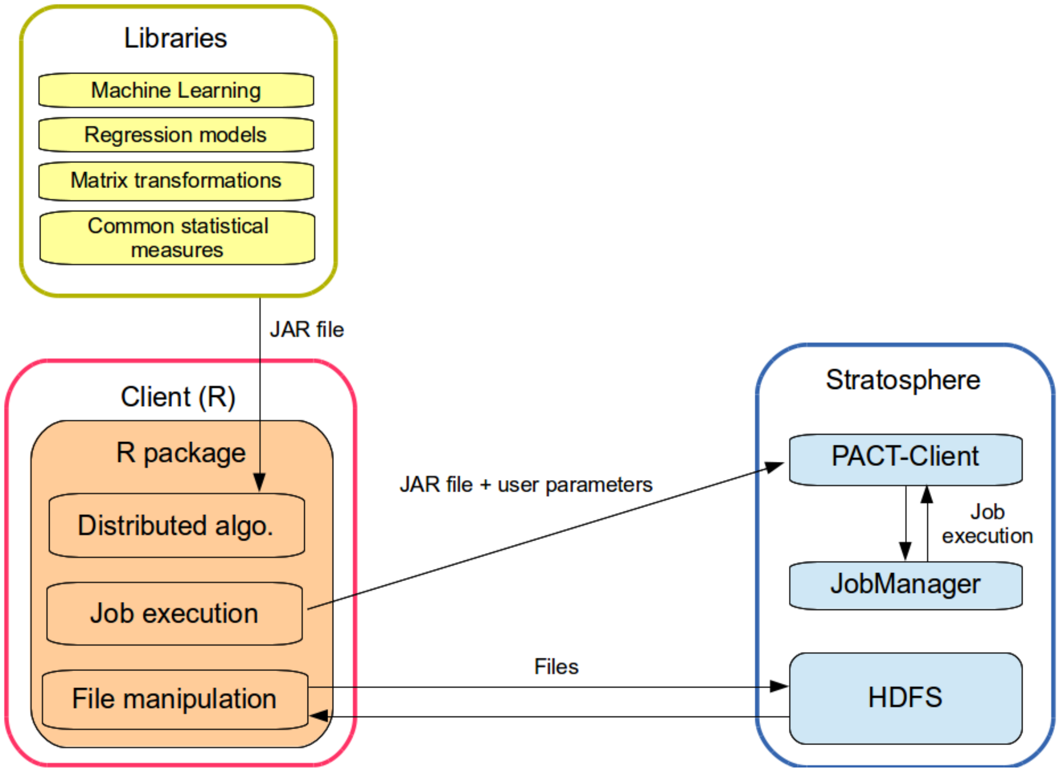


Figure 3: Solution architecture

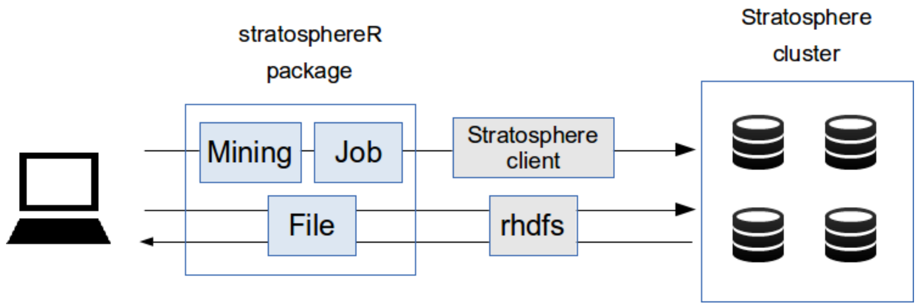


Figure 4: Execution of jobs using the stratosphereR package

an output of this study we will have a list of algorithms that would conform an ideal library for large-scale data processing and mining. The resulting library is not related to the choice of R as language and could be useful for any other interface, which makes sense since R is not the language of choice for everyone as we discussed in Section 1.2.

Fayyad et al.[FPSS96a] identified five groups of data mining methods: classification, regression, clustering, summarization, dependency modeling and change detection. Due to their impact on the machine learning libraries used as reference, we are going to focus on the first three methods. **Classification** and **clustering** have already been presented in Subsection 2.1.2 and 2.1.1, respectively. **Regression** consists in discovering how a variable is affected by other variables in order to predict the value when the values of other variables are known.

Our goal: clustering, classification and regression

For each group, we will choose one or more than one algorithms. The *No Free Lunch Theorem* for machine learning[Wol96] refuted the hypothesis that it is possible to find an inherently superior classifier to all the others. In Magdon-Ismail[MI00] words, "No Free Lunch theorems have shown that learning algorithms cannot be universally good". In consequence, we will not restrict the choice to one algorithm for classification, one for clustering and one for regression.

We will choose 1+ algorithms

We will explain now how we decide which mining algorithms are relevant. The selection will be based on the criteria exposed in Section 2.4: presence in other machine learning libraries, adequacy for large-scale parallelization and capability of participation in ensemble scenarios. We have chosen four libraries among those covered in Section 4.1: *sci-kit*[PVG$^+$11], a very comprehensive machine learning for Python; *Mahout*[OADF11], a scalable library with algorithms implemented on Hadoop and Spark; *MADLib*[HRS$^+$12], a library for highly-scalable relational database; and *MLbase*[KTD$^+$13], a library for Spark. We believe that the chosen libraries have a relevant impact in academia and industry. Furthermore, we have included in tables 1, 2 and 3 the R package that contains the implementation of the algorithm.

How to choose the methods?

Based on the analysis shown in Table 1, the most popular techniques for classification are **Support Vector Machine**, **Logistic Regression**, **Naive Bayes** and **Random Forests**. Table 2 shows that **K-Means** is the only technique with strong presence for clustering. Similarly, we can observe in Table 3 that **Logistic Regression** is the only technique with strong acceptance among the large-scale libraries for regression analysis. While elaborating this table, we have noticed that the **stochastic gradient**

Conclusions

Table 1: Classification algorithms in different machine learning libraries

Technique	R	scikit	Mahout	MADlib	MLbase
Logistic regression	glm	Y	Y[†]	Y	Y
Naive Bayes	CORElearn	Y	Y	Y[‡]	N
Perceptron	rminer	Y	Y	N	N
SVM	e1071	Y	Y	N	Y
Quadratic classifiers	DiscriMiner	Y	N	N	N
K-Nearest neighbor	CORElearn	Y	N	N	N
Boosting	gbm	Y	N	N	N
Random forests	randomForest	Y	Y	Y[‡]	N
Neural network	nnet	Y	N	N	N
Gene Expression	GeneReg	N	N	N	N
Bayesian networks	BayesTree	N	N	N	N
Hidden Markov models	HiddenMarkov	N	Y	N	N
Learning vector quantization	LVQTools	N	N	N	N

[†] Although Mahout is a scalable machine learning library, Logistic Regression implementation is not parallel.
[‡] These methods are in early stage of development and subject to change, in accordance with the official documentation.

Table 2: Classification algorithms in different machine learning libraries

Technique	R	scikit	Mahout	MADlib	MLbase
Canopy	Not in CRAN	N	Y	N	N
K-Means	kkmeans	Y	Y	Y	Y
Fuzzy K-Means	fanny	N	Y	N	N
Streaming K-Means	stream	N	Y	N	N
Spectral	kernlab	Y	Y	N	N
DBSCAN	fpc	Y	N	N	N

descent[Zha04] method for optimization seems to be widely used in combination with **regression** techniques. Multilayer perceptron network and neural networks have no big presence in libraries, although they have been proven effective in many applications, due to the complexity of finding suitable initial weights and learning parameters for them [CCM+00, MPC+93].

3.3 Creating the R package

What are we presenting here?

In this section, we present the R package developed which is part of the deliverables of this work. R packages are distributable, cross-platform, easy to implement, memory-friendly and easy to maintain[Lei08]. They have emerged as the *de facto* standard for distributing cutting-edge data analysis functionalities. The package that we are going to present here follows the suggestions and guidances from the *R Foundation for Statistical Computing*[Tea99].

Structure of the package

As we have explained in Section 3.1, the package is composed of different files and the namespace is organize accordingly. The package has three namespaces that we believe conceptually divide the functions: functions to apply machine learning methods, functions to interact with files in the cluster and functions to control job execution. The last namespace (job) it is intended to cover the needs of power users and ideally the users only have to use it if the library does not cover their needs or want to have full control of the execution in the cluster. The exact members of the package provided are represented as a tree in Figure 5. Moreover, Figure 4 shows how the different component interact with the cluster.

3.3.1 Submitting jobs to the cluster

How do we submit the jobs?

Submitting jobs to the Stratosphere JobManager is obviously one of the main responsibilities of the *stratosphereR* package. In Stratosphere, *Stratosphere-Client* (previously know as *PACT-Client*) is responsible for parsing the arguments, checking them and executing either a blocking or non-blocking call to execute the PACT programs on the cluster. We have integrated this interface into an R package by invoking Java classes from R[Ver10]. Thanks to the *rJava* package, we are able to access any object defined in Java. In our case, we have decided to interact with the command line interface of the client. By default, the calls are blocking in order to mimic other R packages' behaviour.

Table 3: Regression algorithms in different machine learning libraries

Technique	R	scikit	Mahout	MADlib	MLbase
Least Squares	lsmeans	Y	N	Y	N
Ridge Regression	bigRR	Y	N	N	Y
Lasso	glmnet	Y	N	N	Y
Proportional Hazards	glmpath	N	N	Y	N
Elastic Net	elasticnet	Y	N	Y	N
Least Angle	Rxshrink	Y	N	N	N
Bayesian	BGLR	Y	N	N	N
Logistic	ncvreg	Y	Y	Y	Y
Perceptron	N	Y	N	N	N

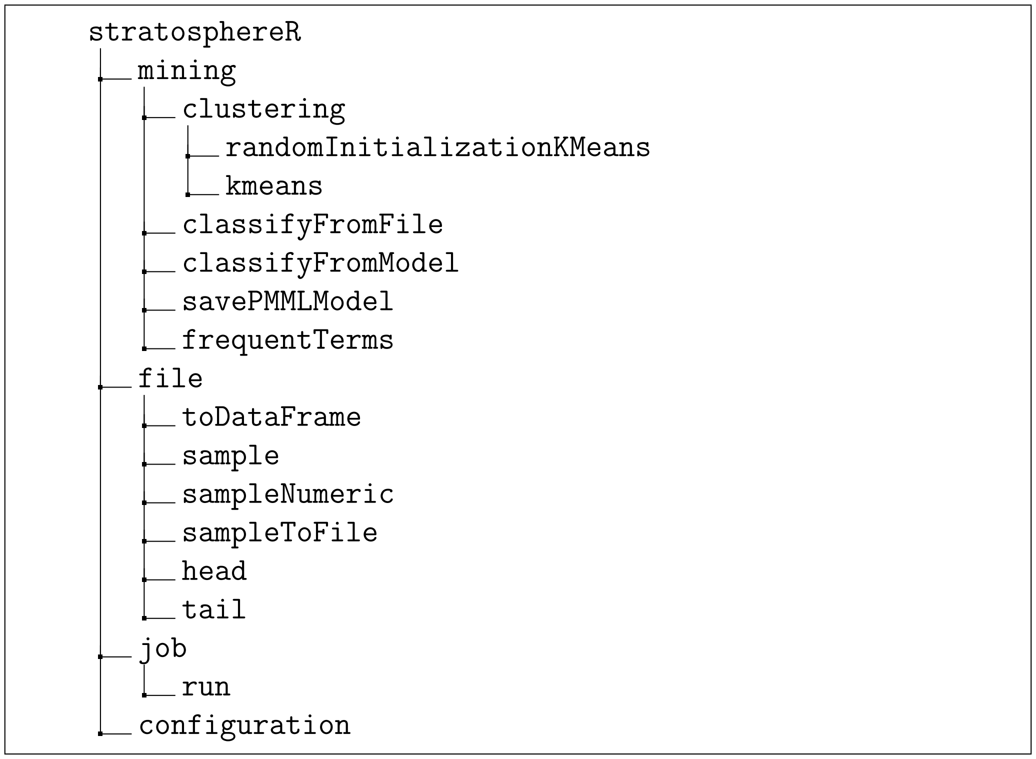


Figure 5: Tree representing the *stratosphereR* package

As we have mentioned in section 2.4, it is not our intend to "reinvent the wheel" and hence one of our goals is to maximize the reutilization of the already existing code. By reusing this component, we also assure that the package is going to be compatible with new upcoming versions of the platform as long as the interface of this package does not change. At the same time, the exceptions that could occur in the java Stratosphere client will be caught in the R code and showed when the client fails, giving the same level of transparency to the R users.

