Giving Recommendations Using Apache Mahout



Presentation by
Anthony Panozzo
@panozzaj

Presentation overview

What was the problem that I was trying to solve?

How do people make recommendations?

What is Mahout?

What did I end up doing?

How could we use Mahout with Hadoop?

Problem statement

Given ratings, what are deals users want?

Challenges:

- Deal ephemerality
- City segmentation

What can we use to our advantage?

Sometimes deals will be repeated or very similar

We might know metadata like percentage off, retailer, etc.

Some users rate things similarly

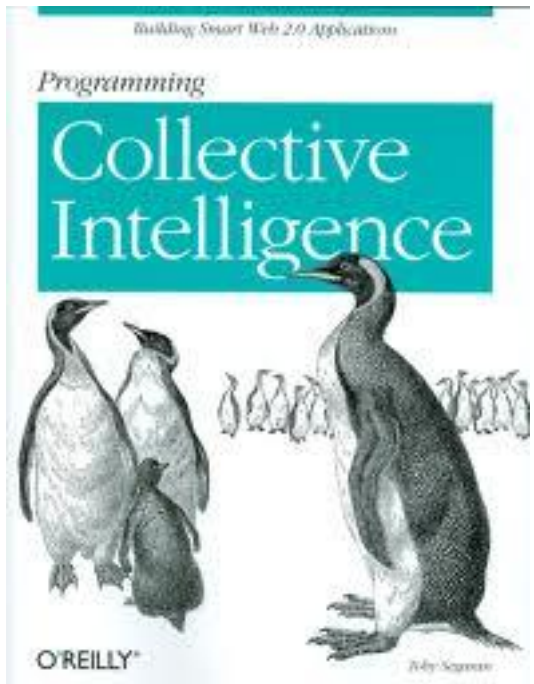
We have a double for rating, which is more fine-grained than a simple boolean value

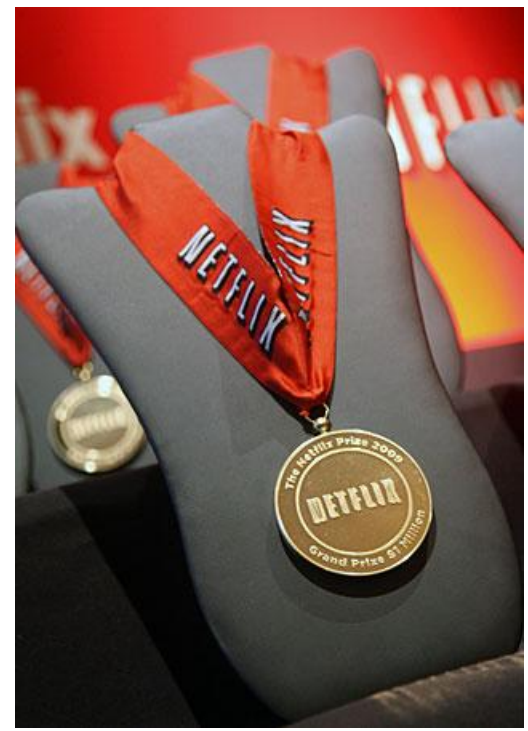
Collaborative filtering

User-based

Item-based

For starters, check out
http://en.wikipedia.org/wiki/Collaborative_filtering
http://en.wikipedia.org/wiki/Recommendation_system


Building Smart Web 2.0 Applications
Programming
Collective Intelligence
O'REILLY


NETFLIX
The Netflix Prize 2009
NETFLIX
Grand Prize $1 Million

Slope One

"Arguably, it is the simplest form of non-trivial item-based collaborative filtering based on ratings."

Benefits:

- queries are fast
- can update on the fly
- a visitor with few ratings should get good results
- fairly accurate
- easy to implement and maintain

Slope One, continued

http://lemire.me/fr/documents/publications/lemiremaclachlan_sdm05.pdf

See also:
http://books.dzone.com/articles/slope-one-recommender

for every item i
  for every other item j
    for every user u expressing preference for both i and j
      add the difference in u's preference for i and j to an average

Surely, someone has done this before....?

Apache Mahout

History of Mahout

0.1 - September 30, 2008

0.2 - January 3, 2010

0.3 - April 19, 2010

0.4 - November 2, 2010

0.5 - May 27, 2011

Trunk updated... today!

