Comparison of Machine Learning Algorithms Run in MapReduce Versus SQL
By Anthony Hsu
Advisor: Daniel Abadi
CPSC 490: Senior Project, Fall 2011

Abstract

In this paper, we examine the performance of a couple machine learning algorithms run using MapReduce in an HDFS system compared to the same algorithms run using SQL in a database system. Specifically, we look at two relatively new open-source machine learning libraries – Mahout, which is built on top of Hadoop, and is thus MapReduced-based, and MADlib, a collection of various SQL modules that allow machine learning algorithms to be performed in databases, such as PostgreSQL. We discuss the implementation details and the tradeoffs of each approach.

1. Introduction

Since Edgar Codd published his seminal 1970 paper [3] on a relational database model, this has become the most popular database design. After its publication, IBM, where Codd was working, soon began working to implement a system based on the relational model. The result was System R [2], which also introduced the SQL language for accessing and manipulating data stored in a relational database system. Over the past few decades, the popularity of the relational model and SQL grew, and both are now mainstream.

However, in 2004, researchers at Google published a paper on MapReduce [6], a new approach to data storage and manipulation in a distributed file system that is designed to be fault-tolerant and scalable. Since the publication, the Apache Software Foundation has implemented an open-source version of MapReduce called Hadoop [7]. Hadoop uses its own block-structured distributed file system known as the Hadoop Distributed File System (HDFS). Files must be accessed and manipulated through the Hadoop framework using Java.

Since 2004, Hadoop and the MapReduce paradigm have become quite popular. Many companies, such as Google, Facebook, and Yahoo!, now use MapReduce to process and analyze enormous volumes of data. Nonetheless, most companies still use a traditional relational databases and SQL to store and analyze their data, perhaps with a business intelligence interface layer between the database and the analysts.

With the huge growth of data, companies must find new ways to process and harness the information to improve business operations. One such method is to use machine learning algorithms that can be trained on the existing data and can then make predictions about future data or recommendations to users.

This project looks at two relatively new machine learning libraries: (1) Apache Mahout [10], which is a MapReduce library, and (2) MADlib [9], which is a SQL library. They are designed to run on two very different systems. Mahout runs on a Hadoop system using MapReduce, while MADlib runs in a traditional database system (such as PostgreSQL [11], which is used in this project) using SQL.

In this project, we looked closely at the implementations and performance of two machine learning algorithms – Bayesian classification and k-means clustering – in these two systems: Mahout + Hadoop and MADlib + PostgreSQL. Though not all of our experiments succeeded, we were able to gain some insight into the differences and tradeoffs of the two systems and their implementations of these algorithms.

The structure of our paper is as follows. We first present some background about the systems used
in Section 2. Then, we discuss the algorithms and their implementations in Section 3. In Section 4, we discuss our experimental setup and present our results. In Section 5, we discuss what remains to be done and areas for future work. We finish with some concluding remarks in Section 6 and acknowledgments in Section 7. Appendix A contains a detailed report about the difficulties faced in this project.

2. Background

In this project, our goal was to compare the performance of several machine learning algorithms run in a MapReduce system against the same algorithms run in SQL system. We utilized two machine learning libraries, Apache Mahout (MapReduce) and MADlib (SQL). The systems we ran these algorithms on were Apache Hadoop and PostgreSQL, respectively. The libraries and the storage systems used are all open-source software (links can be found in the References section).

2.1 Hadoop

Apache Hadoop [7] is a MapReduce framework written in Java. This framework assures you scalability and fault tolerance by handling load distribution as well as recovery and redistribution of jobs in the case of failures. This is well suited for heterogeneous environments with many nodes, where processes may very likely run at different speeds and there is a significant probability that during any execution, a node may fail. However, data is stored in large chunks (default is 64 MB) known as blocks, and relational information about the data is not captured or reflected by the file system.

