Distributed Machine Learning with Apache Mahout

By Ian Pointer and Dr. Ir. Linda Terlouw

INTRODUCTION

Apache Mahout is a library for scalable machine learning. Originally a subproject of Apache Lucene (a high-performance text search engine library), Mahout has progressed to be a top-level Apache project.

While Mahout has only been around for a few years, it has established itself as a frontrunner in the field of machine learning technologies. Mahout has currently been adopted by: Foursquare, which uses Mahout with Apache Hadoop and Apache Hive to power its recommendation engine; Twitter, which creates user interest models using Mahout; and Yahoo!, which uses Mahout in their anti-spam analytic platform. Other commercial and academic uses of Mahout have been catalogued at https://mahout.apache.org/general/powered-by-mahout.html.

This Refcard will present the basics of Mahout by studying two possible applications:

- Training and testing a Random Forest for handwriting recognition using Amazon Web Services EMR

AND

- Running a recommendation engine on a standalone Spark cluster.

MACHINE LEARNING

Machine learning algorithms, in contrast to regular algorithms, improve their performance after they acquire more experience. The "intelligence" is not hard-coded by the developer, but instead the algorithm learns from the data it receives. Computer scientist Tom Mitchell defines machine learning as follows:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E (Mitchell, 1997).*

MACHINE LEARNING TASKS

The most common tasks in machine learning are:

- **Recommendation Using Association Rules**
  Recommendation tasks predict items that have a high similarity to others within a given set of items.
  - Example: Predicting movies or books based on someone's historic purchase behavior.
- **Classification**
  Classification tasks predict to which class/category a certain item belongs. These categories are predefined. A classification task can be binary or multi-class.
  - Example: Determining whether a message is spam or non-spam (binary); determining characters from a handwriting sample (multi-class).
- **Regression**
  Regression tasks focus on predicting numeric values.
  - Example: Predicting the number of ice cream cones to be sold on a certain day based on weather data.
- **Clustering**
  Clustering tasks divide items into groups, but unlike in classification tasks, these groups are not previously defined.
  - Example: Grouping customers based on certain properties to discover customer segments.

Supervised Learning

A supervised learning task is a task in which the training and testing data is labeled with both inputs and their desired outputs. These tasks search for patterns between inputs and outputs in training data samples, determine rules based on those patterns, and apply those rules to new input data in order to make predictions on the output. Classification and regression are examples of supervised learning tasks.

Unsupervised Learning

An unsupervised learning task is a task in which data is unlabeled. These tasks attempt to discern unknown structures that tie unlabeled input data together to produce organized output data. Clustering and association rule discovery are unsupervised learning tasks.

A machine learning algorithm creates a *model* based on a training data set. In supervised learning, the performance of the model can be determined by providing the test data set to the model and comparing the given answers to the expected (labeled) answers. If a model is overly-tuned to the training set (and thus not generalizable), then the model is referred to as *overfitting*. If a model is too generic (and does not discover the right trend in the data), the model is referred to as *underfitting*. Both overfitting and underfitting models result in bad predictions.

TERM	EXPLANATION
Training data set	Subset of the available data used to create the model
Test data set	Subset of the available data used to evaluate the performance of the model
Feature	Input variable used by the model to generate its output
Target variable	Variable that the model aims to predict
Class	Predefined group to which individuals (items) from a data set may or may not belong
Overfitting	Describes a model that focuses too much on the specifics (noise) of the training data set and does not make accurate predictions
Underfitting	Describes a model that is too generic (does not include trends in the training data set) and does not make accurate predictions

ALGORITHMS SUPPORTED IN APACHE MAHOUT

Apache Mahout implements sequential and parallel machine learning algorithms, which can run on MapReduce, Spark, H2O, and Flink*. The current version of Mahout (0.10.0) focuses on recommendation, clustering, and classification tasks.

