Google News Personalization: Scalable Online Collaborative Filtering
Abhinandan Das, Mayur Datar, Ashutosh Garg
WWW 2007, May 8-12, 2007
Presented by: Jerry Fu
4/24/2008
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
Problem Setting

- Google news aggregates articles from several thousand news sources daily.
- Users do not know what they want, but want to see something “interesting.”
- Present several articles that are recommended specifically for user based on:
  - User click history
  - Community click history

Friday, May 9, 2008
Problem Statement

Given:
- $N$ users $U = u_1, u_2, ..., u_N$
- $M$ news articles $S = s_1, s_2, ..., s_M$
- For each user $u$, click history $C_u = h_1, h_2, ..., h_{|C_u|}$, where $h_i \in S$

Recommend $K$ stories to user $u$, within a few hundred milliseconds

Approach: collaborative filtering

Treat user clicks as noisy positive votes
A tough problem indeed
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
Memory-based algorithms

- Maintain similarity between users (common measures include Pearson correlation coefficient and cosine similarity)
- For a story $s$, calculate recommendation by weighing other user ratings with similarity
- “Ratings” in this case are binary (click or not clicked)
Model-based algorithms

- Create model for each user based on past ratings
- Use model to predict ratings on new items
- Recent work captures multiple interests of users
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
Combined Algorithm for Google News

- Use combined memory-based and model-based algorithms
- Here, model-based approaches are
  - MinHash
  - Probabilistic latent semantic indexing (PLSI)
- Memory-based approach is item covisititation
MinHash Algorithm

- Clustering method that assigns users to clusters based on their overlapping set of clicked articles
- Uses Jaccard coefficient, with every user represented by click history

\[ S(u, v) = \frac{|C_u \cup C_v|}{|C_u \cap C_v|} \]

- Recommend stores clicked on by user \( v \) to user \( u \) with weight \( S(u, v) \)
Probabilistic latent semantic indexing (PLSI)

- Users \((u \in U)\) and news stories \((s \in S)\) are random variables.

- \(Z\) is a hidden variable models the relationship between \(U\) and \(S\) as follows:

\[
p(s|u; \theta) = \sum_{z=1}^{L} p(z|u)p(s|z)
\]

- \(Z\) represents user and item communities.

- Generative model of stories \(s\) for user \(u\).
Recommendations based on covisitation

- Covisitation is defined as two stories clicked by the same user within a given time interval.
- Store as a graph with nodes at stories, edges as age discounted covisitation counts.
- Update graph (using user history) whenever we receive a click.
Combined Algorithm for Google News

- Combined memory-based and model-based algorithms
- Here, model-based approaches are
  - MinHash
  - Probabilistic latent semantic indexing (PLSI)
- Memory-based approach is item covisitation
Algorithm scores

For clustering (model) algorithms:
Score of story $s$ for user $u$
\[ r_{u,s} \propto \sum_{c: u \in c} w(u, c) \sum_{v: v \in c} I(v, s) \]

fractional membership in cluster

For covisitation (memory) algorithm:
\[ r_{u,s} \propto \sum_{t \in C_u} I(s, t) \]
$I(s,t)$ indicates whether stories $s$ and $t$ were covisited
Combined Scores

Scores for stories combined by:

\[ \sum_{a} w_{a} r_{s,a} \]

\[ w_{a} = \text{weight for algorithm } a \]
\[ r_{s,a} = \text{score for } s \text{ from algorithm } a \]

Appropriate weights are learned experimentally.
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
MapReduce Overview

- MapReduce is a method to process large amounts of data in a cluster.
- Inspired by Map and Reduce in Lisp.
- Data set split across machines (shards).
- Map produces key/value pairs.
- Key space partitioned into regions (hashed).
- Reduce merges values for key.
MapReduce Overview

- MapReduce is a method to process large amounts of data in a cluster
- Inspired by *Map* and *Reduce* in Lisp
- Data set split across machines (shards)
- *Map* produces key/value pairs
  - Ex. Counting web page accesses
  - *Emit*(URL, “1”)
MapReduce Overview (cont.)

