Collaborative Filtering on Very Sparse Graphs
A Recommendation System for Yelp.com

Team 54
Phil Chen and Dan Posch
pcchen,dcposch@cs.stanford.edu

2012 Nov 15

Abstract

We have developed a recommendation system for the Yelp graph. Yelp is a popular restaurant and business review site. We are modeling Yelp's data as a bipartite graph. Nodes representing users have edges to nodes representing businesses; the edges are reviews.

Yelp currently lacks a collaborative filtering recommendation system, despite having a large dataset. We researched and tested different filtering methods, including a novel method that uses Yelp metadata, to either create the first Yelp recommendation system or understand why none exists.

Ultimately, we did both. We have a recommendation system that performs slightly better than an optimized baseline, and which performs reasonably on an absolute scale. We also learned that Yelp's graph is unusual–it is very sparse, has a large diameter, and it is clustered into nearly disjoint local graphs. Each of these properties make it challenging to achieve accurate collaborative filtering.

Introduction

Users today often have a choice among a vast set of products–for example, all books on Amazon or all restaurants in San Fransisco. The goal of collaborative filtering systems is to make these choices easier. In the typical scenario, users provide information regarding their own preferences. The system uses all of the preference data together to make recommendations.

Collaborative filtering comes in two primary forms, **neighborhood models** and **latent-factor models**. Neighborhood models are either **user-centric** or **item-centric**, locating users with similar tastes and items with similar audiences respectively. To predict a user u's preference for item i, neighborhood models aggregate existing ratings from similar users to the same item, or conversely from user u to items similar to i. **Latent factor models** represent a different approach. Here, the system learns a vector of factors for each user and each item. These vectors of factors encode what kinds of users like which types of item. Latent factor models generally express predicted ratings in terms of linear algebra and employ global optimization to find the factor matrices. (For example, they might optimize globally for minimum RMSE.) They have the advantage of good performance, often superior to neighborhood models. Neighborhood models, on the other hand, have the advantage that they provide a natural way to explain recommendations to users: in the case of item-item neighborhood models, a product is recommended because the user previously liked similar products. They also have the advantage of incremental training: a new rating can be incorporated immediately to provide new recommendations, without requiring global retraining.

We discovered that Yelp data has unusual properties that make it difficult to apply standard collaborative filtering approaches. The data is sparse for both users and items–a typ-

ical restaurant on Yelp has far fewer ratings than, for example, a successful movie on Netflix.

Related Work

There is a strong commercial motive for good user recommendations. As a result, there has been a sizeable amount of research, both commercial and academic, into collaborative filtering methods. Yehuda Koren, winner of the Netflix Challenge, unifies the two primary approaches to collaborative filtering, neighborhood models and latent-factor models [**?**]. His models for both disciplines of collaborative filtering lay the foundations for our models.

Su and Khoshgoftaar provide an extensive overview of collaborative filtering recommendation systems [**?**]. Notably, they list tradeoffs of the various categories of CF. They point out that memory-based CF algorithms, like neighbor models, tend to suffer from the cold start problem - the lack of data for new users and items that limit the recommendations for that user as well as performance decreases for sparse data. On the other hand, model-based CF algorithms, such as the latent factor models often handle sparsity better, although they are more computationally expensive to build.

Paterek examines singular value decomposition (SVD) as a form of CF. He examines regularized SVD models, as well as post-processing the SVD data, and attempts to evaluate these models on the Netflix data. He post-processes using KNN and kernel ridge regression.

Data Collection and Summary Statistics

We worked with the Yelp Academic Dataset. This consists of 130873 users, 13490 businesses, and 330071 reviews. The data set is divided into 30 university towns, each with their own set of businesses. We model users and businesses as nodes, with reviews as directed edges. User meta-data includes names, total reviews, and average stars. Business meta-data includes category, total reviews, location, and stars. Review meta data includes review text, date, and stars.

We began by exploring our dataset. The distributions of key data is shown in Figure 1. We note some important facts about that data. The distribution of degrees for businesses appear to follow a power law distribution, while the distribution for users falls off faster. The distribution of ratings is negative skewed–ratings tend to more positive than negative. The graph is very sparse (99.98% sparsity).

