Dan Gillick
Josh Blumenstock
CS294-10 Final Write-up
12/11/2006

Netflix:
Combining meta-data features with neighbor-based models

1. Introduction

There are around 20,000 movies available for rental through Netflix and over 2,500,000 songs available on Yahoo! Music. While some users know exactly what movies they want to watch and which music they want to listen to, the majority of users are likely to be overwhelmed by the range of possibilities. One way to narrow down the selection problem is to provide suggestions to users based on their preferences – that is – based on ratings they've given to some subset of the movies or songs.

To collect this preference data, both Netflix and Yahoo!, along with Amazon and other companies with a wide variety of products, typically ask users to rate items on a 5-star scale. A user can then be represented by a list of (product, rating) pairs, perhaps along with some metadata such as age, gender, zip-code, and so on. By looking for patterns in a user's data and for similarities between users and between products, we hope to predict how that user would rate new products, using these predicted ratings in a suggestions engine.

This task is broadly referred to as Collaborative Filtering. With the recent expansion of internet-accessible catalogues of movies, music, etc., this has suddenly become quite valuable technology. A few months ago, Netflix announced a competition to improve on their collaborative filtering algorithm. In this paper, we motivate some general classes of techniques to address the Netflix problem and present some results and analysis. As this is a relatively unexplored area, and the massive data-set presented a host of initial challenges, the work presented here is fairly preliminary. We are encouraged, however, by early results, and intend to continue this research over the next few months.

The organization of the paper is as follows: section 2 describes the training data, test data, and evaluation criteria; section 3 introduces a few baseline methods and scores; section 4 outlines two approaches we have begun investigating along with early prediction results and analysis; finally, section 5 summarizes some conclusions and suggests future work.

2. The Netflix Data

Netflix provides ~20,000 movie IDs, ~500,000 user IDs, and ~100,000,000 examples of the form (movie ID, user ID, rating). The data is quite sparse, with roughly 200

ratings/user, and 5,000 ratings/movie (Figure 1). We can imagine a matrix M where each user is a row and each movie is a column. M is thus 20,000 by 500,000 and we are provided with 100,000,000 entries, or about 1% of all possible entries (if every user rated every movie).

