UNIVERSITÉ LAVAL

Personalized Large Scale Classification of Public Tenders on Hadoop

Mathieu Dumoulin

Maîtrise en informatique

Québec, Canada

Résumé

Ce projet a été réalisé dans le cadre d'un partenariat entre Fujitsu Canada et Université Laval. Les besoins du projets ont été centrés sur une problématique d'affaire définie conjointement avec Fujitsu. Le projet consistait à classifier un corpus d'appels d'offres électroniques avec une approche orienté big data. L'objectif était d'identifier avec un très fort rappel les offres pertinentes au domaine d'affaire de l'entreprise. Après une séries d'expérimentations à petite échelle qui nous ont permise d'illustrer empiriquement (93% de rappel) l'efficacité de notre approche basé sur l'algorithme BNS (Bi-Normal Separation), nous avons implanté un système complet qui exploite l'infrastructure technologique big data Hadoop. Nos expérimentations sur le système complet démontrent qu'il est possible d'obtenir une performance de classification tout aussi efficace à grande échelle (91% de rappel) tout en exploitant les gains de performance rendus possible par l'architecture distribuée de Hadoop.

Abstract

This project was completed as part of an innovation partnership with Fujitsu Canada and Université Laval. The needs and objectives of the project were centered on a business problem defined jointly with Fujitsu. Our project aimed to classify a corpus of electronic public tenders based on state of the art Hadoop big data technology. The objective was to identify with high recall public tenders relevant to the IT services business of Fujitsu Canada. A small scale prototype based on the BNS algorithm (Bi-Normal Separation) was empirically shown to classify with high recall (93%) the public tender corpus. The prototype was then re-implemented on a full scale Hadoop cluster using Apache Pig for the data preparation pipeline and using Apache Mahout for classification. Our experimentation show that the large scale system not only maintains high recall (91%) on the classification task, but can readily take advantage of the massive scalability gains made possible by Hadoop's distributed architecture.

Contents

List of Tables

List of Figures

To my parents and brothers, my directors François Laviolette and Luc Lamontagne as well as Andriy Burkov, Sébastien Paquet and Albert Craig of Fujitsu Canada who believed in me from the start. To Shigemi and all my children who gave me purpose and motivation.

“Listening to the data is important, but so is experience and intuition. After all, what is intuition at its best but large amounts of data of all kinds filtered through a human brain rather than a math model?”

Steve Lohr

Thanks

This project would not have been possible without the generous support from many people and organisations.

From Université Laval, I thank my director François Laviolette and co-director Luc Lamontagne for their steadfast support, encouragement. They have been mentors and friends. I will prize the relationship we have built over the course of this project for the rest of my life.

Also from Université Laval, I have to thank Mario Marchand who convinced me to choose to do my Master's degree in Québec.

Special thanks go to Fujitsu Canada's Albert Craig and Guy Michaud. Albert's leadership and drive were contagious. I learned a lot from a business point of view from him. Guy's input helped a lot at the beginning of the process. He was always patient and kind in equal parts. We would not have understood the problem as well without his help.

I also thank former members of the Fujitsu Innovation Center's, Sébastien Paquet and Andriy Burkov, who initially believed in me.

Finally thanks to the Natural Sciences and Engineering Research Council of Canada and the Fonds de recherche du Québec – Nature et technologies as well as Fujitsu Canada for their financial support through the BMP Innovation scholarship program.

Chapter 1

Introduction

Big data is about building new analytic applications based on new types of data, in order to better serve your customers and drive a better competitive advantage
- David McJannet, Hortonworks

1.1 The Age of Big Data

Since 2012, big data considerably matured. From 2006-2010, big data was on the bleeding edge of technology, mastered only by the technical wizards of the internet giants like Google, Yahoo and Facebook. Now, the tools and technologies have evolved to the point where large-scale data analytics (e.g. big data analytics) are in use at nearly all Fortune 500 companies, and in many internet startups too.

Big data is certainly a technological problem. In that sense, big data is about how to store and process massive amounts of information that can't be handled by traditional relational databases.

More importantly, the true benefit of big data lies in the ability to gain new insights only possible through the analysis of massive datasets. In this sense, big data can make possible new kinds of data-driven decisions that be turned into a competitive advantage for companies, better outcomes for patients at hospitals or better services to citizens.

Google is one of the first companies to master big data. From its very foundation, Google's big data processing mastery has served it as an unrivaled competitive advantage that has seen them out-compete rivals like Altavista, Yahoo and Microsoft in the web search and web advertising spaces.

Since then, other technological leaders such as Amazon, Facebook, Linkedin, Twitter and many others have followed building ever more innovative big data systems, scaling their rapidly expanding services to global scale at a previously unheard of speed.

There is an ever widening variety of potential applications that can benefit enormously from big data analytics. One such problem is large-scale unstructured data analytics, or text analytics. In this case, the datasets are web pages, emails, reports and other textual documents.

Current research often focuses on narrow aspects of text analytics as machine learning algorithms or improvements of the distributed processing system. There is very little published research showing all the working parts of a complete large-scale text analytics pipeline as it might be implemented in a real-world company. Furthermore, current research typically shows results based on toy or artificial datasets with limited real-world applicability.

There would be much to be learned from trying to solve a real big data business problem using a real-world dataset on Apache Hadoop, an emerging standard software infrastructure for big data processing.

This masters thesis presents the results of such a project, the result of an innovative partnership between Université Laval and the Fujitsu Innovation Center.

1.1.1 Initiating Big Data R&D at Université Laval and Fujitsu

Our project was the first project for both the Fujitsu Innovation Center as well as the Computer Science Department of Université Laval on big data based on Apache Hadoop. In fact, it marked the launch of the Big Data Laboratory initiative at Fujitsu Innovation Center. We are also the precursors to a proposed new research center on big data at Université Laval.

We successfully tackled two main challenges:

- Using machine learning, we brought a solution to a previously non-working prototype application developed internally at Fujitsu called Sieve.
- We built a fully functional Hadoop-based Text classification system based on that solution so it could scale to fill the needs of a global enterprise as Fujitsu.

In the rest of this chapter, we will explain the context of our master's project, starting with the Fujitsu Innovation Center. We will then explain how our project, funded by a BMP Innovation scholarship, fits within the FIC's mission. Then, we will introduce the business problem that led to the development of the Sieve application. Finally, we tie together the business problem with machine learning and big data, by showing how the business problem can scale globally for a large enterprise such as Fujitsu.

1.2 The Fujitsu Innovation Center (FIC)

Fujitsu is a leading Japanese information and communication technology (ICT) company offering a full range of technology products, solutions and services. Approximately 170,000 Fujitsu people support customers in more than 100 countries. Renowned for innovation in software and hardware applications, Fujitsu holds over 97,000 patents worldwide.

Fujitsu Innovation Center in Quebec City is one of Fujitsu's newest laboratories, opening in November 2011. It's a collaborative space where business partners and customers are invited to work with Fujitsu specialists. Together, they put new ideas into practice to build innovative business solutions.

The very first of its kind in Canada, the FIC is the results of significant investments by Fujitsu as well as Quebec's Ministry of Economic Development, Innovation and Exports as part of the mobilizing ÉcoloTIC project, which stems from the *Stratégie québécoise de la recherche et de l'innovation 2010-2013*.

Fujitsu North-America (FNA) as well as Fujitsu Japan has shown interest in a project that could integrate Fujitsu's server hardware, virtualization and cloud expertise and apply them to the emerging fields of big data and machine learning.

For the FIC, our project was a prime opportunity to explore the new space of big data from a technological point of view.

1.3 Collaboration for Big Data Innovation

This masters project, funded by the BMP Innovation scholarship, is a great fit with the mission of the FIC. It marks the launch of a Big Data research laboratory and brings together the expertise of Fujitsu scientists Andriy Burkov ph.D., Sebastien Paquet ph.D. and Guy Michaud ph.D. with two scientists of the *Group d'Apprentissage Automatisé de Laval* (GRAAL) Francois Laviolette and Luc Lamontagne. It also leverages the enterprise hardware resources available at Fujitsu's Quebec City office.

The BMP Innovation Program
Offered jointly by NSERC and the FRQNT, the BMP Innovation program offers scholarships to students enrolled in a master's or doctoral program and interested in undertaking research in natural sciences and engineering, through a university-business partnership.

This project was awarded a prize in the Young Innovator - BMP Innovation 2013 category by the *Association pour le développement de la recherche et de l'innovation du Québec* (ADRIQ) and the NSERC[1].

Association pour le développement de la recherche et de l'innovation du Québec (ADRIQ)
For the past 35 years, the ADRIQ has led a vast business network that is both unique and influential to support technological innovation in Québec and foster partnerships between decision-makers. ADRIQ addresses and disseminates the key concerns and issues of the innovative companies and organizations it represents to develop favorable factors and conditions that are conductive to their success.

1.4 The Business Problem

1.4.1 Fujitsu Canada in early 2012

As we mentioned, this project is the result of a strategic decision by the Fujitsu Innovation Center to foster collaborative innovation by initiating a partnership project through the BMP Innovation scholarship program.

In early 2012, Fujitsu Canada was already interested in the big data field but had no experts in this new technology on staff. The staff scientists, already busy with their own R&D work proposed to use the BMP Innovation program to build up big data expertise. The potential customers for big data technology were the National Defence scientists of the Valcartier military base, large insurance companies of Quebec City and the Québec government.

With the general research conditions settled, an existing business problem was chosen to solve using big data technology. This was an ongoing in-house project that had proven to be hard to solve using the existing enterprise programming skills of the Quebec City office of Fujitsu Canada. Specifically, the project's goal was to automatically discover public tenders for government contracts relevant to Fujitsu's IT consulting business. The tender discovery process can now be explained in some detail.

[1] `http://www.adriq.com/fr/article/d%C3%A9couvrez-laur%C3%A9ats-des-prix-innovation-2013`

1.4.2 The Public Tender Bidding Process at Fujitsu

Fujitsu's Quebec City office does IT consulting for business as well as for Canadian federal and provincial governmental organizations such as Revenue Quebec and Defence Research and Development Canada (DRDC) Valcartier. In most cases, Fujitsu wins governmental IT service contracts by bidding on public tenders. These tenders are found on web sites that provide electronic tendering services such as MERX (`www.merx.com`).

On a regular basis, these web sites must be manually checked for public tenders in the IT services field that Fujitsu might be interested to bid on. In fact, several Fujitsu managers go through the tenders looking for projects that are of special interest for their areas of expertise, such as defence R&D projects, GIS projects, IT integration projects and so on. Projects are selected based on the nature of the work; Fujitsu does only software IT services, but other factors also come into play such as whether the necessary skills are available among the IT consultants currently available to work, the project deadlines and project budget and so on.

This fastidious work is an inefficient use of highly skilled IT experts' time, so the decision was taken to try and build an in-house application that could help automate the process.

1.4.3 Automating the Discovery of Relevant Tenders with Sieve

The project was called Sieve and Guy Michaud was one of its leaders, serving as the target business user of the project. The project was developed from 2010 to 2012 through the combined efforts of various Fujitsu developers and several interns.

The project was fairly ambitious: crawl web sites, extract information, compare it with existing employee resumes and display a customized and ranked list of relevant public tenders to a user.

Lets examine each functionality in turn:

- Sieve gathers its data using a web crawler that automatically extracts the information from the public tender web pages of the Research and Development category from MERX and saves it in a relational database.

- The tender text content is compared with the resumes of Fujitsu's Quebec City employee skills profiles (their consultant profiles) so as to identify relevant tenders.

- The tenders are displayed in a web front-end, ranked by relevance in some way.

- The user can give feedback to the system on whether a given public tender is in fact relevant or not with a “thumbs up” and “thumbs down” button next public tender in the list.

We were given a demonstration of Sieve, and all features listed above were indeed functioning correctly. But there was a problem with the lists of tenders that Sieve was showing to the users: the list was a seemingly randomized selection of public tenders, most of which had nothing to do with Fujitsu at all. This was a critical flaw that led to the project’s cancellation.

1.4.4 Why Sieve Failed

Ultimately, the Sieve project was abandoned. The main reason being that the application could not produce good results. Indeed, even on the very first page of results, the listing of public tenders consisted in large part of completely irrelevant tenders. Guy Michaud candidly said the results were useless for him. He was still gathering the information directly from the web sites. As explained in later sections, the application was showing poor precision (subsection 3.5.3).

Guy did not trust the application results at all for another critical reason: There was no guarantee that all relevant notices would be displayed, regardless of ranking. This is problematic because if a Fujitsu manager doesn’t see a relevant public tender, then Fujitsu can’t bid for the contract, meaning a direct financial loss for the company. This, in turn is a problem of low recall (subsection 3.5.3).

In retrospect, we believe that Sieve failed because it was built as an IT project. Fujitsu tried to solve this problem by building a software application, and solving software application problems.

The real business value of the project, however, was showing a list of potentially relevant public tenders to a Fujitsu manager. That was the problem that should have been solved first.

This is an important lesson that strongly influenced how we would approach our problem.

1.4.5 Sieve: The Starting Point of the Project

The business problem that Sieve attempted to solve is a real one at Fujitsu. Our project was an ideal chance to try and solve this problem with a new approach.

We believed that the ranking relevancy problem could be solved successfully using supervised machine learning techniques.

1.5 Project Objectives

Our project essentially comes down to trying to answer two questions:

1. Can we address Sieve's shortcomings using machine Learning?
2. If such a solution can be found, could it be implemented as part of a full text processing pipeline on a Apache Hadoop cluster?

Each question will be treated in turn in this section.

1.5.1 Solve Sieve's Problem with Machine Learning

As mentioned earlier, the fundamental problem of the Sieve prototype was that it didn't produce a list of relevant results to its users. This is compounded by the issue of trust, as Fujitsu's business user had no confidence that all relevant public tenders would be displayed in the first few pages. Since missing what would have been a relevant tender has a negative economic cost to Fujitsu, Sieve's potential users could not rely on it for their work.

We decided to try and solve this problem using standard text classification techniques. The new solution would try and predict whether a public tender is relevant or not using the "notice" field of the public tender information.

The business requirements put special constraints on the machine learning prototype. Our business user wants to make sure that all possible relevant notices are displayed, even if it means that some irrelevant notices are displayed as well. In addition to tuning the classification for this requirement, we provided solid experimental metrics of classification performance to build trust in the system.

1.5.2 Make it Scalable with Hadoop

Here, the objectives of the project are much more technical in nature. Indeed, true to the innovation mandate of the FIC, our project would have to explore and prototype new emerging big data technologies.

Once we have a running machine learning prototype that can successfully identify potentially relevant public tenders, we would have to try and deploy this solution as a scalable, big data system.

Our target big data infrastructure would be Apache Hadoop, an increasingly popular open-source software framework for storage and large scale processing of data-sets on clusters of commodity hardware.

The goal is to learn to build a fully working large scale machine learning pipeline that can scale to work on a (hypothetical) large-scale collection of arbitrary text documents.

1.6 Sieve as a Big Data Problem

It's important to clearly state that the specific business problem of public tender classification for Fujitsu's Québec city office is not a Big Data problem. The problem is very specific and the data is not especially large by modern standards. Given the current state of the art in applied machine learning, the problem is easily handled by a single computer.

Our project is a proof of concept for Fujitsu, with the goal of exploring the big data space through the real world implementation of a working system solving a realistic problem.

That being said, our final system is a fully functional large-scale, Hadoop text classification pipeline. It can serve as a starting point for a real production system. The business problem is real and relevant not just to Fujitsu's Quebec City office but also to Fujitsu's huge global consulting business.

The fact that our system could potentially serve the needs of such a large organization also gives us a legitimate reason to implement our solution to be highly scalable by using Apache Hadoop.

1.7 Organization of This Memoir

This work is divided into six sections. We start with background material on Hadoop, big data. Then we move on to machine learning and then to large-scale machine learning. A short chapter on methodology will finally be followed by an account of our experimental work and our results.

Specifically, chapter 2 is a high-level introduction to big data, focusing on Apache Hadoop and MapReduce (section 2.3). This will naturally lead to section 2.4 in which we discuss the

Hadoop ecosystem of emerging technologies that run on top of a Hadoop cluster. These tools provide more high level and specialized services that vastly simplify working with a Hadoop cluster.

The chapter 3 will be a high level introduction to Machine Learning. We focus on supervised methods (subsection 3.3.1), which we used for our experiments. In section 3.6, we focus on the problem of text classification.

We explain in subsection 3.6.1 how text can be modeled to be used as input by a supervised learning algorithm. One aspect of text modeling is called term weighting (subsection 3.6.2), where we had one of our critical breakthroughs, replacing the traditional tf-idf algorithm (section 3.6.2)by an alternative algorithm called Bi-Normal Separation (section 3.6.2).

Then in subsection 3.6.3 we explain from a slightly more theoretical perspective the different machine learning classification algorithms we use in our experiments: SVM and a parallel version of Logistic Regression.

We discuss the important topic of large-scale text classification to close the chapter in section 3.7.

The following chapter brings together machine learning and Hadoop in chapter 4. We present Apache Mahout, a high performance Hadoop-based machine learning framework in section 4.3. We also discuss Apache Pig in subsection 2.4.1. Pig is a high level scripting alternative to raw Java MapReduce development that has allowed us to dramatically reduce the amount of code needed to produce a fully functional text analytics system.

Before we can present our experimental results, we give a brief explanation of our methodological approach through the data mining methodology called CRISP-DM in chapter 5.

Now, all the pieces are set, and we can present our experimental process and results, broken down into two major development iterations: first in chapter 6, we developed a prototype that solves the machine learning problem of sorting public tenders as relevant or not relevant.

This section is broken down following the CRISP-DM methodology, starting with an in-depth review of our understanding of the business problem in section 6.1, a detailed description of our dataset in section 6.2 and how the dataset was prepared for processing in section 6.3.

We explain the construction of the prototype in section 6.4. The prototype applies state of the art machine learning techniques presented in chapter 3. Experimental results obtained using the prototype follow the model description in section 6.5.

Given a working proof-of-concept (or POC) prototype, we can now move on to implementing the same series of processing steps, but this time using Hadoop, running on a cluster of computers. The pipeline of the full-scale system was largely the same as the prototype. but the implementation could not have been more different. The main challenges and solutions are explained in section 7.6.

Again, for the large scale Hadoop system, we go through the steps of data preparation (section 7.3), modeling (section 7.4) and evaluation (section 7.5).

For the evaluation of our large-scale system, we not only review the classification performance (subsection 7.5.1) but also give some indicators of the scaling performance (subsection 7.5.2).

This master thesis will end with a final report on our large-scale text classification system in chapter 8. This final report (section 8.1) will briefly go over the salient points of our system and how our solution has reached the technical and business goals of the project.

We will end with a discussion of potentially fruitful avenues for further work on the system in the Future Works section in section 8.2.

Chapter 2

Big Data and Apache Hadoop

2.1 Introduction

2.1.1 What is Big Data

It seems like "Big Data" has become a catch-all buzz word in the enterprise IT space over the past two years. It's a deceptive term however, since it implies that the data we already have is small, which is not exactly true. Another misconception is that the challenge with big data comes from its size alone. It's certainly true that processing a dataset in the terabytes to petabytes range poses unique IT challenges in terms of storage and processing, but that's certainly not the only one.

In short, the term big data refers to data that can't be processed or analyzed using traditional processes or tools. We can understand that in the enterprise market, traditional processes might refer to software running on a single computer, no matter how powerful. For tools, it's usually relational databases and SANs (Storage Area Network).

The modern mid-sized to global 500 company generates a wealth of data about their products, sales and customers. Typically, they will have some capabilities to analyze this data using business intelligence tools.

There are two problems these companies are facing with their data we will focus on: The volume of data they would like to analyze is increasing at a pace that is outgrowing what traditional relational databases and data warehousing systems can handle cost-effectively. There is a wealth of unstructured data (think emails, web pages and scanned documents as opposed to well formed data in an SQL table) that hold potentially valuable business insights that remains beyond their grasp.

Data is growing more and more rapidly and the rise of unstructured data accounting for 90% of the data today, the time has come for enterprises to re-evaluate their approach to data storage, management and analytics.

Even worse, much of our enterprise process exists as unstructured data, in the heads of workers and lacking any systematic approach for capture, management, communication, measurement and improvement.

While traditional systems remain necessary for specific workloads, the need to address the challenges posed by rising data volume, variety and velocity cost-effectively have created an opportunity for innovation. This framework for understanding the central challenges of big data was first presented in a 2001 Gartner research report by Doglas Laney ([Laney, 2001])[1]. These "3 V's" explain the need that has led to what we know today as big data with Apache Hadoop as its standard bearer.

Gartner's 3 V's of Big Data

Volume The increase in data volumes within enterprise systems is caused by transaction volumes and other traditional data types, as well as by new types of data. Too much volume is a storage issue, but too much data is also a massive analysis issue.

Variety IT leaders have always had an issue translating large volumes of transactional information into decisions — now there are more types of information to analyze — mainly coming from social media and mobile (context-aware). Variety includes tabular data (databases), hierarchical data, documents, e-mail, metering data, video, still images, audio, stock ticker data, financial transactions and more.

Velocity This involves streams of data, structured record creation, and availability for access and delivery. Velocity means both how fast data is being produced and how fast the data must be processed to meet demand.

2.1.2 Hadoop for Enterprise Big Data

The Hadoop platform was designed to solve problems where there is a lot of data, from terabytes to petabytes or more. The data can be a mixture of unstructured and structured data, and it doesn't have to fit nicely into SQL tables.

Hadoop is for situations where you want to run analytics that are deep and computationally extensive, like clustering and targeting. Hadoop is especially suitable in situations where the

[1]Interestingly, this report predates the now well-established term *big data*.

cost of a huge traditional server running a relational database becomes prohibitive compared with that of building a cluster of cheap commodity PCs. Furthermore, when the data is really huge, it may just be the only way of processing the data, regardless of cost.

The underlying ideas used to build Hadoop were invented by Google back in their earlier days as a very scalable platform to index the web, their core business. There was nothing on the market that would let them do that, so they built their own platform: MapReduce and the Google File System (GFS). Google's innovations were incorporated into Nutch, an open source project, and Hadoop was later spun-off from that. Yahoo has played a key role developing Hadoop for enterprise applications.

Hadoop itself has two major components: the Hadoop Distributed File System (HDFS), modeled on GFS, and MapReduce. HDFS is where all the data is stored, MapReduce is the framework used to build applications that can process the data living in HDFS.

There is some confusion about MapReduce, mainly because it is both a Google framework and the algorithmic abstraction that Hadoop and other frameworks use to run parallel and/or distributed programs. From this point on, we will use Google MapReduce to refer to Google's framework, and MapReduce to refer to the programming abstraction used in Hadoop.

Hadoop is a batch-oriented, large-scale, distributed, data processing platform:

- batch-oriented refers to how hadoop processes data by running jobs, one at a time. Each job might take minutes to hours to complete. As such, by design, it is not suitable to answer ad-hoc queries (think SQL query on a RDBMS), or perform real-time or stream processing.

- Hadoop is best on large-scale, read-only datasets. The cost of starting a Hadoop job and Hadoop's overhead is significant. This means unless there is a lot of data to process, think TB to PB, there are likely better alternatives.

- Hadoop is a distributed computing platform, meaning it stitches together tens to thousands of seperate computers to perform as one giant super-computer. It automatically splits up work to perform jobs as fast as possible, on as much data as possible.

- Data processing like analytics is at the heart of the type of problems Hadoop addresses. More on this below.

2.1.3 Use Cases

Hadoop is a good fit for problems that can easily be divided into a number of smaller pieces, which can thus be solved independently. The data is ideally (but not necessarily) in the form of lists, or just a huge chunk of raw information waiting to be processed — be it log files, geospatial data, astronomy data, bioinformatics data, or web pages to be indexed in search engines. Whatever it might be, it is a necessary condition that the final output must not depend on the order in which a lot of data needs to be processed.

The use of Hadoop is now in production in the largest enterprises and organisations for large-scale business intelligence, predictive analytics and data mining. It's helped make possible the ultra fast growth of some of the most well known technology giants of today such as Yahoo, Facebook, LinkedIn and Twitter.

MapReduce and Hadoop is also present in the scientific litterature, with many applications in such fields as data mining [Cardosa et al., 2011] and bioinformatics [Han and Ong, 2012], among many others. We recommend Lee et al's paper for a survey of Academic use of MapReduce in Lee et al. [2012] for a comprehensive first-look tour of the topic. The MapReduce programming model was also implemented in C++, Haskell and python, as well as implementations that run completely in memory[2], on GPUs [He et al., 2008, Chen et al., 2012] and recently in the cloud [Gunarathne et al., 2013, Palanisamy et al., 2011, Fazenda et al., 2012].

Also, Amazon Elastic MapReduce (EMR)[3] has allowed this paradigm to gain much more exposure than ever before, and made it a one-click process. Amazon hosts the Hadoop framework running on Amazon EC2 (Elastic Cloud Computing)[4] and Amazon S3 (Simple Storage Service) [5]. It has allowed MapReduce computations without the headache of setting up servers and clusters, making them available as a resource that can be easily rented out and charged for, on an hourly basis.

2.2 History of Hadoop

2.2.1 Google's MapReduce Innovation

MapReduce is a framework, a pattern, and a programming paradigm that allows us to carry out computations over several terabytes of data in a matter of seconds. When it comes to

[2]`http://www.scaleoutsoftware.com/`
[3]`http://aws.amazon.com/elasticmapreduce`
[4]`http://aws.amazon.com/ec2`
[5]`http://aws.amazon.com/s3`

massive-scale architecture and a huge amount of data, with built-in fault tolerance, there's nothing better than this. But when we come to define MapReduce programming, it is basically just a combination of two functions — a map function and a reduce function. This shows not just the amount of simplicity exposed by the framework in terms of the efforts of the programmer, but also the sheer power and flexibility of the code that runs under the hood.

It was Google that first introduced MapReduce as a framework. It is used to build indexes for Google Web searches! It handles many petabytes of data every day, where programs are executed on a large-scale cluster.

Dean and Ghemawat published the core ideas of MapReduce and the Google File System in two influential papers Dean and Ghemawat [2004], Ghemawat et al. [2003]. Later, Dean published another paper in the topic detailing his experience with the platform in Dean [2006]. Lastly, there was an updated version of the famous MapReduce paper by the original authors in Dean and Ghemawat [2008].

However, this should not push you into thinking that MapReduce is effective only for large datasets and data-intensive computations; what can be more important here is that programmers without any experience with parallel and distributed systems can easily use distributed resources, with the help of MapReduce programming.

There is a difference between distributed computing and parallel computing. Although parallel computing is more of a modified form of distributed computing, parallel computing generally refers to a large number of processors sharing the same memory or the same disks, while distributed computing is increasingly referred to as a cluster of nodes, where each node is an independent unit with its own memory and disk.

These days, the computational power of carefully tuned and configured clusters of such computers can easily match, or even exceed, the performance of several supercomputers that you might have heard of. Another advantage of using such clusters is that they scale horizontally, improving performance roughly in proportion with the number of computers in the cluster. This is a key benefit that has enabled Google to use cheap, commodity hardware to build up their system as they needed it.

MapReduce started life in 2003 as the solution for the future of Google's web index, but quickly grew as Google's engineers learned and made use of it's capabilites. Now, it's used by Google Analytics, Google Earth and Google Maps, among many of Google's famous offerings. It could be said that Google grew to be the internet giant of today through the technical edge of the innovative MapReduce system and its successors.

2.2.2 Doug Cutting and Yahoo Create Hadoop

Hadoop finds its origins in the unlikely project of two developers, Doug Cutting and Mike Cafarella, to build a web-scale indexing system in Java in their spare time back in 2002. To make their vision a reality, Doug created Lucene, which is now the premier Java open-source information retrieval library.

The web, even at that time, far exceeded the processing and storage capabilities of most computers at the time. In 2002, according to Netcraft, there were already upwards of 30M registered domains and 15M live web sites [6]. Scaling to this kind of size would require a sophisticated distributed architecture for storing such large amount of data. Drawing from Google's 2003 paper describing the Google File System Ghemawat et al. [2003], Doug and Mike were quickly able to get a distributed storage component working.

All the while, the web was growing at an explosive rate. The indexable web had already reached around 11.5B pages by 2005-2006 [7]. Distributed storage was not enough, running the indexer also required a lot more computational resources. In response, in 2004, they again turned to an exciting Google publication detailing their MapReduce system Dean and Ghemawat [2004], a distributed computing architecture featuring massive scalability, easy and general programming model and great robustness. They integrated the ideas from MapReduce and had a working implantation in 2005, where the major components of Nutch were implemented using MapReduce and running on their reimplementation of the GFS, which they called the Nutch Distributed File System (NDFS).

Yahoo, in 2006, interested by the technology's potential, offered Doug Cutting the resources to develop the distributed computing component. It was split off from Nutch and renamed Hadoop, the name of Cutting's young son's yellow elephant stuffed animal [8]. From the start, Hadoop was developed as an open source project. It joined the Apache Foundation as an Incubator project and has been a top level project of the Apache foundation since 2008. Doug Cutting views the open-source nature of Hadoop (and Lucene/Nutch/Solr) as one of it's core advantages [9].

Work on Hadoop at Yahoo (Yahoo! at the time) progressed quickly, soon bocaming an essential part of Yahoo's data processing infrastructure. As an open source project, Hadoop was free to be used by anyone, and it was: some of its earliest adopters are notable companies

[6]`http://www.statisticbrain.com/total-number-of-websites/` (Feb 2014)
[7]`http://tinyurl.com/lb6p3zg`
[8]Cutting has a gift for names, as another of his major contributions, Lucene, is named after his wife
[9]Visit `http://www.cloudera.com` and search for Doug Cutting's 2012 Hadoop Word keynote video

such as the New York Times, Last.fm and Facebook. Now, Hadoop is at the center of the Enterprise Big Data movement now worth billions of dollars (TODO-> REFERENCE). Hadoop has spawned three major conferences (Hadoop World NY, Hadoop World Santa-Barbara and Hadoop Summit).

2.2.3 Apache Hadoop Today

Hadoop is now the go-to solution for data-driven companies as well as scientific researchers who want to store, process and analyze massive datasets.

Figure 2.1: The three main Hadoop distributions

There are currently dozens of Hadoop related projects at the Apache Foundation. So much so that we now talk about the Hadoop Ecosystem (see section 2.4).

Installing and configuring a Hadoop cluster using only the Apache Hadoop JAR files can be a daunting task. Up to about 2008-2009 it was also the only way to install a Hadoop cluster. The process is error prone and requires expertise in Linux and networking (software and hardware) just to get a cluster up and running. Adding computers to a running cluster is also a difficult task.

Things have improved a lot in the last few short years. Hadoop is now available as part of tested packaged distributions from several companies. Building a Hadoop cluster using such distributions is now vastly simplified. These companies also provide enterprise grade support and training. Figure 2.1 shows the logos for the three main companies who make Hadoop distributions: Amazon, Cloudera, Hortonworks and MapR.

There are currently four major players building Hadoop distributions, which integrate the most important members of the Hadoop ecosystem into a well integrated and tested Enterprise-ready system:

Amazon While not strictly speaking a software distribution, Amazon's Elastic MapReduce (EMR) is a major player that has lowered the bar to try out Hadoop cheaply and easily with an 100% cloud based system that is well integrated with the equally popular S3 cloud storage.

Cloudera This is the biggest player right now. It's where Doug Cutting works. They include proprietary innovations like a graphical administration console and Impala (a Google Big Query implementation) in their distribution.

Hortonworks Formed from a spinoff of Yahoo's core team that implemented Hadoop. They are a major contributor to the Hadoop 2 (or YARN) effort, led by Arun Murthy. An important part of their Hortonworks Data Platform is that it's 100% open source

MapR Their distribution is focused on providing unique, and proprietary, architectural improvements. Their CTO is Ted Dunning, an important Mahout contributor and renouned machine learning expert.

Other major gobal IT companies are investing heavily in Apache Hadoop and big data technologies like IBM, EMC and even Intel. As of the end of 2013, none of these alternative distributions have gained any market share. According to slides seen at Strat's Hadoop World 2013 conference, the market share seems to be evenly split between open-source 'vanilla' Hadoop, Cloudera's CDH enterprise-oriented distribution and Hortonworks' Hadoop Data Platform distribution. MapR, with 10%, holds the 4th place, and all other distributions together are at less than 1%.

Moving forward, we expect the market share of Apache Hadoop to melt away, as Cloudera and Hortonworks battle it out as the premier, 'standard' Hadoop for the enterprise. The market is in rapid evolution, and we can expect the next few years to evolve rapidly, making our prediction tentative at best.

2.3 Hadoop: Technical Overview

2.3.1 How Does a Hadoop Cluster Work?

Hadoop follows some very important principles in its goal of providing a robust and scalable platform for computation over very large data sets. Firstly, the system must manage itself, and heal itself automatically in the face of inevitable hardware failure. Secondly, its performance should increase linearly as nodes are added to the system. Thirdly, computation moves to the data and not the other way around. Fourthly, the system should be general enough to allow any algorithm to run on the system, and be reasonably simple to develop.

A Hadoop cluster is centrally and automatically managed. The distributed storage is managed in a master-worker architecture. The master is the NameNode, the workers are DataNodes. The programmer doesn't have to do anything to manage the distributed storage in any way. The jobs are managed by the JobTracker and taskTrackers. The jobTracker automatically breaks down a job into tasks and assigns it to taskTrackers on individual nodes that actually perform the computation required by the task. The programmer does not have to explicitly do any kind of management of what portion of the code is executed where. Both storage and job management duties are performed by the framework independently of the algorithms and data that run on the cluster.

Hardware failure is an inescapable certainty when a cluster reaches a certain size, regardless of the reliability of that hardware. Indeed, with thousands of even the most reliable enterprise hard disk drives, we can expect several failures a week. These failures must be automatically and transparently managed by the framework. By transparent we mean that the developer does not need to factor in failures when developing their code. Jobs might be slowed down, but will eventually complete independently of nodes dropping out of the cluster. Even using commodity hardware, a Hadoop cluster, in aggregate, is very reliable. A Job is given strong guarantees that it will complete without loss of data, so long as there is at least one working node for all the blocks of the data for the application, and as long as there is at least one working node to perform work.

Linear scalability is the holy grail of scalable systems. It is also not possible in practice, as there is always some loss due to overhead. While Hadoop does suffer from this overhead problem (Ranger et al. [2007]), mostly due to communication between the workers and the master, it is still usable up to a cluster size of 4000 nodes with some gains of performance. This is the very definition of horizontal scaling at work.

A limiting factor for parallel performance is how much work can be done in parallel. As there will be some necessary data shuffled between nodes for all but the most embarrassingly parallel algorithm, network performance can quickly become a bottleneck. Distributed storage is not like a single hard disk! The data actually resides on real nodes. The idea is that instead of moving a large amount of data to a work node, it's a lot more efficient to package the code up (in java, a jar archive) and ship it to the nodes with the data. This is a critical optimization that was copied from the MapReduce paper.

Hadoop uses the MapReduce programming model. MapReduce is a general programming model that enables programmers to write algorithms for a distributed cluster in a relatively simple and general way. While some might dispute how simple it is to formulate a known algorithm in terms of the Map and reduce primitives, one must keep in mind the alternative of using MPI or other very low level libraries for distributed computing which have a level of complexity orders of magnitude greater.

2.3.2 Map and Reduce as Functional Programming Idioms

Functional Programming is a programming paradigm that uses the mathematical concept of a function as it's core processing structure. This paradigm has a long history of use with such notable languages as Common Lisp and Scheme which are intimately linked to the very beginnings of the computing revolution of the 60's and 70's.

One of the main modern benefits of functional programming comes from the fact that changeable (mutable) state is not allowed. This makes running complex computing on multiprocessor or distributed clusters of computers much easier than the vastly more popular object-oriented programming paradigm (See chapter 13 in Spinellis and Gousios [2009]).

One of the features of functional programming is called *Higher Order Functions*. This refers to a function taking another function as parameter and/or producing a new function as its output. Among various higher order functions we find map and reduce:

- Map: this function takes a list of elements and a transformation function as input, and returns a new list of elements where the new elements are the result of applying the transformation function on each of the original elements. Example: aList=[A,B,C,D] map(aList,toLower()) = [a,b,c,d]
- Reduce: Collect the elements of a list into a single value. In this case, the input is again, a list of elements, and a function that can combine the elements until there is only one left.

These concepts map[10] directly to Hadoop's core processing approach, called *MapReduce*.

2.3.3 MapReduce: Parallel by Default

Hadoop is the name of a distributed computing framework. It has two major components: HDFS for distributed storage and MapReduce as a framework for performing distributed, parallel processing on potentially huge datasets stored on HDFS. Originating with a Google system of the same name, MapReduce implemented in C++, It is named for the two primitives available to the developer for expressing algorithms: Map and Reduce (see figure 2.2).

[10]pun intended

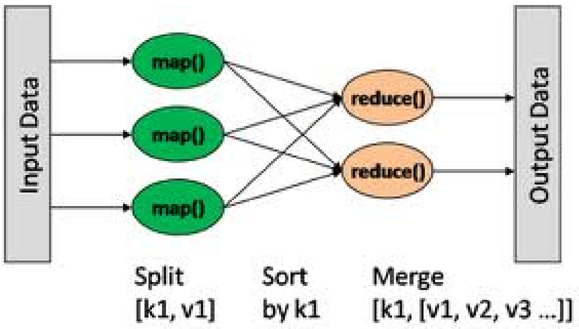


Figure 2.2: Map and Reduce, from input to output (source: ibm.com)

The **Map** step performs a computation on the entire input dataset, where each individual element takes the form of a key-value pair (Ex: line number as a key, the line of text as the value, in the case of a text file). It then emits (returns) 0 or more new key-value pairs (called intermediate data). The output may be of the same type as the input, or it can be different For example: Integer-Text to Integer-Integer in the case of the canonical Word Count.

The **Reduce** step combine all the intermediate key-value pairs emitted by the map operations and outputs 0 or more output key-value pairs. All intermediate keys of the same values are sent to the same reducer. The intermediate key-value pairs are automatically combined such that the reduce function's input is actually a key and a list of all values with have that same key.

Let's work through a complete, high-level example: The data is a list of all the cities and the city's population for the country of Canada. The goal is to calculate the total population of each province. The map function reads the input as a key-value pair of <city:Text,population:Int>and performs a transformation to <province:Int,population:Int>. The processing required is to find the correct province for each city by some method (Google Map API, etc.). The reduce operation simply performs a sum of populations for each province, outputting the output key-value pair of <province:Text,population:Int>. In the case of Canada, it will output 10 key-value pairs, one for each Canadian province and it's total population.

A MapReduce program can be seen as a 5-step parallel and distributed computation (see fig Figure 2.3)[11]. The step are as follows:

1. The data, in the HDFS distributed storage, is divided into splits (generally 64MB in size). Each split is assigned to a map task to a certain node of the cluster. Best effort is

[11]`http://tinyurl.com/kqlckq9` (Feb 2014)

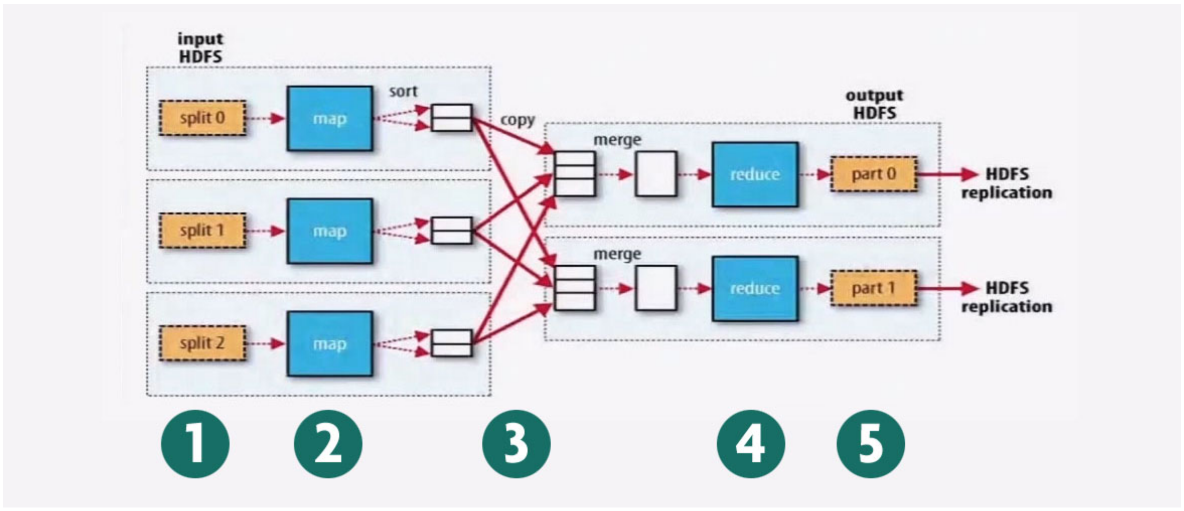


Figure 2.3: How MapReduce Works (source: blog.pivotal.io)

made to assign the split to a map on the computer where the data is located to conserve bandwidth.

2. The map computation is performed on the split, one entry at a time. The map computation has no information on the other splits, the work is performed in parallel, limited by the hardware resources of the cluster.

3. The output of the map (called intermediate results) is sorted locally by its key and all data with the same key is sent to the same reducer. This step may require a lot of network bandwidth. The values with the same key are merged into an iterable list of values.

4. The reduce computation is performed on the list of values and a final key-value pair is emited.

5. Each reducer's output is written back to HDFS, which will automatically start to replicate it to other nodes based on the replication factor (usually 3).

We will use the canonical WordCount example to illustrate MapReduce. WordCount is just as its name implies, an application that will output the list of words as their counts from some input text document. Note that for a much more detailed and complete WordCount example, we recommened the excellent Hadoop Tutorial on the Apache Hadoop web site [12]

[12] `http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html`

The WordCount Example

Let's walk through an example MapReduce application, WordCount, to get a flavor for how MapReduce works. WordCount is a simple application that counts the number of occurrences of each word in an input set

Before we begin, note that the WordCount example, while seemingly trivial, illustrates several key concepts for MapReduce programming. Unusually for a "toy" example. WordCount is a practical, real-life MapReduce algorithm. Counting items in a large-scale dataset is useful for statistics like averages and total counts. Wordcount is also a useful building block for more complex algorithms such as those used for this project.

