Predictive Analytics On the Cloud

Master's Project submitted to
The Department of Computer Science at
Montclair State University

In Partial fulfillment of the requirements for
a degree in Master of Science in Computer
Science

Klavdiya Hammond
Faculty Advisor: Prof. Aparna Varde
May 10, 2013

Table of Contents

Abstract

This project studies Hadoop and MapReduce technologies available over the cloud for large-scale data processing and predictive analytics. Although some studies have shown that technologies based on the MapReduce framework may not perform as well as parallel database management systems, especially with ad hoc queries and interactive applications, MapReduce has been adopted by many organizations for big data storage and analytics. A number of MapReduce tools are broadly available over the cloud. In this work we explore Apache Hive data warehousing solution and Mahout Data mining libraries for predictive analytics over the cloud.

1 Introduction

Advances in many technologies as well as continued reduction in data storage and processing costs have led to data explosion, including human data in the form of emails, photos, messages, blogs and tweets on social media and digital data generated by sensors, such as telescopes, cameras and GPS to name a few. The available data are big in volume and complex in terms of the number of data sources and interrelationships, making it difficult to store, query, share and analyze. Distributed File System and MapReduce programming model originally introduced by Google have been proposed and became very popular as the technology for big data storage and processing. MapReduce is a programming model introduced and successfully used at Google in order to perform various computations, such as inverted indices and Web crawl data summaries over large volumes of data distributed across thousands of machines [2].

MapReduce takes a programming operation and applies it in parallel to gigabytes and terabytes of data. Apache Hadoop quickly became one of the most popular open-source implementations of the MapReduce framework and distributed file system (HDFS). The framework is designed to automatically divide applications into small fragments of work, each of which can be executed on any node in the cluster and to handle node failures gracefully [3].

Apache Hadoop is not a substitute for a database. Hadoop stores data in files without indices; to find something, a MapReduce job is needed which usually takes a brute force approach of scanning the entire dataset. It works great where a dataset is too large for a database and the cost of regenerating indices becomes prohibitive. A number of institutions and organizations extensively use Hadoop for educational and production purposes [4]. Because Hadoop is not a database system, we use Hive, which was originally developed at Facebook, and then open-sourced to store and pre-process data for data mining and predictive analytics [8]. Hive as an Apache open-source project, is a data warehouse system for Hadoop. It allows users to apply table definitions and structure on top of existing data files stored either directly in HDFS on in other data storage systems and to further query the data in HiveQL, a SQL-like language. Hive queries are executed in MapReduce [6].

Mahout is an open source machine learning Java library from Apache implementing recommender, clustering and classification algorithms with some portions build upon Apache Hadoop framework.

Cloud computing has been proposed as a new way to operate business on the web. Computing service providers offer on-demand network access to configurable computing resources, data management, analytics and knowledge discovery; and Apache Hadoop, Hive and Mahout are available as cloud technologies.

The work described in [17] considers the latest developments in cloud computing and lists the following advantages of cloud services: (1) no initial investment into infrastructure is required, companies pay for what they use, which leads to cost savings; (2) large institutions requiring a cluster of machines for their analytical applications may also benefit in terms of energy–efficiency when deploying applications over the cloud; and (3) the ability to share documents worldwide supports globalization and collaboration across multiple locations. Some of the other reasons why an institution may choose to use cloud technologies[1], specifically Apache Hadoop over the cloud, are:

- Hadoop clusters are not a place to learn Unix/Linux system administration [3];
- Minimal installation and configuration required, for example, Amazon Web Services (AWS) offers machine images with Hive, cmd3s, maven already installed on EC2, thus requiring minimal configuration before users can start developing or running their applications;
- Hadoop, Hive and Mahout are open-source and available on the cloud.

More recently, research in data analytics is focused on the area of predictive analysis on large datasets. The goal of this work is to investigate the use of cloud technologies for the storage, data mining and predictive analysis of data over the cloud. We present a case study on how, using technologies available via cloud computing, large-scale data can be mined for new information and how knowledge discovered in the data-mining process can be used for predictive analysis.

2 Background - Hadoop, MapReduce and Hive

2.1 Hadoop and MapReduce

Apache Hadoop is an open source implementation of the MapReduce framework and distributed file system (HDFS). The framework is designed to automatically divide applications into small

[1] Examples are based on Amazon Web Services http://aws.amazon.com/

fragments of work, each of which can be executed on any node in the cluster, handle node failures as well as "schedule inter-machine communication to make efficient use of the network and disks" [2, 3]. Hadoop MapReduce solution has been referred to in the literature as "one of the best solutions for batch processing and aggregating flat file data in recent years" [10]. In [7] they emphasize high fault tolerance of MapReduce model for large jobs; its usefulness for handling data processing and data loading in a heterogeneous system; and ability to handle the execution of more complicated functions than are supported directly in SQL in parallel database systems. They also make several suggestions for working with MapReduce, such as avoiding using inefficient textual format for the data (and use HDFS binary format instead); taking advantage of natural indices (such as timestamps in log file names); and leaving MapReduce output unmerged, as there is no benefit to merging it if next consumer is another MapReduce program.

Some of the recent research has focused on the comparison of the MapReduce technology to parallel DBMSs [9, 18, 19]. For example, part of work done in [9] compares performance of parallel RDBMS (specifically, Microsoft SQL Server 2008 R2 Parallel Data Warehouse (PDW)) and NoSQL database systems (such as Hive) in analytical workloads, mainly heavy querying performed on big data. They conclude that although NoSQL system provides for more flexibility and scales better with the increase in the data size, PDW employs "a robust and mature cost-based optimization and sophisticated query evaluation techniques that allow it to produce and run more efficient plans than the NoSQL system" and suggest that MapReduce-based systems adopt such techniques to improve query performance.

Work has also been done to develop new models for parallel and distributed data management and analysis systems. For example, [11] describes ASTERIX, a UC project which began in early 2009. The project's goal is to create a new parallel, semi-structured information management system. While [12] presents a new programming model and architecture for iterative programs – HaLoop – an extended and modified version of the Hadoop MapReduce framework which holds Hadoop's fault-tolerance properties, allows programmers to reuse existing mappers and reducers from conventional Hadoop applications. Their parallel and distributed system supports large-scale iterative data analysis applications, including statistical learning and clustering algorithms.

2.2 Apache Hive

Hadoop is not a database system, so we use Hive, which was originally developed at Facebook, and then open-sourced [8]. Hive as an Apache open-source project, is intended to be a data warehouse system for Hadoop. It allows users to apply table definitions and structure on top of existing data files stored either directly in HDFS on in other data storage systems and further query the data in HiveQL, a SQL-like language. Hive queries are executed in MapReduce [6].

In [19], the authors present a comparative study of Hive and MySQL data systems with respect to large-scale data management on the cloud. The study notes advantages and disadvantages of each system. They specifically note that Hive is designed for a high-latency, batch-oriented type of processing as a disadvantage. For analytical systems this may not be an issue, since many companies may choose to pre-filter and pre-process their data before it is loaded into Hive warehouse for storage and analysis. Because Hive is Hadoop based it does not work well for ad-hoc queries and interactive responses. "The infrequent writes in analytical database workloads, along with the fact that it is usually sufficient to perform the analysis on a recent snapshot of the data (rather than on up-to-the-second most recent data) makes the 'A', 'C', and 'I' (atomicity, consistency, and isolation) of ACID easy to obtain. Hence the consistency tradeoffs that need to be made as a result of the distributed replication of data in transactional databases are not problematic for analytical databases." [1]

Abadi in [1] also calls for a hybrid solution for cloud based systems. The paper presents an analysis of deploying transactional and analytical database systems in the cloud. They conclude that the characteristics of the data and workloads of typical analytical data management applications are well-suited for cloud deployment. Specifically, (1) shared-nothing architecture scales well with the increasing data sizes; (2) since analysis can typically be performed on a snapshot of the data as opposed to in real-time, writes in analytical databases are infrequent and performed in a batch as such making consistency non-problematic for analytic databases; and (3) data security is usually a big concern for cloud based systems, in analytic databases, however, data can be pre-processed ahead of time leaving out particularly sensitive data or by applying an anonymization function. They further look into MapReduce systems and shared-nothing parallel databases as two platforms for data management on the cloud. While they conclude that a hybrid solution is needed to satisfy all of the

desired qualities of a cloud-based database system, they note the ability of MapReduce software to immediately read data off the file system and answer queries without any kind of loading stage as an advantage.

3 Related Work

3.1 Machine Learning and Data Mining on MapReduce

MapReduce has been adopted by a large number of organizations as the technology for large-scale data storage and processing. Today enterprises do not only store 'big-data', but use it to discover knowledge. A number of articles have been published on the implementation of parallel data mining and machine learning algorithms on MapReduce framework [13, 20, 21, 22]. Typical characteristics of data mining algorithms include (1) iterative behavior, meaning they require performing multiple passes over the data; (2) the input data set contains numeric and discrete categorical attributes; (3) models are computed and represented with vectors, matrices and histograms [14].

One of the available tools for data-mining and machine-learning on the cloud includes Mahout, an open source machine learning library from Apache focused on three key areas of machine learning, such as recommender engines, clustering, and classification. Machine Learning algorithms are written in Java and some portions are built upon Apache Hadoop distributed computation project. Mahout does not provide a GUI; it is a Java library to be used and adapted by developers.

Other work published on the subject includes HaLoop, an extended and modified version of the Hadoop MapReduce framework which holds Hadoop's fault-tolerance properties, allows programmers to reuse existing mappers and reducers from conventional Hadoop applications. Their parallel and distributed system supports large-scale iterative data analysis applications, including statistical learning and clustering algorithms [12].

The research in [13] presents a portable infrastructure designed with the intention of parallelizing machine-learning data-mining (ML-DM) computations. It provides built-in support to process data stored in a variety of formats. NIMBLE is being used by programmers at IBM Corporation. The article discusses deficiencies of using MapReduce for ML-DM computations: (1) the need for custom code to manage large computations (iterative and recursive); (2) when multiple computations can be

performed inside a single MapReduce job, users are responsible for co-scheduling and pipelining these computations.

The paper in [20] implements as a Java library a parallel algorithm, PARMA, which uses sets of small random samples for approximate frequent itemset or association rules mining in Hadoop MapReduce. As noted in the article, because the algorithm is programmed in Java, there is a possibility of future integration with Mahout.

The work in [23] implements and compares performance of Naïve Bayes algorithm in SQL queries, UDF, and MapReduce over different data set sizes. Here they find that SQL and UDF outperform MapReduce, although they emphasize the usefulness of MapReduce in large clusters of inexpensive hardware and its fault tolerances. They also call for future research into hybrid solutions combining SQL and MapReduce.

Twitter presents a case study of how machine-learning tools are integrated into their analytics platform using Hadoop based Apache Pig, a high-level language for large-scale data analysis[2]. It is interesting to note, that while Twitter had the goal of using off-the shelf solutions for their machine-learning and analytics processes, Mahout did not work for them, as Twitter had already established production workflows for analytics and integrating Mahout would have required significant reworking of production code. They did however open-source their code which provides the ability to encode data in Pig using the hashed encoding capabilities of Mahout (SequenceFiles of VectorWritables)[3] and allows training of logistic regression models [15].

In this project we use AWS Elastic MapReduce and Amazon S3 services to store data on the cloud. We use Apache Hive to pre-process data, when necessary, so that it is in the appropriate format to be used as an input to Apache Mahout's machine learning algorithms, such as (1) Naïve Bayes for text classification, (2) an item-based recommender algorithm to prototype a recommender engine and (3) random forests to predict property values based on a set of numerical and categorical attributes. Finally we develop programs in Java which use these models for predictive analysis.

[2] http://pig.apache.org/
[3] https://github.com/tdunning/pig-vector

4 Cloud Environment

The following Amazon Web Services (AWS) resources were used in this project for the cloud environment:

- Amazon Elastic MapReduce (EMR)
- Amazon Elastic Compute Cloud (EC2) and
- Amazon Simple Storage Service (S3) to store data files.

EMR Job Flow manages EC2 instances which in our case are started from Amazon Machine Images (AMI's) and run Hadoop under Linux and Apache Hive. Small EMR clusters are used for most of the work in this project; programs were also run on a cluster of Large EC2 instances. Characteristics of small clusters are as follows:

- 1.7 GiB memory
- 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
- 160 GB instance storage
- I/O Performance: Moderate
- API name: m1.small
- US East (N.Virginia) pricing for Amazon EC2 is $0.065 per hour, Elastic MapReduce is an additional $0.015 per hour.

The versions of Linux, Hadoop, Maven, Java, Hive and Mahout used in the project are as shown in the Figure 1 below.

A sample shell script, included as part of Appendix A – Environment Setup, contains commands to download and install Apache Mahout Distribution-0.7.

Additional Hive configurations are required to run Recommender Engine Hive queries. The configurations enable dynamic partitions and increase the default number of dynamic partitions per node (see Appendix A).

Fig. 1. Linux, Hadoop, Maven, Java, Hive and Mahout versions

```
ec2-107-21-72-233.compute-1.amazonaws.com - PuTTY
hadoop@domU-12-31-39-10-88-EB:~$ uname -mrs
Linux 3.2.30-49.59.amzn1.i686 i686
hadoop@domU-12-31-39-10-88-EB:~$ hadoop version
Hadoop 1.0.3
hadoop@domU-12-31-39-10-88-EB:~$ mvn --version
Apache Maven 2.2.1 (rdebian-4)
Java version: 1.6.0_26
Java home: /usr/lib/jvm/java-6-sun-1.6.0.26/jre
Default locale: en_US, platform encoding: ANSI_X3.4-1968
OS name: "linux" version: "3.2.30-49.59.amzn1.i686" arch: "i386" Family: "unix"
hadoop@domU-12-31-39-10-88-EB:~$ cat hive/version
0.8.1.6
hadoop@domU-12-31-39-10-88-EB:~$ ls mahout*
mahout-distribution-0.7-src.tar.gz

mahout-distribution-0.7:
LICENSE.txt  README.txt  buildtools  distribution  forests      math     src
NOTICE.txt   bin         core        examples      integration  pom.xml  target
hadoop@domU-12-31-39-10-88-EB:~$
```

5 Text Classification with Mahout's Naïve Bayes Algorithm

In practice, text classification techniques have a wide application, including in news gathering and email systems. Extremely large data sets which are becoming increasingly widespread require an increased amount of data for training for improved accuracy. Mahout's algorithms are designed to be highly scalable and with the increase of the number of records required to train a model, the time and memory required for training a Mahout algorithm may not increase linearly, making scalable algorithms in Mahout widely useful [26]. We use Mahout's Naïve Bayes algorithm and implement a program which can be used as a prototype in news gathering and email systems.

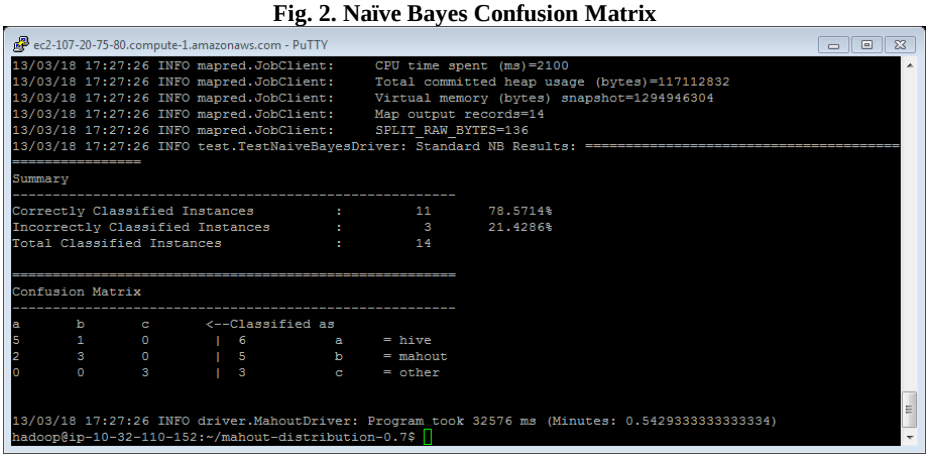
We collected a set of e-mails from Mahout and Hive user lists [24, 25] as well as emails from other newsletters and use them to train Naïve Bayes algorithm in order to classify text input between 'hive', 'mahout' and 'other' categories. The emails were saved as text files in HDFS on Amazon's Elastic Compute Cluster running Hadoop 1.0.3 with Apache Mahout 0.7 installed.