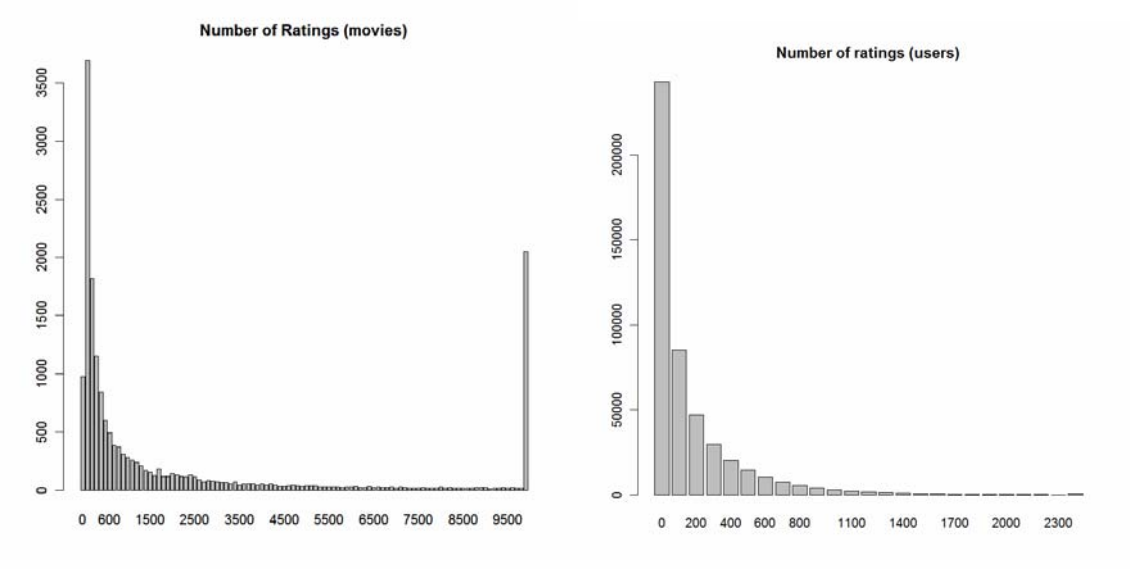

Figure 1: Number of ratings per movie and per user

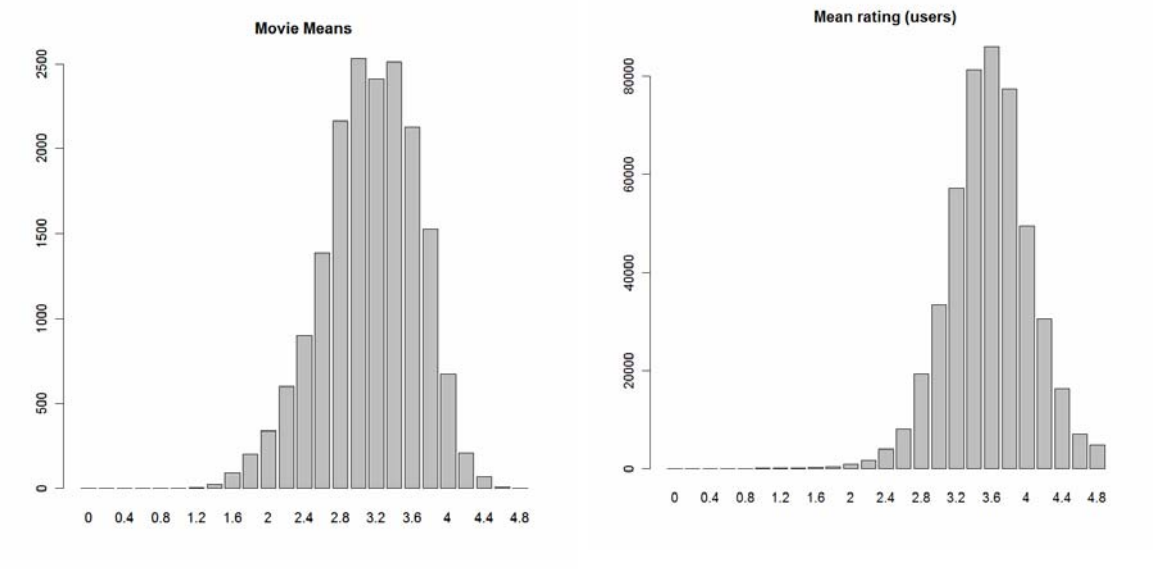

Figure 2: Average rating by movie and by user.

3. Baselines

Before delving into more sophisticated techniques, we ran a simple set of baseline experiments (Table 1) on the probe set. This included, for every (movie, user) pair:

a) Predicting "3 stars", irrespective of the movie and user
b) Predicting the user's average rating (mode), irrespective of the movie
c) Predicting the user's average rating (mean), irrespective of the movie
d) Predicting the movie's average rating (mean), irrespective of the user
e) Predicting the average of the user and move means.

Baseline Method	**RMSE**
Guess "3 stars"	1.313
Guess global mean ~ 3.60	1.130
Guess movie mean	1.052
Guess user mean	1.043
Guess (movie mean + user mean) / 2	1.004
Netflix "cinematch" system	.951

Table 1: RMSE on probe set for different baseline methods

A more interesting baseline combines these aggregate-style statistics. Taking a probabilistic view, we are interested in calculating P(rating|user, movie), which we will abbreviate as P(r|u,m). Applying Bayes Rule, we have:

$$P(r|u,m) = P(r,u,m) / P(u,m) = P(u,m|r)P(r) / P(u,m)$$

Note that some of these quantities do not have particularly meaningful interpretations. It is easiest to think about these probabilities as maximum-likelihood estimates over data of the from (u,m,r), of which we have many examples. P(r) for example can be estimated by summing over all users and all movies and normalizing so that we have the ratio of the number of ratings of a particular value over the total number of ratings.

We cannot directly use this equation since the joint probability P(u,m) is either 0 or 1, depending on whether user u has rated movie m. Making some independence assumptions allows us to estimate this quantity easily from our data. We assume that (1) u and m are independent and that (2) u and m are conditionally independent given r. Assumption (1) has a meaningful interpretation, though it is most likely not correct: it says that the probability that a user u has rated a movie m is equal to the product P(u)P(m). P(u) is estimated as the number of ratings made by u as a fraction of all ratings and P(m) is estimated as the number of ratings on m as a fraction of all ratings. This seems generally correct: the probability that u has rated m is high if u rates many movies and if m has many ratings. Assumption (2) is a little less intuitive, but has basically the same interpretation. We can then reduce our equation as follows:

$$\begin{aligned} P(r|u,m) &= P(u|r)P(m|r)P(r) / P(u)P(m) && \text{[independence assumptions]} \\ &= P(r|u)P(u)\ P(r|m)P(m)\ P(r) / P(u)P(m)P(r)P(r) && \text{[Bayes Rule again]} \\ &= P(r|u)P(r|m) / P(r) && \text{[simplifying]} \end{aligned}$$

We can easily estimate P(r|u), P(r|m), and P(r) from our data using the principle of maximum likelihood. Finally, we need to make a prediction given this distribution. While we might select the value of r which has the largest value, we are not practically constrained to making integer predictions, and in fact, taking the expectation of this distribution provides far better predictions.

Another nice feature of the distribution over ratings is that we can estimate confidence based on the sharpness of the distribution. Intuitively, the sharper the distribution, the more confident we are of our prediction. The table below shows the results over various confidence levels.

confidence	Rmse	Frac. of total	Confidence	Rmse	frac. of total
>0.99	0.2985	0.0000078	>0.99	0.2985	0.0000078
>0.90	0.5537	0.0058151	>0.90	0.554	0.0058073
>0.80	0.7027	0.0233911	>0.80	0.7455	0.017576
>0.70	0.7768	0.0594605	>0.70	0.8212	0.0360694
>0.60	0.818	0.1373335	>0.60	0.8481	0.077873
>0.50	0.8432	0.3251507	>0.50	0.8611	0.1878173
>0.40	0.8892	0.6883717	>0.40	0.9284	0.3632213
>0.30	0.9527	0.9597528	>0.30	1.0975	0.2713813
>0.20	0.9693	1	>0.20	1.3038	0.0402465

Table 2: For each confidence level, cumulative rmse (left) and per-bin rmse (right).