Another reason the example is deceptive is because our example input is tiny, so we don't need Hadoop to count its words at all! While this will be true for documents up to about 100M word in size, there is a point where simple naive techniques aren't able to process the collection in a reasonable amount of time. As the size grows, the techniques necessary to get the counts will increase in complexity until nothing short of a cluster of computers will be able to perform the task. That's when Hadoop will shine.

This will be our example document:

```
Mary had a little lamb,
Little lamb, little lamb,
Mary had a little lamb,
Its fleece was white as snow
```

Here is the expected output for WordCount:

```
Mary    2
 had    2
little  3
 ...
```

Let us now begin. Counting the words of a small document is a trivial task, and can be done in Python like so:

```
wordcount = {}
textfile = open(r'Mary.txt', 'r')
```

```
#For each word in the file, split on empty char
for word in textfile.readlines().split("␣"):
  #Count the word
  if word in wordcount:
    wordcount[word] += 1
  else:
    wordcount[word] = 1

#print the word counts
for word in wordcount:
        print(word, wordcount[word])
```

So the solution is to read the text file, and for each line, split the line into tokens[13]. Each token is added to a dictionary, increasing the associated count as appropriate.

We can now see how we can solve the problem using the MapReduce abstraction. After that, we can give the gory details for how Hadoop will process our WordCount job with the example input document.

WordCount as a MapReduce Job needs to be expressed in a sequence of map and reduce tasks, a high level view of which can be seen in the following figure 2.4.

Map algorithm (pseudo-code):

```
Map(Key: line num, value: line text)
Tokenize the line
for each word emit(key: word, value: 1)
```

Reduce algorithm (pseudo-code):

```
Reduce(Key: word, iterator: list of 1)
emit(key: word, value: sum of the list)
```

We need to split the data first, and one reasonable way to do that for a text document might be to split it line by line. Line by line input happens to be the default way text is handled by

[13]In this example we split on the empty character, but it could be easily done very smartly using NLTK (`http://nltk.org`)

Hadoop. Each worker nod now has a bunch of (word/1) pairs stored on the local disk. There is a combiner step now, which can perform a pre-grouping of the intermediate results available on each node, which is programmatically configurable by the user. Without going too much into details, this is often the same as the Reduce code and defining it is optional. This process can be seen in Figure 2.4.

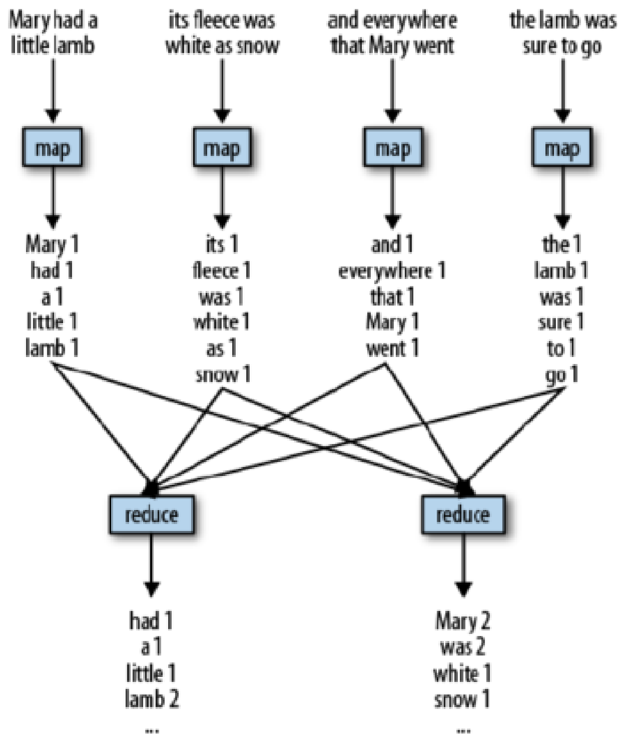


Figure 2.4: Mary and MapReduce: From Input to output (source: Alan Gates in Programming Pig)

Next, the pairs (word,1) will be shuffled and sorted so that all the pairs with the same key are sent to the same reducer. This step will require sending data on the network as pairs are shuffled as it were from the node where they were emitted to the node where the reduce task will be run (possibly on the same node). The node where each task is run is defined by the Namenode based on what nodes have open reduce slots. As slower workers tend to lag behind, reduce tasks often start before all map tasks are finished, as data is ready to process and nodes are free to perform the work.

The reduce node then merges the data into an iterable list and sends it as input to the reduce function. The algorithm for the reduce operation is trivial, just sum the list of 1 values and emit that. This is illustrated in Figure 2.5. Although not for the Mary example, the whole process is shown from start to finish in Figure 2.6.

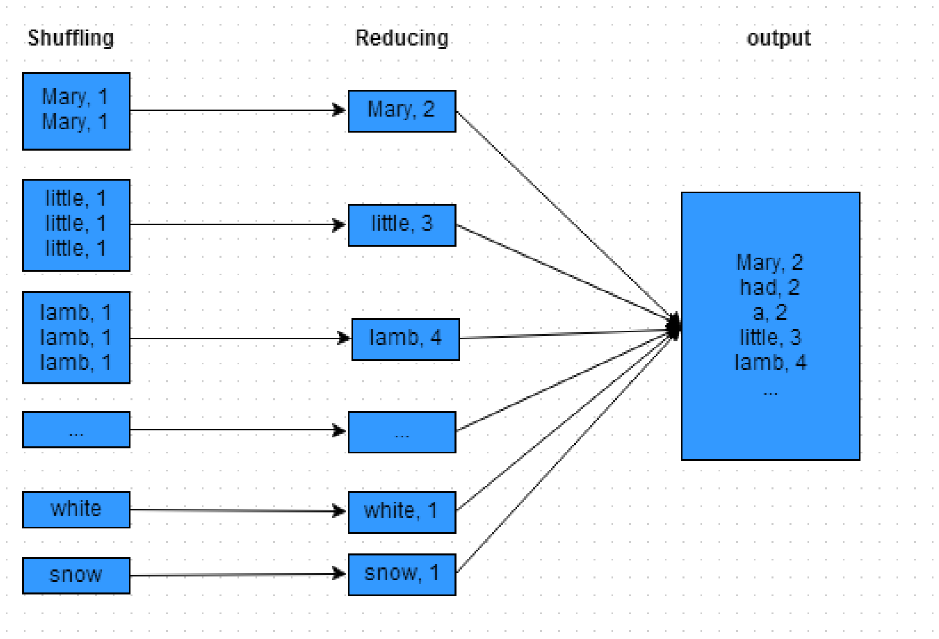


Figure 2.5: Mary and MapReduce: Shuffle and Reduce

The Hadoop framework will write the output of the reduce operation to a file on the distributed storage, and report to the user that the task is done. Good work, Hadoop!

2.3.4 MapReduce Jobs in More Detail

We will go into some additional details but only to a point. Anybody who wishes to go further we refer to the one true, authoritative reference on Hadoop, Tom White's brilliantly written Hadoop Definitive Guide 3rd Edition (White [2012]).

First, some terminology. A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks.

There are two types of nodes that control the job execution process: a jobtracker and a number of tasktrackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers.

Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker. Hadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user defined

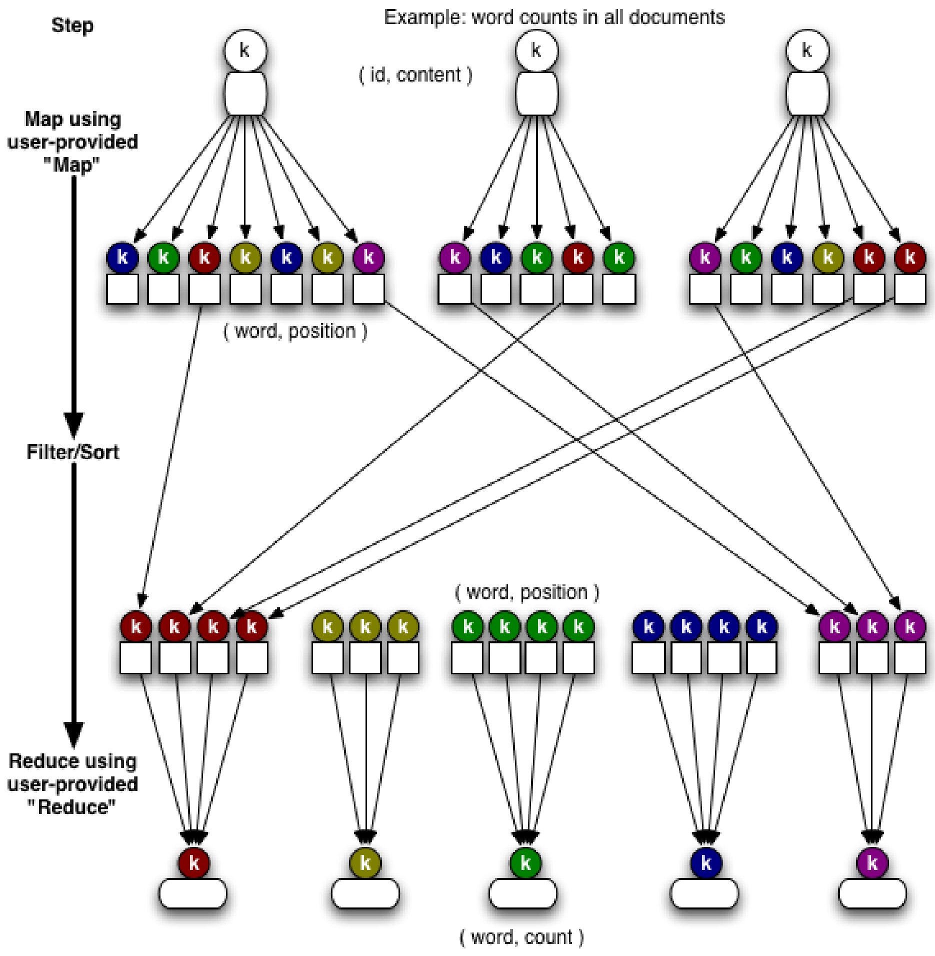

Step
Example: word counts in all documents
k
(id, content)
Map using user-provided "Map"
(word, position)
Filter/Sort
(word, position)
Reduce using user-provided "Reduce"
(word, count)

Figure 2.6: Wordcount solved with MapReduce whole system view (source: www.yuyellowpages.net)

map function for each record in the split. Each split will map with HDFS blocks, which are by default 64MB in size.

1. Job Submission The WordCount code is packaged as a Jar and sumitted to the cluster. The JobTacker is now in control, the user will wait for completion (success or failure with a report of various counters). The jobtracker assigns a new job ID and computes the input splits for the job. The jobtracker needs to know what data the task needs and where it is located on the cluster. Resources needed for the job like the job Jar file, libraries and config files are copied over to all the nodes that will perform work on the job.

2. Job Initialization The job is now added to a queue and will be scheduled by a job scheduler. This is where the job tasks will be computed. Computing the map operations

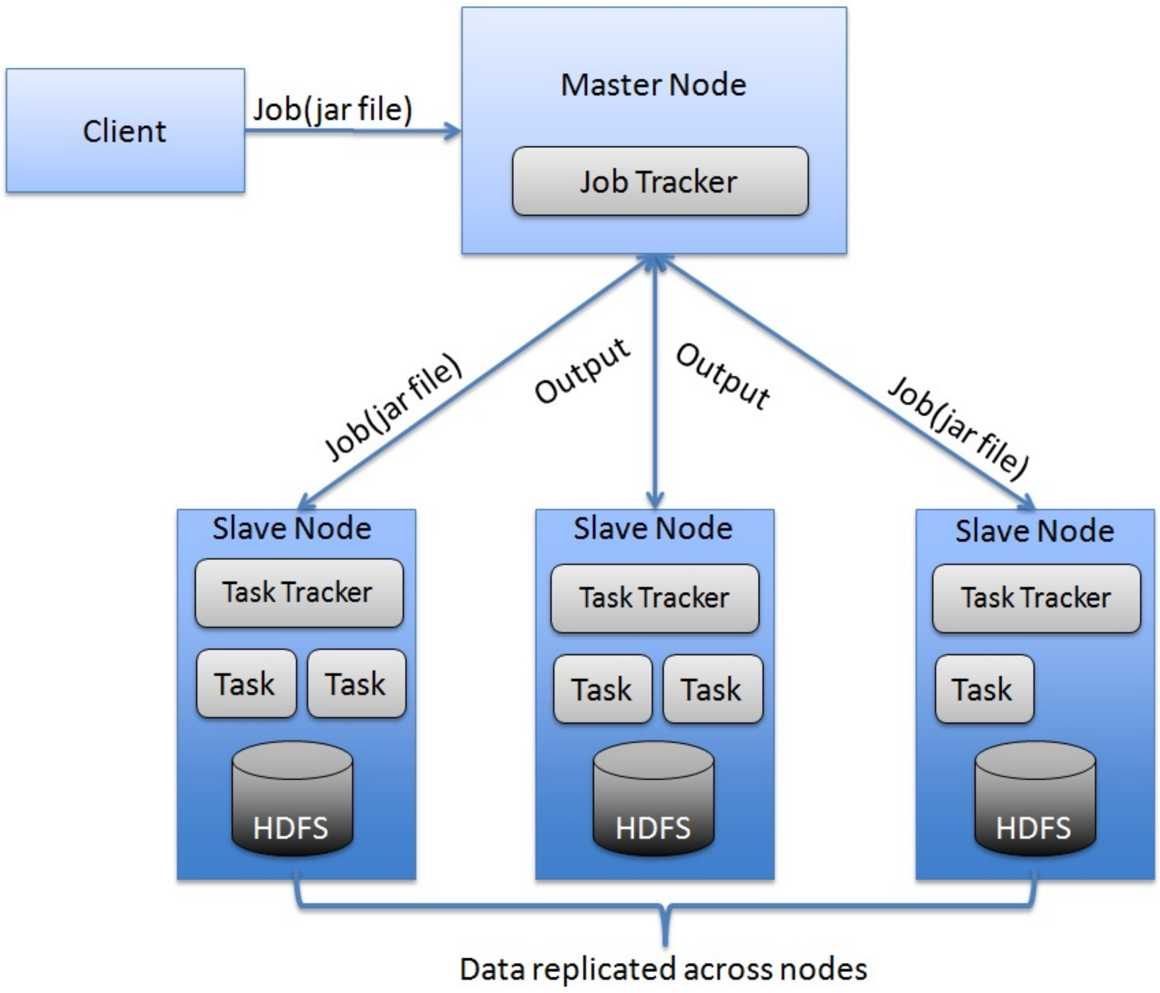


Figure 2.7: Hadoop Job Architecture

is just a matter of counting the splits, one map per split. The number of reduce tasks is simply a property of the job, and a good rule of thumb is about one reduce per node. Tasks are assigned IDs.

3. Task assignment JobTracker assigns tasks to taskTrackers. It checks up on their progress by polling them in a process called a "heartbeat" method. This checks if the tasktracker is alive and making progress on the task. This allows the jobTracker to know if a task is failed or if a node goes down. In such a case, the task will be reassigned to another tasktracker. Tasks are assigned so that the taskTracker of the machine where the data split is located will be preferred. This is following the principle of moving computation to data. A single machine can have many map and reduce slots and may perform many tasks in separate threads. This depends on the cluster configuration and depends on the hardware running the cluster.

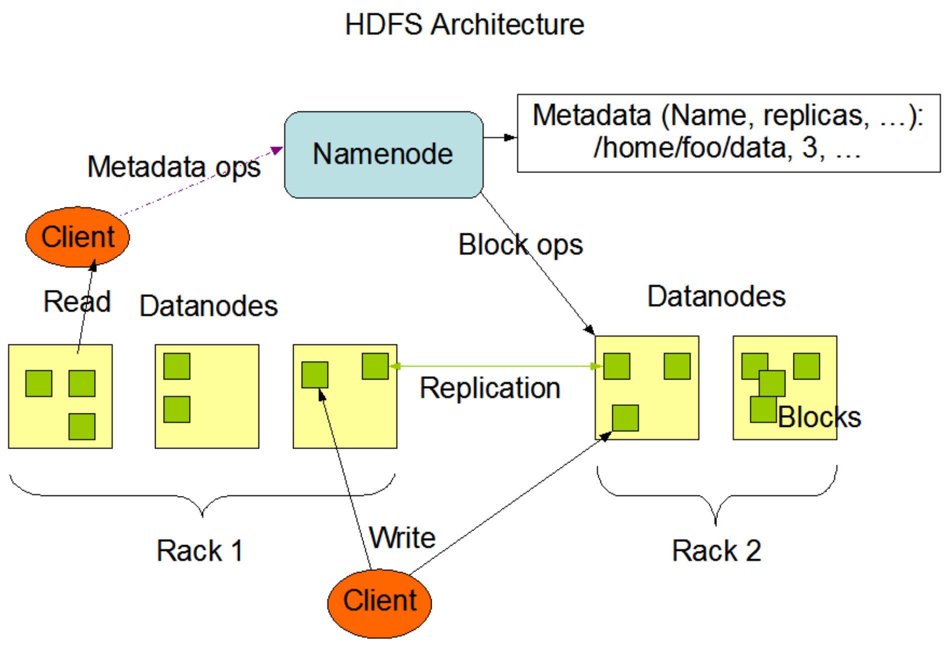


Figure 2.8: The HDFS Architecture (source: Apache Hadoop)

4. Task execution Each task is run in a separate JVM. Progress is communicated back to the tasktracker by a separate thread. These status updates can be displayed as a job status with an estimated completion rate. Map task estimate is based on the data processed and the data yet to be processed on the split, the reduce estimation is more difficult and so in practice is not really all that meaningful beyond knowing it's running and is not done yet.

5. Job Completion Intermediate results for maps are stored on the local hard drive. The results of reduce tasks are stored on the distributed storage.

2.3.5 The Hadoop Distributed File System (HDFS)

The Hadoop Distributed File System (HDFS)[14] is a distributed file system designed to run on commodity hardware. HDFS is optimized for applications that have large data sets and is an implementation of the ideas and techniques described in the Google File System (GFS) paper Ghemawat et al. [2003]. HDFS is very robust through replication and is designed to run on low-cost hardware, scaling horizontally. It exposes a single, POSIX-style file system to users.

How is it possible to build a highly fault-tolerant file system using hardware with comparatively bad reliability? The trick is to factor in hardware failure in the design from the very start.

[14] `http://hadoop.apache.org/hdfs`

All data is stored in blocks that are replicated across the cluster. The default block replication rate is three, with at least one block kept in a seperate rack, if possible. For improved reliability, HDFS also keeps redundant copies of checkpoints and journals, spread across multiple remote nodes.

The cluster storage master, known as the Namenode, keeps a master list of all the blocks in the cluster and their physical location. It constantly monitors the health of the nodes of the cluster by polling the nodes at a regular interval. As soon as a node becomes unavailable, presumably due to some sort of failure, the Namenode can begin replication of the blocks that don't have the required replication factor to other available nodes. In practice, Google, using the very same technique on the GFS found this system to provide the highest level of reliability.

As illustrated by the figure 2.8, HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.

The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode [15].

For more technical/programming details on HDFS, we will refer to White [2012]. For an even more in-depth tour of HDFS's architecture, see Brown and Wilson [2011] in the excellent *The Architecture of Open Source Applications* series.

Finally, there are a number of projects that allow to use other programming languages to use Hadoop API without the limitations of Hadoop Streaming. One of the very best such efforts is the Pydoop project[16] described in Leo and Zanetti [2010]. Another well-known Hadoop-for-Python project is Dumbo[17].

[15]`http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html` (see sections NameNode and DataNodes) (Feb 2014)

[16]`http://pydoop.sourceforge.net/` (Feb 2014)

[17]`https://github.com/klbostee/dumbo/wiki` (Feb 2014)

2.3.6 Hadoop Pipes and Streaming

Apache Hadoop was programmed in Java and requires Java 1.6 and Maven to compile. Hadoop's native API is therefore also in Java. However, it is also possible to program Hadoop jobs in a variety of programming languages.

It is possible to build a Hadoop job using C++ with Hadoop Pipes. Pipes is a library which allows C++ source code to be used for Mapper and Reducer code. Applications which require high numerical performance may see better throughput if written in C++ and used through Pipes. This library is supported on 32-bit Linux installations.

Whereas Pipes is an API that provides close coupling between C++ application code and Hadoop, Streaming is a generic API that allows programs written in virtually any language to be used as Hadoop Mapper and Reducer implementations.

Hadoop Streaming allows you to use arbitrary programs for the Mapper and Reducer phases of a MapReduce job. Both Mappers and Reducers receive their input on stdin and emit output (key, value) pairs on stdout.

Provided it can handle its input in the text format described above, any Linux program or tool can be used as the mapper or reducer in Streaming. It is possible write scripts in bash, python, perl, etc. provided that the necessary interpreter is present on all nodes of the Hadoop cluster.

2.4 The Hadoop Ecosystem

The Hadoop ecosystem, as shown in Figure 2.9[18] is a growing collection of specialized projects that simplify various use-cases or ease use of a Hadoop cluster as a back end for more high-level analysis tools such as Apache Pig.

At its core, Hadoop is only made up of HDFS and MapReduce. But people and companies have made contributions that make Hadoop a more complete platform.

As of the time this was written in 2013, some of the main projects in the Hadoop ecosystem are the following:

- Apache HBase: The premier NoSQL database, giving Hadoop a way to index data. It's tremendously scalable and powerful but also very difficult to administer, especially at

[18] `http://www.developer.com/services/getting-started-with-azure-hdinsight.html` (May 2014)

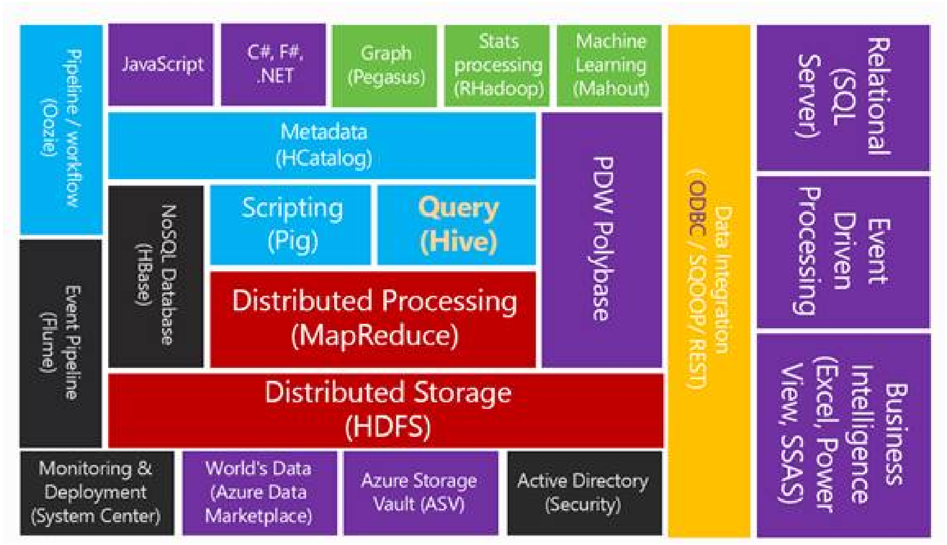

Pipeline / workflow (Oozie)
JavaScript
C#, F#, .NET
Graph (Pegasus)
Stats processing (RHadoop)
Machine Learning (Mahout)
Data Integration (ODBC / SQOOP/ REST)
Relational (SQL Server)
Metadata (HCatalog)
PDW Polybase
Event Driven Processing
NoSQL Database (HBase)
Scripting (Pig)
Query (Hive)
Distributed Processing (MapReduce)
Event Pipeline (Flume)
Distributed Storage (HDFS)
Business Intelligence (Excel, Power View, SSAS)
Monitoring & Deployment (System Center)
World's Data (Azure Data Marketplace)
Azure Storage Vault (ASV)
Active Directory (Security)

Figure 2.9: Hadoop is at the center of a growing number of supporting components collectively called the Hadoop Ecosystem

scale. Made to work with Hadoop, with which it integrates very nicely. HBase, as most NoSQL DBs pushes consistency and syncronisation responsability on the developer.

- Hive: An SQL centric abstraction on MapReduce. Hive automatically compiles a HiveQL script into an optimized MapReduce equivalent without the user ever needing to touch Java. Heavily used by Facebook, where it was first developed.

- Pig: A dataflow oriented abstration on MapReduce, which similarly allows coding an equivalent of an SQL execution plan, which Pig compiles into an optimized MapReduce program. Pig's script is called PigLatin and looks a bit more like code, compared with Hive's very SQL looking scripts. It's very flexible with possibilities for adding user defined functions in a variety of programming languages.

- Mahout: A machine learning library which is implemented using basic MapReduce. Elegant, powerful and state of the art, used by companies like Linkedin for their recommendations like "People you might know". Includes recommenders, clustering and classification algorithms.

- Sqoop: Bridges the gap between a RDBMS and Hadoop's HDFS, allowing simplified import and export of data between the two.

- Flume: Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.

2.4.1 Pig: High-Level MapReduce Programming

What is Pig

Apache Pig is a platform for analyzing large data sets and is designed to run on top of Apache Hadoop. It consists of a high-level language for expressing data analysis programs and a compiler that automatically produces sequences of Map-Reduce programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

There are some problems that come with writing Java MapReduce programs. The developer must be comfortable with advanced Java programming, learn the Hadoop Java API. On top of that, the developer must also learn to structure algorithms with the MapReduce paradigm and manage long chains of jobs, which is pretty hard. Pig solves all this problems.

Pig programs are written in Pig Latin. Pig scripts are compiled and optimized into a chain of MapReduce jobs automatically by Pig and can run either on a Hadoop cluster or on a developer's local file system. Pig scripts are much smaller than their Java equivalents, and tend to be roughly of comparable speed. Tests have shown Pig to be roughly 30% slower than an expertly tuned Java MapReduce equivalent program, but a beginner's Pig script is likely to be much faster than a beginner's MapReduce program, in but speed of execution and developer time.

From Pig's description page, Pig has three key properties:

- Ease of programming. It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- Optimization opportunities. The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- Extensibility. Users can create their own functions to do special-purpose processing.

As an Apache open source project, Pig is open-source and has a business use friendly license. This means users are free to download it as source or binary, use it for themselves, contribute to it, and—under the terms of the Apache License—use it in their products and change it as they see fit.

The authoritative book on Pig is Gates [2011]. It was written by a member of the core team that developed Pig at Yahoo. The online documentation is an essential reference due to the rapid evolution of the API and progressive increase in built-in operators[19].

Scientific Pig

As with Hadoop, the team developing Pig also contributed back to the academic community by writing two papers detailing Pig in Gates et al. [2009] and Olston et al. [2008].

MapReduce is a relatively low level abstraction, making the need for a higher level query language clear to several teams as it was to Pig's R&D team. Hive presented in Thusoo et al. [2009], which tackles the problem with a more SQL-like approach, originates from Facebook as is hugely popular for data analysis type workloads. IBM Research proposes another similar solution, but focusing on JSON with JAQL in Beyer et al. [2011].

All these systems, including Pig, where conveniently compared with Java MapReduce in Stewart et al. [2011]. While not especially showing one HLQL's superiority over the other, the paper makes a great point of how much better they are to Java MapReduce code on a lot of common workloads, especially for the reduction in code size, with Java requiring 10-12 times more code than the HLQL's! This is exactly why we set out focusing on using Pig as a primary development tool for our system.

We can close this section with a short list of recent research work on improving Pig performance or ensuring service execution time bounds by Zhuoyao Zhang and al in Zhang et al. [2012b], Zhang et al. [2013] and Zhang et al. [2012a]. Pig is also quite useful for data mining applications at web scale, as shown in Shang et al. [2012]. Lastly, is also some work trying to optimize Pig for running over RDF data in Ravindra et al. [2010] and Schätzle et al. [2011] that show how Pig can be a very practical tool for semantic web type research.

Igpay atinlay siay unfay (PigLatin is fun)

As described in wikipedia, dataflow programming is a programming paradigm that models a program as a directed graph of the data flowing between operations. In the dataflow paradigm there are no for-each loops or conditional branches, but rather operations that transforms, partitions or filters data. SQL is the prototypical example of a dataflow language. Pig fits this definition perfectly as it allows the programmer to express the processing of data as a sequence of SQL-like statements. Behind the scenes, the Pig runtime will transform the script into an optimized series of Map-Reduce operations directly using Hadoop's native API.

[19] As of this writing, for version 0.12: `http://pig.apache.org/docs/r0.12.0/` (Feb 2014)

The benefit of using Pig can be illustrated using the wordcount example. Even though wordcount is the simplest possible useful MapReduce algorithm, using the Java API will still required a lot of lines of code and non-trivial understanding the the Hadoop Java API. The Pig version is just a few lines of easy to understand script (see section 2.4.1).

The higher level view made possible by Pig should not be interpreted as a loss of efficiency or capability. The Pig developers are experts of Hadoop optimization, an expertise out of the reach of the vast majority of current and potential Hadoop programmers, and all Pig scripts fully take advantage of each new optimization simply by using a newer runtime. As for capability, our own system's pipeline makes heavy use of Pig to express non-trivial, state of the art algorithms (see Figure 7.1 in chapter 7). With modern libraries like Linkedin's DataFu and Twitter's elephant-bird, Pig should be viewed as a first choice for Hadoop implementation.

Another way to explain this important dataflow concept is by focusing on the schema of the data, not the actual data. The Pig programmer's task is to perform operations on the schema of the input data and transform that schema until it matches the desired output schema. The data will follow the specified path and take care of itself. Again, it's a very SQL-like approach: Pig requires the programmer to specified the desired result, and takes care of the real details of how to get that result just like a database engine takes care of the retrieval of records specified by an SQL query.

There is no better way to get used to it than an example, so lets dive right in!

WordCount with Pig: Back to Mary

Lets pick up where we left off with our WordCount example, working with Mary and her little lamb.

```
A = load 'mary.txt' as (line:chararray);
B = foreach A generate flatten(TOKENIZE(line)) as word;
C = group B by word;
D = foreach C generate group as word, COUNT(B) as count;
E = order D by count desc;
store E into 'wordcount';
```

The easiest way to understand a Pig script is to think in terms of SQL. Let us look at a working WordCount Pig Latin script shown below:

1. A: The `load` command loads a file line by line. This is the default behavior for text, but can be configured to read anything. `A` is a relation with one chararray column, and each line (as in each line of an SQL DB table) holds one line of text from the mary.txt document. We reiterate, do not confuse A with a variable, it is a relation.

2. B: TOKENIZE will produce the expected result of taking a line of text and breaking it up into tokens. `foreach` applies the TOKENIZE to all lines in the A relation. B is a relation (keep thinking of an SQL table) with one column word which will hold one word. FLATTEN un-nests the items in the bag produced by the TOKENIZE function. Flatten requires some effort to understand and use well, We recommend looking at many other examples of its use. Suffice to say it is necessary here for now.

3. C: groups B by word. Think of the GROUP BY operator of an SQL SELECT query. C is a bag with the schema:
   `C: {group: chararray,B: {(word: chararray)}}` where group is an automatically assigned name that contains the key grouped on, followed by a bag holding all the elements of the relation B that match the `group by` key.

4. D: does the actual counting by taking all the elements of the D relation and counting the number of items in the B bag. The `as` allows renaming columns, which is only for convenience but is not strickly necessary.

5. E: the D relation is ordered by count in descending order. The final schema of our data is `D: {word: chararray,count: long}` which is exactly what we wanted.

6. Finally, the results are stored into a text file.

So six short and simple lines of Pig script are equivalent to about 120 lines of java code[20].

Of course, a more involved and complex Pig script, involving a few UDF functions will be a bit longer and be a bit harder to write. On the whole, Pig is frankly so good that it should always be used first before writing any kind of Java code. As we'll show later, Twitter have been able to run their entire machine learning pipeline using Pig, in such a way that no Java MapReduce code is every required.

User Defined Functions Add Flexibility and Power

The Pig library contains basic functions that can perform some generic computing tasks such as computing sums. But how can these allow Pig to perform arbitrary computations?

[20]`http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html` (Feb 2014)

Pig includes a means for programmers to supply their own functions and use them in their scripts. This is called User Defined Functions (UDFs). Recent versions of Pig make writing such function libraries using a variety of JVM programming languages (Java, Jython, Ruby, Javascript Groovy, etc.) straightforward.

Using UDFs, the Pig programmer is able to do almost anything that can be done in native Hadoop API, but with the productivity and optimization benefits of Pig.

In addition, there are libraries of additional Pig UDF functions out there: the Piggybank, by Pig's development team, DataFu by Linkedin[21] and Elephant-Bird by Twitter[22]. They are all very useful and should be part of any Pig programmer's basic toolkit.

2.5 The Future of Hadoop is YARN

Apache Hadoop is an increasingly battle-tested enterprise data-processing framework. But as of 2012-2013, the needs of big data processing are expanding outwards from batch processing into real-time streaming. More and more complex algorithms can be very difficult to express using map-reduce only. In addition, the programming world is moving in the direction of an increasing number of languages and approaches, leading edge companies like Google do not limit themselves to C or Java or Python, but embrace several languages and frameworks, often in the same projects.

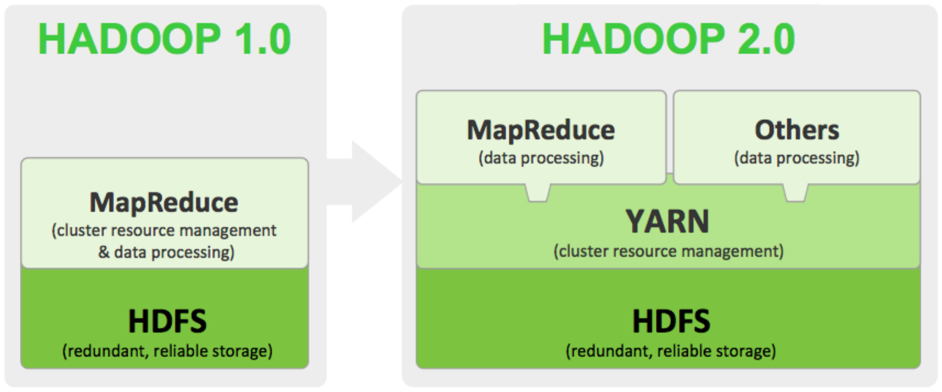

HADOOP 1.0
MapReduce
(cluster resource management & data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)

Figure 2.10: YARN adds an abstraction layer over HDFS (source:Hortonworks)

The Apache Hadoop of the future is YARN, for Yet-Another-Resource-Negociator. Simply put, it's a new, thin abstraction layer that decouples the HDFS datastore from the Map-Reduce algorithm. This opens up Hadoop to be used as cluster management and processing

[21]https://github.com/linkedin/datafu (Feb 2014)
[22]https://github.com/kevinweil/elephant-bird (Feb 2014)

allocation framework on the one hand, and programmers can now port existing distributed processing frameworks to run on it. Hadoop becomes a true multi-purpose big data processing infrastructure.

Backward compatibility is retained, as MapReduce programs continue to work as-is. Also, new processing frameworks, often focusing on real-time analytics, are emerging at an amazing rate, led by projects such as Apache Tez, Apache Storm and Apache Spark, among many others. Javi Roman keeps an update list of Hadoop related projects on GitHub at: `http://hadoopecosystemtable.github.io` (Feb 2014).

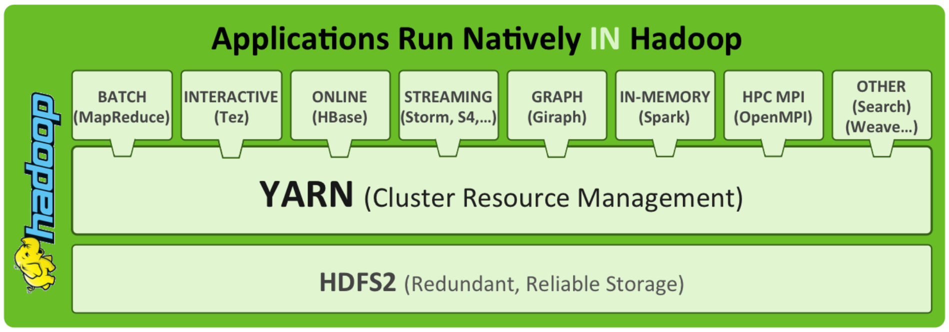

Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

Figure 2.11: With YARN, MapReduce is one of many ways to process data in HDFS (source:Hortonworks)

2.6 Alternatives and Criticisms

Hadoop and MapReduce is not a be-all-end-all solution for solving big data problems. It is in fact a very specific tool, appropriate for batch processing of very large, read-only datasets, possibly from multiple sources. It competes with large-scale data warehouse and very large relational database systems.

DeWitt and Stonebraker in DeWitt and Stonebraker [2008] question the novelty of MapReduce, citing prior art with Teradata. They also view the programming model as a step back because it's too low-level, and unfavorably compare MapReduce with relational databases. These charges led to Dean and Ghemawat publishing an update of their MapReduce paper to address the issues in Dean and Ghemawat [2008].

It's interesting to see how, just a few years later, today, these criticisms are completely outdated, as Google advanced on MapReduce with BigTable [Chang et al., 2006] which is a NoSQL database, Dremel [Melnik et al., 2010] which allows SQL like ad-hock queries at extreme scale, and most recently F1 [Shute et al., 2013] which combines BigTable with traditional SQL capabilities. Google also addresses the need for a high level programming alternative to MapReduce with Sawzall [Pike et al., 2005].

The tradition of open-source implementations based on Google papers is not limited to Hadoop. Apache HBase, among others, is based on BigTable, Apache Drill is based on Dremel and Apache Pig is based on Sawzall.

Hadoop may not be appropriate if a problem can be dealt with in a more traditional way. Chris Stucchio in a blog post[23] gives some examples of various problems where Hadoop is arguably not the proper approach.

We can summarize his post in the following table:

Problem	Stucchio's Recommendation
Excel won't load the data (100-1000MB)	R or Pandas and Numpy will handle this with ease
My data is 10GB	It's easy and cheap to order a laptop with 16GB of ram and a 256GB SSD drive. Still no big deal for R or Pandas.
My data is 100GB-1TB	An SQL RDBMS will perform very well here.
my data is more than 5TB!	Maybe it's time to think about Hadoop after all.

Stuccio also makes the point of SQL being stictly better than MapReduce as an abstraction. This is a fair criticism when comparing Java MapReduce programming with SQL, but is addressed by already existing solutions like Hive that allow SQL-like processing on Hadoop. Also, there are several innovative solutions for running very fast SQL-like queries on data, such as Cloudera's Impala and Apache's Drill. There is already one vendor proposing ACID compliant SQL on Hadoop, with Pivotal's innovative HAWQ. Hadoop is quite friendly with SQL, really.

On to the question of data size, brought up in the post. Hadoop is not the right tool for performing spreadsheet style analysis on data of less than several TB in size.

However, big data is not about only strict size. There is also the question of velocity, variety and scalability. Analysing data like logs that accumulate several GB a day from several

[23] `http://www.chrisstucchio.com/blog/2013/hadoop_hatred.html`

systems at once can be handled very elegantly and cheaply with Hadoop, despite not being in the TB range in size. Similarly, Hadoop's HDFS is an exceptional data warehouse solution, allowing cheap horizontal scaling and easy analysis on data without any need for filtering or processing. Integrating several data sources from what are known as data silos is another very strong area for Hadoop.

Hadoop indeed has many problems, ranging from the MapReduce abstraction being inneffective for some problems (thinkg graphs) and its difficult programming model to its poor performance. But coming out of 2013, Hadoop is emerging as the premier enterprise data processing infrastructure. Jimmy Lin, in Lin [2012], explains that while it may be true that certain classes of algorithms aren't naturally suited to the MapReduce model, Hadoop has evolved and matured to the point where it is now "good enough". This means it is now more effective to work on transforming the problem to fit onto MapReduce for large scale processing rather than changing MapReduce to better fit the problems.

As we argued in this section, Hadoop is at a point of maturity where the weaknesses of the platform are well understood and have been addressed. Hadoop 2.0 is a major step in this direction, eliminating the troublesome single point failure problem, as well as uncoupling HDFS from MapReduce.

The major Hadoop distributions are quickly gaining maturity, with greatly improved ease of installation, administration and error recovery and diagnostic, though more work remains to be done on those points.

Finally, thanks to the business-friendly Apache open-source license, Hadoop is evolving at great speed literally for free for the average enterprise or academic user, and is likely to become part of the standard tech infrastructure of most mid to large scale organizations in the near future.

2.7 Conclusion

These days, big data in general and Hadoop in particular have rapidly become a fixture in technology news. We explained how Apache Hadoop is a Java implementation of Google's MapReduce system. MapReduce is one of the most important technological foundations that have allowed Google to become the tech behemoth of today, and it's truly exciting and important that such a technology can now be used with relative ease by much less technically savvy companies and even university research groups.

The MapReduce functional approach is based on ideas of developer productivity, presenting a relatively high level abstraction on the difficult problems presented by programming applications that can scale on clusters made of thousands of “regular PC” computers.

Chapter 3

Machine Learning

3.1 Artificial Intelligence and Machine learning

3.1.1 Defining Artificial Intelligence

> Artificial Intelligence is the theory and development of computer systems able to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages. (Oxford Dictionary)

Doing those tasks involve knowledge that is currently implicit, but we have information about those tasks through data and examples (e.g. observations of what a human would do given a particular request or input). How do we get machines to acquire that kind of intelligence? Using data and examples to build operational knowledge is what learning is about.

3.1.2 Machine Learning

Since the very beginnings of the computer revolution, we have learned to explicitly instruct computers to perform almost any task we can think of... in advance. While this has proved useful, it is a far cry from the vision of the future shown in Star Wars or Star Trek. Tinsel town robotic legends such as C3PO show us just how primitive the robotic and AI of today is.

