CS345 Data Mining

Recommendation Systems
Netflix Challenge
Course Projects

From scarcity to abundance
- Shelf space is a scarce commodity for traditional retailers
  - Also: TV networks, movie theaters, ...
- The web enables near-zero-cost dissemination of information about products
  - From scarcity to abundance
- More choice necessitates better filters
  - Recommendation engines
  - How Into Thin Air made Touching the Void a bestseller

The Long Tail

Recommendation Types
- Editorial
- Simple aggregates
  - Top 10, Most Popular, Recent Uploads
- Tailored to individual users
  - Amazon, Netflix, ...

Formal Model
- $C =$ set of Customers
- $S =$ set of Items
- Utility function $u: C \times S \rightarrow R$
  - $R =$ set of ratings
  - $R$ is a totally ordered set
  - e.g., 0-5 stars, real number in $[0,1]$
Utility Matrix

<table>
<thead>
<tr>
<th></th>
<th>King Kong</th>
<th>LOTR</th>
<th>Matrix</th>
<th>Nacho Libre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>0.5</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td>0.2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>David</td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

Key Problems

- Gathering "known" ratings for matrix
- Extrapolate unknown ratings from known ratings
  - Mainly interested in high unknown ratings
- Evaluating extrapolation methods

Gathering Ratings

- Explicit
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- Implicit
  - Learn ratings from user actions
  - e.g., purchase implies high rating
  - What about low ratings?

Extrapolating Utilities

- Key problem: matrix U is sparse
  - most people have not rated most items
- Three approaches
  - Content-based
  - Collaborative
  - Hybrid

Content-based recommendations

- Main idea: recommend items to customer C similar to previous items rated highly by C
- Movie recommendations
  - recommend movies with same actor(s), director, genre, ...
- Websites, blogs, news
  - recommend other sites with "similar" content

Plan of action

- Item profiles
  - Red Circles
  - Triangles
- User profile
  - match
  - build
- likes
- recommend
Item Profiles

- For each item, create an item profile
- Profile is a set of features
  - movies: author, title, actor, director, ...
  - text: set of "important" words in document
- How to pick important words?
  - Usual heuristic is TF.IDF (Term Frequency times Inverse Doc Frequency)

TF.IDF

\[ f_{ij} = \text{frequency of term } t_i \text{ in document } d_j \]

\[ TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \]

\[ n_i = \text{number of docs that mention term } i \]
\[ N = \text{total number of docs} \]

\[ IDF_i = \log \frac{N}{n_i} \]

TF.IDF score \( w_{ij} = TF_{ij} \times IDF_i \)
Doc profile = set of words with highest TF.IDF scores, together with their scores

User profiles and prediction

- User profile possibilities:
  - Weighted average of rated item profiles
  - Variation: weight by difference from average rating for item
  - ...
- Prediction heuristic
  - Given user profile \( c \) and item profile \( s \),
    estimate \( u(c,s) = \cos(c,s) = c.s/(||c|| ||s||) \)
  - Need efficient method to find items with high utility: later

Model-based approaches

- For each user, learn a classifier that classifies items into rating classes
  - liked by user and not liked by user
    - e.g., Bayesian, regression, SVM
- Apply classifier to each item to find recommendation candidates
- Problem: scalability
  - Won't investigate further in this class

Limitations of content-based approach

- Finding the appropriate features
  - e.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

Collaborative Filtering

- Consider user \( c \)
- Find set \( D \) of other users whose ratings are "similar" to \( c \)'s ratings
- Estimate user's ratings based on ratings of users in \( D \)
Similar users
- Let \( r_x \) be the vector of user \( x \)'s ratings
- Cosine similarity measure
  \[ \text{sim}(x,y) = \cos(r_x, r_y) \]
- Pearson correlation coefficient
  \[ S_{xy} = \text{items rated by both users } x \text{ and } y \]
  \[ \text{sim}(x,y) = \frac{\sum_{s \in C_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in C_{xy}} (r_{xs} - \bar{r}_x)^2 (r_{ys} - \bar{r}_y)^2}} \]

Rating predictions
- Let \( D \) be the set of \( k \) users most similar to \( c \) who have rated item \( s \)
- Possibilities for prediction function (item \( s \)):
  \[ r_{cs} = \frac{1}{k} \sum_{d \in D} r_{ds} \]
  \[ r_{cs} = \frac{\sum_{d \in D} \text{sim}(c,d) \times r_{ds}}{\sum_{d \in D} \text{sim}(c,d)} \]
- Other options?
- Many tricks possible...

