Collaborative Filtering, Predicting ratings, with a Flexible Mixture model

CMPS 242 March 15, 2009

Lanbo Zhang lanbo@soe.ucsc.edu
Michael Cutter mcutter@ucsc.edu
Michael Percy mpercy@ucsc.edu

ABSTRACT

This paper discusses the effectiveness of the Flexible Mixture Model (FMM) in predicting user ratings on unseen data. The authors implemented a FMM from scratch in Python and ran the algorithm on a Yahoo! Music user rating data set. The authors also created baseline predictors with which to compare the results of the FMM algorithm. In short, the FMM algorithm beat two baseline results substantially.

INTRODUCTION

Recommendation systems are becoming the standard on many websites, from Amazon to Yahoo, offering mass personalization for customer’s preferences based on explicit and implicit user inputs, mimicking a word-of-mouth recommendation. In order for customers to make informed decisions it requires prior knowledge that they may or may not have. This is where recommendation systems are able to decrease the effort and cost to the customer by providing the opinions of past customers who have similar tastes.

The dataset we looked at was from Yahoo Music, which is from registered users, denoted by their user ID, the item ID, and their rating of that item. The data set contained an unspecified amount of random noise, in both the attributes and the labels. The rating system is a discrete [1, 5] star rating. The dataset contains meta-information for approx. 10% of ratings, including genre and artist.

Collaborative Filtering is the process of making predictions about the tastes of users by collecting large sets of data from prior user information. The assumption is that users who have had similar preferences in the past will have similar preferences in the future. The basic process behind Collaborative Filtering requires large data sets in order to be reliable since average item and user ratings are important. A related method is to collect
data from a user’s actions implicitly, for example implying preference for music users purchase over the music they do not purchase. However, these techniques have their own problems, since there is value in recommending items that a user does not know about.

A common way of improving upon simple item average ratings and user average ratings is using a clustering approach. It is possible, using a similarity metric, to look at subgroups of users who have preferences that are similar to a given user. For the similarity metric, we are interested in item-to-item and user-to-user similarities. In this way we can make better inferences for recommending an item to a given user.

Once the clusters are defined, there needs to be some weighted average of the preferences for each cluster. If the similarity metric used has clustered users appropriately, the average of the clusters should accurately reflect the probability that a new user falls into each cluster. In assigning users to clusters, we also encounter the difficulty in choosing whether each cluster is exclusive or overlapping. Since underlying clusters may be vaguely defined, and thus users and items may not be exclusive from each other, it is important to keep the prediction model flexible enough to incorporate this characteristic of our data. An example of this is the thin divide of Hip-Hop and R&B. Since the distinctions between these two genres of music are often fuzzy, users who like one will often like the other. Therefore, when clustering, these two items’ users may fall into multiple clusters. A model that expresses this idea of flexibility is the Flexible Mixture Model (FMM). FMM addresses the fact that each user and item may belong to multiple clusters, and modeling these clusters can occur separately. FMM works in this way: Let

\[ C = [c_1, c_2, \ldots, c_n] \]  
\[ D = [d_1, d_2, \ldots, d_j] \]

Be the classes of users and  
Be the classes of items

Then let \( Z_y \) be the latent variable indicating the class membership of user \( y \) and the probability of the user \( y \) being a member of that class \( P(Z_y) (1 \leq Z_y \leq n) \) is multinomial distribution on the user class. There is also an analogous latent variable \( Z_x \) indicating the class membership for item \( x \) with the same multinomial distribution for the probability \( P(Z_x) (1 \leq Z_x \leq j) \) that item \( x \) belongs to a certain class. Now let the conditional probability of a user \( y \) given a specific user class \( Z_y \) be \( P(Y|Z_y) (1 \leq Z_y \leq n, 1 \leq Y \leq k) \) and the conditional probability of item \( x \) given a specific item class \( Z_x \) is \( P(X|Z_x) (1 \leq Z_x \leq j, 1 \leq X \leq m) \). For the probability of the ratings we again use a conditional multinomial distribution of the probability of rating \( r \) given a specific user class \( Z_y \) and item class \( Z_x \) :  
\[ P(r|Z_x, Z_y) (1 \leq r \leq R, 1 \leq Z_x \leq j, 1 \leq X \leq m). \]

**RELATED WORK**
Filtering systems are generally categorized into two classes: Collaborative Filtering (CF) systems and Content-based Filtering (CBF) systems. The difference between these two systems lies that collaborative filtering systems utilize the given ratings of training users to make recommendations for test users while content-based filtering systems rely on contents of items for recommendation. In our project, we try to build a collaborative filtering system.