Methods

We implemented and tested four models for recommending Yelp businesses to users.

- Item-item neighborhood model
- Iterative Singular Value Decomposition, a factor model
- Koren's hybrid approach. This is a neighborhood method that trains a set of factor vectors for each item; those vectors define which items are neighbors.
- A Yelp-specific model based on business category metadata

We started by implementing a baseline model based on average ratings overall, per-user, and per-item.

Baseline Model

Many collaborative filtering methods, both nearest-neighbor and factor methods, begin with a common baseline. This baseline takes the form

$$b_{ui} = \mu + \beta_u + \beta_i$$

where μ is the global average rating, β_u is the user bias, β_i is the restaurant bias-positive if it's a better-than-average businesses. There are multiple ways to calculate the bias terms. Different authors use either global optimization or averaging. We used averages with reg-

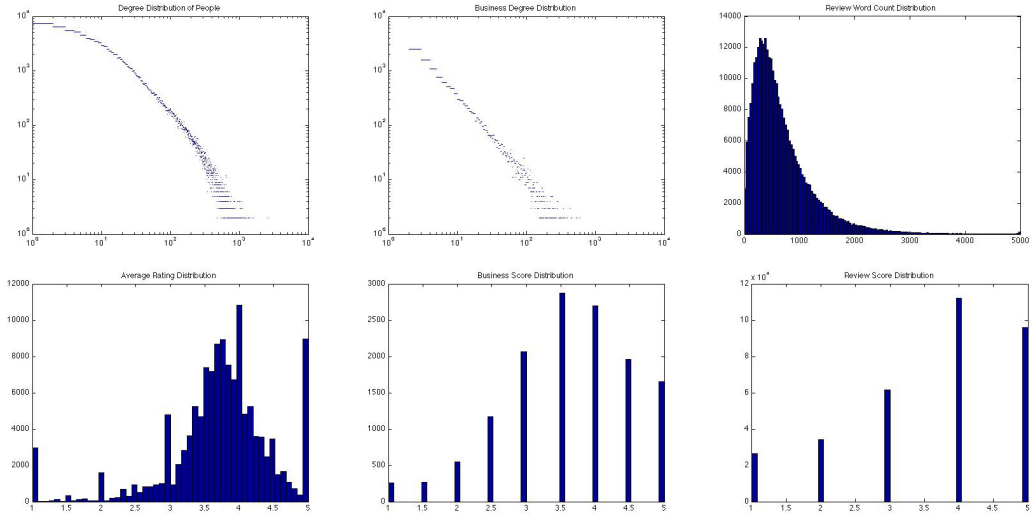


Figure 1: Top Row, Left to Right: Degree Distribution of User Review Counts, Degree Distribution of Item Review Counts, Distribution of Review Word Counts. Bottom Row, Left To Right: Distribution of User Average Stars, Distribution of Business Average Stars, Distribution of Review Stars

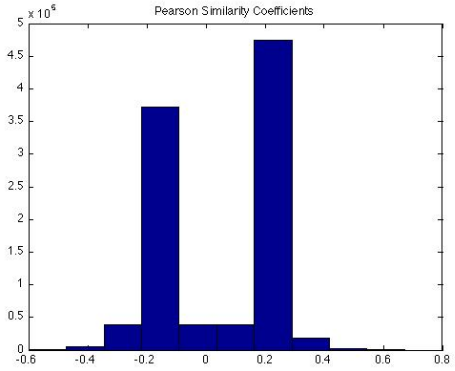


Figure 2: Pearson Correlation Coefficient distributions for a 3-fold Validation

Table 1: Network Statistics

Statistic	Yelp	Netflix	MovieLens	Yahoo! KDD
Users	130,873	480,189	72,000	1,000,990
Items	13,490	17,770	10,000	624,961
Ratings	330,071	100,480,507	10,000,000	262,810,175
Sparsity	99.98%	98.82%	98.61%	99.96%

ularization to mitigate overfitting:

Let K be the set of all ratings (u, i) and r_{ui} represent each rating.