However, over the past decades, things have finally started to improve, thanks to advances in the field of machine learning. One notable example is the self driving car found on Google's campus (based on the original project led by Sebastian Thrun in Thrun et al. [2006]). Machine learning as also been a key driver in a recent surge of powerful real-world technologies in

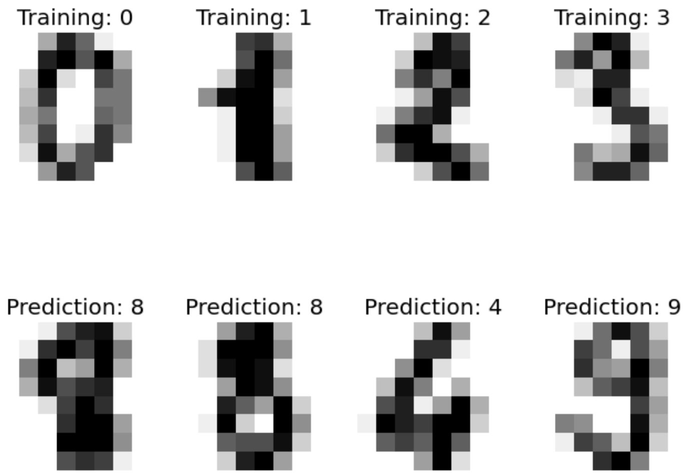


Figure 3.1: Handwritten digits classification using machine learning (source: `scikit-learn.org` (Feb 2014))

such diverse areas as speech recognition and question-answering (Siri), image recognition [Jurafsky and Martin, 2008] or even genomics [Larrañaga et al., 2006].

Machine learning is pervasive today, especially for those using any of the various services offered by cutting edge companies such as Google, Facebook, Twitter and Microsoft.

Techniques in machine learning concern themselves with the construction and study of systems that can learn from data. For example, a machine learning system could be trained to recognize correctly handwritten digits (Figure 3.1). After learning, it can then be used to classify new images of handwritten digits into their correct category, ranging from 0 to 10. Another such example might be spam filtering, which would distinguish between legitimate and spam emails.

More formally: "The core of machine learning deals with representation and generalization. Representation of data instances and functions evaluated on these instances are part of all machine learning systems. Generalization is the property that the system will perform well on new, unseen instances"[1].

There is a wide variety of machine learning tasks and successful applications. Some can be

[1]`http://csip.ece.gatech.edu/?q=technical-area/machine-learning` (Feb 2014)

seen when using Facebook with their “You might know...” feature, or when using Amazon with their fantastic (and sometimes creepy) recommendations. Other well known examples are Google News, which automatically groups news into topics, or Google’s Gmail service’s Priority Inbox that intelligently sorts some new emails to a special “Important and unread” category.

Recently, there has been an increase in the attention given to machine learning techniques. New fields are experimenting with machine learning techniques and finding much success. We discuss some of those in the State of the Art section 3.2.

More formally, machine learning can be separated into supervised and unsupervised methods. Supervised methods can be further split up into regression problems and classification problems.

The study of machine learning can be approached more theoretically, focusing on algorithms, proofs and bounds (Alpaydin [2010], Bishop [2006]). It can also be approached a little bit more like a black box that gives pretty good answers to potentially very difficult problems. We will give some concrete, real world examples of machine learning in the next section.

3.2 Solving Problems with Machine Learning

It’s increasingly difficult to spend a day without any contact with a system that includes some machine learning. Especially on the internet today, giants like Google, Facebook, LinkedIn, Twitter and Amazon increasingly rely on machine learning as a source of competitive advantage.

The leading edge of online advertising (think Google’s AdSense) heavily rely on machine learning to optimize ad placement and effectiveness. Agarwal gives a detailed account of this process at LinkedIn in Agarwal [2013]. Also, a recent survey of the field can be found in Shatnawi and Mohamed [2012].

3.2.1 Supervised Versus Unsupervised Learning

Supervised Learning

In *supervised learning*, the data comes with additional attributes that we want to predict. Hence, a machine learning algorithm is trained on those labeled examples, in order to produce a predictor (a model) that can then be used to label new examples.

Regression for when the desired output consists of a continuous variable (also sometimes referred to as a scalar output). An example of a regression problem would be the prediction of sales, temperature or any other measurable quantity.

Classification example of classification problem would be the handwritten digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

Structured prediction A more recent development in supervised machine learning, the objective is to predict structured objects (a picture, a vector, a protein, etc.) instead of just a single number or a class. It is a very difficult problem due to the enormous space of potential output objects.

Unsupervised Learning

Where supervised learning uses a labeled collection of examples as a starting point, *Unsupervised learning* learns to assign a label to examples using only the examples themselves.

One of the most common unsupervised learning method is called clustering. This learning technique, separates the input dataset the into groups, or clusters. Data instances that that are similar (based on a certain similarity criteria, or function) will be assigned to the same cluster. Conversely, examples that are very different from each other will be in different clusters.

Clustering, or cluster analysis, is often used as an exploratory data analysis to find hidden patterns or grouping in data.

Other common clustering algorithms include:

k-Means clustering partitions data into k distinct clusters based on distance to the centroid of a cluster

Hierarchical clustering builds a multilevel hierarchy of clusters by creating a cluster tree

Deep learning part of a broader family of machine learning methods based on learning representations that originates from neural network research.

Hidden Markov models uses observed data to recover the sequence of states

Semi-supervised Learning

There is much potential in combining supervised and unsupervised techniques. Such a combination has come to be known as *semi-supervised learning*. Semi-supervised learning tries to bridge the information gap often present in real-world problems where there is only a small amount of labeled data. All other data is of unknown label. The idea is to inform the supervised learning task using only a relatively small set of labeled examples combined with additional information gained by applying unsupervised learning techniques on a large set of additional unlabeled data.

Typically, there is a small amount of labeled data and a large amount of unlabeled data. One common cause is that while it may be easy to get data, it may be costly to label it. For example, think of the cost of testing a molecule for antibacterial properties in a laboratory, or hand labeling a text file with part-of-speech tags. Still, many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy.

The causes of having a small training set is usually the cost of labeling data, be it in terms of money, time or resources. One example of this is in biology, where expensive and time consuming lab work is required to know some property of a protein or other biological structure.

Another situation might be because of the extreme rarity of examples of the other class. For example, a data set on the operation of a nuclear power plant, where negative examples are (thankfully) fairly rare compared with data on normal operation. In such situations, semi-supervised learning can be of great practical value.

3.2.2 Applications of Machine Learning

Recommender Engines

Recommender engines are at the very core of the business model of many well known companies such as Netflix and Amazon.

The task of a recommender is to offer users recommendations of new items (think music, movies or books) to a target user. The recommendations are usually based on the concept of similarity between users or between items. It's generally a good idea to recommend to a user what a similar user has previously enjoyed. In the case of item-based recommendations, the idea is to recommend items that are similar to items the user enjoyed in the past.

The quality of the results shown to users can make a very large difference in the sales of an online vendor, and is a very challenging problem, as explained in Cremonesi et al. [2012]. Some recent results from research in the field can be found in Amatriain [2012], Jahrer et al. [2010]. Recommender systems are often used on datasets with millions of products and thousands to millions of users, making large scale a relevant and important context for such systems. Some examples of recent publications dealing with this issue are Takács et al. [2009], Amatriain [2013].

Clustering

Clustering can be used for grouping things without having prior knowledge of the data. A recent survey of the topic and its application to data mining can be found in Dalal and Harale [2011]. Clustering is especially useful when applied to web pages, which is explored in Carpineto et al. [2009]. One particular clustering algorithm that has been the subject of a lot of research is K-means, with a huge amount of published research. Most especially, at large scale as in Bahmani et al. [2012], Chitta et al. [2011].

Classification

Classification can be applied to an incredibly wide range of problems. Classification can be used to sort news stories, identify spam or classify medical images (e.g. cancer vs. not-cancer).

The most natural use for classification is documents classification, and this is still an active area of research ([Akritidis and Bozanis, 2013]). But beyond document classification, this versatile machine learning technique is also applied successfully for fraud detection in Choi et al. [2013], language detection in Baykan et al. [2013] or even music in Kotsifakos et al. [2013].

The medical field can also benefit a lot by the use of machine learning. Classification techniques applied to this field can be found as early as 2002 ([Guyon et al., 2002]). Recent research shows classification used in medical imaging ([Wang and Summers, 2012, Singh and Chetty, 2012]) and cancer diagnosis ([Kim et al., 2013, Rathore et al., 2013]) among others.

We would be remiss if we did not mention deep learning, One of the most exciting and promising field of machine learning must be *deep learning*. Deep learning is a highly complex unsupervised machine learning technique pioneered by Joshua Bengio (University of Montreal), Jeffery Hinton (University of Toronto) and Yan LeCun (NYU). It learns feature representations is a way that emulates the human brain (or at least, it is is thought to do so).

Deep learning, also known as deep belief networks or deep convolutional neural networks, have been applied to fields like computer vision, automatic speech recognition, natural language processing, and music/audio signal recognition where they have been shown to produce state-of-the-art results on various tasks.

Google has an active research teach working on deep learning. One of their recent, influential papers showed results where they trained a machine to recognize cats in youtube videos ([Le et al., 2012]).

Good introductions of this techniques can be found in Bengio et al. [2012] and in Arel et al. [2010].

For the latest on this exciting, but very complex field of applied machine learning, see `http://deeplearning.net` (Feb 2014).

3.3 Supervised Machine Learning Fundamentals

3.3.1 The Supervised Machine Learning Problem

The goal of supervised machine learning methods is to learn a model from the labeled examples. The learned model would then be able to predict with high precision the correct label of new, previously unseen data (examples).

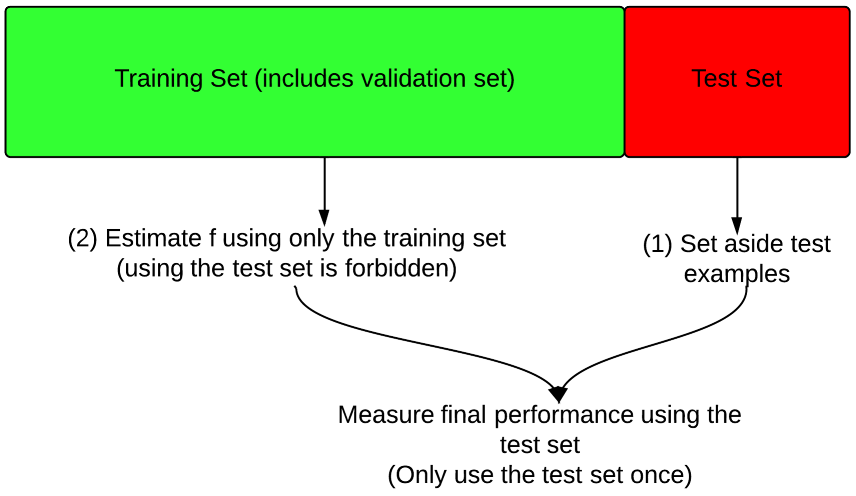

Training Set (includes validation set)
Test Set
(2) Estimate f using only the training set (using the test set is forbidden)
(1) Set aside test examples
Measure final performance using the test set (Only use the test set once)

Figure 3.2: A high level view of the usual experimental process when using supervised machine learning techniques (source: Leon Bottou)

Solving a problem using supervised machine learning will usually follow the process outlined in Figure 3.2. As it's supervised machine learning, by definition we will start with a dataset of n labeled examples of m *features*. Each example $x \in X$ is a *vector* of m real values.

Example This is an individual data item of the dataset. It could be a document in a collection of documents, or an individual measurement from a larger collection produced by a sensor. It is usually modeled as a vector of features.

Label (taget) This is the category of an example. If an example is an email, it's label could be spam or not spam.

A (real) vector is a collection of real numbers, referred to as the components (or, elements) of the vector; $\mathbb{R}^n$ denotes the set of vectors with m elements. If $x \in \mathbb{R}^m$ denotes a vector, we use subscripts to denote elements, so that x_i is the i-th component of x.

Feature A numeric or categorical quantity. Each feature models a characteristic of the example in some way, like temperature, size. Features can also be produced by an algorithm from a non-numerical input document (i.e. raw data).

P(x, y) distribution Is the "true", unknown distribution that generates the correct label y for any valid input x.

Loss Function This is a non-negative function that measures the disagreement between its arguments. Noted as $\ell(\widehat{y}, y)$, the loss function measures the cost of making a mistake. i.e. predicting $\widehat{y}$ when the true answer is y. A common loss is function is **0-1 Loss**, which returns 0 when the arguments agree, and 1 otherwise.

Risk $R(f, P)$ is the expected risk for data drawn from distribution P. In other words, it measures the probability of $f(x)$ predicting the incorrect label $\widehat{y}$ as measured by a given loss function. As the true probability distribution P is unknown, the expected risk cannot be calculated precisely.

$$E(f) = \int \ell(f(x), y) dP(x, y) \tag{3.1}$$

Empirical Risk is an approximation of the expected risk. It is the average loss of an estimator for a finite set of data drawn from P.

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x), y) dP(x, y) \tag{3.2}$$

Generalization Given a previously unseen example x, find a suitable label y. Good generalization performance is the goal of supervised machine learning.

Independent and identically distributed hypothesis (i.i.d.) We suppose that the examples of the train and test sets are picked independently from the $P(x, y)$ distribution described above. While this hypothesis is almost certainly wrong in some degree in practice, it is mathematically useful and, if sufficiently true, will still lead to meaningful results. Some consideration to the i.i.d. hypothesis is necessary when looking at any new data set, or the results may be very unreliable.

Each example is associated with a label $y \in Y$ where the i^th example x_i is associated with the label y_i. The model to be learned is a function f such that $f(x) = y$. This is an unknown function that may or may not exist.

Unfortunately, finding f, which represents the best possible model, is not possible given that we do not know the true probability distribution. In practice, we will settle for an approximation which we will call $f*$.

We assume that examples are drawn independently from an unknown probability distribution $P(x, y)$ that represents the laws of Nature. This is known as the independent and identically distributed (i.i.d.) hypothesis. Note this is not usually strictly true in practice.

To learn the function f, supervised machine learning can often be expressed in terms of an optimization problem. The goal of the optimization problem is is to find the function f that minimizes the expected risk.

The idea of risk minimization is not only measure the performance of an estimator by its risk, but to actually search for the estimator that minimizes risk over distribution P!

$$min_f \quad E(f) = \int \ell(f(x), y) dP(x, y) \tag{3.3}$$

Naturally, f^* gives the best expected performance for loss ℓ over the distribution of any estimator in $\mathcal{F}$.

It is not feasible to search f^* among all possible functions. Instead, we search for $f^*_{\mathcal{F}}$ that minimizes the Expected Risk $E(f)$ within some parametrized family of functions $\mathcal{F}$.

It is also not feasible to minimize the expectation $E(f)$ because $P(x, y)$ is unknown. Instead we search f_n that minimizes the Emprirical Risk $E_n(f)$. In other words, the average loss as computed by the chosen loss function, over the training set examples.

$$min_{f \in \mathcal{F}} \quad E_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i) \tag{3.4}$$

This general learning technique is called empirical risk minimization. Under the appropriate conditions, empirical risk minimization is known to converge $(f_n \longrightarrow f^*)^2$. The convergence is generally at a rate of $1/\sqrt{n}$.

To prevent overfitting (explained in subsection 3.3.6), the risk is often regularized to penalize complex hypotheses; this is called regularization:

$$f_n = min_{f \in \mathcal{F}} \quad E_n(f) + R(f) \tag{3.5}$$

We presented here a very brief problem statement on statistical machine learning for supervised methods. A much more complete and rigorous theoretical explanation can be found in Hastie et al. [2009].

3.3.2 Training Set and Test Set

The training set is used to train out supervised algorithm. It is a list of labeled examples where each example is modeled as a numerical vector, as we have explained above.

Once the supervised algorithm is trained, the next order of business is to evaluate it's performance. This is done by comparing the predicted label of the trained model on data for which the known label is known. However, we cannot re-use the same data for evaluation that was used to train the model. For evaluation (or testing) purposes, a second set of labeled examples becomes necessary.

For this purpose, the complete set of training examples is split into two sets at the very beginning:

train set this is the set which is used to find the classification function f_n.

test set this set of examples, for which the correct class is known is used for testing the effectiveness of the classifier. Each example of the test set is given to the classifier, and the classifier's decision is compared with the actual class of the example.

[2]The proof for this statement is beyond the level of this paper, but it can be found in Hastie et al. [2009]

Note that the accuracy of the predictions of the trained model on the test set can give some guarantees on the risk of that model on any data from the same distribution.

It bears to reiterate this important point: the examples from the test set cannot be used in any way in the process of learning the classification function f_n. This is a classic error of applied machine learning known as "peeking".

Peeking is a problem because if any information from the test set is presented to help learn the classification function, then its performance will be improved on the test set. It's like a student having a cheat sheet of answers when taking a test!

Supervised typically have tunable parameters that can have an important impact on final performance. How can the algorithm find the best possible parameters without ever using the test set examples? There are a number of possible approaches to this problem, we present here the most standard approach called *k-fold cross validation* in the next section.

3.3.3 Cross Validation

Cross Validation for Model Evaluation

Cross-validation is a technique evolved from the field of statistics and is used to assess how the results of an analysis will generalize to an independent data set.Since it is mainly used in settings where the goal is prediction, it is very applicable to machine learning problems.

Cross validation helps to estimate how accurately a predictive model will perform in practice. In a sense, it is an alternative to using a seperate test set, as we explain in subsection 3.3.2.

The goal of cross validation is to define a dataset to test the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting (subsection 3.3.6) and give an insight on how the classifier will generalize when used in production.

One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

K-fold cross-validation

In k-fold cross-validation, the data set is shuffled and then split into k partitions, or folds, of equal size.

Repeat k times for each of the k folds:

1. A model is trained using k-1 of the folds as training data
2. The resulting model is validated on the remaining part of the data according to some metric (see section 3.5)

The estimated performance reported by k-fold cross-validation is then the average of the values computed over all loops.

The advantage of this method over repeated random sub-sampling is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter. Figure 3.3[3] illustrate the process over 4 folds.

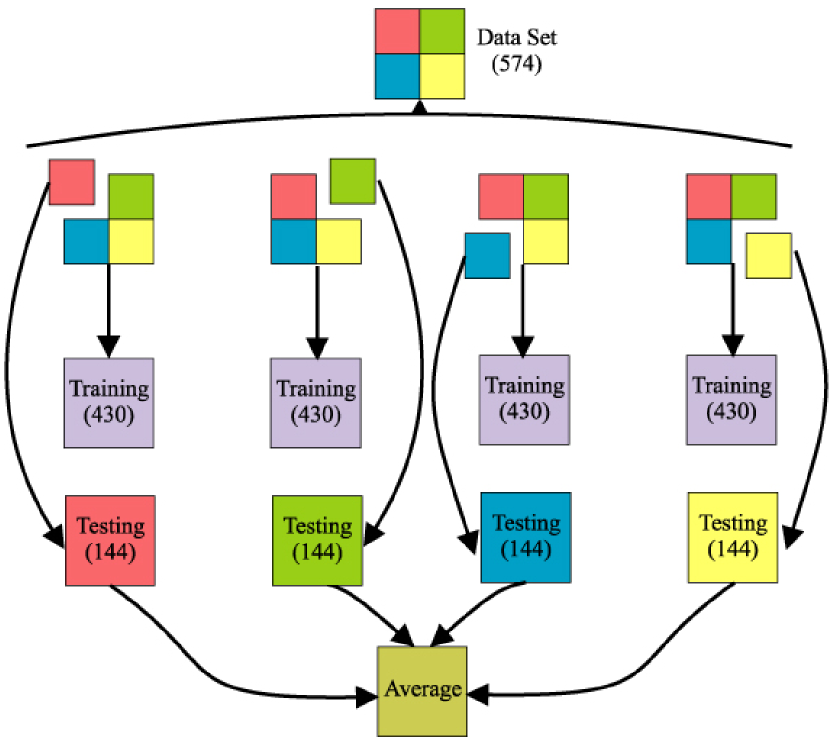


Figure 3.3: Cross Validation with 4 folds (source: Ruiz et al.

There are many variations on the idea, k-fold is but one of them, albeit the most popular in practice. In the machine learning library Sci-kit Learn, there are seven variations offered[4]: K-

[3]source: Ruiz et al. [2013]
[4]`http://scikit-learn.org/stable/modules/cross_validation.html` (Feb 2014)

fold, Stratified k-fold, Leave-One-Out, Leave-P-Out, Leave-One-Label-Out, Leave-P-Label-Out, and Random permutations cross-validation a.k.a. Shuffle & Split.

Cross Validation for Parameter Tuning

Most machine learning algorithms have parameters that can be tuned. SVM, with the RBF kernel has two: C and α. For a given dataset, it's unknowable *a priori*, what the best values for this parameters will be. Keep in mind that the best parameters will minimize the expected rate of errors of the learned model on new data. Finding the best possible parameters can be a critical factor in the final performance of the classifier.

When evaluating different settings, also called "hyperparameters", for the learning algorithm, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This is an important methodological mistake. The hyperparameters must be tuned without touching the test set in any way.

To solve this problem, one technique is to split yet another part of the dataset off as a so-called "validation set". Training proceeds on the training set, after which algorithm parameters can be found by maximizing the classifier's performance on held-off validation set. Final evaluation is then performed on the test set.

However, by partitioning the available data into three sets (train, validation and test), we greatly reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets. This is especially a problem when the number of labeled examples is relatively small, in the order of a few hundred examples or less.

A solution to this problem is to come back to cross-validation, but this time re-purposed to tuning the parameters of the classification algorithm. A test set should still be held out for final evaluation, but the validation set is no longer needed when doing cross validation. k-fold cross validation, as explained above, is a popular instance of cross validation.

While computationally expensive, this process makes full use of the training data, both to train the classifier and evaluate it.

3.3.4 Training and Testing the Classifier

Supervised ML applications usually consist of two phases: The training phase and the testing phase, as illustrated in Figure 3.4.

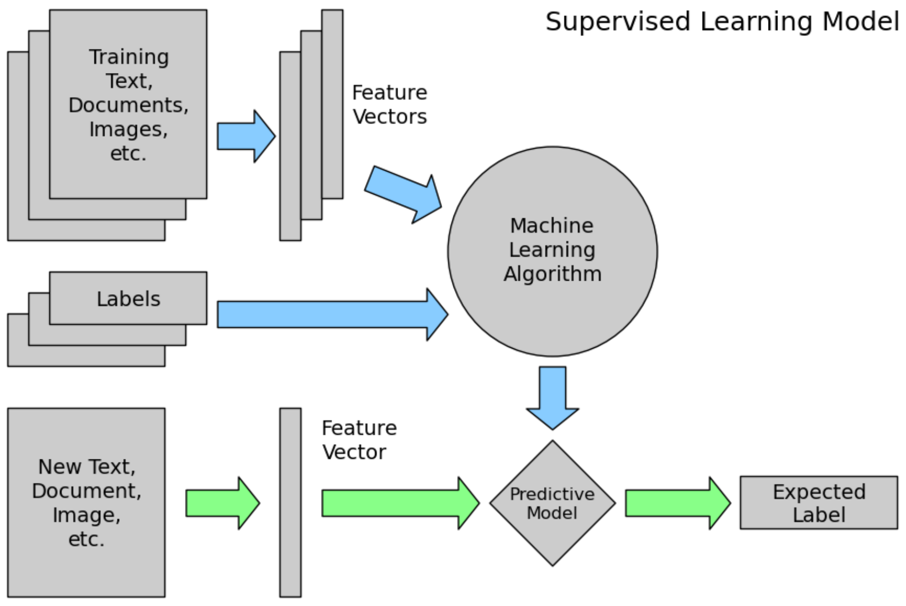

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Figure 3.4: Flow diagram for supervised learning (source: Scikit-Learn)

The first phase is called the training phase. In the training phase, the learning algorithm will build a predictive model using the input examples from the training set (represented in the form of feature vectors), using their known labels to guide the process. Note that the learning algorithm's best parameters are found using cross-validation. The final predictive model, the classifier, is trained using those best parameters and using all the labeled training data.

In the testing phase, the best classifier is used to predict the previously-unseen data from the testing set. That is, the testing data input is used as an unlabeled test case, and the aim is to determine its expected label $\hat{y}$. As we have the actual labels Y_test for the training set X_test, it will be possible to compare the prediction of the model with the actual label, and produce metrics. The topic of metrics is examined in some detail in section 3.5.

3.3.5 Feature Selection

Feature selection techniques focus on constructing and selecting subsets of features that are useful to build a good predictor. They are an essential part to most machine learning pipelines.

There are many potential benefits of variable and feature selection: facilitating data visualization and data understanding, reducing the measurement and storage requirements, reducing training and utilization times, defying the curse of dimensionality to improve prediction performance.

We can view feature selection as a method for replacing a complex classifier (using all features) with a simpler one (using a subset of the features). It may appear counterintuitive at first that a seemingly weaker classifier is advantageous in statistical text classification, but as discussed in the bias-variance tradeoff in subsection 3.3.7, having few features can indeed do so.

Some of the classical and most used feature selection algorithms are χ^2, Information Gain (IG) and Mutual Information. Seminal studies feature selection were done by Rogati and Yang [2002] and especially Forman in Forman [2003]. The latter is especially recommended for its clarity as well as authoritative treatment of the topic.

3.3.6 The Problem of Overfitting

Overfitting "In statistics and machine learning, overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model which has suffers from overfitting will generally have poor predictive performance, as it can exaggerate minor fluctuations in the data"[5].

The concept of overfitting is important in machine learning, especially for the supervised kind where a model is learned from a set of examples of which the answer is already known. If the model becomes "too good" at predicting correctly the answers of the training set, there is absolutely no guarantee it will perform at the same level when given new data. This is classic overfitting. The goal of performing well "in general", i.e. generalization is not reached.

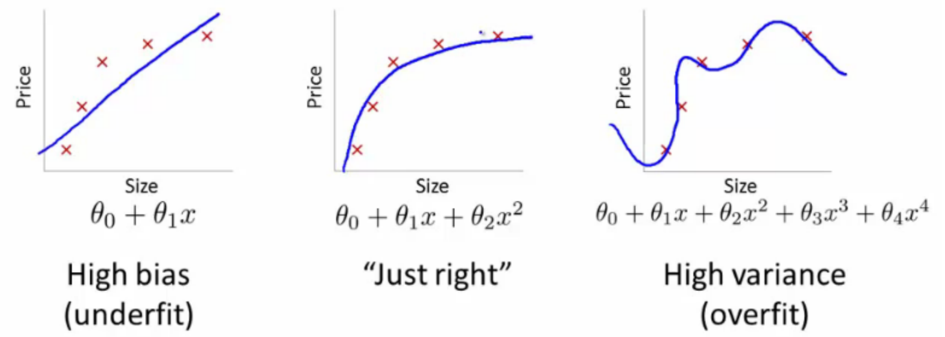


Figure 3.5: From underfitting to overfitting the model (source: Andrew Ng)

Overfitting can happen especially in situations where the training set is done too many times on the same data, which is itself often a consequence of having too few training examples to

[5]`http://en.wikipedia.org/wiki/Overfitting` (Jan 2014)

work with. But overfitting can also be caused randomly where a specific set of parameters are learned from the training data that just get unusually good performance on the test set, despite having no causal relation to the target function. The graph in Figure 3.5 is a good visual example of an algorithm likely overfitting its training data. The performance on the training examples (test set) increases while the performance on unseen data (test set) becomes worse.

More formally, a learning algorithm is said to overfit relative to a simpler one if it is more accurate in fitting known data but less accurate in predicting new data. In the case of supervised learning, a classifier can be tuned until it can classify perfectly or near enough all examples of the training set, but will make perform a lot worse on the test set, as well as on new data if the model is put into production. This is an undesirable result.

Understanding this topic, as well as the next on Bias vs. Variance, is of considerable importance for correct practical application of machine learning to real-world problems. The best treatment of this topic, in our opinion, is given by the deservedly famous Andrew Ng in his seminal class on machine learning available online for free from Coursera.org.

3.3.7 Bias vs. Variance Tradeoff

Bias and variance are sources of error typical, but not limited to, supervised learning tasks such as classification. Bias relates to the ability of your model function to approximate the data, and so high bias is related to *under-fitting*. On the other hand, variance is about the stability of your model in response to new training examples. High variance is related to *over-fitting* while low variance means high precision (see Figure 3.6.

Understanding how different sources of error lead to bias and variance helps us improve the data fitting process resulting in more accurate models. We liberally quote from the excellent blog post on this topic by Scott Fortmann-Roe[6].

Error due to Bias "The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict."

Error due to Variance "The error due to variance is taken as the variability of a model prediction for a given data point. "

[6] `http://scott.fortmann-roe.com/docs/BiasVariance.html` (Feb 2014)

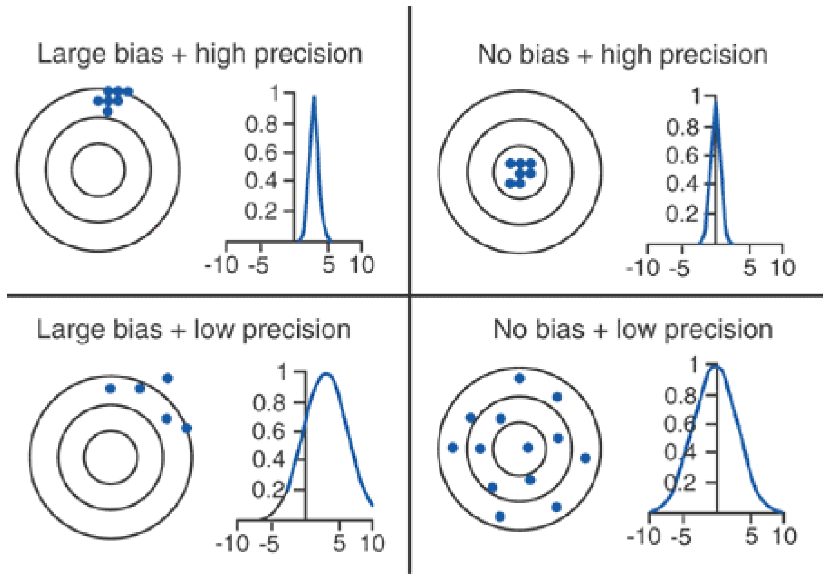


Figure 3.6: Bias vs. Variance Tradeoff (source: Scott Fortmann-Roe)

"Talking about the average prediction values might seem a little strange, since our task is to build a single classifier. However, imagine you could repeat the whole process many times. Due to randomness in the underlying data sets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value. The variance is how much the predictions for a given document vary between different versions of the trained classifier, obtained by randomly shuffling the documents each time it is retrained." (also from Scott Fortmann-Roe)

The figure 3.6 is a bulls-eye diagram. The target on the left shows some data points and how close they are to the goal at the center of the target. On the right, we can see whether the data points are clustered close together (a sharp point) or more spread about (a round hill) and whether the data is on target or not (the summit is at 0 or not). High bias means not getting the right answer, the dots are far from the target. High variance means the dots are very scattered.

The upper left target shows a base where the data cluster together, exhibiting low variance, but are far off target, meaning high bias. Inversely, the lower right target shows that while the dots are generally around the target, they are a bit more scattered, an example of low bias and high variance.

A source of bias is when the i.i.d. assumption is in fact wrong. This means the training set is not, in fact, a representative sample of the true distribution of documents in general. In other words, the documents may have been cherry-picked, knowingly or not.

A source of variance is usually, but not always, a sign of not having enough data, and so each time the documents are shuffled and a new classifier is trained, the results vary unpredictably. Increasing the sample size would make the estimates clump closer together.

In general the data set used to build the model is provided prior to model construction and the modeler cannot simply say, “Let’s increase the sample size to reduce variance.” In practice an explicit tradeoff exists between bias and variance where decreasing one increases the other. Minimizing the total error of the model requires a careful balancing of these two forms of error.

The tradeoff between bias and variance, and many learning algorithms have a built-in way to control this tradeoff, like for instance a regularization parameter that penalizes complex models in many linear modeling type approaches. Two examples might be the C parameter for SVM, or the K value in the K-NN algorithm. For a complete and formal treatment of this topic, see Hastie et al. [2001].

The excellent Andrew Ng, in his now famous Coursera MOOC class on Machine Learning, talks in detail of this issue and how to deal with it in the context of a machine learning task[7].

Common techniques

A popular feature selection method is χ^2. In statistics, the χ^2 test is applied to test the independence of two events, where two events A and B are defined to be independent if $P(AB) = P(A)P(B)$ or, equivalently, $P(A|B) = P(A)$ and $P(B|A) = P(B)$. In feature selection, the two events are occurrence of the term and occurrence of the class. Computing this probability allows the terms to be ranked.

Another common method is Information gain (IG), which measures the amount of information in bits about the class prediction, if the only information available is the presence of a feature and the corresponding class distribution. Concretely, it measures the expected reduction in entropy.

The best features have the most information. With IG, as with χ^2, features can once again be ranked and the worse features can be easily eliminated.

There are several other methods presented in Forman [2003], as well as others in the scientific literature. The important point here is to understand how feature selection is not a one-size-fits-all approach. Each technique will perform better or worse on a given data set and only

[7] `http://www.youtube.com/watch?v=g4XluwGYPaA` (Feb 2014)

experimental data will help the practitioner chose the best feature selection technique to use on their own particular problem.

3.4 Supervised Learning Algorithms

First, a brief recap of notation which is standard for describing machine learning algorithms. As described above in subsection 3.3.1, a supervised machine learning problem involves trying to estimate a function composed of a set of parameters θ, that maps a vector of features, x, onto a scalar target, y. The set of vector/target pairs are indexed using a subscript such that the i^th example (or observation) is referred to as (x_i, y_i).

3.4.1 Support Vector Machines (SVM)

Support vector machines are a instance of supervised learning. They are mainly used for classification and regression analysis.

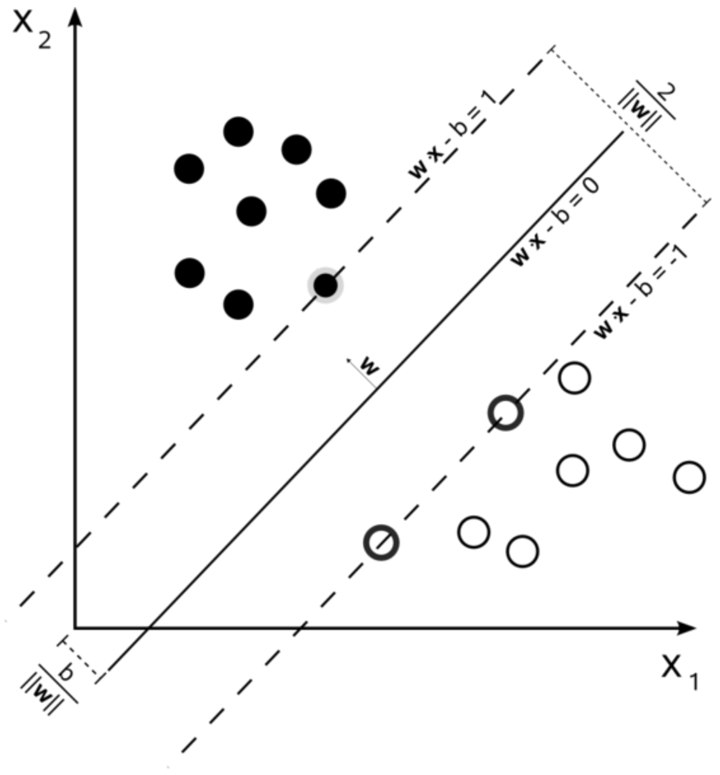
Intuitively, {0,1}, or binary classification can be seen as trying to split data (or observations, or examples) points such that points on one side are labeled as class 1 and those on the other side labeled as 0. We will call this separating line the decision boundary. When training a classifier using supervised methods, the goal is to feed correctly labeled examples to the algorithm so that it can learn the best possible decision boundary. such that the training data will be neatly separated on either sides of the boundary (see Figure 3.7).

From a high level perspective, the goal of classification is to learn a decision boundary such that *new* data will be correctly classified.

If the data used to train the classifier is correctly classified, but new data isn't, it's a case of *overfitting*, which we discuss in subsection 3.3.6. The problem therefore becomes to try and learn the best decision boundary using the training data, in order to correctly classify the training data **and** new data.

Without loss of generality, we can illustrate the problem in 2D. In that case, the simplest decision boundary will be a line. In 3D it would be a plane. More realistically, for data with hundreds, thousands or even more features, we will call the decision boundary a *hyperplane*.

Looking at the data in Figure 3.7, we can see that there are two easily identifiable groups. There are an infinite number of straight lines that could correctly split the red and blue data points so that all the points of the same color are on the same side of the line.



X_2
X_1
$\mathbf{w}\cdot\mathbf{x} - b = 1$
$\mathbf{w}\cdot\mathbf{x} - b = 0$
$\mathbf{w}\cdot\mathbf{x} - b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Figure 3.7: Maximal margins for SVM trained with the data shown. The points on the margins are called the support vectors (source: Wikipedia)

Intuitively, most people will put the line in the middle of the groups such that the line is maximally far from either groups. This allows for the largest *margin* between them. This is the core concept behind the Support Vector Machine (SVM) supervised learning algorithm introduced by Vapnik in Cortes and Vapnik [1995].

The SVM algorithm looks for a decision boundary, or hyperplane (in more than three dimensions), such that the boundary is maximally far from any data point. This distance from the boundary to the closest data points determines the *margin* of the classifier.

The data points that are right on the edge of the margin are called the *support vectors*, and they fully define the position of the separator. Once the support vectors are found, all other data points become irrelevant to determining the decision boundary.

The margin is where the classification power of SVM comes from. Indeed, maximizing the margin seems a good way to increase the confidence of classification decisions, as new data is most likely to be farther from the decision boundary. Near the decision boundary, a very small change in the data values may change the predicted class, meaning the prediction becomes a 50-50 chance to be either class.

The Figure 3.7shows that as data get father away from the margin, there is more of a safety margin, meaning a slight error or variation in the data will not cause a mis-classification. Thus, we expect the SVM classifier to generalize well.

SVM More Formally

We will now give a more mathematical explanation. the optimization problem for a linear SVM is as follows (also illustrated in Figure 3.7)

We would like to find the function which minimizes an objective like:

Training Error + Complexity term

We write that as:

$$\frac{1}{n}\sum_{i=1}^{n}\ell(f(x_i,\alpha),y_i) + Complexityterm \tag{3.6}$$

Where the function f is now parametrized with α.

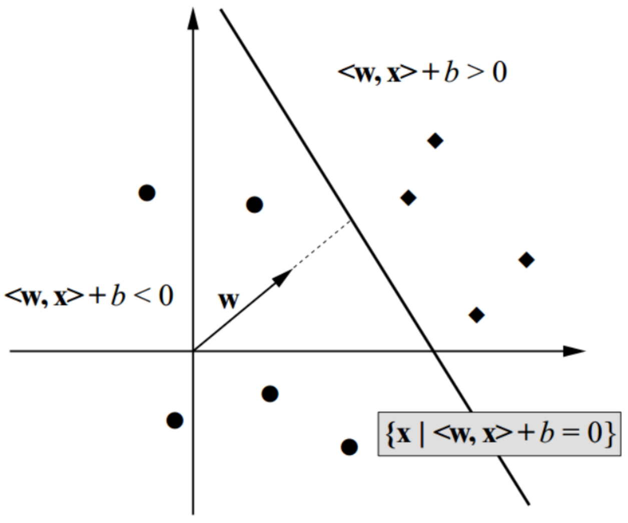


Figure 3.8: The term w is a vector that describes a hyperplane with offset b from the origin (source: Wikipedia)

Any hyperplane can be written as the set of points x in a hyperspace with n dimensions satisfying $w\dot{x} - b = 0$ (see Figure 3.8).

For now we will choose the set of hyperplanes so $f(x) = (w\dot{x}) + b$:

$$\frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i, \alpha), y_i) + \|w\|^2 \tag{3.7}$$

subject to $min_i|w\dot{x}_i| = 1$.

Assuming we can separate the data perfectly (for now). Then we can optimize the following quadratic program:

Minimize $\|w\|^2$, subject to:

$$(w\dot{x}_i + b) \geq 1, if y_i = 1(w\dot{x}_i + b) \leq 1, if y_i = -1$$

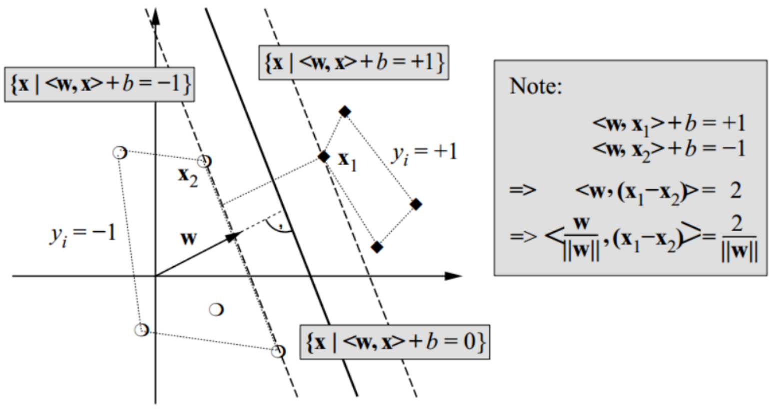


Figure 3.9: The margins are constructed from a linear combination of examples x. The support vectors are those examples right on the margin.

The last two constraints can be compacted to:

$$y_i(w\dot{x}_i + b) \geq 1$$

This is illustrated in Figure 3.9

Sometimes, it is not possible (or desirable) to perfectly separate the two classes. One common reason is when there are mislabeled examples, another is just because the problem itself is difficult.

