Apache Mahout - Tutorial

Cataldo Musto, Ph.D.

Corso di Accesso Intelligente all’Informazione ed Elaborazione del Linguaggio Naturale

Università degli Studi di Bari – Dipartimento di Informatica – A.A. 2012/2013

11/01/2013

Outline

- What is Mahout ?
  - Overview
- How to use Mahout ?
  - Java snippets

Part 1

What is Mahout?

Goal

- Mahout is a **Java library**
  - Implementing Machine Learning techniques

Goal

- Mahout is a **Java library**
  - Implementing Machine Learning techniques
    - **Clustering**
    - **Classification**
    - **Recommendation**
    - **Frequent ItemSet**

Goal

- Mahout is a **Java library**
  - Implementing Machine Learning technique
    - **Clustering**
    - **Classification**
    - **Recommendation**
    - **Frequent ItemSet**
  - Scalable

What can we do?

- Currently Mahout supports mainly four use cases:
  - **Recommendation** - takes users' behavior and from that tries to find items users might like.
  - **Clustering** - takes e.g. text documents and groups them into groups of topically related documents.
  - **Classification** - learns from exisiting categorized documents what documents of a specific category look like and is able to assign unlabelled documents to the (hopefully) correct category.
  - **Frequent itemset mining -** takes a set of item groups (terms in a query session, shopping cart content) and identifies, which individual items usually appear together.

Algorithms

- **Recommendation**
  - User-based Collaborative Filtering
  - Item-based Collaborative Filering
  - SlopeOne Recommenders
  - Singular Value Decomposition

Algorithms

- **Clustering**
  - Canopy
  - K-Means
  - Fuzzy K-Means
  - Hierarchical Clustering
  - Minhash Clustering

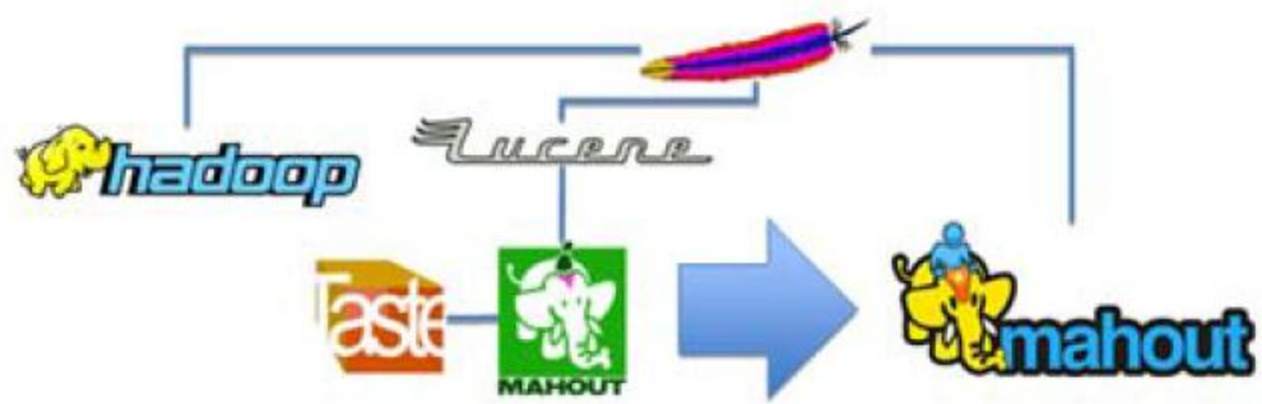
(and much more…)

Algorithms

- **Classification**
  - Logistic Regression
  - Bayes
  - Support Vector Machines
  - Perceptrons
  - Neural Networks
  - Restricted Boltzmann Machines
  - Hidden Markov Models

  (and much more…)