ALGORITHM	STANDALONE	MAPREDUCE	SPARK	H2O
User-Based Collaborative Filtering	✔		✔	
Item-Based Collaborative Filtering	✔	✔	✔	
Matrix Factorization with ALS	✔	✔		
Matrix Factorization with ALS on Implicit Feedback	✔	✔		
Weighted Matrix Factorization, SVD++	✔			
Logistic Regression - trained via SGD	✔			
Naive Bayes / Complementary Naive Bayes		✔	✔	
Random Forest		✔		
Hidden Markov Models	✔			
Multilayer Perceptron	✔			
k-Means Clustering	✔	✔		
Fuzzy k-Means	✔	✔		
Streaming k-Means	✔	✔		
Spectral Clustering		✔		
Singular Value Decomposition	✔	✔	✔	✔
Stochastic SVD	✔	✔	✔	✔
PCA	✔	✔	✔	✔
QR Decomposition	✔	✔	✔	✔
Latent Dirichlet Allocation	✔	✔		
Row-SimilarityJob		✔	✔	
ConcatMatrices		✔		
Collocations		✔		
Sparse TF-IDF Vectors from Text		✔		

* The Mahout Distributed Basic Linear Algebra Subprograms (BLAS) Core Library, currently supported on the Spark and H2O engines, is currently in development for the Flink engine.

INSTALLING APACHE MAHOUT

Mahout requires Java 7 or above to be installed, and also needs a Hadoop, Spark, H2O, or Flink platform for distributed processing (though it can be run in standalone mode for prototyping). Mahout can be downloaded from http://mahout.apache.org/general/downloads.html and can either be built from source or downloaded in a distribution archive. Download and untar the distribution, then set the environment variable MAHOUT_HOME to be the directory where the distribution is located.

Note: You will also need JAVA_HOME set.

Mahout comes packaged as a command-line application:

```
$ $MAHOUT_HOME/bin/mahout
```

This will give you a list of available operations that Mahout can run (e.g. k-means clustering, vectorizing files, etc.). Note that this list is not exhaustive. Passing in an operation name will give you a main page for that operation. For example:

```
$ $MAHOUT_HOME/bin/mahout vectordump

Usage:

 [--input <input> --output <output> --useKey <useKey>
--printKey <printKey> --dictionary <dictionary>
--dictionaryType <dictionaryType> --csv <csv>
--namesAsComments <namesAsComments> --nameOnly <nameOnly>
--sortVectors <sortVectors> --quiet <quiet> --sizeOnly
<sizeOnly> --numItems <numItems> --vectorSize <vectorSize>
--filter <filter1> [<filter2> ...] --help --tempDir
<tempDir> --startPhase <startPhase> --endPhase <endPhase>]

Job-Specific Options:

  --input (-i) input          Path to job input directory.
```

EXAMPLE OF MULTI-CLASS CLASSIFICATION USING AMAZON ELASTIC MAPREDUCE

We can use Mahout to recognize handwritten digits (a multi-class classification problem). In our example, the features (input variables) are pixels of an image, and the target value (output) will be a numeric digit—0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

CREATING A CLUSTER USING AMAZON ELASTIC MAPREDUCE (EMR)

While Amazon Elastic MapReduce is not free to use, it will allow you to set up a Hadoop cluster with minimal effort and expense. To begin, create an Amazon Web Services account and follow these steps:

Create a Key Pair

1. Go to the Amazon AWS Console and log in.
2. Select the **EC2** tab.
3. Select **Key Pairs** in the left sidebar.

If you don't already have a key pair, create one here. Once your key pair is created, download the .pem file.

Note: If you have made a key value pair, but don't see it, you may have the wrong region selected.

Configure a Cluster

1. Go to the Amazon AWS Console and log in.
2. Select the **Elastic MapReduce** tab.
3. Press **Create Cluster**.
4. Fill in the following data:

CLUSTER CONFIGURATION	
Termination protection	No
Logging	Disabled
Debugging	Disabled
TAGS	
Empty	
SOFTWARE CONFIGURATION	
Hadoop distribution	Amazon AMI version 3.3.2
Additional applications	Empty
HARDWARE CONFIGURATION	
Network	Use default setting
EC2 availability zone	Use default setting
Master/Core/Task	Use default setting or change to your own preference (selecting more machines makes it more expensive)
SECURITY AND ACCESS	
EC2 key pairs	Select your key pair (you'll need this for SSH access)
IAM user access	No other IAM users
IAM ROLES	
Empty	
BOOTSTRAP ACTIONS	
Empty	
STEPS	
Add step	Empty
Auto-terminate	No

5. Press **Create Cluster** and the magic starts

Log Into the Cluster

When your cluster is ready, you can login using your key pair with the following command:

```
ssh -i key.pem hadoop@[public_ip]
```

You can find the public IP address of your Hadoop cluster master by going to Elastic MapReduce → Cluster List → Cluster Details in the AWS Console. Your Mahout home directory is /home/hadoop/mahout.