Using command line interface, Mahout commands are run to train and test the classifier model, the commands are included in Appendix B (see email.sh). The process involves the following steps:

- First, we convert directories containing the text files into sequential file format;

- Then document vectors are created with term frequency–inverse document frequency (TF-IDF) weighing (implemented by Mahout), to give the topic words more importance in the resulting document vectors;
- Mahout provides a command to split the training input into training and testing set with a specified split percentage. In this example, 80 percent of the input data set was used to train the model, and 20 percent to test it;
- We then train the algorithm using the training data set to create a Naive Bayes classifier model;
- Finally, we test the performance of the model with the testing data set by running Mahout's `testnb` command, which produces a confusion matrix as shown in Figure 2. In our case the confusion matrix shows that the model predicts correct category labels for 78.57% of files that we used to test the model.

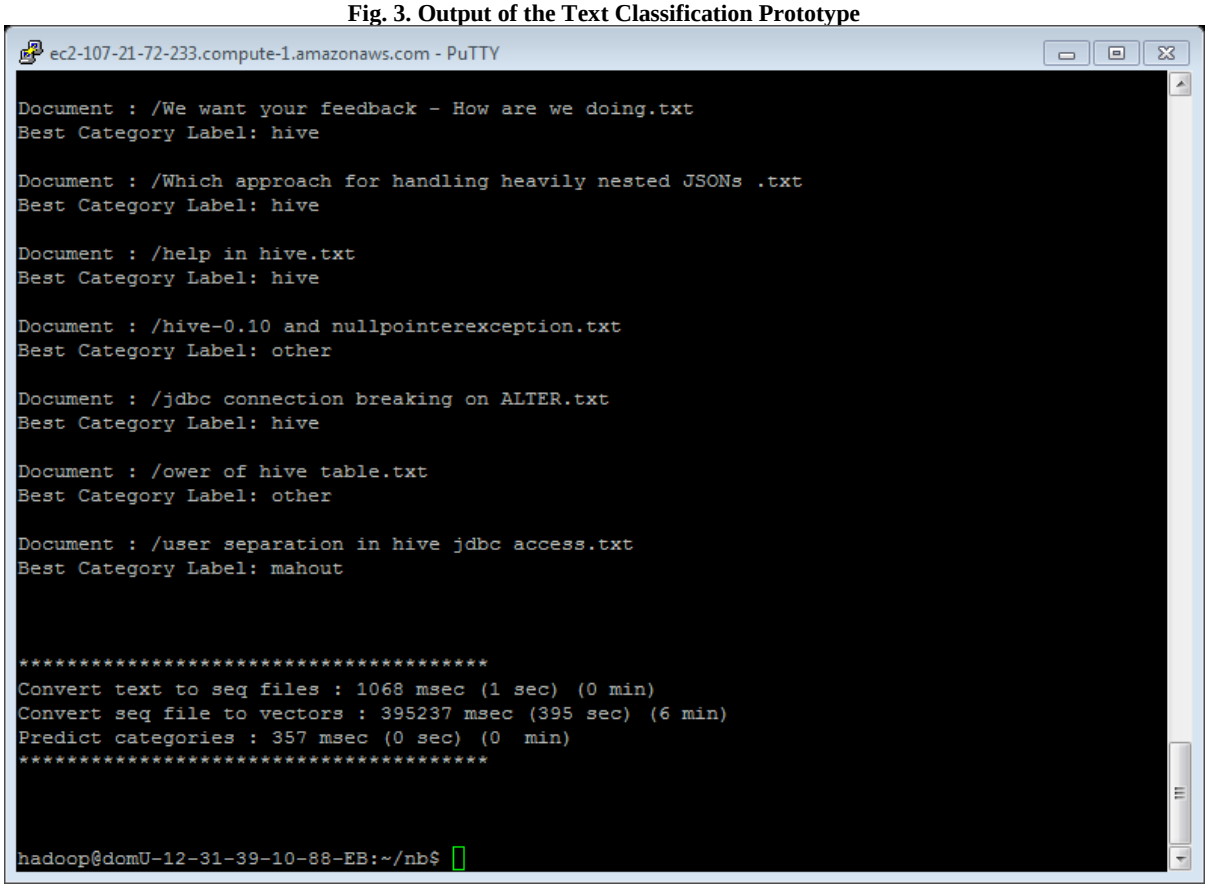
Fig. 2. Naïve Bayes Confusion Matrix



Next, in order to use the model to classify new text files we wrote a Java program which builds upon Mahout libraries to take a directory of text files as an input, convert it to a SequenceFile format and generate TF-IDF weighted sparse vectors, which are then classified with the existing model. Maven is used to manage dependencies, build and run the program on the cloud. Java code, Maven Project Object Model file (pom.xml) and script containing commands to build and run the jar file (see "naïve bayes maven project.sh") are included in the Appendix B of the report; these files are also submitted electronically along with the Naïve Bayes Model file, Label index file, data which can be used to test the model and an executable jar file.

For each text file in the input directory the classification result contains the index of the category label and the associated score; the output of our program is the category label which corresponds to the best score produced by the classification model as shown in Figure 2 below. Specifically, "Key:" in Figure 3 represents the name of the file we need to classify and is followed by the "Best Category Label:" which is the predicted category for this text document. The predicted category can be used in email systems to automatically classify incoming email messages into various categories; it can also be used to classify incoming news feeds into appropriate categories or in any other system dependent upon text classification. The email/text classification prototype is built upon Mahout Java libraries and runs on the cloud.

Fig. 3. Output of the Text Classification Prototype



The program also prints some basic running time statistics showing that converting sequential files to sparse TF-IDF weighted vectors took the most time to execute. It should be noted that Mahout's SparseVectorsFromSequenceFiles which is used in our program is implemented in MapReduce which should be highly scalable with the increase in the number of documents to classify and the number of nodes in the Hadoop cluster.

6 Building a Recommender Engine Prototype with Mahout and Hive

Recommender systems are used in many e-commerce, retail, financial services, insurance and marketing systems. We propose a recommender system prototype built upon cloud technologies using Mahout and Hive. We store users' buying history on the cloud, specifically Amazon S3 as a text file, which consists of transactional data containing user id, item id, product description, quantity purchased as well as other information (data file: recommender/data/input/useranditem). We use Amazon Elastic MapReduce (EMR) service to start a cluster of Hadoop nodes running Hive. Then install Mahout and configure Hive (see Appendix A for the list of commands).

HiveQL queries are then run in the command line interface or as a Java program (see PreProcessData.java) to (1) create an external table over the data file and (2) create a new table and insert formatted data from a select statement into it. This formatted data serves as the input to Mahout's Recommender algorithm. Java and HiveQL code are included in the Appendix C of this report.

Once the pre-processing of the data is complete, we can run Mahout's Recommender job, which is implemented with Apache Hadoop Distributed File System and MapReduce. The recommender job then starts a series of five MapReduce jobs, where the input data derived with HiveQL in the form of userid, itemid, preference, is first used to generate user vectors. The vectors are then used to compute a co-occurrence matrix, from which the algorithm derives recommendation vectors and eventually user recommendations in a compressed text file of the following format:

```
Userid1  [item1:pref1, item2:pref2, item3:pref3,..]
Userid2  [item1:pref1, item2:pref2, item3:pref3,..]
...
...
...
Useridn  [item1:pref1, item2:pref2, item3:pref3,..]
```

Figure 4 shows a printout of the recommender output from HDFS.

Fig. 4. Recommender output in HDFS

```
ec2-107-21-72-233.compute-1.amazonaws.com - PuTTY
7772    [7702:4.2,87:4.0,416:4.0,534:3.6666667,2781:3.6666667,4154:3.6666667,5302:3.6666667,7110:3.6666667,14
41:3.6666667,2971:3.6666667]
7788    [4148:36.666668,5289:36.666668,5292:36.666668,640:36.0,323:35.0,532:35.0,1985:35.0,444:35.0,1998:35.0
,445:34.0]
7888    [7889:9.0,5948:8.75,5044:8.333333,7789:8.333333,3405:8.222222,3309:8.125,8013:8.0,7124:8.0,8019:8.0,6
12:7.857143]
7968    [7664:22.11111,7036:21.75,3468:21.454546,3575:21.1,7629:20.625,6697:20.571428,3982:20.0,1942:20.0,481
8:20.0,7882:18.857143]
7976    [4692:3.5,295:3.5,2489:3.5,7807:3.3333333,5302:3.3333333,7966:3.25,1200:3.25,4105:3.1111112,7667:3.06
66666,703:3.0]
8008    [968:7.5,886:7.0,1926:7.0,2314:6.0,5983:6.0,977:6.0,3056:5.75,1865:5.5,7044:5.428571,8008:5.428571]
8020    [6458:18.666666,4862:18.0,4791:18.0,7692:18.0,7682:18.0,5207:17.5,5242:17.5,5203:17.5,1715:16.666666,
2864:16.2]
8208    [2855:4.0,1216:3.5,1212:3.5,41:3.5,2670:2.6666667,982:2.6666667,4520:2.6666667,4018:2.6666667,5125:2.
6666667,6901:2.6666667]
8244    [5907:14.166667,5258:13.0,4398:13.0,6916:12.0,5388:11.875,1997:11.428572,1600:11.25,2764:10.833333,27
69:10.833333,6056:10.833333]
8252    [5386:7.3333335,2306:7.0,1229:7.0,2339:7.0,1143:7.0,3828:7.0,7534:7.0,3795:7.0,1059:5.6666665,3824:5.
6666665]
8300    [103:13.25,3405:12.285714,2110:12.0,3992:11.6,7584:11.6,6697:11.142858,5388:11.090909,5104:11.0,1029:
11.0,994:11.0]
8348    [5136:3.3333333,1148:3.3333333,5134:3.3333333,5137:3.3333333,6530:3.25,4566:3.25,4808:3.1818182,563:3
.142857,4115:3.125,1384:3.125]
8376    [976:3.5,1448:3.5,329:3.0,2017:3.0,1047:3.0,466:2.5,6079:2.3333333,7664:2.3333333,2591:2.3333333,984:
```

We use HiveQL to parse the output into a hive table partitioned by userid with recommended items in each partition sorted by preference score in the descending order. Partitioning and pre-sorting is done to speed up querying of the output. We provide HiveQL queries to answer the following questions:

- Top K recommended items query returns most frequently recommended items,
- Recommended items for a specific User Id, and
- Top K recommended items for a user.

Fig. 5. Top 15 Recommended Items

```
Top K Parameter or 0 to exit:
15
Running: SELECT * FROM item_counts LIMIT 15
545     13882   PENCIL-#2-TICONDEROGA<1388 DZ   11
1863    51505   CLEANER-DUSTER-10 OZ-2/PK( PK   11
6062    MMMC60BK        """DISPENSER-TPE 1""""CORE-BK EA"""     10
492     12242   PENCIL-#2-EARTHWRITE-BK<12 DZ   10
499     124     PAD-DESK-MLY 22X17 RECYC EA     10
4818    EVEEN91 BATTERY-AA-24/PK BX     10
4315    CYO501402       CHALK-ANTI-DUST-12/BX-WHT BX    9
4148    CE505A  TONER-F/P2035/2055-2.3K EA      9
4808    EVEE91SBP36H    BATTERY-ENERGIZER-AA-36PK PK    9
2489    72100   CLIP-BND #100 DZ        9
295     10330   FOLDER-MLA-1/3 CUT-LTR BX       9
5258    HEWCC530A       TONER-F/ CP2025-BK-3.5K EA      8
530     13402   SCISSORS-8*BENTSTR-3PK-BK PK    8
612     16113   FOLDER-REIN-1/3CT-MLA-LTR BX    8
4692    ESS415215       FOLDER-HANG-LTR-25/BX-GN BX     8
```

Figure 5 shows a listing of 15 most frequently recommended items, where the first column represents the recommended item ID, followed by product code, description and calculated preference for that item.

Fig. 6. Top K Recommendations for a User ID

```
Enter userid or 0 to exit:
9924
Top K Parameter or 0 to exit:
5
Running: SELECT * FROM recommendations_descr WHERE userid = 9924 LIMIT 5
1329    2.0     3321131 PEN-BALLPOINT-MED PT-RD DZ
1331    2.0     3331131 PEN-BALLPOINT-MED PT-BK DZ
1403    2.0     35669   PAD-12 SLFSTCK 3X3-AST PK
1326    2.0     3311131 PEN-BALLPOINT-MED PT-BE DZ
6305    2.0     PAP3311131      PEN-BALLPOINT-MED PT-BE DZ
Enter userid or 0 to exit:
```

Figure 6 shows top 5 recommended items for user id 9924, where the first column represents the recommended item ID, followed by the calculated user preference for that item, product code and description.

Fig. 7. Recommendations for a User ID

```
Enter userid or 0 to exit:
9924
7026    1.9     SAN30001        MARKER-SHARPIE-BK DZ
2486    1.91    72010   CLIP-BINDER-MINI-12/BX DZ
921     1.92    21200   TISSUE-FACIAL TISSUE-WE PK
3273    1.93    ACC72020        CLIP-BND #20 DZ
1331    2.0     3331131 PEN-BALLPOINT-MED PT-BK DZ
1329    2.0     3321131 PEN-BALLPOINT-MED PT-RD DZ
1326    2.0     3311131 PEN-BALLPOINT-MED PT-BE DZ
6305    2.0     PAP3311131      PEN-BALLPOINT-MED PT-BE DZ
6307    2.0     PAP3331131      PEN-BALLPOINT-MED PT-BK DZ
1403    2.0     35669   PAD-12 SLFSTCK 3X3-AST PK
```

Figure 7 lists all recommended items for user id 9924, where the first column represents the recommended item ID, followed by the calculated user preference for that item, product code and description.

As stated previously, Hive is a data warehouse system and is not typically intended for online transaction processing where fast response time matters. So, we experiment with additional Java classes. One class, RecomCSVFile, converts the recommender engine output into a CSV file so it can be used in another application with an underlying relational database, for example MySQL, SQL Server or Oracle. Sample output of the RecomCSVFile is shown in Figure 8 below, where UserId is followed by the ItemId and the preference value. The other class, RecomXMLFile, produces user recommendations as an XML file which can be used in web applications. Sample output of the RecomXMLFile is shown in Figure 9 below.