To deal with the non-separable case, one can rewrite the problem as: Minimize:

$$\|w\|^2 + C \sum_{i=1}^{n} \xi_i \tag{3.8}$$

subject to:

$$y_i(w\dot{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0$$

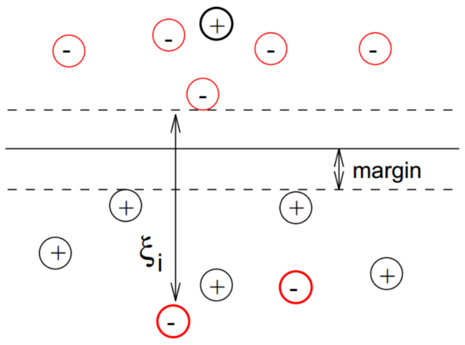


Figure 3.10: When the data is not linearly separable, we must tolerate incorrectly classified examples, but with a penalty ξ.

As shown in Figure 3.10, ξ_i is the distance of an incorrectly classified example to the separation hyperplane.

This is the same as the original objective from Equation 3.7, except ℓ is no longer the 0-1 loss, but is called the *hinge loss* as shown in Figure 3.11:

$$H_1 = \ell(y, \widehat{y}) = max(0, 1 - y\widehat{y}) \tag{3.9}$$

SVM can be reformulated in what is known as its Primal form. The factor of $\frac{1}{2}$ being used is for mathematical convenience without changing the solution:

$$minP(w, b) = \frac{1}{2} \|w\|^2 + C \sum_{i} H_1 \left[y_i f(x_i) \right] \tag{3.10}$$

Where the first term maximizes the margin and the second minimizes the training error.

For the non-linear case, where the linear SVM is not complex enough to classify the examples well, we use what is know as the Kernel Trick. This technique maps data into a richer set of features where the data may become once again linearly separable in the new, higher dimensional space.

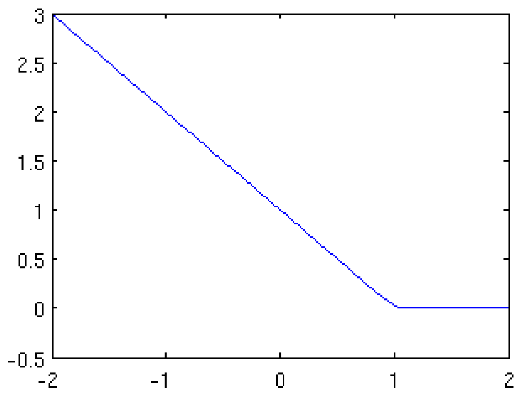


Figure 3.11: The hinge loss owes its name to the graph, with a clear hinge at 1

In this case, we refer to the SVM's Dual form. The usual formulation is as follows: Maximize $(in\alpha_i)$:

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{3.11}$$

subject to (for any $i = 1, \ldots, n$)

$$* \, \alpha_i \geq 0 and \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3.12}$$

The kernel could be the linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$.

Other common kernels are the polynomial ($k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^d$) or the Gaussian radial basis function (RBF) kernel: $k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2), for \gamma > 0$. Usually parametrized using $\gamma = 1/2\sigma^2$.

The Radial Basis Function (RBF) Kernel

The advantage with the RBF kernel is that the higher dimensional space becomes infinite, making it a very powerful and versatile algorithm.

When training an SVM with the Radial Basis Function (RBF) kernel, two parameters must be considered: C and γ. The choice for these parameters has a large impact on the final performance of the SVM classificator.

The parameter C, common to all SVM kernels, trades off mis-classification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. gamma defines how much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.

3.4.2 Logistic Regression

Logistic regression is a simple model for predicting a probability of an event and can be readily adapted to binary classification. In this case, we are interested in finding the probability $P(y=0|x)$ and $P(y=1|x)$ where the two probabilities sum to 1.

Interestingly, the name of logistic regression is somewhat misleading. It's not really doing a regression. The word regression rather comes from fact that it fits a linear model to the feature space.

Given two outcomes p and q where $P(y=1|x_1)=p$ and $P(y=0|x_2)=q$, then we can express the probability of $y=1$ as:

type of probability	notation	range equivalent		
standard probability	p	0	0.5	1
odds	pq	0	1	$+\infty$
log odds	log(pq)	$-\infty$	0	$+\infty$

Numeric treatment of outcomes $y=1$ and $y=0$ is equivalent: if neither outcome is favored over the other, then log odds will be 0. If one outcome is favored with log odds >0, the other outcome is disfavored with log odds <0.

Logistic regression represents log odds of the event as a linear model, if p is the probability $P(y=1)$:

$$\log(\frac{p}{1-p}) = \theta \cdot x \tag{3.13}$$

The logistic regression model consists of a vector θ in an n-dimensional feature space as described in subsection 3.3.1.

The *logistic* of logistic regression refers to the logistic function (Figure 3.12), which can be easily shown to be equivalent to the log odds equation (Equation 3.13):

$$f(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}} \tag{3.14}$$

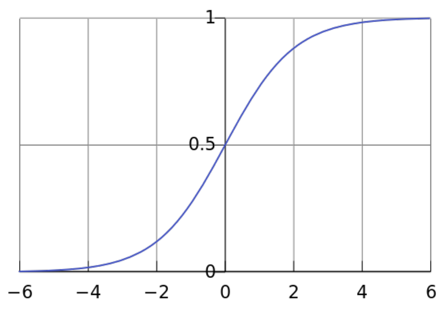


Figure 3.12: The standard logistic function (source: Wikipedia)

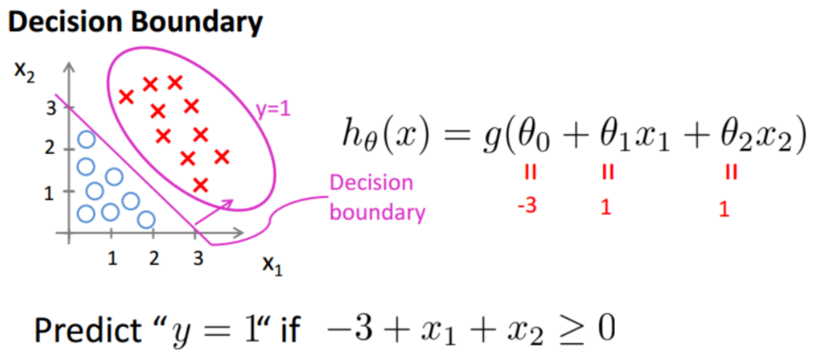
And so we have the hypothesis h:

$$h_theta(x) = f(\theta \cdot x) = \frac{1}{1 + e^{-\theta \cdot x}} \tag{3.15}$$

We can interpret the output of the hypothesis as the estimated probability that $y = 1$ on input x with parameter vector (i.e. model) θ.

The decision boundary is usually chosen such that logistic regression will predict: $y = 1 if h_theta(x) \geq 0.5$ and $y = 0$ if $h_theta(x) < 0.5$

By changing the decision boundary it becomes possible to change the degree of certainty required for a prediction.



Decision Boundary
x2
3
2
1
1
2
3
x1
y=1
Decision boundary
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
-3
1
1
Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

Figure 3.13: The decision boundary of logistic regression in two dimensions (source: Andrew Ng)

For example, in Figure 3.13 we can see that one possible decision boundary for a two dimensional space.

The question now becomes, how can we find the values for θ to get optimal classification? One well-known technique is gradient descent, which we explain below.

3.4.3 Learning with Gradient Descent

"Regression is a well known technique from the world of statistics that models the relationship between a scalar dependent variable y and one or more explanatory variables denoted X[8]."

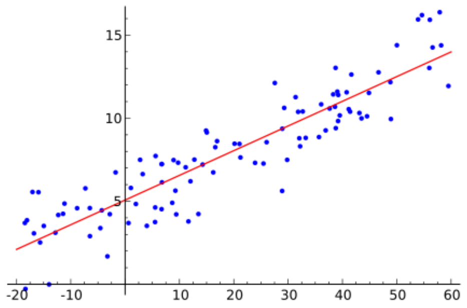


Figure 3.14: Linear Regression fits a line through the data (source: Wikipedia)

Gradient descent is a first-order optimization algorithm. Also known as steepest descent, or the method of steepest descent, it finds a local minimum of a function by making a series of steps that follow the negative of the gradient (or of the approximate gradient) of the function at the current point.

We are interested in the problem of supervised learning as covered in subsection 3.3.1. Note that the exposition of this technique is a summary of the Bottou [2010].

To briefly recapitulate: Given an example $z = (x, y)$ we choose a family $\mathcal{F}$ of functions $f_w(x)$ parametrized by a weight vector w. We seek the function $f \in \mathcal{F}$ which minimizes the loss $Q(z, w) = \ell(\widehat{y}, y)$ averaged on the examples. Minimizing the empirical risk $E_n(f)$ as a target over the samples $z_1, ..., z_n$ instead of the expected risk $E(f)$is well known in statistical learning theory[9].

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)) \tag{3.16}$$

Using gradient descent to minimize the empirical risk as a supervised learning algorithm is a well established technique (e.g. [Rumelhart et al., 1988]). Each iteration updates the weights w on the basis of the gradient of the empirical risk.

$$w_{t+1} = w_t - \gamma\frac{1}{n}\sum\nolimits_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{3.17}$$

[8]`http://en.wikipedia.org/wiki/Linear_regression` (Feb 2014)

[9]Which dates back to the 1970's and Vapnik's pioneering work

where γ is a parameter that can be used to tune the learning speed. It can be shown that under sufficient regularity assumptions, this algorithm converges linearly.

There are many variations such as *second order gradient descent* (2GD) which is a variation on the well known Newton's algorithm.

3.4.4 Stochastic Gradient Descent

The *stochastic gradient descent* (SGD) is a simplification of gradient descent[10]. Gradient descent computes the gradient of the empirical risk, whereas SGD settles for an estimate of the gradient based on a single randomly chosen example z_t:

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t) \tag{3.18}$$

z_t is chosen randomly at each iteration, hence the name of SGD. The idea is that the update rule Equation 3.18 will behave like Equation 3.17 despite some additional noise due to the simplification.

The new update rule makes each update independent of previous iterations, as there is now no need to remember which examples were visited previously. We can see SGD as minimizing directly the expected risk, because examples are chosen randomly from the ground truth distribution. Its convergence is well studied in the field of optimization. Bottou establishes almost certain convergence under mild conditions in Bottou [1998].

3.5 Classification Metrics

Now, we come to the evaluation of the classifier system. This is a critical step and requires some careful consideration as the choice of metric has a large impact on how the system itself is built, and not only on determining whether it's performance is satisfactory or not. Also, the choice of metric must be made with some consideration of the classification task itself, different measure optimize for different performance.

3.5.1 Confusion Matrix

When referring to the performance of a classification model, we are interested in the model's ability to correctly predict or separate the classes. When looking at the errors made by a classification model, the confusion matrix gives the full picture. Consider e.g. a three

[10]qualified as "drastic" by Bottou ([Bottou, 2010])

class problem with the classes A, B, and C. A predictive model may result in the following confusion matrix when tested on independent data.

Predicted class	Known class A	B	C
A	25	5	2
B	3	32	4
C	1	0	15

Table 3.1: A confusion matrix for a three class problem (classes A, B and C)

The confusion matrix from Table 3.1 shows how the predictions are made by the model. The rows correspond to the known class of the data, i.e. the labels in the data. The columns correspond to the predictions made by the model. The value of each of element in the matrix is the number of predictions made with the class corresponding to the column for examples with the correct value as represented by the row. Thus, the diagonal elements show the number of correct classifications made for each class, and the off-diagonal elements show the errors made.

3.5.2 Accuracy and Precision

Classification brings its own set of canonical metrics, evolved for the field of statistics. Accuracy and precision (Figure 3.15) are a good place to start, but need to be defined specifically in the context of a binary classification task:

Accuracy the proportion of correctly labeled examples, positive or negative, to the **total number of examples**.

Precision the proportion of the positive examples that were correctly labeled to all **positively labeled examples**.

To define accuracy and precision more formally, we need to give a more strict name to all possible labeling situations, staying within the binary classification problem. There are two classes, positive and negative, and for each, the classifier can either correctly or incorrectly label a given example.

true positives (tp) the number of items correctly labeled as belonging to the positive class.

false positives(fp) the number of items that are really in the negative class, but were incorrectly labeled as belonging to the positive class.

true negatives (tn) the number of items correctly labeled as belonging to the negative class.

false negatives(fn) the number of items that are really in the positive class, but were incorrectly labeled as belonging to the negative class.

These can be visualized in the following table:

		True Class of Example	
		True	False
Classification Result	Positive	True Positive	False Positive
	Negative	True Negative	False Negative

And precision and accuracy are then defined in Equation 3.19 and Equation 3.20, respectively.

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{fp} + \text{fn} + \text{tn}} \tag{3.19}$$

$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \tag{3.20}$$

The confusion matrix is a table with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives. It is a powerful visualization of the performance of a supervised binary classification task. According to Wikipedia, the name stems from the fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).

3.5.3 Precision and Recall

Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced. An dataset is said to be unbalanced when the number of samples of one class differs greatly from the other. This is a common problem in classification, especially in text classification.

The Reuters-21578 Text Categorization Collection Data Set[11] holds 20 categories, each with around 800-1000 examples, for a total of 18821 documents. A one-vs-all binary classification

[11] Available at the UCI KDD Archive (`http://kdd.ics.uci.edu/`)

task will have one category be the positive one and all other be negative. This means there is a 1-9 ratio of positive examples to negative ones. A null classifier, which predicts negative for all documents, will have an accuracy of around 90% for this problem. While the classifier might be very happy, the user will definitely not!

Precision and Recall

In information retrieval, precision and recall are commonly used as metrics to measure relevance.

Precision the fraction of retrieved instances that are relevant. In a classification task, the precision for a class is the number of true positives.

Recall the fraction of relevant instances that are retrieved. In classification, recall is the number of true positives divided by the total number of elements that actually belong to the positive class. Recall is also called sensitivity or true positive rate (TPR).

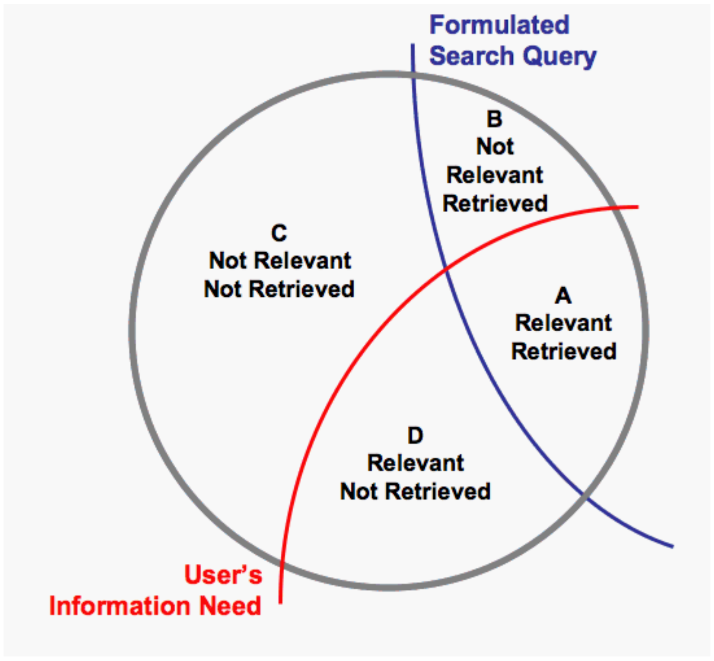


Figure 3.15: Precision and Recall (source: University of Toronto)

In Information Retrieval (a search on Google), precision and recall can be tied to the results of a query, as shown in Figure 3.15. A query will return a set of results (Formula Search Query in the figure), which is a subset of all possible relevant documents that exist (the user's information need in the figure). The ratio of the area A to the user's information need is the recall, whereas the ration of A to the result of the search is the precision.

Going back to the Reuter-21587 example, the null classifier's poor performance can be illustrated with recall, which will be zero, as no positive documents are found at all.

3.5.4 F^1 Score

A classifier with a high recall means that most of the relevant results where indeed correctly labeled, while high precision means that the classifier, overall, is making very few mistakes. Both measures can be blended into a third metric called the F^1 score.

In statistics, the F^1 score (also F-score or F-measure) is a measure of a test's accuracy (see Equation 3.21). It considers both the precision p and the recall r of the test to compute the score

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{3.21}$$

Using the F^1 score, it is possible to get a more balanced view of the performance of the classifier. An unbalanced classifier, whether with high precision and low recall or the opposite, will lead to a poor F^1 score. Mathematically, the only way to reach over 90% in F^1 score is to have **both** precision and recall as a similarly high level.

3.5.5 The Learning Curve

A learning curve is a plot of the prediction accuracy (or error) vs. the training set size. In other words, the plot shows how much better the model gets at predicting the target as you the increase number of examples used to train it.

These plots can give a quantitative view into whether it will be beneficial to add more examples to the training set used by the classification algorithm[12].

The learning curve is a very useful tool in helping to guide and inform the process of getting the best performance from a supervised machine learning solution.

On the left plot in Figure 3.16, we have the learning curve for a high-bias classifier. This is a problem of under-fitting. Both the training and cross-validation errors are very high. Even

[12]Explained in greater detail at: `http://www.astroml.org/sklearn_tutorial/practical.html` (Feb 2014)

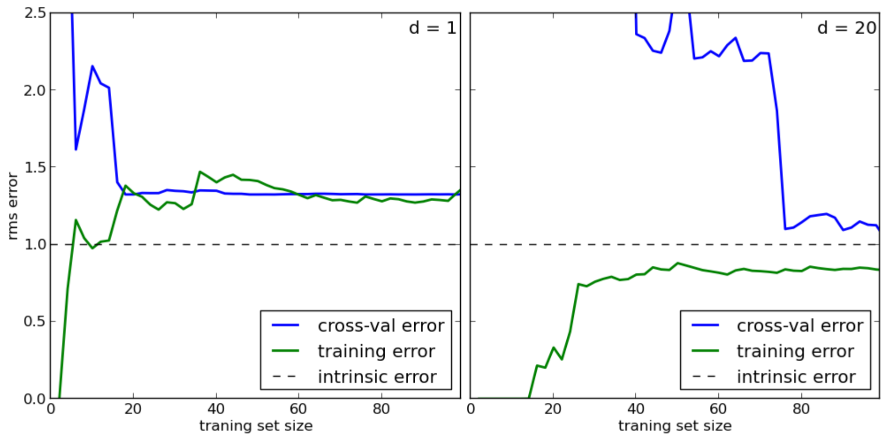


Figure 3.16: The learning curves can show problems of bias and variance (source: `www.astroml.org`)

if more training data is gathered, possibly at some cost, the classification performance is not likely to improve much. This is because both lines have converged to a relatively high error.

In the right plot, we have the learning curve for d = 20. The high-variance issue is indicated by the fact that the training error is much less than the cross-validation error. The model learned is over-fitting the data.

As we add more samples to this training set, the training error will continue to climb, while the cross-validation error will continue to decrease, until they meet in the middle. Adding more data should help the classification algorithm to more closely match the cross-validation error.

3.5.6 ROC Curve and AUC

The relationship between sensitivity and specificity, as well as the performance of the classifier, can be visualized and studied using the reciever operation characteristic (ROC) curve. ROC analysis is a classic methodology from signal detection theory used to depict the tradeoff between hit rates and false alarm rates of classifiers.

The ROC curve (Figure 3.17) is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. It plots the true positive rate vs. the false positive rate at various threshold settings. The technique is presented in detail by Fawcett in Fawcett [2006].

"A common method to transform ROC performance to a scalar value, that is easier to manage,

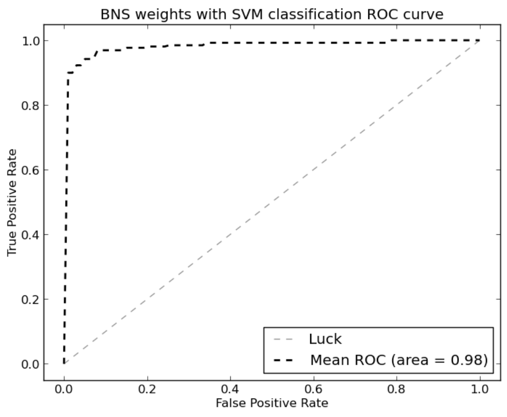


Figure 3.17: The ROC Curve helps to evaluate the performance of a classifier (made with R)

consists on calculate the area under the ROC curve (AUC). The area under the curve is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one."[13]

As the ROC curve is represented in a unit square, the AUC value will always be between 0.0 and 1.0, being the best classifiers the ones with a higher AUC value. As random guessing produces the diagonal line between (0,0) and (1,1), which has an area of 0.5, no real classifier should have an AUC less than 0.5.

The machine learning community most often uses the ROC AUC statistic for model comparison, although the metric does has its critics.

3.6 Machine Learning for Text Classification

The goal of text categorization is the classification of documents into a fixed number of predefined categories.

A useful analogy for this task is to think about an assistant sorting a stack of business documents such as email of two types, say "marketing" and "sales". The assistant will be shown some examples of each type, then asked to sort the lot of them.

Using machine learning, the objective is to learn classifiers from examples which do the same type of category assignments automatically. This is a supervised learning problem. Despite being a bit dated, Sebastiani [2002] is an excellent tour of the topic.

[13]`http://machine-learning.blogspot.ca/2008/07/auc-as-performance-metric-in-ml.html` (Feb 2014)

Documents might belong to one and only one category, known as a single-label case. The case where a document may belong to multiple categories is known as the multi-label case. The case where there are only two possible categories is known as binary classification.

As it is possible to use binary classification for multi-label classification, it is more general and is therefore the main focus of academic research. The reason this is so is because it is always possible to transform the multi-label problem into $|C|$ independent problems of binary classification, as long as all categories are stochastically independent of each other, a common assumption in practice [Sebastiani, 2002].

We also focus on binary classification because of its relevance for filtering problems, which naturally map to it (a document is either filtered or it is not).

3.6.1 Modeling Text as Vectors

The Bag-of-Words Model

The most common representation of documents used is the *bag-of-words* representation. It is a simplifying representation used in natural language processing and information retrieval (IR).

The bag-of-words is a sparse vector of occurrence counts of words [Salton and McGill, 1983]. Note that in computer science, a bag is a set that allows multiple occurrences of the same element, which makes perfect sense in this context.

Using a dictionary of words found in the collection of documents, we are then able to build a vector representation of the document with each word of the dictionary being associated with an index in the vector.

For a large, vocabulary-rich collection, we might have as many as a few tens of thousands of different words, meaning each document would be a few tens of thousands elements in length. The order of words is lost, as each index is associated to a word independently of any other, hence the *bag-of-words* metaphor.

To better understand, we illustrate the ideas of the bag-of-word with this example:

> A fool thinks himself to be wise, but a wise man knows himself to be a fool.

The row vector $x = [2, 2, 1]$ contains the number of times each word in the list fool, wise, man appear in the above quote. We can now show that this vector representation can become a powerful tool to perform computation and is much more readily used than the original text version. Note that the vector representation we just showed is not an exact copy of the text, since some words were removed. Common words (also known as *stop words*) are nearly always removed from the text before it is transformed into a vector and dictionary pair in order to focus on the words with the highest information value. In this vector form, the order of the words is lost. While this might seem like a step back, it turns out in practice that for most purposes, the exact order of words is not needed to get useful results.

It's important to mention, regarding stop words, that what constitutes a stop word is language and problem specific. Indeed, what is a stop word in English may not be one in French. As well, a word might be so common as to be useless in one context but be very significant in another.

Thus, as we have shown, vectors can be used to represent text documents. The representation, often referred to as the *bag-of-words*, is used as a basic text representation in general practice for text processing and especially for applying machine learning algorithms to text documents.

Vector Space Model

The vector space model (VSM) allows efficient analysis of text collections despite not using any explicit semantic information like grammar and word ordering. Originally introduced within the information retrieval community.

Today, it is a well established representation for text documents. It is commonly used in text mining, information extraction (IE), information retrieval (IR) and machine learning. The cited original paper most often cited with regards to the VSM is Salton et al. [1975]. However, according to Dublin in Dubin [2004], the VSM may have been more the result of a gradual evolution than of one singular *ah-ha!* moment.

Assume t distinct terms remain after prepossessing; call them index terms or the vocabulary V. These "orthogonal" terms form a vector space.

$$Dimension = t = |V| \tag{3.22}$$

Documents are expressed as t dimensional real valued vectors as per the bag-of-words model. The vocabulary is a dictionary that maps between the vector space representation of the documents and the actual words.

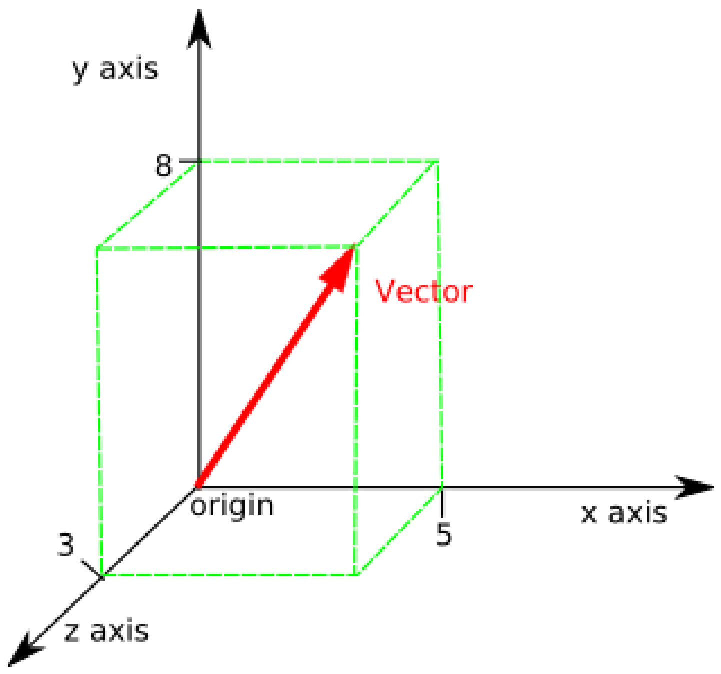


Figure 3.18: Each document is represented as a vector in the vector space model (source: Wikipedia)

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj}) \tag{3.23}$$

As shown in Equation 3.23, the i^{th} term in the j^{th} document (noted d_j) is given a real-valued weight, w_{ij}. The value for i is the index of the term in the list of vocabulary used in the document collection. Each index in the vector corresponds with the index in a list of words, the vocabulary V of the text collection. Therefore, $n = |V|$.

A term weight of zero means the term has no significance in the document or it simply doesn't exist in the document. On the other hand, a high term weight can be interpreted as a term of high information value with regards to this collection.

We can expect that most documents to be full of zeroes, meaning they will be sparse vectors. Why is that? The typical English document collections' vocabulary of 10,000 to 15,000 words. A short document of a few hundred words can't have more than just that, a few hundred non-zero elements in its vector space representation. Most machine learning libraries have optimizations for sparse vectors.

A collection of n documents can be represented in the vector space model by a term-document matrix, where each row is a document, and each column maps to a term in the vocabulary.

Documents can now be compared using simple and well known linear algebra operations, upon which are based most machine learning algorithms.

Tokenization

Tokenization is the process by which we cut the document into a series of individual tokens which roughly correspond to words. Depending on the tokenizer used, punctuation and white space may be remove. Proper tokenization is extremely application dependent and the techniques to perform them can be quite involved and sophisticated (see Huang et al. [2003]).

Let's take a simple example and see the difficulties emerge:

Love all, trust a few, do wrong to none.

Could be tokenized as so: Love-all-trust-a-few-do-wrong-to-none. If the punctuation is kept, the tokenized sentence becomes: Love-all,-trust-a-few,-do-wrong-to-none. But then, should "all" be treated differently from "all," or not?

Academic research has also been very much interested more difficult applications of this problem for languages like Chinese [Tsai, 2010] or Japanese [Ando and Lee, 2003] where there are no white space characters to help out (and the authors often can't read the language in question). Then there is the question of separating chucks within sentences and sentences themselves, also active research topics [Wu et al., 2006].

Filtering, Lemmatization and Stemming

According to the Global Language Monitor, the number of words in the English language is estimated to be slightly over 1M, as of January 1, 2012. This does not account for misspellings, neologisms and other variations found in real documents. Google's N-Gram Viewer project, with a scan of 20M books includes around half a billion words! If this is our dictionary size, then the dimensionality of our vectors will accordingly be of the same length. Sparse data structures can make the space requirements tolerable, but it may be possible to greatly reduce their size, or dimensionality through simple techniques we will outline here.

Firstly, we can limit the size of the vocabulary to the words present in the document collection. This simple technique on most corpus will probably reduce the vocabulary size to the tens of thousands, but this greatly depends on the collection, in terms for how it was created as well as its size.

Filtering methods will further remove words from the dictionary that are found in the document collection. The standard method is to filter out what are known as stop words. Stop words are words with little information content on their own (remember we are operating under the bag-of-words representation). While no definitive list exists for English, let alone other languages, most NLP packages include a stop-words list. More formally, stop words are words that occur very seldom or in nearly all documents such that they are likely to be of no particular statistical relevance and can be removed from the dictionary. Feature selection techniques can be applied to find additional such words (see section 3.3.5).

"Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. The processing requirements for performing lemmatization are relatively high and the error rates still not very low, so a less sophisticated approach is often substituted called Stemming."[14]

Stemming methods try to build basic forms of words and use a more crude method to strip words of affixes like 's' and 'ing'. A stem is a group of words that can be treated as having equivalent meaning, from the point of view of the classification task. The most well known stemming software is the Porter stemmer [Porter, 1980], which uses a rules-based approach to iteratively transform English words into their stems. Note that there is no one recognized stem form for all English words. Stemming should be rather understood as a best-effort technique that works well in practice.

3.6.2 Term Weighting Algorithms

Now we approach the problem of how to score the documents, which is to say, what value should be entered in the document vectors, such that classification performance will be maximized when the learning algorithm is trained on the vectorized collection of text.

Towards this end, we assign to each term in a document a weight for that term, that depends on the number of occurrences of the term in the document. We would like to compute a score between a term t and a document d, based on the weight of t in d.

Term Frequency

The simplest way of document encoding is to use a binary term vectors. In this simple encoding, each word of the document vocabulary counts for 1 in the document vector v at the words index in the collection dictionary. All other elements of the vector will be 0. This is the

[14]`http://en.wikipedia.org/wiki/Lemmatisation` (Feb 2014)

bag of word model at work: the word order, grammar and all is completely lost; all words are taken in isolation, present or not-present.

One easy improvement would be to add back in the information about words that appear numerous times in the same document. A text which mentions cats ten times and house once in a text of about 200 words is more likely to be about furry felines then about real estate. We are now applying the Term Frequency or tf scoring.

There are some problems with tf scoring. Remember that we want to classify the documents. To classify documents, the learning algorithm will look for words or combination of words that help it discriminate documents between two separate classes. But just term frequency, by itself, doesn't help a lot to discriminate documents because all terms are considered roughly equally important, for all documents.

$$\mathrm{tf(t, d) = f(t, d)} \tag{3.24}$$

Let us illustrate the issue with a more concrete example: We are given articles from the sports and the economy sections of a newspaper and wish to classify them. Intuitively, we'd expect that the term "GDP" is very likely to be from the economy section, and the word "baseball" from the sports section. On the other hand, the word "schedule" could be from either and is therefore of little discriminative value. Or if we want to classify documents from an IT firm in some way, the words "IT" or "computer" probably won't help us much to determine into which category a document should go, simply because it's likely those words are found in most documents of the collection.

Inverse Document Frequency (Idf)

Inverse document frequency is a way to help penalize those words that are present in most documents of the collection, which is to say, those words that are unlikely to help the learning algorithm classify documents. Documents with a high frequency across the entire collection of documents would see their weights reduced, ultimately down to zero if the term is found in all document of the collection.

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3.25}$$

Term Frequency-Inverse Document Frequency (Tf-Idf)

As suggested above, combining tf and idf, we obtain tf-idf.

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D) \tag{3.26}$$

with:

1. $|D|$: cardinality of D (i.e. the total number of documents in the corpus)
2. $|\{d \in D : t \in d\}|$: number of documents where the term t appears (i.e., $\mathrm{tf}(t,d) \neq 0$)[15].

Note that mathematically, the base of the log function does not matter. It constitutes a constant multiplicative factor towards the overall result.

A high value, or weight, in tf–idf is the result of a high term frequency of a term in a given document and a low document frequency of the term in the whole collection of documents. Tf-Idf will tend to filter out common terms since the idf value will tend to be close to 0 for terms found in nearly all documents.

This is usually a desirable behavior that complements stop word filtering, by zeroing out the weights for stop words that are specific to the collection without the need for identifying those words ahead of time.

Since the ratio inside the idf's log function is always greater than or equal to 1, the value of idf (and tf-idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf-idf closer to 0. More concretely:

Highest score a term appears frequently in a document and is found in a small proportion of the document collection, say 10% or lower.

Lower a term which has a lower frequency in a document, or occurs in many documents (around half for example)

Lowest the term is present in nearly all the documents of the collection.

Improving Tf-Idf

Tf-Idf has become a standard term weighting scheme for processing text documents to make them suitable for using in information retrieval or text mining applications. However, it has

[15]If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the formula to $1 + |\{d \in D : t \in d\}|$

issues that have led to considerable research in applying tweaks of various kinds to the basic algorithm [Liu and Loh, 2007, Paik, 2013, Soucy and Mineau, 2005, Zhu and Xiao, 2011].

In Soucy and Mineau [2005], the authors explain that this weighting method does not leverage the information implicitly contained in the categorization task to represent documents. This is a low-hanging fruit in a supervised classification problem, because by definition, we start with a set of documents for which we know the class. It seems wasteful to ignore the information of the class of the document when computing the term weights. can we do better?

Ko [2012] is a survey of methods that add class information in term weighting for text classification. BNS is one example of such a scheme and we describe it in detail in the next section (3.6.2).

Bi-Normal Separation (BNS)

First introduced as a feature selection algorithm in Forman [2002, 2003], BNS was later adapted as a term weighting alternative to Tf-Idf by Forman in Forman [2008].

Bi-Normal Separation (BNS):

$$F^{-1}(tpr) - F^{-1}(tnr) \tag{3.27}$$

Where:

pos $|D_{positive}|$

neg $|D_{negative}|$

tp (*true positive*) $f(t,d) : d \in D_{positive}$

tn (*true negative*) $f(t,d) : d \in D_{negative}$

tpr (*true positive rate*) tp/pos

tnr (true negative rate) tn/neg

F^{-1} the cumulative inverse normal distribution

Forman further suggest to constrain the values for tpr and tnr in the range [0.0005, 0.9995] as the inverse normal distribution tends to infinity at 0 and 1.

The reason why BNS works is not immediately intuitive, but can be understood by breaking it down and examining each step in the context of a classification task.

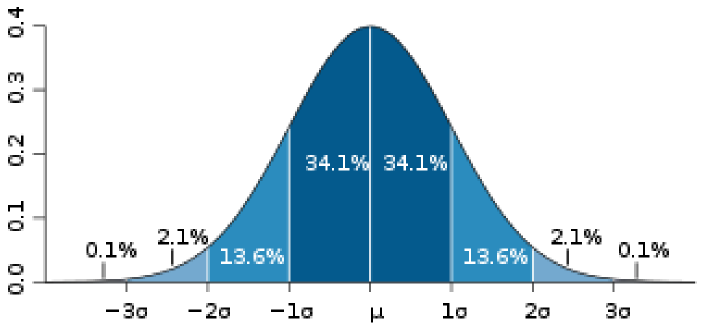


Figure 3.19: The normal distribution (made with R)

First, the normal distribution is shown at Figure 3.19 and its inverse at Figure 3.20[16].

The BNS algorithm seeks to increase the weight of documents that are common in one class, and rare in the other, preferentially. Indeed, these words have a lot of predictive value, as opposed to words that are common to all documents, or rare in both the positive and negative class. The cumulative inverse normal distribution (figure 3.20) is a model well suited for this purpose, as the curve gradually rises in the range $[0.2, 0.8]$, but beyond those values, the curve goes to $\pm\infty$ quickly.

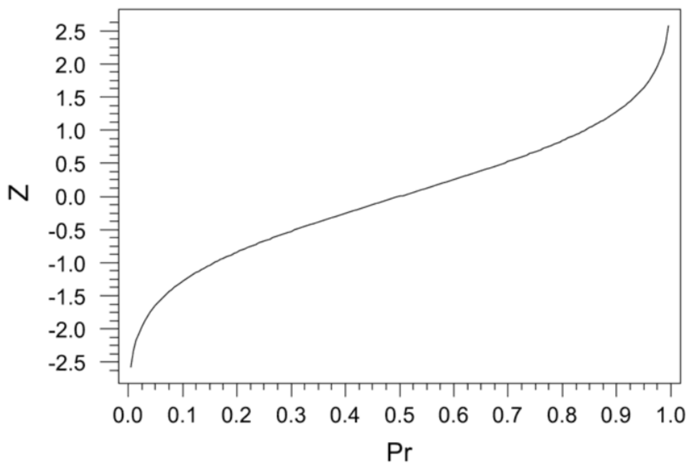


Figure 3.20: The cumulative inverse normal distribution (source: Pat Bartlein)

Going back to the formula, what is feeding into F^{-1} is the true positive rate and the true negative rate. This is the rate at which a term is present in the positive and negative document sets, respectively.

[16]`http://geog.uoregon.edu/bartlein/old_courses/geog314f00/images/icdf.gif` (Feb 2014)

Since the BNS formula is the absolute value of the subtraction of those two values, what we get is the following: The BNS weight will be largest when a term is present at a high rate in one class, an not in the other. The weight will be smallest either when a term is present in a comparable rate in both classes.

3.6.3 Machine Learning Algorithms for Text Classification

SVM

Joachim Thorsten was among the first scientists to apply the new SVM algorithm to text categorization in Joachims [1998] and Joachims [1999].

According to Thorsten, there are several theoretical arguments that can explain why SVM performs very well on text categorization problems:

High dimensional input space The dimensions of the input space for text categorization are usually the size of the vocabulary, which can reach into the tens of thousands. SVM is very robust and handles this well.

Few irrelevant features One way to deal with large number of features is to aggressively perform feature selection. But in the case of text, there are very few irrelevant features.

Document vectors are sparse While the input space is very large, each document only has a relatively small subset of words, making each vector sparse (with most features set to 0). SVM works very well with sparse instances.

Most text categorization problems are linearly separable Documents are typically linearly separable. SVM works by finding a linear separator in some space (linear, poly, RBF, ...), which is again very suitable to the particular problem of text.

Documents themselves are however map to sparse vector representation, since a single document will only use a very small subset of the total vocabulary of the language. Lastly, SVM in Kernel space are uniquely effective in linearly separating text categories. On that front, soft-margin SVM will also show robustness to misclassified documents and still generalize well [Shanahan and Roma, 2003].

Joachim has recently published a book on the topic of text classification with SVM (Joachims [2012]).

Gradient Descent and Stochastic Gradient Descent

Gradient descent is an optimization technique that minimizes the gradient iteratively.

$$Repeat : w \longleftarrow w - \gamma \left(\lambda w + \frac{1}{n} \sum_{i=1}^{n} \frac{\delta Q}{\delta w}(x_i, y_i, w) \right) \tag{3.28}$$

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

SGD has been successfully applied to large-scale and sparse machine learning problems often encountered in text classification and natural language processing ([Bottou, 2010, 1998]). It is also used heavily to help solve deep belief networks in a recent NIPS paper by Dean (of MapReduce fame) in Dean et al. [2012].

3.7 Scaling up Text Classification

3.7.1 At Large Scale, Data >Algorithm?

Having more data leads to better models. While this may sound relatively uncontroversial today, it was very much so when first proposed at the very beginning of the 2000's by Banko and Brill [Banko and Brill, 2001].

This article was among the first to suggest that research on ever more sophisticated algorithms was maybe misguided. More data combined with simple algorithms could lead to better performing models (Figure 3.21[17]). Their paper shows that, for the problem of natural language disambiguation, changing from one state of the art algorithm to another have little impact on performance. But just by adding (a lot) more examples (words) to the training set monotonically increases the accuracy of the model.

More research supported and formalized these findings in the following years. Bottou and Lecun in Bottou and LeCun [2004] and Bottou and LeCun [2005] argue for the superior performance of online learning on massive datasets to be asymptiomatically superior to any

[17]`http://technocalifornia.blogspot.jp/2012/07/more-data-or-better-models.html` (Feb 2014)

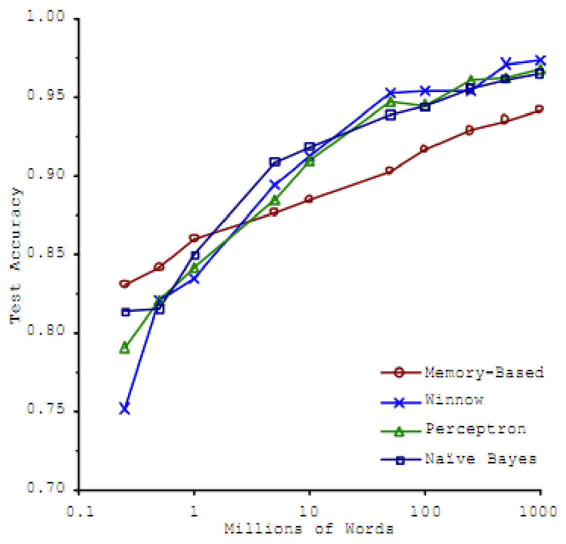


Figure 3.21: All models perform the same on the same data, but more data improves performance (source: Xavier Amatriain)

batch learning algorithm. They also show that a well designed online algorithm will converge to the minimum expected cost just as fast as a batch algorithm.

Norvig et al. in their influential article The Unreasonable Effectiveness of Data (Halevy et al. [2009]), hammer the point yet again, arguing for the superiority of more data over better models as a means for getting better performance from machine learning applications.

The quotes are certainly provocative, but we can't be too quick to jump on the data bandwagon either. Indeed, the famous 1M$ Netflix prize led to one top team publishing a paper (Pilászy and Tikk [2009]) that showed just the opposite! In their case, a highly optimized model was very beneficial, and adding tons of additional data had not helped at all.