Why we do it like that?

In order to be able to submit a job to the cluster, the package needs some basic information. First of all, it is necessary to know the host and the port that it is necessary to connect to, if the values given in the config file are not correct. It is also necessary to know the degree of parallelism that we want to use for the execution of the jobs, again in case we are not going to use the value given by default in the configuration file. The path of this configuration file must be also defined if it cannot be found in the *STRATOSPHERE_CONF_DIR* environment variable.

Configuration

3.3.2 Working with files from the distributed file system

Low-level functionalities to work with files like reading or writing lines and binary blocks are provided by *rhdfs* package. We believe these low-level functionalities make processing distributed files hard and hence we have provided new functionalities built on top of this package.

Why we need this?

First, we allow sampling a text file and bringing it to R memory (*file.sample*). It is also possible to sample the file and write the output back in HDFS (*file.sampleToFile*). This option is particularly interesting because most of the file stored in a distributed filesystem do not fit in main memory. In consequence, we can take a representative sample of the file for different purposes like running accuracy tests (see Subsection 2.1.1). In addition, we have included also a function (*file.sampleNumeric*) that converts numeric values into R native numeric data types at the same type as the sample is taken, taking this load off the user and speeding up the process since it can be performed in only one step. Finally, we have included the option of taking a sample of the first or last lines of the file using *file.head* and *file.tail* respectively.

Sampling options

Data frames are lists of vectors and they are the most common way of manipulating data in R[Wic14]. A proof of this success is the package *pan-*

Also the famous data frames

das[McK12], which brings data frames to Python. Surprisingly, the *rhdfs* package do not offers any operation oriented to data frames. We believe that files from HDFS must be easily exported into data frames and hence the *file.toDataFrame* allows this operation. This would be particularly interesting for those jobs that use a big chunk of data as input but produce an output that fits into R memory limitations, as the KMeans scenario that we presented in 2.1.1, the frequent terms in 2.1.3 or aggregation of data.

Hitting R memory limits

Obviously, these operations might try to bring to memory files that do not fit into R memory. These situations are obviously not wished and result in disappointment for the user. Although there are different ways of estimating the space available in R[Wic14], we failed to calculate or approximate the amount of memory required to perform these operations (sampling files or loading frames in a table). Furthermore, we found that not only the space needed to bring the file might cause problems due to R's memory limitations. We experienced this with a simple operation like selecting the blocks that we are going to sample. The problem happened because the number of possible blocks were too high due to the big size of the file and the decision of bringing to memory only small blocks to reduce the communication between the cluster and the client. Since the number of possible blocks were too big, the system could not keep a reference to all of them and we had to work around this problem.

3.3.3 Data mining functions

What do we present here?

Finally, as a result of the analysis of data mining libraries presented in Subsection 3.2, some data mining functionalities have been introduced in the R package. These functions are defined within the *mining* namespace and it is shown in Figure 5.

Related with the KMeans scenario

In relation with the KMeans scenario, it is possible to execute the fully distributed KMeans algorithm (which dataflow is represented in Figure 1) using either random initialization or any other initialization points defined by the user. The random initialization samples the file from the distributed file system.

Related with the Naive Bayes scenario

In relation with the PMML models that we will discuss in detail in Subsection 3.4.2, it is possible to classify a file of unlabelled instances using a PMML file (*classifyFromFile*), to classify the instances using an R model (*classifyFromModel*) and also to store a PMML model either in the distributed file system or in a traditional one (*savePMMLModel*). An example of how PMML models look like can be found in Appendix A.

Finally, we also have covered a very simple text mining use case for the problem proposed in Subsection 2.1.3. The function *mining.frequentTerms* allows user to find the most frequent terms in a large corpus. In this case, the only difference with the functionality provided in the *qdap* package is that the set of stop words is fixed and it is not possible to define new ones.

Related with the Frequent Terms scenario

3.4 Examples

In this section, we retake the problems presented in Section 2.1 and discuss about how they can be solved using our approach. This section can be considered a kind of evaluation since we will provide the code that uses the *stratosphereR* package to solve the problems and discuss the differences between both versions, differences which are summarized in a comparative table.

What is this section about

3.4.1 Clustering with K-Means

In this subsection, we proceed to present the differences introduced in the code to execute the code in the cluster using the *stratosphereR* package. First, we consider the scenario presented in Subsection 2.1.1 as a starting point. In this scenario, a dataset is clustered using different parameter configuration, these solutions are inspected visually and finally the best solution is kept. The scenario is summarized in 9 steps defined in Algorithm 1 and the R code presented in Listing 2.1 is tantamount to this algorithm.

Recap

It goes without saying that there are clear differences between the code to run the clustering scenario in a single machine (Listing 2.1) and the code to run it in the clustered (Listing 3.1). The main differences come from the fact that there is a big change in how the computation of the clustering is done. In the fist case, the data is moved to memory in order to do the processing. In the second case, the processing is moved to the data to efficiently perform parallelization and data-intensive computation. To get a detailed view of how the data is transferred between the client and the cluster and what are the components in charge of this communication (sometimes the HDFS service and sometimes a Stratosphere program), Figure 6 can be used. An equivalent diagram for the non parallel version is available in Figure 7.

Moving the computation to the data makes a difference

The code used to solve this scenario using the cluster is provided in Listing 3.1. The differences between the single-machine version and the cluster one are summarized in Table 4. In comparison with the single-machine version, an extra step for loading the package is added. On the other hand, some steps are not necessary since there is no need of moving the data from the file system (Steps 1 and 9). The data **sampling step** is somehow different. In the cluster version, it is not possible to sample the same points of the original file for each execution of the algorithm. This is due to the impossibility of predicting in which position of the text file each point it. To accomplish this we would need to store the information in binary files. This difference is relevant because the selection of different points might lead to small variations in the computation of the Dunn index. As a result of this change, the distances in the cluster-aware code have to be computed for the three samples.

Differences in detail

3.4.2 Classification with Naive Bayes

In this subsection, we proceed to present the differences introduced in the code to execute the code in the cluster using the *stratosphereR* package. First, we consider the scenario presented in Subsection 2.1.2 as a starting point. In this scenario, two datasets are used as an input. The training dataset is used to construct a probabilistic model and the unlabelled data is used to classify its instances using the mentioned model. The scenario is summarized in 5 steps defined in Algorithm 2 and the R code presented in Listing 2.2 is tantamount to this algorithm.

Recap

As we have already explained in Subsection 2.1.2, the goal is to allow the training of the dataset in the R environment and the classification of unlabelled instances in the Stratosphere platform. The main obstacle in this process is transferring the predictive model from the R environment to the scoring algorithm. In order to overcome this obstacle, it is necessary to store the model in a way that the scoring algorithm can use it. Fortunately, a standard based on XML called PMML[GHM02] was proposed by the Data Mining Group consortium. This standard supports Naive Bayes models[AG13] among others. The standard is already in a mature stage of development and had become a de-facto standard adopted by a variety of tools and products like *Weka*[HFH+09], *RapidMiner*[PMHGP11], *KNIME*[BCD+09] and *Oracle*. For our purposes, we will use the *pmml* package[ZGW09] which allows users to export some models into PMML files. An example of a PMML

How to share the model?

Table 4: Differences between parallel and non-parallel clustering programs

	Step	**Single-machine**	**Cluster**
0	Loading package.	-	Install and load the *stratosphereR* package
1	Read.	Read the csv file into a data frame	Cluster the points using the different parameters configuration
2	Cluster.	Cluster the points using the different parameters configuration	
3	Sample.	Sample one hundred points from the output of the clustering	Sample one hundred points from **each** output of the clustering.
4	Distances.	Compute the distances between the points sampled.	Compute the distances between the points of each sample.
5	Evaluation.	Compute the Dunn index for each execution of the algorithm.	Compute the Dunn index for each execution of the algorithm.
6	Plot evaluation.	Plot in a bar chart the results of the previous step.	Plot in a bar chart the results of the previous step.
7	Choose the best solution.	Assign a new name to the best solution and the best sample.	Assign a new name to the best solution and the best sample.
8	Visual inspection.	Plot the sample using a colour code for the clusters.	Plot the sample of the best solution using a colour code for the clusters.
9	Write	Append the clusters to the original data frame and write it into secondary memory.	*Already done in step 2*

Listing 3.1: Distributed clustering with K-Means using stratosphereR

```
# 0 Load stratosphereR package
install.packages("stratosphereR")
library(stratosphereR)

# 1 We don't have to read because the file is already in the HDFS system

# 2 K-Means to cluster
cl_outputPath2 <- stratosphere.mining.clustering.randomInitializationKMeans
    (2, "/points")
cl_outputPath3 <- stratosphere.mining.clustering.randomInitializationKMeans
    (3, "/points")
cl_outputPath4 <- stratosphere.mining.clustering.randomInitializationKMeans
    (4, "/points")

# 3 Sample data
stratosphere.file.sample(paste(cl_outputPath2, "points", sep = ""), "/
    sampleclustered2", 100)
stratosphere.file.sample(paste(cl_outputPath3, "points", sep = ""), "/
    sampleclustered3", 100)
stratosphere.file.sample(paste(cl_outputPath4, "points", sep = ""), "/
    sampleclustered4", 100)

# 4 Compute the distances between points
cl_d2 <- dist(cl_datasample2[c("V2", "V3")], method = "euclidean")
cl_d3 <- dist(cl_datasample3[c("V2", "V3")], method = "euclidean")
cl_d4 <- dist(cl_datasample4[c("V2", "V3")], method = "euclidean")

# 5 Evaluate the clustering
library(fpc)
cl_dunn <- c(
        cluster.stats(cl_d2, as.numeric( cl_datasample2$V1 ))$dunn2,
        cluster.stats(cl_d3, as.numeric( cl_datasample3$V1 ))$dunn2,
        cluster.stats(cl_d4, as.numeric( cl_datasample4$V1 ))$dunn2
)

# 6 Plot dunn indexes
barplot(cl_dunn, names.arg=c(
        "k=2", "k=3", "k=4"
))

# 7 Decide which solution is the best one
cl_best_sample <- cl_datasample2
cl_best_path <- cl_outputPath2

# 8 Plot to visualize the best solution
plot(cl_best_sample$V2, cl_best_sample$V3, col=cl_best_sample$V1)

# 9 The solution is already on the file system
```