Most collaborative filtering methods fall into two categories: Memory-based algorithms and Model-based algorithms. In memory-based algorithms, rating instances of different users are simply stored in a training database, and the rating of a test user on a specific item is predicted based on the corresponding ratings of training users with similar tastes as the test user. In model-based algorithms, statistical models are learned from the given ratings of training users, and ratings of test users are predicted by using the learned model.

Generally speaking, most collaborative filtering approaches assume that: 1) Users with similar “tastes” would rate items similarly; 2) Similar items would have similar ratings from the same user. Intuitively, the idea of clustering can be exploited in collaborative filtering systems. Compared with memory-based approaches, model-based approaches provide a more principled way of performing clustering, and is also often much more efficient in terms of the computation cost at the prediction time. The basic idea of a model-based approach is to cluster items and/or training users into classes explicitly and predict ratings of a test user using the ratings of classes that fit in well with the test user and/or items. (Jin et al., 2006)

Until now, several different graphical models have been proposed and studied (Breese et al., 1998, Hofmann & Puzicha 1998, Pennock et al., 2000, Popescul et al., 2001, Ross & Zemel 2002, Si et. al., 2003, Jin et. al., 2003, Hofmann, 2003, Si & Jin, 2003). These models are successful in capturing similarities among both users and items by probabilistic clustering, and have all been shown to be quite promising in the collaborative filtering task.

**Bayesian Clustering**

The basic idea of Bayesian Clustering (BC) is to assume that the same type of users would rate items similarly, and thus users can be grouped together into a set of user classes according to their ratings of items. The joint probability of user class ‘C’ and ratings of items can be written as the standard naive Bayes formulation:

$$P(C, r_1, r_2, ..., r_M) = P(C) \prod_{i=1}^{M} P(r_i \mid C)$$

(1)
The joint probability for the rating patterns of user $y$, i.e. \{R_y(x_1), R_y(x_2), \ldots, R_y(x_M)\} can be expanded as:
\[
P(R_y(x_1), R_y(x_2), \ldots, R_y(x_M)) = \sum_C P(C) \prod_{i \in X(y)} P(R_y(x_i) | C)
\]

**Aspect Model**

The aspect model assumes that users and items are independent from each other given the latent class variable. Thus, the probability for each observation pair $(x,y)$ is calculated as follows:
\[
P(x, y) = \sum_{z \in Z} P(z)P(x | z)P(y | z)
\]
where $P(z)$ is class prior probability, $P(x|z)$ and $P(y|z)$ are class-dependent distributions for items and users, respectively.

There are two ways to incorporate the rating information ‘$r$’ into the basic aspect model, which are expressed in Equation (4) and (5), respectively.
\[
P(x(l), y(l), r(l)) = \sum_{z \in Z} P(z)P(x(l) | z)P(y(l) | z)P(r(l) | z)
\]
\[
P(x(l), y(l), r(l)) = \sum_{z \in Z} P(z)P(x(l) | z)P(y(l) | z)P(r(l) | z, x(l))
\]

Unlike the Bayesian Clustering algorithm, where only the rating information is modeled, the aspect model is able to model the users and the items with conditional probability $P(y|z)$ and $P(x|z)$.

**Flexible Mixture Model**

The Flexible Mixture Model (FMM) is a probabilistic model that creates a parameterized number of item and user clusters, classifies items and users as members of these clusters, and predicts unseen values based on a weighted average of item and user cluster membership. FMM has, like the Expectation-Maximization algorithm, both an E-step and an M-step. We based our implementation of this model largely on the work done by (Si & Jin, 2003) at CMU.

