Web Mining

Introduction to Mahout and Taste
1. Intro
2. Overview
3. Tasks
4. Other useful libraries
Intro

- **Whats Mahout?**
  - Java-Written scalable machine learning library
  - Includes variety of machine learning algorithms
    - Clustering
    - Classifier
    - And many more
  - Scales on Apache Hadoop
  - Includes Single Server Version

- **Where to get Mahout?**

- **Documentation**
  - [https://cwiki.apache.org/confluence/display/MAHOUT/Mahout+Wiki](https://cwiki.apache.org/confluence/display/MAHOUT/Mahout+Wiki)
Overview: Mahout

- Machine-learning library with focus on large-scale data
- Offers similar to rapid-miner a huge toolset of pre-build algorithms
- Runs on Hadoop (distributed system)
- Features:
  - Collaborative Filtering: User and Item Based Recommender
  - Classification: Naive Bayes, Stochastic Gradient Descent, Decision Forests
  - Clustering: k-means, fuzzy k-means
  - Frequent Pattern Mining
  - Vectorization: TF-IDF
  - Number of mathematically functions for large-scale data
First Task

- We own a movie rating network
- Users can rate movies with a 1-5 star rating
- We have some information about the user
- We have also some information about the movies

**TASK:** We want to show users, coming back to the platform movies they have not seen so far and are related to their former ratings.
First Task: What to do?

1. Get to know your data
2. Look what tools are available
3. Start with simple easy to apply solution
4. Optimize your solution
   1. Runtime
   2. Resource
   3. Special Cases
Overview: Data

- Data consists of
  - Users (with or without additional information)
  - Items (e.g. Products, with or without additional information)
  - Ratings (Link between users and items)
Data Set: MovieLens1M

- General
  - Data from grouplens web site (http://grouplens.org/)
  - Ratings of Movies (1-5) by users with some demographic data

- Data
  - Users (~6,000)
    - UserID::Gender::Age::Occupation::Zip-code
  - Movies (~4,000)
    - MovieID::Title::Genres
  - Ranking (1,000,000)
    - UserID::MovieID::Rating::Timestamp
    - Average 167 ratings/user
    - Average 250 ratings/movie

- Download
  - http://grouplens.org/datasets/movielens/
Overview: Mahout-Taste

- **Whats Taste?**
  - Collaborative Filtering and Recommender Library
  - Earlier own separated library/project - now part of Mahout
  - Can be run non-distributed and non-Hadoop-based

- **Where to get Taste?**
  - Included in Mahout distribution (e.g. 0.9)

- **Documentation**
  - [https://cwiki.apache.org/confluence/display/MAHOUT/Recommender+Documentation](https://cwiki.apache.org/confluence/display/MAHOUT/Recommender+Documentation)
Taste Recommender Architecture

- **Recommender**
  - Produce item suggestions for a given user

- **Data Model**
  - Representation of the data

- **Similarity/Correlation**
  - Defines how similarity of user or items is calculated
  - Can be exchanged for each recommender

- **Neighborhood**
  - Only for user based recommendations
  - Defines methodology to find neighborhood around a given user
Pre-Requirements to Run?

- Get source and data set
  - 1M MovieLens: [http://www.grouplens.org/system/files/ml-1m.zip](http://www.grouplens.org/system/files/ml-1m.zip) (or ILIAS)
  - Get Playground (ILIAS)

- Setup
  - Install Maven Plugin for IDE
  - Import Playground

- Use Eclipse on windows or linux to run to code
Task 1: Run „Application Mode“

- Follow Pre-Requirements
- Download project from ILIAS and import into eclipse
- Enter the locations of the three different files from the dataset
- **Complete** `RecommendationRunner.getRecommendations()`

```java
// the data model based on MovieLens
DataModel dataModel = new GroupLensDataModel(new File(RATINGS));
// pearson similarity
UserSimilarity similarity = new PearsonCorrelationSimilarity(dataModel);
// the user neighborhood
UserNeighborhood neighborhood = new NearestNUserNeighborhood(10, similarity, dataModel);
// the recommender
Recommender recommender = new GenericUserBasedRecommender(dataModel, neighborhood, similarity);
// run it for a particular user
List<RecommendedItem> recommendations = recommender.recommend(324, 20);
```