Fig. 8. Recommendations in CSV format

```
10016,4030,3.0
10016,3788,3.0
10016,3100,3.0
10016,7048,2.33
10016,7044,2.33
10016,6059,2.33
10016,1249,2.33
10016,3312,2.33
10016,7608,2.33
10016,7629,2.33
10128,3892,11.0
10128,855,11.0
10128,3891,11.0
10128,46,11.0
```

Fig. 9. Recommendations in XML format

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<recommendations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="file:///C:/temp/recom_out.xsd">
  <user itemid="4030" userid="10016">
    <preference>3.0</preference>
    <product_code>C9720A</product_code>
    <description>TONER-BLK-F/4600/4650 (9K) EA</description>
  </user>
  <user itemid="3100" userid="10016">
    <preference>3.0</preference>
    <product_code>99000</product_code>
    <description>SURGE-SUPR-6-OUTLET-GY(990 EA</description>
  </user>
  <user itemid="3788" userid="10016">
    <preference>3.0</preference>
    <product_code>BICWOTAP10</product_code>
    <description>TAPE-CORRECTION-10PK-WE BX</description>
  </user>
  <user itemid="1249" userid="10016">
    <preference>2.33</preference>
    <product_code>31020</product_code>
    <description>PEN-RLRBLL-GELINK-FPT-BK DZ</description>
  </user>
```

Java and HiveQL code as well as input and output data files are submitted with this report.

7 Predicting Property Values with Random Forests

Mahout library includes sequential and MapReduce (parallel) implementations of the Random Forests classifier which can take as inputs data with numeric and categorical attributes.

For this project we use the City of Milwaukee Master Property Record data[4] and attempt to predict property value ranges based on the building area, lot area, zoning, land use, street name, zip code and other attributes. The data is originally available in Excel and fixed length file formats. We use the data in the fixed file format to store it on the cloud and run Hive select queries to create data which will be used to train and test the classification model. Queries used to pre-process the data are included in Appendix D and also as text file (see report/mprop-hive-queries.sql) so they can be executed in Hive with `hive -f <filename>` command.

We use Mahout's commands to first generate the dataset, which creates a description of the data and stores classification labels, and then use the BuildForest command to build a new Random Forests model. Next, we use Mahout's TestForest command to test and evaluate the model. Commands used to build the Random Forests model are provided in the Appendix D of this report.

The TestForest class can produce an output file, which lists for each line of the data input file the index of the predicted classification label. We use the TestForests class with the Random Forests dataset description file and the model file as follows to classify new data:

- For sequential classification, we create a modified version of the Mahout's TestForests class so that for each input line the output file contains the label of the prediction, instead of the index value, followed by the data input line, as shown in Figure 12;
- We also write a Java program (ReadOutput.java) which merges the input and output files line by line, writing the predicted category label followed by the input data to a new file. The merged file looks similar to the output shown in Figure 12. This code can be used to interpret the output of either MapReduce or sequential implementation of the Random Forests algorithm. The merged file is created in the results directory (see Figure 10):

[4] http://city.milwaukee.gov/DownloadTabularData3496.htm

Fig. 10. ReadOutput class merges the input and the output files line by line

```
ec2-107-21-72-233.compute-1.amazonaws.com - PuTTY
hadoop@domU-12-31-39-10-88-EB:~/forests$ java -cp target/forests-1.0-jar-with-dependencies.jar c
om.klav.app.ReadOutput /user/hadoop/temp/forests/mprop_desc2.info /user/hadoop/mpropdata/class /
user/hadoop/forests-out
Output file name is hdfs://10.198.139.25:9000/user/hadoop/forests-out/merged.out
dataFile being read is 0hdfs://10.198.139.25:9000/user/hadoop/mpropdata/class/000000_0
resFile being read is 0hdfs://10.198.139.25:9000/user/hadoop/forests-out/000000_0.out
hadoop@domU-12-31-39-10-88-EB:~/forests$ hadoop fs -ls
Found 9 items
drwxr-xr-x   - hadoop supergroup          0 2013-04-22 20:34 /user/hadoop/forests-out
drwxr-xr-x   - hadoop supergroup          0 2013-04-22 17:02 /user/hadoop/forests-out-seq
drwxr-xr-x   - hadoop supergroup          0 2013-04-20 03:59 /user/hadoop/forests-out1
drwxr-xr-x   - hadoop supergroup          0 2013-04-20 04:03 /user/hadoop/forests-out1seq
drwxr-xr-x   - hadoop supergroup          0 2013-04-19 13:29 /user/hadoop/input
drwxr-xr-x   - hadoop supergroup          0 2013-04-20 02:11 /user/hadoop/mpropdata
drwxr-xr-x   - hadoop supergroup          0 2013-04-20 02:20 /user/hadoop/ob
drwxr-xr-x   - hadoop supergroup          0 2013-04-19 15:32 /user/hadoop/output
drwxr-xr-x   - hadoop supergroup          0 2013-04-20 02:13 /user/hadoop/temp
hadoop@domU-12-31-39-10-88-EB:~/forests$
```

Finally, we write a short Java program to classify one input instance at a time. Instance to be classified consists of a string of attributes in the same format as that which was used to train the model (in our case these are comma separated values). The output of this Java program is the predicted category label, such as the property value range (see Figure 11):

Fig. 11. Classifying one instance at a time

```
ec2-107-21-72-233.compute-1.amazonaws.com - PuTTY
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/ already added, skipping
[INFO] META-INF/MANIFEST.MF already added, skipping
[INFO] org/ already added, skipping
[INFO] org/apache/ already added, skipping
[INFO] org/apache/commons/ already added, skipping
[INFO] META-INF/LICENSE.txt already added, skipping
[INFO] META-INF/NOTICE.txt already added, skipping
[INFO] META-INF/maven/ already added, skipping
[INFO] META-INF/maven/org.apache.commons/ already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 53 seconds
[INFO] Finished at: Mon Apr 22 17:44:55 UTC 2013
[INFO] Final Memory: 36M/86M
[INFO] ------------------------------------------------------------------------
hadoop@domU-12-31-39-10-88-EB:~/forests$ java -cp target/forests-1.0-jar-with-dependenc
ies.jar com.klav.app.ClassifyRandomForestsInstance /user/hadoop/temp/forests/mprop_desc
2.info ob/forest.seq -,S_ILLINOIS_AV,7,RY,100000,1944,100000,RM1,8830,03,020800,4010,53
207
Predicted label : 0-249K
Data string read : S_ILLINOIS_AV,7,RY,100000,1944,100000,RM1,8830,03,020800,4010,53207
hadoop@domU-12-31-39-10-88-EB:~/forests$
```

Figure 12 shows sample classification output for three data records. The predicted value for the first record is “1M+” and for the last two records the predicted value range is “0-249K”.

```
ec2-107-21-72-233.compute-1.amazonaws.com - PuTTY
1M+, W_ZELLMAN_CT,4,RY,73968,0000,108334,PD,7011,06,021700,4000,53221
1M+, W_ZELLMAN_CT,4,R,20440,0000,281180,IL1,4213,09,021700,4000,53221
1M+, S_13TH_ST,4,RY,37823,0000,89995,LB1,7011,06,021700,4000,53221
1M+, S_13TH_ST,4,CY,131846,0000,383633,LB1,7011,06,021700,4000,53221
0-249K, S_13TH_ST,4,RY,0,0000,356931,LB1,8880,13,021700,4000,53221
0-249K, W_GIUFFRE_CT,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, W_GIUFFRE_CT,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, W_ZELLMAN_CT,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, W_ZELLMAN_CT,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
500K-749K, W_COLLEGE_AV,4,RY,990,0000,15600,LB1,5541,05,021700,4007,53221
1M+, S_20TH_ST,2,R,6440,0000,34198,NS1,9999,07,021700,4007,53221
0-249K, S_13TH_ST,9,0,0,0000,2140690,RS5,8771,12,021700,4000,53221
1M+, S_13TH_ST,4,R,103449,0000,196543,LB1,7011,06,021700,4000,53221
500K-749K, S_13TH_ST,4,RY,4254,0000,22500,LB1,5541,05,021700,4000,53221
750K-999K, E_COLLEGE_AV,4,TY,288,0000,328397,IL1,5170,05,160100,4000,00000
0-249K, E_COLLEGE_AV,9,0,0,0000,1072055,IL1,8880,13,160100,4000,00000
500K-749K, S_13TH_ST,4,RY,24500,0000,218410,IL1,4231,09,021700,4000,53221
0-249K, S_13TH_ST,4,CY,14416,0000,268330,IL1,4212,09,021700,4000,53221
0-249K, S_13TH_ST,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, S_13TH_ST,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, S_13TH_ST,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
0-249K, S_13TH_ST,4,RY,0,0000,20,XXXX,0006,00,021700,4000,53221
1M+, S_13TH_ST,4,RY,46342,0000,439956,IL1,4213,09,021700,4000,53221
hadoop@domU-12-31-39-10-88-EB:~/forests$
```

Java programs proposed here are not data specific and can be used to classify new data and read predicted values of any Random Forests model built with Mahout.

8 Conclusions and Future Work

In this work we experimented with open source MapReduce tools that are largely available over the cloud for developers to use in big data analysis. Specifically, we explored Apache Hive data warehousing solution and Hive Query Language to store and pre-process the data and experimented with three machine learning algorithms available in Apache Mahout's library:

- We used Mahout's Naïve Bayes classifier to build an email/text classification prototype, which can be used in email systems to automatically classify incoming email messages into various categories. This prototype can also be used to classify incoming news feeds into appropriate categories or in any other system dependent upon text classification.
- Then, we used an item similarity recommendation algorithm to predict what products a user may be interested in purchasing based on other customers' buying history. We processed the output of Mahout's recommender job in HiveQL to produce a list of recommended items for a user and to answer such queries as top K recommended items for a user and top K most recommended items.

- And finally we used the Random Forests classifier to build a model to predict property value ranges. We wrote Java code which works with the classification model to classify new data and returns predicted category labels.

Future work should include evaluating the prototypes' performance with large data sets on a cluster of distributed machines on the cloud.

Contact Information

If you have any further questions regarding this paper, code or scripts included herewith please do not hesitate to contact me at `klavdiya.hammond@gmail.com`

Appendix A – Environment Setup

1. **Hive configurations** to allow dynamic partitions and override the default number of dynamic partitions per node from 100 to 10000 (important for the Recommender Engine implementation, described in section 6 of this paper); set these at hive prompt:

```
set hive.exec.dynamic.partition=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.exec.max.dynamic.partitions.pernode=10000;
```

2. Script to install Mahout on Hadoop, requires Apache Maven (install-mahout.sh)
(Continues on the next page)

```bash
#!/bin/bash
#
# Downloads and installs mahout-0.7
#
# To run type: /bin/bash install-mahout.sh

if [ "$1" = "--help" ] || [ "$1" = "--?" ]; then
  echo "This script installs mahout-0.7"
  exit
fi

echo "Downloading mahout source files"
  wget http://apache.mirrors.lucidnetworks.net/mahout/0.7/mahout-distribution-0.7-src.tar.gz

echo "Extract mahout files from mahout-distribution-0.7-src.tar.gz"
  tar -zxvf mahout-distribution-0.7-src.tar.gz
  cd mahout-distribution-0.7
echo "Install mahout"
  mvn -DskipTests install
```

Appendix B – Text Classification with Naïve Bayes

1. Download data and build Naïve Bayes model (see *email.sh*)

2. Build maven project or use the Jar file (*naivebayes-1.0-jar-with-dependencies.jar*) to classify new data. See steps for both in the *naive bayes maven project.sh* file.

3. Code included:

    a) ClassifyNBayes.java
    b) ConvertTextToSeq.java (not required, but can be used independently from ClassifyNBayes.java to convert a directory of text files or subdirectories of text files to a sequential file of form <Key(file name), Value(file content)> format)
    c) Pom.xml (Project Object Model file, contains Maven project configuration, including plugins and dependencies)
    d) naivebayes-1.0-jar-with-dependencies.jar (executable, see *'naïve bayes maven project.sh'* on how to run)

4. Data included:
    a) The *Email* folder contains data which was used to build the Naïve Bayes model;
    b) The *Email2classsify* folder contains text files which were used as the new data to classify with the obtained model;
    c) naiveBayesModel.bin (ready to use Mahout's Naïve Bayes text classification model specific to this example);
    d) labelindex (ready to use file containing labels for classification categories which correspond to predictions produced by the above model).

```
#Klavdiya Hammond
#April 24, 2013
#Script contains commands to
# (1) convert directory of files into a sequential file
# (2) convert sequential file into tf-idf vectors
# (3) split vectors into training and testing datasets
# (4) build naive bayes classification model using the training dataset
# (5) test naive bayes model useing the test data
# mahout must be installed first - see install-mahout.sh
#
mkdir email
s3cmd get --recursive s3://klav.project/email/ email/
#
#copy dir to hdfs
hadoop fs -copyFromLocal email/ /user/hadoop/email/
#
#then convert to sequence files:
cd mahout-distribution-0.7
bin/mahout seqdirectory -i /user/hadoop/email -o /user/hadoop/email-seq
#
#then create vectors:
bin/mahout seq2sparse -i /user/hadoop/email-seq -o /user/hadoop/email-vectors -lnorm -nv -wt
tfidf
#
#split into train/test 80-20:
bin/mahout split -i /user/hadoop/email-vectors/tfidf-vectors --trainingOutput /user/hadoop/
myemail-train-vectors --testOutput myemail-test-vectors --randomSelectionPct 20 --overwrite
--sequenceFiles -xm sequential
#train NB:
bin/mahout trainnb -i /user/hadoop/myemail-train-vectors -el -o /user/hadoop/myemail-model -
li labelindex -ow
#test NB:
bin/mahout testnb -i /user/hadoop/myemail-test-vectors -m /user/hadoop/myemail-model -l
labelindex -ow -o /user/hadoop/myemail-testing
```

```
#Klavdiya Hammond
#April 24, 2013
#Script contains commands to
# (1) create maven project for naive bayes text classifier in a 'nb' directory
# (2) download pom.xml and java files
# (3) compile the project
# (4) download data to be classified
# (5) copy data directory to HDFS
# (6) run the jar file to classify text files
# mahout must be installed first - see install-mahout.sh
# Naive Bayes model and label index files have to exist - see email.sh
#
#create maven project
#replase "s3://klav.project/naivebayes/" with appropriate s3 bucket name
#containing the files
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.klav.app
-DartifactId=nb -DinteractiveMode=false
cd nb
s3cmd get s3://klav.project/naivebayes/pom.xml --force
s3cmd get s3://klav.project/naivebayes/ConvertTextToSeq.java src/main/java/com/klav/app/
ConvertTextToSeq.java --force
s3cmd get s3://klav.project/naivebayes/ClassifyNBayes.java src/main/java/com/klav/app/
ClassifyNBayes.java --force
#compile the project and build the jar file
mvn clean compile assembly:single
#
#get data to play with (a.k.a. to classify new instances)
mkdir newemail
s3cmd get --recursive s3://klav.project/naivebayes/toclassify/hive/ ./newemail/
s3cmd get --recursive s3://klav.project/naivebayes/toclassify/mahout/ ./newemail/
s3cmd get --recursive s3://klav.project/naivebayes/toclassify/other/ ./newemail/
#copy data to HDFS
hadoop fs -copyFromLocal newemail newemail
#
#classify new data
#run from nb directory
hadoop jar target/naivebayes-1.0-jar-with-dependencies.jar com.klav.app.ClassifyNBayes
myemail-model labelindex newemail
#
#convert input directory to sequential file
java -cp target/naivebayes-1.0-jar-with-dependencies.jar com.klav.app.ConvertTextToSeq
newemail
```

```java
/*
 * Program classifies text files using an existing Apache Mahout
 * Naive Bayes classification model and label index files.
 * <p>
 * Runtime arguments are (1) HDFS directory containing naiveBayesModel.bin,
 * (2) path to labelindex in HFDS and (3) HDFS directory containg text
 * files to be classified.
 * <p>
 * The program outputs the name of the text file and the predicted
 * classification category.
 * <p>
 * See "naive bayes maven project.sh" on how to run the jar file containing this class
 * <p>
 *
 * @author Klavdiya Hammond
 * @version April 26, 2013
 *
 * Classes in this package use Apache libraries, therefore
 * Apache License notice is included herewith.
 * <p>
 *
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package com.klav.app;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.LinkedList;

import com.google.common.collect.Lists;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.SequenceFile.Reader;
import org.apache.hadoop.io.SequenceFile.Writer;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.mahout.classifier.naivebayes.AbstractNaiveBayesClassifier;
import org.apache.mahout.classifier.naivebayes.NaiveBayesModel;
import org.apache.mahout.classifier.naivebayes.StandardNaiveBayesClassifier;
import org.apache.mahout.classifier.naivebayes.BayesUtils;
import org.apache.mahout.math.Vector;
import org.apache.mahout.math.RandomAccessSparseVector;
import org.apache.mahout.math.VectorWritable;
```

```java
import org.apache.mahout.text.SequenceFilesFromDirectory;
import org.apache.mahout.vectorizer.SparseVectorsFromSequenceFiles;

import org.apache.mahout.common.iterator.sequencefile.SequenceFileIterable;
import org.apache.mahout.common.Pair;
import java.util.Map;
import java.util.HashMap;
import org.apache.hadoop.io.IntWritable;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ClassifyNBayes {

    private static final Logger log = LoggerFactory
            .getLogger(ClassifyNBayes.class);

    /*
     * Runtime arguments are (1) HDFS directory containing naiveBayesModel.bin,
     * (2) path to labelindex in HFDS and (3) HDFS directory containg text files
     * to be classified
     */
    public static void main(String[] args) throws IOException, Exception,
            InterruptedException {