Note that the RMSE for the entire probe set is 0.969, a rather impressive result given the simplicity of the model (compared to the Netflix baseline of 0.951 on the same set). Also, our confidence measure is reasonably well-calibrated, which suggests that we could combine multiple systems by weighting the contribution of each system by its confidence.

Thinking about this model more generally, we can rewrite our equation for P(r|m,u) in the log domain. Letting Q stand for the log probability, we obtain a linear model:

$$Q(r \mid m,u) = w_0 + \sum_{i=1}^{5} w_i Q(r = i \mid m) + \sum_{i=1}^{5} w_{i+5} Q(r = i \mid u)$$

Note that P(r), above, which served as a normalizing constant is excluded from the model here as these values would be the same for each data point. We are then left with a regression problem in which we learn a set of 11 weights, W, for each value of r. At prediction time, we compute the expectation as before:

$$\mathrm{E}(r \mid m,u) = \sum_r r \cdot P(r = i \mid m,u) = \sum_r r \cdot \exp(W_i \cdot F_{m,u})$$

This formulation gives us a general framework for predicting ratings. Given some set of features for a (movie, user) paring, we learn a set of weights to predict the probability of each rating class. Thus far, we have been talking about a global set of weights, learned over all examples. Besides the fact that 100 million data points is hard to load into memory (making the least-squares computation difficult), we'd really like to model each

user or perhaps each movie separately. To do this effectively, we need features that capture more about movies and users than simply the distribution over ratings. The remainder of this paper can be viewed as an exploration of novel features to use in these models.

4. New Methods

The task before us is to predict the rating given to movie m_j by user u_i for an unseen pair (u_i, m_j). In this section we discuss the three main approaches we took to the problem.

The first technique relies on $P(r \mid u)$, using each user's aggregate rating behavior to cluster users (4.1). While effective at identifying users with unique rating behavior (e.g. only rating movies 1-star), it is not very robust, as it does not account for concepts like movie content, user tastes, etc. The second technique presented (4.2) is a collaborative filtering method using nearest neighbors, where the rating $r(u_i, m_j)$ is based on ratings u_i gave to movies similar to m_j (4.2.1), and ratings given to m_j by users similar to u_i (future work). Lastly, we discuss a content-based approach, where the features are terms commonly associated with movies, such as "Al Pacino" or "Thriller" (4.2.3). Using these terms as features, we are able to assess movie similarity, do simple clustering, and make predictions based on the ratings u_i gave to the movies similar to m_j.

4.1. Distribution-based clustering

In the sample of 500,000 users, there are people with vastly different rating behavior. Some users only rate movies 1-star, some rate only 5-stars, and some users distribute their ratings with relatively uniform frequency (Figure 3). Similarly, the 20,000 movies encompass a wide range of distributions – some with consistently high ratings (e.g. Battlestar Galactica), some with evenly distributed ratings (e.g. Miss Congeniality), etc. For predictive purposes, the distribution profile of a user or movie can be extremely useful – if every movie rated by u_i has been rated 4-stars, it is probable that the next movie will also be rated 4-stars.

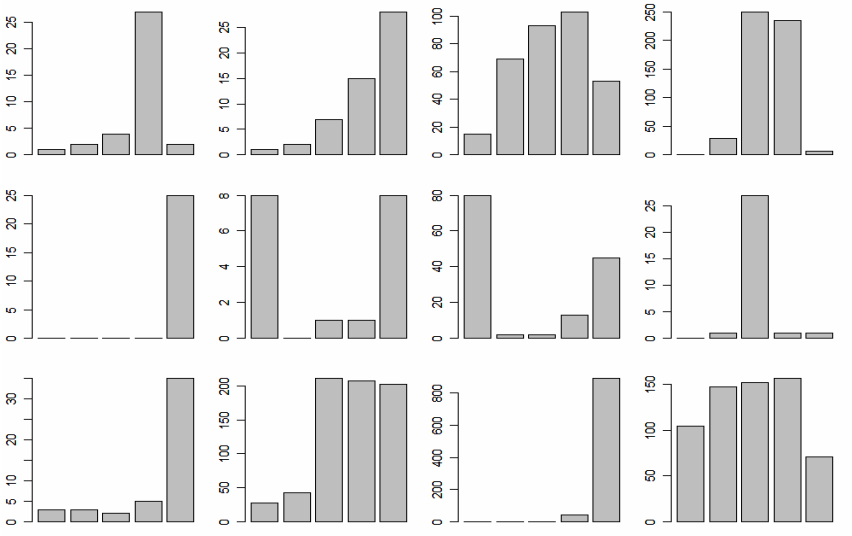

Figure 3: Sample rating distributions from representative users

A variety of statistics exist for measuring the dispersion and deviation of a distribution [Joanes]; however, the standard deviation is a simple metric that allows us to identify users and movies with consistent ratings. By limiting our predictions on the probe set to those users and movies with relatively low standard deviation, we were able to drastically lower the predictive error of the relevant baselines. For instance, 33% of users have a standard deviation less than .90. Predicting the user's average for this subset of users produced an error of .8 – a 30% reduction over the baseline error (baseline c). Figure 4 illustrates the relationship between standard deviation, predictive accuracy, and probe set coverage.