2.2 PostgreSQL

PostgreSQL [11] is a relational database originally developed by Michael Stonebraker at UC Berkeley in the mid-1980s. In contrast to the MapReduce framework, a database system like PostgreSQL provides the notion of a table, with rows and columns, in which data is stored. These tables can be sorted (indexed) on a column and compressed in order to speed-up search times and to minimize disk I/O. Many queries involve combining data from multiple tables (known as JOIN operations), and decades of research have been spent optimizing JOINs between tables and carefully constructing a query execution plan in order to minimize the amount of disk I/O and make the query execution as efficient as possible. Much research has also focused on optimizing table storage, compression, indexes, and cached table snapshots (materialized views) in order to further improve performance. As a result of all this research, database systems have become highly optimized for fast query execution. However, less research has been done on parallel database systems, so they may not handle failures and parallelization of queries as well as MapReduce systems.

2.3 Mahout

Mahout [10] is a collection of machine learning algorithms written to run inside Hadoop’s MapReduce framework. Each algorithm has its own driver, mapper, reducer, combiner, and possibly input and output format Java classes. In addition, Mahout provides many utility classes, such as for converting text files to sequence files (compressed binary files stored in HDFS), for converting sequence files to sparse vector files (compressed vector representation), and for dumping the contents of sequence and sparse files as text to the console or an output file. Since Mahout uses the MapReduce paradigm, it is quite flexible and allows one to write custom mapper and input/output format classes that can process many different kinds of data and data formats.
2.4 MADlib

MADlib [9] is a collection of machine learning algorithms designed for use within a SQL environment, such as PostgreSQL or Greenplum. It is based on the ideas published in a paper by researchers from Greenplum, Fox Audience Network, Evergreen Technologies, and U.C. Berkeley [4]. In addition to providing many custom SQL functions, MADlib also utilizes PL/Python and some linear algebra libraries to assist with the logic and computation involved in various machine learning libraries. When we began this project, MADlib was using the LAPACK/BLAS linear algebra libraries. However, during the course of the project, MADlib migrated to using the Eigen linear algebra library.

3. Theory

In this project, the machine learning algorithms we focused on were (1) Bayesian classification and (2) k-means clustering. Bayesian classification is an example of a supervised algorithm, where a model is trained on pre-classified data before being applied to some unclassified test data. K-means clustering is an example of an unsupervised algorithm, where the goal is to discover structure among unlabeled data. In the following subsections, we give a more detailed overview of each algorithm.

3.1 Bayesian Classification

The goal of Bayesian classification is to classify data into one of several predetermined groups. The idea is that elements of each group have certain characteristic features, so the more features of a certain group a datum has, the more likely it is to belong to that group. By training the model on some data that has already been classified, the model is able to calculate the probabilities of various features occurring within each group. Then, when tested on some new data, the Bayesian classifier will categorize the data in the group for which it has the greatest probability. The Bayesian classifiers we used were both naïve, which means that all the features of a group are treated as independent variables.

Mahout and MADlib take different approaches in implementing Bayesian classification. Mahout’s classifier is geared toward classifying documents into categories based on the words in the documents, whereas MADlib expects tuples as input, where each element in each tuple represents a distinct feature. As a result of the differences in implementation, we were unable to compare the performance of the two systems on the same data set. In the following subsections, we go into detail about the Bayesian classifier implementations in the two libraries.

3.1.1 Mahout Bayesian Classifier

Mahout’s Bayesian classifier expects a training corpus of pre-classified documents, from which it calculates the term frequency-inverse document frequency (tf-idf) score of each word in each category. It then uses the tf-idf scores to assign weights to every word, and uses these weights to categorize new, unclassified documents by placing them in the category for which the sum of the weights of the words is highest.

The Mahout implementation follows that described in a paper by Rennie et al. from the MIT Artificial Intelligence Laboratory [12]. This implementation uses some weight normalization to improve the performance of the Bayesian classifier. The term frequency (TF), which is the number of occurrences of a word divided by total words in a document, is divided by the root mean square of all the TFs in that document to get the normalized frequency (NF) for a term. The sum of the NFs of a term in all the documents in a category becomes the weight normalized term frequency (WNTF) for a
word for that category. The inverse document frequency (IDF) of a term in a category is the logarithm of the quotient of the total number of documents and the number of documents containing the term:

\[
\text{IDF} = \log \frac{|D|}{1 + |\{d: t \in d\}|}
\]

