Scalable Machine Learning with Mahout

Advanced Analytics with Machine Learning and Hadoop: Week Four Lecture

Machine learning computations aim to derive predictive models from current and historical data. The inherent premise is that a learned algorithm will improve with more training or experience, and in particular, machine learning algorithms can achieve extremely effective results for very narrow domains using models trained from large data sets[1].

As a result, computations of scale are implicated in most machine learning algorithms. For this reason, machine learning computations are well suited to a distributed computing paradigm, like Hadoop, in order to leverage large training sets to produce meaningful results. This lesson introduces the open source machine learning library, Mahout, which includes a number of machine learning algorithms and tools that are specifically intended to operate efficiently on massively-scaled data.

Table of Contents

[1] Benjamin Bengfort (April 2014). Introduction to Analytics with Hadoop: Lesson 4. Statistics.com.

Mahout – Scalable Machine Learning on Hadoop

Mahout is an open-source machine learning library that focuses on three key areas of machine learning, called the 3 C's of machine learning:

1. **Collaborative Filtering** – also known as recommender engines, which produce recommendations based on past behavior, preferences, or similarities to known entities/users
2. **Clustering** – also known as unsupervised learning, which groups data into clusters based on similar characteristics
3. **Classification** – also known as supervised learning, which learns from a supervised training set and assigns a category to unclassified items based on that training set

Mahout actually started in 2008 as a subproject of Apache Lucene, an open-source indexing and search library[2]. The Lucene project bundled several of the machine learning techniques it employed in its implementations of search, text mining, and information-retrieval, and later spun off Mahout into its own independent top-level Apache project in 2010.

[2] Apache Mahout: https://mahout.apache.org/

Since then, Mahout has significantly expanded its core algorithms particularly for collaborative filtering, and started work on a Scala-based DSL for linear algebraic oeprations called Mahout Scala. Most notably in April 2014 the Mahout community announced that it will no longer accept MapReduce-based algorithms into its library, although it will continue to keep and maintain the existing MapReduce algorithms in the codebase[3]. While the Mahout project seems to be decoupling the codebase from Hadoop, many of the distributed algorithms in the core library still employ Hadoop and MapReduce; we will be focusing on these algorithms for the scope of this lesson, but we encourage you to browse the list of Mahout Algorithms to learn about Mahout's other algorithms in detail and see which algorithms a single-machine, MapReduce, or the Apache Spark engine.

Installation

Mahout is a Java-based library and requires Java 1.6+, as well as Hadoop, if using any MapReduce-based Mahout algorithms. In this lesson, we will invoke the Mahout libraries “out-of-the-box" from the Mahout command-line, but in a practical scenario you would want to import the required Mahout libraries in the context of a larger application and would thus likely use a Java IDE like Eclipse or IntelliJ to develop and compile your code.

[3] Apache Mahout News “April 2014”: https://mahout.apache.org/

Let's start by downloading the most recent stable release of Mahout from the list of Apache Mahout Download Mirrors (currently at version 0.9). Unpack the downloaded Mahout distribution, and move it to the desired run location (/srv in this example) creating a symlink for convenience:

```
~$ wget http://apache.osuosl.org/mahout/0.9/mahout-distribution-
0.9.tar.gz
~$ sudo mv mahout-distribution-0.9.tar.gz /srv/ && cd /srv
/srv$ sudo tar -xvf mahout-distribution-0.9.tar.gz
/srv$ sudo chown -R analyst:analyst mahout-distribution-0.9/
/srv$ sudo ln -s $(pwd)/mahout-distribution-0.9 $(pwd)/mahout
```

Ensure that the JAVA_HOME environment variable is set to your Java distribution and while we're at it, let's add the following environment variables to our bash profile for convenience:

```
export JAVA_HOME=/usr/lib/jvm/java-7-oracle

export MAHOUT_HOME=/srv/mahout
export PATH=$PATH:$MAHOUT_HOME/bin
```

Adding Mahout to a Maven Project

Alternatively, Mahout is also available via a maven repository under the group id `org.apache.mahout`. If you would like to import the latest release of mahout into a Maven Java project, add the following dependency in your pom.xml:

```
<dependency>
```

```
        <groupId>org.apache.mahout</groupId>
        <artifactId>mahout-core</artifactId>
        <version>0.9</version>
    </dependency>
```

Mahout's Command Line

Mahout now ships with its own command-line interface that makes it easier to submit and run Mahout programs both locally and on Hadoop. It's located at $MAHOUT_HOME/bin and assumes that the $HADOOP_CONF_DIR environment variable is set to your Hadoop's configuration directory:

```
export HADOOP_CONF_DIR=$HADOOP_PREFIX/conf
```

To run Mahout in local mode, you can set an additional environment variable $MAHOUT_LOCAL to "TRUE" (or any non-empty value):

```
export MAHOUT_LOCAL="TRUE"
```

This will cause Mahout to skip adding HADOOP_CONF_DIR to the classpath, and will force Mahout to run locally even if HADOOP_CONF_DIR and HADOOP_HOME are set. Running `bin/mahout` without any arguments will cause the command line interface to prompt the user for a valid Mahout program name as the first argument:

```
/srv/mahout$ bin/mahout
An example program must be given as the first argument.
```

```
Valid program names are:
  arff.vector: : Generate Vectors from an ARFF file or directory
  baumwelch: : Baum-Welch algorithm for unsupervised HMM training
  canopy: : Canopy clustering

 .. snip..
```