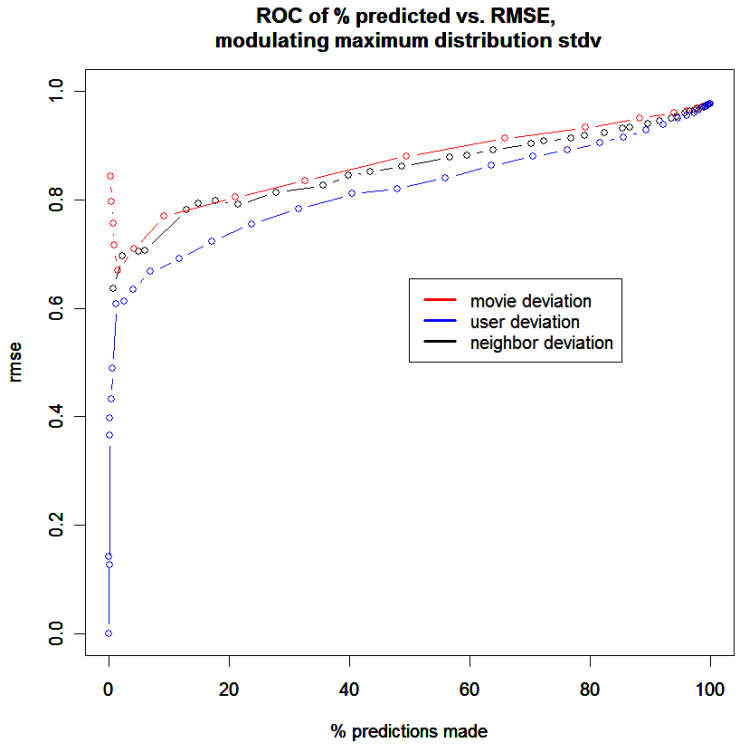

Figure 4: For users with very low standard deviation, predicting the user's mean is extremely accurate. As the standard deviation grows, the user's mean becomes less accurate. The same is generally true for the movie – the movie mean is a less accurate when the movie has a high standard deviation. The "neighbor deviation" is the standard deviation of the ratings given movie Y by the k nearest neighbors to user X.

4.1.1. Computing nearest neighbors

Using standard deviation as a measure of dispersion is a useful technique for finding users and movies with extremely consistent rating behavior. However, it is also conceivable that the shape of the *distribution itself* would be a useful means for comparing two users, or two movies. To this end, we attempted to run k-nearest neighbor

clustering on both (a) users and (b) movies, where distance between, for instance, u_1 and u_2 was measured as the distance between the two distributions. For example, using the symmetrised Kullback-Leibler divergence:[1]

$$D(u_i, u_j) = KL(u_i,u_j) + KL(u_j,u_i), \text{ where}$$

$$u_i = \{r(u_i, m_1), \ldots, r(u_i, m_N)\}, \text{ and}$$

$$KL(u_i, u_k) = \sum_{r=1}^{5} u_i(r) \log \frac{u_i(r)}{u_j(r)}$$

4.1.2. Predicting $r(u_i, m_j)$ from nearest neighbors

Such a distance function gives a distance matrix D of pairwise distances between all users, from which the k-nearest neighbors to each user can be extracted. The hope is that the rating behavior of u_i's neighbors will better predict $r(u_i, m_j)$ than the aggregate ratings for all users. Figure 5 shows the distinction in ratings for a given movie between all users (grey histogram) and the 10-nearest neighbors (yellow histogram).

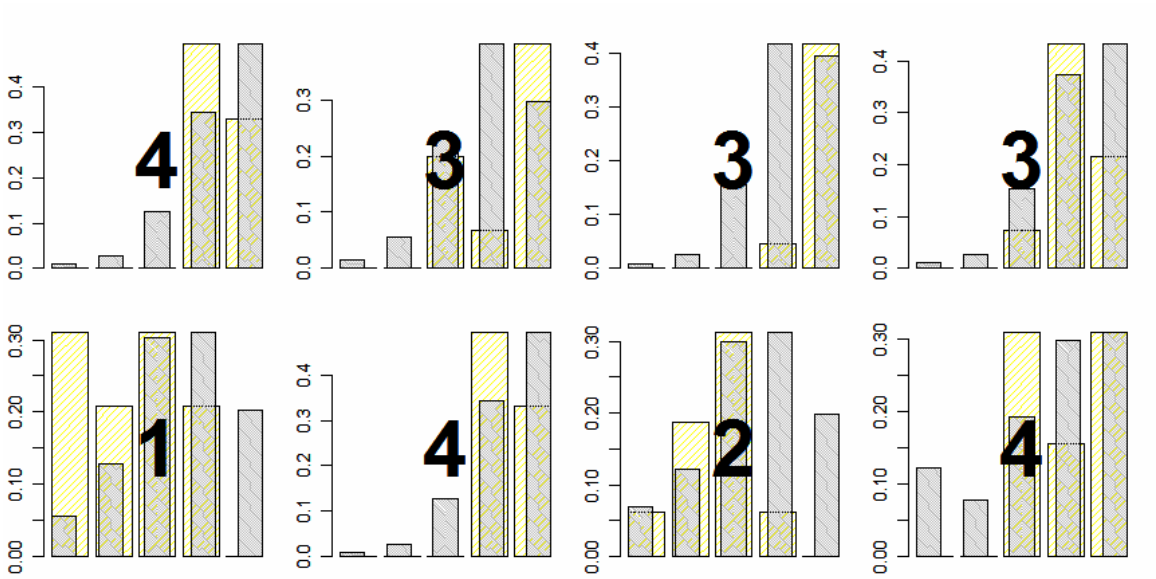

Figure 5: different distributions of ratings for 16 different movies, for all users (grey histogram) and the k-nearest neighbors to u_i (yellow histogram). The true score is displayed over the histograms.