Mahout in the Apache Software Foundation



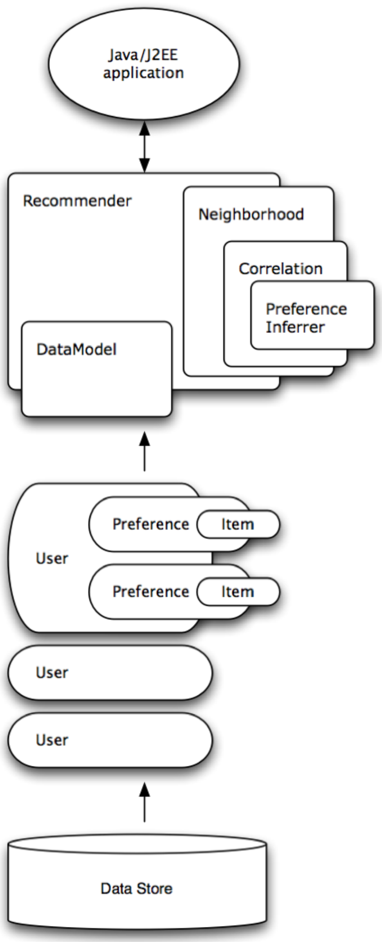
Focus

- In this tutorial we will focus just on:
  - **Recommendation**

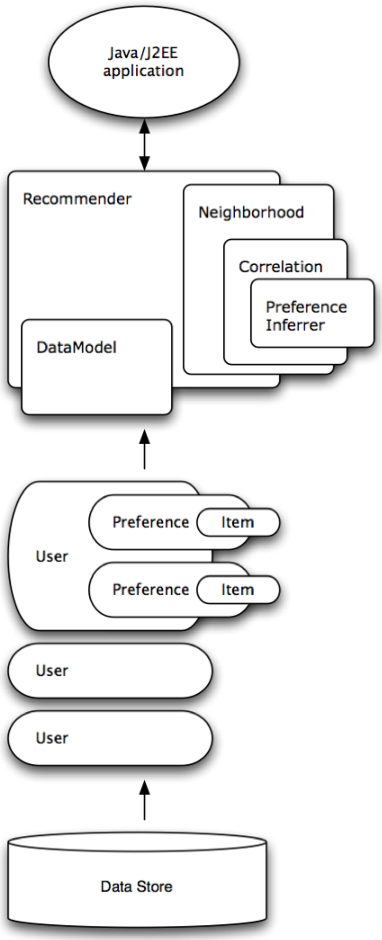
Recommendation

- Mahout implements a **Collaborative Filtering framework**
  - Popularized by Amazon and others
  - Uses hystorical data (ratings, clicks, and purchases) to provide recommendations
    - **User-based**: Recommend items by finding similar users. This is often harder to scale because of the dynamic nature of users.
    - **Item-based**: Calculate similarity between items and make recommendations. Items usually don't change much, so this often can be computed offline.
    - **Slope-One**: A very fast and simple item-based recommendation approach applicable when users have given ratings (and not just boolean preferences).