        Configuration conf = new Configuration();
        conf.addResource(new Path("/home/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/hdfs-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/mapred-site.xml"));
        FileSystem fs = FileSystem.get(conf);

        String model_path = args[1]; // path to the NB model created when
                                     // training
        Path modelPath = fs.makeQualified(new Path(model_path));

        System.out.println("\n*************************************");
        System.out.println("Model path : " + modelPath.toString());
        System.out.println("*************************************\n");

        String label_path = args[2];
        Path labelsPath = fs.makeQualified(new Path(label_path)); // path to the
                                                                  // file
                                                                  // containing
                                                                  // category
                                                                  // labels
        System.out.println("\n*************************************");
        System.out.println("Labels path : " + labelsPath.toString());
        System.out.println("*************************************\n");


        String textdir_path = args[3];
        System.out.println("\n*************************************");
        System.out.println("Textdir path : " + textdir_path);
        System.out.println("*************************************\n");

        //to track program execution times
        long time = System.currentTimeMillis();

        // convert dir of text files to sequential file
        String seqdir_path = convertToSeq(textdir_path);

        // time it took to convert text to seq files
```

```
        long seqtime = System.currentTimeMillis() - time;
        time = System.currentTimeMillis();
//      System.out.println("Convert text to seq files : " + seqtime);

        // System.out.println("Vectors output dir : " + vecdir_path);
        Path vecdir_path = convertSeqToSparse(fs, seqdir_path);

        // time it took to convet seq file to vectors
        long vectime = System.currentTimeMillis() - time;
        time = System.currentTimeMillis();
//      System.out.println("Convert seq file to vectors : " + vectime);

        // read labels from labelindex
        Map<Integer, String> labels = readLabels(labelsPath, conf);

        // read NB model from file
        NaiveBayesModel nbmodel = NaiveBayesModel.materialize(modelPath, conf);

        // create classifier
        StandardNaiveBayesClassifier classifier = new StandardNaiveBayesClassifier(
                nbmodel);

        Path vectorsPath = new Path(vecdir_path + "/tfidf-vectors/");
        // for every file in the tfidf-vectors directory
        for (Path file : listFiles(fs, vectorsPath)) {

            SequenceFile.Reader reader = new SequenceFile.Reader(fs, file, conf);
            // read <key, value> pairs
            Text key = new Text();
            VectorWritable vw = new VectorWritable();
            while (reader.next(key, vw)) {
                Vector toclass = new RandomAccessSparseVector(vw.get());

                Vector result = classifier.classifyFull(toclass);

                int i = result.maxValueIndex(); // index of the max probability
                                                // value
                double value = result.get(i); // returns the value at the given
                                                // index = value of the max
                                                // probability
                String label = labels.get(i); // returns the label for the
                                                // category index

                System.out.println("\nDocument : " + key);
                // System.out.println("Best Category: " + i + " Best Score: " +
                // value + " Label: " + label);
                System.out.println("Best Category Label: " + label);
            }
            reader.close();
        }
        // time it took to classify vectors
        long classtime = System.currentTimeMillis() - time;

        // print stats
        System.out.println("\n\n");
        System.out.println("****************************************");
        System.out.println("Convert text to seq files : " + seqtime + " msec ("
                + seqtime / 1000 + " sec) (" + seqtime / 1000 / 60 + " min)");
        System.out.println("Convert seq file to vectors : " + vectime
                + " msec (" + vectime / 1000 + " sec) (" + vectime / 1000 / 60
                + " min)");
        System.out.println("Predict categories : " + classtime + " msec ("
```

```
                + classtime / 1000 + " sec) (" + classtime / 1000 / 60
                + "  min)");
        System.out.println("****************************************");
        System.out.println("\n\n");
    }

    /*
     * Read category labels from a sequential file
     *
     * @param labelsPath path to the labelindex file in HDFS
     *
     * @param conf Configuration
     *
     * @return labels Map of index value (int) and category label (String)
     */
    public static Map<Integer, String> readLabels(Path labelsPath,
            Configuration conf) {
        Map<Integer, String> labels = new HashMap<Integer, String>();
        for (Pair<Text, IntWritable> pair : new SequenceFileIterable<Text, IntWritable>(
                labelsPath, true, conf)) {
            labels.put(pair.getSecond().get(), pair.getFirst().toString());
        }
        return labels;
    }

    /*
     * List files contained in an HDFS directory
     *
     * @param path path to HDFS directory
     *
     * @param fs FileSystem
     *
     * @return files Path array
     */
    public static Path[] listFiles(FileSystem fs, Path dirs) throws IOException {
        List<Path> files = Lists.newArrayList();
        for (FileStatus s : fs.listStatus(dirs)) {
            if (!s.isDir() && !s.getPath().getName().startsWith("_")) {
                files.add(s.getPath());
            }
        }
        if (files.isEmpty()) {
            throw new IOException("No output found !");
        }
        return files.toArray(new Path[files.size()]);
    }

    /*
     * Convert directory of text files (or subdirectories containing text files)
     * to the sequntial file format Key is the filename, value is the text
     * content of the file
     *
     * @param input path to the directory with text file(s)
     *
     * @return outputPath path to the sequential file(s) directory
     */
    public static String convertToSeq(String input) throws Exception {

        String outputPath = input + "-seq";

        // add arguments
        List<String> argList = new LinkedList<String>();
```

```
        argList.add("-i");
        argList.add(input);
        argList.add("-o");
        argList.add(outputPath);

        String[] args = argList.toArray(new String[argList.size()]);

        SequenceFilesFromDirectory.main(args); // mahout library class

        return outputPath;
    }

    /*
     *
     */
    public static Path convertSeqToSparse(FileSystem fs, String input)
            throws Exception {

        String outputPath = fs.makeQualified(new Path(input + "-vec"))
                .toString();

        // add arguments
        List<String> argList = new LinkedList<String>();
        argList.add("-i");
        argList.add(input);
        argList.add("-o");
        argList.add(outputPath);

        argList.add("-lnorm");
        argList.add("-wt");
        argList.add("tfidf");

        String[] args = argList.toArray(new String[argList.size()]);

        SparseVectorsFromSequenceFiles.main(args); // mahout library class

        return new Path(outputPath);
    }
}
```

```java
/*
 * Program iterates over a direcotory of text files (or directories of text files)
 * and converts them to a sequential file
 * <p>
 * Runtime argument is the HDFS directory containing text
 * files to be classified or subdirectories of text files.
 * <p>
 * The program outputs a sequential file named '<input dir name>-seq'
 * <p>
 * See "naive bayes maven project.sh" on how to run the jar file containing this class
 *
 * @author Klavdiya Hammond
 * @version April 26, 2013
 *
 * Classes in this package use Apache libraries, therefore
 * Apache License notice is included herewith.
 * <p>
 *
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.klav.app;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.charset.Charset;
import java.util.Scanner;
import java.util.Arrays;
import java.util.List;
import com.google.common.collect.Lists;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.SequenceFile.Writer;
import org.apache.hadoop.io.Text;
import org.apache.mahout.classifier.df.DFUtils;
import org.apache.hadoop.fs.FSDataInputStream;

public class ConvertTextToSeq {

    public static void main(String[] args) throws IOException {

        Configuration conf = new Configuration();
```

```
        conf.addResource(new Path("/home/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/hdfs-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/mapred-site.xml"));

        FileSystem fs = FileSystem.get(conf);
        Path outpath = null; //output path

        String input = args[0];

        Path inpath = fs.makeQualified(new Path(input));
        if (!fs.exists(inpath)) {
            System.out.println("Input directory does not exist");
        }

//      File[] dirs = new File(args[0]).listFiles(); // input directory
        Path[] dirs = listFiles(fs, inpath); //input directory
        System.out.println("Input path: " + input);

//      Path path = new Path(args[1]); // output directory name
//      System.out.println("Output path: " + path.toString());

        if (fs.exists(new Path(input + "-seq"))) {
            System.out.println("Output directory already exist");
        } else {
            outpath = fs.makeQualified(new Path(input + "-seq")); // output directory
        }

        Writer writer = new SequenceFile.Writer(fs, conf, outpath, Text.class,
                Text.class);

        for (Path dir : dirs) {
            String dirLabel = dir.getName();
            if (fs.getFileStatus(dir).isDir()) {
                System.out.println("dirLabel : " + dirLabel);
                for (Path file : listFiles(fs, dir)) {
                    // Path filepath = new Path(path+Integer.toString(i));
                    // for (File file : dirs) {
                    String fileLabel = file.getName();
                    System.out.println("String label (file name processed): "
                            + dirLabel + "/" + fileLabel);
                    String text = readFile(fs, file);
                    // System.out.println(text);
                    writer.append(new Text("/" +dirLabel +"/"+ fileLabel), new Text(text));
                    // i++;
                    // writer.close();
                }
            } else {
                // String fileLabel = dir.getName();
                System.out.println("String label (file name processed): " + "/"
                        + dirLabel);
                String text = readFile(fs, dir);
                // System.out.println(text);
                writer.append(new Text("/" + dirLabel), new Text(text));
            }
        }
        writer.close();
//      return outpath.toString();
    }

    public static String readFile(FileSystem fs, Path path) throws IOException {
//      FileInputStream stream = new FileInputStream(path);
        FSDataInputStream dataStream = fs.open(path);
```

```
        String content = new Scanner(dataStream, "UTF-8").useDelimiter("\\A")
                .next();
        return content;
    }

    public static Path[] listFiles(FileSystem fs, Path dirs) throws IOException {
        List<Path> files = Lists.newArrayList();
        for (FileStatus s : fs.listStatus(dirs)) {
          if (!s.isDir() && !s.getPath().getName().startsWith("_")) {
            files.add(s.getPath());
          }
        }
        if (files.isEmpty()) {
          throw new IOException("No output found !");
        }
        return files.toArray(new Path[files.size()]);
      }
}
```

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.klav.app</groupId>
    <artifactId>naivebayes</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>my-app</name>
    <url>http://maven.apache.org</url>
    <parent>
        <artifactId>mahout</artifactId>
        <groupId>org.apache.mahout</groupId>
        <version>0.7</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>com.klav.app.ClassifyNBayes</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id> <!-- this is used for inheritance merges -->
                        <phase>package</phase> <!-- bind to the packaging phase -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <encoding>UTF-8</encoding>
                    <source>1.6</source>
                    <target>1.6</target>
                    <optimize>true</optimize>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <executions>
                    <execution>
```

```
                    <goals>
                        <goal>test-jar</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>copy</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>
                            lib
                        </outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.mahout</groupId>
        <artifactId>mahout-core</artifactId>
        <version>0.7</version>
    </dependency>
    <dependency>
        <groupId>org.apache.mahout</groupId>
        <artifactId>mahout-math</artifactId>
        <version>0.7</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.6.0</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-jcl</artifactId>
        <version>1.6.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-core</artifactId>
        <version>1.0.3</version>
```

```
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>14.0.1</version>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-integration</artifactId>
            <version>0.7</version>
        </dependency>
    </dependencies>
</project>
```

Appendix C – Recommender Engine Prototype

1. Apache Mahout job to create recommendations *(also see 'run recommender command.sh')*:

```
hadoop jar mahout-core-0.7-job.jar
org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-Dmapred.input.dir=/user/hadoop/input/
-Dmapred.output.dir=/user/hadoop/output/
--similarityClassname SIMILARITY_COOCCURRENCE
```

2. Java Classes:
   a) PreProcessData.java
   b) ReadInput.java
   c) Recommendations.java
   d) ProcessRecommendations.java
   e) RecomCSVFile.java
   f) RecomXMLFile.java
   g) hive_jdbc-0.8.1.tar (contains required JARs)

3. Data:

   Input
   a) useranditem – transaction data
   b) itemlistdescr – list of items with descriptions

   Output
   c) output of the reccomender job (part-r-00000, part-r-00001, part-r-00002, part-r-00003 and part-r-00004)
   d) recom_out.csv (generated by RecomCSVFile)
   e) recom_out.xml (generated by RecomXMLFile)

```
#Klavdiya Hammond
#April 27, 2013
#Mahout command to generate recommendations
#
#Apache Mahout job to create recommendations:
hadoop jar mahout-core-0.7-job.jar  org.apache.mahout.cf.taste.hadoop.item.RecommenderJob
-Dmapred.input.dir=/user/hadoop/input/
-Dmapred.output.dir=/user/hadoop/output/
--similarityClassname SIMILARITY_COOCCURRENCE
```

```java
/*
 * PreProcessData class creates and populates Hive tables
 * formatting the data to be used as the input to Apache Mahout's
 * Recommender Job
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */

package recommender;

import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class PreProcessData {

    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

    /**
     * @param args  Master Node DNS Name
     * @throws SQLException
     */

    public static void main(String[] args) throws SQLException {
        String server = "";

        if (args.length > 0) {
            server = args[0];
        } else {
            System.out
                    .println("Required parameter missing - Master Node DNS Name");
            System.exit(1);
        }

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        Connection con = DriverManager.getConnection("jdbc:hive://" + server
                + ":10003/default");
        Statement stmt = con.createStatement();
        String tableName = "spend_data_ext";
        Boolean insertdata = true;
        // create external table containing customer's spend history
        String sql = "CREATE EXTERNAL TABLE IF NOT EXISTS spend_data_ext ("
                + "UserId int, " + "TxnNum String, " + "CommodityCode String, "
                + "ProductCode String, " + "Quantity  Double, "
                + "Amount Double, " + "Description String, " + "Type String ) "
                + "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' "
                + "LINES TERMINATED BY '\n' " + "STORED AS TEXTFILE "
                + "LOCATION 's3://klav.project/supplies/useranditem/'";

        ResultSet res = stmt.executeQuery(sql);
        System.out.println("Creating table: spend_data_ext");

```

```java
        // show tables
        sql = "show tables";
        res = stmt.executeQuery(sql);
        System.out.println("Running: " + sql);

        if (res.next()) {
            System.out.println(res.getString(1));
        }

        // select * query
        sql = "select * from " + tableName + " limit 10";
        System.out.println("Running: " + sql);
        res = stmt.executeQuery(sql);
        while (res.next()) {
            for (int i = 1; i <= 8; i++) {
                System.out.print(String.valueOf(res.getString(i)) + "\t");
            }
            System.out.print("\n");
        }

        // create external table containing item ids
        sql = "create external table if not exists items_descr_ext ("
                + "ItemId int, " + "ProductCode String, "
                + "Description String) "
                + "ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' "
                + "LINES TERMINATED BY '\n' " + "stored as textfile "
                + "location 's3://klav.project/supplies/itemlistdescr/'";

        res = stmt.executeQuery(sql);

        System.out.println("Creating table: items_descr_ext");

        // select all items
        sql = "select * from items_descr_ext limit 10";
        System.out.println("Running: " + sql);
        res = stmt.executeQuery(sql);
        while (res.next()) {
            for (int i = 1; i <= 3; i++) {
                System.out.print(String.valueOf(res.getString(i)) + "\t");
            }
            System.out.print("\n");
        }
        // create table to format data to serve as the input to the recommender
        sql = "create table if not exists input_data (" + "userid int, "
                + "itemid int, " + "pref double) "
                + "ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' "
                + "LINES TERMINATED BY '\n' " + "stored as textfile "
                + "location '/user/hadoop/input/'";

        res = stmt.executeQuery(sql);
        System.out
                .println("Creating table: input_data in '/user/hadoop/input/");

        // testing - show list of tables
        sql = "show tables";
        res = stmt.executeQuery(sql);
        System.out.println("Running: " + sql);
        while (res.next()) {
            System.out.println(res.getString(1));
        }
        // end of testing - show list of tables

