Search Engine Architecture

13. Recommender Systems
Agenda

• Recommender systems
  • Content filtering
  • Collaborative filtering
    • Nearest neighbors
    • Matrix factorization
• Semester in review
Recommender Systems
Motivation

• Contrast:
  • Hit-driven economics
    • Not enough shelf space for all CDs, DVDs
    • Not enough screens to show all movies
    • Not enough channels to show all TV programs
    • Not enough spectrum to play all music
  • Cf. online distribution
    • None of these issues!
    • We can capture the long tail of options
• From scarcity of choices to abundance...
  • A solution: recommendation engines!
Types of Recommender Systems

- Hand-curated
  - Editorial lists
- Aggregates
  - Top 10
  - Recent Uploads
- Tailored to users (another long tail)
  - Amazon
  - Pandora
  - Netflix
Two Approaches

• Content filtering – e.g., Pandora
  • Find items with content similar to other items user already likes

• Collaborative filtering – e.g., Netflix
  • Nearest neighbors
    • Find items rated highly by similar users
    • Find items rated similarly to those user already likes
  • Matrix factorization
    • Decompose ratings matrix $R$ into $PQ$
    • $P, Q$ are skinny factor loadings
Content Filtering
Content Filtering

• Create feature vector for each item
  • E.g., bag of words document-term matrix
• Create user profile vector
  • E.g., weighted average of rated items
• Score candidate items
  • E.g., cosine similarity between item and user vectors
Content Filtering

- **Pros**
  - No need for data on other users
  - No cold start problem for new items
  - Model is transparent – can look at features to find out why a recommendation was made

- **Cons**
  - Feature design requires domain expertise
  - Unable to use quality judgments from other users
Collaborative Filtering
Collaborative Filtering

- Start with ratings (a.k.a. utility) matrix:

```
<table>
<thead>
<tr>
<th></th>
<th>movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 4</td>
</tr>
<tr>
<td>3</td>
<td>5 5</td>
</tr>
<tr>
<td>4</td>
<td>5 5</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2 2</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 1</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
```

Source: Stanford C246 Mining Massive Datasets
Collaborative Filtering

- Nearest neighbors
  - Find items rated highly by similar users
  - Find items rated similarly to those user already likes
- Matrix factorization
  - Latent factor model
  - Decompose ratings matrix R into PQ
    - P, Q are skinny factor loadings
Collaborative Filtering: Nearest Neighbors
Nearest Neighbors

- **User-user**
  - Find items rated highly by similar users
  - Compute user similarity with, e.g., Pearson correlation over users’ common item ratings
  - Define a user’s neighborhood N of similar users
  - Then predicted rating for an item is the weighted average of ratings over user’s neighborhood
Nearest Neighbors

- **Item-item**
  - Find items similar to those rated highly
  - Compute item similarity with, e.g., Pearson correlation over common users’ ratings
    - Cf. content filtering which uses item feature vector
  - Define item’s neighborhood $N$ of similar items
  - Predicted rating for an item is weighted average over item’s neighborhood
Nearest Neighbors

• **Pros**
  • No domain expertise needed for feature design

• **Cons**
  • Cold start problem for new items
  • Requires users to have rated the same items
    • Problematic for sparse ratings matrix (long tail!)
Collaborative Filtering: Matrix Factorization
Latent Factor Models

- The Color Purple
- Amadeus
- Ocean's 11
- The Lion King
- The Princess Diaries
- Independence Day
- Funny
- Braveheart
- Lethal Weapon

Geared towards females

Geared towards males
SVD Recap

- **Remember SVD:**
  - **A**: Input data matrix
  - **U**: Left singular vectors
  - **V**: Right singular vectors
  - **Σ**: Singular values

- **So in our case:**
  - "SVD" on Netflix data: \( R \approx Q \cdot P^T \)
  - \( A = R, \; Q = U, \; P^T = \Sigma \cdot V^T \)

\[ \hat{r}_{xi} = q_i \cdot p_x \]
Latent Factor Models

- SVD isn’t defined when entries are missing!
- Use specialized methods to find $P$, $Q$
  \[
  \min_{P,Q} \sum_{(i,x) \in R} \left( r_{xi} - q_i \cdot p_x \right)^2
  \]
  \[
  \hat{r}_{xi} = q_i \cdot p_x
  \]

- Note:
  - We don’t require cols of $P$, $Q$ to be orthogonal/unit length
  - $P$, $Q$ map users/movies to a latent space
  - The most popular model among Netflix contestants

Source: Stanford C246 Mining Massive Datasets
Latent Factor Models

- Our goal is to find $P$ and $Q$ such that:

$$\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i \cdot p_x)^2$$
Overfitting

- **Want to minimize SSE for unseen test data**
- **Idea:** Minimize SSE on **training data**
  - Want large $k$ (# of factors) to capture all the signals
  - But, **SSE on test data** begins to rise for $k > 2$

- This is a classical example of **overfitting**:
  - With too much freedom (too many free parameters) the model starts fitting noise
    - That is, it fits too well the training data and thus **not generalizing** well to unseen test data

Source: Stanford C246 Mining Massive Datasets
Regularization

- **To solve overfitting we introduce regularization:**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

\[
\min_{P,Q} \sum_{\text{training}} (r_{xi} - q_i p_x)^2 + \left[ \lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]
\]