Chris Anderson, editor of Wired, a famous technology magazine, heated up the debate with his article Anderson [2008]. The article explains several examples of how the abundance of data helps people and companies take decision without even having to understand the meaning of the data itself. Norvig even had to write up a rebuttal[18]. The data deluge does not make the scientific method obsolete, quite the opposite, it is now more important than ever! The scientific method is the only protection a user of data has to know the conclusions that are reached aren't fanciful in the first place. Case in point, Andrew Gelman's blog post gives a important reality check on the blind use of a single number, the (in)famous p-value, as a

[18]`http://norvig.com/fact-check.html` (Feb 2014)

passport to claim anything as scientific truth [19].

That said, leading technology companies like Google really do have huge datasets to work with, and really do wring out class leading performance out of their systems. The impact of Big Data on machine learning is only beginning.

Langford's Vowpal Wabbit Agarwal et al. [2011], Ghoting et al. [2011b], GraphLab Low et al. [2012], HaLoop Iterative MapReduce Bu et al. [2010], Langford edited book on large scale learning Bekkerman et al. [2011].

3.7.2 Online, Linear Algorithms are Very Scalable

At large scale, computational complexity of learning algorithms is a limiting factor when faced with a very large amount of training data.

One interesting empirical example compares LIBLINEAR to LIBSVM[20]. For a certain dataset of 20,242 instances and 47,236 features, the cross-validation time is significantly with the linear classifier taking only 3s compared with the kernel SVM taking over 345s. The cross validation accuracy is comparable at 97% and 96%.

In other words, for some large datasets (number of instances and features) such as text documents, linear and nonlinear mappings give very similar performances but with the benefit that without using kernels, one can quickly train a much larger set via a linear classifier.

The state of the art in supervised machine learning remains SVM. But its high quality predictions come at the price of computational complexity.

The two most popular ways to solve an SVM learning problem is using the kernel trick and the Sequential Minimal Optimization (SMO) by Platt Platt [1998], with an accepted worse case complexity of $O(n^3)$. At the scale of 10,000 examples, SVM becomes completely impractical.

Despite considerable work on adapting SVM to work on larger datasets, ([Kashima et al., 2009]) no real answers beyond approximation methods (e.g. Nystrom) have been found.

There has been much work on trying to find effective parallel implementations of SVM, such as PSVM [Chang et al., 2007] or the Cascade SVM [Graf et al., 2004]. But these algorithms still require running over the dataset multiple times until convergence, which can really add up on very large-scale datasets.

[19] `http://tinyurl.com/k5nk4pn` (Feb 2014)
[20] `http://www.csie.ntu.edu.tw/~cjlin/liblinear/` (Feb 2014)

The only way to feasibly learn on datasets of millions, or billions of examples is to use much more simple algorithms with a complexity of O(n) or less. These are called linear algorithms. Even better is to train the algorithm one example at a time continuously, which is called online learning.

Online algorithms for large scale learning is a very active topic of research, we refer to the excellent works by luminaries such as Langford and Smola in Hsu et al. [2011] which really focuses on algorithms with some mathematical formality. Also see *Scaling Up Machine Learning* [Bekkerman et al., 2011].

Ericson in Ericson and Pallickara [2013] gives a detailed performance tour of large scale clustering and classification using Mahout on a Hadoop back-end.

3.7.3 Large-scale Machine Learning Algorithms

Feature Hashing a.k.a the Hashing Trick

The common approach to use machine learning algorithms with text is to construct, at learning time or prior to that, a dictionary representation of the vocabulary of the training set, and use that to map words to indexes. This is the vector space model that we described in the previous section (subsection 3.6.1). This is exactly the approach used with Tf-Idf and especially BNS.

One problem with this process is that such dictionaries take up a large amount of storage space, and grow in size as the training set grows. Another is that it is necessary to build the dictionary itself. This extra processing can become especially problematic at large scale as the time required to build the dictionary can become significant.

Instead of maintaining a dictionary, a feature vectorizer that uses the hashing trick can build a vector of a pre-defined length by applying a hash function h to the features (e.g., words) in the items under consideration, then using the hash values directly as feature indexes and updating the resulting vector at those indexes.

This technique is shown to be very effective on large scale problems in Weinberger et al. [2009a]. It has been implemented in several machine learning libraries such as Mahout, Scikit-learn, Gensim and Langford's Vowpal Wabbit.

3.8 Conclusion

Machine learning has proven itself to be a remarkably useful and powerful tool to add intelligence to computer systems. In supervised learning, the machine learning algorithm will automatically build a model from a training set of labeled examples. The model can then be used to make a label predictions on new, previously unseen data.

Natural language text can be suitably transformed to be used as input in these algorithms, using algorithms such as Tf-Idf and BNS. This has opened up all written documents to applied machine learning techniques like supervised learning.

The latest research on large scale learning has further opened up exciting possibilities. Not only is it now becoming possible to run machine learning algorithms on huge text collections, but new previously impossible tasks are being put into production seemingly as soon as they are invented.

In the next chapter, we will show such an application, focused on solving a business need present at Fujitsu Canada's Quebec City office. Our system uses state of the art machine learning algorithms like SVM and an advanced parallel version of Logistic Regression to predict whether public tenders will be relevant or not for a Fujitsu manager.

We will also demonstrate how our machine learning solution might scales to fit the needs of Fujitsu, one of the world's largest technology companies and third largest IT consulting company in the world. We successfully build a running distributed system based on a Apache Hadoop and the Mahout machine learning library and make use of all the tools and techniques we have presented in the last two chapters.

Chapter 4

Machine Learning, Hadoop and MapReduce

4.1 What Can Hadoop Do for Machine Learning

Currently, academic machine learning research teams with high computational needs will take advantage of supercomputing facilities. In the case of large businesses, the needs for machine learning applications are currently centered around clustering and recommender engines.

A Hadoop cluster brings with it several advantages over supercomputers and very large enterprise class systems. First and foremost, Hadoop has a crushing cost advantage as a large scale, distributed, batch processing infrastructure.

The reasons for this cost advantage starts with Hadoop having been designed from its very inception to run on cheap commodity PC hardware as opposed to the costly hardware requirements of enterprise and HPC (high performance computing) hardware. One global energy giant, Cheveron, found a factor of ten cost advantage from using Hadoop, according to a Cloudera paper[1].

Running on lower-power, cheaper hardware also leads to better energy use per dollar. Google, who is the leader for using commodity PC hardware to run its huge computing clusters, is known to have some of the most energy efficient datacenters in the world (see `http://www.google.ca/about/datacenters/`). That said, the verdict on Hadoop itself leading to energy savings are somewhat mixed and subject of several recent studies ([Feller et al., 2013, Chen et al., 2009, Leverich and Kozyrakis, 2010]).

[1]`http://blog.cloudera.com/blog/2012/01/seismic-data-science-hadoop-use-case/`

Of course, the top performance of a world-class supercomputer will always be much faster than an equivalent (number of cores, memory size, cost, etc.) map-reduce implementation. But this would be missing the point. Hadoop brings with it a growing number of tools and libraries that make the development of truly global scale machine learning systems accessible in a way it has never been accessible before, as much in terms of cost as in terms of software engineering expertise.

The benefits of large-scale machine learning and Hadoop will be the subject of this chapter.

4.2 MapReduce and Machine Learning in the Scientific Literature

The first paper that explored how machine learning algorithms could work with map-reduce was Chu et al. [2006]. The paper shows how well-known machine learning algorithms such as logistic regression and k-means can be converted to run on multi-core systems using the map-reduce paradigm[2]. Their experimental results showed linear speedup, thus confirming that map-reduce has a lot of potential for applied machine learning.

One of the most computationally demanding tasks of applied machine learning is parameter tuning, which involves training and testing new models over a potentially very large parameter space in order to identify the best performing parameters. In this case, given access to a Hadoop cluster, a scientist could distribute the parameter search by wrapping them in map tasks, and then use the reduce phase to return the best parameters. this approach was first described in Ganjisaffar et al. [2011]. We show in Figure 4.1 a simplified illustration of the process.

Another aspect to running machine learning algorithms on a Hadoop cluster involves finding ingenious, new map-reduce implementations of known algorithms as found in several articles such as Ghoting et al. [2011a], Panda et al. [2009], Li and Schuurmans [2012] and Dede et al. [2011].

Some of these cutting-edge implementations have since been added to the Apache Mahout library (described in section 4.3), which offers users a (relatively) easy to use machine learning library that can take advantage of a Hadoop cluster (relatively) automatically.

Machine learning algorithms are often iterative and need a shared memory for optimal performance. Hadoop is a slow, batch processing system. Newer research projects have emerged

[2]Notably, Andrew Ng (of Stanford and Coursera.org fame) was a contributor to this important paper

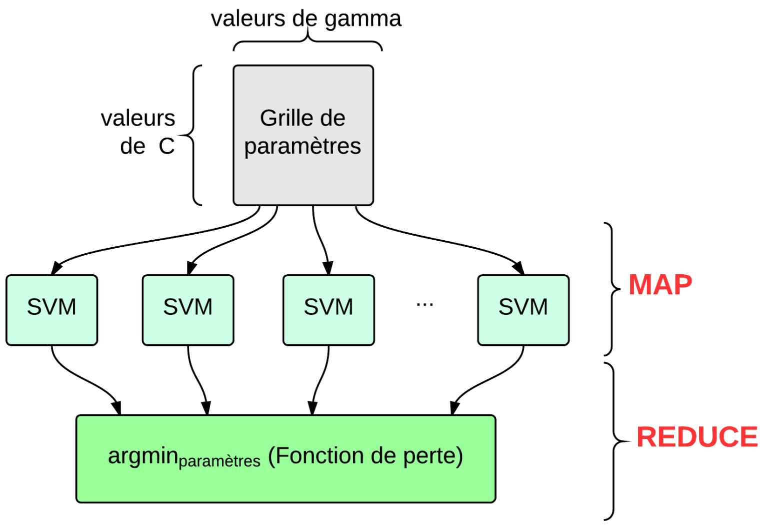


Figure 4.1: Distributing the parameter search using MapReduce

that address these problems with modified versions of Hadoop that such as Haloop ([Bu et al., 2010]) and Twister ([Ekanayake et al., 2010] and [Gunarathne et al., 2013]) and others ([Zhang and Chen, 2013, Tudoran et al., 2012]). We should note that none of these modified Hadoop systems have gained any popularity in practice.

Finally, the first few months of 2014, Cloudera announced Oryx, an open source alternative to Mahout. Oryx is intended to help Hadoop users build machine learning models and then deploy them so they can be queried and serve results in real time. Some examples of applications are spam filters and recommendation engines. The new machine learning library will eventually support models that can update themselves in real time.

Our project's large-scale machine learning core makes use of the Apache Mahout, with all the pre-processing and data preparation work performed using Apache Pig. These will be explained in greater detail in the next section.

4.3 Mahout the Elephant Driver

4.3.1 What is Mahout?

Mahout is a scalable Machine Learning library, focusing on recommender, clustering and classification algorithms. Scale in this case refers to learning from the largest possible datasets, from gigabytes to petabytes (and beyond?).

Mahout is an offshoot of the Apache Lucene project. Lucene provides advanced implementations of search, text mining and IR techniques, which are adjacent to machine learning techniques such as clustering and classification. In 2008, The work by Lucene committers focusing more on machine learning was spun off into their own project: Mahout. The recommender part of Mahout came from an existing open source recommender library called Taste.

To scale to the largest datasets, the machine learning algorithms in Mahout have Hadoop MapReduce implementations behind the scenes. This explains why the name Mahout was chosen. In the Hindi language, *mahout* means an elephant driver.

This is a library that is meant to be used in production, and there is a strong focus on performance at the largest scales in the project.

The development community around Mahout is very active, with a major new version released roughly every six months. Also, just like Hadoop, Mahout is backed by the liberal Apache license.

Many of the leaders of the technology world are Mahout users, such as Last.fm, LinkedIn and Intel ([Anil et al., 2010]).

4.3.2 The Goal of Apache Mahout

Quoting from Apache Mahout's web site[3], Mahout's goal is to build fast, scalable machine learning libraries. With scalable we mean:

- Scalable to reasonably large data sets. The core algorithms for clustering, classification and batch based collaborative filtering are implemented on top of Apache Hadoop using the map/reduce paradigm. The core libraries are highly optimized to allow for good performance also for non-distributed algorithms.

- Mahout is distributed under a commercially friendly Apache Software license. Mahout scales to your business case.

- Open-source development to foster a vibrant and active community. New releases every year (or more!) with substantial improvements.

[3]`https://cwiki.apache.org/confluence/display/MAHOUT/Overview`

4.3.3 Machine Learning With Apache Mahout

There are three main types of algorithms in the Apache Mahout library (as of version 0.9): Collaborative filtering, or recommender engines, clustering and classification. All are designed to scale up to collections with millions to billions of data points comprised of potentially millions of features. To make the processing faster, just add nodes to the cluster until it runs at the desired speed.

We detail each type of algorithm in term in the next several sections.

4.3.4 Recommender Engine

The most visible machine learning in use today are recommender engines. Think of Facebook and LinkedIn "People you might know". Amazon, another great example, knows so much about its customers that Amazon engineers have to tune their recommenders down to "not be creepy" and spook customers. For example, a customer might get a recommendation for a swimsuit just as they are thinking of a tropical getaway, or getting recommended diapers before knowing a spouse is pregnant.

Amazon and Netflix are examples of companies that live and die by the quality of their recommendations, proving that recommender engines are of real commercial value.

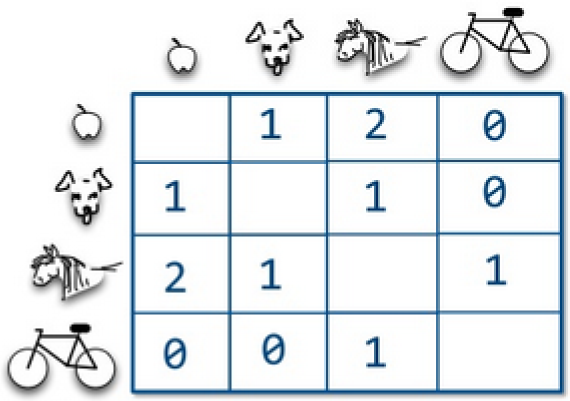
In mahout, the focus is on both user-based and item-based collaborative filtering [Goldberg et al., 1992]. User-based recommendation is where the user is recommend items (think movies, music or books) based on what similar users might have purchased or liked. Item based recommendations returns a list of items that are similar to items that the user likes, or has interacted with.

To implement collaborative filtering at scale, Mahout relies on a co-occurrence matrix as that shown in Figure 4.2[4]).

The co-occurrence matrix implementation relies on vectors and matrices and lots of linear algebra, which all have well understood techniques for distributed implementations. Indeed, all portions of the process require only a subset of all the data, which maps naturally to the MapReduce paradigm.

For those interested, we can suggest two surveys that can help get up to speed on the topic in Su and Khoshgoftaar [2009], Adomavicius and Tuzhilin [2005].

[4]source:Ted Dunning (`http://www.slideshare.net/tdunning`)



	🍎	🐶	🐴	🚲
🍎		1	2	0
🐶	1		1	0
🐴	2	1		1
🚲	0	0	1	

Figure 4.2: Item-Item co-occurrence matrix common to recommender engines (source: Ted Dunning for MapR)

4.3.5 Clustering

Clustering identifies structure, even hierarchy, among large collections of things that might otherwise be difficult or impossible to make sense of.

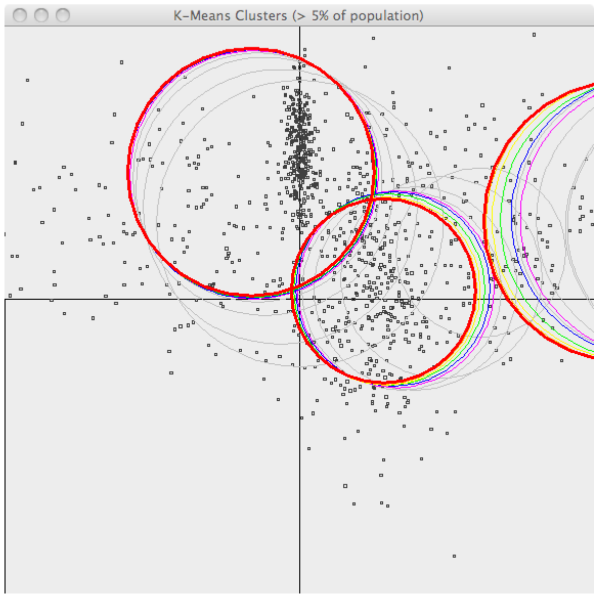


Figure 4.3: Kmeans clustering in Mahout (source: Apache Mahout)

Clustering is an example of non-supervised machine learning. It is akin to the task of sorting a single pile of documents into neat related piles, without any guidance. Some practical examples of clustering at work is Google News, which clusters news into topics by logical story, rather than just an unorganized list.

The premier example of clustering algorithm is K-means clustering (see Figure 4.3), and it is duly included in Mahout. The most recent version of Mahout, released in the summer of 2013, includes an streaming k-means implementation that allows fast, online clustering. It based in part of K-means++ by Arthur and Vassilvitskii in Arthur and Vassilvitskii [2007], and for streaming, on some estimation, as in Braverman et al. [2011].

There are other clustering algorithms available in Mahout such as Fuzzy k-means and LDA. Experimentation as always is required to know which will be best for a given situation, although there are some useful general guidelines in the Mahout in Action book (Anil et al. [2010]). Especially, the information on how to run the algorithms on a Hadoop cluster are well made and informative, as are the real-world examples clustering users with Tweet data or music artists using Last.fm data.

4.3.6 (Text) Classification

SVM is still, after almost 20 years, the leading edge classification algorithm (see Cortes and Vapnik [1995]). It is a ruthless benchmark that all new algorithms must compare themselves. But with the leading implementations have a complexity between $O(n^2)$ and $O(n^3)$ (see Bottou and Lin [2007] and Simon and List [2009]). In practice, past 10,000 examples, the training time becomes prohibitive. Up to 100,000 examples, linear algorithms, which train in time proportional to the number of examples (and number of features) like logistic regression will be efficient and accurate. But as the number of examples continues to scale, say to between 1-10M, the power of a distributed cluster will become necessary. This is where Mahout truly shines.

As Mahout originates from the Lucene project, it's not surprising that there is a lot of support for text classification built in, as well as good integration with Lucene. It's possible to vectorize and classify (or cluster) an arbitrary document collection using only the command line, running entirely on a Hadoop cluster in parallel. This is an impressively useful and powerful library.

As data scales to such huge sizes, implementation issues may cause alternatives with theoretically linear training complexity to slow down. A doubling in data size might take 5 times longer, past a certain dataset size threshold. Mahout, on the other hand, still scales linearly at huge sizes, limited only by the cluster processing the data. This allows the practitionner to trade off the number of computers against the time required to solve a problem.

Classification algorithms are at the heart of what we call predictive analytics. The goal of

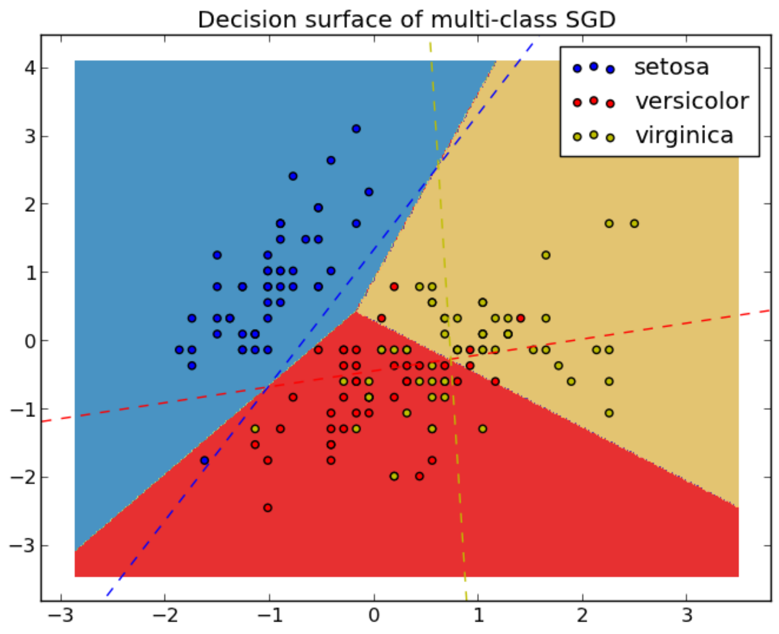


Figure 4.4: The Stochastic Gradient Descent algorithm on the Iris dataset (source: Scikit-learn

predictive analytics is to predict some output based on new, previously unseen data, making decisions that mimic human judgment. Where clustering allows the practitioner to get information from data without any previous information, classification is a supervised method, and so requires a training set to build a model that will be used to build the model that will make the predictions. As strongly argued by Norvig et al. in Halevy et al. [2009], increased training data typically leads to better models. Mahout makes processing very large datasets not only possible, but practical. This is the component used for our project.

The main algorithms for classification in Mahout are Stochastic Logistic Regression (see Figure 4.4), Random Forest or Online Passive Aggressive algorithm. These are state of the art parallel algorithms (see Bottou [2010], Mitchell et al. [2011] and Crammer et al. [2006].

Mahout includes a lot of built-in support for classification of text documents, with direct support for Lucene, TF-IDF term weighting and leading edge techniques such as feature hashing.

While we do not use it for our project, Mahout implements the idea of feature hashing, also known as the "Hashing Trick". It allows mapping features to vector space efficiently without needing a dictionary. It also has the advantage to let the user decide the vector size *a priori* by using a hashing function to map features (such as words in a document, for example) to an index in vector space (see Weinberger et al. [2009b]).

4.4 Machine Learning with Pig

Twitter's experience writeup by Jimmy Lin and Alex Kolcz in Lin and Kolcz [2012] shows how a top tier web-scale company like Twitter can reap incredible gains from using a high level approach to Hadoop. Twitter is running a system with huge demands of streaming bandwidth. As of August 2013, Twitter processes an average of 500M tweets a day or over 5K tweets per second [5].

Twitter uses Hadoop heavily as the core of their analytics infrastructure. According to the Hadoop Hardware @Twitter presentation by two Twitter engineers at Hadoop Summit in July 2013, Twitter is now running several clusters in the thousands of node range. Twitter's analytics is done primarily with Pig and a custom Pig job scheduler called Oink (thus continuing the trend of whimsical names common to everything Hadoop).

Lin and Kolcz show how, using Pig's built-in extensibility of UDF functions and Storage functions, it is possible to build en entire Machine Learning workflow. Core libraries for Machine Learning are "fairly standard and should come as no surprise to the reader".

The primary challenge that had to be overcome was the mismatch between the models used by Pig and those of the Machine Learning libraries. The solution was a very creative engineering solution. They used PigStorage to read data and save it to a model and then passed it through to the learning algorithm. As a bonus, the storage function can advantage of Pig's built in parallelization feature to split the task between a user-configurable number of reducers (via the default.parallel parameter). This naturally leads to the ability to train many models on independent partitions of the data and ensemble methods (see Figure 4.5)[6], and can scale to arbitrarily large scales.

The paper further shows how, using only 'regular' Pig script, it is possible to shuffle and split a labeled set of examples into a train and test set. They also developed several wrappers on Pig to allow using classifiers directly in Pig. The key contribution here is that a machine learning pipeline, from data suitable for classification on to model training and use of learned models can be done in Pig. The advantages of using Pig are significant in terms of scaling seamlessly from the individual developer's laptop to a huge Hadoop cluster with thousands of nodes, simplified workflow and increased developer productivity. When working in the

[5]`https://blog.twitter.com/2013/new-tweets-per-second-record-and-how`

[6]Figure 1 from Machine Learning at Twitter

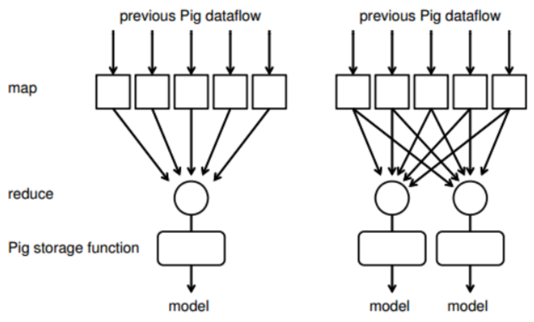


Figure 4.5: Learning in parallel naturally leads to ensemble methods (source: Jeremy Lin)

high speed arena of a hyper-growth company such as Twitter, these are tangible benefits that justify this kind of R&D work.

We use quite a few of those techniques in our own project, they are explained in the relevant section (section 7.3).

4.5 Conclusion

This chapter presented how it is possible to use state of the art machine learning algorithms on large-scale data living on the distributed storage of a Hadoop cluster.

The machine learning services offered by Mahout are easy to use, powerful and very fast. The main difficulty with regards to using Mahout in a real project lies in the data pre-processing required to transform the raw input data into a format that Mahout can use.

In the case of text classification, this pre-processing work involves a non-trivial sequence of steps. Mahout does offer a default text classification pipeline but it is limited.

This problem was solved at Twitter by using Pig as the glue between the data and the algorithms. Their solution was an inspiration for our own project.

The next chapters will present our project from the business problem to its final implementation on a working Hadoop cluster, starting by a description of our experimental methodology. This will be followed in turn by the experimental log for the development of our prototype system and then the final large-scale system.

Chapter 5

Methodology - Process

While no specific process was explicitly chosen at the start of the project, we did follow a logical and well organized process. Later review of the field has shown that we had followed remarkably closely the CRISP-DM process ([Shearer, 2000, Wirth, 2000]).

CRISP-DM stands for Cross Industry Standard Process for Data Mining. It is a data mining process model that describes commonly used approaches that expert data miners use to tackle problems.

It was developed by a core industry group including SPSS and Teradata. We have therefore chosen to present our experimental process using the CRISP-DM process as an explanatory framework.

CRISP-DM breaks the process of data mining into six major phases, as illustrated in Figure 5.1. The key point of this process is its iterative nature, very much suited with experimental and R&D projects and contrary to the traditional waterfall software project. Indeed, there is much moving back and forth between different phases. The arrows in the process diagram indicate the most important and frequent dependencies between phases. The outer circle in the diagram symbolizes the cyclic nature of data mining itself.

As described in Wirth [2000], the steps of CRISP-DM are as follows:

Business Understanding In the first phase, the focus is on getting to grips with the business problem itself in order to translate the original business problem into a problem that can be solved using data mining techniques. This phase should produce a preliminary plan.

Data Understanding First task of this phase is to collect some initial data. It is usually necessary to do some exploratory analysis and exploration to gain familiarity with

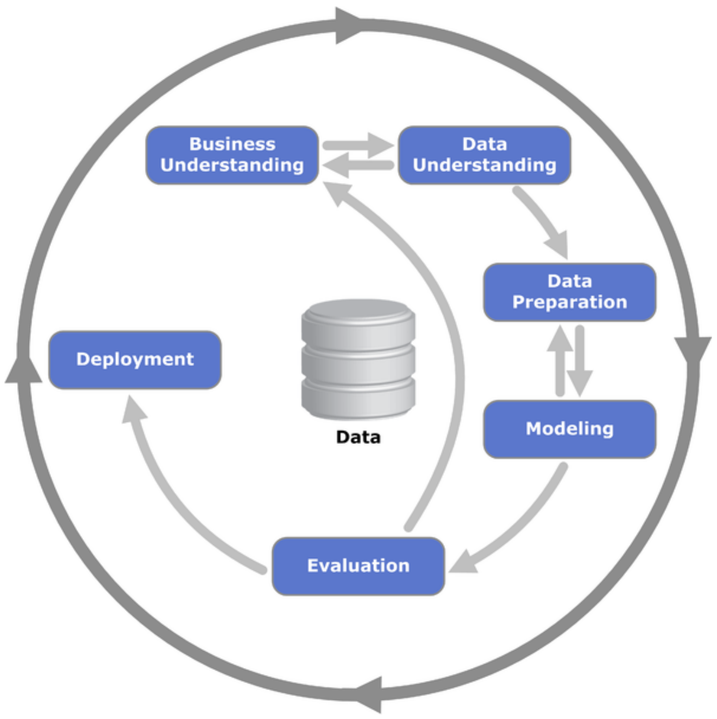


Figure 5.1: Process diagram showing the relationship between the different phases of CRISP-DM (source: Wikipedia)

the new data and identify data quality issues, or identify additional data that might be required to solve the problem.

Data Preparation This phase is for performing all the processing necessary to construct the final dataset that will be used to learn models. This step is likely going to be done multiple times as successive model learning efforts yield greater insight into the data and reveal previously hidden problems with the current dataset. This is the step where data cleaning and transformation, also known as data wrangling, is performed.

Modeling The modeling phase is where the dataset produced in the previous phase is used to learn a new model. This is also where the model is calibrated to find optimal values for parameters. As several techniques might lead to solving the problem, it is often necessary to take a step back to the data preparation phase several times before a satisfactory model is found.

Evaluation As models are built, they must be evaluated to find if the performance is high enough to consider the problem solved. Thorough evaluation of a model before production is essential to ensure that the model indeed achieves the business objectives set at the beginning of the project. The evaluation phase should yield enough information to take the decision to take the model into production or go back and build a new, better performing model. It is especially important to make sure all business issues are sufficiently considered in this phase, before production use.

Deployment Simply creating a high performance model in a test or development environment with well behaved data is not the end of a project. The model(s) selected in the evaluation phase must now be put into actual production. This might be as simple as generating a report and presenting it to customers, or it might mean the implementation of the model in a production environment with additional requirements specific to its production environment. Even in such cases, it is important for the data analyst to communicate the requirements are assumptions of the model to ensure it performs optimally once deployed.

Chapter 6

Experimental Log: Machine Learning Problem

6.1 Business Understanding: The Problem Statement

This project finds its origin in a business problem relevant to the Fujitsu Quebec city office, as described in the introduction (see section 1.4). In short, we want to help automate the process of discovering potential business opportunities by identifying public tenders that are relevant to a Fujitsu manager's area of expertise.

The process started with several meetings with Guy Michaud, who acted as a business stakeholder for the project. Guy started by contextualizing the general problem and then demonstrated a previous internal R&D project called Sieve, which had similar aims to our project, but was not ultimately successful.

The first step was to come to terms with the business problem itself. Before any kind of programming effort, it is essential to understand the business problem, the related data and define metrics that can will define when the problem is solved satisfactorily.

Thus, we will cover the following topics in this section on business understanding:

- Input data: the sieve database, which contains the data extracted from a web site listing Canadian public tenders.
- Classification: supervised machine learning approach to identify relevant tenders.
- Metrics: what performance metrics should be used to determine if the problem was solved successfully or not.

- Change of vision: After some initial work, the project as a classification problem evolved to a filtering problem.

The Data

The original prototype was doing a bit of word cross-referencing between Fujitsu employee resumes and the content of the *notice field* taken from electronic public tenders of MERX.com's R&D category.

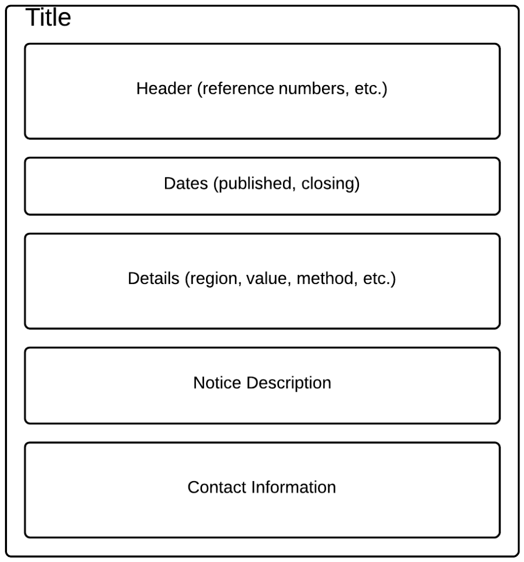

Title
Header (reference numbers, etc.)
Dates (published, closing)
Details (region, value, method, etc.)
Notice Description
Contact Information

Figure 6.1: Layout of tenders on merx.com

On MERX.com, each page corresponds to one public tender. Web pages have the layout shown in Figure 6.1. The sieve application included a web crawling function to extract the content of those pages and save the data to a MySQL database.

As shown in Figure 6.1, each public tender is composed of several sections of related information. The decision to use the notice field was taken early on in the project because that field was usually sufficient for Fujitsu managers to decide if a tender was a good business opportunity or not.

The notices are a short text description of the content of the public tender. The next section on data understanding covers this text in detail in section 6.2.

Classification

The original prototype was cross-referencing words between Fujitsu employee resumes and the content of the notice field. The more matches, the higher the relevance. This approach failed as the lists displayed to users were essentially random.

Given the data, the initial hypothesis was that classification, a supervised machine learning approach, would successfully tackle the problem.

The hypothesis emerges naturally given that some of the public tenders in the Sieve dataset was already labeled by a business user as *relevant* or *not relevant*. These labeled example could then be used to train a classifier that would then be able to classify any tender.

Several assumptions are made here:

- The training set we have is a representative and i.i.d. sampling of all possible tenders that could be posted on merx.com's R&D section. This is certainly not true, our dataset is a minuscule fraction of this total.

- The notice field contains enough information that relevant and not relevant public tenders can be separated in two distinct classes.

We then decided to use the contents of the notice description as a proxy for the public tenders. This text, suitably transformed, could be used as the input data for the classification algorithm.

There were many discussions about the obvious benefits of additional classifiers based on other features of public tenders. Ensemble method that would combine many different classifiers were considered, but not implemented given the time constraints of the project.

Performance Evaluation
Next, was the whole question of performance evaluation, a crucial point. The metrics we choose will then be used to optimize the system and determine objectively whether the system is performing at an acceptable level or not.

The Fujitsu manager has two concerns we must satisfy: be able to go through a reasonably short list of potentially relevant public tenders, and more importantly, be able to trust that all relevant public tenders will be in the list. In other words, it is several times worse to make a mistake on relevant tenders than irrelevant ones.

Not including a relevant tender in the final list means that Fujitsu will miss out on revenue from a service contract it might otherwise have won. In other words, missing notices directly translate into lost revenue for Fujitsu.

The choice of performance metrics should reflect this priority, and so we decided to pay most attention to the classic information retrieval metric of recall rather than precision. More precisely, we focus on the recall value for the positive (i.e. relevant) class of tenders[1]

Based on feedback from the business user, we settled on a goal of 95% recall on the relevant class as our most critical metric (see section 3.5.3). As long as the prototype can reach the target recall value on the relevant class, the user will have confidence that nearly all relevant tenders are indeed included in the list of results.

Of course, no machine learning system (or human for that matter) ever reaches 100% on a non-trivial task. However tradeoffs can be made by tuning the classification algorithm. In our case, our business user is willing to accept being shown a larger list in exchange for a stronger guarantee that almost all relevant notices will be shown and not be filtered out.

In practice, we will tune the classifier to be very conservative and favor predicting the positive class at the expense of making more mistakes on the negative one (i.e. predicting that a not-relevant tender is relevant).

Filtering
After much experimental work, we came to realize that the problem was similar to spam filtering. This new understanding does not change the technical choice of using a classifier. However, it directly led to the insight of dividing the problem into two separate steps. This was a key insight to help us achieve our best results.

Under this new understanding, the first step would now be to filter out as many irrelevant notices (*spam*) as possible. This is a much easier problem than the first, since the difference between relevant notices and the spam notices is much bigger. The second step is then to rank tenders by relevance, which can be obtained by using the class membership probability, for example.

The second task is now more difficult, given the small number of labeled examples, but it's an acceptable tradeoff given our business user's priorities. This means that the final list of potentially relevant tenders will be more poorly ranked, and include a little more irrelevant tenders, but with high confidence, nearly all relevant tenders will be listed.

As the first step filters easily over 95% of tenders, it's much less important that the most relevant tenders necessarily be ranked ahead of relatively less relevant ones. The largest benefit comes from the elimination of the time wasted on checking irrelevant tenders.

[1]High precision means that an algorithm returned substantially more relevant results than irrelevant, while high recall means that an algorithm returned most of the relevant results. See subsection 3.5.3.

This scheme could be further improved over time by including information gained from users interactions (i.e. clickstream analysis) to increase its performance in a real production system.

The the next section on data understanding explains the consequences of this new business understanding.

6.2 Data Understanding: The Sieve Corpus

The raw data came in the form of a MySQL database dump file. The data had been collected by the Sieve prototype over a period of a few months in 2011, covering the Research & Development category from Merx.com.

The R&D category lists public tenders from the Canadian federal government and the Canadian army as well as tenders from any of Canada's 10 provincial governments and governmental organizations. While the category's name might imply high-tech R&D tenders, this is not the case at all.

The merx.com R&D category includes a large number of miscategorized tenders. A rough estimate is that as many as half or more of the R&D category actually has nothing to do with research & development. Some examples are tenders for snow removal or army rations supply. Miscategorized tenders are an important source of noise that make the classification task much more difficult that it would be otherwise. Noisy data is common in real-world datasets, and the Sieve data is no exception.

Only the short description text of the notice field is used from each tender. The text of this field can vary considerably in length and is composed in a highly specialized and formal English.

In addition, not all tenders have the notice field. Some tenders, about 10%, only include a short message referring the reader to external documentation. This is equivalent to a null value. All tenders with such messages were culled from the dataset, as they hold no useable information.

6.2.1 Corpus Size

As shown in Table 6.1, the sieve user (Guy Michaud) had already labeled about 122 notices, 23 of which were labeled as "relevant" and the rest as "not-relevant". All other notices of the corpus were unlabeled. The training corpus is thus around 1% of the whole corpus. Unbalanced classes are typical for real-world data such as this.

Category	Count
Relevant	23
Not-relevant	99
Unlabeled	11456

Table 6.1: The original sieve dataset had very few labeled notices

As explained in chapter 3, for machine learning to be successful, the training set needs to be as close to representative of the entire set of possible data. This was clearly not the case with the initial data.

The solution was to ask our business user to label additional tenders to increase the number of labeled examples.

6.2.2 Language

The notice field for the tenders of the Sieve dataset are either written in English, in French, or are bilingual with the same description written in both languages. Such bilingual notices are another source of noise in the data. Classification using the bag of words approach will tend to work best if trained on a single language.

We analyzed the language of all notices of the dataset. As shown in Table 6.2, the labeled notices are almost all in English, with only a few written in French. The unlabeled notices are much more diverse, with a significant number of French and Bilingual notices.

Language	Relevant	Not-Relevant	Unlabeled
English	23	96	3926
French	0	3	1155
Bilingual	0	0	475

Table 6.2: The language of the notices of the original corpus are either in English, French or are bilingual

Languages other than English, such as French, use accents. Practical data cleaning must take this fact into consideration. Limiting the dataset to English does not make the accent problem go away unfortunately, as English-only notices still contain words with accents. For example, the names of people, places and/or organizations often have accents.

Dealing with accents is a constant source of grief that must be addressed early on in the data cleaning process. Our approach is detailed in the data preparation step (section 6.3).

6.2.3 Notice Length

Notices almost always have a title. The title is usually followed by a short paragraph which explains the work required by the tender.

Category	Mean Length	Median Length	Range (min,max)	Standard Deviation
Relevant	812	637	$[158, 3038]$	579.7
Not-relevant	989	611	$[29, 8466]$	1199.9
Unlabeled	512	259	$[3, 27289]$	1125.8

Table 6.3: The original sieve dataset notices are short but with extreme ranges in sizes

Looking at dozens of notices, one can immediately see that the length of notices, in terms of word count, varies tremendously. Some notices are very short and terse, others a few paragraphs long and some are detailed multi-page documents.

A simple analysis of word counts of notices confirms this intuition, as shown in Table 6.3. For all categories, the median length is far off the mean length, with a large standard deviation. The numbers for relevant and not-relevant are relatively similar in terms of length. This means word counts are probably not a feature that could help us discriminate between relevant and not relevant notices.

6.2.4 Corpus Vocabulary

The top 10 words for the original corpus are shown in Table 6.4. There is remarkably little useful information in this table. Especially between the categories of interest, relevant and not-relevant, there is hardly any difference.

The lists do eventually diverge. Notably, the relevant category has some words that hint to its true purpose with words like “technology”, “development”, “drdc”, “software”, and “technical” all appearing in the top 50.

Indeed, the relevant notices are relevant with regards to the Fujitsu IT consulting business, which mainly deals with software development. For example, “drdc” refers to “Defence Research and Development Canada”, an important target customer for Fujitsu’s Quebec city office, as it is a short 30 minutes drive away from the Valcartier military base.

The plots of Figure 6.3 and Figure 6.4 show the same information more graphically with the associated counts. The curve is a Zipf distribution typical for natural language text (see "Zipf's law: Modeling the distribution of terms" in Manning et al. [2008a]).

Zipf's Law "Zipf's law is an empirical law formulated using mathematical statistics. It refers to the fact that many types of data studied in the physical and social sciences can be approximated with a Zipfian distribution, one of a family of related discrete power law probability distributions. It is characterized by a few very high probability or frequency items followed by a long tail of much lower probability or frequency. Zipfian distributions are often found in language, such as a word frequency graph, a classical example (see Figure 6.2)."[2]

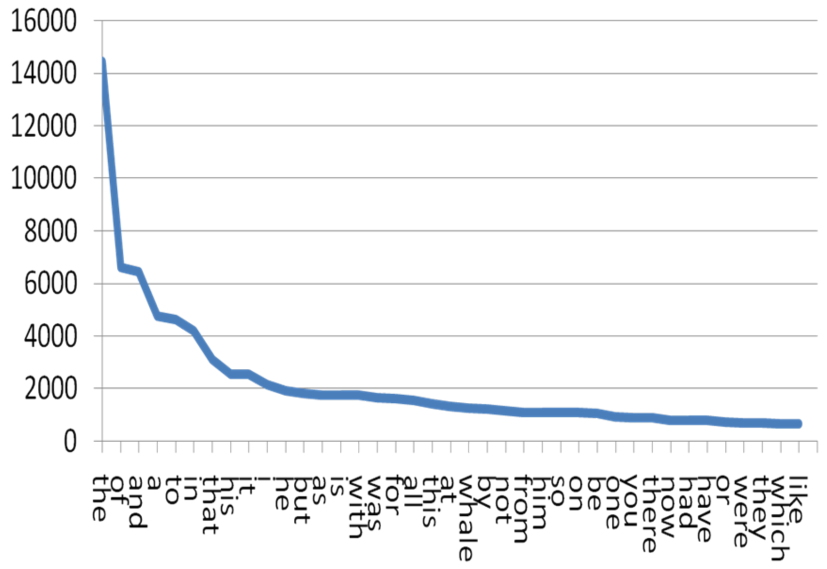


Figure 6.2: The word count graph for the book Moby Dick shows a classical Zipf curve. (source: NLTK)

We can see that proportionally to the total vocabulary of words used in the collection; only a few words account for the highest frequency. The drop-off of word frequency is very quick and followed by a "long tail" of low frequency words where the majority of words are found.