The addition of one in the denominator is added to prevent division by zero in case a term does not appear in any documents. The WNTF and the IDF of a term are then multiplied to get the weight normalized tf-idf (WNTF-IDF) of a term. The weight of a term \(t\) for a category is then:

\[
\text{weight}_t = \log \frac{\text{WNTF-IDF}_t + \alpha_t}{\sum_t (\text{WNTF-IDF}_t + \alpha_t)}
\]

where \(\alpha_t\) is a smoothing factor to prevent trying to take the logarithm of 0 for terms that do not occur in a category (and would thus have a WNTF-IDF of 0). Mahout uses a smoothing factor \(\alpha_t\) of 1, which is known as Laplacian smoothing and is a typical value. The reason that a logarithm is taken is due to underflow concerns. The probability of each word is very small, and if we were to multiply probabilities together, almost all our calculations would underflow to 0. Thus, we take logarithms of the probabilities and add weights together. When classifying a new document, we place the document in the category for which the sum of the weights of the words is largest.

### 3.1.2 MADlib Bayesian Classifier

The MADlib implementation of a Bayesian classifier is much simpler than the implementation in Mahout. It expects as training input a table of data points, where each datum (tuple) consists of a class (category) and an ordered array of attributes (features). (By ordered, we mean that if there are \(n\) distinct features, all the data points must list the features in the same order.) From this training table, the prior probabilities of each category and the feature probabilities for each category are calculated. The prior probability of a category is just the number of training data belonging to that category divided by the total number of training data. The probability of a feature \(f\) for a category \(c\) is just the relative frequency of that feature in the training data:

\[
P(F_i = f | C = c) = \frac{\#(c, i, f) + \alpha}{\#c + \alpha \cdot \#i}
\]

where \(\#(c, i, f)\) is the number of training samples where the category was \(c\) and the \(i^{th}\) feature was \(f\) and \(\#c\) is the number of training samples where the category was \(c\). \(\alpha\) is a smoothing factor, and \(\#i\) is the number of distinct values for feature \(i\) in all categories. Just like the Mahout implementation, MADlib uses a smoothing factor \(\alpha\) of 1, which is Laplacian smoothing. A new datum point is then classified in the category for which the product of the feature probabilities is the highest. Again, like the Mahout implementation, rather than multiplying probabilities directly, logarithms of the probabilities are taken and the logarithms are summed to avoid underflow errors.

### 3.2 K-means Clustering

The goal of k-means clustering is to divide some unlabeled data into \(k\) groups or clusters. The idea behind this algorithm is to initially choose \(k\) random points, group each datum with the random point it is closest to, shift the random point to the mean (or centroid) of all the data points in the group, and then continue re-iterating this procedure until the means converge. The only parameters that must be specified are \(k\), the number of desired clusters, and a definition of distance between data points.

#### 3.2.1 Mahout K-means Clusterer

The Mahout k-means clusterer chooses \(k\) random points as initial cluster centers. By default, the
distance measure is the squared Euclidean distance. The Euclidean distance between two points \((x_1, x_2, \ldots, x_n)\) and \((y_1, y_2, \ldots, y_n)\) is the standard \(\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}\). The reason the square of the Euclidean distance is used is to save on computation. We do not care about the actual value of the distance; we just want to compare points and find out which point is closest. Hence, the square root is unnecessary.

In addition to specifying the number of initial clusters, one can also specify the number of iterations to run the algorithm for and a convergence delta, which is used to determine if the algorithm has converged. If, from one iteration to the next, the cluster centers have not shifted by more than the convergence delta, then convergence is declared and the algorithm stops.

### 3.2.2 MADlib K-means Clusterer

The MADlib k-means clusterer uses the k-means++ algorithm [8] to choose the initial cluster centers. By using this algorithm to choose initial cluster centers, the convergence time and error of the final clusters are reduced. The algorithm is iterated until the clusters converge (which is defined as when the number of reassigned nodes between iterations is less than some percentage (default is 0.1%) of all nodes) or until the maximum number of iterations is reached (default is 20).