Don't forget to terminate your cluster when you're done, because Amazon will charge for idle time.

Note: *If a cluster is terminated, it cannot be restarted; you will have to create a new cluster.*

GETTING AND PREPARING THE DATA

We are going to use a data set from Kaggle.com, a data science competition website. Register at Kaggle.com and go to the competition "Digit Recognizer." Two data sets are available: train.csv and test.csv. We are only going to use train.csv, because test.csv is unlabeled (this file is used for scoring the data and getting a position on the Kaggle leader board). We want labeled data, because we want Mahout to evaluate our classification algorithm; to do this, Mahout needs to know the answers (i.e. the labels). Later on, we will use Mahout to split train.csv into a training data set and a test data set. For now, let's have a look at the data by opening train.csv. We see a header row that looks like this:

```
label,pixel0,pixel1,pixel2,pixel3,…,pixel782,pixel783
```

And rows of data that look like this:

```
9,0,0,0,...,0,0
```

Every now and then we see some non-zero numbers in the data rows, e.g. 2,15,154,221,254,254. What does this data mean? The first column of the training set is the label value; this is the digit. The values pixel0 to pixel783 represent the individual pixel values. The digit images are 28 pixels in height and 28 pixels in width, making 784 pixels in total. The value indicates the darkness of that pixel, ranging from 0 to 255.

The only step we need to take to prepare our data:

Remove the header of train.csv

CLASSIFYING FROM COMMAND LINE USING AMAZON ELASTIC MAPREDUCE

We are going to use the Amazon S3 file storage, but it's also possible to use the HDFS file system. Upload train.csv to an S3 bucket by:

1. Go to the Amazon AWS Console and log in.
2. Select the **S3** tab.
3. In the **Buckets** sidebar on the left, select the bucket to which you would like to add the .csv file.
4. In the **Objects and Folders** pane, click **Upload**.
5. Follow the steps presented to upload train.csv.

To split our data into a training data set and a test data set, we need to run the following commands:

```
$ cd mahout
```

```
$ ./bin/mahout splitDataset --input s3://<bucketname>/
train.csv --output s3://<bucketname>/data
--trainingPercentage 0.7 --probePercentage 0.3
```

When we explore our S3 bucket, we see that a new directory "data" has been created. This directory contains two subdirectories: "trainingSet" (training set) and "probeSet" (test set).

We need to tell Mahout which type of data we are dealing with by making a file descriptor:

```
$ hadoop jar mahout-core-0.9-job.jar org.apache.mahout.
classifier.df.tools.Describe -p s3://<bucketname>/train.csv
-f s3://<bucketname>/data.info -d L 784 N
```

This command generates a new file (data.info) that describes the data. The –d argument creates the actual description. In our case we have 1 label (the recognized digit) and 784 numeric values (the pixel values).

We train our Random Forest using the following command:

```
$ hadoop jar mahout-examples-0.9-job.jar org.apache.
mahout.classifier.df.mapreduce.BuildForest -Dmapred.max.
split.size=1874231 -d s3://<bucketname>/data/trainingSet
-ds s3://<bucketname>/data.info -sl 5 -p -t 100 -o
s3://<bucketname>/digits-forest
```

Then we test our Random Forest using the following command:

```
$ hadoop jar mahout-examples-0.9-job.jar org.
apache.mahout.classifier.df.mapreduce.TestForest -ds
s3://<bucketname>/data.info -i s3://<bucketname>/data/
probeSet -m s3://<bucketname>/digits-forest -a -mr -o
predictions
```