- Key space partitioned into regions, or shards, so that *Reduce* can be performed across many machines

- *Reduce* merges the values that share same key

  - Combines the data derived in Map in an appropriate manner

  - Ex. for web page accesses, sum all values for a given URL
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
MinHash implementation

- As presented before, Jaccard similarity is infeasible to implement in this setting
- Apply Locality Sensitive Hashing (LSH), or MinHashing
- Create random permutation \( P \) of \( S \) (set of news articles)
- Calculate user hash value as index of first item in user’s click history
- Users \( u, v \) in same cluster with probability equal to their similarity, \( S(u, v) \)
MinHash Impl (cont.)

- To further refine clusters, concatenate $p$ hash keys for each user. $u, v$ in same cluster with probability $S(u, v)^p$

- High precision, low recall

- Can improve recall by hashing user to $q$ clusters

- Typical values: $p$ ranges from 2 to 4, $q$ ranges from 10-20

- Instead of permuting $S$, generate random seed value for each of the $p \times q$ hash functions
MinHash and MapReduce

- Iterate over user click history, and calculate $p \times q$ MinHash values
- Group calculated values into $q$ groups of $p$ hashes
- Concatenate $p$ MinHash values to get cluster-id
- cluster-id = key, user-id = value
MinHash and MapReduce

- Split key-value pairs into shards by hashing keys
- Sort shard by key (cluster-id), so all users mapped into same cluster appear together
- In Reduce phase, obtain cluster membership list, and inverse list (user membership in clusters)
- Prune away low membership clusters
- Store user history and cluster-id’s together
PLSI Model

Model: \( p(s|u; \theta) = \sum_{z=1}^{L} p(z|u)p(s|z) \)

- \( Z \) represents user communities and like-minded users
- Generative model of stories from users with conditional probability distributions (CPDs) \( p(z|u) \) and \( p(s|z) \)
- Learn CPDs using Expectation Maximization (EM)
PLSI EM Algorithm

- Estimate CPDs
- Minimize \[ L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \log(p(s_t|u_t; \theta)) \]
- Calculate distribution of hidden variable \( Z \)
  \[
  \text{E-step: } q^*(z; u, s; \hat{\theta}) = p(z|u, s; \hat{\theta}) = \frac{\hat{p}(s|z)\hat{p}(z|u)}{\sum_{z \in Z} \hat{p}(s|z)\hat{p}(z|u)}
  \]
- Use distribution as “weights” for calculating CPDs
  \[
  \text{M-step: } p(s|z) = \frac{\sum_u q^*(z;u,s;\hat{\theta})}{\sum_s \sum_u q^*(z;u,s;\hat{\theta})}
  \[
  p(z|u) = \frac{\sum_s q^*(z;u,s;\hat{\theta})}{\sum_z \sum_s q^*(z;u,s;\hat{\theta})}
  \]
MapReduce for EM

- Rewrite EM equations - replace $p(s \mid z)$

- E-step: $q^*(z; u, s; \hat{\theta}) = p(z \mid u, s; \hat{\theta}) = \frac{\frac{N(z, s)}{N(z)} \hat{p}(z \mid u)}{\sum_{z \in Z} \frac{N(z, s)}{N(z)} \hat{p}(z \mid u)}$

\[
N(z, s) = \sum_u q^*(z; u, s; \hat{\theta}) \\
N(z) = \sum_s \sum_u q^*(z; u, s; \hat{\theta})
\]

- Calculating $q^*$ can be performed independently for every $(u, s)$ pair in click logs

- Map loads CPDs from a single user shard and a single item shard - key
Sharding for EM

- Users and items hashed into \( R \) and \( K \) groups
- Map loads needed CPDs, calculates \( q^* \)
- key-value: \((u, q^*), (s, q^*), (z, q^*)\)

- Depending on key-value pair received, reduce calculates
  - \( N(z, s) \) if it receives \((s, q^*)\)
  - \( p(z \mid u) \) if it receives \((u, q^*)\), or \( N(z) \) for \( z \)
  - \( N(z) \) if it receives \((z, q^*)\)
PLSI on a dynamic dataset