Thus, a simple method for computing $r(u_i, m_j)$ for an unseen movie m_j is given by:

$$r(u_i, m_j) = \frac{c}{M} \sum_{k=1}^{M} R(u_k, m_j) \bullet D(u_i, u_k)^{1/\alpha}$$

[1] We use the symmetrised K-L divergence instead of the simpler K-L distance in order to reduce computational cost.

where $R(u_k, m_j)$ is the known rating given m_j by u_k. Here, the rating given m_j by u_i is predicted using the ratings given m by neighbors to u_i, weighted according to the neighbor's distance from u_i by the weighting parameter α. The constant c is a normalization parameter.

Unfortunately, computation of the entire 500,000 x 500,000 user matrix D was not feasible with the computing resources we could muster, but preliminary tests on subsets of the matrix are encouraging. Blindly choosing a subset of 10,000 users, and following the above technique with α=1, produces an overall RMSE of .972, a small improvement over the baseline predictor. However, clever selection of the users (e.g. by randomly choosing users who have seen many movies), can produce an RMSE as low as .896, though such a technique does not generalize to the entire data set.

4.2. Item-based collaborative filtering

4.2.1. Distance Metric

The distribution-based technique described above uses individual user distributions to measure distance between users, then makes predictions $r(u_i, m_j)$ based on the ratings given m_j by users near u_i. The intuition here is that if many users rate two movies the same, the movies should be considered similar. Conversely, if many users rate two movies differently, the movies should be considered different.

Thus, a simple method for assessing similarity between two movies is:

$$d(m_i, m_j) = \sum_{k}^{N} \left| R(u_k, m_i) - R(u_k, m_i) \right| \cdot (c_r \left| R(u_k, m_i) - R(u_k, m_i) \right|)$$

where $R(u_k, m_i)$ is the known rating given m_i by u_k, and c_r weights the term by the difference in ratings given m_i and m_j by u_k. Intuitively, c_0 should be zero and c_4 should be large.

Ideally, we would like to compute the 17,000 x 17,000 distance matrix using the full data set with M=500,000. Unfortunately, this computation requires significant resources[2], so we instead tested the method on a subset of the data with M=500.[3] As can be seen in Table 3, this method of assessing movie similarity accurately reflects our intuitive notion of similarity.

[2] We initially wrote most of our code in Python, which was a regrettable decision. We're currently porting the code to C, which should make the full computation possible.

[3] Choice of the subset of M users significantly affects the resulting clustering; we didn't fully explore this parameter as we intended to later use the entire dataset (M=500,000).

LOTR: Fellowship of the Ring	Sex and the City: Season 4	Finding Nemo
1. LOTR: The Two Towers	1. Sex and the City: Season 3	1. Monsters
2. LOTR: The Return of the King	2. Sex and the City: Season 5	2. Shrek (Full-screen)
3. LOTR: The Fellowship of the Ring: Extended Edition	3. Sex and the City: Season 1	3. Shrek 2
	4. Sex and the City: Season 2	4. LOTR: The Two Towers
4. LOTR: The Two Towers: Extended Edition	5. Sex and the City: Season 6: Part 1	5. Pirates of the Caribbean: The Curse of the Black Pearl
5. Raiders of the Lost Ark	6. Sex and the City: Season 6: Part 2	6. The Incredibles
6. LOTR: The Return of the King: Extended Edition	7. The Sixth Sense	7. The Sixth Sense
7. Pirates of the Caribbean: The Curse of the Black Pearl	8. Finding Nemo (Widescreen)	8. The Shawshank Redemption: Special Edition
8. The Matrix	9. Forrest Gump	9. LOTR: The Fellowship of the Ring
9. The Shawshank Redemption: Special Edition	10. The Shawshank Redemption: Special Edition	10. Forrest Gump
10. Braveheart		

Table 3: Sample nearest neighbors using user-based collaborative filtering

4.2.2. Predicting $r(u_i, m_j)$ from nearest neighbors

Using a technique very similar to the one described in (4.2.1), we can compute $r(u_i, m_j)$ for an unseen movie m_j using:

$$r(u_i, m_j) = \frac{c}{N} \sum_{k=1}^{N} R(u_k, m_j) \bullet d(m_j, m_k)^{1/\alpha}$$

The results were not as good as we had hoped, producing an RMSE of .992. However, the overall error depends, to some extent, on M (RMSE=1.02 for M=50), and as we scale M up by 2-3 orders of magnitude we expect the RMSE to continue to drop.

User-based collaborative filtering is perhaps a more interesting approach, though the computational constraints are larger given the potential 500,000 x 500,000 distance matrix.