The vocabulary of the tenders is relatively limited and uses a formal, contract language, makes use of jargon (words with meaning specific to Canadian public tenders and the Canadian military) and abbreviations. There are also other non-words in the form of single numbers,

[2]`http://en.wikipedia.org/wiki/Zipf's_law` Jan 2014

Relevant	Not-Relevant	Unlabeled
inc	canada	must
canada	must	may
services	services	manitoba
contract	contract	canada
requirement	requirements	services
support	may	bid
task	requirement	contract
must	information	tender
consulting	agreement	agreement
requirements	bid	requirement

Table 6.4: Top 10 words by frequency for each of the original corpus categories (after removing stop words and punctuation, all words to lowercase)

letters and special characters used as reference numbers or bullet points, among other uses. These all make the text very difficult for a default classification pipeline[3].

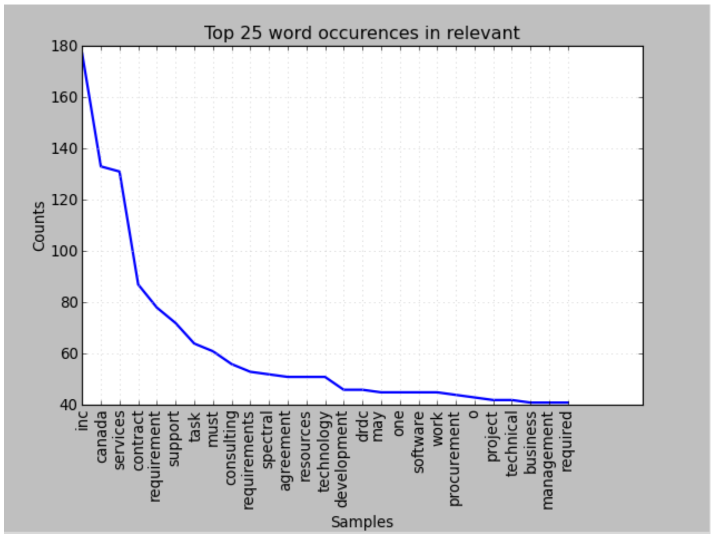


Figure 6.3: The top 25 words by frequency for the relevant notices of the original corpus

From Figure 6.3 and Figure 6.4, we can see how the top words are very similar.

Without some clever text modeling, the notices are difficult to separate cleanly into distinct *relevant* and *not-relevant* classes.

[3]Indeed, our first try with the raw data using all default settings with a state of the art machine learning package (the excellent scikit-learn) could not even beat a coin flip!

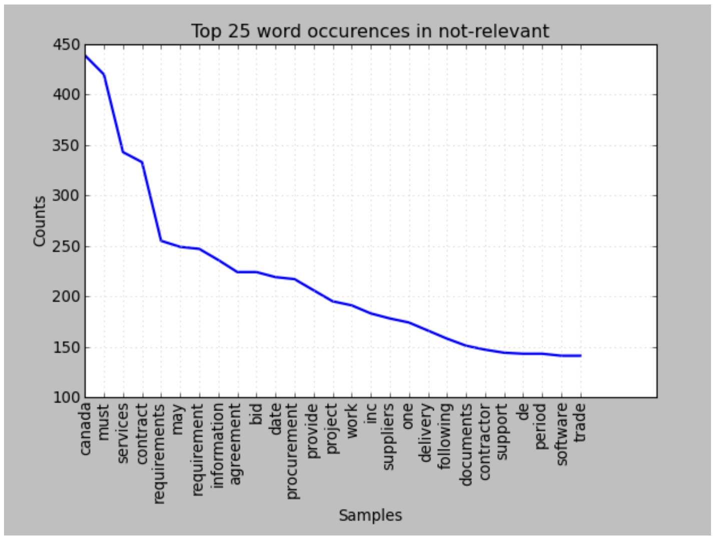


Figure 6.4: The top 25 words by frequency for the not relevant notices of the original corpus

6.2.5 Abbreviations and Acronyms

Looking through a sample of notices, there were many abbreviations and acronyms. These are often found with slight variations that if normalized to a single form, could help try a better performing classificator. However, the decision was taken early on to not build or purchase any specialized ontology.

6.2.6 Contact Information

Most notices have contact information at the end, with a name, telephone number and address. This is a potential source of information but requires specialized processing in order to extract it, and doesn't fit with the Bag of Words approach of our main machine learning system.

In Borkar et al. [2001], a system that can automatically extract postal address and such from text is presented. The problem, while not of extreme difficulty, is difficult enough that it has the potential to become a project in itself.

Therefore, while keeping contact information has the potential to be useful, we chose to not to pursue this avenue for lack of time.

6.2.7 Complications Due to Special Characters

Machine learning algorithms don't have any understanding of what working on beyond the fact they are different numbers. Those vectors are built from the tokens of each document.

Words that are the same should map to the same entry in the dictionary. Words that look similar should therefore have a similar score (or term weight).

The problem is that what our human eye sees as being the same can in fact be read as completely distinct words by the computer.

To give a concrete example, let's look at the single quote character. There are seven (!) similar characters that look like it: ASCII Apostrophe, backtick, opening single quote, closing single quote, prime, acute accent and grave accent ([4]).

It results from the differences of character codes (ISO Latin-1, UTF-8, ASCII, etc.) and keyboard types of the writer's environment (OS/software) and the character code of the reader's environment. Tokenization will not be able to differentiate leading to a diminished signal for words that have them.

There were also issues because of the mix of French and English, since even though we will only focus on English text, they still contain French-language proper nouns as well as stray French words.

6.2.8 Named Entity Recognition (NER)

We had initial hopes of using NER to help get information from the notices. We tried the Stanford NER Tagger[5] as well as the Natural Language Toolkit's (NLTK) NER module. Our experiments showed very poor results, so this technique was abandoned.

A NER tagger must be pre-trained on a tagged corpus first. It will only perform well if the text to be tagged is very similar to the text the tagger was trained on.

In this case, our data is very specific and we have no way to train the tagger on similar data.

Adding NER to the pipeline is a low-hanging fruit for future improvements. At first, having correctly tagged entities would allow positive identification of relevant organizations and companies, which is of obvious benefit. It is an admittedly labor-intensive task to hand tag the training data though. We believe that a cost-benefit analysis needs to be done before any further work on this topic.

[4] `http://www.cs.sfu.ca/~ggbaker/reference/characters`
[5] `http://nlp.stanford.edu/software/CRF-NER.shtml`

6.2.9 Reinterpreting the Categories: an Important Breakthrough

After much experimentation, we eventually re-interpreted the problem by proposing a new split of the notices.

The original set had only two labeled categories: Relevant and Not-Relevant.

Relevant This tender is relevant to the manager using the application.

Not-Relevant The tender is not relevant to the manager using the application.

Our new understanding lead us to add a new category called spam and redefine not-relevant.

Relevant This tender is relevant to the manager using the application.

Not-Relevant The tender is not relevant to the manager using the application, *but in the general domain of technology and consulting*.

Spam The tender has nothing to do with Fujitsu and IT consulting. It's completely irrelevant.

We will show statistics on the evolved versions of the corpus in the following step of the CRISP-DM process (section 6.3).

6.2.10 Initial Exploratory results

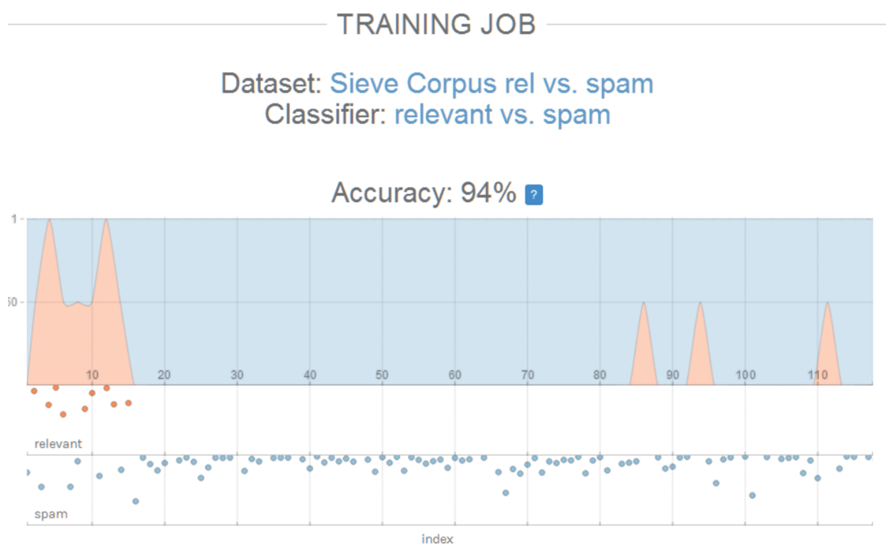
From the raw data, it is already possible to use text classification packages and libraries or even internet text classification APIs (Application Programming Interface). This will provide important early information on the difficulty of the problem.

Using Scikit-learn on the raw data with all default options and the SVM classifier, we could barely reach over 50% accuracy on predictions of relevant vs. not-relevant. The top features lists were full of 'junk' as well, which indicates we will need to spend a lot of time on data preparation and cleaning.

Running the model on the unlabeled notices and checking the ones predicted to be relevant was equally unpromising, they were mostly notices that were not even about IT, let alone have any chance of being labeled relevant by a real business user.

Another useful tool for exploratory analysis is to use one of a growing number of machine learning API services. Some of them offer text classification facilities that are very easy to use and claim very high performance.

One such service is called etcML, short for Easy Text Classification with Machine Learning. It is the result of work by Richard Socher, a doctoral candidate in computer science at Stanford (see `http://www.etcml.com/`).


TRAINING JOB

Dataset: Sieve Corpus rel vs. spam
Classifier: relevant vs. spam

Accuracy: 94%

Figure 6.5: etcML classification results for relevant vs. spam

We tried our corpus on their system, and we can see from Figure 6.5 that the results are very similar to our first results, which used a completely unoptimized classifier with Tf-Idf.

In general, we would expect that such generalist services will not be able to perform well on a specialized, difficult corpus such as the one we have to work with for this project. Still, the value of getting such quick and visually interesting results can be a useful tool to obtain preliminary results.

The next section will present our first attempts at solving the problem and making the corpus more amenable to vectorization. We have to keep in mind that the machine learning algorithm working on the vectors can only work with the information that is there. Time spent pre-processing the data to remove noise and make the tokenization process as successful as possible can have a strong influence on the final performance of our system.

6.2.11 Conclusion

As we have explained in this section, the sieve corpus proved to be difficult to deal with "as is". This is a common problem when dealing with real-world data and especially common when dealing with natural language text content ([Manning et al., 2008b]).

The next step of the process was to iteratively prepare the data, use it to build classification models and evaluate the models.

6.3 Data Preparation for the Prototype

In this phase, we explain what data was selected and how it was cleaned and formatted for use in the subsequent modeling phase.

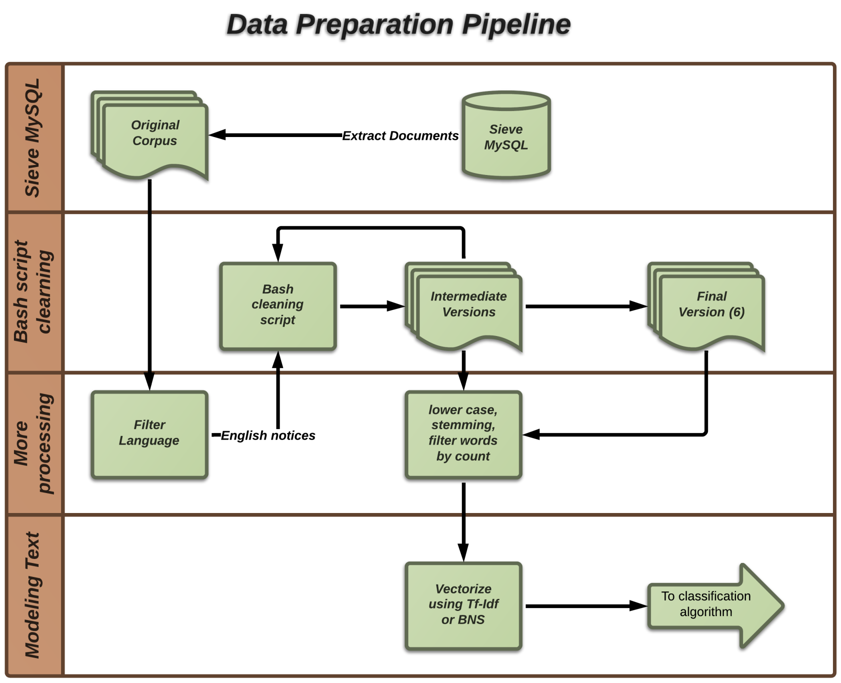


Figure 6.6: The data preparation step was an iterative process involving many round trips from the bash script to classification

The data preparation process is illustrated in Figure 6.6. As was explained earlier, the data was extracted from the Sieve database. Then we filtered out notices that were not in English. Next we process the notices with a bash script to clean unwanted special characters, standardize the formatting. The cleaned corpus is then further processed in Python to make the words all lowercase, perform stemming and other operations. This is the version that is finally converted to vectors and which can be classified.

The process was highly iterative. Considerable effort was put into looking at intermediary results to spot problems like unwanted meaningless words. As such problems were identified, the relevant script was improved.

Note: A short description of the data flow diagram shown in Figure 6.6 and later figures can be found in Appendix A.

6.3.1 Tools

We chose to use Bash Shell scripts because Bash programming it's a very productive scripting language for performing file and text oriented operations using regular expressions.

Language detection and filtering was done using a simple Python script and the natural language processing library NLTK.

In terms of performance, our scripts ran in minutes through the complete corpus on a standard laptop [6].

Had we had a much larger corpus, the tools are still usable as-is, by combining them with GNU Parallel. The data clean task can be safely run in parallel as each file is processed independently of others. For a corpus in the order of terabytes, it might become necessary to turn to a solution like Hadoop streaming.

Using Hadoop streaming, it's possible to run Python or bash scripts on a Hadoop cluster via the standard input and output (STDIN/STDOUT). This strategy, would require to load the raw corpus on the HDFS distributed storage first.

6.3.2 Extract Data

As discussed in Business Understanding section (section 6.1), we started with a dump of the sieve MySQL database. We also knew we would be focusing on the notice field and that some of them had been tagged as relevant or not-relevant by Sieve's users.

Our first job was to extract the notice field of each public tender into separate text documents, sorted by category: relevant, not-relevant and unlabeled. This was done using a simple bash script which piped the output of simple SQL queries to individual text files, one per tender.

We show the breakdown of number of notices by categories in Table 6.5 at this point in the process:

As is the case for text classification problems, the number of notices marked as relevant and not-relevant is very small compared with the total number of notices in the system (see Table 6.6).

[6]Fujitsu Lifebook E Series (Core i5 processor, 6GB ram)

Category	Number of notices
Relevant	23
Not-relevant	99
unlabeled	11456
Total	11578

Table 6.5: The original Sieve notices extracted from the MySQL database

What is more, looking at the Unlabeled notices, we quickly realized that there are much fewer relevant notices than not-relevant ones. This was also confirmed by our business user, who told us that in his experience, it was about a 1-100 ratio of relevant notices to not-relevant ones. So the ratio of 23 relevant to 99 not-relevant notices is misleading.

6.3.3 Get More Data

Category	Count
Relevant	23
Not-relevant	96

Table 6.6: The first version of the dataset was very small.

When moving to the large scale, we would expect the available French and bilingual notices to be enough to handle, and our system is indeed built in such a way that handling these notices is possible, all data being treated in a language-independent way. A classifier needs to be coded for each language and add an extra processing step at the beginning of the pipeline to perform language detection, an easy problem with many existing libraries that can do this task with near-perfect results.

Considering the very low number of labeled examples to the size of the collection, we can already expect that the classification will not be solved easily from the original data.

Our first classification results were indeed very low. The classification results were more often then not null classifiers, which predicted only 'not-relevant'.

In order to address this problem, we asked our business user to label more examples. Throughout the process, we refined the data cleaning scripts as well as eliminated non English notices. In total, we went through six versions of the corpus, as shown in Table 6.7.

Category	Original Corpus	Corpus 2	Corpus 3	Corpus 4	Corpus 5	Corpus 6
Relevant	23	51	56	58	62	86
Not-relevant	96	384	541	541	91	121
spam					450	513
Total	119	435	597	599	603	720

Table 6.7: The corpus evolved over the course of the project until its final version

6.3.4 Select Data

Focus on English Notices

We decided to restrict our project to English notices. The main reason for this choice is because of the lack of sufficient French language notices.

Our corpus contains a mix of English, French and bilingual English and French notices so we had to filter out the French and bilingual notices. The filtering was performed by using a simple Python script using a stop word detection model to distinguish between French and English notices. Bilingual notices were identified by their high counts of both French and English stop words.

The data preparation pipeline performs the standard step of converting all the words to lowercase and removing stop words ([Manning et al., 2008b]). No punctuation is kept either.

Contact Information

Most notices contain contact information at the end. We want to focus on the contents to classify the notices, so we want to filter out this information.

It turns out that filtering out the whole contact information in a reliable way is a problem in its own right, equivalent to extracting the fields ([Borkar et al., 2001]). This avenue was eventually dropped in favor of a more simple solution we explain in below.

6.3.5 Clean Data

For numerical data, cleaning the data is usually understood to mean dealing with missing or obviously incorrect values and so on. In the case where the data is text documents, we might consider data cleaning as standardizing the formatting of each document.

As we have explained previously, the corpus in its raw form contains notices of very different basic formatting, from no formatting at all, with the content of the notice being a single sentence or paragraph, to a highly formatted document with lists, headers, contact information footers.

Contact Information Fields

We converted the phone number, email and web links to standardized tags as follows:

- Phone numbers to PHONE_NUMBER_TAG
- Email addresses to EMAIL_TAG
- URLs (typically web site links) to URL_TAG

The cleaning script transforms the major contact information fields to common tags. It's a poor substitute for extracting the whole contact information, but we hoped it would lead to the tags getting very low Tf-Idf or BNS scores, and so be naturally filtered out as important features for the learning algorithm.

We checked the score for the tag EMAIL_TAG on the final corpus, shown in Table 6.8. As expected, the scores are close to zero, which means the tags will not be given much importance by the classification algorithm.

It's interesting to see that BNS scores the tag is almost equal to zero. This is not a surprise considering that the TAG is distributed evenly across the notices which means and considering how BNS works (see section 3.6.2).

Note that the Td-Idf score of the a given term depends on the document, so the score shown is the average score for all labeled notices.

Term Weighting	Score
Tf-Idf	0.143
BNS	0.005

Table 6.8: The Tf-Idf and BNS weights for the EMAIL_TAG is very low (final corpus).

Spacing

Spacing can help humans read a text and speed up comprehension, but it is pretty meaningless for a computer. Therefore all extra spaces and empty lines were removed.

Fancy ASCII Formatting

Notices are in a very basic text format, not in a sophisticated word-processor's format. The notices nonetheless include some creative use of various special characters in order to display lines and bullet points (e.g. *****, ———, _____).

Such efforts at formatting, while pleasing for their intended human viewers, wreck havoc on automated parsers. We removing these through regular expressions and checked their effect by checking the output we got from the later modeling steps. This illustrates the iterative nature of the process we followed, as we came back to this several times throughout the prototype construction every time we would notice garbled output downstream, such as in the top scoring words from BNS encoding.

We assume that such characters are fairly unlikely to yield information that could help a learning algorithm to separate relevant notices from not-relevant ones.

Special Characters

Our normalization process includes a step that addresses the special character problem using regular expressions to replace known variations with a common one.

The script is the result of successive small improvements from manually examining outputs. As such it is by no means complete, but reached sufficient coverage of the main problems where special characters did not seem to influence the results anymore. The code of the script is in the Annex in section D.2.

For accents, we stripped them away. Given a much larger corpus, it might be worth investigating if keeping the accents is a worthwhile change or not.

We made a point of grouping up everything we possibly could and live with some losses of bad groupings rather than letting the slight variations all count as separate tokens.

Code Pages and File Format

For historical reasons, computers, operating systems and tools were largely developed with only English in mind. Our corpus includes French and all related French characters like accents and quotes and the ISO Latin-1 character set preferred by Windows.

All documents are set to be in the UTF-8 format. This format is based on Unicode and is the default format of operation systems such as Linux and MacOS X and is well supported in Windows. UTF-8 is also the native format used in the programming languages such as java and python 3. Using UTF-8 throughout the pipeline eliminates the thorny issue of conversion between character encodings.

The sieve tool is a windows application, and so the text saved to the database follows the windows line ending conventions which uses carriage return and line feed ("\r\n") as a line ending, whereas Unix uses just line feed ("\n"). Our development environment is Linux, so the cleaning scripts perform appropriate conversions to Unix-style line endings.

Stemming

The goal of stemming is to group up words that have a high likelihood of sharing the same meaning despite small differences in spelling. The best example would be "cat" vs. "cats".

We used the porter stemmer in both the prototype and the Hadoop system ([Porter, 1980]). The prototype used NLTK's implementation, the Hadoop system used the Lucene implementation.

The vocabulary size, shown in in Table 6.9 is reduce by about 25% for the relevant notices. The not-relevant notices are even more noticeably reduced, with less than half of the vocabulary remaining.

Category	Vocabulary Size (no stemming)	Vocabulary Size (with stemming)
Relevant	1903	1490
Not-relevant	6011	2556

Table 6.9: The original corpus vocabulary is reduced after stemming with the Porter stemmer.

The corpus evolved as the data cleaning script was refined and further notices were tagged by our business expert. The final version also benefits from stemming, as the vocabulary size is reduced by about a third in all three categories.

Category	Vocabulary Size (no stemming)	Vocabulary Size (with stemming)
Relevant	3820	2846
Not-relevant	5369	3856
Spam	10349	7398

Table 6.10: The final corpus vocabulary is also reduced from using the Porter stemmer.

Stop Words

The stop words are removed following the standard practices of the field.

The prototype uses NLTK for sentence tokenization and stop words removal. The list of stop words are comparable for both, and listed in section B.2.

6.3.6 The Full Data-Preparation Pipeline

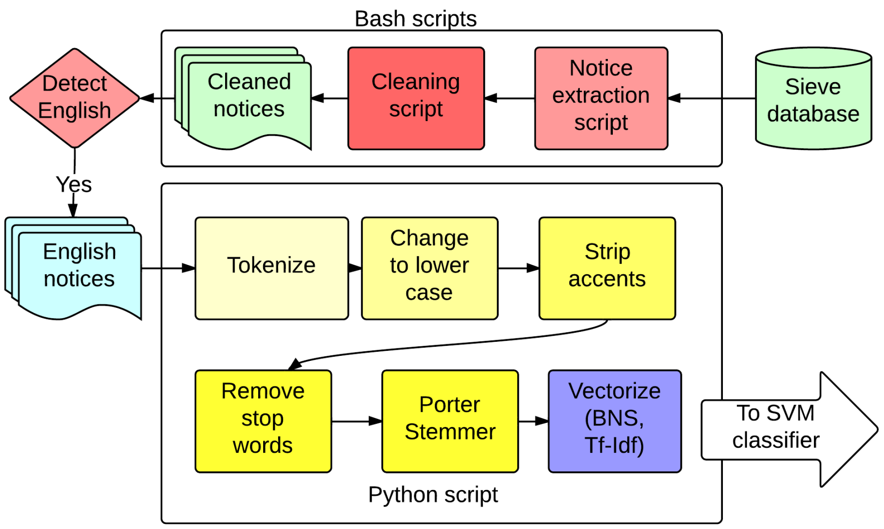


Figure 6.7: The full data preparation pipeline begins with the raw data and ends with vectors ready for use as inputs for the machine learning algorithms.

It is now possible to describe the full pre-processing pipeline that was used with the first prototype (Figure 6.7). First, in the top row, we can see how the data is extracted from the Sieve database and cleaned by the bash cleaning script. Then the language detection scripts filters out French and bilingual notices, leaving only English notices.

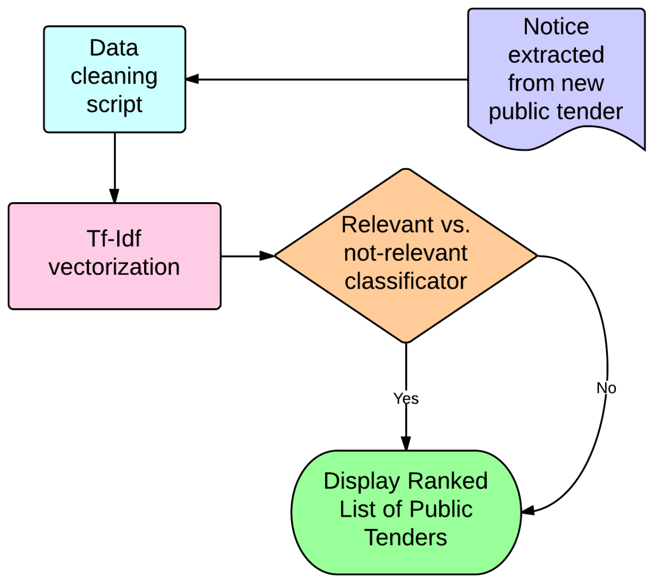
Next, the text from the notices is tokenized and processed using standard text processing techniques (see Manning et al. [2008b]). We change all tokenized words (also called tokens) to lowercase, then strip the accents, remove stop words and use the Porter stemmer to further reduce slight variations of similar words.

Finally, the data is ready to be transform the pre-processed notices into feature vectors. This will be described in the next section.

It's interesting to note that the process we describe here remains exactly the same for the final Hadoop large-scale system. While the technology (programming language etc.) used for Hadoop is quite different, it's solving the same problem and using the same ideas as solutions. Only their implementations, and potential scale, differ.

6.4 Modeling: Building the Prototype

6.4.1 Classification Pipeline



Notice extracted from new public tender
Data cleaning script
Tf-Idf vectorization
Relevant vs. not-relevant classificator
Yes
No
Display Ranked List of Public Tenders

Figure 6.8: The first classification model for public tender notices was too simple.

Our first model led to the process shown in Figure 6.8. The labeled examples are cleaned and vectorized using Tf-Idf. The data is used to train the SVM algorithm which outputs a trained prediction model.

We show in Figure 6.9 how new public tenders will go through the proposed pipeline for this prototype.

New tenders will have the notice field extracted. The notice information is then cleaned and vectorized. The vectorized notice can then go through the first classifier, which will either filter it out as spam or hand it off to the next classification model. If it's classified as spam, the public tender will be identified as spam in the database.

If it is classified as potentially relevant, then it will be classified by the second ranking classifier, which will label it with a relevance score.

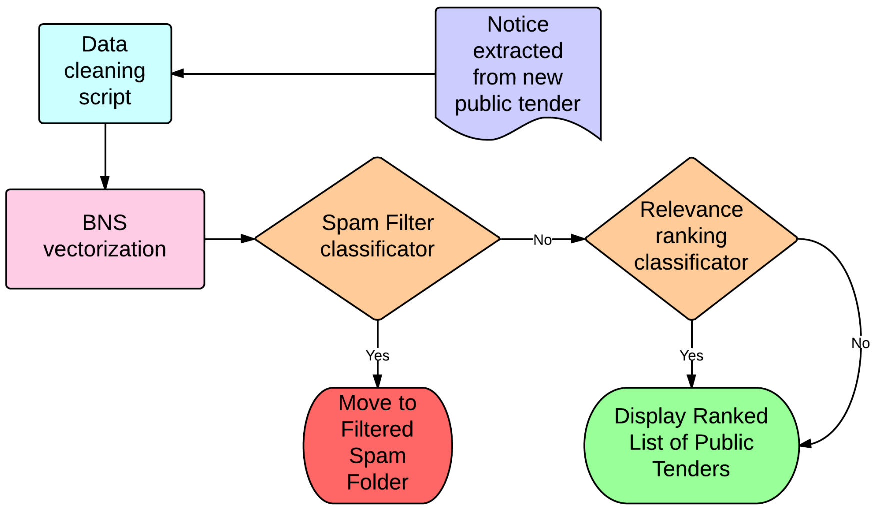

Notice extracted from new public tender
Data cleaning script
BNS vectorization
Spam Filter classificator
No
Relevance ranking classificator
Yes
Move to Filtered Spam Folder
Yes
Display Ranked List of Public Tenders
No

Figure 6.9: New tenders will go through two classificators after pre-processing and vectorization.

When a user accesses the system and asks for a listing of relevant notices, the list of all potentially relevant notices is displayed ranked by relevance. The relevance score and the highest scoring features of the notice can be shown as well to inform the business user and build trust.

6.4.2 Select Modeling Technique

There are two levels of modeling we used in our project. First is how to model the text documents as vectors. Second is the classification model.

Modeling Text

Representing text documents as vectors uses the well-known Vector Space Model (subsection 3.6.1).

Our experimentation naturally turned to Tf-Idf, the default algorithm for converting text to numerical vectors. At first, the classification algorithms with Tf-Idf on the original corpus often produced null classifiers, which always predict only one class.

Considering the very small sample size, and the noisy data, the initial results were not very meaningful. We worked with our business user who helped label additional notices to grow the labeled examples from the initial 119 notices to the final 720 notices.

We then tried several different classifiers, which we explain in the next section (section 6.4.2).

We experimented with n-grams and named entity recognition taggers without much success. Throughout this period, we were constantly refining the data cleaning scripts fixing issues as we identified them. We often looked at word frequency distributions and we constantly looked at the notices based on classification results. This step was critical in helping us deeply understand the characteristics of the corpus we had to work with.

Finally, after investigating the scientific literature led to our major breakthrough: Changing Tf-Idf itself for a better, class-aware alternative.

The idea came from George Forman's 2008 Paper on Bi-Normal Separation (BNS) for text classification ([Forman, 2008]).

Among Tf-Idf's flaws, the main one is that the label information is not used to compute the score. This is a major problem because we already have the class information in our dataset! The corpus of labeled examples is at the heart of supervised machine learning, and each document from the training set is labeled with its class. BNS uses this information in a simple but elegant algorithm that turns out to be very well suited to our problem.

The BNS algorithm was coded to integrate into Scikit-learn as a Transformer, relying on CountVectorizer to get the word counts used in the computations as described in Forman's paper.

Classification Model

Preliminary Modeling Results

For the classification model, we initially aimed to use the SVM classification algorithm. because SVM is well known to produce good results in text classification applications ([Joachims, 2012, 1998, Forman, 2008]).

While it is not known to scale much, we intend to use SVM as a gold standard, a useful benchmark against which we can compare the Hadoop version planned for deployment.

We used the state of the art Python machine learning library called Scikit-Learn ([Pedregosa et al., 2011]). The SVM estimator in Svikit-Learn uses LIBSVM ([Chang and Lin, 2011]) and LIBLINEAR ([Fan et al., 2008]) internally to handle the computations.

In keeping with our project methodological goals, we made a point of reusing as much of the built-in features of the library as possible.

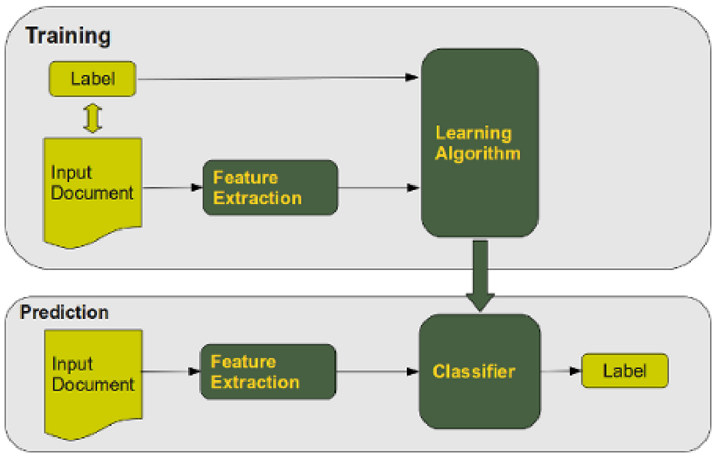


Figure 6.10: The classification process from input to results

While the initial plan was to use kernel support vector machines, we still experimented with Multinomial Naive Bayes as a comparison baseline. The superiority of SVM over Naive Bayes is well established ([Rennie and Rifkin, 2001]) and our system is no different. We also ran experiments with several different kernels for SVM including linear, polynomial and RBF kernels.

The RBF kernel SVM turned out to be the best classifier on the final corpus using all the best strategies and with optimized parameters. We checked regularly with the other two classifiers to confirm we were indeed using the best classification algorithm.

For SVM tuning, we chose a limited subset of possible values for the C and γ parameters and used a grid search approach to find the best parameters. The parameters we investigated are in the appendix's machine learning section (section B.3).

6.4.3 Model Assessment of the Prototype

The modeling step of this process turned out to be critical in reaching our final performance results. This is a good reminder that the best results are the result of hard work on this phase, as opposed to blindly throwing data at a machine learning library and working on parameter tuning.

Parameter tuning is important, but it is only the final step of the model construction phase. The most important factor turns out to be the insight of proper model selection.

The next section will show the results of our experiments throughout the process.

6.5 Evaluation of the Model

The test design follows the domain best practices, as explained in subsection 3.3.1. As shown in Figure 6.10, the classification algorithm, SVM in our case, is trained with labeled examples suitably formatted into numerical vectors. The resulting model can then be used to make predictions on new data.

For the prototype, we started with quick iterations over alternative approaches. Classification results often prompted changes to the data cleaning files, or led to improved data understanding or changes in the text or classification modeling.

With the BNS term weighting scheme implementation complete, we focused more on the parameter tuning work.

Our experimental process use a 50% train-test split and we use 5-fold cross-validation.

In the following sections, we will highlight the three main contributions to the prototype's final results, in chronological order.

As determined in partnership with our business user, the metric of highest interest is the recall for the relevant class. For Fujitsu, missing a relevant public tender directly leads to lost contracts. This is the metric that was the target of our optimization.

First we will show the gains of moving from Tf-Idf to the category-aware BNS term weighting scheme. Then, we will show results obtained after splitting out the spam category from the not-relevant category. Finally we will show how an unbalanced loss function which penalizes the classifier for making mistakes on the relevant class helped us reach a performance close to our initial business goals.

6.5.1 BNS an Improved Representation Over Tf-Idf for Text Classification

The results of Table 6.11 show the impact of changing the term weighting scheme from Tf-Idf to BNS. The numbers are not very reliable, because of the variance problem caused by the dataset's small size (see variance explained in 3.3.7). The results we show are indicative of the boost in performance from using BNS instead of Tf-Idf.

On the final corpus, with the number of examples increased from a little over 100 examples to over 700, the results speak for themselves. The recall for the relevant class for BNS is increased by about 10% over Tf-Idf. These results agree with the results from Forman [2008].

Metric	Original Corpus			
	Tf-Idf		BNS	
	Relevant	Spam	Relevant	Spam
Precision	0.65	0.94	0.78	0.95
Recall	0.38	0.96	0.72	0.96
F1	0.48	0.95	0.76	0.96

Table 6.11: BNS improves the classification performance over Tf-Idf on the original corpus

Metric	Final Corpus			
	Tf-Idf		BNS	
	Relevant	Spam	Relevant	Spam
Precision	0.82	0.94	0.93	0.95
Recall	0.71	0.98	0.83	0.96
F1	0.76	0.96	0.87	0.96

Table 6.12: BNS improves the classification performance over Tf-Idf for the final corpus

On the final corpus, BNS term weighting makes a big difference, with recall on the relevant class increasing by 12% over the best results obtained with Tf-Idf.

Word	BNS Score
valcartier	1.8948
drdc	1.8466
fujitsu	1.8130
ibm	1.8130
defence	1.7199
softwar	1.7175
analys	1.6109
test	1.5550
gis	1.4771

Table 6.13: A selection of top words as scored by BNS

Also, looking at the top words by BNS score make a lot of sense (Table 6.13). The relevance was determined by Guy Michaud, who works on very specific types of projects at Fujitsu. The areas are easy to detect from the words selected here, which were all among the top 50 scoring words. *valcartier* and *drdc* refer to the Canadian Defence's Valcartier military base, Guy's most important target customer. We can see that **fujitsu** and **ibm** are other very highly predictive words according to BNS, which makes a lot of sense.

6.5.2 Category Redefinition: Solving the Right Problem

Realizing that the not-relevant class was actually two classes mixed into one was an important breakthrough. It changed the problem from straight classification to the more reasonable filtering and ranking view that was ultimately adopted.

The first classification step filters out the more readily identifiable irrelevant notices we labeled "spam". This step is conservative by design, in order to minimize the chances of incorrectly identifying a relevant notice as spam. With BNS, our final results show that this will happen about once every twenty notices.

The second step is a classifier that isn't used to classify, but rather to output a probability of relevance. The notices that are not filtered as spam will be ordered by this score. This is a qualitative step that our business user has verified to be somewhat reasonable.

The data from the original corpus is very limited, the classification results vary a lot depending on the combination of parameter tuning and how the examples were ordered (we shuffle the data each run). Notices vectorized using Tf-Idf term weighting lead to null classifiers the majority of the time.

We can compare the best results we could get from the original corpus to the much more stable results we got on the recall metric (see Table 6.14). The metrics show how comparing relevant to spam is also worth about a 11% improvement in recall for BNS and 86% for Tf-Idf.

Measured	Original corpus	Final corpus	Increase (%)
	Relevant vs. not-relevant	Relevant vs. Spam	
Recall (Tf-Idf)	0.38	0.71	86.8
Recall (BNS)	0.72	0.83	15.2

Table 6.14: Splitting not-relevant into spam and not-relevant helped the classification algorithm differentiate notices

As we show in Table 6.15, the final, fully tuned system was able to identify spam almost perfectly with a precision of about 99%. This means when the system predicts a notice is spam, it really is spam 99% of the time. The recall of 97% indicates that 97% of all possible spam notices were correctly labeled as spam. This means that about 3% of the spam will remain after the first classification step.

Metric	BNS - Spam
Precision	0.99
Recall	97

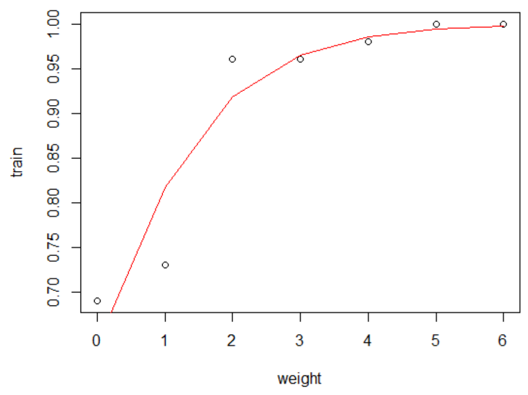
Table 6.15: Final precision for spam detection

As we explained earlier, there is a ratio of about 99-1 of irrelevant tenders to relevant ones. The fact that our prototype is able to reliably filter out about 97 of the 99 means the business user is now able to view a very short list with notices that have a much higher chance to be relevant.

The recall on the relevant notices is the important factor however, and the two step classification makes this clear and gives our system the highest chance to meet our project business goals.

6.5.3 The Final Push: Unbalanced Class Weights

Our experiments led us to deeply investigate the use of Scikit-learn classification with SVM. There is a parameter that allows the data to have unbalanced weights. This means errors of one class can be more expensive than the other. This is obviously relevant to our problem, since the relevant class is much more critical than the not-relevant class.



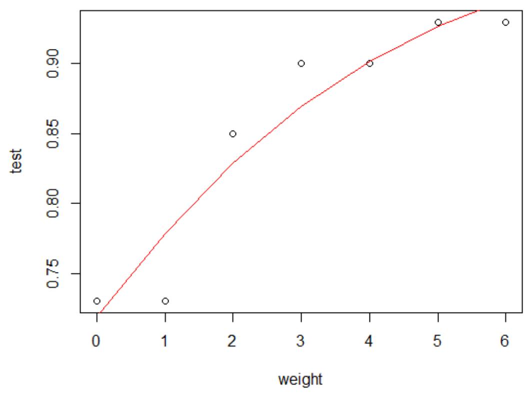


Figure 6.11: The recall increases with the weight on relevant examples when training

Figure 6.12: The cross-validated recall is best at a weight of 5

We experimented with many different weights for the relevant class, from 1 (the default) to 7, as shown in Figure 6.11 and Figure 6.12. As the weight of the relevant class is increased the recall metric also increases. We settled on the weight 5 as the optimal setting, as the precision results drop beyond it.

6.6 Deployment

This was a prototype and was not meant for production use. The true deployment target is the virtualized Hadoop cluster installed at Fujitsu's office.

The process of developing and deploying the large scale Hadoop solution is described in the next chapter (chapter 7).

6.7 Prototype: Final Review

Our final classification system used the 6th revision of the corpus of labeled notices originally extracted from the Sieve application's MySQL database.

The corpus is English-only. All notices in French or in both English and French are filtered out before further processing. Then, the corpus is tokenized, additional pre-processing is performed (stemming, lower-case, stop words, etc.) and then notices are vectorized using the BNS term weighting algorithm.