Recommendation - Architecture

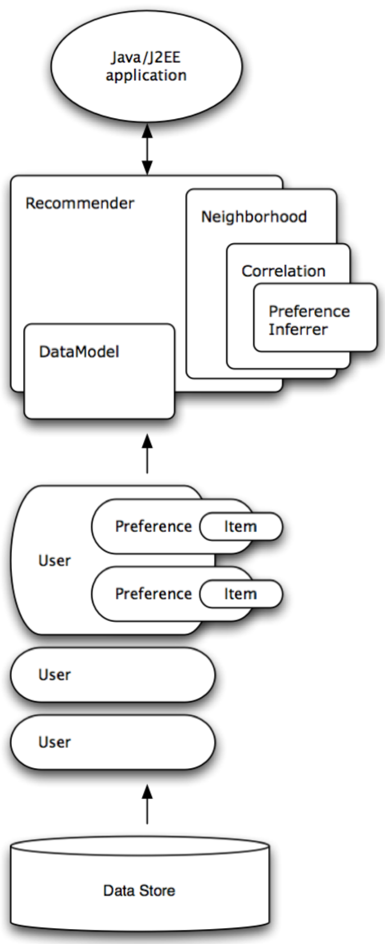


Recommendation - Architecture



Physical Storage

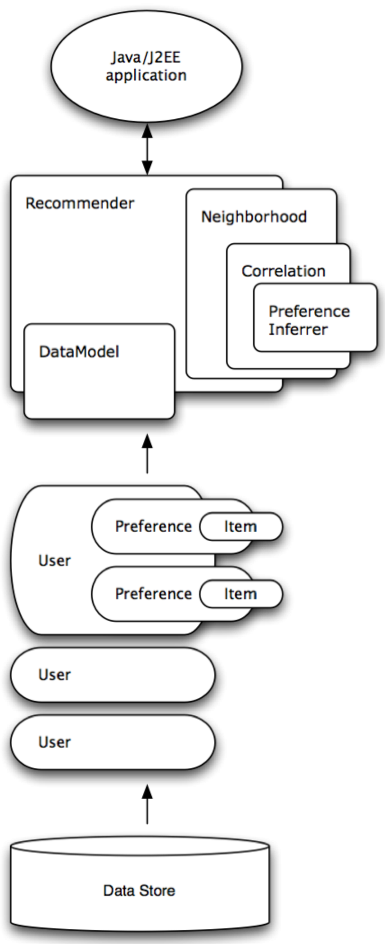
Recommendation - Architecture



Data Model

Physical Storage

Recommendation - Architecture

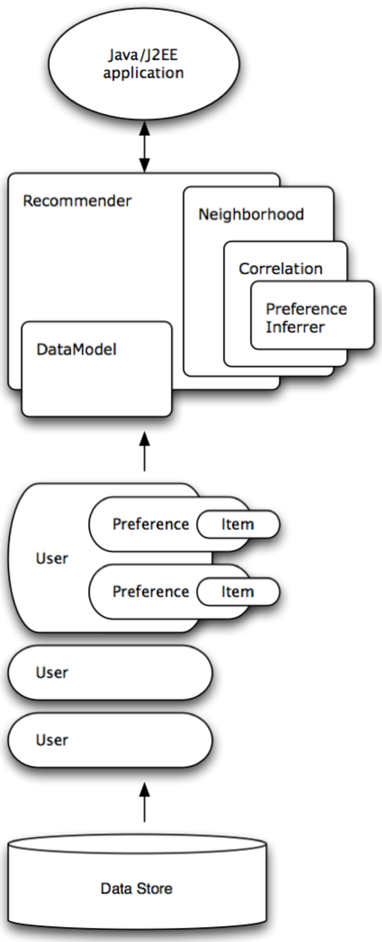


Recommender

Data Model

Physical Storage

Recommendation - Architecture



External Application

Recommender

Data Model

Physical Storage

Recommendation in Mahout

- **Input:** raw data (user preferences)
- **Output:** preferences estimation

- **Step 1**
  - Mapping raw data into a DataModel Mahout-compliant
- **Step 2**
  - Tuning recommender components
    - Similarity measure, neighborhood, etc.

Recommendation Components

- Mahout implements interfaces to these key abstractions:
  - **DataModel**
    - Methods for mapping raw data to a Mahout-compliant form
  - **UserSimilarity**
    - Methods to calculate the degree of correlation between two users
  - **ItemSimilarity**
    - Methods to calculate the degree of correlation between two items
  - **UserNeighborhood**
    - Methods to define the concept of ‘neighborhood’
  - **Recommender**
    - Methods to implement the recommendation step itself

Components: **DataModel**

- A **DataModel** is the interface to draw information about user preferences.

- Which sources is it possible to draw?
  - Database
    - MySQLJDBCDataModel
  - External Files
    - FileDataModel
  - Generic (preferences directly feed through Java code)
    - GenericDataModel

Components: **DataModel**

- **GenericDataModel**
  - Feed through Java calls
- **FileDataModel**
  - CSV (Comma Separated Values)
- **JDBCDataModel**
  - JDBC Driver
  - Standard database structure

user_id	item_id	preference
BIGINT NOT NULL	BIGINT NOT NULL	FLOAT NOT NULL
INDEX	INDEX	
PRIMARY KEY		

FileDataModel – CSV input

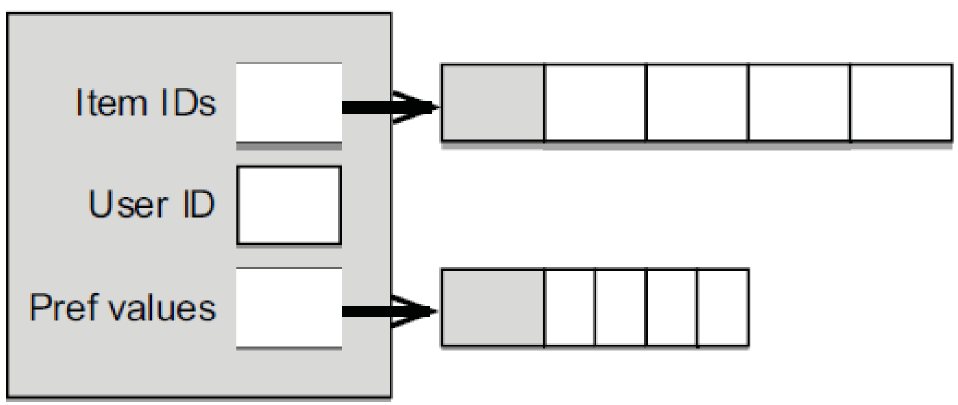
```
1,101,5.0
1,102,3.0
1,103,2.5

2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0
```

