Search Engine Architecture

13. Recommender Systems
Today’s Agenda

• Quick LSH review
• Recommender systems
  • Content filtering
  • Collaborative filtering
    • Nearest neighbors
    • Matrix factorization
• Semester in review
Review: Minhash

A = \{e_1, e_3, e_7\}
B = \{e_3, e_5, e_7\}

<table>
<thead>
<tr>
<th>Element</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e_2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e_5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>e_3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>e_4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>e_1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• Permute set elements
• \(h(A)\) is the first row whose element is in set A

\[ P[h(A) = h(B)] = J(A, B) \]

• But no need to permute!
• Just find set element with minimum hash value
Review: Locality-Sensitive Hashing

- Represent documents with short signatures
  - Minhash
    - Given hash function $h_1$, find term with smallest hash value
    - $P[h_1(D_1) = h_1(D_2)] = \text{Jaccard}(D_1, D_2)$
- Find candidates that are likely similar
  - Compute $k$ minhashes per document (“band”)
  - Documents that match in a band are candidates
  - Repeat for $n$ bands
  - Probability we will miss a pair is $(1 - s^k)^n$
  - Probability that any pair is invalid is $n(1 - s)^k$
- Evaluate pairs of candidates thoroughly
Why not sample?

- Since \( J(A, B) = \frac{|A \cap B|}{|A \cup B|} \),

  we should be able to draw from \( A \cup B \) and count how often the sample is in \( A \cap B \)

- For \( N=2 \) documents, this makes sense

- But as \( N \) grows:
  - Union of document terms gets increasingly sparse, less useful
  - Need to sample from large, sparse set
  - More collisions (all zero inputs) in banding procedure
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:
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
    - C.f. 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
- Braveheart
- Ocean's 11
- Lethal Weapon
- The Lion King
- Independence Day
- Dumb and Dumber

Geared towards females

Geared towards males

Serious

Funny
Remember SVD:

- **A**: Input data matrix
- **U**: Left singular vecs
- **V**: Right singular vecs
- **Σ**: Singular values

So in our case:

"SVD" on Netflix data: \( R \approx Q \cdot P^T \)

\[
A = R, \quad Q = U, \quad P^T = \Sigma \cdot V^T
\]

\[
\hat{r}_{xi} = q_i \cdot p_x
\]
SVD isn’t defined when entries are missing!

Use specialized methods to find $P, Q$

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

$$\hat{r}_{xi} = q_i \cdot p_x$$

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
Our goal is to find \( P \) and \( Q \) such that:

\[
\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - q_i \cdot p_x)^2
\]
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
To solve overfitting we introduce regularization:

- Allow rich model where there are sufficient data
- Shrink aggressively where data are scarce

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

\(\lambda_1, \lambda_2\) 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.
The Effect of Regularization

Factors

- **Factor 1**: Serious vs. Funny
  - Serious: The Color Purple, The Princess Diaries
  - Funny: Dumb and Dumber, Lethal Weapon

- **Factor 2**: Female vs. Male
  - Female: Geared towards females
  - Male: Geared towards males

Mathematical Formulation:

\[
\min_{P, Q} \sum_{i \text{ training}} (r_{xi} - q_i p_x)^2 + \lambda \left( \sum_{x} \|p_x\|^2 + \sum_{i} \|q_i\|^2 \right)
\]

\[
\min_{\text{factors}} \text{“error”} + \lambda \text{“length”}
\]
The Effect of Regularization

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

Minimize factors “error” + \( \lambda \) “length”
Stochastic Gradient Descent

- **Stochastic gradient decent:**
  - Initialize $P$ and $Q$ (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

  **For each** $r_{xi}$:
  - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$  
    (derivative of the “error”)
  - $q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi} p_x - \lambda_2 q_i)$  
    (update equation)
  - $p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi} q_i - \lambda_1 p_x)$  
    (update equation)
  - $\mu$  
    learning rate

- **2 for loops:**
  - For until convergence:
    - For each $r_{xi}$
      - Compute gradient, do a “step”
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
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. query-at-a-time
  - Postings list encoding (d-gaps)
- Partitioning
  - Term vs. document partitioning
5. MapReduce

- Highly-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. Learning to Rank

- ML vs. IR – like ships passing in the night
- 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
10. 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
11. Finding Similar Items

• Represent documents with short signatures
  • Minhash
    • Given hash function, find term with smallest hash value
    • \( P[h_1(D_1) = h_2(D_2)] = \text{Jaccard}(D_1, D_2) \)
• 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
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
13. 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!