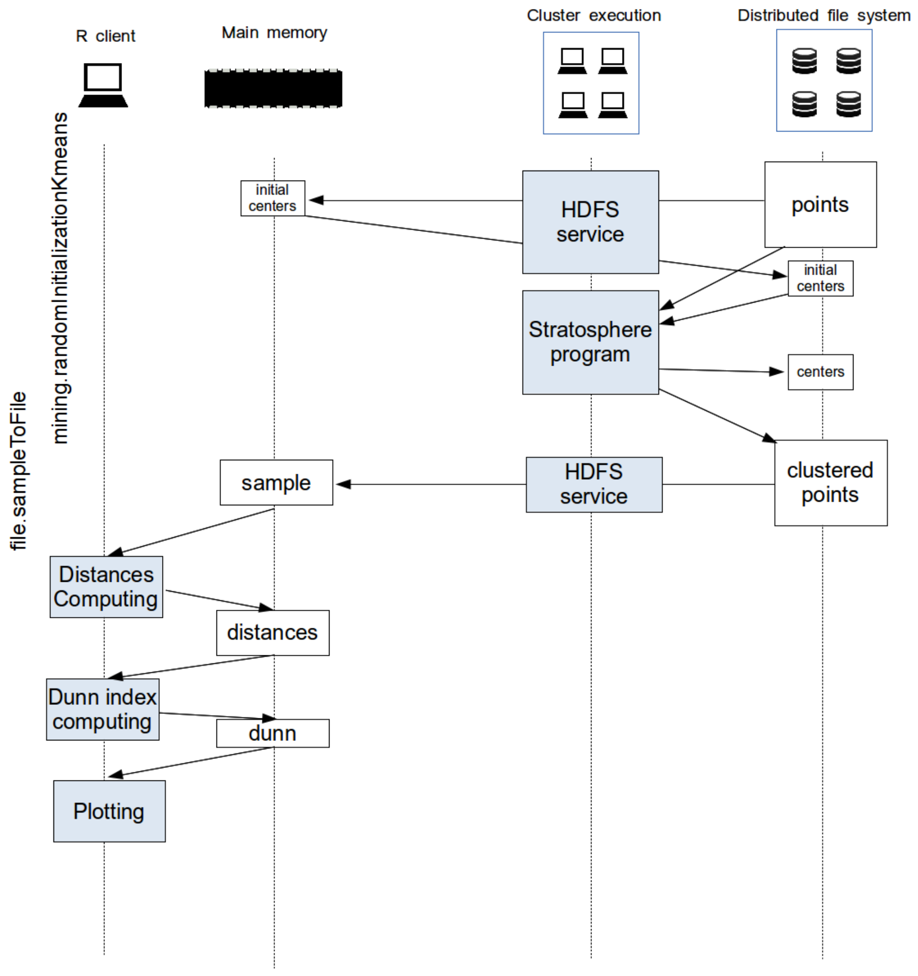


Figure 6: Data flow of the operations of the parallel version of the clustering scenario

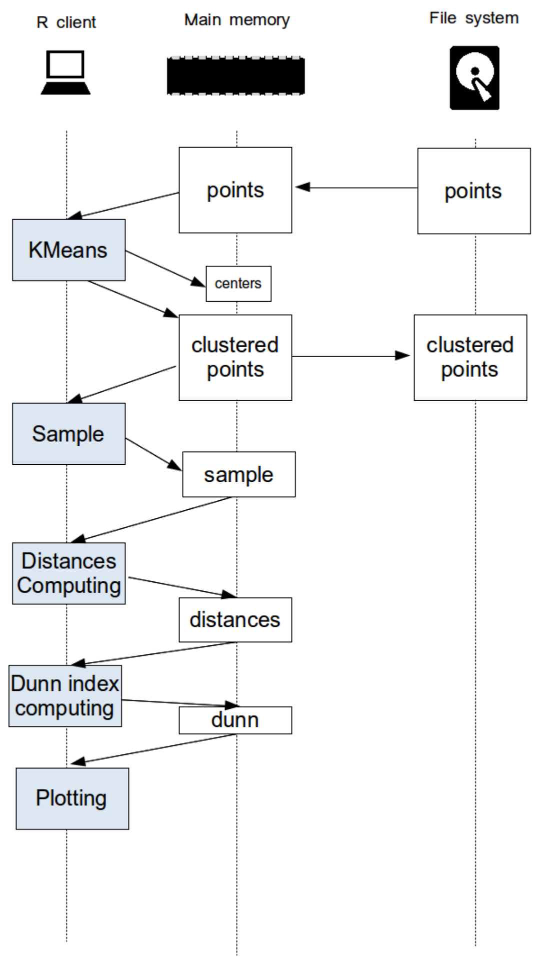


Figure 7: Data flow of the operations of the non-parallel version of the clustering scenario

model is included in the Appendix A.

Implementation details

In order to solve this problem, we have implemented a PMML evaluator for Stratosphere reusing the already existing *jpmml* library[jpm]. This library, among other interesting features, can handle missing and invalid values and outlier instances. The evaluator takes as input the file with the instances that should be classified, the path to output the result and the model generated with R. In order to hide some complexity from the user, the *stratosphereR* package automatically stores the PMML in disk (in the HDFS distributed filesystem) and executes the Stratosphere program using this file.

Differences with the non-parallel version

As it can be noticed in Table 5, Using the parallel version of the Naive Bayes classification (Listing 3.2) introduces a small overhead in the process, in comparison with the memory-limited single instance one. First, it is necessary to load two additional packages (*stratosphereR* and *pmml* and second, it is necessary to convert the model generated by the *naiveBayes* function to PMML format. As we have mentioned in the previous example (Subsection 3.4.1), the main difference in the processing is due to the change of paradigm; in the parallel version, the computation is moved to the data and not the opposite.

3.4.3 Finding frequent terms

The solution in short

In this subsection, we demonstrate how the *stratosphereR* package can solve the problem proposed in Subsection 2.1.3. We are going to use a Stratosphere program written specifically to solve the needs of this scenario and hence the behaviour is aligned with the single-machine counterpart present in the *qdap* package. The difference relies in the use of Stratosphre operations to solve the problem, which allow us to execute the program in a highly parallel system.

Relevant implementation details

As it can be observed in the code that uses the *stratosphereR* package (Listing 3.3), the last step is executed in the R environment since the output of the Stratosphere program is small enough to fit in R memory in spite of its limitations. We have tested that R can handle without problem up to 12 million instances as output of this program and the overload of the communication is insignificant, it takes more time to load some R packages than bringing the output from the cluster to the R environment.

Differences with the non-parallel version

In the parallel version, it is necessary to bring the resulting data of the second step in the cluster to the R environment, which adds an additional

Listing 3.2: Distributed classification with Naive Bayes using stratosphereR

```
# Required libraries
install.packages("stratosphereR")
library(stratosphereR)
library(e1071)
library(pmml)

# Load the required dataset
dataset_train <- stratosphere.file.toDataFrame("/train.csv")

# Create the model
model <- naiveBayes(dataset_train[,1:4], dataset_train[,5])

# Output the PMML representation
pmml_model <- pmml(model, predictedField="Species")

# Classify the unlabelled data using the PMML model
stratosphere.mining.classifyFromModel("unlabelled.csv", "output.csv", pmml_
    model)
```

Table 5: Differences between parallel and non-parallel classification programs

	Step	**Single-machine**	**Cluster**
0	Loading package.	Install and load the *e1071* package	Install and load the *e1071*, *pmml* and *stratosphereR* packages
1	Read.	Read the csv file with the labelled instances into a data frame.	Read the csv file with the labelled instances into a data frame.
2	Train the model.	Train the model.	Train the model.
3	Prediction.	Predict the class of the unlabelled data.	Predict the class of the unlabelled data.
4	Semantic relatedness.	Write the result to disk.	

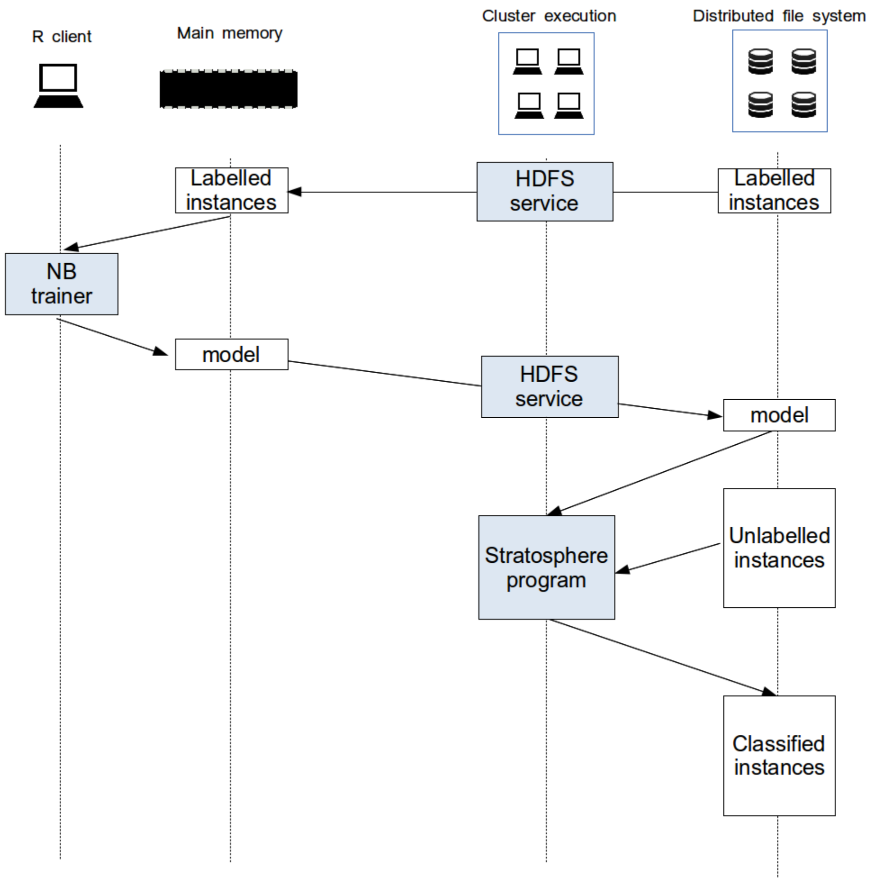


Figure 8: Data flow of the operations of the parallel version of the classification scenario

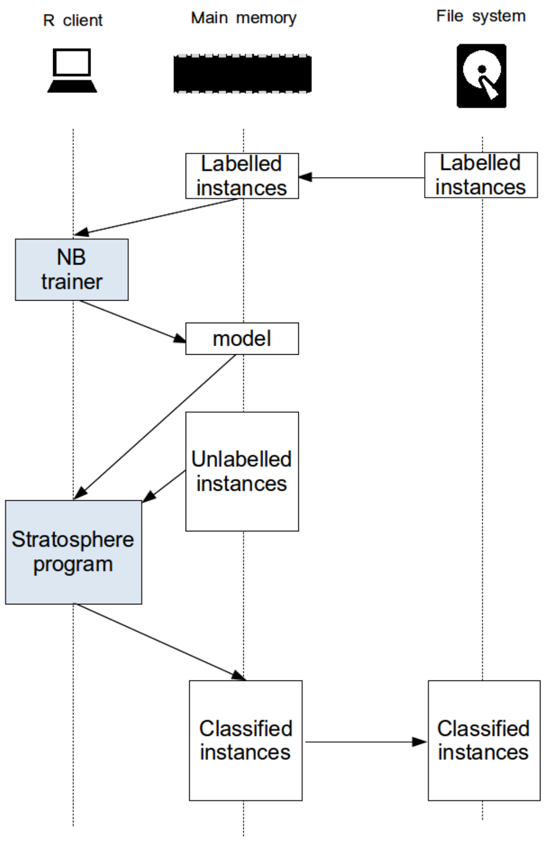


Figure 9: Data flow of the operations of the non-parallel version of the classification scenario

step to the process. In the version however, it is not necessary to load the data since the Stratosphere program already deals with the data in the given format (CSV). If we don't take into account the loading of an additional package, both versions of the program have the same number of steps, unlike the first example since we don't have to write the output to disk. In spite of this, these steps are slightly different as Table 6 shows.

3.5 Performance evaluation

Goal of the tests

We now present a set of experiments studying the viability of our solution and its performance in comparison with traditional non-parallel R programs. At the same time, we conduct experiments benchmarking the scalability of the Stratosphere programs presented. This work did not compared empirically Stratosphere with other large-scale execution engines since already conducted experiments indicate that Stratosphere offers comparable or better than general-purpose execution engines (Hadoop and Hive) and domain-specific ones (as Giraph).[ABE+14]

Environment

All tests have been executed in virtual private servers located in the same data centre with the same configuration in all the nodes: a processor of 2 GHz, 2 GiB of main memory and solid state drives. The data is transferred between nodes using a high speed private interface. We compare the Stratosphere programs running on Stratosphere 0.6 and using HDFS 1.2.1 against programs running on R 3.0.2.