\(\lambda_1, \lambda_2\) ... user set regularization parameters

**Note:** We do not care about the “raw” value of the objective function, but we care in \(P,Q\) that achieve the minimum of the objective

Source: Stanford C246 Mining Massive Datasets
Effect of Regularization

\[
\min_{P,Q} \sum_{\text{training}} (r_{ij} - q_i p_j)^2 + \lambda \left[ \sum_x \|p_x\|^2 + \sum_t \|q_t\|^2 \right]
\]

\[
\min_{\text{factors}} \text{“error”} + \lambda \text{“length”}
\]

Source: Stanford C246 Mining Massive Datasets
Review: Recommender Systems

- Content filtering
  - Find content similar to that user already likes
- Collaborative filtering
  - Nearest neighbors
    - Find items rated highly by similar users
    - Find items rated similarly to those user already likes
- Matrix factorization
  - Latent factor model
  - Decompose ratings matrix $R$ into $PQ$
    - $P$, $Q$ are skinny factor loadings
  - Item2vec? User2vec?
Semester In Review
1. Big Ideas

• Scale out, not up
• Assume failures will happen
• Good APIs hide system details
• Aim for ideal scalability
• Move code to the data
• Avoid random disk access
2. NoSQL

- Key ideas:
  - Partition – for scalability, latency
  - Replicate – for availability, throughput
  - Caching – for latency
- Key-value stores
  - Consistent hashing, hash rings
- Bigtable / LSM trees
- CAP theorem
3. Modeling and Evaluation

- Language models
- Preprocessing
  - Case folding, tokenization, stopwords, stemming
- Boolean retrieval
- Ranked retrieval
  - Vector space model, TF-IDF, cosine similarity
- Model evaluation
  - Unranked – precision, recall, F-measure
  - Ranked – MAP, nDCG
4. Indexing and Retrieval

- Inverted index
  - TF-IDF
  - Positional
- Retrieval
  - Document-at-a-time vs. term-at-a-time
  - Postings list encoding (d-gaps)
- Partitioning
  - Term vs. document partitioning
5. MapReduce

- Constrained API helps with synchronization problems
- Map, combine, partition, shuffle and sort, reduce
- Data locality – pairs and stripes
- Inverted index construction
- Value-to-key conversion
- The datacenter *is* the computer!
6. Link Analysis

- Graph representation
- Shortest path
  - MapReduce – parallel BFS
- PageRank
  - Time on page under random surfer model
  - Static prior for ranking
  - Computed iteratively
- PageRank in MapReduce
  - Iterative algorithms are hard in MapReduce
7. Classification

- Supervised classification in sklearn
- Logistic regression
- Gradient descent
  - MapReduce – M partial gradients, 1 model update
- Stochastic gradient descent
- Ensemble methods
  - MapReduce implementation – mappers only
- Case study: GoogLeNet 2014
8. Clustering

• For exploratory analysis, recommender systems, preprocessing, ...

• Hierarchical agglomerative clustering
  • Start with N clusters, merge until one

• K-means
  • Iteratively recompute centroids and reassign points
  • MapReduce – map: assign, reduce: new centroids

• Gaussian mixture models
  • Soft assignment of points to clusters
  • MapReduce – similar to K-means
9. Distributed Word Representations

- Distributed representations / distributional hypothesis
- Dimensionality reduction
- Artificial neural networks
- Representation learning
- Word2vec
  - Skip-gram
  - CBOW
- Doc2vec
- SVD reduction
10. Learning to Rank

- ML vs. IR
- Classification
  - Predict class of query-document pair
- Pointwise learning
  - Learn thresholds to separate ranks
- Pairwise learning
  - Turns ordinal regression into binary classification
- Issues
  - Cost sensitivity for high-ranked documents
  - Query normalization
11. Beyond MapReduce

- Addressing MapReduce criticisms
  - Schemas with Thrift
  - High-level languages – Hive, Pig
- Dataflow – DAG of transformations
- Spark
  - RDD – store transforms needed to reproduce data
- Pregel
  - Graph-centric, express graph algorithms naturally
  - Each vertex passes messages to neighbors
  - Synchronization via supersteps
12. Streams

- Sampling
- Hashing
  - Set cardinality – HyperLogLog counter
  - Set membership – Bloom filter
  - Frequency estimation – Count-min sketch
- Storm
  - Spouts, bolts, and clever tracking
- Spark Streaming
  - Small, deterministic batch jobs
- Dataflow
  - Windows, triggers, and incremental processing
13. Finding Similar Items

• Represent documents with short signatures
  • Minhash
    • Given hash function, find term with smallest hash value
    • \(P[h1(D_1) = h2(D_2)] = \text{Jaccard}(D_1, D_2)\)

• Find candidates that are likely similar
  • Compute \(k\) minhashes per document (“band”)
  • Documents that match in a band are candidates
    • Evaluate candidates thoroughly
  • Repeat for \(n\) bands
14. Recommender Systems

- **Content filtering**
  - Find content similar to that user already likes

- **Collaborative filtering**
  - Nearest neighbors
    - Find items rated highly by similar users
    - Find items rated similarly to those user already likes

- **Matrix factorization**
  - Latent factor model
  - Decompose ratings matrix R into PQ
    - P, Q are skinny factor loadings
Thank you!