INTERPRETING THE TEST RESULTS

After you have tested your Random Forest, you will get results that look like this (exact values will vary):

```
=======================================================
Summary
-------------------------------------------------------
Correctly Classified Instances          :      11071      88.2222%
Incorrectly Classified Instances        :       1478      11.7778%
Total Classified Instances              :      12549

=======================================================
Confusion Matrix
-------------------------------------------------------
a       b       c       d       e       f       g       h       i       j       <--Classified as
1122    33      42      7       20      7       24      5       22      30      |  1312       a     = 3
19      1050    55      15      28      17      2       16      4       34      |  1240       b     = 2
4       3       1366    0       4       2       0       0       3       0       |  1382       c     = 1
13      3       0       1181    2       19      4       2       2       7       |  1233       d     = 0
1       10      36      3       1205    0       0       11      17      2       |  1285       e     = 7
1       10      20      18      4       1216    11      2       0       4       |  1286       f     = 6
80      6       41      26      5       29      874     15      14      9       |  1099       g     = 5
2       8       21      2       12      23      3       1109    85      6       |  1271       h     = 4
23      2       11      8       80      2       5       78      1024    10      |  1243       i     = 9
86      15      65      16      8       13      27      13      31      924     |  1198       j     = 8

=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.8621
Accuracy                                   88.2222%
Reliability                                79.9493%
Reliability (standard deviation)            0.2743
```

Figure 1: *Results of Random Forest test*

We see that 88% of the instances are correctly classified. In the confusion matrix we can see more details. A confusion matrix presents the predicted classes as columns and the actual classes as rows. In a perfect prediction, all elements on the confusion matrix, except for those on the matrix's main diagonal, would be 0.

Let's have a look at the results for the digit "3". This digit is labeled as "a" in this confusion matrix. In this example we see that 1122 of the cases that are actually the digit "3" are also predicted as a "3", while the total number of actual "3"s is 1312. By continuing to examine the first row of the matrix, we see

that 33 "3"s are incorrectly predicted as "2" (labeled as "b"), 42 are incorrectly predicted as "1" (coded as c), and so on. Looking at the "a" column, we find that 19 cases that are actually "2" (coded as label b) are incorrectly recognized as "3".

In the **Statistics** section of the results report, we can find the accuracy and the reliability of the algorithm based on these results. The accuracy is the probability that the prediction is correct based on the ratio of the number of correctly predicted cases to the total number of cases. The reliability shows the probability that the image is digit x.

USING APACHE MAHOUT WITH APACHE SPARK FOR RECOMMENDATIONS

In the following example, we will use Spark as part of a system to generate recommendations for different restaurants. The recommendations for a potential diner are constructed from this formula:

```
recommendations_for_user = [V'V] * historical_visits_
from_user
```

Here, V is the matrix of restaurant visits for all users and V' is the transpose of that matrix. [V'V] can be replaced with a co-occurrence matrix calculated with a log-likelihood ratio, which determines the strength of the similarity of rows and columns (and thus be able to pick out restaurants that other similar users have liked).

Example User / Restaurant Data (saved as restlog.csv)

USERID	ACTION	RESTAURANT
user001	like	Sage
user001	dislike	Al's_Pancake_World
user002	like	Al's_Pancake_World
user002	like	Dot's_Café
user002	like	Waffles_And_Things
user003	dislike	Al's_Pancake_World
user003	like	Sage
user003	like	Emerald
user003	neutral	Bugsy's
user004	like	Bugsy's
user004	like	Al's_Pancake_World
user004	dislike	Sage

To compute the co-occurrence matrix, we issue this command:

```
$ $MAHOUT_HOME/bin/mahout spark-itemsimilarity -i
restlog.csv -rc 0 -ic 2 -fc 1 --filter1 like -o rec_matrix
```