Scalability test procedure

In our experiments, when the size of the file is doubled, the number of nodes in the cluster is also doubled. In consequence, a perfectly parallel line would represent a perfectly linear scalable implementation, which is impossible to achieve due to unavoidable costs of communication, scheduling, etc.

3.5.1 Naive Bayes

Non-parallel

Our very first experiments using the Naive Bayes classifier evidenced the memory limitations of R working with even medium sized objects. The machine used was not able to classify files over 180 MiB (2000000 instances with 4 dimensions) although the machine had a main memory of 2 GiB. The R implementation of the algorithms neither took advantage of the multicore processor. Figure 12 shows that in the non-parallel implementation, most of the time is consumed by the classification of the unlabelled instances. The

Listing 3.3: Distributed frequent terms using stratosphereR

```
install.packages("qdap")
install.packages("stratosphereR")


# Select the most frequent elements
pathFrequency <- stratosphere.mining.frequentTerms(allwordsPath, top=1000)

# Bring data to R
frequency <- stratosphere.file.toDataFrame("pathFrequency")
words <- frequency[,"WORD"]

# Find coincidences based on synonyms
library(qdap)
lapply(words, function(x) intersect( synonyms(x, return.list=FALSE), words)
    )
```

Table 6: Differences between parallel and non-parallel frequent terms programs

	Step	**Single-machine**	**Cluster**
0	Loading package.	Install and load the *qdap* package	Install and load the *qdap* and *stratosphereR* packages
1	Read.	Read the csv file into a data frame	Find a determined number of most frequent terms
2	Frequent Terms.	Find a determined number of most frequent terms	
3	Read result.	-	Bring the result to a data frame.
4	Semantic relatedness.	Find coincidences based on synonyms.	Find coincidences based on synonyms.

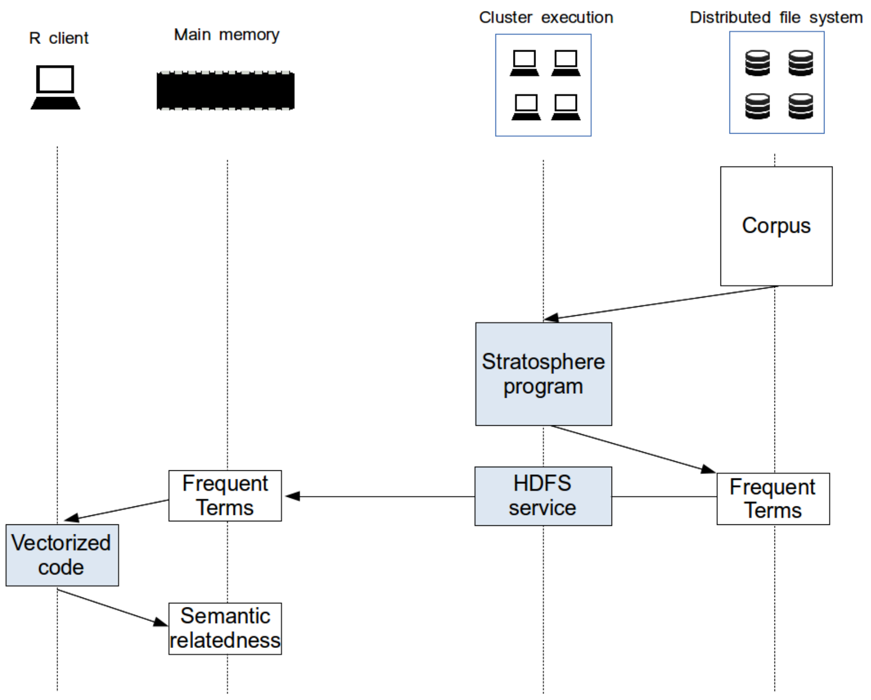


Figure 10: Data flow of the operations of the parallel version of the frequent terms scenario

disk service time probably would be higher in non-solid disks, but do not seem to be decisive in any case.

Parallel

In comparison, Java-based PMML classifier shows a strong performance. The algorithm executed in parallel in two instances of the cluster is able to classify points up to 10 times faster than R running in a single node (Figure 14). We have excluded the time consumed by the generation of the model as it is the same in both cases and we have seen that it is irrelevant considering our hypothesis of having a small dataset. This extraordinary good performance of the *stratosphere* program is in part certainly due to the processing of the data in a pipeline[ABE+14]. On the contrary, the R program has to wait until the whole file is loaded into memory to start processing the points. We have included the time consumed by the installation and loading of the required packages: *rJava*, *rhdfs*, *pmml* and *stratosphereR* itself. Under these circumstances, it is no wonder that the stratosphereR implementation performs specially better than the R implementation when the amount of data is bigger, since this cost is static, it does not increase with the volume (Figure 13).

Parallel in larger datasets

In order to test the behaviour of the Stratosphere program (the classification step only) in bigger amounts of data, we have conducted tests using again two instances but with gigabyte-scale datasets. The performance is observed to increase with the data size, as it is shown the results in terms of number of instances processed per second in Table 7. Probably this effect is caused because by high Stratosphere's overhead as time of launching the jobs and communication costs that do not depend on the size of the dataset. We did not execute the program in bigger datasets since programs that take more than one hour are not useful for interactive programming and because we reached the storage limitations of the cluster. Further scalability tests have not been performed since this is an embarrassing parallel task.

3.5.2 KMeans

Data intensive experiments

All our experiments using the KMeans algorithm confirmed empirically the memory limitations of the R interpreter. Since we could not test the performance of R with medium-sized datasets, our first experiments using this algorithm were designed to test both alternatives with a computer intensive task. For that, we used the scenario defined in Subsection 2.1.1 using the following parameters: 100 iterations; three executions of the algorithm; and 10, 50 and 100 centroids.

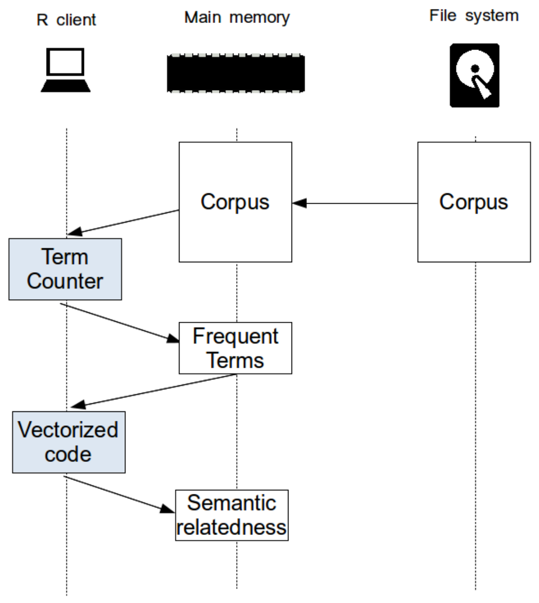


Figure 11: Data flow of the operations of the non-parallel version of the frequent terms scenario

Table 7: Performance in gigabyte scale of the Naive Bayes classifier in Stratosphere

Size	Instances	Execution time (seconds)	Instances processed per second
12 GiB	200000000	4531	44140
6 GiB	100000000	2791	35829
3 GiB	50000000	1425	35088

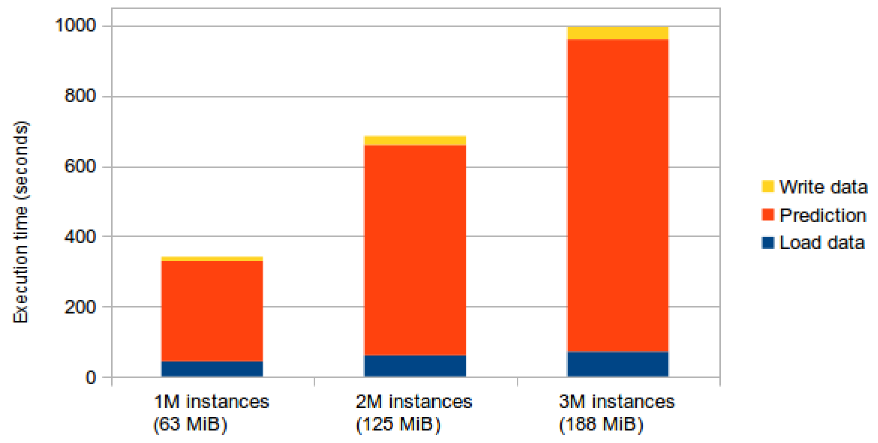


Figure 12: Naive Beyes non-parallel implementation time breakdown

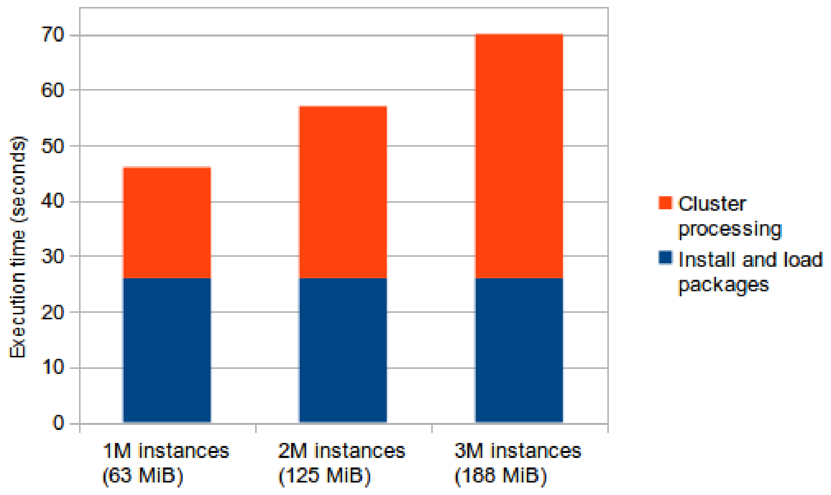


Figure 13: Naive Bayes parallel implementation time breakdown

Figure 15 clearly demonstrates that R KMeans implementation clearly outperforms Stratosphere even considering the time of loading from and writing to disk. This difference of performance is due to the overhead per iteration in Stratosphere. Stratosphere nodes need to communicate a partial solution, schedule new tasks and start them for every iteration. Using the *stratosphereR* package, the time consumed by the sampling of a file stored in the distributed file system is not significant. First, the files that fit into memory cannot be considered big and second, only small blocks of the file are read during the sampling, reducing the amount of data communicated between the cluster and the client. Once again, the R script failed when trying to process significantly small files (over 200 MiB).

The R version performs better

We have observed that the reading and writing time is not so big in comparison with the first example, because this time the case is more data-intensive. The time consumed computing the distances and the Dunn index can be considered big since we are computing this using only a sample of the file and this time could not be affordable if we were using all the points clustered.

KMeans is the bottleneck in the nonparallel

The second set of tests study the scalability of the Stratosphere program using bigger file sizes (i.e. at gigabyte scale). As we have mentioned before, we have duplicated the size of the file at the same time that we duplicate the number of nodes, duplicating hence the computational power of the cluster. Scaling out machine learning algorithms is an intense and troublesome topic of research. We will hence use a straight-forward approach and only discuss the scalability of the scenarios defined. Figure 16 represents time of processing against file size. As it can be observed, the processing of bigger files introduce little overhead and this overhead decreases with the number of nodes. These results among with the not-to-big slope (1.81) observed in the linear regression curve, prove that we can cost-effectively grow the capacity of the cluster.

Scalability tests

3.5.3 Frequent terms

The last set of tests were performed in order to evaluate the scenario presented in Subsection 2.1.3. We recall that this scenario consists of a frequent terms search and the discovery of semantically related words among these terms. We have also left out the last step of the algorithm (semantically relatedness) since this part is always executed within the R environment and no relevant findings were found. Since this algorithm is deterministic, we

What do we test?

