AWS Big Data Blog

Building a Recommender with Apache Mahout on Amazon Elastic MapReduce (EMR)

by Accenture | on 16 JUL 2014 | in Amazon EMR, Amazon Machine Learning, AWS Big Data | Permalink | Comments | Share

This is a guest post by Andrew Musselman, who as chief data scientist leads the global big data practice from the technical side at Accenture. He is a PMC member on the Apache Mahout project and is writing a book on data science for O'Reilly. Accenture is an APN Big Data Competency Partner.

This post introduces machine learning, provides context for the Apache Mahout project, and offers some specifics about recommender systems. Then, using Amazon EMR, we'll tour the workflows for building a simple movie recommender and for writing and running a simple web service to provide results to client applications. Finally, we'll list some ways to learn more and engage with the Mahout community.

Machine Learning

Machine learning has its roots in artificial intelligence. The term implies that machine learning tools bring cognition and automated decision-making to data problems, but currently machine learning methods do not include computer thought. Even so, machine learning tools usually do employ some type of automated decision making, often iteratively working toward minimizing or maximizing a specific measurement about the performance of a model.

The field of machine learning encompasses many topics and approaches, usually falling into the categories of *classification*, *clustering*, and *recommenders*.

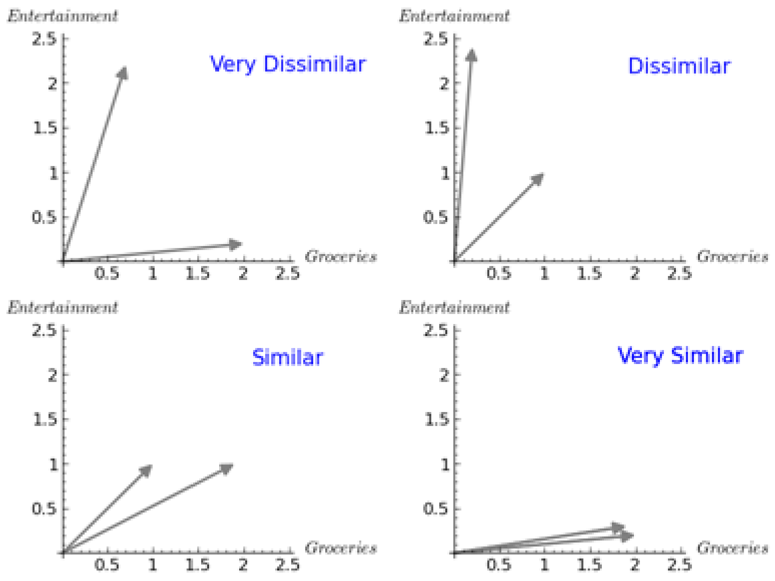
Classification is the process of predicting an unknown (dependent) variable based on a combination of other known (independent) variables, such as predicting a bank's customer attrition or predicting subscribers to a music service. In both of those cases, we use known variables about customers to predict their tendency to cancel their accounts. The following table lists several possible known variables:

Variable Type	Examples
Demographic	City, state, age, gender
Behavioral	Bank customers spending habits How often music service subscribers play certain artists

Variable Type	Examples
Environmental or Circumstantial	Fees assessed to bank customers How often music service customers experienced buffered music stream

During classification jobs, we typically use training data that contains the actual value of the dependent variable. We then assess the model's performance by applying it to a held-out testing data set where we predict the value of the dependent variable and measure how often the predictions are correct.

Clustering, is the process of finding clumps, or groupings, of things in space. Geometrically, we usually talk about clustering vectors in some *n*-dimensional space. For example, the figure below shows four pairs of people represented by vectors in two-dimensional space where each dimension is some type of spending category, in this case *entertainment* or *groceries*.



In the top-left corner are two people who spend a similar amount but whose spending habits are in very different directions, and who therefore are very dissimilar. By the same reasoning, the two pairs of people-vectors in the bottom row have similar spending habits since their spending is in a similar direction to one another. The clustering we do with machine learning tools usually involves more than two dimensions, often in the range of hundreds to thousands, which the math supports by generalizing to arbitrary finite dimensional spaces.