The Mahout CLI is very convenient to test out Mahout algorithms directly from the command-line, but it does expect that we pass in the expected command-line arguments and convert our data inputs into the expected format. Also, not all the Mahout algorithms can be executed from the CLI interface. For the most part the supported CLI programs are distributed Hadoop algorithms that read and write to HDFS. That said, we should ensure that Hadoop is running before attempting to run these programs in the Mahout CLI:

```
/srv/mahout$ jps
6117 SecondaryNameNode
15285 Jps
6200 JobTracker
6458 TaskTracker
5869 DataNode
5626 NameNode
```

Collaborative Filtering

Collaborative filtering or recommendation systems are perhaps most commonly recognized in the e-commerce space, where retailers like Amazon and Netflix mine user-behavior data, such as views, ratings, clicks, and purchases, to generate and suggest other product recommendations. Broadly, there are two types of collaborative filtering algorithms:

1. **User-based recommenders** – find users that are similar to the target user, and uses their collaborative ratings to make recommendations for the target user
2. **Item-based recommenders** – find and recommend items that are similar or related to items associated with the target user

Mahout's collaborative filtering library supports both types of algorithms, but the user-based recommendation algorithm is designed to run on a single JVM and thus cannot run in MapReduce. The item-based recommendation algorithm, however, can run in both single-machine mode and MapReduce, and if you run bin/mahout you will notice that it appears in the list of programs that can be invoked from the Mahout command-line. We will focus on the item-based recommender in this lesson, but the user-based recommender utilizes similar models and concepts.

For both the user and item-based recommenders, Mahout expects user and item input data, called *preferences*. Preferences can be represented as a 3-tuple of (UserID, ItemID, Rating) or a 2-tuple of (UserID, ItemID). In the 3-tuple, the Rating is usually a number expressing the strength of a user's preference for the item, where larger numbers mean stronger positive preferences. In the latter case, the preferences are assumed to be boolean, where there is or is not an association between the User and Item. In Mahout, preferences are represented by a *DataModel* class, which can be created from a file of these tuples (either CSV or TSV), with one preference tuple per line:

```
UserID,ItemID,Rating
100,111,5.0
100,222,3.0
100,333,2.5
200,111,2.0
200,222,2.5
200,333,5.0
200,444,2.0
300,111,2.5
300,444,4.0
300,555,4.5
300,777,5.0
400,111,5.0
400,333,3.0
400,444,4.5
400,666,4.0
500,111,4.0
500,222,3.0
500,333,2.0
500,444,4.0
500,555,3.5
500,666,4.0
```

In a Java project, the DataModel can be loaded using Mahout's FileDataModel class:

```
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel
DataModel model = new FileDataModel(new File("prefdata.csv"));
```

Next we need to define a similarity model that our recommender will use to find similar users or items. In an item-based recommender, the ItemSimilarity model determines the similarity measure between two items. Mahout provides a number of similarity models depending on the type of recommender, but Euclidean and Pearson similarity measures are commonly used as a starting point. You can set the desired similarity model with the corresponding Similarity class and model class:

```
import org.apache.mahout.cf.taste.impl.similarity.*;
ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
```

Pearson's correlation will result in a number between -1 and 1, which indicates how much two series of numbers move proportionate to each other, or exhibit a linear relationship. A value of 1 indicates the highest correlation, while -1 indicates the lowest.

We can now run a test in the Mahout command-line to see how the item-based recommendation algorithm performs on our data set, based on the similarity measure we chose. First let's prepare our data into the format expected by Mahout. For this example we will use the dataset from GroupLens, a research lab in the University of Minnesota's Department of Computer Science and Engineering[4]. They

[4] GroupLens Social Computing Research at the University of Minnesota: http://grouplens.org/

have published a number of data sets based on their research into recommender systems, including several movie-rating data sets from the MovieLens website, collected over various periods of time, depending on the size of the set. We will use the 100K dataset, which you can download from the GroupLens website:

http://files.grouplens.org/datasets/movielens/ml-100k.zip

```
~$ wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
~$ unzip ml-100k.zip
```

We are specifically interested in the file `u.data`, which contains the full data set of 100,000 ratings by 943 users on 1,682 items. Each user has rated at least 20 movies. The file is formatted as a tab-delimited file where each row contains the following fields:

user id | item id | rating | timestamp

This is already fairly close to the format that Mahout's item-based recommender expects; we just need to remove the timestamp from each row. We can write a simple `awk` script to convert it to the proper format:

```
~$ awk -F"\t" '{print $1"\t"$2"\t"$3}' ml-100k/u.data >
movieratings.tsv
```

The recommender also requires a separate users file, which includes a list of user IDs for which item recommendations will be generated. The MovieLens data conveniently provides this for us in the `u.user` file but we'll just take a small subset of the first 10 of those users. For some reason this file is pipe-delimited, so we need to separate on the pipe character:

```
~$ head -n 5 ml-100k/u.user | awk -F"|" '{print $1}' > users.txt
```

Now we can push both files to HDFS:

```
~$ hadoop fs -mkdir recommendations
~$ hadoop fs -put movieratings.tsv users.txt
/user/analyst/recommendations
```

We're now ready to run the item-based recommender from the Mahout command-line. We will run the recommenditembased program and specify the corresponding arguments for the number of recommendations we want for each user, similarity class name, input preference data, user data, temporary directory and output directory:

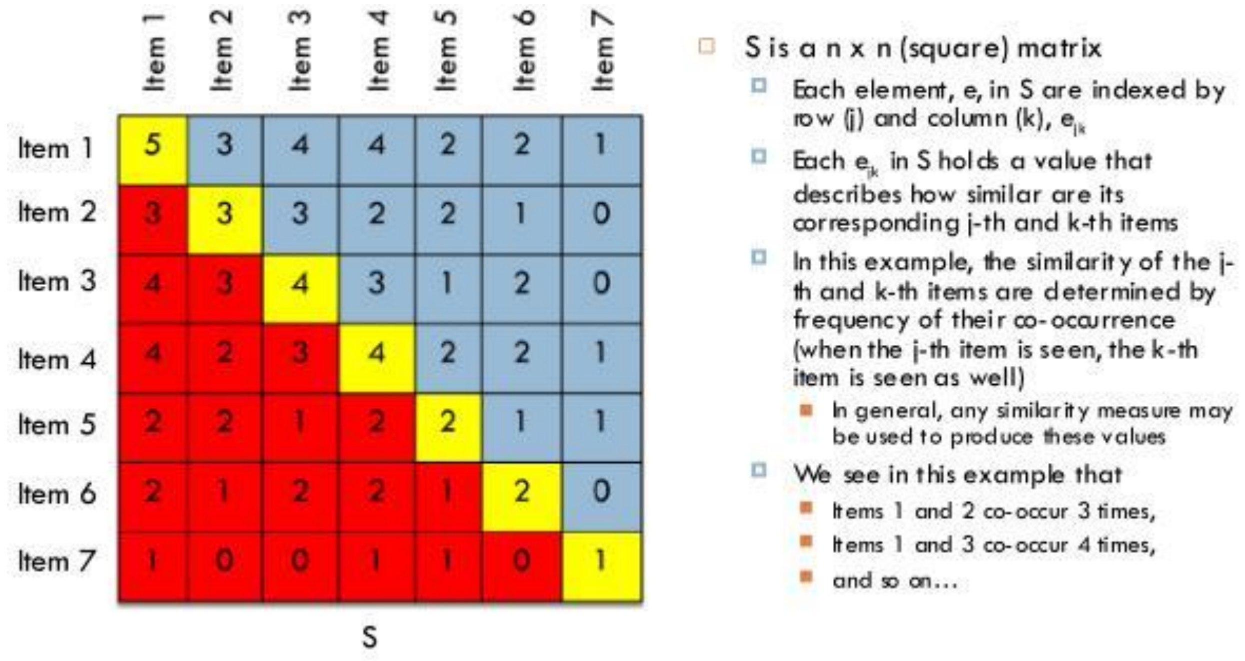
```
~$ cd $MAHOUT_HOME
/srv/mahout$ bin/mahout recommenditembased \
--numRecommendations 10 \
  --similarityClassname SIMILARITY_PEARSON_CORRELATION \
--input /user/analyst/recommendations/movieratings.tsv \
--usersFile /user/analyst/recommendations/users.txt \
--tempDir recommendations-tmp \
--output recommendations-results
```

Even a small recommendation job will take a long time to complete on a pseudo-distributed Hadoop cluster. On the course-provided VM, a single run of this job generally takes 5-7 minutes to complete (hence the emphasis on scalable machine learning!), but once it finishes you can view the output in the HDFS output folder you specified:

```
/srv/mahout$ hadoop fs -cat recommendations-results/part-r-*
1  [964:5.0,199:5.0,1323:5.0,594:5.0,185:5.0,1248:5.0,407:5.0,132:5.0,1265:5.0,615:5.0]
2  [1598:5.0,1281:5.0,1358:5.0,1591:5.0,161:5.0,1388:5.0,682:5.0,598:5.0,256:5.0,209:5.0]
3  [1420:5.0,1237:5.0,690:5.0,875:5.0,632:5.0,1070:5.0,1231:5.0,1610:5.0,1271:5.0,303:5.0]
4  [1326:5.0,1388:5.0,174:5.0,1358:5.0,928:5.0,368:5.0,1085:5.0,1078:5.0,598:5.0,237:5.0]
5  [1560:5.0,1588:5.0,45:5.0,1351:5.0,32:5.0,1194:5.0,853:5.0,69:5.0,571:5.0,1082:5.0]
6  [449:5.0,1560:5.0,853:5.0,45:5.0,495:5.0,869:5.0,838:5.0,565:5.0,69:5.0,32:5.0]
7  [615:5.0,650:5.0,369:5.0,964:5.0,500:5.0,356:5.0,407:5.0,1323:5.0,1265:5.0,1163:5.0]
8  [1123:5.0,503:5.0,722:5.0,47:5.0,1070:5.0,237:5.0,1078:5.0,1206:5.0,928:5.0,226:5.0]
9  [117:5.0,490:5.0,234:5.0,642:5.0,1528:5.0,195:5.0,515:5.0,1202:5.0,121:5.0,210:5.0]
10 [69:5.0,495:5.0,565:5.0,68:5.0,45:5.0,449:5.0,474:5.0,838:5.0,853:5.0,1560:5.0]
```

Each row of the output consists of the user's ID, followed by a comma-separated list of item IDs and their computed score, based on the original rating-scale from 1.0 to 5.0. Mahout will return the recommendations in order of highest recommendation score, with the best match appearing first.