User 1 has preference 3.0 for item 102

User ID, item ID, preference value

Components: **DataModel**

- Regardless the source, they all share a common implementation.
- Basic object: **Preference**
  - Preference is a triple (user,item,score)
  - Stored in **UserPreferenceArray**



Components: **DataModel**

- Basic object: **Preference**
  - Preference is a triple (user,item,score)
  - Stored in **UserPreferenceArray**

- Two implementations
  - **GenericUserPreferenceArray**
    - It stores numerical preference, as well.
  - **BooleanUserPreferenceArray**
    - It skips numerical preference values.

Components: **UserSimilarity**

- **UserSimilarity** defines a notion of similarity between two Users.
  - *(respectively) ItemSimilarity defines a notion of similarity between two Items.*

- Which definition of similarity are available?
  - Pearson Correlation
  - Spearman Correlation
  - Euclidean Distance
  - Tanimoto Coefficient
  - LogLikelihood Similarity

  - **Already implemented!**

Pearson’s vs. Euclidean distance

	Item 101	Item 102	Item 103	Correlation with user 1
User 1	5.0	3.0	2.5	1.000
User 2	2.0	2.5	5.0	-0.764
User 3	2.5	-	-	-
User 4	5.0	-	3.0	1.000
User 5	4.0	3.0	2.0	0.945

	Item 101	Item 102	Item 103	Distance	Similarity to user 1
User 1	5.0	3.0	2.5	0.000	1.000
User 2	2.0	2.5	5.0	3.937	0.203
User 3	2.5	-	-	2.500	0.286
User 4	5.0	-	3.0	0.500	0.667
User 5	4.0	3.0	2.0	1.118	0.472

Components: **UserNeighborhood**

- **UserSimilarity** defines a notion of similarity between two Users.
  - *(respectively) ItemSimilarity defines a notion of similarity between two Items.*

- Which definition of neighborhood are available?
  - Nearest N users
    - The first N users with the highest similarity are labeled as ‘neighbors’
  - Tresholds
    - Users whose similarity is above a threshold are labeled as ‘neighbors’

  - **Already implemented!**

Components: **Recommender**

- Given a **DataModel,** a definition of **similarity** between users (items**) and a definition of neighborhood**, a recommender produces as output an **estimation of** relevance for each unseen item

- Which recommendation algorithms are implemented?
  - **User-based CF**
  - **Item-based CF**
  - **SVD-based CF**
  - **SlopeOne CF**

  (and much more…)

Recommendation Engines

Implementation	Key parameters	Key features
GenericUserBasedRecommender	• User similarity metric • Neighborhood definition and size	• Conventional implementation • Fast when number of users is relatively small
GenericItemBasedRecommender	• Item similarity metric	• Fast when number of items is relatively small • Useful when an external notion of item similarity is available
SlopeOneRecommender	• Diff storage strategy	• Recommendations and updates are fast at runtime • Requires large precomputation • Suitable when number of items is relatively small
SVDRecommender	• Number of features	• Good results • Requires large precomputation
KnnItemBasedRecommender	• Number of means (k) • Item similarity metric • Neighborhood size	• Good when number of items is relatively small
TreeClusteringRecommender	• Number of clusters • Cluster similarity definition • User similarity metric	• Recommendations are fast at runtime • Requires large precomputation • Good when number of users is relatively small

Recap

- Hundreds of possible implementations of a CF-based recommender!
  - 6 different recommendation algorithms
  - 2 different neighborhood definitions
  - 5 different similarity definitions

- Evaluation fo the different implementations is actually very time-consuming
  - The strength of Mahout lies in that it is possible to save time in the **evaluation** of the different combinations of the parameters!
  - **Standard interface for the evaluation of a Recommender System**

Evaluation

- Mahout provides classes for the evaluation of a recommender system
  - **Prediction-based measures**
    - Mean Average Error
    - RMSE (Root Mean Square Error)
  - **IR-based measures**
    - Precision
    - Recall
    - F1-measure
    - F1@n
    - NDCG (ranking measure)