$$\beta_u = \frac{\sum_{i|(u,i)\in K}(r_{ui} - \mu)}{\lambda_u + |\{u|(u,i) \in K\}|}$$

$$\beta_i = \frac{\sum_{u|(u,i)\in K}(r_{ui} - \mu)}{\lambda_i + |\{i|(u,i) \in K\}|}$$

Attempt 1: Item-item neighborhood model

We tried the basic item-centric nearest neighbor model:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j\in R(u)} (r_{uj} - b_{uj}) * s_{ij}$$

In this model, $\hat{r}_{ui}$ is the predicted rating of item i by user u. μ represents the global mean rating. Bias terms b_u and b_i represent user bias and item bias. The first summation is the sum of the ratings of the users multiplied by the similarity between the item rated and the item rating to predict. Most importantly, s_{ij} is a distance metric that defines which pairs of items i and j are "neighbors" and hence likely to be rated similarly by a given user.

We tested several different functions for s_{ij}, described below. We also tested an extension to the neighbor model proposed by Koren, taking implicit feedback into account:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j\in R(u)} (r_{uj} - b_{uj}) * s_{ij} + \sum_{j\in N(u)} c_{ij}$$

Implicit feedback means any input about user-item relationships other than the known rating values. In this case, we use the fact that a user rated a given item, which is separate from the rating they gave. This means that if a user rated an item i, regardless of the actual rating, they may be more or less likely to enjoy item j. The terms c_{ij} grows as i is becomes more predictive of j.

To calculate similarities, we used the Pearson Correlation Coefficient:

$$s_{ij} = \frac{\sum_u (r_{ui} - \mu_i) * (r_{uj} - \mu_j)}{\sqrt{\sum_u (r_{ui} - \mu_i)^2} \sqrt{\sum_u (r_{uj} - \mu_j)^2}}$$

Where r_{ui} represents the rating of item i by user u. μ_i represents the average rating for item i. The Pearson Correlation Coefficient, which ranges from -1 to 1, for our data represents the correlation of user opinions about the businesses. Negative correlations signify that those who rated item i high tended to rate j low, or vice versa. Positive correlations signify that those who rated i high tended to rate j high as well.

We also tried an alternative similarity measure based on Yelp metadata. We computed the Jaccard similarity between the set of categories for items i and j. This is the number of categories in common divided by the number of categories total between the two. For example, if business i has categories $\{Restaurant, Cafe, Asian\}$ and business j has categories $\{Cafe, French\}$, then $s_{ij} = \frac{1}{4}$.

Attempt 2: SVD

Singular Value Decomposition is a latent-factor method. To apply SVD to collaborative filtering, we treat the ratings as a matrix, with one row per user, one column per business, and entries representing reviews [**?**][**?**]. SVD recasts collaborative filtering as a matrix reconstruction problem. Specifically, given a ratings matrix A, we decompose

$$A = U\Sigma V^T$$

...where U and V are orthogonal and Σ is a diagonal matrix. Each row in U represents an user's feature vector and each column in V^T represents an item's feature vector.

The matrix of Yelp ratings is too large to factor directly. Instead, our SVD is derived from Simon Funk's iterative SVD algorithm. [**?**] The algorithm learns the features for users and items without directly calculating the SVD. That is, given a user feature vector u_i and an item feature vector v_i, we produce a rating

$$\hat{r_{ij}} = u_i^T v_j$$

We run updates on each of the k features of these user and item vectors by performing the

following calculations until convergence:

$$\epsilon_{ij} = r_{ij} - \hat{r_{ij}}$$

$$u_{ik} += l(\epsilon_{ij} * v_{jk} - \lambda u_{ik})$$

$$v_{ik} += l(\epsilon_{ij} * u_{jk} - \lambda v_{ik})$$

In these equations, k is the feature we are updating, l is the learning rate, λ serves as a regularization factor.

Attempt 3: Yelp-specific model

Finally, we tried an approach that models each user's preferences, taking advantage of some features unique to Yelp data. Each business is assigned a set of categories, such as $\{\text{Restaurant}, \text{Indian}\}$.