So how does Mahout's item-based recommender actually calculate the recommendations using the specified similarity class? The distributed implementation of the item-based recommender starts with building a co-occurrence matrix based on the specified similarity model (i.e. – Pearson Correlation) to perform pairwise comparisons across the items, and retaining items pairs that are similar[5]:



	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7
Item 1	5	3	4	4	2	2	1
Item 2	3	3	3	2	2	1	0
Item 3	4	3	4	3	1	2	0
Item 4	4	2	3	4	2	2	1
Item 5	2	2	1	2	2	1	1
Item 6	2	1	2	2	1	2	0
Item 7	1	0	0	1	1	0	1

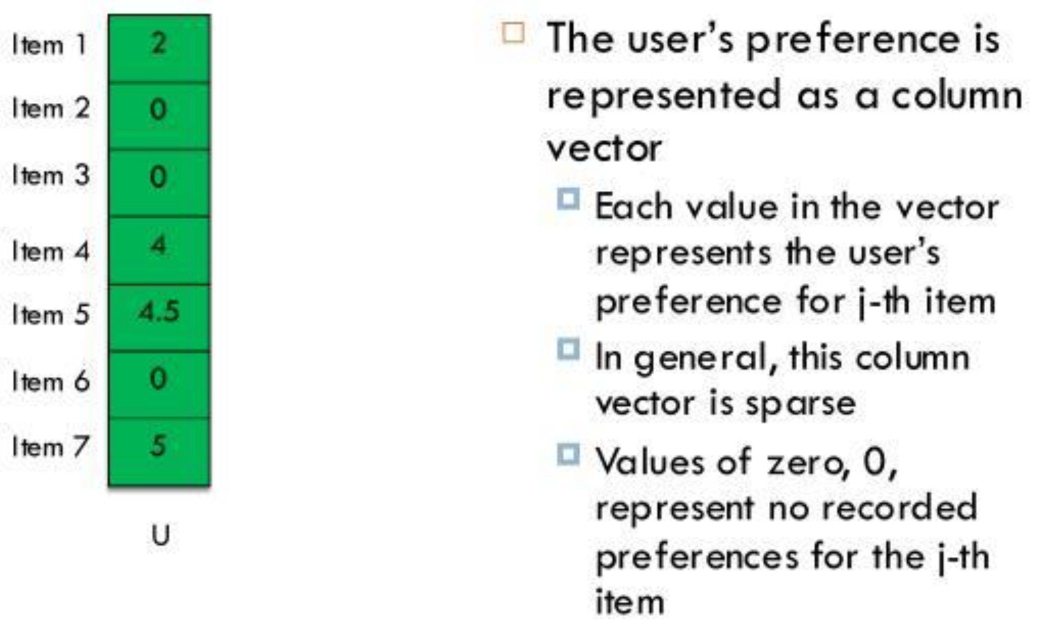
S

- S is a n x n (square) matrix
  - Each element, e, in S are indexed by row (j) and column (k), e_{jk}
  - Each e_{jk} in S holds a value that describes how similar are its corresponding j-th and k-th items
  - In this example, the similarity of the j-th and k-th items are determined by frequency of their co-occurrence (when the j-th item is seen, the k-th item is seen as well)
    - In general, any similarity measure may be used to produce these values
  - We see in this example that
    - Items 1 and 2 co-occur 3 times,
    - Items 1 and 3 co-occur 4 times,
    - and so on…

Figure 1 - Item Similarity Matrix

The user's preferences for each of the retained items are represented as a single-column vector where each row-value represents the user's preference (i.e. – rating) for the item. This vector is often sparse, as a user may only have recorded preferences for a few items.

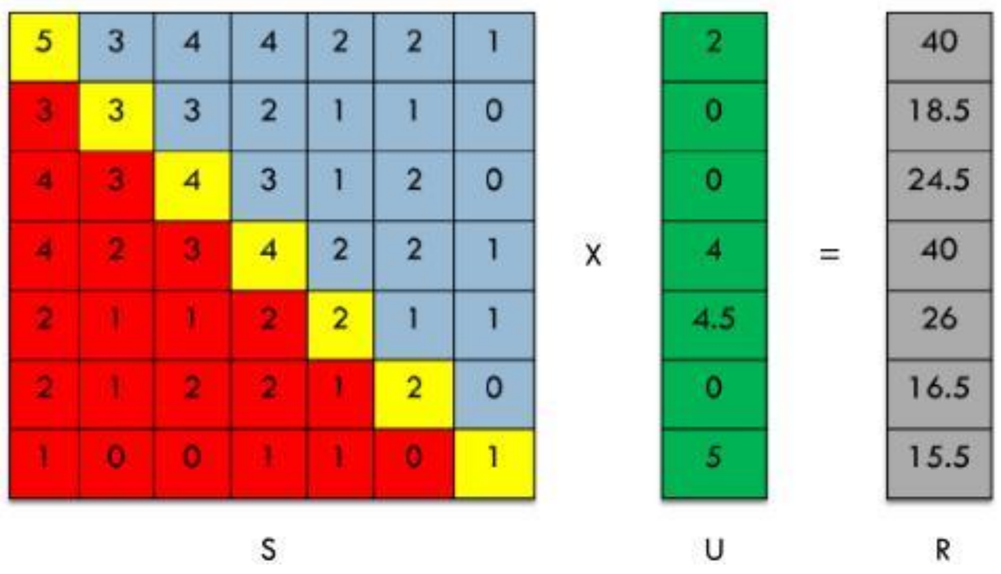
[5] Tutorial on Mahout's Recommendation Algorithm: http://www.slideshare.net/vangjee/a-quick-tutorial-on-mahouts-recommendation-engine-v-04



- The user's preference is represented as a column vector
  - Each value in the vector represents the user's preference for j-th item
  - In general, this column vector is sparse
  - Values of zero, 0, represent no recorded preferences for the j-th item