```

```java
        // next - populate the input_data table
        if (insertdata) {
            sql = "insert overwrite table input_data "
                    + "select userid, itemid, quantity "
                    + "from spend_data_ext u join items_descr_ext i "
                    + "where u.productcode = i.productcode " + "and itemid>1";

            System.out.println("Inserting data into the input_data table");
            res = stmt.executeQuery(sql);
        }

        while (res.next()) {
            System.out.println(res.getString(1));
        }

        sql = "select * from input_data limit 10";
        System.out.println("Running: " + sql);
        res = stmt.executeQuery(sql);
        while (res.next()) {
            for (int i = 1; i <= 3; i++) {
                System.out.print(String.valueOf(res.getString(i)) + "\t");
            }
            System.out.print("\n");
        }
    } // end of main
} // end of class
```

```java
/*
 * ProcessRecommendations class works with the output of
 * Apache Mahout's Recommender Job to create recommendation tables
 * which can then be used for querying
 * <p>
 * This class also contains methods to run the following queries:
 * 1. Top K recommended items
 * 2. Recommended items for a user, and
 * 3. Top K recommended items for a user
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */
package recommender;


import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;


public class ProcessRecommendations {

    /*
     * Method creates temporary tables, followed by recommendations tables
     * containing recommender results
     *
     * @param insertdata    specifies whether to insert data into the tables
     * @param stmt  Statement
     * @param con   Connection to the master node
     */
    public void createTables (Boolean insertdata, Statement stmt, Connection con) throws SQLException {
    //ResultSet res;
    String sql;
    //set some hive parameters to work with dynamic partitions
    stmt.execute("set hive.exec.dynamic.partition=true");
    stmt.execute("set hive.exec.dynamic.partition.mode=nonstrict");
    stmt.execute("set hive.exec.max.dynamic.partitions.pernode=10000");
    stmt.execute("set hive.optimize.s3.query=true");
    stmt.execute("set hive.base.inputformat=org.apache.hadoop.hive.ql.io.HiveInputFormat");
    stmt.execute("set hive.cli.print.header=true");

    sql = "create table if not exists recom_temp1 (" +
        "userid int, " +
        "rec string) " +
        "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' " +
        "LINES TERMINATED BY '\n' " +
        "LOCATION '/user/hadoop/output/'";
    stmt.executeQuery(sql);

    System.out.println("first temp table created");

    //Create second temporary table, split recommendations into array<recommended_item:preference>
    //and explode values so that we have user - recommended_item:preference data structure
    sql = "CREATE TABLE IF NOT EXISTS recom_temp2 (" +
        "userid INT, " +
        "recitem STRING) " +
        "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' " +
```

```
        "LINES TERMINATED BY '\n'";

    stmt.executeQuery(sql);
    System.out.println("second temp table created");

    if(insertdata) {
    sql = "INSERT OVERWRITE TABLE recom_temp2 " +
        "SELECT  userid, recitem " +
        "FROM recom_temp1 " +
        "LATERAL VIEW EXPLODE(SPLIT(SUBSTRING(rec, 2, length(rec)-2), ',')) subView AS recitem";

    System.out.println("populating second temp table...");
    stmt.executeQuery(sql);
    System.out.println("second temp table populated");
    }

    //table to use in queries to return recommendations and preferences for users
    sql = "CREATE TABLE IF NOT EXISTS recommendations (" +
        "recitem int, " +
        "pref double) " +
        "PARTITIONED BY (userid INT) " +
        "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' " +
        "LINES TERMINATED BY '\n'";

    stmt.executeQuery(sql);
    System.out.println("recommendations table created");

    //populate recommendations table
    if (insertdata) {
    stmt.execute("set hive.exec.dynamic.partition.mode=nonstrict");
    sql = "INSERT OVERWRITE TABLE recommendations PARTITION (userid) " +
        "SELECT DISTINCT " +
        "CAST(SPLIT(recitem, ':')[0] AS INT) AS recitem, " +
        "CAST(SPLIT(recitem, ':')[1] AS DOUBLE) AS pref, " +
        "userid " +
        "FROM recom_temp2 " +
        "ORDER BY pref DESC";

    System.out.println("populating recommendations table...");
    stmt.executeQuery(sql);
    System.out.println("recommendations table populated");
    }

    //create table to store recommendation results with product code and descriptions
    sql = "CREATE TABLE IF NOT EXISTS recommendations_descr (" +
        "recitem INT, " +
        "pref DOUBLE, " +
        "productcode STRING, " +
        "description STRING) " +
        "PARTITIONED BY (userid INT) " +
        "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' " +
        "LINES TERMINATED BY '\n'";
    stmt.executeQuery(sql);
    System.out.println("recommendations_descr created...");

    if(insertdata) {
    sql = "INSERT OVERWRITE TABLE recommendations_descr PARTITION (userid) " +
        "SELECT r.recitem, ROUND(r.pref,2) as pref, i.productcode, i.description, r.userid AS userid " +
        "FROM recommendations r JOIN items_descr_ext i " +
        "WHERE r.recitem = i.itemid " +
```

```
        "ORDER BY pref DESC";

    System.out.println("Populating recommendations_descr table...");
    stmt.executeQuery(sql);
    System.out.println("Table recommendations_descr is populated...");
    }

    //create table to store recommended items with counts to use for Top K item queries
    sql = "CREATE TABLE IF NOT EXISTS item_counts (" +
        "recitem INT, " +
        "productcode STRING, " +
        "description STRING, " +
        "count INT) " +
        "ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' " +
        "LINES TERMINATED BY '\n'";
    stmt.executeQuery(sql);
    System.out.println("item_counts created...");

    if(insertdata) {
    sql = "INSERT OVERWRITE TABLE item_counts " +
        "SELECT r.recitem, i.productcode, i.description, COUNT(r.recitem) as count " +
        "FROM recommendations r JOIN items_descr_ext i " +
        "WHERE r.recitem = i.itemid " +
        "GROUP BY r.recitem, i.productcode, i.description " +
        "ORDER BY count DESC";
    System.out.println("Populating item_counts table...");
    stmt.executeQuery(sql);
    System.out.println("Table item_counts is populated...");
    }
  }

    /*
     * Returns all table names in a ResultSet
     *
     * @param stmt  Statement
     * @param con   Connection to the master node
     * @return res  Query results as ResultSet
     */
    public static ResultSet showTables (Statement stmt, Connection con) throws SQLException
  {
        String sql = "show tables";
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        return res;
    }

    /*
     * Select from recommendations table for one user to view results
     *
     * @param user  User id as integer
     * @param stmt  Statement
     * @param con   Connection
     *
     * @return res  Query results as ResultSet
     */
    public ResultSet recommend (Integer user, Statement stmt, Connection con) throws
  SQLException {
        String sql = "SELECT recitem, pref FROM recommendations WHERE userid = " + user;
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        return res;
        }
```

```java
    /*
     * Method returns userid, recommended itemid, preference,
     * product code and product description
     * for a specific userid
     *
     * @param user  User id as integer
     * @param stmt  Statement
     * @param con   Connection
     *
     * @return res  Query results as ResultSet
     */
    public ResultSet recommendWithDesc (Integer user, Statement stmt, Connection con) throws SQLException {
        String sql = "SELECT * FROM recommendations_descr WHERE userid = " + user;
        ResultSet res = stmt.executeQuery(sql);
//      while (res.next()) {
//          System.out.println(res.getString(1) + '\t' +
//              res.getString(2));
        return res;
        }

    /*
     * Method returns items and preferences
     *
     * @param stmt  Statement
     * @param con   Connection
     *
     * @return res  Query results as ResultSet
     */
    public ResultSet recommend (Statement stmt, Connection con) throws SQLException {
        String sql = "SELECT recitem, pref FROM recommendations";
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
//      while (res.next()) {
//          System.out.println(res.getString(1) + '\t' +
//              res.getString(2));
        return res;
        }

    /*
     * Method returns userid, recommended itemid, preference,
     * product code and product description
     *
     * @param stmt  Statement
     * @param con   Connection
     *
     * @return res  Query results as ResultSet
     */
    public ResultSet recommendWithDesc (Statement stmt, Connection con) throws SQLException {
        String sql = "SELECT * FROM recommendations_descr";
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
//      while (res.next()) {
//          System.out.println(res.getString(1) + '\t' +
//              res.getString(2));
        return res;
        }

    /*
     * Method to run Top K items for a userid
     *
```

```
     * @param userid User Id as integer
     * @param stmt  Statement
     * @param con   Connection
     * @param topK  Integer for the number of most recommended items to return
     * @return res  Query results as ResultSet
     */
    public ResultSet recommendTopKItemsForUser
            (Integer userid, Statement stmt, Connection con, Integer topK)
            throws SQLException {
        String sql = "SELECT * FROM recommendations_descr " +
                     "WHERE userid = " + userid + " LIMIT " + topK;
//                   " ORDER BY pref DESC" + "LIMIT " + topK;
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        return res;
    }

    /*
     * Returns top K most recommended items for all users
     *
     * @param stmt  Statement
     * @param con   Connection
     * @param topK  Integer for the number of most recommended items to return
     *
     * @return res  Query results as ResultSet
     */
    public ResultSet TopKItemsRecommended
        (Statement stmt, Connection con, Integer topK)
        throws SQLException {
        String sql ="SELECT * FROM item_counts " +
                    "LIMIT " + topK;
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        return res;
    }
}
```

```java
/*
 * Recommendations class contains the main method to run the following queries:
 * 1. Top K recommended items
 * 2. Recommended items for a user, and
 * 3. Top K recommended items for a user
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */

package recommender;

import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class Recommendations {

    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

    public static void main(String[] args) throws SQLException {

        String server = "";
        ReadInput rd = new ReadInput();
        Integer query;
        Integer user;
        Integer k;
        ProcessRecommendations pr = new ProcessRecommendations();

        if (args.length > 0) {
            server = args[0];
        } else {
            System.out
                    .println("Required parameter missing - Master Node DNS Name");
            System.exit(1);
        }

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        Connection con = DriverManager.getConnection("jdbc:hive://" + server
                + ":10003/default");
        Statement stmt = con.createStatement();

        // check if tables exist; create them if they don't exist
        System.out.println("Checking if data tables exist");

        if (!tablesExist(pr.showTables(stmt, con))) {
            System.out
                    .println("Recommendation tables do not exist and will be created.");
            System.out.println("This may take a few minutes...");
            stmt.execute("set hive.exec.dynamic.partition=true");
            stmt.execute("set hive.exec.dynamic.partition.mode=nonstrict");
            stmt.execute("set hive.exec.max.dynamic.partitions=10000");
            stmt.execute("set hive.optimize.s3.query=true");
```

```java
            stmt.execute("set hive.base.inputformat=org.apache.hadoop.hive.ql.io.HiveInputFormat");
            stmt.execute("set hive.cli.print.header=true");
            pr.createTables(true, stmt, con);
        }

        query = rd.readQuerySelection();
        while (query == -1) {
            query = rd.readQuerySelection();
        }
        while (query > 0 && query <= 3) {
            // Recommended items for a user
            if (query == 2) {
                user = rd.readUser();
                if (user == -1) {
                    user = rd.readUser();
                }
                // get recommendations with descriptions for the user
                if (user == 0) {
                    query = rd.readQuerySelection();
                } else {
                    ResultSet recom = pr.recommendWithDesc(user, stmt, con);
                    while (recom.next()) {
                        System.out.println(recom.getString(1) + '\t'
                                + recom.getString(2) + '\t'
                                + recom.getString(3) + '\t'
                                + recom.getString(4));
                    }
                    // query = rd.readQuerySelection();
                }
            } // end of query option 2

            // Top K for a User
            if (query == 3) {
                user = rd.readUser();
                if (user == -1) {
                    user = rd.readUser();
                }

                if (user != 0) {
                    k = rd.readTopK();
                    if (k == -1) {
                        k = rd.readTopK();
                    }

                    if (k == 0 || user == 0) {
                        query = rd.readQuerySelection();
                    } else {
                        // get recommendations with descriptions for the user
                        ResultSet recom = pr.recommendTopKItemsForUser(user,
                                stmt, con, k);
                        while (recom.next()) {
                            System.out.println(recom.getString(1) + '\t'
                                    + recom.getString(2) + '\t'
                                    + recom.getString(3) + '\t'
                                    + recom.getString(4));
                        }
                        // query = rd.readQuerySelection(); // end of else
                        // option 3
                    }
                } else {
                    query = rd.readQuerySelection();
```

```
                }
            } // end of query option 3

            if (query == 1) {
                // get top k recommended items overall
                k = rd.readTopK();
                if (k == -1) {
                    k = rd.readTopK();
                }
                if (k == 0) {
                    query = rd.readQuerySelection();
                } else {
                    ResultSet recom = pr.TopKItemsRecommended(stmt, con, k);
                    while (recom.next()) {
                        System.out.println(recom.getString(1) + '\t'
                                + recom.getString(2) + '\t'
                                + recom.getString(3) + '\t'
                                + recom.getString(4));
                    }
                    // query = rd.readQuerySelection();
                }
            } // end of else option 1
        } // end while query not 0
    }

    /*
     * Method to check if required tables exist
     *
     * @param res ResultSet containing the list of existing Hive tables
     */
    public static Boolean tablesExist(ResultSet res) throws SQLException {
        Boolean recom = false;
        Boolean recom_d = false;
        Boolean item_cnt = false;

        while (res.next()) {
            if (res.getString(1).equals("recommendations")) {
                System.out.println(res.getString(1));
                System.out.println("recommendations exists");
                recom = true;
                System.out.println("recom = " + recom.toString());
            }
            if (res.getString(1).equals("recommendations_descr")) {
                System.out.println(res.getString(1));
                System.out.println("recommendations_descr exists");
                recom_d = true;
                System.out.println("recom_d = " + recom_d.toString());
            }
            if (res.getString(1).equals("item_counts")) {
                System.out.println(res.getString(1));
                System.out.println("item_counts exists");
                item_cnt = true;
                System.out.println("item_counts = " + recom_d.toString());
            }
        }
        if (recom && recom_d && item_cnt) {
            return true;
        } else
            return false;
    } // end tablesExist
} // end Recommendations
```

```java
/*
 * ReadInput class prompts for and reads user input from console and
 * returns input to the main class
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */
package recommender;

import java.util.Scanner;

public class ReadInput {

    /*
     * Method to prompt for and read user input from console
     *
     * @return  the integer corresponding to the query number user wants to run or -1
     */
    public Integer readQuerySelection() {
        Integer input = null;
        boolean intg = true;
        System.out.println("\nEnter 1, 2 or 3 to the query you want to run:");
        System.out.println("1 : Top K recommended items");
        System.out.println("2 : Recommended items for a user");
        System.out.println("3 : Top K recommended items for a user");
        System.out.println("Or any other key to exit \n");

        Scanner s = new Scanner(System.in);
        try {
            input = Integer.parseInt(s.next());
        } catch (NumberFormatException n) {
            System.out.println("Input must 1, 2 or 3");
            intg = false;
        }
        if (intg && input <= 3) {
            return input;
        } else
            return -1;
    } // end of read query selection method

    /*
     * Method to read user id from console
     *
     * @return  UserId to use as an input
     */
    public Integer readUser() {

        // prompt for userid
        System.out.print("Enter userid or 0 to exit: \n");

        Scanner s = new Scanner(System.in);

        Integer userid = null;
        Boolean intg = true;
        // read userid from the command-line
        try {
            userid = Integer.parseInt(s.next());
        } catch (NumberFormatException n) {
            System.out.println("Input must be an integer");
            intg = false;
        }
```

```java
        if (intg) {
            return userid;
        } else
            return -1;
    } // end of readUser method

    /*
     * Method to read Top K parameter from console
     *
     * @return  K as an integer to be used as the input to Top K query
     */
    public Integer readTopK() {

        // prompt for top k parameter
        System.out.print("Top K Parameter or 0 to exit: \n");

        Scanner s = new Scanner(System.in);

        Integer k = null;
        Boolean intg = true;
        // read userid from the command-line
        try {
            k = Integer.parseInt(s.next());
        } catch (NumberFormatException n) {
            System.out.println("Input must be an integer");
            intg = false;
        }
        if (intg) {
            return k;
        } else
            return -1;
    } // end of readTopK method

} // end of ReadInput class
```

```java
/*
 * RecomCSVFile converts the recommender engine output into a CSV file
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */
package recommender;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class RecomCSVFile {

    public static void main(String[] args) throws Exception {

        String server = "";
        String file = "c:/temp/recom_out.csv";
        String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
        ProcessRecommendations pr = new ProcessRecommendations();

        if (args.length > 0) {
            server = args[0];
        } else {
            System.out
                    .println("Required parameter missing - Master Node DNS Name");
            System.exit(1);
        }

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        Connection con = DriverManager.getConnection("jdbc:hive://" + server
                + ":10003/default");
        Statement stmt = con.createStatement();

        // check if tables exist; create them if they don't exist

        if (!tablesExist(pr.showTables(stmt, con))) {
            System.out
                    .println("Recommendation tables do not exist and will be created.");
            System.out.println("This may take a few minutes...");
            pr.createTables(true, stmt, con);
        }
        createRecomCSVFile(stmt, con, file);
    }

    /*
     * Method to check if required tables exist
     *
     * @param res ResultSet containing the list of existing Hive tables
     */
```

```java
    public static Boolean tablesExist(ResultSet res) throws SQLException {
        Boolean recom = false;
        Boolean recom_d = false;

        while (res.next()) {
            if (res.getString(1).equals("recommendations")) {
                System.out.println(res.getString(1));
                System.out.println("recommendations exists");
                recom = true;
                System.out.println("recom = " + recom.toString());
            }
            if (res.getString(1).equals("recommendations_descr")) {
                System.out.println(res.getString(1));
                System.out.println("recommendations_descr exists");
                recom_d = true;
                System.out.println("recom_d = " + recom_d.toString());
            }
        }
        if (recom && recom_d) {
            return true;
        } else
            return false;
    } // end tablesExist

    /*
     * Method creates CSV file
     *
     * @param stmt  Statement
     * @param con   Connection to the master node
     * @param filename  Name of the output file
     */
    public static void createRecomCSVFile(Statement stmt, Connection con,
            String filename) throws Exception {
        String sql = "SELECT userid, recitem, ROUND(pref,2) FROM recommendations ORDER BY userid";
        System.out.println("Running: " + sql);
        ResultSet res = stmt.executeQuery(sql);
        BufferedWriter out = new BufferedWriter(new FileWriter(filename));
        System.out.println("Writing result set to file");
        try {
            while (res.next()) {
                out.write(res.getString(1) + ',' + res.getString(2) + ','
                        + res.getString(3) + '\n');
            }
            System.out.println("Finished writing to " + filename);
        } catch (IOException e) {

        } finally {
            out.close();
        }
    } // end of createRecomCSVFile
}
```

```java
/*
 * RecomXMLFile converts the recommender engine output into a XML file
 *
 * @author Klavdiya Hammond
 *
 * @version April 20, 2013
 */

package recommender;

import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;

public class RecomXMLFile {

    public static void main(String[] args) throws Exception {

        String server = "";
        String file = "c:/temp/recom_out.xml";
        String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
        ProcessRecommendations pr = new ProcessRecommendations();

        if (args.length > 0) {
            server = args[0];
        } else {
            System.out
                    .println("Required parameter missing - Master Node DNS Name");
            System.exit(1);
        }

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        }

        Connection con = DriverManager.getConnection("jdbc:hive://" + server
                + ":10003/default");
        Statement stmt = con.createStatement();

        // check if tables exist; create them if they don't exist
        if (!tablesExist(pr.showTables(stmt, con))) {
            System.out
                    .println("Recommendation tables do not exist and will be created.");
            System.out.println("This may take a few minutes...");
            pr.createTables(true, stmt, con);
        }
        createRecomXMLFile(stmt, con, file);
    }

    /*
```

```
63      * Method to check if required tables exist
64      *
65      * @param res ResultSet containing the list of existing Hive tables
66      */
67
68     public static Boolean tablesExist(ResultSet res) throws SQLException {
69         Boolean recom = false;
70         Boolean recom_d = false;
71
72         while (res.next()) {
73             if (res.getString(1).equals("recommendations")) {
74                 System.out.println(res.getString(1));
75                 System.out.println("recommendations exists");
76                 recom = true;
77                 System.out.println("recom = " + recom.toString());
78             }
79             if (res.getString(1).equals("recommendations_descr")) {
80                 System.out.println(res.getString(1));
81                 System.out.println("recommendations_descr exists");
82                 recom_d = true;
83                 System.out.println("recom_d = " + recom_d.toString());
84             }
85         }
86         if (recom && recom_d) {
87             return true;
88         } else
89             return false;
90     } // end tablesExist
91
92     /*
93      * Method creates XML file
94      *
95      * @param stmt Statement
96      *
97      * @param con Connection to the master node
98      *
99      * @param filename Name of the output file
100     */
101    public static void createRecomXMLFile(Statement stmt, Connection con,
102            String filename) throws Exception {
103        String sql = "SELECT * FROM recommendations_descr";
104        System.out.println("Running: " + sql);
105        ResultSet res = stmt.executeQuery(sql);
106        DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory
107                .newInstance();
108        DocumentBuilder documentBuilder = documentBuilderFactory
109                .newDocumentBuilder();
110
111        String root = "recommendations";
112        Document document = documentBuilder.newDocument();
113        Element rootElement = document.createElement(root);
114        document.appendChild(rootElement);
115
116        Attr attr;
117        String u = "user";
118        String c = "product_code";
119        String d = "description";
120        String p = "preference";
121        String ui = "userid";
122        String ii = "itemid";
123
124        while (res.next()) {
```

```
            // create user element with userid and itemid attr
            Element user = document.createElement(u);
            rootElement.appendChild(user);
            // create recommended item attribute
            attr = document.createAttribute(ii);
            attr.setValue(res.getString(1));
            user.setAttributeNode(attr);
            // create userid attribute
            attr = document.createAttribute(ui);
            attr.setValue(res.getString(5));
            user.setAttributeNode(attr);

            // append preference element to user
            Element pref = document.createElement(p);
            pref.appendChild(document.createTextNode(res.getString(2)));
            user.appendChild(pref);

            // append product code to user
            Element code = document.createElement(c);
            code.appendChild(document.createTextNode(res.getString(3)));
            user.appendChild(code);

            // append product description to user
            Element desc = document.createElement(d);
            desc.appendChild(document.createTextNode(res.getString(4)));
            user.appendChild(desc);
        }

        TransformerFactory transformerFactory = TransformerFactory
                .newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(document);
        StreamResult result = new StreamResult(new File(filename));
        transformer.transform(source, result);
        System.out.println("Done. The following file has been created: "
                + filename);
    }
}
```