Evaluation

- **Prediction-based Measures**
  - Class: AverageAbsoluteDifferenceEvaluator
  - Method: evaluate()
  - **Parameters:**
    - Recommender implementation
    - DataModel implementation
    - TrainingSet size (e.g. 70%)
    - % of the data to use in the evaluation (smaller % for fast prototyping)

Evaluation

- **IR-based Measures**
  - Class: GenericRecommenderIRStatsEvaluator
  - Method: evaluate()
  - **Parameters:**
    - Recommender implementation
    - DataModel implementation
    - Relevance Threshold (mean+standard deviation)
    - % of the data to use in the evaluation (smaller % for fast prototyping)

Part 2

How to use Mahout?

Download Mahout

- Official Release
  - The latest Mahout release is 0.7
  - Available at: http://www.apache.org/dyn/closer.cgi/mahout

- Java 1.6.x or greater.
- Hadoop is not mandatory!

Example 1: preferences

```
import org.apache.mahout.cf.taste.impl.model.GenericUserPreferenceArray;
import org.apache.mahout.cf.taste.model.Preference;
import org.apache.mahout.cf.taste.model.PreferenceArray;

class CreatePreferenceArray {

  private CreatePreferenceArray() {
  }

  public static void main(String[] args) {
    PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);

    user1Prefs.setUserID(0, 1L);

    user1Prefs.setItemID(0, 101L);
    user1Prefs.setValue(0, 2.0f);

    user1Prefs.setItemID(1, 102L);
    user1Prefs.setValue(1, 3.0f);

    Preference pref = user1Prefs.get(1);
    System.out.println(pref);
  }
}
```

Example 1: preferences

```
import org.apache.mahout.cf.taste.impl.model.GenericUserPreferenceArray;
import org.apache.mahout.cf.taste.model.Preference;
import org.apache.mahout.cf.taste.model.PreferenceArray;

class CreatePreferenceArray {

  private CreatePreferenceArray() {
  }

  public static void main(String[] args) {
    PreferenceArray user1Prefs = new GenericUserPreferenceArray(2);

    user1Prefs.setUserID(0, 1L);

    user1Prefs.setItemID(0, 101L);
    user1Prefs.setValue(0, 2.0f);

    user1Prefs.setItemID(1, 102L);
    user1Prefs.setValue(1, 3.0f);

    Preference pref = user1Prefs.get(1);
    System.out.println(pref);
  }
}
```

Score 2 for Item 101

Example 2: data model

- **PreferenceArray** stores the preferences of a single user

- Where do the preferences of all the users are stored?
  - An HashMap? **No.**
  - Mahout introduces data structures optimized for recommendation tasks
    - HashMap are replaced by **FastByIDMap**

Example 2: data model

```
import org.apache.mahout.cf.taste.impl.common.FastByIDMap;
import org.apache.mahout.cf.taste.impl.model.GenericDataModel;
import org.apache.mahout.cf.taste.impl.model.GenericUserPreferenceArray;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.model.PreferenceArray;

class CreateGenericDataModel {

  private CreateGenericDataModel() {
  }

  public static void main(String[] args) {
    FastByIDMap<PreferenceArray> preferences = new FastByIDMap<PreferenceArray>();
    PreferenceArray prefsForUser1 = new GenericUserPreferenceArray(10);
    prefsForUser1.setUserID(0, 1L);
    prefsForUser1.setItemID(0, 101L);
    prefsForUser1.setValue(0, 3.0f);
    prefsForUser1.setItemID(1, 102L);
    prefsForUser1.setValue(1, 4.5f);

    preferences.put(1L, prefsForUser1);

    DataModel model = new GenericDataModel(preferences);
    System.out.println(model);
  }
}
```

Example 3: First Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("intro.csv"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(1, 1);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

Example 3: First Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("intro.csv"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(1, 1);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

FileDataModel

Example 3: First Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("intro.csv"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(1, 1);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

2 neighbours

Example 3: First Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("intro.csv"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(1, 1);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

Top-1 Recommendation for User 1

Example 4: MovieLens Recommender

- **Download the GroupLens dataset (100k)**
  - Its format is already Mahout compliant

- Now we can run the recommendation framework against a state-of-the-art dataset

Example 4: MovieLens Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("ua.base"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(1, 20);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