Key to the topic of clustering is the concept of *distance metric* or *measure* that we use to define similarity. Some common measures are *Euclidean*, which is distance "as the crow flies"*; cosine similarity,* which maps a small between-vector angle to a high (close to one) similarity, and maps a high between-vector angle to a low (close to zero) similarity, and *Tanimoto*, which intuitively measures how much two vectors have in common versus how much they could have had in common.

Recommenders are systems that take some input, usually behavior-based, and predict which items users would tend to prefer. The popularity of recommenders over the past decade was spurred largely by publicity about the Netflix Prize, which ran from 2006 through 2009 and rewarded recommendations that beat the existing Netflix systems by a stated threshold.

Recommender performance is measured by comparing predictions with actual preferences, and is often enhanced by A/B testing in production.

Apache Mahout

Apache Mahout ships with most Hadoop distributions including Apache Bigtop and EMR. Mahout is a machine-learning library with tools for clustering, classification, and several types of recommenders, including tools to calculate most-similar items or build item recommendations for users. Mahout employs the Hadoop framework to distribute calculations across a cluster, and now includes additional work distribution methods, including Spark, a computing framework originating in UC Berkeley's AMPLab that is now an Apache project.

Mahout saw its first bug filed in January 2008, and from there has seen 1,700 Jira tickets filed, with 54 open at this writing. Refining and improving the code and documentation requires a community of contributors and users; the project released a 0.10 version April 11, 2015 which prominently featured a new math environment code-named *Mahout-Samsara*.

Recommenders

Most people encounter the effect of a recommender system on a website where the recommender's predictions appear in a section of a web page. These recommendations help the site visitor find products to buy, new music to listen to, movies or television shows to watch, colleagues to hire, or potential mates to pursue.

The techniques for building recommenders have evolved since the early 1990s when the GroupLens research team built the USENET article recommender. The raw material has also evolved: in addition to news articles, there is a larger volume of online activity including clickstreams of users who follow links on websites; explicit "likes" and profile views; time spent perusing items or people; listening to music; and watching videos.

These advancements provide hooks into user behavior that help us determine how to recommend things to people. In the USENET example, users scan lists of articles for author and subject, click through, read, and close articles. In an online retail web site, shoppers search for products, browse product pages, click on photos to enlarge them, read reviews, and add products to a shopping cart. On a streaming music site, music consumers search for an artist or album, play tracks, fast-forward through tracks, and add the artist to their favorites, Streaming video sites work in a similar way. On a social networking site for professional or personal contacts, users search for and interact with other people. Each example includes a user interacting with some type of item in several ways.

Building a Recommender

To demonstrate how to build an analytic job with Mahout on EMR, we'll build a movie recommender. We will start with ratings given to movie titles by users in the MovieLens data set, which was compiled by the GroupLens team, and will use the "recommenditembased" example to find most-recommended movies for each user.

1. Sign up for an AWS account.
2. Configure the elastic-mapreduce ruby client.

3. Start up an EMR cluster (note the pricing and make sure to shut the cluster down afterward).

```
./elastic-mapreduce --create --alive --name mahout-tutorial --num-instances
```

4. Get the MovieLens data

```
wget http://files.grouplens.org/datasets/movielens/ml-1m.zip
unzip ml-1m.zip
```

Convert ratings.dat, trade "::" for ",", and take only the first three columns:

```
cat ml-1m/ratings.dat | sed 's/::/,/g' | cut -f1-3 -d, > ratings.csv
```

Put ratings file into HDFS:

```
hadoop fs -put ratings.csv /ratings.csv
```

5. Run the recommender job:

```
mahout recommenditembased --input /ratings.csv --output recommendations --n
```

6. Look for the results in the part-files containing the recommendations:

```
        hadoop fs -ls recommendations
        hadoop fs -cat recommendations/part-r-00000 | head
```

You should see a lookup file that looks something like this (your recommendations will be different since they are all 5.0-valued and we are only picking ten):