Appendix D – Random Forests Classifier

1. HiveQL queries to preprocess flat file
   (See following pages)

2. Commands to create random forests classifier model

   Generating dataset descriptor:
   ```
   hadoop jar core/target/mahout-core-0.7-job.jar
   org.apache.mahout.classifier.df.tools.Describe -p
   /mpropdata/train/000000_0 -f temp/forests/mprop2.info -d L 2 C N C N 6 C
   ```

   -p – path to the data file in HDFS
   -f – path where the dataset will be created in HDFS
   -d – provides description of the parameters stored in the file
   L = classification label
   C = categorical attribute
   N = numeric attribute
   '2 C' = two categorical attributes
   '6 C' = six categorical attributes

   Train the model – Build Forest of 200 trees
   ```
   hadoop jar examples/target/mahout-examples-0.7-job.jar
   org.apache.mahout.classifier.df.mapreduce.BuildForest -d
   /mpropdata/train/ -ds temp/forests/mprop2.info -oob -sl 5 -t 200 -mr
   ```

   -d – path to the data directory (may contain one or more files) in HDFS
   -ds – path to the dataset file created in previous step in HDFS
   -oob – out of bag error estimation
   -sl 5 – select 5 attributes at random when building trees
   -mr – use MapReduce implementation of Decision Forests
   -t – number of trees to build

   Test forest with the test data
   ```
   hadoop jar examples/target/mahout-examples-0.7-job.jar
   org.apache.mahout.classifier.df.mapreduce.TestForest -i
   /mpropdata/test/ -ds temp/forests/mprop2.info -m
   /user/hadoop/ob/forest.seq –a
   ```

   -i – path to the test data directory in HDFS
   -ds – path to the dataset file in HDFS
   -m – path to the Random Forests model in HDFS
   -a – prints confusion matrix and evaluates the model
   -mr – use MapReduce implementation

4. Maven project files (Java classes, pom.xml) and executable jar with dependencies file:
   a) pom.xml
   b) ClassifyRandomForests.java
   c) ClassifyRandomForestsInstance.java
   d) RandomForests.java
   e) ReadOutput.java
   f) forests-1.0-jar-with-dependencies.jar (packaged executable includes above listed files and all required dependencies)
   g) See create "mvn project for forests and run commands.txt" on how to either build the project from scratch or use the jar file provided

5. Data included

Note: programs work with data stored in Hadoop Distributed File System (HDFS)