The MADlib implementation also has a goodness parameter. If enabled, the “goodness of fit” (GOF) of the final clusters will be calculated. The metric used is the sum of the distances between each point and the nearest cluster center divided by the total number of points.

### 4. Experiment

Our experiments were run on a Dell Precision T3500 desktop with eight 2.67 GHz Intel Xeon X5550 processors and 11.8 GB of RAM. Due to disk usage limits, all the data were stored on a 500 GB external Seagate hard drive. We used Hadoop version 0.20.203.0, Mahout version 0.5, PostgreSQL version 9.1.2, and an unreleased version of MADlib [13] provided by its developers.

The Mahout library provided some example scripts for Bayesian classification and k-means clustering which downloaded some data sets for running the machine learning algorithms on. The plan was to also use these data sets as input into MADlib’s implementations of the algorithms. However, due to implementation differences, the data set used for the Mahout Bayesian classifier was incompatible with the MADlib classifier. On the other hand, the data set used for the Mahout k-means clusterer was adaptable for the MADlib clusterer. In the following subsections, we go into detail about our experiments.

#### 4.1 Bayesian Classification

The Mahout library provides an example script, which we modified to use our external hard drive for storage\(^1\), that performs Bayesian classification on a data set consisting of emails from twenty Usenet newsgroup mailing lists, such as comp.windows.x, rec.motorcycles, and talk.politics.guns. First, the training data is processed and weights of all the words for each category are calculated, as described in Section 3.1.1, and stored in a sequence file. The weights are then used to categorize the test data. The results of running the Bayesian classifier on some training and test data generated from the twenty Usenet newsgroups data set are shown in Figure 1\(^2\). The total execution time for loading, training, and

---

\(^1\) Available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/build-20news-bayes-externalHD.sh](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/build-20news-bayes-externalHD.sh)

\(^2\) For full output details, see [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MahoutBayesOutput.txt](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MahoutBayesOutput.txt)
**Summary**

| Correctly Classified Instances | : 6018 | 79.8991% |
| Incorrectly Classified Instances | : 1514 | 20.1009% |
| Total Classified Instances | : 7532 |