User ID	(Movie ID : Recommendation Strength) Tuples
35	[2067:5.0, 17:5.0, 1041:5.0, 2068:5.0, 2087:5.0, 1036:5.0, 900:5.0, 1:5.0, 2081:5.0, 3135:5.0]
70	[1682:5.0, 551:5.0, 1676:5.0, 1678:5.0, 2797:5.0, 17:5.0, 1:5.0, 1673:5.0, 2791:5.0, 2804:5.0]
105	[21:5.0, 3147:5.0, 6:5.0, 1019:5.0, 2100:5.0, 2105:5.0, 50:5.0, 1:5.0, 10:5.0, 32:5.0]
140	[3134:5.0, 1066:5.0, 2080:5.0, 1028:5.0, 21:5.0, 2100:5.0, 318:5.0, 1:5.0, 1035:5.0, 28:5.0]
175	[1916:5.0, 1921:5.0, 1912:5.0, 1914:5.0, 10:5.0, 11:5.0, 1200:5.0, 2:5.0, 6:5.0, 16:5.0]
210	[19:5.0, 22:5.0, 2:5.0, 16:5.0, 20:5.0, 21:5.0, 50:5.0, 1:5.0, 6:5.0, 25:5.0]

User ID	(Movie ID : Recommendation Strength) Tuples
245	[2797:5.0, 3359:5.0, 1674:5.0, 2791:5.0, 1127:5.0, 1129:5.0, 356:5.0, 1:5.0, 1676:5.0, 3361:5.0]
280	[562:5.0, 1127:5.0, 1673:5.0, 1663:5.0, 551:5.0, 2797:5.0, 223:5.0, 1:5.0, 1674:5.0, 2243:5.0]

Where the first number is a user id, and the key-value pairs inside the brackets are movie-id:recommendation-strength tuples.

The recommendation strengths are at a hundred percent, or 5.0 in this case, and should work to finesse the results. This probably indicates that there are many more than ten "perfect five" recommendations for most people, so you might calculate more than the top ten or pull from deeper in the ranking to surface less-popular items.

Building a Service

Next, we'll use this lookup file in a simple web service that returns movie recommendations for any given user.

1. Get Twisted, and Klein and Redis modules for Python.

```
        sudo easy_install twisted
        sudo easy_install klein
        sudo easy_install redis
```

2. Install Redis and start up the server.

```
        wget http://download.redis.io/releases/redis-2.8.7.tar.gz
        tar xzf redis-2.8.7.tar.gz
        cd redis-2.8.7
        make
        ./src/redis-server &
```

3. Build a web service that pulls the recommendations into Redis and responds to queries.
Put the following into a file, e.g., "hello.py"

```
from klein import run, route
import redis
import os

# Start up a Redis instance
r = redis.StrictRedis(host='localhost', port=6379, db=0)

# Pull out all the recommendations from HDFS
```

```
p = os.popen("hadoop fs -cat recommendations/part*")

# Load the recommendations into Redis
for i in p:

  # Split recommendations into key of user id
  # and value of recommendations
  # E.g., 35^I[2067:5.0,17:5.0,1041:5.0,2068:5.0,2087:5.0,
  #       1036:5.0,900:5.0,1:5.0,081:5.0,3135:5.0]$
  k,v = i.split('t')

  # Put key, value into Redis
  r.set(k,v)

# Establish an endpoint that takes in user id in the path
@route('/<string:id>')

def recs(request, id):
  # Get recommendations for this user
  v = r.get(id)
  return 'The recommendations for user '+id+' are '+v


# Make a default endpoint
@route('/')

def home(request):
  return 'Please add a user id to the URL, e.g. http://localhost:8080/1234n

# Start up a listener on port 8080
run("localhost", 8080)
```

4. Start the web service.
   `twistd -noy hello.py &`
5. Test the web service with user id “37”:
   `curl localhost:8080/37`
6. You should see a response like this (again, your recommendations will differ):
   The recommendations for user 37 are
   [7:5.0,2088:5.0,2080:5.0,1043:5.0,3107:5.0,2087:5.0,2078:5.0,3108:5.0,1042:5.0,1028:5.0]
7. When you’re finished, don’t forget to shut down the cluster:
   `./elastic-mapreduce --list`

```
j-UNIQUEJOBID      WAITING        ec2-AA-BB-CC-DD.compute-1.amazonaws.com
```

`./elastic-mapreduce --terminate j-UNIQUEJOBID`

8. Confirm shutdown:
   `./elastic-mapreduce --list`

```
j-UNIQUEJOBID     SHUTTING_DOWN     ec2-AA-BB-CC-DD.compute-1.amazonaws.com
```