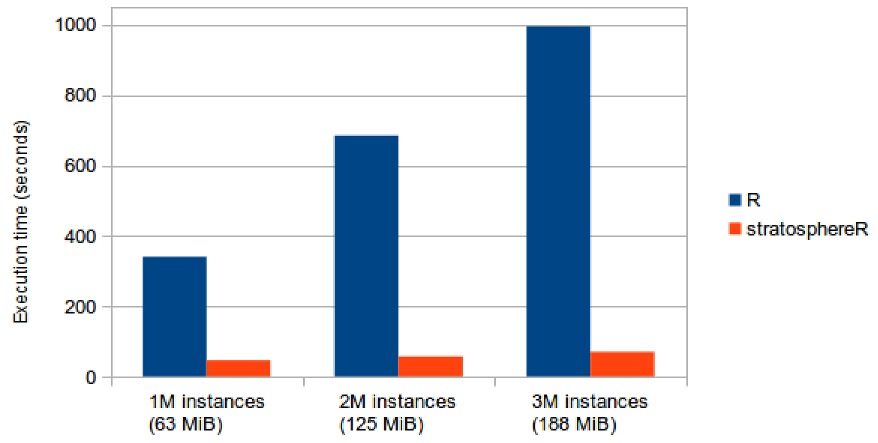


Figure 14: Performance comparison of Naive Bayes classifier implementations in small file sizes. Stratosphere implementation runs on 2 nodes meanwhile R implementation run on a single node

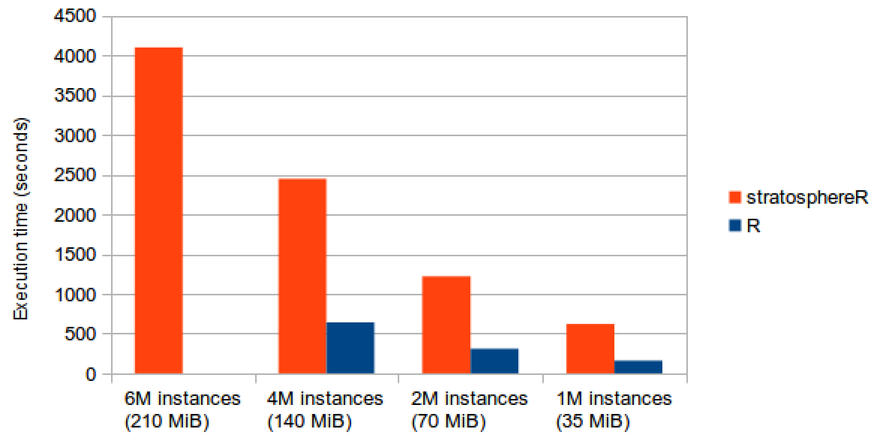


Figure 15: Performance comparison of KMeans implementations in small file sizes. Stratosphere implementation runs on 2 nodes meanwhile R implementation run on a single node. R did not have enough memory to process 6M instances.

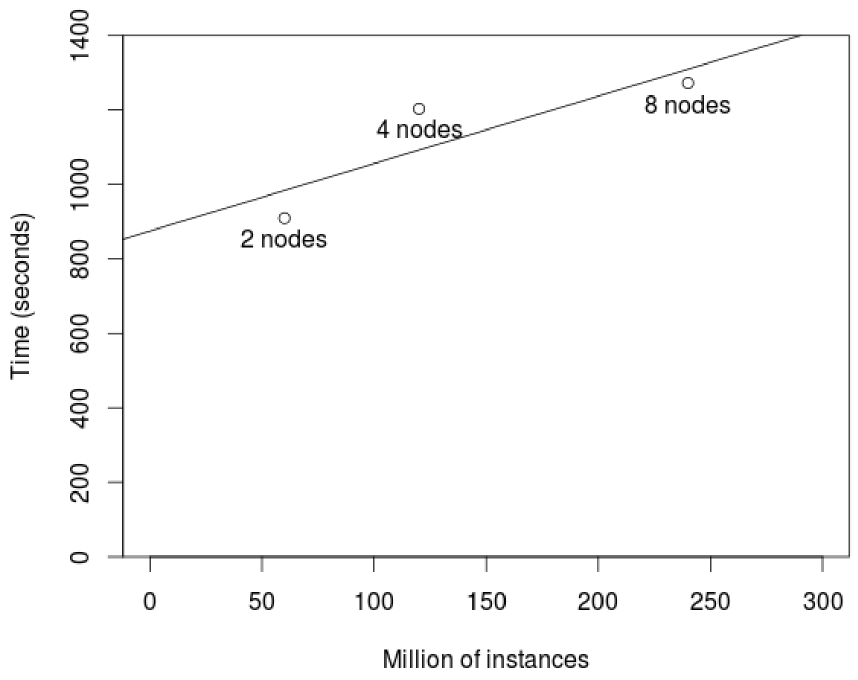


Figure 16: Scalability of the KMeans program in Stratosphere

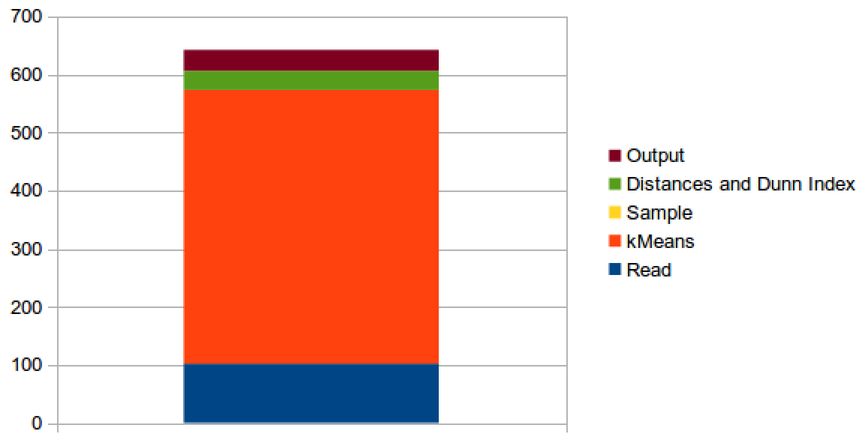


Figure 17: Clustering non-parallel implementation time breakdown

checked that the output of both implementations was the same for the same input. After assuring that the results are identical, we have run a set of tests to measure the **performance** and **scalability**.

Memory limitation in R again

Once again, the R environment proofs to be very constrained by memory limitations. In this case, we were not able execute the algorithm using as input a file of 133 MiB (five million words) because the programs failed when it tried to allocate 1 GiB of memory while trying to perform a pattern matching operation, which is part of the R base package. The allocation of a portion of memory 7 times bigger than the input file for this single operation does not seem to be appropriate, and the only explanation found to this fact is the copy-on-modify principle followed in R.

Result of the comparison

Again, we expected the Stratosphere experiment (Figure 18) in two nodes to finish in a little bit more than half of the time because the computational power was doubled. However, these expectations were overpassed and the Stratosphere program outperforms the R program, specially when we tested it with bigger files. In particular, the performance of the R package dramatically drops when we increase the size of the file despite of the small size of it.

Conclusion of the comparison

After taking a look to the code of the *qdap* package and special close look to the snippet that keeps the most popular elements, there is no significant differences in the implementation that explain that big difference. It was observed that the pipe-lining of the data gives Stratosphere a clear competitive advantage.

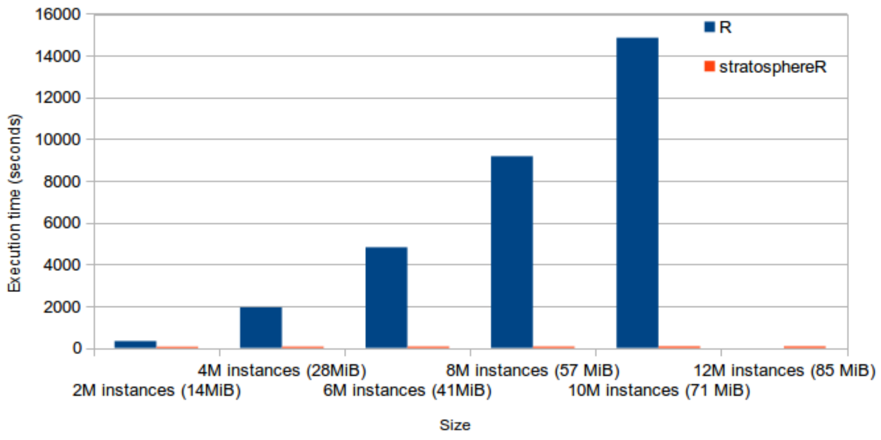


Figure 18: Performance comparison of the Frequent Terms implementations in small file sizes. Stratosphere implementation runs on 2 nodes meanwhile R implementation run on a single node. R did not have enough memory to process 12M instances.

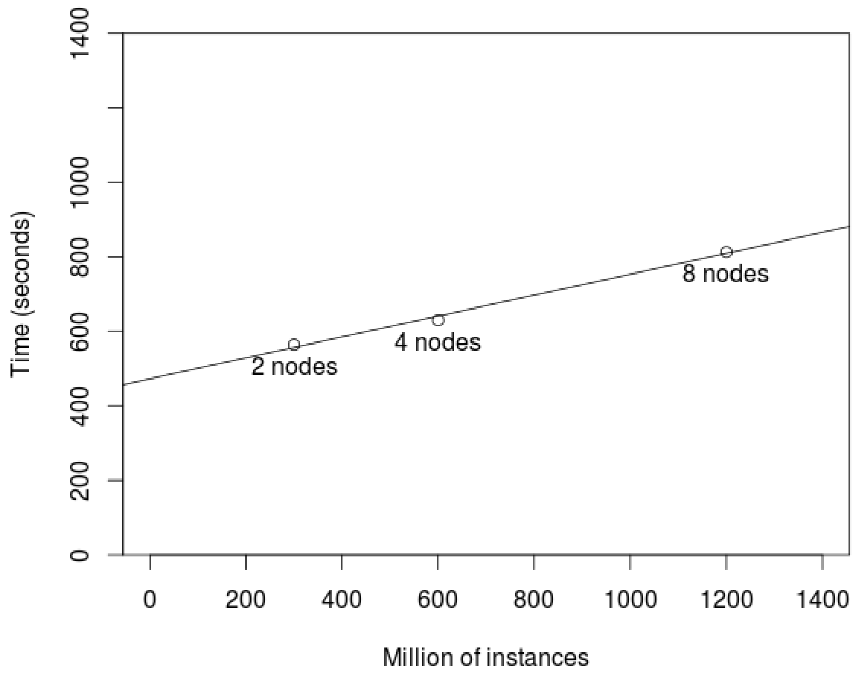


Figure 19: Scalability of the Frequent Terms program in Stratosphere

Chapter 4

Related work

We have categorized the work related to our approach into two groups. The first one (Section 4.1) collects data analysis libraries that facilitate performing data analysis. The second one (Section 4.2) covers the different approaches to do data-intensive computation using R overcoming its main memory limitations.

4.1 Data mining libraries

Popular mining libraries

Libraries for data analysis and mining is a not a novel topic. Some libraries provide already written solutions for **common data mining tasks**. However, even very comprehensive libraries cannot cover all the possible user cases. For instance, *Weka*[HFH+09] (Java) is probably the most well-known tool for data mining on very **limited amounts of data**. Nevertheless, these functionalities are not written to run in parallel and hence Weka is not suitable for training models involving very large datasets. Similarly, sci-kit[PVG+11] is Weka's counterpart in Python, facilitating tasks like classification, regression, clustering, dimensionality reduction, model selection and preprocessing.