Example 4: MovieLens Recommender

```
import org.apache.mahout.cf.taste.impl.model.file.*;
import org.apache.mahout.cf.taste.impl.neighborhood.*;
import org.apache.mahout.cf.taste.impl.recommender.*;
import org.apache.mahout.cf.taste.impl.similarity.*;
import org.apache.mahout.cf.taste.model.*;
import org.apache.mahout.cf.taste.neighborhood.*;
import org.apache.mahout.cf.taste.recommender.*;
import org.apache.mahout.cf.taste.similarity.*;

class RecommenderIntro {

  private RecommenderIntro() {
  }

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

    DataModel model = new FileDataModel(new File("ua.base"));
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);

    Recommender recommender = new GenericUserBasedRecommender(
        model, neighborhood, similarity);

    List<RecommendedItem> recommendations = recommender.recommend(10, 50);

    for (RecommendedItem recommendation : recommendations) {
      System.out.println(recommendation);
    }
  }
}
```

We can play with parameters!

Example 5: evaluation

Ensures the consistency between different evaluation runs.

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   System.out.println(score);
  }
}
```

Example 5: evaluation

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   System.out.println(score);
  }
}
```

Example 5: evaluation

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   System.out.println(score);
  }
}
```

70%training
(evaluation on the whole dataset)

Example 5: evaluation

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   System.out.println(score);
  }
}
```

Recommendation Engine

Example 5: evaluation

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
    RecommenderEvaluator rmse = new RMSEEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   System.out.println(score);
  }
}
```

We can add more measures

Example 5: evaluation

```
class EvaluatorIntro {

  private EvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();
    RecommenderEvaluator rmse = new RMSEEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

   double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
   double rmse = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);