User-based filtering is also less intelligible, as it is impossible to produce a meaningful table such as Table 3. As we refine our computational methods and develop a deeper understanding of what we're doing, we will "flip the axes" and apply these item-based techniques to users.

4.3. Metadata

Thus far, we have been comparing movies based on the ratings provided by users. That is, two movies are similar if many users rate them similarly. While this information is informative, it is neither deep nor interpretable. Moreover, we would expect a person's taste in movies to be multifaceted – that is – I like stupid comedies but also documentaries; Quentin Tarentino but also Woody Allen; the Marx Brothers but also Dustin Hoffman. In light of this observation, we collected metadata for each movie, in the hopes of more precisely featurizing movies based on some set of tangible dimensions.

We also hope to be able to model user preferences based on the ratings given to movies of each type. With these somewhat lofty ambitions, we have:

1. collected a corpus of web-based movie text
2. compiled a list of relevant movie terms
3. extracted features for each movie

While these are not directly machine-learning tasks, we discuss them briefly in part because they are interesting and in part because we spent a lot of time working on this part of the project.

4.3.1. Web Corpus

There is a huge amount of information available on the internet, and in particular, many pages about each movie. Given the list movie titles and release years provided by Netflix, we used the Yahoo! Search API to get the top 10 urls for each movie and downloaded the text from these pages. This raw data comes from many sources, including, but not limited to the Internet Movie Database (IMDB), Wikipedia, personal web pages, reviews, blogs, discussion boards, and Netflix itself. The data is roughly sentencized, punctuation is removed, and cleaned to remove code, html tags, etc. This final step is done by throwing out lines of text with out-of-vocabulary (OOV) rates above a certain threshold (where the vocabulary is an English dictionary). While this ends up pruning a few proper names that appear out of context, adjusting the OOV threshold mostly solves this problem.

In the end, we are left with somewhat clean text for each movie, and around 150 million words over all 17,770 movies, or around 8,500 words per movie. As this is automatically extracted web data, there is a good deal of noise – advertisements, broken urls, bad search results, and so on, but the overall effect appears reasonable. To demonstrate the potential of this data, we begin by computing term frequency-inverse document frequency (TFIDF) over the words in each movie's web-data. TFIDF is roughly the ratio between the number of appearances of word w in a document and the number of documents over the whole corpus in which word w appears. Thus, common words like 'the' and 'of' tend to have small TFIDF values (they have large TF and large IDF), while relevant words tend to have large TFIDF values (they large TF and small IDF). The table shows the top ranked TFIDF words for a few well-known movies.

LOTR: Fellowship of the Ring		Sex and the City: Season 4		Finding Nemo	
fellowship	461	miranda	263	nemo	1127
tolkien	384	carrie	243	pixar	244
rings	345	samantha	159	stanton	196
lord	151	half-hour	100	degeneres	149
ring	115	charlotte	90	finding	96
zealand	99	glaad	84	fish	93
towers	82	excellence	82	sharks	50
storyboards	43	mini-series	81	brooks	49
afi	42	sex	80	exploring	48
merry	37	parker	43	underwater	45
triology	35	housewives	43	thx	37
Jackson	34	hbo	42	ellen	32

Table 4: Top TFIDF words and their values. Note that adding bigrams would capture full names like Ellen Degeneres.

Apart from being amusing, these terms do appear to get at the important components of the movies: we see featured actors, characters, locations, descriptions, directors, etc. Convinced that our web-data contains some useful information, we proceed to step 2.

4.3.2. Feature Selection

We would like to select a fixed set of terms so we can create feature vectors that describe movies. But how to select our terms? Ideally, we would somehow choose the most relevant. We might approximate this set by aggregating the top TFIDF terms we extracted for each movie. This list gets quite large, though, so as a first step, we chose a set of categories by hand and used Wikipedia (and our own brainstorming) to fill them in. Our categories, and some example terms from each are shown below.

Category	**Examples**
Actresses	aaliyah, alicia silverstone, anna pacquin
Actors	adrien brody, alan alda, albert brooks
Directors	woody allen, wes anderson, warren beatty
Genres	drama, comedy, documentary
MPAA ratings	rated g, rated pg, rated r
MPAA descriptions	kids, violence, nudity
DVD words	special features, bonus, exclusive
Awards	academy award, dest actor, costume design
Festivals	cannes film festival, venice film festival, sundance film festival
Countries	albania, austria, Bulgaria
Keywords	racial, intellectual, witty

Table 5: Metadata categories and example terms in each

4.3.3. Extracting Features

In total, we ended up with 2,500 terms across 11 categories which we pruned to 1,900 by removing any which did not appear in at least 10 movies. Of course, the feature vector of any given movie is still fairly sparse as typically only a few actors (from our list of some 500), for example, make an appearance in the text of a movie. Specifically, the value of a feature for a movie is the TFIDF for that term. Features are then normalized per-movie so that features for each movie sum to 1 to allow comparison between movies.

4.3.4. PCA

Now that we have a new metadata representation of the movies, we need to decide what to do with it. As a first step, we construct the complete featurized movie matrix where each row is a movie and each column is a feature. For the sake of comparison, we construct a parallel matrix where each column represents a user and the values are the ratings given to each movie. We use only the 1,900 users who rated the most movies to minimize sparsity and to keep the comparison meaningful. We refer to the metadata

matrix as M_w (for web) and the user-ratings matrix as M_r (for ratings). Both M_w and M_r have dimensionality 17,770 by 1,900. M_w is 98% sparse – that is – 2% of the values are non-zero, while M_r is 87% sparse (though recall that the overall sparsity of the ratings matrix, including all users, is 99%). The table shows the relative sizes of the first few eigenvalues of the covariance matrices for M_w and M_r.