**Confusion Matrix**

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | ---Classified as |
| 383 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | a | rec.sport.baseball |
| 4 | 370 | 0 | 1 | 4 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 3 | 0 | 0 | 1 | 7 | 0 | 2 | 0 | | 396 | b | sci.crypt |
| 9 | 2 | 382 | 0 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | | | | 389 | c | rec.sport.hockey |
| 2 | 12 | 0 | 326 | 1 | 5 | 0 | 2 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 4 | 2 | 0 | 1 | 5 | | | | 364 | d | talk.politics.guns |
| 0 | 0 | 1 | 0 | 372 | 1 | 2 | 1 | 0 | 2 | 0 | 0 | 1 | 0 | 3 | 2 | 0 | 3 | 0 | 6 | | | | 398 | e | soc.religion.christian |
| 2 | 14 | 0 | 0 | 324 | 5 | 8 | 0 | 0 | 0 | 1 | 10 | 5 | 2 | 6 | 4 | 8 | 0 | 4 | 0 | | | | 393 | f | sci.electronics |
| 6 | 9 | 0 | 0 | 1 | 2 | 256 | 3 | 0 | 0 | 5 | 0 | 50 | 8 | 5 | 8 | 1 | 40 | 0 | 0 | | | | 394 | g | comp.os.ms-windows.misc |
| 1 | 0 | 1 | 0 | 6 | 2 | 342 | 0 | 0 | 0 | 12 | 7 | 1 | 4 | 9 | 3 | 0 | 2 | 0 | | | | 390 | h | misc.forsale |
| 8 | 6 | 2 | 32 | 102 | 0 | 0 | 0 | 25 | 29 | 0 | 4 | 0 | 0 | 13 | 7 | 7 | 6 | 0 | 10 | | | 251 | i | talk.religion.misc |
| 6 | 14 | 2 | 10 | 86 | 1 | 1 | 0 | 1 | 152 | 0 | 4 | 2 | 2 | 6 | 10 | 5 | 2 | 0 | 15 | | | | 319 | j | alt.atheism |
| 1 | 6 | 0 | 0 | 0 | 4 | 9 | 4 | 0 | 0 | 283 | 0 | 11 | 6 | 3 | 1 | 0 | 66 | 0 | 1 | | | | 395 | k | comp.windows.x |
| 5 | 7 | 1 | 2 | 12 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 338 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 1 | 0 | | 376 | l | talk.politics.mideast |
| 1 | 1 | 0 | 0 | 0 | 25 | 18 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 291 | 27 | 2 | 0 | 3 | 11 | 0 | 0 | 0 | 392 | m | comp.sys.ibm.pc.hardware |
| 3 | 2 | 0 | 0 | 0 | 12 | 7 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 329 | 5 | 2 | 4 | 5 | 0 | 0 | 0 | | 385 | n | comp.sys.mac.hardware |
| 1 | 2 | 0 | 0 | 0 | 14 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 369 | 0 | 0 | 9 | 0 | 3 | 0 | | | 394 | o | sci.space |
| 1 | 0 | 0 | 0 | 0 | 2 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 381 | 9 | 0 | 0 | 1 | 0 | | | 398 | p | rec.motorcycles |
| 2 | 0 | 2 | 0 | 0 | 6 | 0 | 11 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 10 | 359 | 1 | 0 | 1 | 0 | | | | 396 | q | rec.autos |
| 6 | 13 | 0 | 0 | 0 | 14 | 8 | 7 | 0 | 0 | 11 | 0 | 11 | 10 | 7 | 0 | 3 | 297 | 0 | 2 | 0 | | | | 389 | r | comp.graphics |
| 12 | 30 | 4 | 106 | 11 | 3 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 18 | 9 | 5 | 1 | 97 | 11 | 0 | | | 310 | s | talk.politics.misc |
| 3 | 3 | 3 | 1 | 6 | 12 | 0 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 4 | 4 | 2 | 7 | 0 | 342 | 0 | | 396 | t | sci.med |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | u | unknown |

**Figure 1:** Results of training and testing the Mahout Bayesian classifier on documents from twenty Usenet newsgroups.
testing was about 212 seconds.

We were unable to run the MADlib Bayesian classifier on the Usenet newsgroups data set due to a different implementation approach. As mentioned in Section 3.1.2, MADlib expects every datum in the training and testing sets to be of the same length, with the features (words, in our case) listed in the same order. However, it does not allow features to be omitted or set to NULL. You also cannot use dummy values for omitted features because MADlib will treat them as real features, which would throw off the calculated probabilities. The MADlib Bayesian classifier needs to be rewritten or a new module needs to be added in order for MADlib to be able to classify text documents.

### 4.2 K-means Clustering

The Mahout library provides an example k-means clustering script, which we modified to use our external hard drive for storage³, that operates on a collection of 21,578 old Reuters news articles, which are all in Standard Generalized Markup Language (SGML) format.

The script does some substantial preprocessing before actually running the k-means algorithm. First, an Apache Lucene (a text search engine library) class is used to parse the news articles in SGML format based on a Document Type Definition (DTD) file and outputs the news articles as individual text files. Then, these are converted into sequence files using Mahout’s seqdirectory utility, which also adds key values to each news article. During this processing, the 21,758 articles are reduced to 15,967 articles, perhaps due to articles with markup errors. Then, Mahout’s seq2sparse utility is used to convert the sequence files into a sparse vectorized format, where indices and values are only stored for non-zero values (\(\{i_1:v_1,i_2:v_2,\ldots,i_k:v_k\}\)). This conversion includes calculating the tf-idf scores for all the words and mapping each of the 27,312 different words to a unique value, which is its index in the tf-idf vectors. Each news article is then mapped to a sparse tf-idf vector containing the tf-idf scores of all the words in the article. After the preprocessing finishes, then the k-means algorithm (as described in Section 3.2.1) is run on the set of sparse tf-idf vectors generated during preprocessing.