a) Train data
b) Test data
c) Data to use for classification
d) Random Forests classification model (ready to be used)
e) Dataset file, which works with the above test/classification data and the model

```sql
--Klavdiya Hammond
--April 23, 2013
--update the LOCATION of the data used for training, testing and classification
--before running these scripts
--to run from file:  hive -f <file path>

--this is the ascii fixed length file
CREATE EXTERNAL TABLE IF NOT EXISTS MPROP_EXT (
RECORD STRING)
STORED AS TEXTFILE
LOCATION 's3://klav.project/mproptrain/';

--pull fields that will be used in the dm model; create and then populate the table
CREATE TABLE IF NOT EXISTS MPROP_TEMP (
DIR STRING,
STREET STRING,
STTYPE STRING,
CACLASS STRING,
CATOTAL STRING,
CAEXMTOTAL STRING,
REASON STRING,
BLDGTYPE STRING,
NRSTORIES STRING,
BASEMENT STRING,
ATTIC STRING,
NRUNITS STRING,
BLDGAREA STRING,
YRBUILT STRING,
FIREPLACE STRING,
AIRCONDIT STRING,
BEDROOMS STRING,
BATHS STRING,
POWDERROOM STRING,
GARAGETYPE STRING,
LOTAREA STRING,
ZONING STRING,
LANDUSE STRING,
LANDUSEGP STRING,
GEOTRACT STRING,
GEOBLOCK STRING,
GEOZIPCODE STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hadoop/mpropdata/temp/';

INSERT OVERWRITE TABLE MPROP_TEMP
SELECT
SUBSTRING(RECORD,37,1) AS DIR,
SUBSTRING(RECORD,38,18) AS STREET,
SUBSTRING(RECORD,56,2) AS STTYPE,
SUBSTRING(RECORD,58,1) AS CACLASS,
SUBSTRING(RECORD,78,9) AS CATOTAL,
SUBSTRING(RECORD,108,9) AS CAEXMTOTAL,
SUBSTRING(RECORD,182,3) AS REASON,
SUBSTRING(RECORD,382,9) AS BLDGTYPE,
```

```sql
SUBSTRING(RECORD,391,4) AS NRSTORIES,
SUBSTRING(RECORD,395,1) AS BASEMENT,
SUBSTRING(RECORD,396,1) AS ATTIC,
SUBSTRING(RECORD,397,3) AS NRUNITS,
SUBSTRING(RECORD,400,7) AS BLDGAREA,
SUBSTRING(RECORD,407,4) AS YRBUILT,
SUBSTRING(RECORD,412,1) AS FIREPLACE,
SUBSTRING(RECORD,413,1) AS AIRCONDIT,
SUBSTRING(RECORD,420,3) AS BEDROOMS,
SUBSTRING(RECORD,423,3) AS BATHS,
SUBSTRING(RECORD,426,3) AS POWDERROOM,
SUBSTRING(RECORD,432,2) AS GARAGETYPE,
SUBSTRING(RECORD,434,9) AS LOTAREA,
SUBSTRING(RECORD,445,7) AS ZONING,
SUBSTRING(RECORD,452,4) AS LANDUSE,
SUBSTRING(RECORD,456,2) AS LANDUSEGP,
SUBSTRING(RECORD,459,6) AS GEOTRACT,
SUBSTRING(RECORD,465,4) AS GEOBLOCK,
SUBSTRING(RECORD,469,9) AS GEOZIPCODE
FROM MPROP_EXT;

--data to train the model with the 10 major categories
CREATE TABLE IF NOT EXISTS MPROP_train2 (
VALUE STRING,
STREET STRING,
CACLASS STRING,
REASON STRING,
BLDGAREA BIGINT,
YRBUILT STRING,
LOTAREA BIGINT,
ZONING STRING,
LANDUSE STRING,
LANDUSEGP STRING,
GEOTRACT STRING,
GEOBLOCK STRING,
GEOZIPCODE STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hadoop/mpropdata/train2/';

INSERT OVERWRITE TABLE MPROP_train2
SELECT
CASE WHEN (CATOTAL+CAEXMTOTAL)<250000
    THEN '0-249K'
    WHEN (CATOTAL+CAEXMTOTAL)>=250000 AND (CATOTAL+CAEXMTOTAL)<=499999
    THEN '250K-499K'
    WHEN (CATOTAL+CAEXMTOTAL)>=500000 AND (CATOTAL+CAEXMTOTAL)<=749999
    THEN '500K-749K'
    WHEN (CATOTAL+CAEXMTOTAL)>=750000 AND (CATOTAL+CAEXMTOTAL)<=999999
    THEN '750K-999K'
    ELSE '1M+'
END AS VALUE,
CASE WHEN DIR=' '
    THEN  CONCAT(REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
    ELSE CONCAT(TRIM(DIR),'_',REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
```

```sql
END AS STREET,
CASE WHEN CACLASS=' '
    THEN '0' ELSE REGEXP_REPLACE(CACLASS, ' ', '')
END  AS CACLASS,
CASE WHEN REASON='   '
    THEN '0'
    ELSE TRIM(REASON)
END AS REASON,
CASE WHEN BLDGAREA='       '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(BLDGAREA AS BIGINT)
END AS BLDGAREA,
CASE WHEN YRBUILT='    '
    THEN '0000'
    ELSE REGEXP_REPLACE(YRBUILT, ' ', '')
END AS YRBUILT,
CASE WHEN LOTAREA='        '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(LOTAREA AS BIGINT)
END AS LOTAREA,
CASE WHEN ZONING='       '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(ZONING,' ','')
END AS ZONING,
CASE WHEN LANDUSE='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(LANDUSE, ' ', '')
END AS LANDUSE,
CASE WHEN LANDUSEGP='  '
    THEN 'XX'
    ELSE REGEXP_REPLACE(LANDUSEGP, ' ', '')
END AS LANDUSEGP,
CASE WHEN GEOTRACT='      '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOTRACT, ' ', '')
END AS GEOTRACT,
CASE WHEN GEOBLOCK='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOBLOCK, ' ', '')
END AS GEOBLOCK,
CASE WHEN SUBSTRING(GEOZIPCODE,1,5)='     '
    THEN '0'
    ELSE SUBSTRING(GEOZIPCODE,1,5)
END AS GEOZIPCODE
FROM MPROP_TEMP limit 20000;

--*********************************************************************
--********************TESTING*****************************************
--*********************************************************************
CREATE EXTERNAL TABLE IF NOT EXISTS MPROP_EXT_TEST (
RECORD STRING)
STORED AS TEXTFILE
LOCATION 's3://klav.project/mpropttest17k/';

--pull fields that will be used in the dm model create and then populate the table
CREATE TABLE IF NOT EXISTS MPROP_TEMP_test (
CACLASS STRING,
```

```
DIR STRING,
STREET STRING,
STTYPE STRING,
CATOTAL STRING,
CAEXMTOTAL STRING,
REASON STRING,
BLDGTYPE STRING,
NRSTORIES STRING,
BASEMENT STRING,
ATTIC STRING,
NRUNITS STRING,
BLDGAREA STRING,
YRBUILT STRING,
FIREPLACE STRING,
AIRCONDIT STRING,
BEDROOMS STRING,
BATHS STRING,
POWDERROOM STRING,
GARAGETYPE STRING,
LOTAREA STRING,
ZONING STRING,
LANDUSE STRING,
LANDUSEGP STRING,
GEOTRACT STRING,
GEOBLOCK STRING,
GEOZIPCODE STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hadoop/mpropdata/temptest/';

INSERT OVERWRITE TABLE MPROP_TEMP_test
SELECT
SUBSTRING(RECORD,58,1) AS CACLASS,
SUBSTRING(RECORD,37,1) AS DIR,
SUBSTRING(RECORD,38,18) AS STREET,
SUBSTRING(RECORD,56,2) AS STTYPE,
SUBSTRING(RECORD,78,9) AS CATOTAL,
SUBSTRING(RECORD,108,9) AS CAEXMTOTAL,
SUBSTRING(RECORD,182,3) AS REASON,
SUBSTRING(RECORD,382,9) AS BLDGTYPE,
SUBSTRING(RECORD,391,4) AS NRSTORIES,
SUBSTRING(RECORD,395,1) AS BASEMENT,
SUBSTRING(RECORD,396,1) AS ATTIC,
SUBSTRING(RECORD,397,3) AS NRUNITS,
SUBSTRING(RECORD,400,7) AS BLDGAREA,
SUBSTRING(RECORD,407,4) AS YRBUILT,
SUBSTRING(RECORD,412,1) AS FIREPLACE,
SUBSTRING(RECORD,413,1) AS AIRCONDIT,
SUBSTRING(RECORD,420,3) AS BEDROOMS,
SUBSTRING(RECORD,423,3) AS BATHS,
SUBSTRING(RECORD,426,3) AS POWDERROOM,
SUBSTRING(RECORD,432,2) AS GARAGETYPE,
SUBSTRING(RECORD,434,9) AS LOTAREA,
SUBSTRING(RECORD,445,7) AS ZONING,
SUBSTRING(RECORD,452,4) AS LANDUSE,
```

```sql
SUBSTRING(RECORD,456,2) AS LANDUSEGP,
SUBSTRING(RECORD,459,6) AS GEOTRACT,
SUBSTRING(RECORD,465,4) AS GEOBLOCK,
SUBSTRING(RECORD,469,9) AS GEOZIPCODE
FROM MPROP_EXT_test;

--data to train the model with the 10 major categories
CREATE TABLE IF NOT EXISTS MPROP_test2 (
VALUE STRING,
STREET STRING,
CACLASS STRING,
REASON STRING,
BLDGAREA BIGINT,
YRBUILT STRING,
LOTAREA BIGINT,
ZONING STRING,
LANDUSE STRING,
LANDUSEGP STRING,
GEOTRACT STRING,
GEOBLOCK STRING,
GEOZIPCODE STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hadoop/mpropdata/test2/';

INSERT OVERWRITE TABLE MPROP_test2
SELECT
CASE WHEN (CATOTAL+CAEXMTOTAL)<250000
    THEN '0-249K'
    WHEN (CATOTAL+CAEXMTOTAL)>=250000 AND (CATOTAL+CAEXMTOTAL)<=499999
    THEN '250K-499K'
    WHEN (CATOTAL+CAEXMTOTAL)>=500000 AND (CATOTAL+CAEXMTOTAL)<=749999
    THEN '500K-749K'
    WHEN (CATOTAL+CAEXMTOTAL)>=750000 AND (CATOTAL+CAEXMTOTAL)<=999999
    THEN '750K-999K'
    ELSE '1M+'
END AS VALUE,
CASE WHEN DIR=' '
    THEN  CONCAT(REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
    ELSE CONCAT(TRIM(DIR),'_',REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
END AS STREET,
CASE WHEN CACLASS=' '
    THEN '0' ELSE REGEXP_REPLACE(CACLASS, ' ', '')
END  AS CACLASS,
CASE WHEN REASON='   '
    THEN '0'
    ELSE TRIM(REASON)
END AS REASON,
CASE WHEN BLDGAREA='       '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(BLDGAREA AS BIGINT)
END AS BLDGAREA,
CASE WHEN YRBUILT='    '
    THEN '0000'
    ELSE REGEXP_REPLACE(YRBUILT, ' ', '')
```

```sql
END AS YRBUILT,
CASE WHEN LOTAREA='        '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(LOTAREA AS BIGINT)
END AS LOTAREA,
CASE WHEN ZONING='       '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(ZONING,' ','')
END AS ZONING,
CASE WHEN LANDUSE='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(LANDUSE, ' ', '')
END AS LANDUSE,
CASE WHEN LANDUSEGP='  '
    THEN 'XX'
    ELSE REGEXP_REPLACE(LANDUSEGP, ' ', '')
END AS LANDUSEGP,
CASE WHEN GEOTRACT='      '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOTRACT, ' ', '')
END AS GEOTRACT,
CASE WHEN GEOBLOCK='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOBLOCK, ' ', '')
END AS GEOBLOCK,
CASE WHEN SUBSTRING(GEOZIPCODE,1,5)='     '
    THEN '0'
    ELSE SUBSTRING(GEOZIPCODE,1,5)
END AS GEOZIPCODE
FROM MPROP_TEMP_test limit 10000;

--*********************************************************************
--********************TO CLASSIFY*************************************
--*********************************************************************
CREATE TABLE IF NOT EXISTS MPROP_class (
VALUE STRING,
STREET STRING,
CACLASS STRING,
REASON STRING,
BLDGAREA BIGINT,
YRBUILT STRING,
LOTAREA BIGINT,
ZONING STRING,
LANDUSE STRING,
LANDUSEGP STRING,
GEOTRACT STRING,
GEOBLOCK STRING,
GEOZIPCODE STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION '/user/hadoop/mpropdata/class/';

INSERT OVERWRITE TABLE MPROP_class
SELECT
'-' AS VALUE,
```

```sql
CASE WHEN DIR=' '
    THEN  CONCAT(REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
    ELSE CONCAT(TRIM(DIR),'_',REGEXP_REPLACE(TRIM(STREET), ' ', '_'),'_',TRIM(STTYPE))
END AS STREET,
CASE WHEN CACLASS=' '
    THEN '0' ELSE REGEXP_REPLACE(CACLASS, ' ', '')
END  AS CACLASS,
CASE WHEN REASON='   '
    THEN '0'
    ELSE TRIM(REASON)
END AS REASON,
CASE WHEN BLDGAREA='       '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(BLDGAREA AS BIGINT)
END AS BLDGAREA,
CASE WHEN YRBUILT='    '
    THEN '0000'
    ELSE REGEXP_REPLACE(YRBUILT, ' ', '')
END AS YRBUILT,
CASE WHEN LOTAREA='        '
    THEN CAST(0 AS BIGINT)
    ELSE CAST(LOTAREA AS BIGINT)
END AS LOTAREA,
CASE WHEN ZONING='       '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(ZONING,' ','')
END AS ZONING,
CASE WHEN LANDUSE='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(LANDUSE, ' ', '')
END AS LANDUSE,
CASE WHEN LANDUSEGP='  '
    THEN 'XX'
    ELSE REGEXP_REPLACE(LANDUSEGP, ' ', '')
END AS LANDUSEGP,
CASE WHEN GEOTRACT='      '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOTRACT, ' ', '')
END AS GEOTRACT,
CASE WHEN GEOBLOCK='    '
    THEN 'XXXX'
    ELSE REGEXP_REPLACE(GEOBLOCK, ' ', '')
END AS GEOBLOCK,
CASE WHEN SUBSTRING(GEOZIPCODE,1,5)='     '
    THEN '0'
    ELSE SUBSTRING(GEOZIPCODE,1,5)
END AS GEOZIPCODE
FROM MPROP_TEMP_test;
```

```
#Klavdiya Hammond
#April 27, 2013
#Commands to create random forests classifier model
#using Apache Mahout libraries
#
#Used with the City of Milwaukee Master Property Record data available at:
#http://city.milwaukee.gov/DownloadTabularData3496.htm
#
#Note: data has to be preprocessed and stored in HDFS directories (see mprop-hive-queries.sh)
#
#Step 1:
#Generate dataset descriptor:
hadoop jar core/target/mahout-core-0.7-job.jar org.apache.mahout.classifier.df.tools.Describe
 -p /mpropdata/train/000000_0 -f temp/forests/mprop2.info -d L 2 C N C N 6 C
#Parameters:
#-p – path to the data file in HDFS
#-f – path where the dataset will be created in HDFS
#-d – provides description of the parameters stored in the file
#L = classification label
#C = categorical attribute
#N = numeric attribute
#'2 C' = two categorical attributes
#'6 C' = six categorical attributes
#
#
#Step 2:
#Train the model – Build Forest of 200 trees
hadoop jar examples/target/mahout-examples-0.7-job.jar
org.apache.mahout.classifier.df.mapreduce.BuildForest -d /mpropdata/train/ -ds temp/forests/
mprop2.info -oob -sl 5 -t 200 -mr
#Parameters:
#-d – path to the data directory (may contain one or more files) in HDFS
#-ds – path to the dataset file created in previous step in HDFS
#-oob – out of bag error estimation
#-sl 5 – select 5 attributes at random when building trees
#-mr – use MapReduce implementation of Decision Forests
#-t – number of trees to build
#
#
#Step 3:
#Test forest with the test data
hadoop jar examples/target/mahout-examples-0.7-job.jar
org.apache.mahout.classifier.df.mapreduce.TestForest -i /mpropdata/test/ -ds temp/forests/
mprop2.info -m /user/hadoop/ob/forest.seq –a
#Parameters:
#-i – path to the test data directory in HDFS
#-ds – path to the dataset file in HDFS
#-m – path to the Random Forests model in HDFS
#-a – prints confusion matrix and evaluates the model
#-mr – use MapReduce implementation
```

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.klav.app</groupId>
    <artifactId>forests</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>my-app</name>
    <url>http://maven.apache.org</url>
    <parent>
        <artifactId>mahout</artifactId>
        <groupId>org.apache.mahout</groupId>
        <version>0.7</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>com.klav.app.ClassifyRandomForests</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id> <!-- this is used for inheritance merges -->
                        <phase>package</phase> <!-- bind to the packaging phase -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <encoding>UTF-8</encoding>
                    <source>1.6</source>
                    <target>1.6</target>
                    <optimize>true</optimize>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-dependency-plugin</artifactId>
                <executions>
                    <execution>
                        <id>copy-dependencies</id>
```

```
                        <phase>prepare-package</phase>
                        <goals>
                            <goal>copy-dependencies</goal>
                        </goals>
                        <configuration>
                            <!-- configure the plugin here -->
                            <outputDirectory>${project.build.directory}/lib</outputDirectory>
                            <overWriteReleases>false</overWriteReleases>
                            <overWriteSnapshots>false</overWriteSnapshots>
                            <overWriteIfNewer>true</overWriteIfNewer>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>test-jar</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <archive>
                        <manifest>
                            <addClasspath>true</addClasspath>
                            <classpathPrefix>lib/</classpathPrefix>
                            <mainClass>theMainClass</mainClass>
                        </manifest>
                    </archive>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-examples</artifactId>
            <version>0.7</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-core</artifactId>
            <version>0.7</version>
        </dependency>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-math</artifactId>
            <version>0.7</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.6.0</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-jcl</artifactId>
            <version>1.6.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-core</artifactId>
            <version>1.0.3</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>14.0.1</version>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.1</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-integration</artifactId>
            <version>0.7</version>
        </dependency>
        <dependency>
            <groupId>org.apache.mahout</groupId>
            <artifactId>mahout-examples</artifactId>
            <version>0.7</version>
        </dependency>
    </dependencies>
</project>
```

```java
/*
 * ClassifyRandomForests class contains the main method
 * to classify new data from file with an existing Random Forests
 * model and dataset file (dataset file describes the file layout
 * and contains possible category labels)
 * <p>
 * Classifier can be run in either MapReduce or Sequential mode
 *
 * @author Klavdiya Hammond
 * @version April 22, 2013
 *
 * Classes in this package use Apache Mahout library, therefore
 * Apache License notice is included herewith.
 * <p>
 *
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.klav.app;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.conf.Configuration;
import com.klav.app.RandomForests;

public class ClassifyRandomForests {

    /*
     * Main method Runtime arguments are (all file and directories need to be
     * saved in HDFS): String dPath = path to the data directory (data to be
     * classified, use '-' instead of Label) String dsPath = path to the dataset
     * file String mPath = path to the Random Forests model file String oPath =
     * path to the output directory boolean mr = enter 'true' or 'mr' to run
     * program in MapReduce mode If mr arguement is blank, classifier will run
     * in Sequential mode
     *
     * Sequential mode will create output file in the following format:
     * Predicted category followed by the data instance
     *
     * MapReduce mode produces output as listing of indexes corresponding to
     * predicted categories in the dataset - one for each data line in the input
     * file. <p> Run ReadMROutput to merge the list of indexes to the input
     * file.
     */

    public static void main(String[] args) throws IOException, Exception,
            InterruptedException {

```

```
        // conf, hdfs, String dPath, String dsPath, String mPath, String oPath,
        // boolean mr

        Configuration conf;
        conf = new Configuration();
        conf.addResource(new Path("/home/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/hdfs-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/mapred-site.xml"));

        FileSystem hdfs = FileSystem.get(conf);

        Path dPath = hdfs.makeQualified(new Path(args[0]));
        // make sure the data exists
        if (!hdfs.exists(dPath)) {
            throw new IllegalArgumentException("The Data path does not exist");
        }

        Path dsPath = hdfs.makeQualified(new Path(args[1]));
        // make sure the data set exists
        if (!hdfs.exists(dsPath)) {
            throw new IllegalArgumentException(
                    "The Data set path does not exist");
        }

        Path mPath = hdfs.makeQualified(new Path(args[2]));
        // make sure the decision forest exists
        if (!hdfs.exists(mPath)) {
            throw new IllegalArgumentException("The forest path does not exist");
        }

        // make sure the output file does not exist
        if (args[3] != null) {
            if (hdfs.exists(new Path(args[3]))) {
                throw new IllegalArgumentException("Output path already exists");
            }
        }
        Path oPath = hdfs.makeQualified(new Path(args[3]));

        // determine if mapreduce or sequential
        String mr = "";
        boolean mapreduce = false;

        if (args.length > 4) {
            mr = args[4].toString();
            if (mr.equals("true") || mr.equals("mr")) {
                mapreduce = true;
            } else {
                mapreduce = false;
            }
        }

        // System.out.println(mr + "\n");
        // System.out.println(mapreduce + "\n");

        RandomForests tf = new RandomForests();

        // run classifier
        tf.run(conf, hdfs, dPath, dsPath, mPath, oPath, mapreduce);
    }
}
```

```java
/*
 * ClassifyRandomForestsInstance class contains the main method
 * to classify one data instance at a time with an existing Random Forests
 * model and dataset file (dataset file describes the file layout
 * and contains possible category labels)
 * <p>
 * Classifier runs in Sequential mode. Output (predicted class) is printed to the console.
 *
 * @author Klavdiya Hammond
 * @version April 22, 2013
 *
 * Classes in this package use Apache Mahout library, therefore
 * Apache License notice is included herewith.
 * <p>
 *
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.klav.app;

import org.apache.mahout.classifier.df.mapreduce.Classifier;
import java.io.File;
import java.io.IOException;
import java.util.Random;
import com.google.common.base.Preconditions;
import com.google.common.io.Closeables;
import com.google.common.cache.CacheBuilder;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.conf.Configuration;
import org.apache.mahout.classifier.df.data.Dataset;
import org.apache.mahout.common.RandomUtils;
import org.apache.mahout.classifier.df.DecisionForest;
import org.apache.mahout.classifier.df.mapreduce.Classifier;
import org.apache.mahout.classifier.df.data.DataConverter;
import org.apache.mahout.classifier.df.data.Dataset;
import org.apache.mahout.classifier.df.data.Instance;

public class ClassifyRandomForestsInstance {

    // private static final Logger log =
    // LoggerFactory.getLogger(ClassifyRandomForests.class);
    /*
     * Main method Runtime arguments are (all file and directories need to be
     * saved in HDFS): String modelPath = path to the Random Forests model file
     * String datasetPath = path to the dataset file String input = data input
     * (coma separated values, use '-' instead of label)
     */
    public static void main(String[] args) throws IOException, Exception,
```

```
            InterruptedException {

        Configuration conf = new Configuration();
        conf.addResource(new Path("/home/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/hdfs-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/mapred-site.xml"));

        FileSystem hdfs = FileSystem.get(conf);

        // ds m input
        Path datasetPath; // path to data description
        Path modelPath; // path where the forest is stored

        datasetPath = hdfs.makeQualified(new Path(args[0]));
        // make sure the data set exists
        if (!hdfs.exists(datasetPath)) {
            throw new IllegalArgumentException(
                    "The Data set path does not exist");
        }

        modelPath = hdfs.makeQualified(new Path(args[1]));
        // make sure the decision forest exists
        if (!hdfs.exists(modelPath)) {
            throw new IllegalArgumentException("The forest path does not exist");
        }