Then, we trained a simple model to learn each user's preference for each category of business. We start by computing the average rating μ, and the average offset for each user and item, β_u and β_i. This gives the baseline estimate

$$b_u i = \mu + \beta_u + \beta_i$$

Then, let the category of each item i be c_i. Let $R_c(u)$ be the subset of user u's ratings $R(u)$ where the business has category c. Each business can have multiple categories. We can now compute the any user's preference for category c:

$$\Theta_{uc} = \frac{1}{R(u) + \lambda_{cat}} \sum_{j \in R(u)} r_{uj} - b_{uj}$$

The parameter λ_{cat} is for regularization. For example, if a user gives an Indian restaurant a 5-star rating even though the baseline estimate was 3 stars, we don't want to assume that he will rate all Indian restaurants two stars above baseline. However, if that user rates multiple Indian restaurants two stars above baseline, then we can infer a preference.

Finally, the prediction rule is

$$\hat{r}_{ui} = \mu + \beta_u + \beta_i + \Theta_{uc_i}$$

Attempt 4: Hybrid latent-factor neighborhood model

We implemented and tested the hybrid approach described by [**?**]. The idea is to get the advantages that come with global optimization–higher accuracy and robustness to sparse graphs–while also maintaining the advantages of a neighborhood model.

We train three factor vectors for each item: x_i, y_i, and q_i. Instead of using a correlation coefficient or other closed-form metric to determine which nodes are neighbors, here two nodes' similarity is the dot product $q_i^T x_i$. This approach also includes implicit feedback: feedback that helps predict unknown ratings, other than the past ratings. Ideally, this feedback would come from a second channel: for example, data about which businesses Yelp customers actually visit, as opposed to just the ones they rate. Missing that, however, we still have a form of implicit feedback in the set of businesses that a user rated, independent of what the actual ratings were. (This term might encode, for example, that users who rate their visits to Whole Foods and Trader Joes are more likely to prefer Starbucks, regardless of how many stars they gave.) The latent factors y_i capture this feedback.

The prediction function is

$$\hat{r}_{ui} = \mu + b_u + b_i$$

$$+ \sum_{j \in R(u)} (r_{uj} - b_{uj}) * q_i^T x_j + \sum_{j \in N(u)} q_i^T y_j$$

...in other words, the baseline, plus the contribution of each neighbor j for item i weighted by $q_i^T x_i$, plus the contribution from implicit feedback. In this model, $\hat{r}_{ui}$ is the predicted rating of item i by user u. μ represents the mean rating. The first summation is the sum of the ratings of the users multiplied by the similarity between the item rated and the item rating to predict. The second summation is a term which takes into account implicit feedback.

We trained the model using gradient descent, as described by [**?**].

Results

Evaluation Metric

For evaluating the performance of our models, we used the RMSE error on edge rating predictions with 3-fold cross validation.

Baseline

We did a grid search letting

$$(\lambda_u, \lambda_i) = [0, 10, 20, 30] \times [0, 10, 20, 30]$$

We evaluated median RMSE for the baseline predictions $b_u i$ using 3-fold cross validation.

$$\text{RMSE} = \begin{pmatrix} 1.1213 & 1.1221 & 1.1228 & 1.1265 \\ 1.1088 & \mathbf{1.1087} & 1.1107 & 1.1131 \\ 1.1185 & 1.1187 & 1.1211 & 1.1201 \\ 1.1264 & 1.1270 & 1.1267 & 1.1296 \end{pmatrix}$$

The rows are $\lambda_i = [0, 10, 20, 30]$, the columns correspond to λ_j. The optimum value was found at $\lambda_i = \lambda_j = 10$. This is signifanctly better than naive averages ($\lambda_i = \lambda_j = 0$). We use 10 as the regularization constant going forward. **Baseline RMSE: 1.1087**

We experimented with four ways to improve on this baseline.

Attempt 1: Item-item neighborhood model

We implemented a nearest-neighbor filtering model. It estimates a missing rating r_{ui} (from user u to business i) by finding similar restaurants (neighbors) that u has already rated. It computes Pearson correlation coefficients for each pair of businesses that share reviewers. For each missing rating, it finds the top k most similar businesses based on that coefficient.