- Model needs to be retrained whenever there are new users/items
- Approximate model by using learned values of $P(z \mid u)$
- $P(s \mid z)$ can be updated in real time by updating user clusters on a click
- New users get recommendations from covisitation algorithm
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
Making recommendations by algorithm

- Refined clusters from MinHash, weighted clusters from PLSI
- For each story in cluster, calculate score by counting clicks discounted by age
- For covisititation, recommend article $s$ by for user $u$ adding covisititation entry for each item in $C_u$ and normalizing
Generating candidates for recommendation

- Use stories from news frontend, based on story freshness, news sections, language, etc.
- Alternatively, use all stories from relevant clusters and covisitation
- Benefits of each set
Outline

- Introduction and problem
- Related work on recommendation algorithms
- Overview of combined recommendation algorithm
- Overview of MapReduce
- Algorithm implementation details
- Generation of recommendations
- System architecture
- Evaluation of system
System Architecture

Statistics Server

Personalization Server

StoryTable (cluster + covisit counts)

UserTable (user clusters, click hist)

Bigtables

User clustering (Offline) (Mapreduce)

Update profile

Read Stats

Update Stats

Read user profile

Cache/Buffer

Rank Request

Rank Reply

Click Notify

News Frontend Webserver

*Taken from http://www.sfbayacm.org/events/slides/2007-10-10-google.ppt
System Workflow

- On recommend request - FrontEnd contacts Personalization Server
  - Fetch user clusters and click history from UT
  - Fetch cluster click counts from ST
  - Calculate score for each candidate story $s$
- On story click - FrontEnd contacts Statistics Server
  - Update click histories in UT for every user cluster
  - Update covisitation counts for recent click history
Outline

Introduction and problem
Related work on recommendation algorithms
Overview of combined recommendation algorithm
Overview of MapReduce
Algorithm implementation details
Generation of recommendations
System architecture
Evaluation of system
Summary of Algorithms

- MinHash
  - Each user clustered into 100 clusters
  - Calculate user u’s score for an item s using:
    \[ \sum_{v \neq u} w(u, v) I_{v,s} \]
    where v = all users except for u,
    \[ w(u, v) = \text{similarity between } u \text{ and } v \text{ based on cluster membership} \]
    \[ I = \text{indicator of whether } v \text{ clicked on } s \]

- Correlation
  - Calculate score using same equation as MinHash
Summary of Algorithms (cont.)

- **PLSI**
  - Rating is conditional likelihood calculated from
    \[ p(s|u) = \sum_z p(z|u)p(s|z) \]
  - \( p(z|u) \) and \( p(s|z) \) estimated using EM
  - Rating always falls between 0 and 1, binarized using a threshold
Evaluation on Live Traffic

- Compare three algorithms
  - Covisitation - CVBiased
  - Combined PLSI/MinHash - CSBiased
  - Popular

- To test on live traffic
  - Generate recommendation list from each algorithm.
  - Create combined interleaved list alternating the order of the algorithms
  - Count clicks on each algorithms recommendations
Model-based algorithms win

*Taken from http://www.sfbayacm.org/events/slides/2007-10-10-google.ppt
Comparison of models
Questions?
Equations

E-step: \( q^*(z; u, s; \hat{\theta}) = p(z|u, s; \hat{\theta}) = \frac{N(z, s)}{N(z)} \hat{p}(z|u) \)

\( N(z, s) = \sum_u q^*(z; u, s; \hat{\theta}) \)

\( N(z) = \sum_s \sum_u q^*(z; u, s; \hat{\theta}) \)

\( p(z|u) = \frac{\sum_s q^*(z; u, s; \hat{\theta})}{\sum_z \sum_s q^*(z; u, s; \hat{\theta})} \)

\( r_{u_a, s_k} = \sum_{i \neq a} I_{u_i, s_k} w(u_a, u_i) \)

\( w \) similarity measure, such as Pearson correlation coefficient or cosine similarity

\( I_{u_i, s_k} \) indicates whether user \( i \) clicked on story \( k \)