   After a few minutes:
   `./elastic-mapreduce --list`

```
j-UNIQUEJOBID     TERMINATED     ec2-AA-BB-CC-DD.compute-1.amazonaws.com
```

Summary

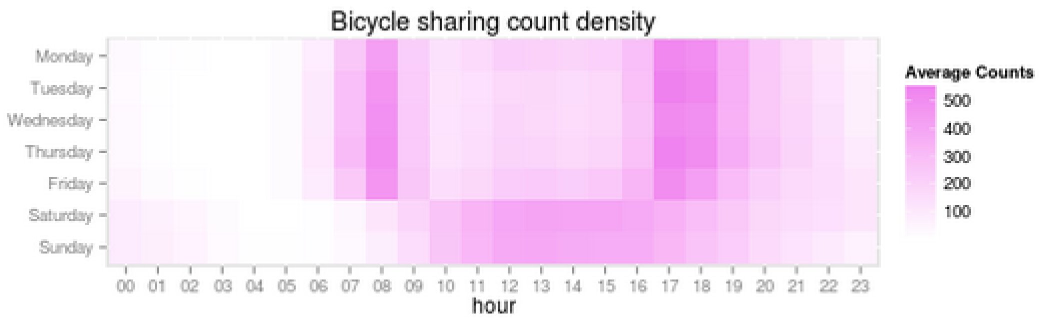
Congratulations! You've built a simple recommender that includes some of the conceptual functions and moving parts required for more sophisticated recommenders:

- Sourcing raw data
- Performing preparatory transformations
- Running an analytic job
- Interpreting the results
- Deploying results in a web service

The Mahout community actively engages with users and developers. You can get involved with Mahout by trying it on EMR, or by downloading your own copy and running it locally or on your own cluster. If you encounter bugs or features you would like to see added to the project, you can file tickets and communicate about specific issues on the project Jira page. Be sure to create an account if you'd like to see the **Create Issue** button at the top of the page.

Related:

Building a Numeric Regression Model with Amazon Machine Learning



TAGS: Amazon EMR, Amazon Machine Learning, Machine Learning

AWS Podcast

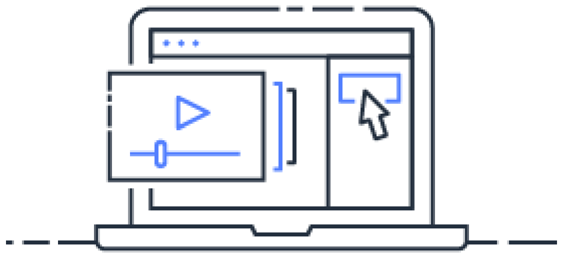
Subscribe for weekly AWS news and interviews

Learn more »

AWS Partner Network

Find an APN member to support your cloud business needs

Learn more »



AWS Training & Certifications

Free digital courses to help you develop your skills

Learn more »

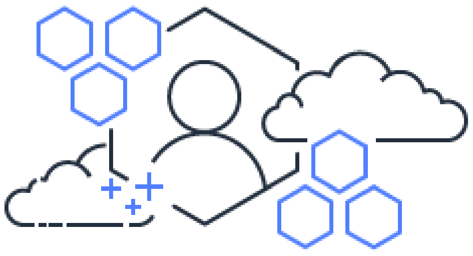
Resources

Amazon Athena
Amazon EMR
Amazon Kinesis
Amazon MSK

Amazon QuickSight
Amazon Redshift
AWS Glue

Follow

- Twitter
- Facebook
- LinkedIn
- Twitch
- Email Updates



Run Apache Airflow at Scale
Learn how to migrate your self-manged Airflow to Amazon Managed Workflows
Register now »

Related Posts

Amazon EMR 6.2.0 adds persistent HFile tracking to improve performance with HBase on Amazon S3

Perform interactive data processing using Spark in Amazon SageMaker Studio Notebooks

Discovery, Inc. is serving up content for the consumer with Food Network Kitchen

Top 9 performance tuning tips for PrestoDB on Amazon EMR

Amazon EMR 2020 year in review

Amazon MSK Backup for Archival, Replay, or Analytics

Watch the re:Invent 2020 Sessions for the Advertising and Marketing Technology Industry

CME Group accelerates cloud migration with AWS Storage Gateway