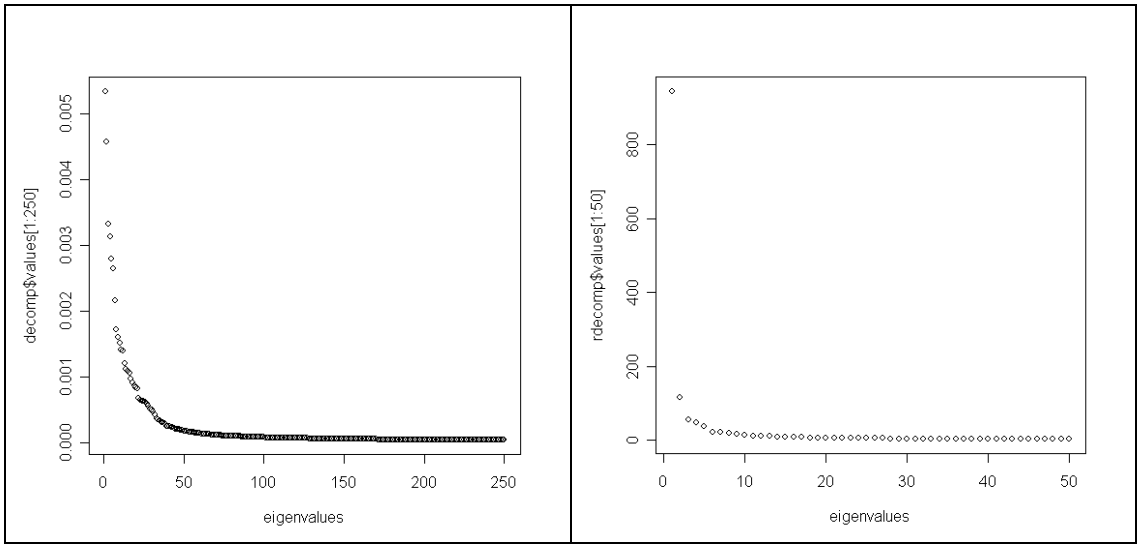

Figure 6: First eigenvalues for cov(M_w) (left) and cov(M_r) (right)

What is most interesting about the eigenvalue decomposition is best shown with a table, shown below. The first principle component of M_r contains far more of the variance than any other component, while the variance in M_w is more evenly distributed among the principle components. Nonetheless, the first 50 principle components of M_w contain a larger percentage of the total variance than in M_r, suggesting that the metadata does in fact provide a richer parameterization of movies than the user ratings. Moreover, we are encouraged that these two sources of data are somewhat different, and may combine to give superior features to either method alone.

Number of eigenvalues	**% of total variance, M_w**	**% of total variance, M_r**
1	0.06	0.36
2	0.12	0.41
5	0.23	0.47
10	0.34	0.50
50	0.61	0.59
100	0.68	0.63
500	0.86	0.80
1000	0.95	0.91

Table 6: Percent of total variance in the first k eigenvectors for the two matrices

The tables below show the nearest neighbors of a few movies given features extracted from M_w and M_r, respectively. Note that while there is some overlap, the features group movies together for different reasons.

LOTR: Fellowship of the Ring	Sex and the City: Season 4	Finding Nemo
1. The Key: Special Edition	1. Sex and the City: Season 1	1. Finding Nemo (Full-screen)
2. LOTR: The Fellowship of the Ring: Extended Edition	2. Sex and the City: Season 2	2. SpongeBob SquarePants: Tide and Seek
3. LOTR: The Two Towers: Bonus Material	3. Sex and the City: Season 3	3. Toy Story
4. Lord of the Rings: The Two Towers	4. Lie Down with Dogs	4. Monsters
5. Doctor Who: Lost in Time: The William Hartnell Years	5. How To Be A Player	5. A Bug's Life: Bonus Material
6. Mission to Mars	6. Jeffrey	6. Robin Hood (Disney)
7. Doctor Who: Lost in Time: The Patrick Troughton Years	7. Sex and the City: Season 6: Part 2	7. Brother Bear (Theatrical Widescreen Version)
8. LOTR: The Fellowship of the Ring: Bonus Material	8. Eddie Griffin: Voodoo Child	8. Balto 2: Wolf Quest
9. LOTR: The Return of the King: Extended Edition	9. Sex and the City: Season 5	9. SpongeBob SquarePants: SpongeBob Goes Prehistoric
10. Star Trek: The Motion Picture	10. Sex Is Comedy	10. The Rescuers Down Under
11. Legend: Director's Cut	11. Tomcats	11. Thumbelina
12. Crossworlds	12. Loving Jezebel	12. Shrek (Widescreen)
13. LOTR: The Two Towers: Extended Edition	13. Why Do They Call It Love When They Mean Sex?	13. Robbie the Reindeer in Hooves of Fire and the Legend of the Lost Tribe
14. The Incredible Hulk: The TV Series Ultimate Collection	14. Teaserama	14. Father of the Pride: The Complete Series