- Run the code
Results

Users Ratings

- 4.0: GoldenEye (1995) (ID: 10)
- 2.0: Muppet Treasure Island (1996) (ID: 107)
- 5.0: Star Wars: Episode IV - A New Hope (1977) (ID: 260)
- 4.0: Clear and Present Danger (1994) (ID: 349)
- 4.0: Client, The (1994) (ID: 350)
- 4.0: Lion King, The (1994) (ID: 364)
- 4.0: Speed (1994) (ID: 377)
- 3.0: In the Line of Fire (1993) (ID: 474)
- 4.0: Ghost (1990) (ID: 587)
- 4.0: Aladdin (1992) (ID: 588)
- 4.0: Snow White and the Seven Dwarfs (1937) (ID: 594)
- 1.0: Fargo (1996) (ID: 608)
- 4.0: Rock, The (1996) (ID: 733)
- 3.0: Ransom (1996) (ID: 832)

Users Recommendations

- 5.0: Babe (1995) (34)
- 5.0: Chinatown (1974) (1252)
- 5.0: Shop Around the Corner, The (1940) (3097)
- 5.0: Virgin Suicides, The (1999) (3556)
- 5.0: It's a Wonderful Life (1946) (953)
- 5.0: Lawrence of Arabia (1962) (1204)
- 5.0: Fight Club (1999) (2959)
- 5.0: Citizen Kane (1941) (923)
- 5.0: Raging Bull (1980) (1228)
- 4.5: Taxi Driver (1976) (111)
- 4.5: This Is Spinal Tap (1984) (1288)
- 4.5: Run Lola Run (Lola rennt) (1998) (2692)
- 4.5: There's Something About Mary (1998) (1923)
- 4.5: Producers, The (1968) (2300)
- 4.5: Toy Story 2 (1999) (3114)
- 4.5: Blade Runner (1982) (541)
- 4.5: Platoon (1986) (1090)
- 4.5: Boys Don't Cry (1999) (2908)

...
How to evaluate?

- Build in in Mahout – Taste

- For Example:
  AverageAbsoluteDifferenceRecommenderEvaluator
  
  - Input
    - RecommenderBuilder
      (Simply returns a recommender)
    - DataModel
      (The data which is used)
    - trainingPercentage
      (Part of the data set which is used to train the algorithm)
    - testPercentage
      (Part of the data set which is used to test the learned model)
Taste Recommender Evaluation

```java
// the data model based on MovieLens
DataModel dataModel = new GroupLensDataModel();

// create evaluator
AverageAbsoluteDifferenceRecommenderEvaluator evaluator =
    new AverageAbsoluteDifferenceRecommenderEvaluator();

// evaluate recommender
double MAE = evaluator.evaluate(new RecommenderBuilder() {
    public Recommender buildRecommender(DataModel dataModel)
        throws TasteException {
        // pearson similarity
        UserSimilarity similarity = new PearsonCorrelationSimilarity(dataModel);
        // the user neighborhood
        UserNeighborhood neighborhood =
            new NearestNUserNeighborhood(10, similarity, dataModel);
        // the recommender
        Recommender recommender =
            new GenericUserBasedRecommender(dataModel, neighborhood, similarity, false);
        return recommender;
    }, null, dataModel, 0.1, 0.1);

// print your stats
System.out.println("Mean Average Error: "+ MAE);
```

Evaluator

Similarity

Neighborhood

Recommender

Training Set Size Split(%)

Data Set Size Used(%)

Evaluator

Recommender

Similarity

Neighborhood

Training Set Size Split (%) Data Set Size Used (%)
Generic User Based Recommender

- **GenericUserBasedRecommender**

  - **Input**
    - UserSimilarity
    - UserNeighborhood
    - Optional: Rescorer (e.g. Inverse preference: a → -a)

  - **Prediction based simplified formular**
    \[
    \text{pred}(a, p) = \frac{\sum_{b \in N} \text{sim}(a, b) \times r_{b,p}}{\sum_{b \in N} \text{sim}(a, b)}
    \]

  - **Depending on used DataModel prediction might be capped (min, max)**
    - If prediction > max → prediction = max
    - If prediction < min → prediction = min

  - **Drawbacks**
    - User preference bias not taken into account
    - No normalization above similarity
User Based Recommender with adaption

- AvgUserPrefAdaptedUserBasedRecommender

  • Input
    • UserSimilarity
    • UserNeighborhood
    • Optional: Rescorer (e.g. Inverse preference: $a \rightarrow -a$)

  • Prediction based simplified formular
    \[
    \text{pred}(a, p) = \bar{r}_a + \frac{\sum_{b \in N} \text{sim}(a,b) \cdot (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} \text{sim}(a,b)}
    \]