   System.out.println(score);
   System.out.println(rmse);
  }
}
```

Example 6: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Precision@5 , Recall@5, etc.

Example 6: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(100, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Precision@5 , Recall@5, etc.

Mahout Strengths

- **Fast-prototyping and evaluation**
  - To evaluate a different configuration of the same algorithm we just need to update a parameter and run again.
  - Example
    - Different Neighborhood Size

Example 6b: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(200, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Set Neighborhood to 200

Example 6b: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(500, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Set Neighborhood to 500

Mahout Strengths

- **Fast-prototyping and evaluation**
  - To evaluate a different configuration of the same algorithm we just need to update a parameter and run again.
  - Example
    - Different Similarity Measure (e.g. Euclidean One)

Example 6c: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        UserSimilarity similarity = new EuclideanDistanceSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood(500, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Set Euclidean Distance

Mahout Strengths

- **Fast-prototyping and evaluation**
  - To evaluate a different configuration of the same algorithm we just need to update a parameter and run again.
  - Example
    - Different Recommendation Engine (e.g. SlopeOne)

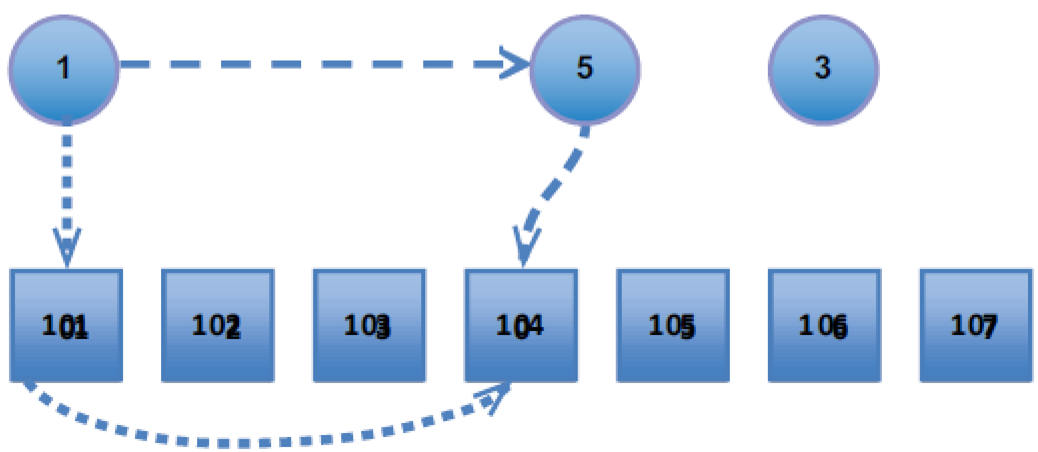
Example 6d: IR-based evaluation

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
          return new SlopeOneRecommender(model);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Change Recommendation Algorithm

Example 7: item-based recommender

- Mahout provides Java classes for building an item-based recommender system
  - Amazon-like
  - Recommendations are based on similarities among items (generally pre-computed offline)



Example 7: item-based recommender

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
        return new GenericItemBasedRecommender(model, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                        GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

Example 7: item-based recommender

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
        return new GenericItemBasedRecommender(model, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1.0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

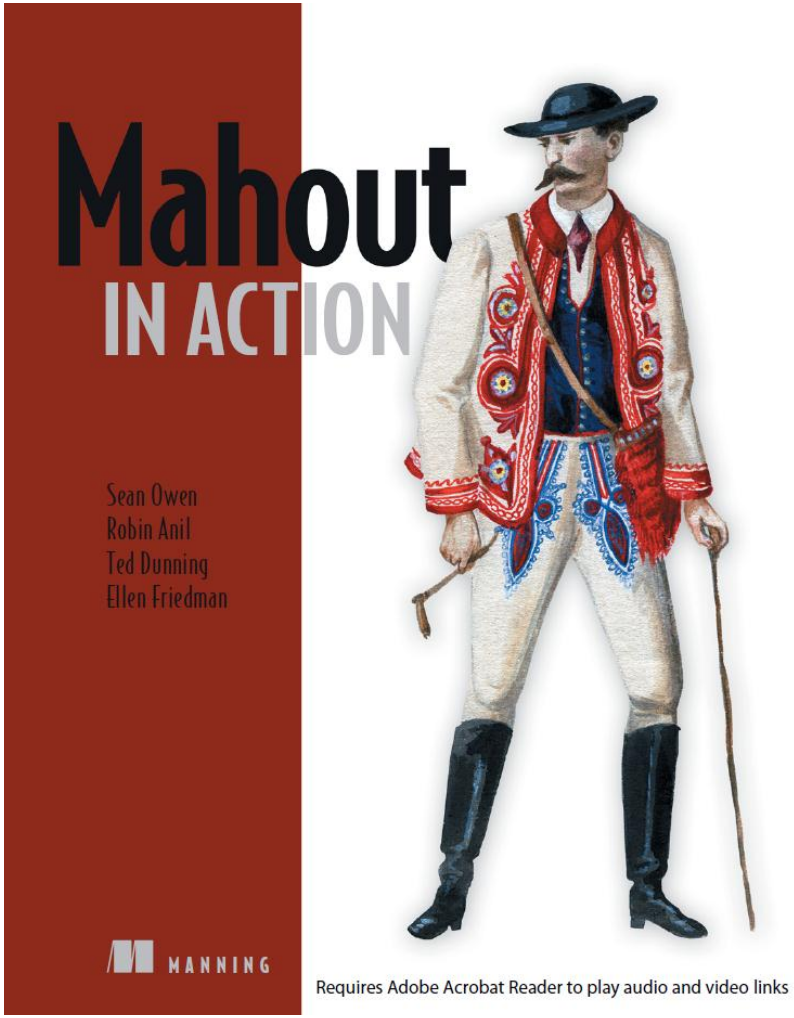
ItemSimilarity

Example 7: item-based recommender

```
class IREvaluatorIntro {

  private IREvaluatorIntro() {
  }

  public static void main(String[] args) throws Exception {
    RandomUtils.useTestSeed();
    DataModel model = new FileDataModel(new File("ua.base"));

    RecommenderIRStatsEvaluator evaluator = new GenericRecommenderIRStatsEvaluator();

    // Build the same recommender for testing that we did last time:
    RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
      @Override
      public Recommender buildRecommender(DataModel model) throws TasteException {
        ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);
        return new GenericItemBasedRecommender(model, similarity);
      }
    };

    IRStatistics stats = evaluator.evaluate(recommenderBuilder, null, model, null, 5,
                    GenericRecommenderIRStatsEvaluator.CHOOSE_THRESHOLD, 1,0);

    System.out.println(stats.getPrecision());
    System.out.println(stats.getRecall());
    System.out.println(stats.getF1());
  }
}
```

No Neighborhood definition for item-based recommenders

End. Do you want more?



Do you want more?

- Recommendation
  - Deploy of a Mahout-based Web Recommender
  - Integration with Hadoop
  - Integration of content-based information
  - Custom similarities, Custom recommenders, Re-scoring functions

- Classification, Clustering and Pattern Mining