We configured the algorithm to run for a maximum 10 iterations with \(k = 20\) clusters and a convergence delta of 0.5. After 10 iterations, 18 of the 20 clusters had converged, containing 15,367 of the 15,967 documents. However, 10 of the 18 converged clusters had one document in them, 14 of them had fewer than 100 documents, and the two largest clusters had 9,260 and 4,425 documents. The top 20 words with the highest weights in the two largest clusters are shown in Figure 2⁴. The preprocessing took about 65 seconds, and the actual k-means algorithm took about 75 seconds, for a total time of about 141 seconds.

[³ Available at http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.sh59/code/build-reuters-externalHD.sh](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.sh59/code/build-reuters-externalHD.sh)

[⁴ To see the top 20 words in the other 16 converged clusters and for full output details, see http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.sh59/code/MahoutKmeansOutput.txt](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.sh59/code/MahoutKmeansOutput.txt)
99.9% of all values were looked up in a dictionary file format terms, a run all documents, and the goodness of fit (as explained in Section 3.2.2) was about 44.6. In all executions, one of the twenty clusters contained over 99.9% of all documents, and the goodness of fit (as explained in Section 3.2.2) was about 44.6. To find the top terms, a run-length encoded centroid vector was converted back to the Mahout sparse vector format (which stores index:value for non-zero values), and then the words corresponding to the largest values were looked up in a dictionary file. The top 20 terms in the largest cluster (the one with over 99.9% of all documents) are shown in Figure 3.

Figure 2: Top 20 terms in the two largest converged clusters after running the Mahout k-means clusterer on some test data generated from a data set of old Reuters news articles.

The MADlib implementation expects as input a series of vector points, each with an optional id. For efficiency, MADlib provides a sparse vector data type, which uses run-length encoding to compress the data. A sparse vector is expressed in the form:

\[ \{ n_1, n_2, \ldots, n_k \}; \{ v_1, v_2, \ldots, v_k \} \]

which represents a vector of \( n_1 \) occurrences of \( v_1 \), followed by \( n_2 \) occurrences of \( v_2 \), etc.

In our experiment, we took the sparse tf-idf vector representations of the Reuters documents generated by the preprocessing step in the Mahout k-means clustering script, and converted them to MADlib sparse vectors using a Python script we wrote. After loading these into a PostgreSQL table, we ran the k-means algorithm. We configured the algorithm to run for a maximum of 10 iterations (same as for our Mahout experiment) with \( k = 20 \) clusters and with a convergence parameter of \( 0.00001 \), which means that if between consecutive iterations, less than 0.001% of the nodes changed clusters, the clusters are deemed to have converged and the algorithm stops. Since our data set had only 15,967 documents, this convergence criterion is only met when no documents change clusters between iterations.

We ran the MADlib k-means clusterer four times. Three of the four times, the clusters converged after only two iterations in a little over 8 minutes, and the fourth time, they converged after three iterations in about 13.5 minutes. In all executions, one of the twenty clusters contained over 99.9% of all documents, and the goodness of fit (as explained in Section 3.2.2) was about 44.6. To find the top terms, a run-length encoded centroid vector was converted back to the Mahout sparse vector format (which stores index:value for non-zero values), and then the words corresponding to the largest values were looked up in a dictionary file. The top 20 terms in the largest cluster (the one with over 99.9% of all documents) are shown in Figure 3.

---

5 Available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MahoutSparseToMadlibSparse.py](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MahoutSparseToMadlibSparse.py)

6 PostgreSQL output from one of the executions is available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/madlibKmeansOutput.txt](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/madlibKmeansOutput.txt)

7 Python script available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MadlibSparseToMahoutSparseWithIDs.py](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MadlibSparseToMahoutSparseWithIDs.py)

8 Python script available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/printTopTerms.py](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/printTopTerms.py), dictionary (adapted from the dictionary generated by the Mahout seq2sparse utility) available at [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/dictionary.txt](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/dictionary.txt)