Figure 2 - User Preference Vector

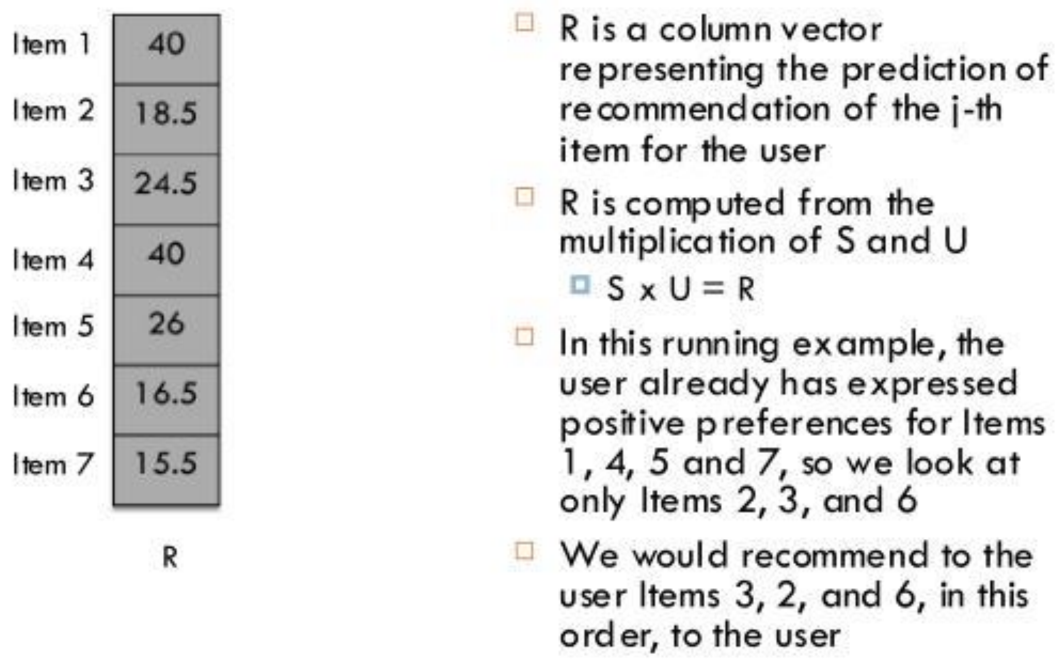
The resulting recommendations vector is calculated by taking the product of the original similarity matrix and the user preference vector.



$$\underset{S}{\begin{bmatrix} 5 & 3 & 4 & 4 & 2 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 & 1 & 0 \\ 4 & 3 & 4 & 3 & 1 & 2 & 0 \\ 4 & 2 & 3 & 4 & 2 & 2 & 1 \\ 2 & 1 & 1 & 2 & 2 & 1 & 1 \\ 2 & 1 & 2 & 2 & 1 & 2 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}} \times \underset{U}{\begin{bmatrix} 2 \\ 0 \\ 0 \\ 4 \\ 4.5 \\ 0 \\ 5 \end{bmatrix}} = \underset{R}{\begin{bmatrix} 40 \\ 18.5 \\ 24.5 \\ 40 \\ 26 \\ 16.5 \\ 15.5 \end{bmatrix}}$$

Figure 3 - Similarity and Preference Multiplication

Items that the user has seen or expressed preference for are removed from the resulting vector, and the remaining results are ranked in order of highest predicted preference.



- R is a column vector representing the prediction of recommendation of the j-th item for the user
- R is computed from the multiplication of S and U
  - S x U = R
- In this running example, the user already has expressed positive preferences for Items 1, 4, 5 and 7, so we look at only Items 2, 3, and 6
- We would recommend to the user Items 3, 2, and 6, in this order, to the user

Figure 4 - Recommendation Vector

While this particular job flow is specific to Mahout's item-based recommendation algorithm, the other available collaborative filtering algorithms also employ matrix representation and factorization in some manner. Mahout's item-based recommender has the advantage of being both simple and straightforward for the base use-case, while also providing the flexibility to support a wide range of similarity measures and filter controls.

Mahout also comes with an evaluation package in `org.apache.mahout.cf.taste.eval` with tools to help you examine the quality of the results. The evaluation tools allow us to hold back part of the data set as test data, so that we can compare the generated recommendations for those users to the actual preference values, using evaluation models like Average Absolute Difference or Root Mean Square. We won't cover the evaluation models in this lesson, but we encourage you to experiment with them, as well as the other algorithms and similarity models, in exploring Mahout's collaborative filtering capabilities.

Classification

Classification attempts to categorize data, usually text or documents, based on supervised training methods that utilize annotated training sets to discover patterns that will allow the learner to quickly label new records[6]. For example, a simple classification algorithm might keep track of the features and words associated with a category, as well as the number of times those words are seen for a given category. Once the learner has extracted the features from the training data, it can generate a

[6] Benjamin Bengfort (April 2014). Introduction to Analytics with Hadoop: Lesson 4. Statistics.com.

feature vector and apply a statistical model to build a predictive model, which can be applied to new data.

Mahout provides several algorithms for classification, although the most popular classification algorithms are: Naive Bayes, Hidden Markov Models, Logistic Regression, and Random. Of these, Naive Bayes and Random Forest can be implemented in MapReduce; we will use the naive Bayes classifier to implement a simple newsgroup post categorizer with Mahout.

Naive Bayes Classification

Mahout has two Naive Bayes implementations: standard Multinomial Naive Bayes (referred to as Bayes), and an implementation of Transformed Weight-normalized Complement Naive Bayes (referred to as CBayes)[7]. Both Bayes and CBayes are trained via MapReduce, although testing and classification can be done with MapReduce or single-machine processing.