Distributed

Some libraries provide ready-to-use **distributed** algorithms that allow the user to execute the most common machine learning tasks on **vast amounts of data**. Hence, they avoid the burden of writing parallel programs on the users. In this category, *Mahout* is probably the most accepted in both academia and business and stands out among the rest of products. In spite of its popularity, *Mahout* intends to cover just a very limited set of problems[OADF11] although its new DSL for linear algebra might open the doors to a number of new algorithms. In a similar way, *MLbase*[KTD+13] provides implementations of some popular machine learning algorithms using Spark[ZCF+10] and facilitates the implementation of them too. More recently, a machine learning project not bound to any execution framework called *Oryx*[ory] was announced. In general, the main problem of these libraries is the difficulty of performing them together with other tasks like data visualization or transformations on small datasets.

In-database

Some databases are adopting a shared-nothing architecture in order to scale-out efficiently. In these architectures, an **in-database machine learning** library like *MADlib*[HRS+12] also seems appropriate for performing data analysis in big amounts of data. *MADlib* includes a respectable number of algorithms covering supervised learning, unsupervised learning and descriptive statistics. Furthermore, the package *PivoltalR*[Qia14] provides an R wrapper for this library making it a solution that can cover cases similar to those that we proposed here: part of the processing in a large-scale facility and part of the processing using R that does not run in parallel. In spite of this advantage, using in-database data mining makes more difficult to implement some algorithms and worse performance if the database has to be accessed frequently.

4.2 Data intensive computation using R

Some efforts have aroused in the last years with the purpose of meeting the demand of the R community for analytics on **large amounts of data**. Many packages and tools have attempted to facilitate the creation of parallel programs using R[SME+09]. Nevertheless, we are going to limit our review to those approaches that offer automatic parallelization, focused on data-intensive computation and paying special attention to those that allow large-scale computation. We classify these approaches in the following groups.

4.2.1 External memory

To overcome the memory limitation, some packages implement external memory algorithms. Most of them are available on the CRAN[Hor12b] repository. Unfortunately, these packages cannot scale-out data processing and hence it is not feasible to run them in a cluster.

- biglm: implements linear models for external memory data. Linear models explain the dependence of one variable on another and are hence used to forecast the value of a variable.

- bigmemory: for massive external memory matrices. It is used jointly with specific packages that provide analytics for these matrices like biganalytics.

- ff: for big data structures that use standard R datatypes. It maps transparently a disk structure to RAM creating the illusion of working with an in-memory structure.

- foreach: scale-out computation but only for parallel tasks that do not require communication among the different nodes. Therefore, it is suitable only for embarrassingly parallel problem.

The *RevoScaleR*[rev] package provides a collection of statistical algorithm that could run in a single machine or in a cluster. These algorithms use *xdf* files. In order to use these algorithms, we have to store the information in this file format and use the pertinent interface to access and manipulate it. Nevertheless, this package also allows users to execute the same operations using a Hadoop cluster.

4.2.2 Divide and recombine

Some packages (*trmr*[rev], *mapReduce*, *Hadoop Interative*[IF] and *RHIPE*[GHR+12]) give the user the possibility to write code in R that runs on a data cluster using the MapReduce programming model. The resulting programs are very similar to those that could be written in other interfaces for different languages (in Java or Scala). The code written in R is translated to Java bytecode in order to run it on the cluster.

Therefore, it is necessary to write programs using the MapReduce programming model. This solution facilitates writing code for tackling all the problems that fit this divide and recombine approach. However, it demands the user to be **trained** to apply the divide and recombine technique. As we have mentioned before (Chapter 2, Section 2.2), using these programming models is a cumbersome process for most analysts. Our approach precisely aims to avoid this.

4.2.3 Query languages

Query languages and their relation with KDD

Some packages offer the possibility of doing data analysis using a query language within R. This is possible, for instance, in NoSQL databases like *MonetDB* (MonetDB.R[MLD]) or HBase[rev] but also in a big data platform like Hadoop (*Ricardo*[DSB$^+$10]). Query languages are a great tool for the first step of the KDD process (discussed in Section 2.3) and good enough for data pre-processing and processing but it is not expressive enough for data mining and knowledge discovery.

Lack of expressivity is not the only drawback

This alternative has three important drawbacks: the user must be skilled in using different language, queries languages are expressively more limited for data analysis[GML14] and queries must produce a query that fits in memory. In order to reduce the memory limitations of this technique, it is possible to use share memory techniques that avoid copying all the retrieved elements of a query to the R environment[GLW$^+$11].

4.2.4 Distributed collection manipulation

Description of the group

Some approaches to data intensive computation introduce a new level of **abstraction** based on distributed collections of objects. Programmers can manipulate these collections with a **limited set of operators** and these operations are automatically distributed. This solution facilitates the implementation of machine learning algorithms but forces the user to use a different programming model (similarly to the packages described in the subsection 4.2.2).

Some examples

We have already introduced *foreach*, a package for scale-out computation on arrays, but only for parallel tasks that do not require communication among different nodes. *Presto*[VBR$^+$13] extends the R programming language with a few clauses, including a *foreach* construct, to allow the programmer to parallelize array computation. Last but not least, *SparkR* goes one step beyond. It introduces the RDD programming model[ZCD$^+$12], which allows the user to perform parallel operations on collections.

Chapter 5

Conclusions and future work

This work has studied the viability of an approach based on the use of ready-to-use algorithms to write programs that can run in both a distributed environment and a single machine using the R language (for the nonparallel parts) and the Stratosphere platform (for the distributed parts). In this chapter we will discussed the findings and contributions of this study among with and some limitations identified.

5.1 Conclusions

Contributions

All the proposed goals have been effectively reached resulting in the four expected deliverables. First, we have defined and explained precisely the functionalities needed in the prospective library (Section 3.2). Second, we have defined those functions that we have considered appropriate based on our research, in order to facilitate the manipulation of files stored using HDFS. Third, we have implemented another module that communicates with the cluster making use of the already existing Stratosphere client. Last but not least, we have implemented, in order to proof the concepts presented here, a PMML classificator that distributes the evaluation of unlabelled instances among the different instances of the cluster, an algorithm to select the most frequent terms in a corpus and the necessary wrapper functions for them and the already existing KMeans program, allowing the users to use them using R. In addition to these deliverables, we have tested the solution and analysed the results of the mentioned tests, in pursuance of proving the viability of this solution.

It seems impossible to keep the single-machine programs as they are and execute them in a distributed system. However, we were able to write code very similar to the original one that efficiently run in the cluster without heavily modifying the syntax. In any case, the complexity of these modifications are trivial compared to the complexity of writing code using the MapReduce paradigm or the PACT operators.

Evaluation of the examples

After executing the tests described in Section 3.5, we have empirically proven that the solution of executing the data processing in the clustered with Stratosphere is:

Performance evaluation

- competitive and even faster than native R programs thanks to the pipelining for every parallelizable programs in the same (small) file size range,
- competitive with R for data mining tasks with a lot of iterations in the same file size range.
- able to process files of a volume that is inaccessible for R, and
- able to scale to gigabyte level without significant loss.

5.2 Future Work

We have intentionally selected only a small number of algorithms to include in the design of the library. However, the already available implementations usually are not ready to work with any kind of input data, they are not type-agnostic, which can produce frustration in the user trying to apply the technique to his data. The algorithms should be really ready-to-use and be able to work with any number of dimensions and any type of data. In addition, machine learning libraries are rapidly evolving and constantly embracing new techniques and algorithm, for instance to include supervised machine learning techniques known as structured learning[MB14]. These techniques follow the principles of approaches for classification and regression, that predict a label or a number, and apply them to predicting any kind of object; which consists in maximizing a given objective function[NL11]. Furthermore, we believe that providing linear algebra primitives would also help users to implement their own statistical algorithms based on them as in the high-level language SystemML[GKP^{+}11].

Improve the library

One of the weak points of this approach is the lack of functionalities to cover all the possible needs of the pre-processing and transforming steps of the KDD process (see Section 2.3). Nevertheless, we believe this weakness can be solved by using an hybrid approach where the user can create these easy-to-distribute programs with tools like those presented in Subsection 4.2.2. In this way, the library would cover those tasks that are difficult to parallelize and the divide and recombine technique would allow the user to program by himself any easy-to-parallelize task. In combination with a query language like those presented in Subsection 4.2.3, every single step of the KDD process could be covered with a suitable solution to apply the KDD process with large amounts of data.

Hybrid approaches

We have previously talked about how sampling can be used for evaluating the solutions that the machine learning algorithms provide. However, many scenarios might need performing these evaluations on large datasets for the sake of accuracy or for different reasons. Regarding this situation, we believe that large-scale libraries should start including algorithms like Dunn index for clustering or the computation of the area under the ROC curve[Bra97b].

Distributed evaluation

Currently, in order to integrate your own Stratosphere programs in this solution, it is necessary to not only include the JAR file but also include a smart portion of code to create an R interface for it. As we have mentioned in 3.1, it could be possible to include a component that automatically generates an R wrapper for any Stratosphere program. This component would improve the extensibility of the tool and also low the entry barrier to new contributions. Besides that point, we believe the architecture is not a bottleneck for the current growth of the software and adding more components would only over-engineer the solution.

Improvements in the architecture

Bibliography

[AAA+14] Daniel Abadi, Rakesh Agrawal, Anastasia Ailamaki, Magdalena Balazinska, Philip A. Bernstein, Michael J. Carey, Surajit Chaudhuri, Jeffrey Dean, AnHai Doan, Michael J. Franklin, Johannes Gehrke, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, H.V. Jagadish, Donald Kossmann, Samuel Madden, Sharad Mehrotra, Tova Milo, Jeffrey F Naughton, Raghu Ramakrishnan, Volker Markl, Christopher Olston, Chin Beng Ooi, R Christopher, Dan Suciu, Michael Stonebraker, Todd Walter, and Jennifer Widom. The beckman report on database research. *ACM SIGMOD Record*, 43(3), September 2014. To appear.

[ABE+14] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinlnder, MatthiasJ. Sax, Sebastian Schelter, Mareike Hger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. *The VLDB Journal*, pages 1–26, 2014.

[AEH+11] Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke. MapReduce and PACT - comparing data parallel programming models. In *Proceedings of the 14th Conference on Database Systems for Business, Technology, and Web (BTW)*, BTW 2011, pages 25–44, Bonn, Germany, 2011. GI.

[AG13] W. Lin M. Zeller A. Guazzelli, T. Jena. Extending the naive bayes model elementin pmml: Adding support for continuous input variables. In *Proceedings of the 19th ACM SIGKDD*

Conference on Knowledge Discovery and Data Mining. ACM, 2013.

[BB08] Enrico Blanzieri and Anton Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.

[BCD+09] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel, and Bernd Wiswedel. Knime - the konstanz information miner. *SIGKDD Explorations*, 11(1), 2009.

[BKSE12] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. *CoRR*, abs/1209.5145, 2012.

[BL14] K. Bache and M. Lichman. Uci machine learning repository. 2014 `http://archive.ics.uci.edu/ml` [Online; accessed 26-Jul-2014].

[BP98] James C Bezdek and Nikhil R Pal. Some new indexes of cluster validity. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(3):301–315, 1998.

[Bra97a] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[Bra97b] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[BTH13] Souhaib Ben Taieb and Rob J Hyndman. A gradient boosting approach to the kaggle load forecasting competition. *International Journal of Forecasting*, 2013.

[CCM+00] PA Castillo, J Carpio, JJ Merelo, Alberto Prieto, V Rivas, and Gustavo Romero. Evolving multilayer perceptrons. *Neural Processing Letters*, 12(2):115–128, 2000.

[CFB08] Mingmin Chi, Rui Feng, and Lorenzo Bruzzone. Classification of hyperspectral remote-sensing data with primal svm for small-sized training dataset problem. *Advances in space research*, 41(11):1793–1799, 2008.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DPC$^+$07] Juan C Dueñas, HA Parada, Félix Cuadrado, Manuel Santillan, and José L Ruiz. Apache and eclipse: Comparing open source project incubators. *Software, IEEE*, 24(6):90–98, 2007.