        String input = args[2];

        // load forest model
        DecisionForest forest = DecisionForest.load(conf, modelPath);

        // load the dataset
        Dataset dataset = Dataset.load(conf, datasetPath);

        DataConverter converter = new DataConverter(dataset);

        Random rng = RandomUtils.getRandom();

        Instance instance = converter.convert(input);
        double prediction = forest.classify(dataset, rng, instance);

        System.out.println("Predicted label : "
                + dataset.getLabelString(prediction)); // KH write label
        System.out.println("Data string read : " + input.substring(2)); // print
                                                                        // out
                                                                        // data,
                                                                        // exclude
                                                                        // 'missing'
                                                                        // label
    }
}
```

```java
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.klav.app;

import java.io.IOException;
import java.util.Collection;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.Arrays;

import com.google.common.collect.Lists;
import com.google.common.io.Closeables;
import org.apache.commons.cli2.CommandLine;
import org.apache.commons.cli2.Group;
import org.apache.commons.cli2.Option;
import org.apache.commons.cli2.OptionException;
import org.apache.commons.cli2.builder.ArgumentBuilder;
import org.apache.commons.cli2.builder.DefaultOptionBuilder;
import org.apache.commons.cli2.builder.GroupBuilder;
import org.apache.commons.cli2.commandline.Parser;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.mahout.common.CommandLineUtil;
import org.apache.mahout.common.RandomUtils;
import org.apache.mahout.common.commandline.DefaultOptionCreator;
import org.apache.mahout.classifier.df.DFUtils;
import org.apache.mahout.classifier.df.DecisionForest;
import org.apache.mahout.classifier.RegressionResultAnalyzer;
import org.apache.mahout.classifier.ResultAnalyzer;
import org.apache.mahout.classifier.ClassifierResult;
import org.apache.mahout.classifier.df.mapreduce.Classifier;
import org.apache.mahout.classifier.df.data.DataConverter;
import org.apache.mahout.classifier.df.data.Dataset;
import org.apache.mahout.classifier.df.data.Instance;
import org.apache.mahout.common.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.apache.mahout.classifier.df.mapreduce.TestForest;

/*
```

```
 * RandomForests class is a modified version of the
 * org.apache.mahout.cf.classifier.df.mapreduce.TestForest class
 * It is used to classify new data using a previously built Decision Forest model
 *
 * @param
 */
public class RandomForests {

    private static final Logger log = LoggerFactory
            .getLogger(RandomForests.class);

    private boolean useMapreduce; // use the mapreduce classifier ?

    private Path dataPath; // test data path

    private Path datasetPath; // dataset path

    private Path modelPath; // path where the forest is stored

    private Path outputPath; // path to predictions file, if null do not output
                             // the predictions

    /*
     * Run classifier with parameters Configuration conf, FileSystem hdfs, Path
     * dataName, Path datasetName, Path modelName, Path outputName, boolean mr
     *
     * @param conf Configuration
     *
     * @param hdfs FileSystem
     *
     * @param dataName path to the directory containing data to be classified
     *
     * @param datasetName path to the dataset file
     *
     * @param modelName path to the Decision Forest file
     *
     * @param outputName path to the output directory
     *
     * @param mr boolean specifying if mapreduce should be used
     */
    public void run(Configuration conf, FileSystem hdfs, Path dataName,
            Path datasetName, Path modelName, Path outputName, boolean mr)
            throws IOException, ClassNotFoundException, InterruptedException {

        if (log.isDebugEnabled()) {
            log.debug("input     : {}", dataName);
            log.debug("dataset   : {}", datasetName);
            log.debug("model     : {}", modelName);
            log.debug("output    : {}", outputName);
            log.debug("mapreduce : {}", useMapreduce);
        }

        dataPath = dataName;
        datasetPath = datasetName;
        modelPath = modelName;
        outputPath = outputName;
        useMapreduce = mr;

        testForest(conf, hdfs);
    }

    private void testForest(Configuration conf, FileSystem hdfs)
```

```
            throws IOException, ClassNotFoundException, InterruptedException {

        if (useMapreduce) {
            mapreduce(conf, hdfs);
        } else {
            sequential(conf, hdfs);
        }

    }

    private void mapreduce(Configuration conf, FileSystem hdfs)
            throws ClassNotFoundException, IOException, InterruptedException {
        if (outputPath == null) {
            throw new IllegalArgumentException(
                    "You must specify the ouputPath when using the mapreduce implementation");
        }

        Classifier classifier = new Classifier(modelPath, dataPath,
                datasetPath, outputPath, conf);

        classifier.run();
    }

    private void sequential(Configuration conf, FileSystem hdfs)
            throws IOException {

        log.info("Loading the forest...");
        DecisionForest forest = DecisionForest.load(conf, modelPath);

        if (forest == null) {
            log.error("No Decision Forest found!");
            return;
        }

        // load the dataset
        Dataset dataset = Dataset.load(conf, datasetPath);
        DataConverter converter = new DataConverter(dataset);

        log.info("Sequential classification...");
        long time = System.currentTimeMillis();

        Random rng = RandomUtils.getRandom();

        List<double[]> resList = Lists.newArrayList();
        if (hdfs.getFileStatus(dataPath).isDir()) {
            // the input is a directory of files
            testDirectory(conf, hdfs, outputPath, converter, forest, dataset,
                    resList, rng);
        } else {
            // the input is one single file
            testFile(conf, hdfs, dataPath, outputPath, converter, forest,
                    dataset, resList, rng);
        }

        time = System.currentTimeMillis() - time;
        log.info("Classification Time: {}", DFUtils.elapsedTime(time));
    }

    private void testDirectory(Configuration conf, FileSystem hdfs,
            Path outPath, DataConverter converter, DecisionForest forest,
            Dataset dataset, Collection<double[]> results, Random rng)
```

```
            throws IOException {
        Path[] infiles = DFUtils.listOutputFiles(hdfs, dataPath);

        for (Path path : infiles) {
            log.info("Classifying : {}", path);
            Path outfile = outPath != null ? new Path(outPath, path.getName())
                    .suffix(".out") : null;
            testFile(conf, hdfs, path, outfile, converter, forest, dataset,
                    results, rng);
        }
    }

    private void testFile(Configuration conf, FileSystem hdfs, Path inPath,
            Path outPath, DataConverter converter, DecisionForest forest,
            Dataset dataset, Collection<double[]> results, Random rng)
            throws IOException {
        // create the predictions file
        FSDataOutputStream ofile = null;

        if (outPath != null) {
            ofile = hdfs.create(outPath);
        }

        FSDataInputStream input = hdfs.open(inPath);
        try {
            Scanner scanner = new Scanner(input, "UTF-8");

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                if (line.isEmpty()) {
                    continue; // skip empty lines
                }

                Instance instance = converter.convert(line);
                double prediction = forest.classify(dataset, rng, instance);

                if (ofile != null) {
                    // ofile.writeChars(Double.toString(prediction)); // write
                    // the prediction
                    ofile.writeChars(dataset.getLabelString(prediction) + ", "); // KH
                                                                                 //
write
                                                                                 //
label
                    ofile.writeChars(line.substring(2)); // KH write the data,
                                                         // exclude 'missing'
                                                         // label
                    ofile.writeChar('\n');
                }

                results.add(new double[] { dataset.getLabel(instance),
                        prediction });
            }

            scanner.close();
        } finally {
            Closeables.closeQuietly(input);
        }
    }
}
```

```java
/*
 * Program merges the input and output files of MapReduce
 * Random Forests classifier line by line,
 * writing the predicted category label followed by the
 * input data to a new file.
 * <p>
 * This code can be used to interpret the output of either MapReduce
 * or sequential implementation of the Random Forests algorithm
 *
 * @author Klavdiya Hammond
 * @version April 22, 2013
 *
 * Classes in this package use Apache Mahout library, therefore
 * Apache License notice is included herewith.
 * <p>
 *
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.klav.app;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.mahout.classifier.df.data.Dataset;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.mahout.classifier.df.DFUtils;

public class ReadOutput {

    // public static void mergeFiles(File dataDir, File resDir, File mergedFile)
    // throws IOException {
    public static void main(String[] args) throws IOException {
        // arguments:
        // String datasetPath, File dataDir, File resDir, File mergedFile
        Configuration conf = new Configuration();
        conf.addResource(new Path("/home/hadoop/conf/core-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/hdfs-site.xml"));
        conf.addResource(new Path("/home/hadoop/conf/mapred-site.xml"));

        FileSystem hdfs = FileSystem.get(conf);

        Path datasetPath = hdfs.makeQualified(new Path(args[0]));
        // make sure the data set exists
```

```java
        if (!hdfs.exists(datasetPath)) {
            throw new IllegalArgumentException(
                    "The Data set path does not exist");
        }

        Dataset dataset = Dataset.load(conf, datasetPath);
        String[] labels = dataset.labels();
        // System.out.println("length of array " + labels.length);
        // for (int l=0; 0<labels.length-1; l++) {
        // System.out.println(l+" "+labels[l]);
        // }

        Path dataDir = hdfs.makeQualified(new Path(args[1]));
        if (!hdfs.exists(dataDir)) {
            System.out.println("Data directory does not exist");
        }

        Path resDir = hdfs.makeQualified(new Path(args[2]));
        // make sure results directory exists
        if (!hdfs.exists(resDir)) {
            throw new IllegalArgumentException(
                    "The results path does not exist");
        }

        Path mergedFile = hdfs.makeQualified(new Path(resDir.toString()
                + "/merged.out"));
        // if (mergedFile.exists()){
        // mergedFile.delete();
        // }
        // mergedFile.createNewFile();
        System.out.println("Output file name is " + mergedFile.toString());

        // Path[] dDirs = dataDir.listFiles();
        Path[] dDirs = DFUtils.listOutputFiles(hdfs, dataDir);
        // Path[] rDirs = resDir.listFiles();
        Path[] rDirs = DFUtils.listOutputFiles(hdfs, resDir);

        FSDataInputStream dataStream = null;
        FSDataInputStream resStream = null;

        Scanner dScanner = null;
        Scanner rScanner = null;

        // FSDataOutputStream output = new FSDataOutputStream(mergedFile, true);
        FSDataOutputStream output = hdfs.create(mergedFile);

        for (int i = 0; i < dDirs.length; i++) {
            System.out.println("dataFile being read is " + i
                    + dDirs[i].toString());
            System.out.println("resFile being read is " + i
                    + rDirs[i].toString());

            // dataStream = new FSDataInputStream(dDirs[i]);
            dataStream = hdfs.open(dDirs[i]);
            resStream = hdfs.open(rDirs[i]);

            dScanner = new Scanner(dataStream, "UTF-8");
            rScanner = new Scanner(resStream, "UTF-8");

            while (dScanner.hasNextLine()) {

                String dLine = dScanner.nextLine();
```

```
                if (dLine.isEmpty()) {
                    continue; // skip empty lines
                }
                String rLine = rScanner.nextLine();
                // System.out.println("rLine read: " + r);

                // String label =
                // dataset.getLabelString(Double.parseDouble(rLine.trim()));
                String label = "unknown";
                String l = rLine.trim().substring(0, 1);
                if (!l.equals("-")) {
                    // System.out.println(l + " " + rLine);
                    label = labels[Integer.parseInt(l)];
                }
                String merged = label + dLine.substring(1) + "\n";
                output.writeChars(merged);
            }
            output.close();
            dScanner.close();
            rScanner.close();
        }
    }
}
```

References

1. Daniel J. Abadi. Data Management in the Cloud: Limitations and Opportunities. ftp://ftp.research.microsoft.com/pub/debull/A09mar/abadi.pdf
2. Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.
3. Apache Hadoop Wiki: http://wiki.apache.org/hadoop/
4. Apache Hadoop Wiki http://wiki.apache.org/hadoop/PoweredBy
5. Apache Hive http://hive.apache.org/
6. Apache Hive Wiki https://cwiki.apache.org/confluence/display/Hive/Home
7. Jeffrey Dean and Sanjay Ghemawat. 2010. MapReduce: a flexible data processing tool. Commun. ACM 53, 1 (January 2010), 72-77.
8. Aravind Menon. Big Data @ Facebook. MBDS'12, September 21, 2012, San Jose, California, USA.
9. Avrilia Floratou, Nikhil Teletia, David J. DeWitt, Jignesh M. Patel, and Donghui Zhang. 2012. Can the elephants handle the NoSQL onslaught?. Proc. VLDB Endow. 5, 12 (August 2012), 1712-1723.
10. Anja Gruenheid, Edward Omiecinski, and Leo Mark. 2011. Query optimization using column statistics in hive. In Proceedings of the 15th Symposium on International Database Engineering & Applications (IDEAS '11). ACM, New York, NY, USA, 97-105.
11. Vinayak Borkar, Michael J. Carey, and Chen Li. 2012. Inside "Big Data management": ogres, onions, or parfaits?. In Proceedings of the 15th International Conference on Extending Database Technology (EDBT '12), Elke Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari (Eds.). ACM, New York, NY, USA, 3-14.
12. Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. 2012. The HaLoop approach to large-scale iterative data analysis. The VLDB Journal 21, 2 (April 2012), 169-190.
13. Amol Ghoting, Prabhanjan Kambadur, Edwin Pednault, and Ramakrishnan Kannan. 2011. 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). ACM, New York, NY, USA, 334-342.
14. Carlos Ordonez and Javier García-García. 2010. Database systems research on data mining. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 1253-1254.

15. Jimmy Lin and Alek Kolcz. 2012. Large-scale machine learning at twitter. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 793-804.
16. https://github.com/tdunning/pig-vector
17. Jonathan Tancer. Aparna S. Varde. The Deployment of MML for Data Analytics over the Cloud. ICDM-11 Workshops, December 2011, Vancouver, Canada.
18. Michael Stonebraker, Daniel Abadi, David J. DeWitt, Sam Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. 2010. MapReduce and parallel DBMSs: friends or foes?. Commun. ACM 53, 1 (January 2010), 64-71.
19. Shireesha Chandra and Aparna S. Varde. Data Management in the Cloud with Hive and SQL. Master's Project.
20. Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal. 2012. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. In Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12). ACM, New York, NY, USA, 85-94.
21. Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. 2012. Apriori-based frequent itemset mining algorithms on MapReduce. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC '12). ACM, New York, NY, USA, , Article 76 , 8 pages.
22. Wei Yin, Yogesh Simmhan, and Viktor K. Prasanna. 2012. Scalable regression tree learning on Hadoop using OpenPlanet. In Proceedings of third international workshop on MapReduce and its Applications Date (MapReduce '12). ACM, New York, NY, USA, 57-64.
23. Sasi K. Pitchaimalai, Carlos Ordonez, and Carlos Garcia-Alvarado. 2010. Comparing SQL and MapReduce to compute Naive Bayes in a single table scan. In Proceedings of the second international workshop on Cloud data management (CloudDB '10). ACM, New York, NY, USA, 9-16.
24. Mailing list archives: http://mail-archives.apache.org/mod_mbox/hive-user/
25. Mailing list archives: http://mail-archives.apache.org/mod_mbox/mahout-user/
26. Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman. Mahout in Action. Manning Publications. 2012.
27. Edward Capriolo, Dean Wampler, and Jason Rutherglen. Programming Hive. O'Reilly Media, Inc. 2012.
28. City of Milwaukee. Master Property Record, Download Tubular Data http://city.milwaukee.gov/DownloadTabularData3496.htm

29. Apache Mahout http://mahout.apache.org/

30. Apache Maven Project http://maven.apache.org/

31. Maven Central Repository Search Engine http://search.maven.org/

32. Apache Hive Language Manual https://cwiki.apache.org/confluence/display/Hive/LanguageManual

33. Apache JDBC Driver https://cwiki.apache.org/Hive/hivejdbcinterface.html

34. Amazon Web Services Documentation http://aws.amazon.com/documentation/

35. Apache License 2.0 http://www.apache.org/licenses/LICENSE-2.0

36. Hadoop Shell Commands http://hadoop.apache.org/docs/r0.18.3/hdfs_shell.html