Bayes' theorem is expressed in the following formula:

[7] Apache Mahout "Naive Bayes": http://mahout.apache.org/users/classification/bayesian.html

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figure 5 - Bayes's Theorem

In terms of text classification, **A** represents a category, while **B** represents the text or document. Thus, **P(A|B)** is the probability that document **B** is in category **A**. Bayes's theorem as applied to text classification states that this probability can be derived by the probability that for a given category **A**, words in **B** appear in that category, **P(B|A)**, multiplied by the probability of a document being in category **A**, **P(A).** This is divided by the probability of encountering the words in document **B**, **P(B)**. In the example of our newsgroup post classifier, where categories may include "autos" or "politics", this theorem might be applied to words found in the post body:

P(autos|Porsche) = P(Porsche|autos) * P(autos)

P(Porsche)

Figure 6 - Bayes Category Classifier Algorithm[8]

To run Mahout's Bayes classification algorithm, we first need to prepare our training and test data, and convert them into TF-IDF vectors that can be fed as inputs into the actual training step. For this example, we'll use the classic 20 newsgroups corpus. The 20 newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned evenly across 20 different newsgroups:

```
~$ wget http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz
~$ mkdir -p 20news-bydate
~$ cd 20news-bydate && tar xzf ../20news-bydate.tar.gz && cd ..
~$ mkdir 20news-all
~$ cp -R 20news-bydate/*/* 20news-all
```

On a Hadoop cluster, we would then upload the directory of newsgroup data to HDFS, but since this is a lot of data that would take a very long time to upload to a pseudo-distributed Hadoop in a VM, we will just run Mahout in Local Mode for this example.

```
~$ cd $MAHOUT_HOME
/srv/mahout$ export MAHOUT_LOCAL="True"
```

[8] Bayes Theorem with Mahout: http://www.mimul.com/pebble/default/2012/04/03/1333431077222.html

We need to first convert our raw data newsgroup messages into sequence files that can be vectorized. Mahout provides the seqdirectory program for this, which will generate sequence files from a directory of text:

```
/srv/mahout$ bin/mahout seqdirectory \
   -i ~/20news-all \
   -o ~/20news-seq \
   -ow
```

Now we can convert the sequence files in the 20news-seq directory into TF-IDF vectors. Mahout provides a tool to perform this conversion, seq2sparse, which will generate sparse vectors from sequence files. We'll pass in the 20news-seq directory as input, specifying the -wt tfidf option for TF-IDF vector format, and –n 2 for L2 length normalization:

```
/srv/mahout$ bin/mahout seq2sparse \
   -i ~/20news-seq \
   -o ~/20news-vectors \
   -lnorm -nv -wt tfidf
```

Now that we've generated the weighted vectors, we need to provide the classifier with training data to generate the model. However, we would like to hold back some of the data to reserve for testing the model too. To do so, we'll divide the vector files into two sets, one for training and one for testing. We'll specify this split at 60/40 using the randomSelectionPct option:

```
/srv/mahout$ bin/mahout split \
   -i ~/20news-vectors/tfidf-vectors \
   --trainingOutput ~/20news-train-vectors \
   --testOutput ~/20news-test-vectors \
   --randomSelectionPct 40 \
   --overwrite --sequenceFiles -xm sequential
```

Now we have two directories, 20news-train-vectors and 20news-test-vectors, that contain our training set and testing set respectively. We can now train our Bayes algorithm and generate the model that we'll use for the classifier:

```
/srv/mahout$ bin/mahout trainnb \
   -i ~/20news-train-vectors -el \
   -o ~/model \
   -li ~/labelindex \
   -ow -c
```

The trainnb command should have generated the classifier model so that we can test it against the remaining 40% of the input data:

```
/srv/mahout$ bin/mahout testnb \
   -i ~/20news-test-vectors \
   -m ~/model \
   -l ~/labelindex \
   -o ~/20news-testing \
   -ow –c
```

The output of the testnb command will include a variety of performance metrics for the classification test, including a *confusion matrix,* which describes how many results were correctly classified and how many were incorrectly classified for each of the categories:

```
=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances          :       6609    89.1903%
Incorrectly Classified Instances        :        801    10.8097%
Total Classified Instances              :       7410

=======================================================
Confusion Matrix
-------------------------------------------------------
a  b c d e f g h i j k l m n o p q r s t <--Classified as
285 00 0 1 0 0 0 0 0 0 0 1 0 4 7 0 1 6 1  |  306     a     = alt.atheism
3  310 9 10  5 21  4 1 0 2 2 5 7 0 2 0 1 2 1 1  |  386     b     = comp.graphics
1  20  311 12  89 1 1 3 1 2 2 3 4 3 2 1 0 0 2  |  386     c     = comp.os.ms-windows.misc
0  9 30  312 11  612  2 1 3 1 5 7 4 2 3 2 3 1 0  |  414     d     = comp.sys.ibm.pc.hardware
1  6 5 5 325 3 5 0 1 0 1 1 3 0 2 0 2 2 0 0  |  362     e     = comp.sys.mac.hardware
1  14  4 2 1 390  0  0 2 0 2 1 0 1 1 2 0 1 0 0  |  422     f     = comp.windows.x
1  8 9 17  14  4  264 17  6  2 5 1 14  4 4 1 5 1 1 1  |  379     g     = misc.forsale
0  0 1 2 3 1 0 344  7  1 0 0 3 0 1 1 1 1 0 2  |  368     h     = rec.autos
0  0 0 3 0 0 0 0 373  0  0 0 1 1 1 1 1 1 0 0  |  382     i     = rec.motorcycles
0  0 0 0 0 0 2 0 2 383  6  0 1 3 0 1 1 0 0 0  |  399     j     = rec.sport.baseball
0  0 0 0 0 0 0 0 0 1 415  0  0 0 0 0 0 0 0 1  |  417     k     = rec.sport.hockey
1  0 1 0 1 1 0 1 0 1 0 380  0  0 2 0 0 2 1 2  |  393     l     = sci.crypt
1  2 5 12  2 1 4 10  2 3 3 1 325  3 3 1 0 1 1 2  |  382     m     = sci.electronics
0  2 1 0 0 2 0 1 0 0 2 0 2 365  3  1 0 1 1 1  |  382     n     = sci.med
0  0 0 0 0 1 2 1 0 1 0 0 0 2 362  0  1 1 1 2  |  374     o     = sci.space
2  0 0 1 0 0 0 0 1 2 0 0 1 3 2 382  2  3 3 0  |  402     p     = soc.religion.christian
0  0 0 0 0 0 0 0 0 2 0 0 0 0 0 1 350  0  0 0  |  353     q     = talk.politics.mideast
1  1 1 0 0 1 0 1 2 1 0 3 0 1 1 0 1 324  1 9  |  348     r     = talk.politics.guns
33  10 0 0 0 0 0 0 0 1 0 0 0 1 4 35  4 6 153 5  |  243     s     = talk.religion.misc
2  0 2 0 0 0 2 0 1 2 3 1 0 1 4 2 11  24  1  256  |  312     t     = talk.politics.misc
=======================================================
Statistics
-------------------------------------------------------
```

```
Kappa                                         0.8566
Accuracy                                     89.1903%
Reliability                                  84.4891%
Reliability (standard deviation)              0.2185

14/05/10 18:44:03 INFO driver.MahoutDriver: Program took 11676 ms
(Minutes: 0.1946)
```

