Clustering-based Collaborative filtering for web page recommendation

CSCE 561 project Proposal

Mohammad Amir Sharif
PhD student

mas4108@louisiana.edu
Presentation Outline

- Introduction to Recommender Systems
- Example
- Clustering based recommendation algorithm
- Implementing Clustering based webpage recommendation using mahout
- Evaluation of the developed System
- References
Information Overload

News items, Books, Journals, Research papers

TV programs, Music CDs, Movie titles

Consumer products, e-commerce items,

Web pages, Usenet articles, e-mails
Introduction

■ What is recommendation system?
  – Recommend related items
  – Personalized experiences

■ Components of a recommender system
  – Set of users, set of items (products)
  – Implicit/explicit user rating on items
  – Additional information: trust, collaboration, etc.
  – Algorithms for generating recommendations
Introduction (cont)

Recommendation techniques
- Collaborative Filtering (CF)
  - Memory-based algorithms: user-based, item-based
  - Model-based algorithms: Bayesian network; Clustering; Rule-based; Machine learning on graphs; PLSA; Matrix factorization
- Content-based recommendation
- Hybrid approaches
Clustering based Collaborative Filtering

- Problems: large-scale data; sparse rating matrix
User-Based CF Algorithm

- Find the most similar cluster for an active user
- Apply Similarity measure among current and other users
- Users’ similarities are used to predict the recommendation value of an item for active user
Experimental set up

- 13745 preprocessed user session data on 683 pages are available for this experiment.

- Dataset are available in a Matrix form of size $13745 \times 683$ where each cell represents the page view time of a user for a page in a particular session.

- Apache Mahout which works on top of Hadoop will be used to make the clustering of user sessions.

- The Apache Hadoop and mahout are open source software library for large scale distributed computing and machine learning respectively.
Experimental set up (cont)

■ Each row of the data sets

0 0 3 0 0 5 0 4 6 2 0 0 0 0 7 0

■ Vector similarity

– similarity among active session and cluster center
Experimental set up (cont)

- Similarity of user $u$ and $v$, $W_{u,v}$

\[
W_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}
\]

Here, $r_{u,i}$ is the user $u$’s pageview time for page $i$
$I$ is set of pages, $r_u$ and $r_{u,i}$ are average pageview time of user $u$ and $v$.

- Predicted Recommendation of user $a$ to item $i$, $P_{a,i}$

\[
P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U} |w_{a,u}|}
\]
Evaluation Metric

Mean Absolute Error and Normalized Mean Absolute Error:

\[
MAE = \frac{\sum_{\{i,j\}} | p_{i,j} - r_{i,j} |}{n}
\]

\[
NMAE = \frac{MAE}{r_{\text{max}} - r_{\text{min}}},
\]

Where, \( r_{\text{max}} \) and \( r_{\text{min}} \) are the upper and lower bounds of pageview time, \( p_{i,j} \) is the prediction for user \( i \) to item \( j \), \( r_{i,j} \) is the pageview time of user \( i \) to page \( j \).
References

- http://mahout.apache.org/
An Efficient Information Retrieval System

Objectives:

- Efficient Retrieval Incorporating keyword’s position; and occurrences of keywords in heading or titles in the inverted index.
- Retrieve relevant documents considering proximity (Example: “dogs” and “race” within 4 words) of query terms
- Evaluation of the system

Extension to Assignment # 3
**Inverted Index**

Index terms  $df$

<table>
<thead>
<tr>
<th>term</th>
<th>$df$</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>3</td>
</tr>
<tr>
<td>database</td>
<td>2</td>
</tr>
<tr>
<td>science</td>
<td>4</td>
</tr>
<tr>
<td>system</td>
<td>1</td>
</tr>
</tbody>
</table>

Postings lists

<table>
<thead>
<tr>
<th>term</th>
<th>$D_j$, $tf_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7, 4</td>
<td></td>
</tr>
<tr>
<td>D1, 3</td>
<td></td>
</tr>
<tr>
<td>D2, 4</td>
<td></td>
</tr>
<tr>
<td>D5, 2</td>
<td></td>
</tr>
</tbody>
</table>

Index file

<table>
<thead>
<tr>
<th>term</th>
<th>$D_j$, $tf_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cats 1 (1,1)</td>
<td>1</td>
</tr>
<tr>
<td>dogs 2 (1,2)</td>
<td>3</td>
</tr>
<tr>
<td>fish 4 (4,1)</td>
<td>1</td>
</tr>
<tr>
<td>goats 3 (3,1)</td>
<td>2</td>
</tr>
<tr>
<td>sheep 3 (2,1)</td>
<td>3</td>
</tr>
<tr>
<td>whales 1 (3,1)</td>
<td>1</td>
</tr>
</tbody>
</table>
Inverted Index (cont.)

Inverted Index stores document content in an index for fast retrieval.

- **HashMap**: Maps tokens to their occurrences.
  - **tokenHash**: String representation of the token.
  - **idf**: Double representing the inverse document frequency.
  - **occList**: ArrayList of occurrences.

- **TokenOccurrence**: Details about the occurrence of a token in a document.
  - **DocumentReference**: Contains file and length information.
  - **int count**: Count of occurrences.

- **DocumentReference** contains information about the document.
  - **File**: Path to the document.
  - **double length**: Length of the document.

- The diagram illustrates how tokens are mapped to their respective occurrences in the index.
Based on frequency, heading etc

Stores the positions of occurrences
Creating an Inverted Index

Create an empty HashMap, H;
For each document, D, (i.e. file in an input directory):
    Create a HashMapVector, V, for D;
    For each (non-zero) token, T, in V:
        If T is not already in H, create an empty
        TokenInfo for T and insert it into H;
        Create a TokenOccurrence for T in D and
        add it to the occList in the TokenInfo for T;
    Compute IDF for all tokens in H;
    Compute vector lengths for all documents in H;
Inverted-Index Retrieval Algorithm

Create a HashMapVector, Q, for the query.
Create empty HashMap, R, to store retrieved documents with scores.

For each token, T, in Q:
   Let I be the IDF of T, and K be the count of T in Q;
   Set the weight of T in Q: \( W = K \times I \);
   Let L be the list of TokenOccurences of T from H;
   For each TokenOccurrence, O, in L:
      Let D be the document of O, and C be the count of O (tf of T in D);
      If D is not already in R (D was not previously retrieved)
         Then add D to R and initialize score to 0.0;
      Increment D’s score by \( W \times I \times C \); (product of T-weight in Q and D)
Precision and Recall

**Recall**
\[
\text{recall} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of relevant documents}}
\]

**Precision**
\[
\text{precision} = \frac{\text{Number of relevant documents retrieved}}{\text{Total number of documents retrieved}}
\]
Computing Recall/Precision

<table>
<thead>
<tr>
<th>n</th>
<th>doc #</th>
<th>relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>588</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>589</td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>590</td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>717</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>984</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>772</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>321</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>498</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>628</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>772</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>592</td>
<td>x</td>
</tr>
</tbody>
</table>

- $R=1/6=0.167;\ P=1/1=1$
- $R=2/6=0.333;\ P=2/3=0.667$
- $R=3/6=0.5;\ P=3/5=0.6$
- $R=4/6=0.667;\ P=4/8=0.5$
- $R=5/6=0.833;\ P=5/9=0.556$
- $R=6/6=1.0;\ p=6/14=0.429$
Evaluation

- Considering position information the system should give better performance.
- The curve closest to the upper right-hand corner of the graph indicates the best performance.
To know more contact me

E-mail: mas4108@louisiana.edu

Thank you