Table 7: Sample nearest neighbors using Euclidean distance on M_w

LOTR: Fellowship of the Ring	Sex and the City: Season 4	Finding Nemo
1. Lord of the Rings: The Two Towers	1. Sex and the City: Season 3	1. Shrek (Full-screen)
2. Lord of the Rings: The Return of the King	2. Sex and the City: Season 5	2. Monsters
3. The Matrix	3. Sex and the City: Season 1	3. Shrek 2
4. Indiana Jones and the Last Crusade	4. Sex and the City: Season 2	4. Pirates of the Caribbean: The Curse of the Black Pearl
5. Raiders of the Lost Ark	5. Sex and the City: Season 6: Part 1	5. Indiana Jones and the Last Crusade
6. Gladiator	6. Sex and the City: Season 6: Part 2	6. Indiana Jones and the Temple of Doom
7. Pirates of the Caribbean: The Curse of the Black Pearl	7. The Sixth Sense	7. Spider-Man
8. Braveheart	8. Finding Nemo (Widescreen)	8. Forrest Gump
9. The Sixth Sense	9. Forrest Gump	9. Jurassic Park
10. Spider-Man	10. The Shawshank Redemption: Special Edition	10. Harry Potter and the Sorcerer's Stone
11. The Terminator		11. Men in Black
12. The Shawshank Redemption: Special Edition		12. Big
13. Star Wars: Episode V: The Empire Strikes Back		13. The Sixth Sense
14. Jurassic Park		14. A Bug's Life

Table 8: Sample nearest neighbors using Euclidean distance on M_r

It is instructive to consider why we are interested in PCA. Afterall, we could simply measure the distance between each pair of movies using all 1,900 features. Or in the case of M_r, the distance between each pair of movies could be computed over all the ratings by users who have rated both. The first reason is computational: the full distance calculation

is on the order of (20,000 x 20,000 x 2,000) = 800 billion. Taking only first 50 principle components reduces the calculation by a factor of 40. A second, and more important reason has to do with noise and generalization. In most applications, we would test reconstruction error on some held-out set of movies to see how our dimensionality reduction generalizes to new data. This problem is somewhat unique, though, in that the set of movies (and users, for that matter) is fixed, at least in the contest formulation. That is, we will never be asked to rate a new movie we have never seen before. Note that our metadata parameterization would be particularly useful in this case (which Netflix needs to address when they add new movies to their collection that nobody has yet rated). Even in our case, though, we can hope that PCA captures most of the important or robust variance, discarding noise and misleading correlations in the training data.

5. A Few In-Progress Results

Now that we've extracted these web features, we need to decide how to apply them. As a first experiment, we tried a linear regression model to learn the importance of each feature to each user. Each user has a number of sample points of the form (x,y) where x is the feature vector for a movie and y is the rating he gave that movie. Based on this data, we learn a set of weights to minimize prediction error (we end up with 1900 weights per user). The best RMSE achieved in this fashion was 1.35.

Realistically, this is probably too much to expect from a regression model, especially when we only have a few example points (many users only have rated a few movies). Thus, our next experiment involved reducing the dimensionality of the feature space from 1900 to 50, as described above. The best RMSE was reduced to around 1.25. For the sake of comparison, we also tried this technique using the 50-dimensional features extracted from M_r as well, with very similar results, RMSE = 1.27. It is also worth noting that this framework for predicting actual ratings (rather than probabilities of ratings) as described in section 3, is problematic because the regression is sensitive to the magnitudes of the features. This is evidenced by the fact that the training error is actually quite low – the average per-user error at training time is around 0.65.

Similarly, we tried SVM-regression on the original feature set, hoping that the SVM would learn to distinguish valuable features while ignoring unimportant features. To start, we used a simple linear kernel, giving RMSE = 1.09. While this result is certainly an improvement, it is not particularly impressive compared with other methods. Linear Discriminant Analysis produced comparable results. With LDA, we tested to see if some set of features did a good job at separating a user's 5 ratings from his 1 ratings, for example. Preliminary tests on a few individual users suggest that this might work, but we have not had a chance to extend this idea into a classification system.

6. Conclusions and Future Work

Intuition tells us that some small set of these meta-data features ought to be important for a particular user. Since we want to build user models, we really want to know which features to select for that user. For instance, some users might best be described by their aggregate distribution, whereas others might be better modeled by the keywords present in the movies they rate. Our current "best method" is to combine the results of the SVM regression with our probabilistic baseline, giving an RMSE of 0.958 a (1% improvement). This gives us hope of combining systems effectively, once we find a way to use the metadata more productively.

7. References

Joanes, D. N. & Gill, C. A. "Comparing measures of sample skewness and kurtosis." Journal of the Royal Statistical Society (Series D): The Statistician 47 (1), 183–189 (1998)

Chumki Basu and Haym Hirsh and William W. Cohen. "Recommendation as Classification: Using Social and Content-Based Information in Recommendation." AAAI, 714-720 (1998)

Hofmann, Thomas. "Learning What People (Don't) Want." Lecture Notes In Computer Science; Vol. 2167 Proceedings of the 12th European Conference on Machine Learning 214 - 225 (2001)