Complexity
- Expensive step is finding \( k \) most similar customers
  \( O(|U|) \)
- Too expensive to do at runtime
  - Need to pre-compute
- Naive precomputation takes time \( O(N|U|) \)
  - Simple trick gives some speedup
- Can use clustering, partitioning as alternatives, but quality degrades

Pros and cons of collaborative filtering
- Works for any kind of item
  - No feature selection needed
- New user problem
- New item problem
- Sparsity of rating matrix
  - Cluster-based smoothing?

Item-Item Collaborative Filtering
- So far: User-user collaborative filtering
- Another view
  - For item \( s \), find other similar items
  - Estimate rating for item based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model
- In practice, it has been observed that item-item often works better than user-user

Hybrid Methods
- Implement two separate recommenders and combine predictions
- Add content-based methods to collaborative filtering
  - item profiles for new item problem
  - demographics to deal with new user problem
- Filterbots
### Evaluating Predictions
- Compare predictions with known ratings
  - Root-mean-square error (RMSE)
- Another approach: 0/1 model
  - Coverage
    - Number of items/users for which system can make predictions
  - Precision
    - Accuracy of predictions
  - Receiver operating characteristic (ROC)
    - Tradeoff curve between false positives and false negatives

### Problems with Measures
- Narrow focus on accuracy sometimes misses the point
  - Prediction Diversity
  - Prediction Context
  - Order of predictions

### Finding similar vectors
- Common problem that comes up in many settings
- Given a large number N of vectors in some high-dimensional space (M dimensions), find pairs of vectors that have high cosine-similarity
- Compare to min-hashing approach for finding near-neighbors for Jaccard similarity

### Similarity-Preserving Hash Functions
- Suppose we can create a family $\mathcal{F}$ of hash functions, such that for any $h \in \mathcal{F}$, given vectors $x$ and $y$:
  - $Pr[h(x) = h(y)] = \operatorname{sim}(x, y) = \cos(x, y)$
  - We could then use $E_{h\in\mathcal{F}}[h(x) = h(y)]$ as an estimate of $\operatorname{sim}(x, y)$
  - Can get close to $E_{h\in\mathcal{F}}[h(x) = h(y)]$ by using several hash functions

### Similarity metric
- Let $\theta$ be the angle between vectors $x$ and $y$
- $\cos(\theta) = \frac{x \cdot y}{||x|| ||y||}$
- It turns out to be convenient to use $\operatorname{sim}(x, y) = 1 - \theta/\pi$
  - instead of $\operatorname{sim}(x, y) = \cos(\theta)$
  - Can compute $\cos(\theta)$ once we estimate $\theta$

### Random hyperplanes
- Vectors $u$, $v$ subtend angle $\theta$
- Random hyperplane through origin (normal $r$)
  - $h_r(u) = 1$ if $r \cdot u \geq 0$
  - $0$ if $r \cdot u < 0$
Random hyperplanes

\[ h_r(u) = 1 \text{ if } r.u \geq 0 \]
\[ 0 \text{ if } r.u < 0 \]

\[ Pr[h_r(u) = h_r(v)] = 1 - \theta/\pi \]

Vector sketch

- For vector \( u \), we can construct a \( k \)-bit sketch by concatenating the values of \( k \) different hash functions
  - \( \text{sketch}(u) = [h_1(u) \ h_2(u) \ldots h_k(u)] \)
- Can estimate \( \theta \) to arbitrary degree of accuracy by comparing sketches of increasing lengths
- Big advantage: each hash is a single bit
  - So can represent 256 hashes using 32 bytes

Picking hyperplanes

- Picking a random hyperplane in \( M \)-dimensions requires \( M \) random numbers
- In practice, can randomly pick each dimension to be +1 or -1
  - So we need only \( M \) random bits

Finding all similar pairs

- Compute sketches for each vector
  - Easy if we can fit random bits for each dimension in memory
  - For \( k \)-bit sketch, we need \( M k \) bits of memory
  - Might need to use ideas similar to page rank computation (e.g., block algorithm)
- Can use DCM or LSH to find all similar pairs

Project Ideas...

- Compare algos for near-duplicates
- Netflix
- Extracting relations, list-building
- Discovering synonyms, spelling variants
- Spam detection
- Identifying website boundaries
- and many, many others...