We experimented with $k = [1, 2, 5, 10, 20, 50]$ and found that $k = 5$ is optimal.

Nearest-neighbor RMSE: 1.1237

This is slightly worse than the baseline. We think that the poor performance of nearest-neighbor filtering is due to the overly restrictive definition of "neighbor" in a dataset where reviews are sparse. Movies in the Netflix data, for example, have thousands of reviewers, so you can calculate a meaningful correlation for a pair of movies. With restaurants—even in the same local area—this is not the case. More on that below.

We tried the same model with the same set $k = [1, 2, 5, 10, 20, 50]$ using an alternative measure of similarity between items. We used Jaccard similarity between the set of categories for each business, as described in Methods. This did not produce improvement for any value of k.

Attempt 2: SVD

Singular Value Decomposition is a factor method that performs well on some sparse datasets. It was basis for several top submissions to the Netflix Prize contest.

We suspected that the Yelp data might simply be too sparse for neighbor-based methods to work, so we implemented SVD.

We implemented a grid search over the number of factors to train, $n_{factors} = [1, 2, 3, 4, 5, 10, 15, 20, \ldots 100]$. We saw slow improvement in RMSE, leveling off above 30 factors.

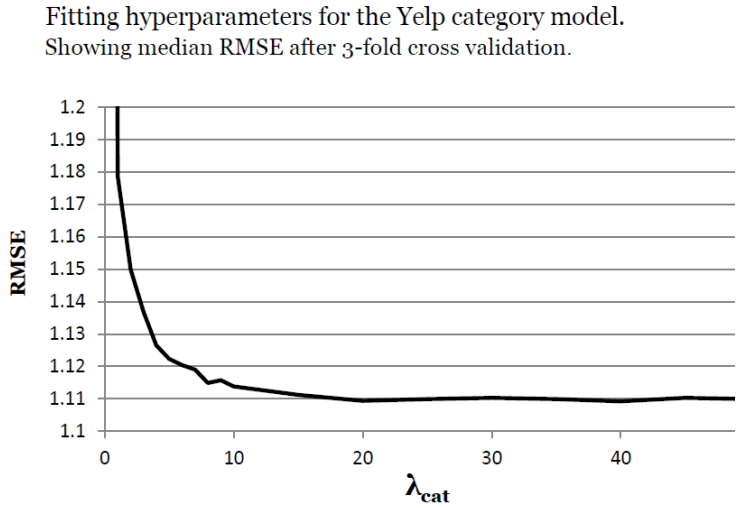
In this case, the data was so sparse that the predictions produced by iterative SVD differed minimally from the baseline predictions, and performed marginally better than the baseline. With 100 factors,

RMSE for singular-value decomposition: 1.1070

Attempt 3: a Yelp-specific model

We tested this for a range of values of the regularization parameter λ_{cat}:

This method did not produce significant improvement. The fact that performance improves monotonically when increasing the regularization parameter is a bad sign. At large values of the regularization parameter, the predictions produced approach those of the base-



line model. With $\lambda_{cat} = 50$, the RMSE is very close to the best baseline RMSE, but not better.

Attempt 4: Hybrid latent-factor neighborhood model

We ran a grid search to find the best set of hyperparameters γ and λ. The first, γ, controls the gradient descent learning rate, while the second, λ, is a regularization constant designed to prevent overfitting. We ran a distributed grid search using ten servers from the Corn cluster. The total runtime was approximately 16 hours.

The best results were achieved with the learning rate $\gamma = 0.004$ and regularization $\lambda = 0.02$. This is similar to the values $\gamma = 0.002$ and $\lambda = 0.04$ that Koren found to be optimal on the Netflix Prize dataset. The best RMSE was again marginally better than the baseline,

RMSE for hybrid model: 1.1082

Conclusion

We tested four broad approaches to producing a recommendation system for Yelp data, with variations and experiments for each one. The best system performs marginally better than our baseline estimates, after using a grid search to optimize the baseline.