9 To see the top 20 terms of all 20 clusters, see [http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MadlibTopTerms.txt](http://zoo.cs.yale.edu/classes/cs490/11-12a/hsu.anthony.ah59/code/MadlibTopTerms.txt)
Cluster 1 Top Terms:
qant => 1.64697029389358
lightning => 1.24154631032648
customers => 1.11176591632388
255,568 => 1.0435880900147
mystified => 1.01603207464994
prepay => 0.992950967140753
highland => 0.960318110734654
europeans => 0.955134456943841
tananbaum => 0.878264365733888
keidanren => 0.871583920357653
trained => 0.86595514094861
forstmann => 0.78478637743995
franchise => 0.74230438378888
stephen => 0.730444566736732
assemblies => 0.729159285055414
collins => 0.726258385156009
tosco => 0.711660096744933
greenspan => 0.706025432048172
caspian => 0.704676966019471
thermal => 0.66701458239965

Figure 3: Top 20 terms in largest cluster after running the MADlib k-means clusterer.

Most of the top 20 terms (the words with the highest weights) seem like obscure, uncommon words, and there is no overlap with the top 20 terms in the two largest clusters generated by the Mahout k-means clusterer. Thus, it is possible that we calculated the top terms incorrectly or that the Mahout implementation does some additional computation or normalization that we were not aware of.

The MADlib execution time only includes the time required to run the k-means algorithm and does not include the preprocessing time required to calculate the tf-idf vectors (since we used the vectors calculated by Mahout). The MADlib execution also terminated within two to three iterations, whereas the Mahout execution iterated the maximum 10 times allowed. Yet, MADlib’s execution time was still three to five times slower than Mahout’s execution time, which includes preprocessing. Part of the difference is due to Mahout storing all data and intermediate files in a compressed sparse vector format, which is probably more efficient than the run-length encoded tables that MADlib uses. However, this alone cannot explain the significant execution time differences. As future work, we should take a closer look at the implementation details to figure out where most of the execution time is being spent.

5. Future Work

In this project, due to implementation differences (as explained in Sections 3.1.2 and 4.1), we were unable to run the MADlib implementation of Bayesian classification on the Usenet data. As future work, one could enhance the MADlib Bayesian classifier to allow it to do text classification.

In our k-means experiment, we used the tf-idf vectors generated by Mahout. However, MADlib does provide some functions, such as svec_sfv, which can create sparse feature vectors from a table of documents, that can be used to assist you in calculating tf-idf vectors. For our Mahout experiment, calculating the tf-idf vectors took nearly as long as running the actually k-means algorithm. It would be interesting to see how long MADlib would take to calculate the tf-idf vectors.

For k-means clustering, both Mahout and MADlib generated one or two very large clusters, with the rest usually having only one or two documents. It is possible that this is due to the curse of dimensionality or having too small of a sample size. We should also test the clusterers on the Usenet 20 newsgroups data set to see if they are able to cluster emails from each newsgroup together.

For this project, we ran all our experiments on a single node. In the future, we should also configure these systems to run in a distributed system with many nodes. This would give much more insight into the scalability of these machine learning libraries and algorithms.

It would also be interesting to test these machine learning libraries with different systems. Mahout

---

10 A more detailed explanation is provided in the sparse vector documentation: http://devdoc.madlib.net/v0.2beta/group__grp__svec.html
could be tested with HadoopDB [1], and MADlib could be tested with Greenplum. One could also
load the data into statistical software and run the algorithms there to see how the load and execution
times compare.

Finally, a broader range of algorithms could be tested in the future.

6. Conclusion

In this paper, we explored two different approaches to running machine learning algorithms: (1) using
MapReduce inside the Hadoop framework, and (2) using SQL inside a traditional relational database
system. We were able to run the Mahout Bayesian classifier on a Usenet newsgroups data set and the
Mahout k-means clusterer on a Reuters data set. Unfortunately, due to implementation differences, we
were unable to run the MADlib Bayesian classifier on the Usenet data set. However, we were able to
run the MADlib k-means clusterer on the Reuters data set.

As this project has shown, there are many challenges and complexities to implementing machine
learning algorithms that run inside the database. Even for relatively simple algorithms, there is much
preprocessing and data manipulation that first needs to be done before the algorithms can be run. This
requires developing a rich library of functions, scripts, and modules.