The pipeline does not do any feature selection, we rely only on the term weights computed using the BNS algorithm, as suggested by Foreman. All features are single words (unigrams).

Finally, the vectorized corpus is shuffled randomly and split into a train and test set. The best classificator, is SVM with RBF Kernel. The best parameters are C=100 and γ=0.0001 and were found using a grid search strategy using 5-fold cross validation. All metrics were calculated as averages of 10 run, with a new random shuffling of the examples. We always used a 50% train-test split.

The results of the first classification step, which filters the spam public tenders, reach our business goals. The number of irrelevant public tenders will be cut down drastically while preserving with high confidence potentially relevant ones.

We reach this conclusion based on the recall metric on the relevant class, which reaches 93%. This is very close to the business target of 95%. We can see that over-weighting the relevant class led to a decrease in the precision. This is normal, since it's much preferable to mislabel

Metric	Spam Filter		Relevance Ranker	
	Relevant	Spam	Relevant	not-relevant
Precision	0.81	0.99	0.72	0.67
Recall	0.93	0.97	0.65	0.60
F1	0.86	0.98	0.68	0.64

Table 6.16: Final classification results for both classification models

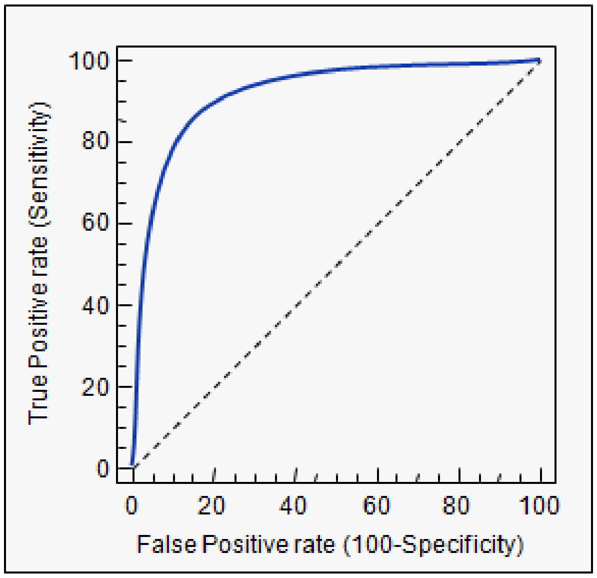


Figure 6.13: The ROC curve for relevant vs. spam

a spam notice as a relevant one rather than the opposite. This is the fundamental difference between precision and recall in the context of this project. This is further supported by the ROC curve shown in Figure 6.13.

The combination of BNS, the two-step classification process and unequal weights led to results that came very close to the high expectations of our business user.

The second classification step's results are less interesting, given the business problem we are trying to solve and the small size of the dataset. First, the labeled examples are just very unlikely to allow the relevant vs. not-relevant classificator to have good performance.

As for the second classification step, our experimentation is only a demonstration of one possible way it could be done. The ranking we obtained show that relevant public notices tend to "bubble up" and less relevant ones tend to rank lower, though its still relatively mixed. Our business user deemed this aspect of the solution acceptable.

We did not use up all possible machine learning techniques on our prototype. It's quite possible that further work on the prototype could push up recall performance above the 95% mark. Among many techniques we did not use, we think n-grams, co-occurrences and a simple ontology would have some potential worth exploring.

The next step is clear, we need to see if the approach we developed for the prototype can be implemented in a large scale system on a Hadoop cluster. Will it be possible to apply the same key insights to perform the task at very large scale, or will there be a bottleneck that makes the whole approach unworkable? Answering this question will be be the goal of the second half of our project, detailed in the next chapter.

Chapter 7

Experimental Log: Moving to the Large Scale with Hadoop

7.1 Business Understanding

Given the results from the prototype, we learned that the machine problem is not only tractable, but not especially difficult.

The final results came close enough to our original targets to be accepted by our business user. Now the challenge is to reuse all the same ideas using a completely different machine learning library, different programming languages and infrastructure.

Our business goal for this phase of the project is to take our prototype and build a proof-of-concept big data system that can solve the same problem. The business goal here is to explore in a practical way the state of the art big data tools and technologies.

In big data terminology, our machine learning prototype can be understood as an unstructured data analytics system. Using machine learning on this problem is very much at the state of the art.

Concretely, our business problem is to implement our prototype to work on a Hadoop cluster, and show that it can scale to potentially serve all of Fujitsu, a huge multi-national company.

The main target changes from solving a straightforward machine learning problem to a combination of technical and scalability problems. Each part of the pipeline must be made to scale well by taking advantage of the cluster resources as much as possible.

7.2 Data Understanding

The Hadoop-based classification pipeline uses the same corpus from the prototype. This is a reasonable choice as the machine learning problem we have to solve remains the same. Keeping the same dataset also makes the comparison of the classification performance more meaningful between the prototype and the Hadoop system.

All we learned about the data from building the prototype is still valid now, but the deployment target is now a large scale distributed system where scalability is a core requirement. Starting with a deep understanding of our data helped to keep the focus on this phase on the implementation and scalability challenges.

7.2.1 Large Scale Synthetic Dataset

Using the final prototype corpus to test the scalability of our system was out of the question, it's just too small.

Our solution was to generate several large synthetic corpus that would have roughly the same statistical properties as the Sieve notice corpus, but at a much larger scale. Concretely, a simple probabilistic trigram model of each category of our corpus generated new random English texts.

7.3 Data Preparation with Pig

Data cleaning required no additional work. All the pre-processing done in the prototype after the data cleaning step did need to be redone, however.

With our new goal of building a Large scale version of our prototype based on Hadoop, we must move to a Java-centric world. Therefore, we relied on the well known Lucene library as the core library for all the pre-processing steps that had been done with Python libraries when developing the prototype. This was not a compromise, we found that each step of the process could be done equally well with Lucene.

It's also important to note that while the processing steps have the same desired effect as for the prototype, they are different in terms of implementation. For example, tokenization is a notoriously subtle process and the NLTK *wordpunct* tokenizer we used for the prototype is not identical to the Lucene StandardAnalyzer tokenizer used in the Hadoop system.

We believe that such differences are relatively small, however. We did not expect these implementation differences to unduly influence the results, and the experimental results do seem to support our position.

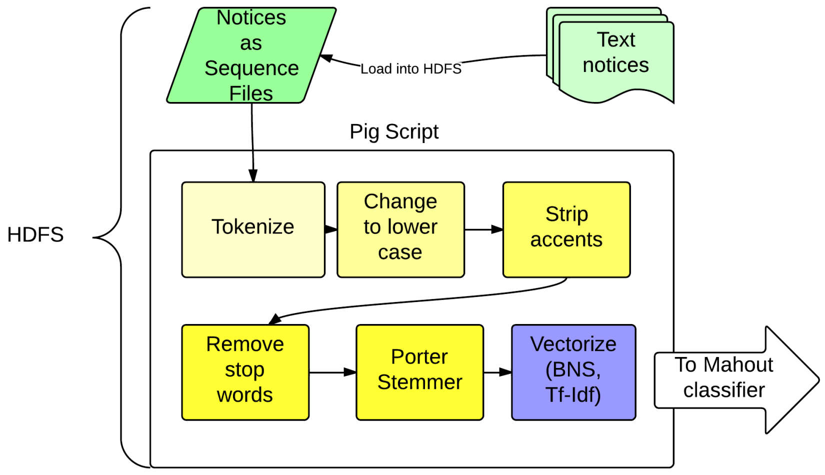


Figure 7.1: For the Hadoop system, we replace Python with Pig

We were able to setup a nearly identical pre-processing pipeline as the prototype by using Apache Pig (explained in subsection 2.4.1). The Pig pipeline is shown in Figure 7.1.

The files are read one line at a time, in parallel. Each line is tokenized and pre-processed appropriately. The tokens are counted and also used to build a dictionary. This dictionary is combined with the counts to get the term weights using either the BNS or Tf-Idf algorithms.

The tokenized documents are then transformed from text to numbers using the computed term weights. Finally, the scored documents are converted into a format suitable for the machine learning library Mahout.

This final, straightforward solution was not the first working solution. Our first implementation used a custom MapReduce job by saving the vectors to a text representation, which is very easy in Pig, and then parsing the file in a Mapper and outputting the result in the correct format in the Reducer. Both methods are equivalent, because that's how the library is implemented behind the scenes.

Using Elephant-Bird did have the advantage not to have to output to text first, a saving on disk access at large scale. Also, it avoids the need to coordinate with another job after that. It's best to avoid dependencies of this type if at all possible in production.

In the end, the simplification afforded by the Elephant-Bird library was worthwhile, despite the (considerable) difficulty of learning how to get it to work.

Note: We remind the reader that the data flow diagram used in Figure 7.1 and other such diagrams in this chapter follow some simple shape conventions as explained in the appendix in Appendix A.

7.3.1 Importing to HDFS

Before anything else, to process data with Hadoop, it's necessary to import that data into Hadoop's distributed storage called HDFS. We explain HDFS in subsection 2.3.5.

The data we have to work with is small, so importing it to HDFS by hand was an acceptable solution.

In a large scale production system, we might connect the database directly to HDFS using Flume or Sqoop (section 2.4). With the raw data on HDFS, it would be fairly easy to adapt our cleaning scripts to work as a Hadoop Streaming job (subsection 2.3.6).

Sequence Files

Hadoop does not work well with many small files. It's a lot more efficient to group them all into one large continuous file, where each document is one line. One way to do this is by storing the data into a Hadoop-specific format called *sequence files*. The Hadoop API includes methods to programmatically transform data formatted as key-value pairs into a sequence file.

Mahout includes a fairly sophisticated command line interface that can further simplify the transformation process called *DirectoryToSequenceFile*.

From Pig, it's possible to read and write directly to and from sequence files using the library Elephant-Bird[1], developed at Twitter.

7.4 Modeling: Large Scale Classification with Mahout

7.4.1 Classification Pipeline for Hadoop

The data processing pipeline is conceptually the same as the one that was developed for the prototype. However, the technical infrastructure and programing tools are completely different as we are targeting a Hadoop cluster. The process is shown in Figure 7.2.

At scale, we have to worry about any bottlenecks to parallel performance. For training, we use Pig to build the models, which guarantees that the computations are done in parallel as

[1]`https://github.com/kevinweil/elephant-bird`

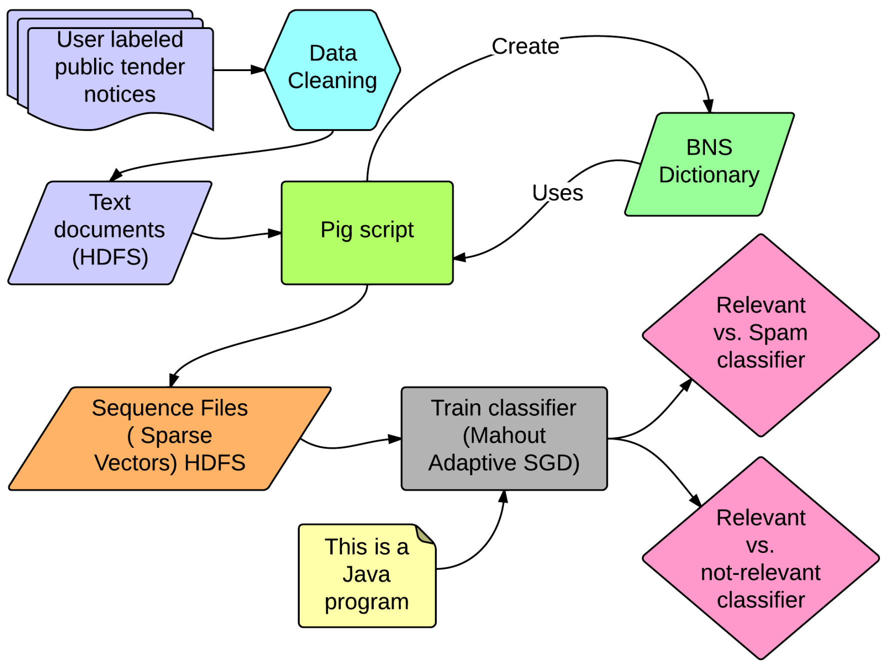

User labeled public tender notices
Data Cleaning
Create
BNS Dictionary
Uses
Text documents (HDFS)
Pig script
Relevant vs. Spam classifier
Sequence Files (Sparse Vectors) HDFS
Train classifier (Mahout Adaptive SGD)
Relevant vs. not-relevant classifier
This is a Java program

Figure 7.2: The processing pipeline on Hadoop uses Pig to vectorize the text notices.

much as possible on the Hadoop cluster. In production, the trained classifiers can be used in parallel to make predictions on new public tenders, as well. In other words, the prototype machine learning model of training two classifiers seems to have good chances to scale well.

In addition, for training, the first classifier will filter out about 97-98% of the public tenders before handing the rest to the second one. The second classification task is, therefore, about 100 times smaller. It is now a much easier task, from a size perspective. Secondly, we will show that in terms of scalability, BNS is a considerable improvement over Tf-Idf.

7.4.2 Text Modeling

In the prototype, we found that BNS was a superior text modeling alternative to the ubiquitous Tf-Idf. We will continue to use this algorithm for computing term weights.

It's notable that the Mahout library only includes the Tf-Idf algorithm for processing text documents.

Learning Algorithm Selection

We move from SVM to a form of parallel logistic regression called Adaptive SGD. It is a self-tuning version of stochastic gradient descent that implemented by Mahout to run in parallel on a Hadoop cluster automatically. The fitness used by default is the AUC (Area Under the Curve) that we explain in subsection 3.5.6.

7.5 Evaluation of the Large-Scale Model

7.5.1 Classification Performance

The results for Tf-Idf were obtained by using Mahout's built-in text classification pipeline and use *feature hashing* (also known as the *hashing trick* and described in Weinberger et al. [2009a]). The BNS vectorized results are based on our own custom pipeline. All our results are the average of 10 runs over all the data, since we are now using an online algorithm.

Metric	Tf-Idf (Mahout)	BNS (Pig)
AUC	0.97	0.97
Positive precision	0.72	0.90
Positive recall	0.76	0.91
Negative precision	0.95	0.98
Negative recall	0.99	0.98
Accuracy	0.95	0.97

Table 7.1: Mahout classification of relevant vs. spam with Adaptive Logistic Regression

The classification results of relevant vs. spam are shown in Table 7.1. The recall on relevant notices for vectorized with BNS is quite good, nearly reaching the 93% result of the prototype.

The results hide a greater variability; each classification run could vary between 86% and 96%. This is in line with expectations for a method using SGD, which uses randomness by nature. We would expect that with (much) more data, the results would converge. The learning curve is shown in Figure 7.3.

The precision and recall of the spam notices is even better but does not exceed error margins. It is not much use however, because we cannot filter by eliminating spam, but rather by keeping relevant notices. The recall of the relevant class is the critical measure to be able to give a guarantee to our business user that no relevant notice will be lost.

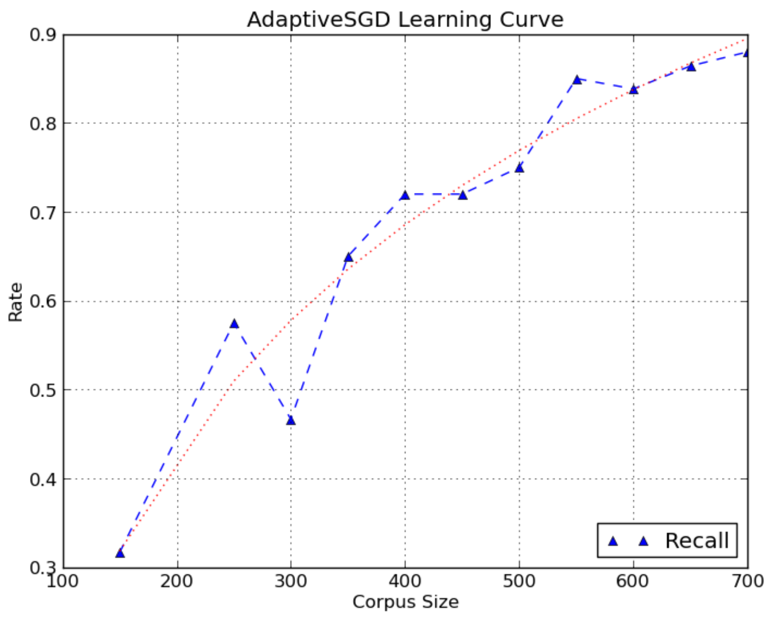


Figure 7.3: The recall improves with corpus size

The relevant vs. spam results using the built-in Mahout text classification pipeline are surprisingly good. They are very similar to the results we were able to achieve with our prototype. The prototype scored 0.82 precision and 0.71 recall on the relevant class, whereas the Mahout text classifier achieves 0.72 in precision and 0.76 recall. We believe we can attribute some of the difference to the benefits of the feature hashing (section 3.7.3).

The results are certainly encouraging for this proof-of-concept project. Further work on a truly large scale text corpus would be of certain benefit for further improving the classification performance.

We compared the results of our prototype with Mahout's SVM and SGD algorithms (Figure 7.4). The SVM results are very close to the prototype's results. This is not surprising because the implementation of the SVM used by Mahout is LIBLINEAR[2]. This is the same state of the art implementation used by Scikit-learn, at least for the linear kernel SVM. The difference can probably be attributed to the difference between the RBF kernel and the linear kernel.

Note that the SVM implementation in Mahout is currently not parallel, so this is nothing but an attempt at an apples-to-apples comparison of SVM (scikit-learn) vs. SVM (mahout) to show that the information is still there and that results are largely similar. We are not using it

[2]see MAHOUT-333 at`https://issues.apache.org/jira/browse/MAHOUT-334`

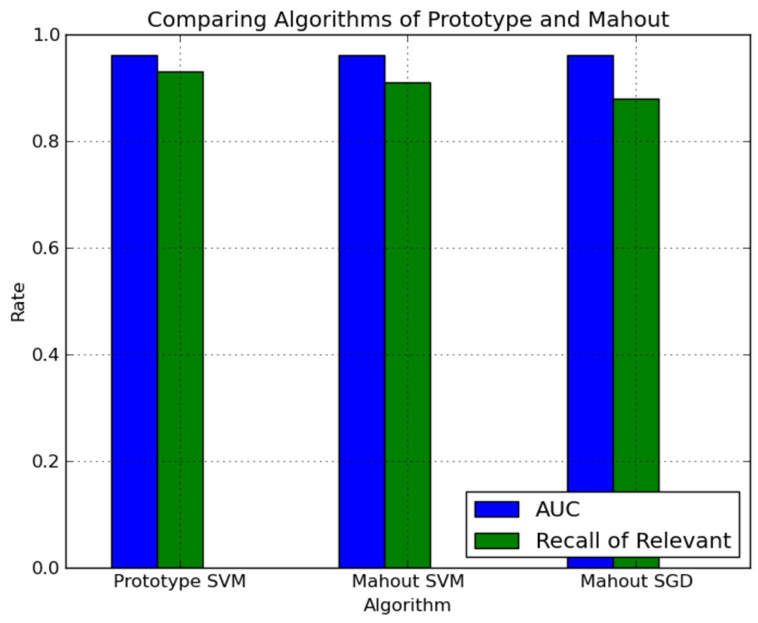


Figure 7.4: Comparing the prototype with Mahout's SVM and SGD algorithms

as a main algorithm as the goal of our Hadoop system is demonstrated scalability.

7.5.2 Scaling Performance

Scalability of the system was tested on a Hadoop cluster with 9 nodes (for details see section 7.6) using a synthetic dataset derived from the sieve corpus.

The main components of our system are the pig script, which transforms the text data into vectors, and then training the classification algorithms. Using Pig, were we able to show very interesting results on running time for processing a 1TB test dataset.

Processing the original corpus with Pig using the BNS vectorization script on our local machine took about 30s. Running it on the cluster took much longer, over two minutes. This is typical of Hadoop jobs on clusters, where there is a considerable start-up penalty on jobs.

We can see in Table 7.2 that as the number of nodes on the Hadoop cluster increases, the running time is cut by about 40%. The benefits promised by using Pig to automatically scale the processing are readily apparent.

Note that the largest size of dataset we could process given our hadoop cluster hardware configuration was 2TB.

Number of Nodes	Time (s)	Speedup (%)
1	782	
2	412	47
4	239	42
8	140	41

Table 7.2: The BNS vectorization Pig script running time vs. Hadoop cluster size for a 1TB dataset

7.6 Deployment: Using the Model in Production

The final deployment target was a Hortonworks Data Platform 1.31 cluster of 9 virtualized nodes. The server running the virtualized cluster was a Fujitsu PRIMERGY BX900 S1 Blade Server. Each blade is a Quad-socket server with an Intel Xeon 7500 series processor and 64 GB ram. The host OS was Microsoft Windows Server 2013 running the HyperV virtualization software. Each virtual machine (VM) was configured with 8GB ram and a 500GB local disk. The OS for each VM was Linux CentOS 6.3.

Deploying the system in production involved changing the paths in configuration files[3] and making sure all the data is imported into the HDFS distributed storage.

The process to run the pipeline, as shown in Figure 7.5, is to run the system is simply to run the pig script on the cluster. The script processes the notice text corpus as input and outputs sequence files containing all the notices as pairs of file name (full path) and it's vectorized contents as a Mahout sparse vector. The java libraries are all stored on HDFS. There are a number of ways to make sure all nodes have access to the necessary libraries; this is but one of them.

Once the pig scripts have finished processing the input files, all that is left is to train and evaluate a Mahout classificator. Our implementation is fairly close to the one proposed in Anil et al. [2010].

In production, the models can be saved to the file system (local, HDFS, etc.). These models can be used to classify new public notices, if they are suitably vectorized.

[3]Make sure the NameNode and the JobTracker addresses are in the hadoop-site.xml file

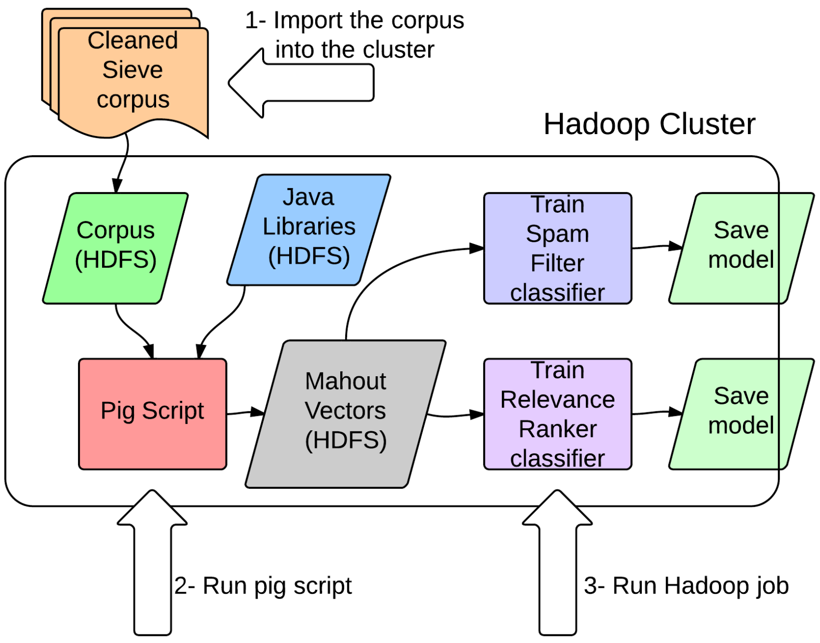


Figure 7.5: Executing the Hadoop solution's pipeline

7.7 Large Scale Hadoop Prototype: Final Review

How did this phase of our project fare, in terms out the initial business goals? There were two main goals: to implement the prototype on a Hadoop cluster and to explore and understand the issues and challenges of building a real-world Hadoop analytics solution.

We believe this phase of our project was successful on both count:

- We were able to learn to install and operate a Hadoop cluster and to run various programs on the cluster and to verify their performance. The project was an ideal context to exploring all activities surrounding the development of a Hadoop system such as the programming tools, the libraries and APIs and the general work-flow as well.
- The machine learning portion of the problem is still solved, the classifiers we trained are performing at a high level consistent with the results obtained with the prototype.

We will present our final report for this project in the next chapter, followed by a discussion of future works.

Chapter 8

Conclusion

8.1 Final Report

In the introduction, we boiled down our project to the task of answering the following two questions:

1. Can we address Sieve's shortcomings using machine Learning?
2. If such a solution can be found, could it be implemented as part of a full text classification pipeline on an Apache Hadoop cluster?

In this section, we will argue that our answer to both of these questions is a resounding "yes!".

8.1.1 The Classification Problem

The first part of the problem was to see if it we could solve the deficiencies of the Sieve prototype. The main problem with that prototype, as is discussed at length in the introduction, is that the list of public tenders shown to the user is not very useful. Relevant and not relevant and spam-like irrelevant notices were mixed haphazardly in such a way that the project was ultimately canceled.

First, we selected the supervised learning technique called classification to try and solve this problem. For this, we asked our business user to provide us with a training set of labeled notices.

With our business user, we agreed that the most important performance measure of the system was the recall metric of the *relevant* class. This means that our classification algorithm must be nearly perfect in identifying correctly that relevant notices are indeed relevant, even at the cost of making more mistakes on 'not-relevant' notices.

This performance goal was nearly reached with our prototype, with a 93% recall. The main elements of our solution that allowed us to reach this performance level are the following:

BNS term weighting BNS includes information of the class in the term weights given to words. It is the single most important breakthrough that has helped us reach our performance goals.

Redefinition of the classes The *not-relevant* was two different classes mixed together. The majority of the notices were spam-like, in that they had nothing to do with Fujitsu's IT consulting business. Once we split them out into their own *spam* class, we saw a marked improvement in our results.

Unbalanced loss Our dataset is severely skewed, with a 100-1 ratio of spam notices to relevant ones. As it is of critical importance that we make few errors on the relevant class, we eventually reached our best results by adding a large 7:1 penalty to the SVM classification algorithm.

8.1.2 Large-Scale Hadoop System

We were almost able to replicate the classification performance of our Scikit-learn-based prototype with the full-scale Hadoop solution. The small loss can be attributed to the small size of the data, which favors the SVM algorithm used by the prototype over Mahout's Logistic Regression. The superiority of SVM as the best general-purpose classification algorithm for a dataset of this size is by now very well established.

However, given a much larger dataset, we would expect that the Mahout online classifier would eventually reach, or even surpass the performance of SVM. This is supported by "big data" machine learning theoretical findings such as those in Bottou and LeCun [2004, 2005] and Bottou [2010] and .

The difference with the large-scale system is that it is engineered to scale in a way that the prototype never could. From the raw text documents in HDFS, the system performs the processing intensive vectorization processing with either BNS or tf-idf in parallel.

The performance gain compared with a single node was not very large, however, due to the set-up cost of the job itself. However, as the dataset grows in size, the set-up cost becomes smaller and smaller until it is negligible compared with the time required to process the job itself.

8.1.3 The Project as a Proof-of-Concept

Our master's project was an opportunity for the Fujitsu Innovation Center to get the first experience with Hadoop and big data analytics applied to text. On this score, our project was unquestionably very successful.

The expertise we gained throughout the project was shared through several presentations to Fujitsu's staff. In addition, the project made possible a technology transfer of a new Fujitsu Hadoop distribution from Japan to the FIC. Finally, the project served as the first contact with Hadoop technology for several Fujitsu consultants, some of whom were finally assigned to work with Hadoop in subsequent assignments.

Within the project, we also had a chance to join the Hadoop World conference in New York in 2012 and 2013. This privileged contact with Hadoop and big data's technological leaders was another factor in the success of the project. It helped us orient our technical choices as well as contributing to the learning and expertise of the FIC with big data and Hadoop.

The Hadoop system was technically challenging. The tools are recent, with still many rough edges. Indeed, the learning curve was very steep. Truly, this is one area that is at the very leading edge of applied computing technology.

8.2 Future Works

It is unfortunately impossible to work on a project of this scope without having some ideas that could not be pursued due to a lack of time or resources. Also, our solution is not a production-ready system yet, and there are some improvements that would be necessary for the system to be used by Fujitsu managers. Finally, in the medium to long term, we also have several ideas of how we could evolve the system into a true production grade Hadoop unstructured data analytics system.

8.2.1 Going Beyond the Notice Field

The first thing to say is that the entire machine learning model is based on the notice field. Public tenders have much more information available, including titles, dates, budgets and so on. It would make sense that including these kinds of information into our learning process would increase the performance of the system considerably.

One way to do this is through ensemble methods, where we train many machine learning models on different features, then produce a final decision by combining the output of each model. A popular form of ensemble learning that is known to work very well is majority voting. In this ensemble method, the final output is the output of the majority.

8.2.2 Ontologies and NER

For example, we could have a model trained on words from a Fujitsu ontology, another trained only on the title of public tenders and so on. We would reasonably expect to substantially increase the performance and robustness of the system in this way.

We strongly believe that adding a named entity recognition component would have made a big difference as well. Using NER, we could have reliably identified out relevant organizations like the Valcartier base as well as Fujitsu and its competitors. These would undoubtedly made strongly predictive features.

Unfortunately, it's not possible to get good results from NER without training it over a corpus that is similar to the one it will be used on. We had no way to properly train it, and so the approach was dropped.

8.2.3 Ranking Public Tenders

Using our system to only filter out easily identifiable not-relevant tenders still reduces the number of tenders displayed to the user by between 95 and 99%. The ranking is then obtained by a second classification model on the relevant vs. not-relevant notices.

As the classifier of the Hadoop system is based on logistic regression, we automatically get a probability score between 0 and 1 that we can use to rank the tenders. Based on feedback we got from our business user, this is already sufficient to make the system useful. In this context, further work to thoroughly evaluate the quality and relevance of the ranking was not a high priority.

Nevertheless, given more time, we would have done additional experimental work on the ranking issue.

8.2.4 Predicting the Chances of Winning a Bid

This system can be seen as only a first step in a much more capable and sophisticated system. But what if we could give it access to new, related sources of information?

Another project at the Fujitsu Innovation Center was a large-scale ontology-based document indexing project based on Apache Solr[1]. This is a system that can open up employee resumes, reports, contracts and bid documents to our own system.

Firstly, using employee resumes can be readily used to better learn how to identify relevant public tenders, as we can add information on the skills currently available to perform the work required by the public tender. Another potentially rich source of information would be the history of public tenders as well as their associated bid documents.

But more speculatively, this information could conceivably be used to learn a regression model that could compute the odds of winning the contract of relevant public tenders. This is potentially valuable information that can help decision makers prioritize which public tenders to bid on and which to pass on. Such information leads to improved decision making and resource allocation strategy based on real data, as opposed to only the experience, knowledge and instincts of managers.

8.2.5 An Intelligent Global Tender Agent

Ultimately, our system could become the first step in an intelligent IT public tender virtual assistant. Such an agent could produce near real-time, personalized listings of public tenders that various Fujitsu offices around the world might bid on.

Given enough data sources, the agent could recommend optimal bid prices that would balance the low price necessary to win the contract and high price needed to maximize profits.

By taking into account the skills and availability of software developers, project managers and other specialized knowledge workers, the agent could suggest the best Fujitsu branch to do the work.

For a global company such as Fujitsu, there is a clean benefit to scaling up such a system to encompass its huge IT consulting business. It is currently very difficult for global organizations to properly coordinate their resources of skilled knowledge workers and infrastructure across regions and countries.

[1]`https://lucene.apache.org/solr/`

Having the means of keeping track of private tenders and public tenders from municipalities, provinces, states and countries in a globally integrated and intelligent way presents huge business opportunities. Such a system would represent a very real competitive advantage against it's global rivals IBM and HP.

8.3 Parting Words

In conclusion, this project attacked a problem with a large scale approach that opens up exciting new possibilities and tangible benefits. Such possibilities are simply not possible without the technological revolution that is big data.

Today, there is not much work left to convince businesses of the potential benefits of big data. The true challenge is to reach this potential. This is exactly where machine learning will shine.

We believe this project makes a case for how machine learning applied to big data is a game changer upon which the data-driven enterprise of tomorrow will be built.

Bibliography

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.99. URL `http://dx.doi.org/10.1109/TKDE.2005.99`.

Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *CoRR*, abs/1110.4198, 2011.

Deepak Agarwal. Computational advertising: The linkedin way. In *Proceedings of the 22Nd ACM International Conference on Conference on Information &#38; Knowledge Management*, CIKM '13, pages 1585–1586, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2263-8. doi: 10.1145/2505515.2514690. URL `http://doi.acm.org/10.1145/2505515.2514690`.

Leonidas Akritidis and Panayiotis Bozanis. A supervised machine learning classification algorithm for research articles. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 115–120, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480388. URL `http://doi.acm.org/10.1145/2480362.2480388`.

Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X, 9780262012430.

Xavier Amatriain. Building industrial-scale real-world recommender systems. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 7–8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1270-7. doi: 10.1145/2365952.2365958. URL `http://doi.acm.org/10.1145/2365952.2365958`.

Xavier Amatriain. Mining large streams of user data for personalized recommendations. *SIGKDD Explor. Newsl.*, 14(2):37–48, April 2013. ISSN 1931-0145. doi: 10.1145/2481244.2481250. URL `http://doi.acm.org/10.1145/2481244.2481250`.

Chris Anderson. The end of theory: The data deluge makes the scientific method obsolete. *Wired*, June 2008.

Rie Kubota Ando and Lillian Lee. Mostly-unsupervised statistical segmentation of japanese kanji sequences. *Nat. Lang. Eng.*, 9(2):127–149, June 2003. ISSN 1351-3249. doi: 10.1017/S1351324902002954. URL `http://dx.doi.org/10.1017/S1351324902002954`.

Robin Anil, Sean Owen, Ted Dunning, and Ellen Friedman. *Mahout in Action*. Manning Publications Co., Sound View Ct. 3B Greenwich, CT 06830, 2010. URL `http://manning.com/owen/`.

Itamar Arel, Derek C. Rose, and Thomas P. Karnowski. Research frontier: Deep machine learning–a new frontier in artificial intelligence research. *Comp. Intell. Mag.*, 5(4): 13–18, November 2010. ISSN 1556-603X. doi: 10.1109/MCI.2010.938364. URL `http://dx.doi.org/10.1109/MCI.2010.938364`.

David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5. URL `http://dl.acm.org/citation.cfm?id=1283383.1283494`.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=2180912.2180915`.

Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 26–33, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics. doi: 10.3115/1073012.1073017. URL `http://dx.doi.org/10.3115/1073012.1073017`.

Eda Baykan, Monika Henzinger, and Ingmar Weber. A comprehensive study of techniques for url-based web page language classification. *ACM Trans. Web*, 7(1):3:1–3:37, March 2013. ISSN 1559-1131. doi: 10.1145/2435215.2435218. URL `http://doi.acm.org/10.1145/2435215.2435218`.

Ron Bekkerman, Mikhail Bilenko, and John Langford. Scaling up machine learning: Parallel and distributed approaches. In *Proceedings of the 17th ACM SIGKDD International Conference Tutorials*, KDD '11 Tutorials, pages 4:1–4:1, New York, NY, USA, 2011.

ACM. ISBN 978-1-4503-1201-1. doi: 10.1145/2107736.2107740. URL `http://doi.acm.org/10.1145/2107736.2107740`.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

Kevin S. Beyer, Vuk Ercegovac, Rainer Gemulla, Andrey Balmin, Mohamed Y. Eltabakh, Carl-Christian Kanne, Fatma Ozcan, and Eugene J. Shekita. Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283, 2011. URL `http://dblp.uni-trier.de/db/journals/pvldb/pvldb4.html#BeyerEGBEKOS11`.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. *SIGMOD Rec.*, 30(2):175–186, May 2001. ISSN 0163-5808. doi: 10.1145/376284.375682. URL `http://doi.acm.org/10.1145/376284.375682`.

Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL `http://leon.bottou.org/papers/bottou-98x`. revised, oct 2012.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer. URL `http://leon.bottou.org/papers/bottou-2010`.

Léon Bottou and Yann LeCun. Large scale online learning. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. URL `http://leon.bottou.org/papers/bottou-lecun-2004`.

Léon Bottou and Yann LeCun. On-line learning for very large datasets. *Applied Stochastic Models in Business and Industry*, 21(2):137–151, 2005. URL `http://leon.bottou.org/papers/bottou-lecun-2004a`.

Léon Bottou and Chih-Jen Lin. Support vector machine solvers. In Léon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007. URL `http://leon.bottou.org/papers/bottou-lin-2006`.

Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on well-clusterable data. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 26–40. SIAM, 2011. URL `http://dl.acm.org/citation.cfm?id=2133036.2133039`.

Amy Brown and Greg Wilson. *The Architecture Of Open Source Applications*. lulu.com, June 2011. ISBN 1257638017. URL `http://www.aosabook.org/en/`.

Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. Haloop: Efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1920841.1920881`.

Michael Cardosa, Chenyu Wang, Anshuman Nangia, Abhishek Chandra, and Jon Weissman. Exploring mapreduce efficiency with highly-distributed data. In *Proceedings of the Second International Workshop on MapReduce and Its Applications*, MapReduce '11, pages 27–34, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0700-0. doi: 10.1145/1996092.1996100. URL `http://doi.acm.org/10.1145/1996092.1996100`.

Claudio Carpineto, Stanislaw Osiński, Giovanni Romano, and Dawid Weiss. A survey of web clustering engines. *ACM Comput. Surv.*, 41(3):17:1–17:38, July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541884. URL `http://doi.acm.org/10.1145/1541880.1541884`.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011. ISSN 2157-6904. doi: 10.1145/1961189.1961199. URL `http://doi.acm.org/10.1145/1961189.1961199`.

Edward Y. Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. Psvm: Parallelizing support vector machines on distributed computers. In *NIPS*, 2007. Software available at `http://code.google.com/p/psvm`.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1267308.1267323`.

Linchuan Chen, Xin Huo, and Gagan Agrawal. Accelerating mapreduce on a coupled cpu-gpu architecture. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 25:1–25:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press. ISBN 978-1-4673-0804-5. URL `http://dl.acm.org/citation.cfm?id=2388996.2389030`.

Yanpei Chen, Laura Keys, and Randy H. Katz. Towards energy efficient mapreduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-109.html`.

Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 895–903, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020558. URL `http://doi.acm.org/10.1145/2020408.2020558`.

Keunho Choi, Gunwoo Kim, and Yongmoo Suh. Classification model for detecting and managing credit loan fraud based on individual-level utility concept. *SIGMIS Database*, 44(3):49–67, August 2013. ISSN 0095-0033. doi: 10.1145/2516955.2516959. URL `http://doi.acm.org/10.1145/2516955.2516959`.

Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 281–288. MIT Press, 2006. URL `http://dblp.uni-trier.de/rec/bibtex/conf/nips/ChuKLYBNO06`.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995. ISSN 0885-6125. doi: 10.1007/BF00994018. URL `http://dx.doi.org/10.1007/BF00994018`.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1248547.1248566`.

Paolo Cremonesi, Franca Garzotto, and Roberto Turrin. Investigating the persuasion potential of recommender systems from a quality perspective: An empirical study. *ACM Trans. Interact. Intell. Syst.*, 2(2):11:1–11:41, June 2012. ISSN 2160-6455. doi: 10.1145/2209310.2209314. URL `http://doi.acm.org/10.1145/2209310.2209314`.

M. A. Dalal and N. D. Harale. A survey on clustering in data mining. In *Proceedings of the International Conference &#38; Workshop on Emerging Trends in Technology*, ICWET '11, pages 559–562, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0449-8. doi: 10.1145/1980022.1980143. URL `http://doi.acm.org/10.1145/1980022.1980143`.

Jeffrey Dean. Experiences with mapreduce, an abstraction for large-scale computation. In *PACT*, volume 6, pages 1–1, 2006.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251254.1251264`.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, and Lavanya Ramakrishnan. Riding the elephant: Managing ensembles with hadoop. In *Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers*, MTAGS '11, pages 49–58, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1145-8. doi: 10.1145/2132876.2132888. URL `http://doi.acm.org/10.1145/2132876.2132888`.

David J. DeWitt and Michael Stonebraker. Mapreduce: A major step backwards, 2008. URL `http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/`.

D. Dubin. The most influential paper gerard salton never wrote. *Library trends*, 52(4):748–764, 2004. URL `http://scholar.google.com.au/scholar.bib?q=info:2YGmxssyCvQJ:scholar.google.com/&output=citation&hl=en&as_sdt=0,5&ct=citation&cd=0`.

Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: A runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC

’10, pages 810–818, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8. doi: 10.1145/1851476.1851593. URL http://doi.acm.org/10.1145/1851476.1851593.

Kathleen Ericson and Shrideep Pallickara. On the performance of high dimensional data clustering and classification algorithms. *Future Gener. Comput. Syst.*, 29(4):1024–1034, June 2013. ISSN 0167-739X. doi: 10.1016/j.future.2012.05.026. URL http://dx.doi.org/10.1016/j.future.2012.05.026.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1390681.1442794.

Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006. ISSN 0167-8655. doi: 10.1016/j.patrec.2005.10.010. URL http://dx.doi.org/10.1016/j.patrec.2005.10.010.

Pedro Fazenda, James McDermott, and Una-May O’Reilly. A library to run evolutionary algorithms in the cloud using mapreduce. In *Proceedings of the 2012T European Conference on Applications of Evolutionary Computation*, EvoApplications’12, pages 416–425, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29177-7. doi: 10.1007/978-3-642-29178-4_42. URL http://dx.doi.org/10.1007/978-3-642-29178-4_42.

Eugen Feller, Lavanya Ramakrishnan, and Christine Morin. On the Performance and Energy Efficiency of Hadoop Deployment Models. In *The IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*, Santa Clara, États-Unis, October 2013. URL http://hal.inria.fr/hal-00856330. Grid’5000 Grid’5000.

George Forman. Choose your words carefully: An empirical study of feature selection metrics for text classification. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 2431 of *Lecture Notes in Computer Science*, pages 150–162. Springer, 2002. ISBN 3-540-44037-2. URL http://dblp.uni-trier.de/db/conf/pkdd/pkdd2002.html#Forman02.