Four main use cases (currently)

Recommendation
takes user behavior and tries to predict items users might like

Clustering
takes documents and groups topically related documents

Classification
tries to assign documents to a correct category based on examples

Frequent itemset mining
takes a set of item groups and identifies which items usually appear together

Buzzword/feature dump

Collaborative Filtering
User and Item based recommenders
K-Means, Fuzzy K-Means clustering
Mean Shift clustering
Dirichlet process clustering
Latent Dirichlet Allocation
Singular value decomposition
Parallel Frequent Pattern mining
Complementary Naive Bayes classifier
Random forest decision tree based classifier
High performance java collections
A vibrant community

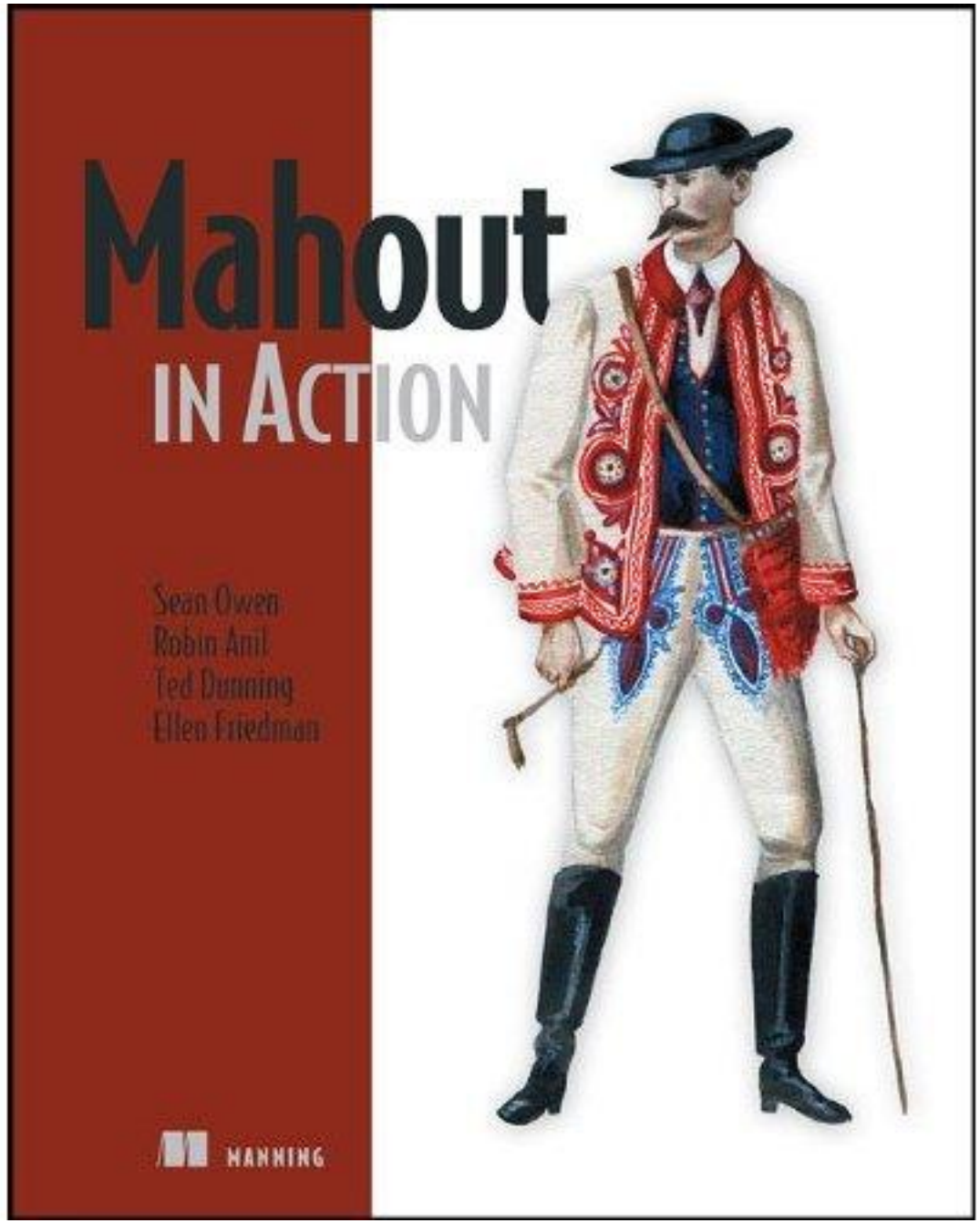
Considerations

Pros

- open source
  - use someone else's effort
  - reasonably tested
  - always improving
- map/reduce support

Cons

- open source
  - not spectacular documentation
  - API subject to change due to pre 1.0
  - no support
- grey-box
- maybe more than you need?