Ideally, machine learning algorithms would be integrated into your data warehouse or business
intelligence tool, allowing you to perform more sophisticated analyses in-database without having to
export your data to another program. Mahout and MADlib are promising first steps toward developing
such integrated, flexible machine learning libraries.

7. Acknowledgments

I would like to thank Daniel Abadi for serving as my advisor for this project. My sophomore year, he
taught a class on database systems and architectures that piqued my interest in databases and inspired
me to take on a databases-related project. At the beginning of the semester, he also discussed many
potential project ideas with me and helped me narrow my choices down to this project. During the
course of the semester, Professor Abadi also directed me to many useful web resources and papers that
assisted me with this project.

I would also like to thank Florian Schoppmann, one of the MADlib developers, for providing me
with a new, custom, unreleased version of MADlib [13] that removed the dependencies that were
causing me installation errors.

8. References

[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin,
"HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical


[3] Edgar F. Codd, "A Relational Model of Data for Large Shared Data Banks," Communications of


Appendix

A. Difficulties

We were able to install Hadoop and Mahout without many problems. Originally, we stored our data directly on the hard drive of the Dell Precision T3500 that provided the processors and memory. However, we quickly exceeded our disk quota. Consequently, we had to store all our data on an external hard drive and rewrite our scripts to accommodate this.

PostgreSQL, which has been widely tested and developed for several decades, also installed without any major issues. The problem was installing MADlib. MADlib requires the PL/Python server scripting language, so we had to reinstall PostgreSQL with this feature enabled. MADlib also depends on the LAPACK and BLAS linear-algebra libraries. However, the versions of these libraries installed on our machines were out-of-date and caused an “undefined symbol: cgemv_” installation error. Despite downloading the latest version of the LAPACK/BLAS libraries, and adding the path where the new versions of these libraries were stored to the LD_LIBRARY_PATH environment variable used by the dynamic linker, we continued to get the same undefined symbol error. We also tried using different versions of PostgreSQL as well as the head version of MADlib, but every combination yielded the same error.

After several days of trying various other solutions, to no avail, we emailed the MADlib developers list (madlib-dev-forum@googlegroups.com) for help. Developers Caleb Welton and Florian Schoppmann responded to our emails and offered several suggestions and possible solutions to our problems. However, despite trying all their suggestions, the error remained. Some of their suggestions were also not possible for us to execute because we did not have root privileges on our machine.

Finally, after a week and a half of correspondence, Florian Schoppmann emailed me a link to a new, custom version of MADlib [13] in which he had removed all the LAPACK/BLAS dependencies, replacing them with the Eigen C++ template library. Using this version, we were finally able to install MADlib successfully. However, this occurred on the day I was leaving campus. We did not want to leave our external hard drive plugged into a school computer over break, for fear of it being stolen, so we had to come up with an alternative plan to run our MADlib experiments.

After getting home, we installed Ubuntu on a desktop, with the plan of installing PostgreSQL and MADlib on this machine and running our experiments there. However, during installation, we ran into many dependency issues again, and we were unable to get the system working. We then decided to just remotely run our experiments on the school machines. To avoid disk quota issues, we deleted many
files and were prepared to reduce the data set size if necessary.

Finally ready to perform our experiments, we delved into the implementation details of the machine learning algorithms in Mahout and MADlib. At a high level, Bayesian classification and k-means clustering are fairly simple and straightforward algorithms. However, when poring into the implementation details, it was often difficult to understand how exactly the algorithms were implemented, since there are many variations to these algorithms. One particularly confusing point was understanding how the k-means algorithm works with documents of text. The k-means algorithm requires a distance measure, and there did not seem like an obvious way to compute the distance between two documents. It was only after several days of struggling through the code and reading through documentation that was often incomplete or unclear that we realized the tf-idf scores of the words in the documents are used to calculate distances. We also realized that in its current implementation, MADlib’s Bayesian classifier would not work for document classification.

Fortunately, after understanding the k-means implementations, we were able to write some Python scripts to adapt the Reuters data set for MADlib and run the MADlib k-means clusterer on the data.