  • Depending on used DataModel prediction might be capped (min, max)
    • If prediction > max $\rightarrow$ prediction = max
    • If prediction < min $\rightarrow$ prediction = min
Nearest N User Neighborhood

- NearestNUserNeighborhood
  - Input
    - \( N \) (Number of neighbors)
    - UserSimilarity (Similarity Measure)
    - DataModel (The model of the data)
    - Optional: minSimilarity (threshold for the minimum similarity of two users)
  - Neighborhood is based on similarity measure
    - Jaccard, Pearson, Spearman etc.
  - User are returned sorted based on similarity
Task 2: Mean Average Error Evaluation

- Use the evaluation framework shown and evaluate the first example and evaluate which combination might be the most accurate one. To do so, adjust the following 3 setscrews:
  - Different neighborhood sizes
  - Recommender
    - GenericUserBasedRecommender
    - AvgUserPrefAdaptedUserBasedRecommender
  - Different neighborhood thresholds
IR Stats Evaluator

- **GenericRecommenderIRStatsEvaluator**

- **Input**
  - RecommenderBuilder
  - DataModel
  - at \((\text{number of top recommendations taken into account})\)
  - relevanceThreshold
  - evaluationPercentage

- **Output: So called: IRStatistics**
  - Recall
  - Precision
  - Reach – *Fraction of users with predictions*
  - F1 Measure
  - FallOut
    \[
    \text{fall-out} = \frac{|\{\text{non-relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{non-relevant documents}\}|}
    \]
Task 3: Evaluation of Top-k Lists

– Rerun the experiment from before using the different setscrews and evaluate the results based on precision and recall which is returned by the `GenericRecommenderIRStatsEvaluator`. Does the optimal setup change or become more clearer by using this evaluation metrics?
Task 4: Cold-Start Problem

- Think about a new user coming to your movie portal. The user did not rate any movies yet but has registered and logged in
  - Think about possibilities how you could create a recommendation for such users?
  - Have a closer look into the provided project from ILIAS. Try out the pre-build recommender with the similarity measure using user demographics. Evaluate this approach using MAE. Where are the benefits, where are the drawbacks?
  - Think about a possibility to use the benefits of the different approaches, using demographic and rating data for a prediction?
Task 5: Additional evaluation metrics

- Beside the already used evaluation measures, in a real-world problem as your movie recommendation platform there might be more critical points:
  - Think about requirements your movie-recommender should have besides being accurate?
  - Use item-based recommendation (GenericItemBasedRecommender) and find out if this would serve your requirement according to answering times.
Taste Toolbox - What is already there?

- Taste comes with a variety of pre-build recommender (see following slides)

- Implementations are in the core project of mahout

- Almost everything which is used for recommendation is under the package `org.apache.mahout.cf.taste`.

- Packages in detail
  - Recommender: `impl.recommender`
  - Similarity Measures: `impl.similarity`
  - Neighborhood: `impl.neighborhood`
  - Evaluation: `impl.eval`
Generic Item Based Recommender

- **GenericItemBasedRecommender**
  - **Input**
    - **DataModel**
    - **ItemSimilarity**
  - **Optional:** CandidateItemStrategy
    - **Default:** PreferredItemsNeighborhoodCandidateItemsStrategy
  - **Optional:** MostSimilarItemsCandidateItemsStrategy
    - **Default:** PreferredItemsNeighborhoodCandidateItemsStrategy
  - **Predictions are based on similarity between new item and user preferend items**
    - \[ \text{pred}(a,p) = \frac{\sum_{i \in \text{rated Item}(a)} \text{sim}(i,p) \times r_{u,i}}{\sum_{i \in \text{rated Item}(a)} \text{sim}(i,p)} \]
  - Depending on used **DataModel** prediction might be capped (min, max)
    - If prediction > max → prediction = max
    - If prediction < min → prediction = min
Item User Average Recommender

- ItemUserAverageRecommender
  - Input
    - DataModel
  - Prediction does not use any similarity measure
    - \( \text{pred}(a, p) = \bar{r}_p + (\bar{r}_a - \bar{r}) \)
    - For example, say user X has not rated item Y. Item Y's average preference value is 3.5. User X's average preference value is 4.2, and the average over all preference values is 4.0. User X prefers items 0.2 higher on average, so, the estimated preference for user X, item Y is 3.5 + 0.2 = 3.7.
  - Based on ItemAverageRecommender
    - \( \text{pred}(a, p) = \bar{r}_p \)
Slope One Recommender