Mahout Inaction, er. In Action

My general solution

Connects to a PostgreSQL database to get data

Recommender using Slope One algorithm

Provide a means of evaluating results

Architecture (beta)

User

iPhone

Web Service

Recommender Engine (RecommendationEngine)
    has a Recommender (using DealRecommender)

DealRecommender (implements Recommender interface)
    has a generic Recommender object using Slope One

Data Model (standard Mahout data model loaded via Postgres)

Connect to Database

```
public DealRecommender() {
    // parameters for the dedicated database on production
    PGPoolingDataSource source = new PGPoolingDataSource();
    source.setServerName("ec2-174-XXX-XXX-XXX.compute-1.amazonaws.com");
    source.setDatabaseName("ds20XXXXXXXXXXXXX");
    source.setUser("uqXXXXXXXXXXX");
    source.setPassword("XXXXXXXXXXXXXXXXXXXXXX");

    PostgreSQLJDBCDataModel model =
        new PostgreSQLJDBCDataModel(source, "ratings", "user_id",
                                    "deal_id", "rating", "created_at");
    recommender = new DealRecommenderBuilder().buildRecommender(model);
}
```

Slope One in Mahout

```
public class DealRecommenderBuilder implements RecommenderBuilder {
    @Override
    public Recommender buildRecommender(DataModel model)
            throws TasteException {
        return new SlopeOneRecommender(model);
    }
}
```

Override methods for SlopeOneRecommender

```
@Override
public float estimatePreference(long userID, long itemID) {
    float preference = 0.0f;
    try {
        preference =
                recommender.estimatePreference(userID, itemID);
    } catch (TasteException e) {
        Logger l = Logger.getLogger("rs.DealRecommender");
        if (e instanceof NoSuchUserException) {
            // User not found in data model
            preference = averageRatingForItem(itemID);
        } else {
            // Item not found in data model
        }
    }
    return preference;
}
```

Mahout + Web interface

```
public int numberOfUsers() throws Exception {
    return recommender.getDataModel().getNumUsers();
}
```

```
public float[] estimateForDeals(long userID, long[] dealIDs) throws
Exception {
    float[] estimates = new float[dealIDs.length];
        for (int i = 0; i < dealIDs.length; i++) {
        long dealID = dealIDs[i];
        float estimate =
            recommender.estimatePreference(userID, dealID);
        estimates[i] = (Float.isNaN(estimate)) ? 0 : estimate;
    }
    return estimates;
}
```

Evaluating solution fitness

```
public double getStatistics() throws Exception {
    AverageAbsoluteDifferenceRecommenderEvaluator evaluator =
            new AverageAbsoluteDifferenceRecommenderEvaluator();
    RecommenderBuilder builder = new DealRecommenderBuilder();
    double score = evaluator.evaluate(builder, null,
            recommender.getDataModel(), 0.9, 1.0);
    return score;
}
```


hadoop

Mapper

```
public class PartialMultiplyMapper extends
Mapper<IntWritable,VectorAndPrefsWritable,VLongWritable,VectorWrita
ble> {
    public void map(IntWritable key,
                    VectorAndPrefsWritable vectorAndPrefsWritable,
                    Context context) throws
            IOException, InterruptedException {
        int itemIndex = key.get();
        Vector cooccurrenceColumn =
                vectorAndPrefsWritable.getVector();
        List<Long> userIDs = vectorAndPrefsWritable.getUserIDs();
        for (int i = 0; i < userIDs.size(); i++) {
            long userID = userIDs.get(i);
            float prefValue = prefValues.get(i);
            Vector partialProduct =
                    cooccurrenceColumn.times(prefValue);
            context.write(new VLongWritable(userID),
        }}}
```

Reducer

```
public class AggregateCombiner extends
Reducer<VLongWritable,VectorWritable,VLongWritable,VectorWritable>
{
    public void reduce(VLongWritable key,
                       Iterable<VectorWritable> values,
                       Context context)
            throws IOException, InterruptedException {
        Vector partial = null;
        for (VectorWritable vectorWritable : values) {
            partial = partial == null
                    ? vectorWritable.get()
                    : partial.plus(vectorWritable.get());
        }
        context.write(key, new VectorWritable(partial));
    }
}
```

Discussion



Presentation by
Anthony Panozzo
@panozzaj