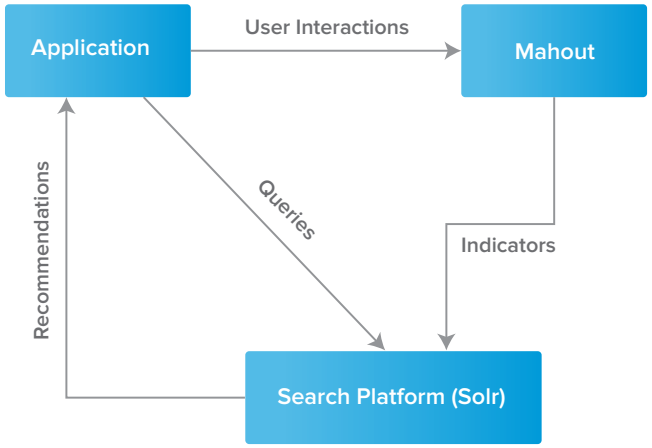
This selects the `spark-itemsimilarity` module, giving input via a filename passed into the `-i` parameter (directories and HDFS URIs are also supported here). The `-rc` parameter indicates the row column of the matrix (in this case, the userID and column 0) and the `-ic` parameter is the column in which Mahout can find our item/restaurant. As we are attempting to find similar "liked" restaurants, we add a filter by firstly using the `-fc` option to tell Mahout to filter on column 1 in the .csv file, and then by using `--filter1` to provide the text we are matching against (this can also be a regular expression, and up to two filters can be applied in this operation). Finally, the `-o` is for output, and again can be a file, directory, or HDFS URI.

Running the command produces this output:

```
Dot's_Café
Waffles_And_Things:4.498681156950466
Al's_Pancake_World:1.7260924347106847

Sage
Emerald:1.7260924347106847

Waffles_And_Things
Dot's_Café:4.498681156950466
Al's_Pancake_World:1.7260924347106847

Bugsy's
Al's_Pancake_World:1.7260924347106847

Emerald
Sage:1.7260924347106847

Al's_Pancake_World
Dot's_Café:1.7260924347106847
Waffles_And_Things:1.7260924347106847
Bugsy's:1.7260924347106847
```

Each line is a row of the collapsed similarity matrix. We can run the command with the `--omitStrength` parameter if we are just interested in a recommendation without a ranking. We can see that users who like Dot's Café will likely also enjoy Waffles and Things (and to a lesser degree, Al's Pancake World).

This approach can be used in a real-time setting by feeding the matrix data into a search platform (e.g. Solr or ElasticSearch). The Spark-Mahout algorithm can compute co-occurrences from user interactions and update the search engine's indicators, which are then fed back into the application, providing a feedback loop.



RUNNING MAHOUT FROM JAVA OR SCALA

In the last example we used Mahout from the command line. It's also possible to integrate Mahout with your Java applications. Mahout is available via Maven using the group ID org.apache.mahout. Add the following dependency to your pom.xml to get started:

```
<dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-core</artifactId>
    <version>0.10.0</version>
</dependency>
```

Common classifiers are located in the following packages:

- org.apache.mahout.classifier.naivebayes (for a Naive Bayes Classifier)
- org.apache.mahout.classifier.df (for a Decision Forest)
- org.apache.mahout.classifier.sgd (for Logistic Regression)

When using sbt (e.g. for a Scala application), add this to your library dependencies:

```
libraryDependencies ++= Seq(
[other libraries]
"org.apache.mahout" % "mahout-core" % "0.10.0"
)
```

REFERENCES

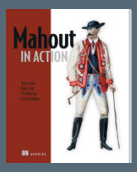
Mitchell, T. (1997). *Machine Learning*, McGraw Hill. ISBN 0070428077, p.2.

ABOUT THE AUTHORS

Ian Pointer is a Senior Consultant at Mammoth Data, a Big Data/NoSQL consulting firm based in Durham, NC. Ian specializes in Hadoop infrastructure and Spark solutions, with over 15 years of development and operations experience.

Dr. Ir. Linda Terlouw works as an Enterprise Architect and Data Scientist for ICRIS (www.icris.nl). She studied Computer Science as well as Business Information Technology at the University of Twente, the Netherlands. She has a PhD from the Delft University of Technology. After 10 years of working in Enterprise Architecture and SOA, she developed an interest in Data Science (especially Machine Learning) and Process Mining. You can contact her at linda.terlouw@icris.nl

RECOMMENDED BOOK



Sean Owen, Robin Anil, Ted Dunning, and Ellen Friedman provide an excellent in-depth explanation of the Mahout framework. This book covers the basics of machine learning and shows how to implement different algorithms using Apache Mahout. It contains many code examples and also shows how to tune for performance and move to a production environment.

BUY NOW

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 **$7.95**