- **SlopeOneRecommender**
  - **Input**
    - **DataModel**
    - Optional: Switch un-/weighted recommender (default: weighted)
    - Optional: Switch un-/weighted diffs of standard deviation (default: weighted)
  - Prediction does not use any similarity or neighborhood measure

\[
pred(a, p) = \bar{r}_a + \frac{\sum_{i \in R_p} dev_{p,i}}{|R_p|} \quad \text{with} \quad dev_{p,j} = \sum_{u \in S_{p,j}(\chi)} \frac{u_p - u_j}{|S_{p,j}(\chi)|}
\]
  - \(R\) is set of relevant items for item \(p\)
  - \(S_{p,j}\) is set of ratings of uses containing item \(p\) and \(j\)
  - \(\chi\) is a training set

- Improvements by using weighting as there might be an unbalance between size of rating sets
- **Scientific paper:** [http://lemire.me/fr/documents/publications/lemiremaclachlan_sdm05.pdf](http://lemire.me/fr/documents/publications/lemiremaclachlan_sdm05.pdf)
Overview: Provided Recommender – Part I

- Generic Item Based
  - Based on Item Similarity
  - Fast as item similarity is relatively static
- Generic Boolean Pref Item Based
  - Similar to generic item based
  - Does not take preferences/ratings into account
- Item Average Recommender
  - Estimates preference for an item on the average of all known preferences
  - Simple & fast but may not produce good results
- Item User Average Recommender
  - Like Item Average Recommender but
  - Takes users average preference values into account
- Knn Item Based
  - Weights for final predicted preferences calculated using linear interpolation
Overview: Provided Recommender – Part II

• Generic User Based
  • Based on User Neighborhood

• Generic Boolean Pref User Based
  • Similar to Generic User Based Recommender
  • Does not take preferences/ratings into account

• Tree Cluster
  • Creates clusters for users and recreates recommendation per cluster

• Slope-one (Rating Based)
  • Especially good for frequently updated user preferences
  • Based on http://lemire.me/fr/abstracts/SDM2005.html

• Singular Value Decomposition Recommender
  • Uses Matrix Factorization

• Random Recommender
  • Its random 😊

• Caching Recommender
  • Wrapper that caches results of included recommender
Overview: Provided Similarity Measures

- CityBlock (Manhattan) Distance
- Euclidean Distance
- Log-Likelihood
- Pearson Correlation
- Spearman Correlation
- Tanimoto Coefficient
- Uncentered Cosine

- Generic Item Similarity
- Generic User Similarity
Outlook: Maven

- Basically a build tool
- More exactly a project management framework
- Major Feature: Dependency Management
  - Dependencies are defined in the POM-file within the project
  - Artifacts are identified by groupId, artifactId and version
  - Maven downloads the dependencies when needed

- Basic Commands
  - `mvn compile` → compiles the project
  - `mvn package` → creates war/jar/... files from compiled source
  - `mvn install` → install package into local repository

- More information
  - Detailed intro slides by Daniel Fleischhacker
  - Web http://maven.apache.org/
Outlook: Jetty

- Features
  - HTTP Server
  - HTTP Client
  - javax.servlet container
  - Open Source

- Further Readings/Information
  - http://jetty.codehaus.org/jetty/
  - http://docs.codehaus.org/display/JETTY/Jetty+Wiki
  - http://docs.codehaus.org/display/JETTY/Maven+Jetty+Plugin
Other Libraries

- **Duine**
  - [http://www.duineframework.org/](http://www.duineframework.org/)

- **Cofi**

- **LensKit**
  - [http://lenskit.grouplens.org/](http://lenskit.grouplens.org/)

- **RapidMiner Recommender Extension**
RapidMiner Recommendation Extension

- Extension for RapidMiner:

- Code-free interface by simply combining operators

- Single Maschine

- More difficult to create own operators and adjust code

- Includes Functionalities:
  - Collaborative Filtering
    - Item Based
    - User Based
  - Content-Based Recommendation
    - Item Attributes
    - User Attributes
RapidMiner Collaborative Filtering

Load Data  Specify User Id  Specify Item Id

Retrieves Movie Data  Set User Role  Set Item Role

Learn Model

Filter Examples  Item k-NN

Measure Performance

Performance Vector:
AUC: 0.223
prec@5: 0.003
prec@10: 0.003
prec@15: 0.003
NDCG: 0.439
MAP: 0.034
RapidMiner Content-Based Recommendation

Load Rating

Load User Attribute

Learn Model

Measure Performance

PerformanceVector:
AUC: 0.741
prec@5: 0.415
prec@10: 0.398
prec@15: 0.392
NDCG: 0.655
MAP: 0.201
Questions?