[DRSG04] Anne De Roeck, Avik Sarkar, and Paul Garthwaite. Frequent term distribution measures for dataset profiling. In *LREC*, 2004.

[DSB$^+$10] Sudipto Das, Yannis Sismanis, Kevin S Beyer, Rainer Gemulla, Peter J Haas, and John McPherson. Ricardo: integrating r and hadoop. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 987–998. ACM, 2010.

[EST$^+$13] Stephan Ewen, Sebastian Schelter, Kostas Tzoumas, Daniel Warneke, and Volker Markl. Iterative parallel data processing with stratosphere: An inside look. In *Proceedings of the 2013 international conference on Management of data*, pages 1053–1056. ACM, 2013.

[ETKM12] Stephan Ewen, Kostas Tzoumas, Moritz Kaufmann, and Volker Markl. Spinning fast iterative data flows. *Proceedings of the VLDB Endowment*, 5(11):1268–1279, 2012.

[Fel98] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[FFFP03] Li Fe-Fei, Robert Fergus, and Pietro Perona. A bayesian approach to unsupervised one-shot learning of object categories. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1134–1141. IEEE, 2003.

[FFFP07] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[FPSS96a] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining : Towards a unifying framework. In *Proceedings of the 2nd international conference on Knowledge Discovery and Data mining (KDD'96)*, pages 82–88. AAAI Press, August 1996.

[FPSS96b] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[FPSS96c] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. The kdd process for extracting useful knowledge from volumes of data. *Commun. ACM*, 39(11):27–34, November 1996.

[GF13] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):13–18, 2013.

[GHM02] Robert L Grossman, Mark F Hornick, and Gregor Meyer. Data mining standards initiatives. *Communications of the ACM*, 45(8):59–61, 2002.

[GHR$^+$12] Saptarshi Guha, Ryan Hafen, Jeremiah Rounds, Jin Xia, Jianfu Li, Bowei Xi, and William S Cleveland. Large complex data: divide and recombine (d&r) with rhipe. *Stat*, 1(1):53–67, 2012.

[GKP$^+$11] Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 231–242. IEEE, 2011.

[GLW+11] Philipp Große, Wolfgang Lehner, Thomas Weichert, Franz Färber, and Wen-Syan Li. Bridging two worlds with rice integrating r into the sap in-memory computing engine. *PVLDB*, 4(12):1307–1317, 2011.

[GML14] Philipp Große, Norman May, and Wolfgang Lehner. A study of partitioning and parallel udf execution with the sap hana database. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, page 36. ACM, 2014.

[HAMA+11] Abdelaali Hassaïne, Somaya Al-Maadeed, Jihad Mohamad Alja'am, Ali Jaoua, and Ahmed Bouridane. The icdar2011 arabic writer identification contest. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1470–1474. IEEE, 2011.

[HD14] Richard Heimann and Nathan Danneman. Social media mining with r. 2014.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[HMV13] Harlan Harris, Sean Murphy, and Marck Vaisman. *Analyzing the Analyzers: An Introspective Survey of Data Scientists and Their Work*. O'Reilly Media, Inc., 2013.

[HNP05] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

[Hor12a] Kurt Hornik. Are there too many r packages? *Austrian Journal of Statistics*, 41(1):59–66, 2012.

[Hor12b] Kurt Hornik. The comprehensive r archive network. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(4):394–398, 2012.

[HRS$^+$12] Joseph M Hellerstein, Christoper Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, et al. The madlib analytics library: or mad skills, the sql. *Proceedings of the VLDB Endowment*, 5(12):1700–1711, 2012.

[HW79] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[IF] Stefan Theussl Ingo Feinerer. Hadoop interactive. `http://cran.at.r-project.org/web/packages/hive/hive.pdf`. Accessed: 2013-11-27.

[JBB14] Alan Jović, Karla Brkić, and Nikola Bogunović. An overview of free software tools for general data mining. In *37th International Convention MIPRO 2014*, 2014.

[jpm] Jpmml source code. `http://github.com/jpmml`. Accessed: 2014-05-07.

[KM06] Lukasz A. Kurgan and Petr Musilek. A survey of knowledge discovery and data mining process models. *Knowl. Eng. Rev.*, 21(1):1–24, March 2006.

[KPHH12] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. Enterprise data analysis and visualization: An interview study. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2917–2926, 2012.

[KTD$^+$13] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[Lei08] Friedrich Leisch. Creating r packages: A tutorial. In Paula Brito, editor, *Compstat 2008 - Proceedings in Computational Statistics*. Physica Verlag, Heidelberg, Germany, 2008.

[Lin08] Jimmy Lin. Scalable language processing algorithms for the masses: A case study in computing word co-occurrence matrices with mapreduce. In *Proceedings of the Conference on Em-*

pirical Methods in Natural Language Processing, pages 419–428. Association for Computational Linguistics, 2008.

[LLC⁺12] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: a survey. *ACM SIGMOD Record*, 40(4):11–20, 2012.

[LLM10] Zhiqiang Liu, Hongyan Li, and Gaoshan Miao. Mapreduce-based backpropagation neural network over large scale mobile data. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 4, pages 1726–1730. IEEE, 2010.

[LS08] Chi-Jung Lu and Stuart W Shulman. Rigor and flexibility in computer-based qualitative research: Introducing the coding analysis toolkit. *International Journal of Multiple Research Approaches*, 2(1):105–117, 2008.

[LWK13] Björn Lohrmann, Daniel Warneke, and Odej Kao. Nephele streaming: stream processing under qos constraints at scale. *Cluster Computing*, pages 1–18, 2013.

[MAEA05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005*, pages 398–412. Springer, 2005.

[Mar14] Volker Markl. Breaking the chains: On declarative data analysis and data independence in the big data era. *Proceedings of the VLDB Endowment*, 7(13), 2014.

[MB14] Andreas C. Müller and Sven Behnke. pystruct - learning structured prediction in python. *Journal of Machine Learning Research*, 15:2055–2060, 2014.

[MBF⁺90] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3(4):235–244, 1990.

[MBR12] Diana Maynard, Kalina Bontcheva, and Dominic Rout. Challenges in developing opinion mining tools for social media. *Proceedings of@ NLP can u tag# usergeneratedcontent*, 2012.

[McK12] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, 2012.

[MHF08] David Meyer, Kurt Hornik, and Ingo Feinerer. Text mining infrastructure in r. *Journal of Statistical Software*, 25(5):1–54, 2008.

[MI00] Malik Magdon-Ismail. No free lunch for noise prediction. *Neural computation*, 12(3):547–564, 2000.

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Mit97] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45, 1997.

[MLD] Hannes Muehleisen, Thomas Lumley, and Anthony Damico. Monetdb.r - monetdb to r connector.

[MPC$^+$93] JJ Merelo, M Patón, Antonio Cañas, Alberto Prieto, and Federico Morán. Optimization of a competitive learning neural network by genetic algorithms. In *New Trends in Neural Computation*, pages 185–192. Springer, 1993.

[NA08] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008.

[NL11] Sebastian Nowozin and Christoph H Lampert. Structured learning and prediction in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[NSR11] Arvind Narayanan, Elaine Shi, and Benjamin IP Rubinstein. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *Neural Networks (IJCNN),* *The 2011 International Joint Conference on*, pages 1825–1834. IEEE, 2011.

[OADF11] Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman. *Mahout in Action*. Manning Publications Co., Manning Publications Co. 20 Baldwin Road PO Box 261 Shelter Island, NY 11964, first edition, 2011.

[ory] Oryx source code. `https://github.com/cloudera/oryx`. Accessed: 2014-03-01.

[OW11] R Wayne Oldford and Adrian Waddell. Visual clustering of high-dimensional data by navigating low-dimensional spaces. In *58th Congress of the International Statistical Institute, Special Topics Session*, volume 57, 2011.

[Pal07] Nikhil Pal. *Advanced techniques in knowledge discovery and data mining*. Springer, 2007.

[PMHGP11] Zoltán Prekopcsák, Gábor Makrai, Tamás Henk, and Csaba Gáspár-Papanek. Radoop: Analyzing big data with rapidminer and hadoop. In *Proceedings of the 2nd RapidMiner Community Meeting and Conference (RCOMM 2011)*, 2011.

[PPM$^+$11] Serguei VS Pakhomov, Ted Pedersen, Bridget McInnes, Genevieve B Melton, Alexander Ruggieri, and Christopher G Chute. Towards a framework for developing semantic relatedness reference standards. *Journal of biomedical informatics*, 44(2):251–265, 2011.

[PRGK14] Ivanilton Polato, Reginaldo Ré, Alfredo Goldman, and Fabio Kon. A comprehensive view of hadoop research-a systematic literature review. *Journal of Network and Computer Applications*, 2014.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Qia14] Hai Qian. Pivotalr: A package for machine learning on big data. *The R Journal*, 6(1):57–67, June 2014.

[RAAQ11] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M Quiroz. Internal versus external cluster validation indexes. *International Journal of computers and communications*, 5(1):27–34, 2011.

[rev] Revolution analytics. `http://www.revolutionanalytics.com/`. Accessed: 2014-01-07.

[RQRSD13] Stefan Richter, Jorge-Arnulfo Quiané-Ruiz, Stefan Schuh, and Jens Dittrich. Towards Zero-Overhead Static and Adaptive Indexing in Hadoop. *VLDB Journal*, 2013.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[SETM13] Sebastian Schelter, Stephan Ewen, Kostas Tzoumas, and Volker Markl. All roads lead to rome: optimistic recovery for distributed iterative data processing. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1919–1928. ACM, 2013.

[SG14] S. Sakr and M. Gaber. *Large Scale and Big Data: Processing and Management*. Taylor & Francis, 2014.

[SLF13] Sherif Sakr, Anna Liu, and Ayman G Fayoumi. The family of mapreduce and large-scale data processing systems. *ACM Computing Surveys (CSUR)*, 46(1):11, 2013.

[SME$^+$09] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. State-of-the-art in parallel computing with r. *Journal of Statistical Software*, 47(1), 2009.

[SP10] Skipper Seabold and Josef Perktold. Statsmodels: econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, pages 57–61, 2010.

[Sto86] Michael Stonebraker. The case for shared nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.

[TdJDN13] Martijn Tennekes, Edwin de Jonge, Piet JH Daas, and Statistics Netherlands. Visualizing and inspecting large datasets with tableplots. *Journal of Data Science*, 11(1):43–58, 2013.

[Tea99] R Core Team. Writing r extensions. *R Foundation for Statistical Computing*, 1999.

[Vav13] Vinod K. Vavilapalli. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proc. SOCC*, 2013.

[VBR$^+$13] Shivaram Venkataraman, Erik Bodzsar, Indrajit Roy, Alvin AuYoung, and Robert S Schreiber. Presto: distributed machine learning and graph processing with sparse matrices. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 197–210. ACM, 2013.

[VDWCV11] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[Ver10] Tobias Verbeke. Hello java world! a tutorial for interfacing tojava archives inside r packages. 2010.

[WF05] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

[Wic07] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007.

[Wic11] Hadley Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1):1–29, 2011.

[Wic14] Hadley Wickham. *Advanced R Programming.* CRC Press, 2014. To appear.

[Wil09] Graham J Williams. Rattle: a data mining gui for r. *The R Journal*, 1(2):45–55, 2009.

[Wil11] Graham J. Williams. *Data Mining with Rattle and R: The art of excavating data for knowledge discovery.* Use R! Springer, 2011.

[WO11] Adrian Waddell and Wayne Oldford. Rnavgraph: A visualization tool for navigating through high-dimensional data spaces. In *Proceedings of the 58th ISI World Statistics Congress, Dublin*, pages 3294–3303, 2011.

[Wol96] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[YW02] Ying Yang and Geoffrey I Webb. A comparative study of discretization methods for naive-bayes classifiers. In *Proceedings of PKAW*, volume 2002. Citeseer, 2002.