Your results may vary, depending on how the split operation partitioned your training and test sets. From here we can evaluate the accuracy of our classifier model and either revisit our training set, or apply the model on an actual unlabeled set of data. Mahout is optimized for large sets of supervised training data, so in general, more data will yield better results than small, but higher-precision data.

Clustering

Unlike collaborative filtering and classification algorithms, clustering utilizes unsupervised learning techniques to build a model. Clustering algorithms attempt to organize a collection of data into groups of similar items. Examples of clustering might include finding groups of customers with similar characteristics or interests, or grouping animals/plants into common species. The goal of clustering is to partition data into a number of clusters such that the data within each cluster is more similar to each other than to data in other clusters[9].

Mahout provides a number of clustering algorithms, including Canopy clustering, k-Means Clustering, Fuzzy k-Means, Streaming k-Means, and Spectral Clustering. k-Means is probably the simplest of these algorithms, but requires that we represent all objects as a set of numerical features, and that we specify the number of clusters (k clusters) we want up front.

Mahout's implementation of k-Means clustering starts again with vectorizing the data set, and representing each object within a feature vector in *n*-dimensional space, where *n* is the number of all features used to describe the objects to be clustered. The algorithm randomly chooses *k* points in that vector space, which serve as the initial centers, or *centroids*, of the clusters. The algorithm then assigns each object to the centroid that it's closest to, recalculating the centroid point using the average of the coordinates of all the points in the cluster and reassigning objects to their closest cluster, as necessary. The process of assigning objects and re-computing centers is repeated until the process converges[10].

---

[9] Alex Holmes. Hadoop In Practice (2012). Manning Publications.

[10] Apache Mahout "k-Means Clustering Basics": http://mahout.apache.org/users/clustering/k-means-clustering.html

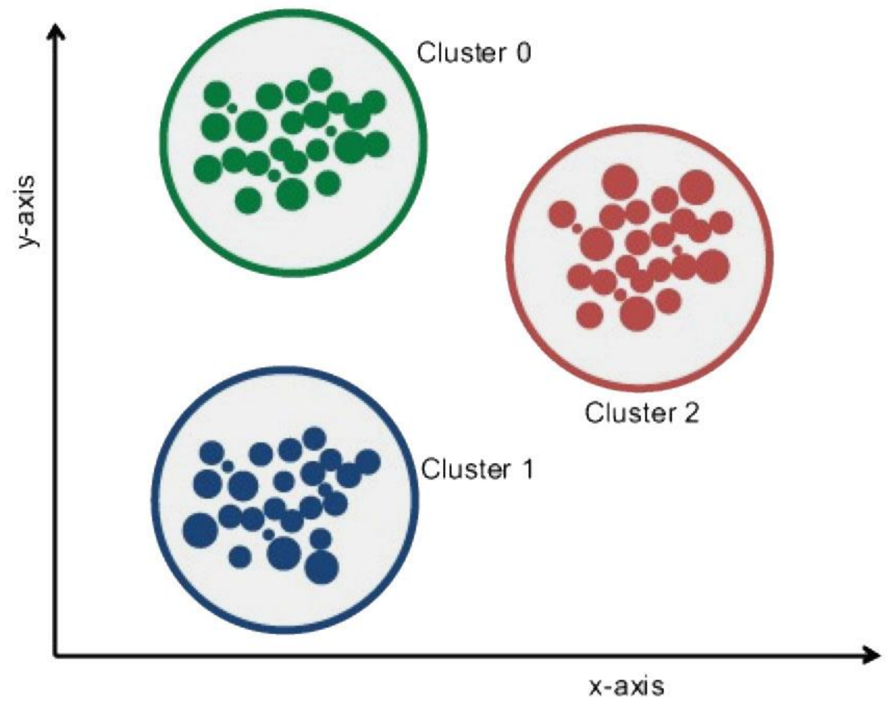


Figure 7 - k-Means clustering

The most important issue in clustering is to determine how to quantify the similarity of the objects being clustered. The weighting method may be derived from TF-IDF (term frequency-inverse document frequency) which is particularly useful for text-documents, or it may be determined by a function of other custom properties in our data (i.e. – segmenting customers based on total $ purchases x average engagement. For Mahout's clustering input, we need to express whatever weighting method we use as a feature vector. For example, if we determined that we want to cluster all customer by 3 features: total purchase amount, average purchase frequency, and average per/purchase amount, then a sample of our customers might be represented as:

	Total Purchase in $	Average Purchases per Month	Average Per/Purchase Amount	Vector
Jane	825	5	115	[825,5,115]
Bob	201	1	45	[201,1,45]
Emma	649	2	65	[649,2,65]

With multiple features, we must be mindful of dimension values that are expressed in different units, or are not normalized with respect to each other. If we applied a simple distance-based metric to determine similarity between these vectors, total purchase amounts would dominate the results. Weighting the different dimensions solves this problem[11].

k-Means Clustering

In the following example, we will use a simple TF-IDF weighting method to vectorize the text-based input from the Reuters 21578 corpus[12].

```
~$ wget
http://www.daviddlewis.com/resources/testcollections/reuters21578/reute
rs21578.tar.gz
~$ mkdir reuters
```

[11] Sean Owen, Robin Anil, Ted Dunning, Ellen Friedman. Mahout in Action (Kindle Locations 3447-3450). Manning Publications. Kindle Edition.

[12] Reuters Corpora: http://about.reuters.com/researchandstandards/corpus/

```
~$ cd reuters && tar xvf ../reuters21578.tar.gz
```

As in our classification example, we need to convert the text data into sequence files, and then convert the sequence files into vector format:

```
~$ cd $MAHOUT_HOME
/srv/mahout$ bin/mahout seqdirectory -i ~/reuters -o ~/reuters-seq -c
UTF-8 -chunk 5
/srv/mahout$ bin/mahout seq2sparse -i ~/reuters-seq -o ~/reuters-vec --
maxDFPercent 85 --namedVector
```

Now we can run the kmeans program to cluster the data based on a Cosine Distance Measure with 20 clusters (-k 20) and 10 maximum iterations (-x 10):

```
/srv/mahout$ bin/mahout kmeans -i ~/reuters-vec/tfidf-vectors -c
~/reuters-clusters -o ~/reuters-kmeans -dm
org.apache.mahout.common.distance.CosineDistanceMeasure -x 10 -k 20 -ow
--clustering -cl
```

The resulting output directory, reuters-clusters, will include a directory called clusters-N which contains the clustered points as (clusterId, vectorObject) pairs, stored as SequenceFiles. To dump these files into a format that we can read or feed as input into another application, we need to dump the cluster files and then use seqdumper to convert back to a text format:

```
/srv/mahout$ bin/mahout clusterdump -i ~/reuters-kmeans/clusters-*-
final -d ~/reuters-vec/dictionary.file-0 -dt sequencefile -b 100 -n 20
--evaluate -dm org.apache.mahout.common.distance.CosineDistanceMeasure
--pointsDir ~/reuters-kmeans/clusteredPoints/ -o ~/cluster-output.txt
```

```
 /srv/mahout$ bin/mahout seqdumper -i ~/reuters-
kmeans/clusteredPoints/part-m-00000 > ~/cluster-points.txt
```

Conclusion

In this chapter, we implemented a simple item-based recommender, categorized documents using a Naive Bayes classifier, clustered a collection of documents using a k-Means clustering algorithm, and learned a bit about vector representation of input data. But we have only just scratched the surface of Mahout's predictive analytics capabilities. In addition to other algorithms and data preparation tools, Mahout also offers quality evaluation tools to evaluate the performance of our algorithms. We hope that this short introduction has demonstrated the potential of using Mahout to apply powerful statistical learning techniques to large data sets, and we encourage you to learn more about Mahout's ML offerings and upcoming developments as it continues to evolve into a broader distributed machine-learning framework.

Discussion Questions

1. How might we be able to evaluate the quality of a supervised learning algorithm (i.e. – recommender or classifier)?
    a. What are some potential strategies to improve the quality of results generated?
2. As mentioned previously, Mahout was borne out of the Apache Lucene search engine project before coming its own top-level project. What application would machine-learning techniques have in the area of search, and what unique implications does the problem of search pose for machine-learning?

Assignment (Total 15 points)

The Resources section contains a zip file, dating.zip, with 2 CSVs containing datasets from a dating agency:

The ratings.csv file includes the fields: UserID, ProfileID, Rating

- UserID is the user who provided rating
- ProfileID is user who has been rated
- Ratings are on a 1-10 scale where 10 is best (integer ratings only) scores

The users.csv file contains a list of users with fields: UserID

(5 points)

Using the item-based recommender with the Pearson Correlation similarity metric, generate 2 recommendations for User IDs 1 through 5. Copy and submit any commands used, and the results. (NOTE: You can use Mahout in Local Mode, if you prefer.)

(10 points)

Build a spam filter using a Naive Bayes classifier with categories "spam" and "ham". Use the SpamAssassin corpus at: http://spamassassin.apache.org/publiccorpus/, specifically the 20021010_spam_tar.bz2 and 20021010_easy_ham.tar.bz2 corpora for your training and test data.

HINT: You should start by unpacking the spam and easy_ham directories within a containing directory, which will serve as the input for the seqdirectory tool. You can use a 40% split for the training/testing data split.

Submit all commands used, output summary, and confusion matrix.