We qualified some unique features of the Yelp graph that make ratings difficult to predict. First, the graph is very sparse–more so than other datasets used in recommendation systems. Second, while some users post many reviews, the majority of users post less than five reviews. The distribution of degree follows a power law for businesses, but falls off much more quickly for users. Third, Yelp data is uniquely local. The clusters around each university represented in the dataset are nearly disjoint–users mostly rate restaurants in their local area.

Despite these challenges, we believe we have a reasonable recommendation system for Yelp users. On an absolute scale, with an RMSE of 1.1082, most predictions are correct, rounded to the nearest star. The hybrid approach we implemented has an advantage over the baseline, even though the RMSE is only slightly better, in that the system can explain its recommendations in terms of the user's previous reviews.

References

[1] Yehuda Koren, *Factor in the Neighbors: Scalable and Accurate Collaborative Filtering*. KDD 2008.
`http://public.research.att.com/~volinsky/netflix/factorizedNeighborhood.pdf`

Fitting hyperparameters for Koren's hybrid method.
Showing median RMSE after 3-fold cross validation.

γ \ λ	0.002	0.005	0.01	0.02	0.04	0.08	0.16	0.32	0.64
0.0002	1.1119	1.1114	1.1101	1.1105	1.1113	1.1133	1.1127	1.1125	1.1161
0.0004	1.1111	1.1097	1.1113	1.1107	1.1122	1.1121	1.1129	1.1148	1.1192
0.0008	1.1103	1.1102	1.1106	1.1109	1.1119	1.1099	1.1118	1.1136	1.1215
0.002	1.109	1.1099	1.11	1.1092	1.1104	1.1116	1.1114	1.1154	1.123
0.004	1.1102	1.1107	1.1088	1.1082	1.111	1.11	1.112	1.1152	1.1237
0.008	1.1151	1.1124	1.1146	1.113	1.1139	1.1116	1.1139	1.1159	1.1222
0.02	1.1287	1.1294	1.128	1.1295	1.1286	1.1256	1.1254	1.1256	1.13
0.04	1.1567	1.1554	1.1536	1.1502	1.1494	1.1483	1.1449	1.1383	1.1375
0.08	1.1903	1.1913	1.1898	1.1858	1.1821	1.1794	1.1683	1.1592	1.1494
0.2	1.2577	1.2551	1.2545	1.2532	1.2402	1.2331	1.2087	1.1905	1.176

[2] Leskovec, Ajit Singh, and Jon Kleinberg *Patterns of Influence in a Recommendation Network* 2006. `http://snap.stanford.edu/class/cs224w-readings/leskovec06recommendation.pdf`

[3] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Viscek, *Uncovering the overlapping community structure of complex networks in nature and society.* `http://snap.stanford.edu/class/cs224w-readings/palla05overlapping.pdf`

[4] Jure Leskovec, Kevin J. Lang, and Michael W. Mahoney, *Empirical Comparison of Algorithms for network Community Detection.* `http://snap.stanford.edu/class/cs224w-readings/leskovec10communitydetection.pdf`

[5] Paul N. Bennett, Filip Radlinski, Ryen W. White, and Emine Yilmaz. *Inferring and Using Location Metadata to Personalize Web Search* `http://research.microsoft.com/en-us/um/people/ryenw/papers/BennettSIGIR2011.pdf`

[6] Arkadiusz Paterek. *Improving regularized singular value decomposition for collaborative filtering* `http://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/Regular-Paterek.pdf`

[7] Badrul Sarwar et al. *Item-Based Collaborative Filtering Recommendation Algorithms* `http://www.ra.ethz.ch/cdstore/www10/papers/pdf/p519.pdf`

[8] Anand Rajaraman and Jeff Ullman. {Mining of Massive Datasets, ch. 9: Recommendation Systems} `http://infolab.stanford.edu/~ullman/mmds/ch9.pdf`

[9] Piccart, B., Blockeel, H., & Struyf, J. (2010, April). Alleviating the sparsity problem in collaborative filtering by using an adapted distance and a graph-based method. In Proceedings of the Tenth SIAM International Conference on Data Mining (pp. 189-199).

[10] Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009, 4.