[YW09] Ying Yang and Geoffrey I Webb. Discretization for naive-bayes learning: managing discretization bias and variance. *Machine learning*, 74(1):39–74, 2009.

[ZC13] Yanchang Zhao and Yonghua Cen, editors. *Data Mining Applications with R.* Academic Press, Elsevier, December 2013.

[ZCD$^+$12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[ZCF$^+$10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[ZGW09] Michael Zeller, Wen-Ching Lin Alex Guazzelli, and Graham Williams. PMML: An Open Standard for Sharing Models . *The R Journal*, 1(1):60–65, jun 2009.

[Zha04] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.

[Zic13] Roberto V. Zicari. Big data: Challenges and opportunities. In Rajendra Akerkar, editor, *Big Data Computing*, page 564. Chapman and Hall/CRC, October 2013.

[ZMH09] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Cloud Computing*, pages 674–679. Springer, 2009.

Appendix A

PMML model example

.

```
<?xml version="1.0"?>
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.
    org/PMML-4_1␣http://www.dmg.org/v4-1/pmml-4-1.xsd">
 <Header copyright="Copyright␣(c)␣2014␣jlpino" description="NaiveBayes␣
     Model">
  <Extension name="user" value="jlpino" extender="Rattle/PMML"/>
  <Application name="Rattle/PMML" version="1.4"/>
  <Timestamp>2014-04-27 21:48:49</Timestamp>
 </Header>
 <DataDictionary numberOfFields="6">
  <DataField name="Species" optype="categorical" dataType="string">
   <Value value="setosa"/>
   <Value value="versicolor"/>
   <Value value="virginica"/>
  </DataField>
  <DataField name="Sepal.Length" optype="continuous" dataType="double"/>
  <DataField name="Sepal.Width" optype="continuous" dataType="double"/>
  <DataField name="Petal.Length" optype="continuous" dataType="double"/>
  <DataField name="Petal.Width" optype="continuous" dataType="double"/>
  <DataField name="DiscretePlaceHolder" optype="categorical" dataType="
      string">
   <Value value="pseudoValue"/>
  </DataField>
 </DataDictionary>
 <NaiveBayesModel modelName="naiveBayes_Model" functionName="classification
     " threshold="0.001">
  <MiningSchema>
   <MiningField name="Species" usageType="predicted"/>
   <MiningField name="Sepal.Length" usageType="active"/>
   <MiningField name="Sepal.Width" usageType="active"/>
   <MiningField name="Petal.Length" usageType="active"/>
   <MiningField name="Petal.Width" usageType="active"/>
   <MiningField name="DiscretePlaceHolder" usageType="active"
       missingValueReplacement="pseudoValue"/>
  </MiningSchema>
  <Output>
   <OutputField name="Predicted_Species" feature="predictedValue"/>
   <OutputField name="Probability_setosa" optype="continuous" dataType="
       double" feature="probability" value="setosa"/>
   <OutputField name="Probability_versicolor" optype="continuous" dataType=
       "double" feature="probability" value="versicolor"/>
   <OutputField name="Probability_virginica" optype="continuous" dataType="
       double" feature="probability" value="virginica"/>
  </Output>
  <BayesInputs>
   <Extension>
    <BayesInput fieldName="Sepal.Length">
     <TargetValueStats>
      <TargetValueStat value="setosa">
       <GaussianDistribution mean="5.006" variance="0.124248979591837"/>
      </TargetValueStat>
      <TargetValueStat value="versicolor">
       <GaussianDistribution mean="5.936" variance="0.266432653061224"/>
      </TargetValueStat>
      <TargetValueStat value="virginica">
       <GaussianDistribution mean="6.588" variance="0.404342857142857"/>
      </TargetValueStat>
     </TargetValueStats>
    </BayesInput>
   </Extension>
   <Extension>
    <BayesInput fieldName="Sepal.Width">
     <TargetValueStats>
      <TargetValueStat value="setosa">
       <GaussianDistribution mean="3.428" variance="0.143689795918367"/>
      </TargetValueStat>
      <TargetValueStat value="versicolor">
       <GaussianDistribution mean="2.77" variance="0.0984693877551021"/>
      </TargetValueStat>
      <TargetValueStat value="virginica">
       <GaussianDistribution mean="2.974" variance="0.104004081632653"/>
      </TargetValueStat>
     </TargetValueStats>
    </BayesInput>
   </Extension>
   <Extension>
    <BayesInput fieldName="Petal.Length">
     <TargetValueStats>
      <TargetValueStat value="setosa">
       <GaussianDistribution mean="1.462" variance="0.0301591836734694"/>
      </TargetValueStat>
      <TargetValueStat value="versicolor">
       <GaussianDistribution mean="4.26" variance="0.220816326530612"/>
      </TargetValueStat>
      <TargetValueStat value="virginica">
       <GaussianDistribution mean="5.552" variance="0.304587755102041"/>
      </TargetValueStat>
     </TargetValueStats>
    </BayesInput>
   </Extension>
   <Extension>
    <BayesInput fieldName="Petal.Width">
     <TargetValueStats>
      <TargetValueStat value="setosa">
```

```
       <GaussianDistribution mean="0.246" variance="0.0111061224489796"/>
      </TargetValueStat>
      <TargetValueStat value="versicolor">
       <GaussianDistribution mean="1.326" variance="0.0391061224489796"/>
      </TargetValueStat>
      <TargetValueStat value="virginica">
       <GaussianDistribution mean="2.026" variance="0.0754326530612245"/>
      </TargetValueStat>
     </TargetValueStats>
    </BayesInput>
   </Extension>
   <BayesInput fieldName="DiscretePlaceHolder">
    <PairCounts value="pseudoValue">
     <TargetValueCounts>
      <TargetValueCount value="setosa" count="50"/>
      <TargetValueCount value="versicolor" count="50"/>
      <TargetValueCount value="virginica" count="50"/>
     </TargetValueCounts>
    </PairCounts>
   </BayesInput>
  </BayesInputs>
  <BayesOutput fieldName="Species">
   <TargetValueCounts>
    <TargetValueCount value="setosa" count="50"/>
    <TargetValueCount value="versicolor" count="50"/>
    <TargetValueCount value="virginica" count="50"/>
   </TargetValueCounts>
  </BayesOutput>
 </NaiveBayesModel>
</PMML>
```

Appendix B

Hyperlinks to the R packages referred

Package	URL
apply	http://cran.r-project.org/web/packages/apply/
BayesTree	http://cran.r-project.org/web/packages/BayesTree/
BGLR	http://cran.r-project.org/web/packages/BGLR/
bigRR	http://cran.r-project.org/web/packages/bigRR/
canopy	http://github.com/lefthandedgoat/canopy
CORElearn	http://cran.r-project.org/web/packages/CORElearn/
DiscriMiner	http://cran.r-project.org/web/packages/DiscriMiner/
e1071	http://cran.r-project.org/web/packages/e1071/
elasticnet	http://cran.r-project.org/web/packages/elasticnet/
fanny	http://cran.r-project.org/web/packages/fanny/
foreach	http://cran.r-project.org/web/packages/foreach/
fpc	http://cran.r-project.org/web/packages/fpc/
gbm	http://cran.r-project.org/web/packages/gbm/
GeneReg	http://cran.r-project.org/web/packages/GeneReg/
glm	http://cran.r-project.org/web/packages/glm/
glmnet	http://cran.r-project.org/web/packages/glmnet/
glmpath	http://cran.r-project.org/web/packages/glmpath/
HiddenMarkov	http://cran.r-project.org/web/packages/HiddenMarkov/
kernlab	http://cran.r-project.org/web/packages/kernlab/
kkmeans	http://cran.r-project.org/web/packages/kkmeans/
lsmeans	http://cran.r-project.org/web/packages/lsmeans/

Package	URL
LVQTools	http://cran.r-project.org/web/packages/LVQTools/
ncvreg	http://cran.r-project.org/web/packages/ncvreg/
nnet	http://cran.r-project.org/web/packages/nnet/
plyr	http://cran.r-project.org/web/packages/plyr/
pmml	http://cran.r-project.org/web/packages/pmml/
qdap	http://cran.r-project.org/web/packages/qdap/
randomForest	http://cran.r-project.org/web/packages/randomForest/
rattle	http://cran.r-project.org/web/packages/rattle/
rhdfs	http://cran.r-project.org/web/packages/rhdfs/
RJDBC	http://cran.r-project.org/web/packages/RJDBC/
rminer	http://cran.r-project.org/web/packages/rminer/
RXQuery	http://github.com/omegahat/RXQuery/
Rxshrink	http://cran.r-project.org/web/packages/Rxshrink/
sqldf	http://cran.r-project.org/web/packages/sqldf/
stats	http://cran.r-project.org/web/packages/stats/
stream	http://cran.r-project.org/web/packages/stream/
tabplot	http://cran.r-project.org/web/packages/tabplot/

Appendix C

Stratosphere-client dependency tree

```
eu.stratosphere:stratosphere-clients
|__eu.stratosphere:stratosphere-core
|  |__org.apache.commons:commons-lang3
|__eu.stratosphere:stratosphere-runtime
   |  |__commons-cli:commons-cli
   |  |__com.amazonaws:aws-java-sdk
   |  |  |__org.apache.httpcomponents:httpclient
   |  |  |__org.apache.httpcomponents:httpcore
   |  |  |__org.codehaus.jackson:jackson-core-asl
   |  |  |__javax.mail:mail
   |  |  |__javax.activation:activation
   |  |__stax:stax-api
   |__org.apache.hadoop:hadoop-core
   |__xmlenc:xmlenc
   |__com.sun.jersey:jersey-core
   |__com.sun.jersey:jersey-json
   |  |__org.codehaus.jettison:jettison
   |  |__com.sun.xml.bind:jaxb-impl
   |  |  |__javax.xml.bind:jaxb-api
   |  |  |__javax.xml.stream:stax-api
   |  |__org.codehaus.jackson:jackson-jaxrs
   |  |__org.codehaus.jackson:jackson-xc
   |__com.sun.jersey:jersey-server
```

```
        commons-httpclient:commons-httpclient
        org.apache.commons:commons-math
        commons-configuration:commons-configuration
            commons-collections:commons-collections
            commons-lang:commons-lang
            commons-digester:commons-digester
                commons-beanutils:commons-beanutils
            commons-beanutils:commons-beanutils-core
        commons-net:commons-net
        org.mortbay.jetty:jetty
        org.mortbay.jetty:jetty-util
        tomcat:jasper-runtime
        tomcat:jasper-compiler
        org.mortbay.jetty:jsp-api-2.1
            org.mortbay.jetty:servlet-api-2.5
        org.mortbay.jetty:jsp-2.1
            ant:ant
        commons-el:commons-el
        net.java.dev.jets3t:jets3t
        hsqldb:hsqldb
        oro:oro
        org.eclipse.jdt:core
        org.codehaus.jackson:jackson-mapper-asl
    eu.stratosphere:stratosphere-compiler
        eu.stratosphere:stratosphere-java
    org.eclipse.jetty:jetty-server
            org.mortbay.jetty:servlet-api
            org.eclipse.jetty:jetty-continuation
        org.eclipse.jetty:jetty-http
        org.eclipse.jetty:jetty-io
        org.eclipse.jetty:jetty-util
    org.eclipse.jetty:jetty-security
    org.eclipse.jetty:jetty-servlet
    commons-fileupload:commons-fileupload
    commons-io:commons-io
    commons-logging:commons-logging
    log4j:log4j
    commons-codec:commons-codec
```

```
│
├── com.google.guava:guava
├── junit:junit
├── org.mockito:mockito-all
├── org.powermock:powermock-module-junit4
│   ├── org.powermock:powermock-module-junit4-common
│   ├── org.powermock:powermock-core
│   │   └── org.javassist:javassist
│   ├── org.powermock:powermock-reflect
│   └── org.objenesis:objenesis
├── org.powermock:powermock-api-mockito
│   └── org.powermock:powermock-api-support
└── org.hamcrest:hamcrest-all
```