George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3:1289–1305, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944974.

George Forman. Bns feature scaling: An improved representation over tf-idf for svm text classification. In *Proceedings of the 17th ACM Conference on Information and Knowledge*

Management, CIKM '08, pages 263–270, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-991-3. doi: 10.1145/1458082.1458119. URL `http://doi.acm.org/10.1145/1458082.1458119`.

Yasser Ganjisaffar, Thomas Debeauvais, Sara Javanmardi, Rich Caruana, and Cristina Videira Lopes. Distributed tuning of machine learning algorithms using mapreduce clusters. In *Proceedings of the Third Workshop on Large Scale Data Mining: Theory and Applications*, LDMTA '11, pages 2:1–2:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0844-1. doi: 10.1145/2002945.2002947. URL `http://doi.acm.org/10.1145/2002945.2002947`.

Alan Gates. *Programming Pig - Dataflow Scripting with Hadoop.* O'Reilly, 2011. ISBN 978-1-449-30264-1.

Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava. Building a high-level dataflow system on top of map-reduce: The pig experience. *Proc. VLDB Endow.*, 2(2):1414–1425, August 2009. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1687553.1687568`.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM. ISBN 1-58113-757-5. doi: 10.1145/945445.945450. URL `http://doi.acm.org/10.1145/945445.945450`.

Amol Ghoting, Prabhanjan Kambadur, Edwin Pednault, and Ramakrishnan Kannan. Nimble: A toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 334–342, New York, NY, USA, 2011a. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020464. URL `http://doi.acm.org/10.1145/2020408.2020464`.

Amol Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Yuanyuan Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 231–242, 2011b. doi: 10.1109/ICDE.2011.5767930.

David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

ISSN 0001-0782. doi: 10.1145/138859.138867. URL http://doi.acm.org/10.1145/138859.138867.

Hans Peter Graf, Eric Cosatto, Léon Bottou, Igor Dourdanovic, and Vladimir Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*, 2004. URL http://dblp.uni-trier.de/db/conf/nips/nips2004.html#GrafCBDV04.

Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, and Judy Qiu. Scalable parallel computing on clouds using twister4azure iterative mapreduce. *Future Gener. Comput. Syst.*, 29(4): 1035–1048, June 2013. ISSN 0167-739X. doi: 10.1016/j.future.2012.05.027. URL http://dx.doi.org/10.1016/j.future.2012.05.027.

Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Mach. Learn.*, 46(1-3):389–422, March 2002. ISSN 0885-6125. doi: 10.1023/A:1012487302797. URL http://dx.doi.org/10.1023/A:1012487302797.

Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, March 2009. ISSN 1541-1672. doi: 10.1109/MIS.2009.36. URL http://dx.doi.org/10.1109/MIS.2009.36.

Liangxiu Han and Hwee Yong Ong. Accelerating biomedical data-intensive applications using mapreduce. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 49–57, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4815-9. doi: 10.1109/Grid.2012.24. URL http://dx.doi.org/10.1109/Grid.2012.24.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

Bingsheng He, Wenbin Fang, Qiong Luo, Naga K. Govindaraju, and Tuyong Wang. Mars: A mapreduce framework on graphics processors. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, pages 260–269, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-282-5. doi: 10.1145/1454115.1454152. URL http://doi.acm.org/10.1145/1454115.1454152.

D. Hsu, N. Karampatziakis, J. Langford, and A. Smola. Parallel Online Learning. *ArXiv e-prints*, March 2011.

Xiangji Huang, Fuchun Peng, Dale Schuurmans, Nick Cercone, and Stephen E. Robertson. Applying machine learning to text segmentation for information retrieval. *Inf. Retr.*, 6(3-4): 333–362, September 2003. ISSN 1386-4564. doi: 10.1023/A:1026028229881. URL `http://dx.doi.org/10.1023/A:1026028229881`.

Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 693–702, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0055-1. doi: 10.1145/1835804.1835893. URL `http://doi.acm.org/10.1145/1835804.1835893`.

Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-64417-2. URL `http://dl.acm.org/citation.cfm?id=645326.649721`.

Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL `http://dl.acm.org/citation.cfm?id=645528.657646`.

Thorsten Joachims. *Learning to Classify Text Using Support Vector Machines*. Springer Publishing Company, Incorporated, 2012. ISBN 1461352983, 9781461352983.

Daniel Jurafsky and James H Martin. *Speech and language processing, 2nd edition*. Prentice Hall, 2008.

Hisashi Kashima, Tsuyoshi Idé, Tsuyoshi Kato, and Masashi Sugiyama. Recent advances and trends in large-scale kernel methods. *IEICE Transactions*, 92-D(7):1338–1353, 2009.

Hyun I. Kim, Sung Shin, Wei Wang, and Soon I. Jeon. Svm-based harris corner detection for breast mammogram image normal/abnormal classification. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, RACS '13, pages 187–191, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2348-2. doi: 10.1145/2513228.2513324. URL `http://doi.acm.org/10.1145/2513228.2513324`.

Youngjoong Ko. A study of term weighting schemes using class information for text classification. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1029–1030, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1472-5. doi: 10.1145/2348283.2348453. URL http://doi.acm.org/10.1145/2348283.2348453.

Alexios Kotsifakos, Evangelos E. Kotsifakos, Panagiotis Papapetrou, and Vassilis Athitsos. Genre classification of symbolic music with smbgt. In *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '13, pages 44:1–44:7, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1973-7. doi: 10.1145/2504335.2504382. URL http://doi.acm.org/10.1145/2504335.2504382.

Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001. URL http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

Pedro Larrañaga, Borja Calvo, Roberto Santana, Concha Bielza, Josu Galdiano, Iñaki Inza, José A Lozano, Rubén Armañanzas, Guzmán Santafé, Aritz Pérez, et al. Machine learning in bioinformatics. *Briefings in bioinformatics*, 7(1):86–112, 2006.

Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *ICML*, 2012.

Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. Parallel data processing with mapreduce: A survey. *SIGMOD Rec.*, 40(4):11–20, January 2012. ISSN 0163-5808. doi: 10.1145/2094114.2094118. URL http://doi.acm.org/10.1145/2094114.2094118.

Simone Leo and Gianluigi Zanetti. Pydoop: a Python MapReduce and HDFS API for Hadoop. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 819–825, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8. doi: 10.1145/1851476.1851594. URL http://doi.acm.org/10.1145/1851476.1851594.

Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, March 2010. ISSN 0163-5980. doi: 10.1145/1740390.1740405. URL http://doi.acm.org/10.1145/1740390.1740405.

Yuxi Li and Dale Schuurmans. Mapreduce for parallel reinforcement learning. In *Proceedings of the 9th European Conference on Recent Advances in Reinforcement Learning*, EWRL'11, pages 309–320, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-29945-2. doi: 10.1007/978-3-642-29946-9_30. URL `http://dx.doi.org/10.1007/978-3-642-29946-9_30`.

J. Lin. MapReduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *ArXiv e-prints*, September 2012.

Jimmy Lin and Alek Kolcz. Large-scale machine learning at twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 793–804, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1247-9. doi: 10.1145/2213836.2213958. URL `http://doi.acm.org/10.1145/2213836.2213958`.

Ying Liu and Han Tong Loh. A simple probability based term weighting scheme for automated text classification. In *Proceedings of the 20th International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems*, IEA/AIE'07, pages 33–43, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73322-5. URL `http://dl.acm.org/citation.cfm?id=1769938.1769944`.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=2212351.2212354`.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008a. ISBN 0521865719, 9780521865715.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008b.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. In *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pages 330–339, 2010. URL `http://www.vldb2010.org/accept.htm`.

Lawrence Mitchell, Terence M. Sloan, Muriel Mewissen, Peter Ghazal, Thorsten Forster, Michal Piotrowski, and Arthur S. Trew. A parallel random forest classifier for r. In *Proceedings of the Second International Workshop on Emerging Computational Methods* *for the Life Sciences*, ECMLS '11, pages 1–6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0702-4. doi: 10.1145/1996023.1996024. URL `http://doi.acm.org/10.1145/1996023.1996024`.

Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376726. URL `http://doi.acm.org/10.1145/1376616.1376726`.

Jiaul H. Paik. A novel tf-idf weighting scheme for effective ranking. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 343–352, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2034-4. doi: 10.1145/2484028.2484070. URL `http://doi.acm.org/10.1145/2484028.2484070`.

Balaji Palanisamy, Aameek Singh, Ling Liu, and Bhushan Jain. Purlieus: Locality-aware resource allocation for mapreduce in a cloud. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 58:1–58:11, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0771-0. doi: 10.1145/2063384.2063462. URL `http://doi.acm.org/10.1145/2063384.2063462`.

Biswanath Panda, Joshua S. Herbach, Sugato Basu, and Roberto J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. *Proc. VLDB Endow.*, 2(2): 1426–1437, August 2009. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1687553.1687569`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal*, 13:277–298, 2005. URL `http://research.google.com/archive/sawzall.html`.

István Pilászy and Domonkos Tikk. Recommending new movies: Even a few ratings are more valuable than metadata. In *Proceedings of the Third ACM Conference on Recommender*

Systems, RecSys '09, pages 93–100, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-435-5. doi: 10.1145/1639714.1639731. URL `http://doi.acm.org/10.1145/1639714.1639731`.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. URL `http://research.microsoft.com/~jplatt/smo.html`.

Martin F. Porter. An algorithm for suffix stripping. *Program: Electronic Library & Information Systems*, 40(3):211–218, 1980. ISSN 0033-0337. doi: 10.1108/00330330610681286. URL `http://dx.doi.org/10.1108/00330330610681286`.

Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, pages 13–24, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 1-4244-0804-0. doi: 10.1109/HPCA.2007.346181. URL `http://dx.doi.org/10.1109/HPCA.2007.346181`.

Saima Rathore, Mutawarra Hussain, Ahmad Ali, and Asifullah Khan. A recent survey on colon cancer detection techniques. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 10(3):545–563, May 2013. ISSN 1545-5963. doi: 10.1109/TCBB.2013.84. URL `http://dx.doi.org/10.1109/TCBB.2013.84`.

Padmashree Ravindra, Vikas V. Deshpande, and Kemafor Anyanwu. Towards scalable rdf graph analytics on mapreduce. In *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, MDAC '10, pages 5:1–5:6, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-991-6. doi: 10.1145/1779599.1779604. URL `http://doi.acm.org/10.1145/1779599.1779604`.

Jason D. M. Rennie and Ryan Rifkin. Improving multiclass text classification with the support vector machine. AI Memo AIM-2001-026, Massachussetts Institute of Technology, 2001. URL `http://people.csail.mit.edu/jrennie/papers/aimemo2001.pdf`.

Monica Rogati and Yiming Yang. High-performing feature selection for text classification. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 659–661, New York, NY, USA, 2002. ACM. ISBN 1-58113-492-4. doi: 10.1145/584792.584911. URL `http://doi.acm.org/10.1145/584792.584911`.

M Ruiz, L E Mujica, X Berjaga, and J Rodellar. Partial least square/projection to latent structures (pls) regression to estimate impact localization in structures. *Smart Materials and Structures*, 22(2):025028, 2013. URL `http://stacks.iop.org/0964-1726/22/i=2/a=025028`.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pages 673–695. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL `http://dl.acm.org/citation.cfm?id=65669.104449`.

G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975. ISSN 0001-0782. doi: 10.1145/361219.361220. URL `http://doi.acm.org/10.1145/361219.361220`.

Gerard Salton and Michael J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, New York, 1983.

Alexander Schätzle, Martin Przyjaciel-Zablocki, and Georg Lausen. Pigsparql: Mapping sparql to pig latin. In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 4:1–4:8, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0651-5. doi: 10.1145/1999299.1999303. URL `http://doi.acm.org/10.1145/1999299.1999303`.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002. ISSN 0360-0300. doi: 10.1145/505282.505283. URL `http://doi.acm.org/10.1145/505282.505283`.

James G. Shanahan and Norbert Roma. Boosting support vector machines for text classification through parameter-free threshold relaxation. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 247–254, New York, NY, USA, 2003. ACM. ISBN 1-58113-723-0. doi: 10.1145/956863.956911. URL `http://doi.acm.org/10.1145/956863.956911`.

Weiyi Shang, Bram Adams, and Ahmed E. Hassan. Using pig as a data preparation language for large-scale mining software repositories studies: An experience report. *J. Syst. Softw.*, 85(10):2195–2204, October 2012. ISSN 0164-1212. doi: 10.1016/j.jss.2011.07.034. URL `http://dx.doi.org/10.1016/j.jss.2011.07.034`.

Maad Shatnawi and Nader Mohamed. Statistical techniques for online personalized advertising: A survey. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 680–687, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0857-1. doi: 10.1145/2245276.2245406. URL http://doi.acm.org/10.1145/2245276.2245406.

Colin Shearer. The crisp-dm model: The new blueprint for data mining. *Journal of Data Warehousing*, 5(4), 2000.

Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, and Eric Rollins. F1 a distributed sql database that scales. In *VLDB*, 2013.

Hans-Ulrich Simon and Nikolas List. Svm-optimization and steepest-descent line search. In *COLT*, 2009. URL http://dblp.uni-trier.de/db/conf/colt/colt2009.html#SimonL09.

Lavneet Singh and Girija Chetty. A comparative study of mri data using various machine learning and pattern recognition algorithms to detect brain abnormalities. In *Proceedings of the Tenth Australasian Data Mining Conference - Volume 134*, AusDM '12, pages 157–165, Darlinghurst, Australia, Australia, 2012. Australian Computer Society, Inc. ISBN 978-1-921770-14-2. URL http://dl.acm.org/citation.cfm?id=2525373.2525392.

Pascal Soucy and Guy W. Mineau. Beyond tfidf weighting for text categorization in the vector space model. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1130–1135, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1642293.1642474.

Diomidis Spinellis and Georgios Gousios. *Beautiful Architecture: Leading Thinkers Reveal the Hidden Beauty in Software Design*. O'Reilly Media, Inc., 1st edition, 2009. ISBN 059651798X, 9780596517984.

Robert J. Stewart, Phil W. Trinder, and Hans-Wolfgang Loidl. Comparing high level mapreduce query languages. In *Proceedings of the 9th International Conference on Advanced Parallel Processing Technologies*, APPT'11, pages 58–72, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24150-5. URL http://dl.acm.org/citation.cfm?id=2042522.2042527.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009. ISSN 1687-7470. doi: 10.1155/2009/421425. URL http://dx.doi.org/10.1155/2009/421425.

Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1577069.1577091.

S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, June 2006.

Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, August 2009. ISSN 2150-8097. URL http://dl.acm.org/citation.cfm?id=1687553.1687609.

Richard Tzong-Han Tsai. Chinese text segmentation: A hybrid approach using transductive learning and statistical association measures. *Expert Syst. Appl.*, 37(5):3553–3560, May 2010. ISSN 0957-4174. doi: 10.1016/j.eswa.2009.10.004. URL http://dx.doi.org/10.1016/j.eswa.2009.10.004.

Radu Tudoran, Alexandru Costan, and Gabriel Antoniu. Mapiterativereduce: A framework for reduction-intensive data processing on azure clouds. In *Proceedings of Third International Workshop on MapReduce and Its Applications Date*, MapReduce '12, pages 9–16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1343-8. doi: 10.1145/2287016.2287019. URL http://doi.acm.org/10.1145/2287016.2287019.

Shijun Wang and Ronald M. Summers. Machine learning and radiology. *Medical Image Analysis*, 16(5):933–951, 2012.

Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification, 2010.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1113–1120, New York, NY, USA, 2009a. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553516. URL http://doi.acm.org/10.1145/1553374.1553516.

K.Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009b.

Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 3rd edition, 2012. ISBN 0596521979, 9780596521974.

Rüdiger Wirth. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, 2000.

Yu-Chieh Wu, Jie-Chi Yang, Yue-Shi Lee, and Show-Jane Yen. Efficient and robust phrase chunking using support vector machines. In *Proceedings of the Third Asia Conference on Information Retrieval Technology*, AIRS'06, pages 350–361, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-45780-1, 978-3-540-45780-0. doi: 10.1007/11880592_27. URL `http://dx.doi.org/10.1007/11880592_27`.

Yanfeng Zhang and Shimin Chen. I2mapreduce: Incremental iterative mapreduce. In *Proceedings of the 2Nd International Workshop on Cloud Intelligence*, Cloud-I '13, pages 3:1–3:4, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2108-2. doi: 10.1145/2501928.2501930. URL `http://doi.acm.org/10.1145/2501928.2501930`.

Zhuoyao Zhang, Ludmila Cherkasova, Abhishek Verma, and Boon Thau Loo. Automated profiling and resource management of pig programs for meeting service level objectives. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 53–62, New York, NY, USA, 2012a. ACM. ISBN 978-1-4503-1520-3. doi: 10.1145/2371536.2371546. URL `http://doi.acm.org/10.1145/2371536.2371546`.

Zhuoyao Zhang, Ludmila Cherkasova, Abhishek Verma, and Boon Thau Loo. Optimizing completion time and resource provisioning of pig programs. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 811–816, Washington, DC, USA, 2012b. IEEE Computer Society. ISBN 978-0-7695-4691-9. doi: 10.1109/CCGrid.2012.56. URL `http://dx.doi.org/10.1109/CCGrid.2012.56`.

Zhuoyao Zhang, Ludmila Cherkasova, Abhishek Verma, and Boon Thau Loo. Performance modeling and optimization of deadline-driven pig programs. *ACM Trans. Auton. Adapt. Syst.*, 8(3):14:1–14:28, September 2013. ISSN 1556-4665. doi: 10.1145/2518017.2518019. URL `http://doi.acm.org/10.1145/2518017.2518019`.

Dengya Zhu and Jitian Xiao. R-tfidf, a variety of tf-idf term weighting strategy in document categorization. In *Semantics Knowledge and Grid (SKG), 2011 Seventh International Conference on*, pages 83–90, 2011. doi: 10.1109/SKG.2011.44.

Appendix A

Data Flow Diagrams

"A data flow diagram is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. Data flow diagrams can also be used for the visualization of data processing (structured design)."[1]

We use data flow diagrams heavily in this thesis, to illustrate how data is processed from the raw input into a structured representation suitable for use by machine learning algorithms. Some examples of such diagrams are Figure 6.7 and Figure 7.1.

The key for the various shapes is shown in Figure A.1.

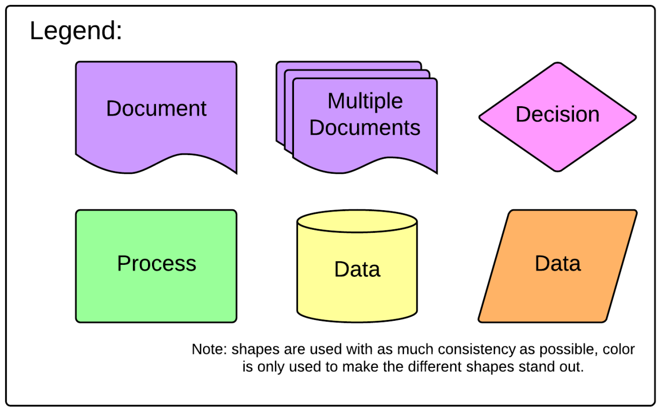

Legend:
Document
Multiple Documents
Decision
Process
Data
Data
Note: shapes are used with as much consistency as possible, color is only used to make the different shapes stand out.

Figure A.1: Legend for the data flow diagrams.

[1]`http://en.wikipedia.org/wiki/Data_flow_diagram` Jan 2014

Appendix B

Machine Learning

B.1 Advice on Work Flow for Applied Machine Learning

Technically speaking, it can be fairly easy to use supervised machine learning to classify some data. For example, the Python machine learning library Scikit-learn is backed with great tutorials and sample code. It also includes sample data that's ready to use.

However, we want to stress the importance of following a proper work-flow when working on a real project. The sequence of steps, from raw data to getting performance metrics, makes a big difference! Here we describe in more detail the sequence of steps that allow the practitioner to use machine learning with confidence.

This section is a shortened version of wei Hsu et al. [2010]. We recommend reading the whole paper for all the details and justifications.

B.1.1 What Not to Do

Commonly, the beginner's approach to using a machine learning package for a classification task is the following:

1. Transform the data to the format of the library used (Scikit-learn, SVMLight, Weka, Mahout, etc.)
2. Use the data as input on an algorithm (SVM, Logistic Regression, etc.)
3. Check the metrics
4. If metrics aren't satisfactory, randomly try different parameters values

5. Back to 2

Following this approach can sometimes be successful, which is a hint that the problem may have been an ’easy’ one. It also attests to the robustness of the machine learning approach.

However, this naive approach can easily lead to the case where the algorithm is over-trained on the example data, and will make many mistakes on new data, which is useless. It also makes the metrics very unlikely to have any bearing on real life performance. This can cause problems down the road when the production system doesn’t live up to expectations. Fixing a system with unreliable metrics is like driving with one’s eyes closed.

B.1.2 The Recommended Process

Applying a more rigorous and scientific approach is very much in the best interests of the practitioner. The goal is to drive the process in an informed way. More importantly, a more rigorous process is essential so that the results can provide provable guarantees on the generalization performance.

By generalization performance, we mean how well the learned model can be expected to perform on new, previously unseen data. It is indeed one thing to get 90%+ accuracy after training a model with SVM, and wholly another to get over 90% performance on previously unseen data.

The process we recommend is as follows:

1. Transform the data to the format of the library used (Scikit-learn, SVMLight, Weka, Mahout, etc.)
2. Conduct simple scaling on the data
3. Split the data into a train and test split. The 60-40 split is common, for approximately 1000 examples dataset.
4. Use cross validation to find the best parameters for the algorithm
5. Use the best parameters to train a model on the training set
6. Check the metrics on the test set

B.2 Stop Words

- NLTK: 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now'
- Lucene: "a", "an", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "such", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with"

B.3 Grid Search Parameters

Parameters:

n-grams 1,2,3,4,5

vocabulary size no limit, 15,000, 10,0000, 5,000

stop words with or without

vectorization model TF-IDF or BNS

minimum term count for a token none, 5-10-15-25-50

maximum term document frequency none, 0.5,0.6,0.7,0.8,0.9,0.99

SVM Parameter: C

SVM Parameter: Gamma

Unbalanced loss weight 1-1 to 15-1

Appendix C

Applied Hadoop: Some Technical Advice

C.1 Introduction

This section is a list of some topics we found to be special sources of errors and wasted learning time. We are brief by design, as all are well explained in HDG3.

C.2 Training

The major challenge facing the beginner trying to get into Hadoop related projects is the sheer breadth of expertise required to really ’get’ the technology. Starting with a keen mind, understanding MapReduce and HDFS is the first step. On that note, a quick web search on “Hadoop MapReduce introduction” will lead to numerous results including blogs, videos and slides. Nearly all of the top results are instructive and high quality, though they tend to stick to basics and quickly become repetitive once one gets the point.

For specific development, we recommend intermediate to expert Java programming skills as well as good understanding of SQL and comfort using Linux as a development environment. The Hortonworks virtual machine is an excellent resource, constantly updated with the latest versions of their tools and jam-packed with tutorials and great way to easily get one’s hands dirty. High level programming, with Pig and Hive are great, and analysts can quickly learn to use them effectively, but for more complex tasks, understanding the basics is pretty much a requirement.

Tackling larger problems with Hadoop requires expertise in GNU/Linux operations, system architecture, programming, databases and data analysis. These diverse skills requirements are a big part of what makes getting into Hadoop from the very start fairly challenging.

This problem is made all the more difficult by the large number of separate components that all interact together as the data flows from one to the other.

There is a lot of high quality material available from the Strata Hadoop World 2011-2013, some free, some available for purchase from O'Reilly's web store. The presentations detailing Hadoop systems in production are especially valuable and instructive.

Cloudera and Hortonworks are the two leaders in Hadoop related training. Their courses may be an effective way for the enterprise user to get an employee up to speed quickly. Their courses lead to certifications that have some value, though it's hard to measure just how much.

There are free alternatives to the expensive enterprise training courses. In that space, Coursera[1] and Udacity[2] are increasingly important leaders. They offer free, university level courses online.

Of note, we recommend Andrew Ng's Machine Learning class, as well as everything connected with Coursera's data science track. Udacity, in partnership with Cloudera and starting from January 2014, will offer several new courses on Hadoop, MapReduce and data science. These classes are of the highest quality and well worth the time and effort. Please check them out!

C.3 Technical Advice on Hadoop Programming

C.3.1 Input and Output formats

Hadoop uses the notion of Input and Output formats to identify the Key-Value pairs as they move from Input into the Map and reach Reduce and are output. Hadoop in this sense is really just a big Extract-Transform-Load (ETL) machine, with the Map and Reduce parts nothing more than format converters or schema converters. This very common use case led to the development of Pig at Yahoo Labs.

Hadoop already comes with the means to read data as Text, and has Writer classes for dealing with integer and double and the other data types of Java. It is also possible to write our own Reader and Writer classes and "natively" be able to read and write objects and files in any kind of format. As such, it is also possible to read and write from the HBase NoSQL database for example.

[1]`www.coursera.org`
[2]`www.udacity.com`

C.3.2 Matching up the map output and the reduce input

This is a big gotcha of developing a Hadoop MapReduce Job. First, at the job level, we have to tell Hadoop what the input and output formats will be, so that we can read the input file correctly, and Hadoop knows what to do with the Reduce output and write it into a file with the format we want. The tricky part is that at each step of the way, the input format and the output format must match perfectly or chaos ensues. The Job input must match the map input. The map output must match the reduce input, and the reduce output must match the Job output. Beware!!

C.3.3 Distributed memory and libraries

Hadoop doesn't distribute a job's library dependencies by default. One way to deal with this problem is to pack all the job's dependencies into a so-called super-JAR. Potential problems of this solution is increasing the job's startup time and increased network usage as the large jar is copied to each node of the cluster.

Another solution we explored is to store the dependencies on HDFS and so making them available to all nodes as needed. The dependencies can also be manually copied to the local machines. This is one examples of the many hidden technical issues in Hadoop that a beginner must inevitably deal with.

Also, it is sometimes useful to have shared information available to all map or reduce tasks. This is especially true for iterative algorithms. Again, there is the way of writing the information to a file on HDFS. Another possibility is to use distributed memory programatically. For more details,

C.3.4 Chaining Jobs

It's pretty hard to do any kind of useful real world application using only one set of map and reduce. Sometimes, it's hard to find a way to express an algorithm as a single map and reduce phase, even if it's technically possible. For the case of running an iterative algorithm, multiple steps are just the very nature of the task.

In those types of cases, the application will consist of a sequence of jobs, each taking the output of the previous job as input. This is a little bit hard! Firstly, the task of chaining the jobs is left entirely in the developer's hands. She must now programatically control the input and output of jobs so that the data can be moved from one job to the next. Each reducer outputs its own file to HDFS in the job's output folder. It's quite possible that more than one file be output. Depending on the type of processing, it might be necessary to deal with this issue by hand.

C.3.5 Learning to work with HDFS

Expect to have to deal with HDFS a lot. Real world production use of Hadoop requires learning to use HDFS for permissions, import and export of files between HDFS and the data sources like MySQL databases or logs on distant servers. This is hidden work that is often overlooked at a project kickoff that can actually take a significant effort until Hadoop is well integrated into the data environment and data flow of the organization.

As an organization's data flow becomes increasingly complex, large-scale development of custom systems to integrate Hadoop and data sources and data consumers may become necessary. For example, this is exactly what Facebook (Scribe [3]and Twitter[4] had to do.

C.3.6 Sequence files

Sequence files are a preferred file format by Hadoop, with several notable advantages:

- Allow compression and all its benefits, with minimal programmatic effort.
- Efficient use of storage: HDFS is more effective when processing a small number of large, continuous files, rather than a very large number of very small files. With Sequence Files, we can group up these small files into the preferred large files in a pre-processing step. It's a low hanging fruit to easily increase a job's performance.
- Automatically used by Hadoop Map and Reduce functions, Pig and Hive and even Mahout.

C.3.7 File compression and serialization

File compression brings two major benefits: it reduces the space needed to store files, and it speeds up data transfer across the network or to or from disk. When dealing with large volumes of data, both of these savings can be significant, so it pays to carefully consider how to use compression in Hadoop.

[3]`http://en.wikipedia.org/wiki/Scribe_(log_server)`
[4]`http://tinyurl.com/twitter-architecture`

1. Compressing input files If the input file is compressed, then the bytes read in from HDFS is reduced, which means less time to read data. This time conservation is beneficial to the performance of job execution.

   If the input files are compressed, they will be decompressed automatically as they are read by MapReduce, using the filename extension to determine which codec to use. For example, a file ending in .gz can be identified as gzip-compressed file and thus read with GzipCodec.

2. Compressing output files Often we need to store the output as history files. If the amount of output per day is extensive, and we often need to store history results for future use, then these accumulated results will take extensive amount of HDFS space. However, these history files may not be used very frequently, resulting in a waste of HDFS space. Therefore, it is necessary to compress the output before storing on HDFS.

3. Compressing map output Even if your MapReduce application reads and writes uncompressed data, it may benefit from compressing the intermediate output of the map phase. Since the map output is written to disk and transferred across the network to the reducer nodes, by using a fast compressor such as LZO or Snappy, you can get performance gains simply because the volume of data to transfer is reduced.

Presently, there are three major serialization formats: Google's Protocol Buffers[5], Facebook's Thrift (now an Apache project)[6] and the new Apache Avro[7]. All allow exchanging well formatted, compressed data efficiently in a language-independent way.

While all have their particular pros and cons, they are roughly equivalent in functionality. That said, with the benefit of large companies behind them, Protocol Buffers and Thrift are sure to remain in popular use for the foreseeable future and have the large market share at this time.

Avro is covered in Tom White's HGD3.0 [White, 2012], and was made to integrate well with Hadoop from the ground up. In addition, it doesn't require running a code-generation utility before it can be used. It may be the best technical choice for a new project right now, other factors being equal.

We strongly recommend using a serialization framework for any real-world project.

[5]`https://code.google.com/p/protobuf/`
[6]`http://thrift.apache.org/`
[7]`http://avro.apache.org/`

C.4 Conclusion

Good luck with your experiments! I hope you have as much fun as I had working with this technology.

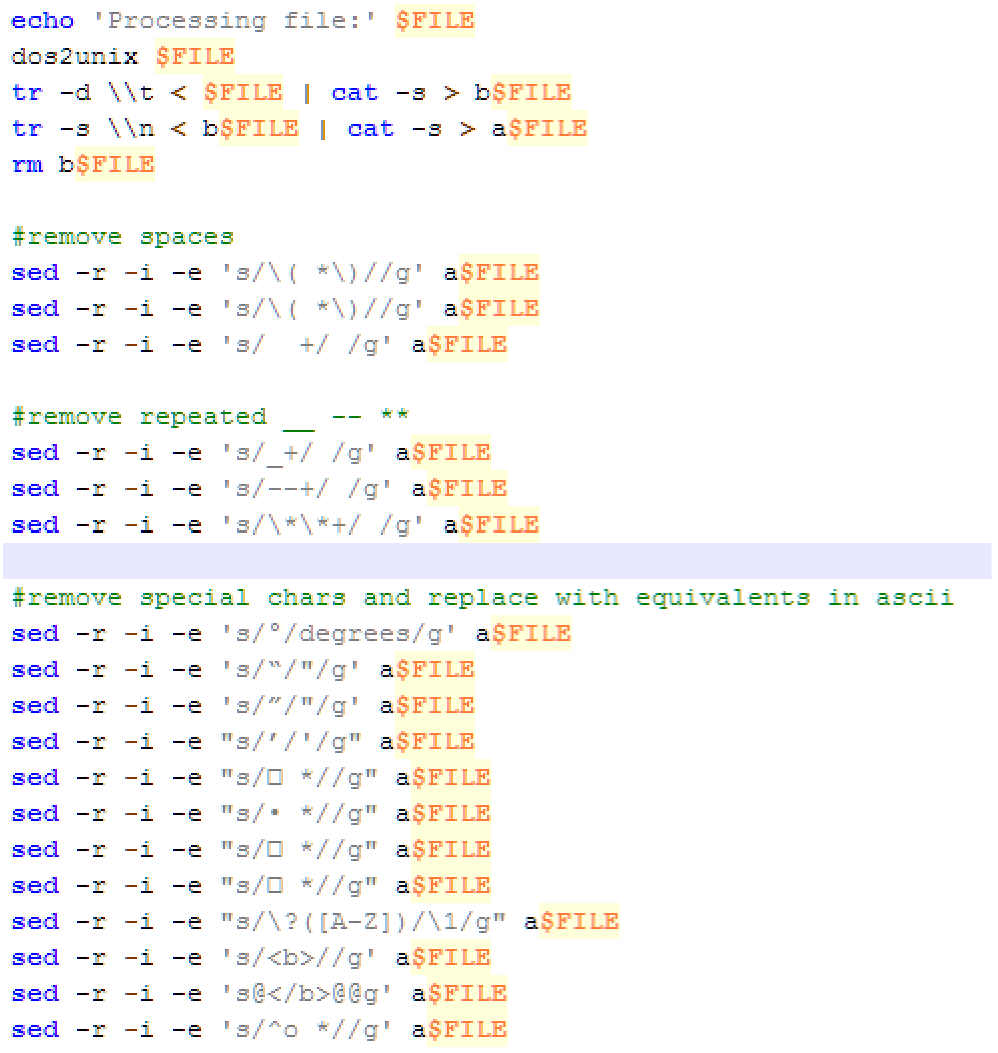
Appendix D

Code Listings

D.1 Language Detection Code

```
#!/usr/bin/env python
#coding:utf-8
# Author: Alejandro Nolla - z0mbiehunt3r
# Purpose: Example for detecting language using a stopwords
# based approach
# Created: 15/05/13
# Modified to detect only French and English
# Mathieu Dumoulin 28/01/14

try:
    from nltk import wordpunct_tokenize
    from nltk.corpus import stopwords

def calculate_languages_ratios(text, \
        languages = ['french', 'english']):
    """
    Calculate probability of given text to be written in French
        or English and return a dictionary that looks like:
        {'french': 2, 'english': 0}

    @param text: Text whose language want to be detected
    @type text: str
```

```
@return: Dictionary with languages and unique stopwords seen
    in analyzed text

@rtype: dict
"""

    languages_ratios = {}

# Compute per language included in nltk number of unique
# stopwords appearing in analyzed text
for language in languages:
    stopwords_set = set(stopwords.words(language))
    words_set = set(words)
    common_elements = words_set.intersection(stopwords_set)

    # language "score"
    languages_ratios[language] = len(common_elements)

return languages_ratios
```



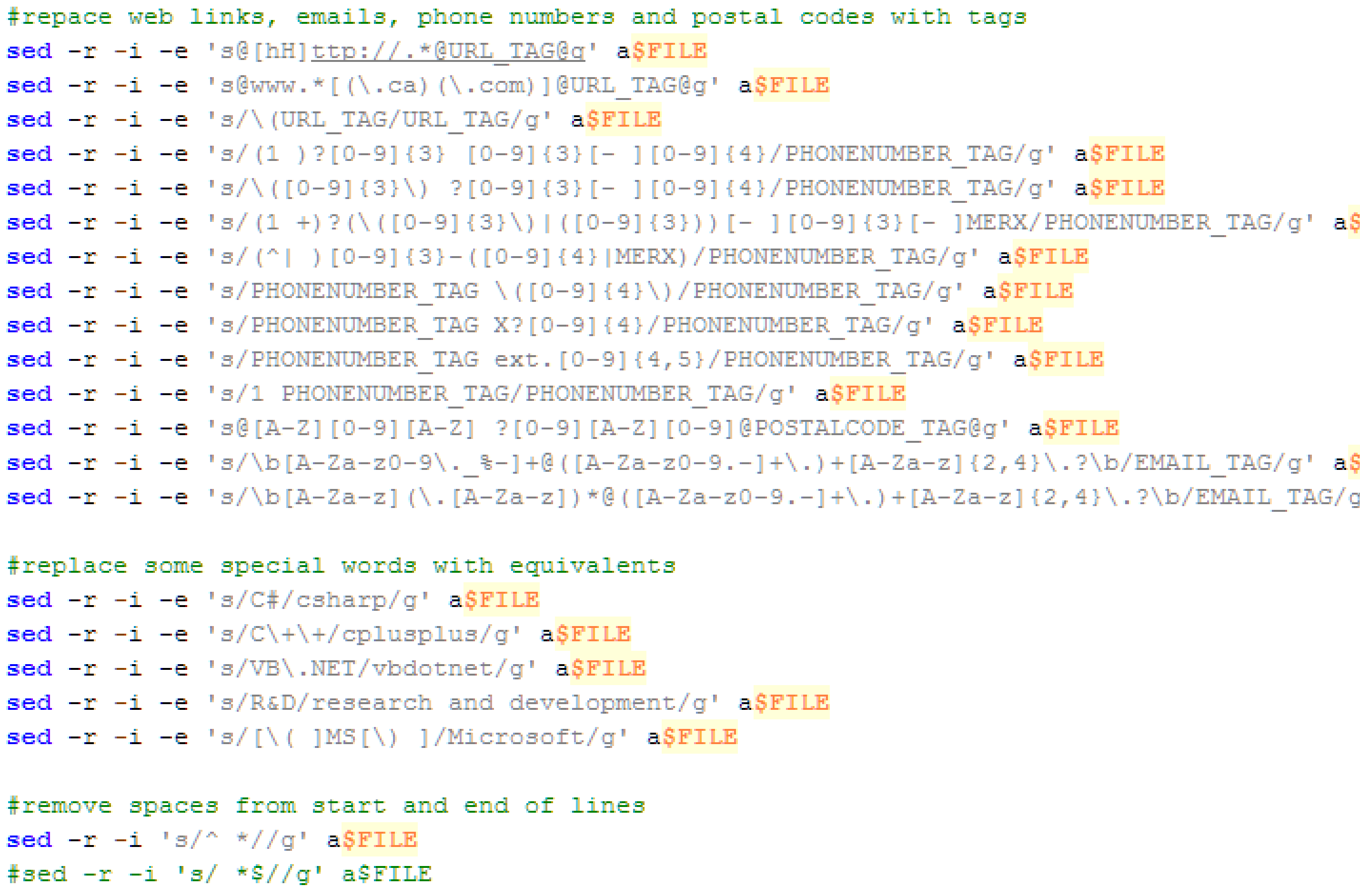
```
echo 'Processing file:' $FILE
dos2unix $FILE
tr -d \\t < $FILE | cat -s > b$FILE
tr -s \\n < b$FILE | cat -s > a$FILE
rm b$FILE

#remove spaces
sed -r -i -e 's/\( *\)//g' a$FILE
sed -r -i -e 's/\( *\)//g' a$FILE
sed -r -i -e 's/  +/ /g' a$FILE

#remove repeated __ -- **
sed -r -i -e 's/_+/ /g' a$FILE
sed -r -i -e 's/--+/ /g' a$FILE
sed -r -i -e 's/\*\*+/ /g' a$FILE

#remove special chars and replace with equivalents in ascii
sed -r -i -e 's/°/degrees/g' a$FILE
sed -r -i -e 's/“/"/g' a$FILE
sed -r -i -e 's/”/"/g' a$FILE
sed -r -i -e "s/’/'/g" a$FILE
sed -r -i -e "s/□ *//g" a$FILE
sed -r -i -e "s/• *//g" a$FILE
sed -r -i -e "s/□ *//g" a$FILE
sed -r -i -e "s/□ *//g" a$FILE
sed -r -i -e "s/\?([A-Z])/\1/g" a$FILE
sed -r -i -e 's/<b>//g' a$FILE
sed -r -i -e 's@</b>@@g' a$FILE
sed -r -i -e 's/^o *//g' a$FILE
```

*[ht]

Figure D.1: Clean and normalize script part one

D.2 Data Cleaning Bash Script

*[ht]



```
#repace web links, emails, phone numbers and postal codes with tags
sed -r -i -e 's@[hH]ttp://.*@URL_TAG@g' a$FILE
sed -r -i -e 's@www.*[(\.ca)(\.com)]@URL_TAG@g' a$FILE
sed -r -i -e 's/\(URL_TAG/URL_TAG/g' a$FILE
sed -r -i -e 's/(1 )?[0-9]{3} [0-9]{3}[- ][0-9]{4}/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/\([0-9]{3}\) ?[0-9]{3}[- ][0-9]{4}/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/(1 +)?(\([0-9]{3}\)|([0-9]{3}))[- ][0-9]{3}[- ]MERX/PHONENUMBER_TAG/g' a$
sed -r -i -e 's/(^| )[0-9]{3}-([0-9]{4}|MERX)/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/PHONENUMBER_TAG \([0-9]{4}\)/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/PHONENUMBER_TAG X?[0-9]{4}/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/PHONENUMBER_TAG ext.[0-9]{4,5}/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's/1 PHONENUMBER_TAG/PHONENUMBER_TAG/g' a$FILE
sed -r -i -e 's@[A-Z][0-9][A-Z] ?[0-9][A-Z][0-9]@POSTALCODE_TAG@g' a$FILE
sed -r -i -e 's/\b[A-Za-z0-9\._%-]+@([A-Za-z0-9.-]+\.)+[A-Za-z]{2,4}\.?\b/EMAIL_TAG/g' a$
sed -r -i -e 's/\b[A-Za-z](\.[A-Za-z])*@([A-Za-z0-9.-]+\.)+[A-Za-z]{2,4}\.?\b/EMAIL_TAG/g

#replace some special words with equivalents
sed -r -i -e 's/C#/csharp/g' a$FILE
sed -r -i -e 's/C\+\+/cplusplus/g' a$FILE
sed -r -i -e 's/VB\.NET/vbdotnet/g' a$FILE
sed -r -i -e 's/R&D/research and development/g' a$FILE
sed -r -i -e 's/[\( ]MS[\) ]/Microsoft/g' a$FILE

#remove spaces from start and end of lines
sed -r -i 's/^ *//g' a$FILE
#sed -r -i 's/ *$//g' a$FILE
```

Figure D.2: Clean and normalize script part two