The objective function which will be maximized is
\[
P(x(l), y(l), r(l)) = \sum_{Z_x, Z_y} P(Z_x)P(Z_y)(P(x_l | Z_x)P(y_i | Z_y)P(r_i | Z_x, Z_y)
\]
EM algorithm in FMM

In the E-step, a posterior is calculated for

\[ P(itemCluster, userCluster|itemId, userId, rating) \]

\[
P(Z_x, Z_y|x_l, y_l, r_l) = \frac{P(z_x)P(z_y)(P(x_l|Z_x)P(y_l|Z_y)P(r_l|Z_x, Z_y)}{\sum_{z_x, z_y} P(z_x)P(z_y)(P(x_l|Z_x)P(y_l|Z_y)P(r_l|Z_x, Z_y)}
\]

In the M-step, priors are calculated, \( P(itemCluster) \) for each item cluster, \( P(userCluster) \) for each user cluster, \( P(itemId|itemCluster) \) for each item-cluster combination, \( P(userId|userCluster) \) for each user-cluster combination, and \( P(rating|itemCluster, userCluster) \) for all combinations of those variables as well.

\[
P(z_x) = \frac{\sum_{y} \sum_{z_y} P(Z_x, Z_y|x_l, y_l, r_l)}{L}
\]

\[
P(z_y) = \frac{\sum_{x} \sum_{z_x} P(Z_x, Z_y|x_l, y_l, r_l)}{L}
\]

\[
P(x|z_x) = \frac{\sum_{l:x_l=x} \sum_{z_y} P(Z_x, Z_y|x_l, y_l, r_l)}{L * P(z_x)}
\]

\[
P(y|z_y) = \frac{\sum_{l:y_l=y} \sum_{z_x} P(Z_x, Z_y|x_l, y_l, r_l)}{L * P(z_y)}
\]

\[
P(r|z_x, Z_y) = \frac{\sum_{l:r_l=r} P(Z_x, Z_y|x_l, y_l, r_l)}{\sum_{i} P(Z_x, Z_y|x_l, y_l, r_l)}
\]

The priors are initialized to random distributions, and then the E-step and the M-step are run in succession until overfitting starts to occur.

Annealing in FMM

In order to avoid the unfavorable local maximum problems, we use a general form of the EM algorithm named annealed EM algorithm (AEM) (Hofmann & Puzicha, 1998), which is an EM algorithm with regularization. In Annealing EM algorithm, we use following equation to calculate posterior probability in E step,
The annealing EM algorithm includes the following steps (Thomas, 1999):
1) Set $\beta = 1$ and perform EM until the performance on held-out data deteriorates (early stopping).
2) Decrease $\beta$ by setting $\beta = \eta \times \beta$, with some rate parameter $\eta < 1$.
3) As long as the performance on held-out data improves, continue EM iterations at this value of $\beta$.
4) Stop on $\beta$ when decreasing $\beta$ does not yield further improvements.

Rating Prediction in FMM
After training is complete, the rating prediction can be computed using the following equation:

$$R_y(x) = \sum_r r \times \frac{P(x, y^t, r)}{\sum_r P(x, y^t, r)}$$

METHODODOLOGY & EXPERIMENTS

Terminology

1. **MAE**: Mean absolute error
   $$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |f_i - y_i| = \frac{1}{n} \sum_{i=1}^{n} |e_i|.$$
2. **RMSE**: Root mean squared error
   $$\text{RMSE}(x_1, x_2) = \frac{\sum_{i=1}^{n} (x_1, i - x_2, i)^2}{n}$$
3. **VLDS**: Very large dataset, the Netflix prize dataset and the Yahoo music dataset are both examples of a very large dataset.

Pre-Processing the Data
The Yahoo! Music data set already comes very well organized and normalized. Thus, we did not need to modify the format of the data records.

Initially, we had envisioned running our Flexible Mixture Model algorithm on the entire 10GB data set provided by Yahoo! However, we quickly realized that the amount of memory required by our implementation of FMM is approx. 20x the size of the data set. Because of this, we had to create a reduced data set due to limitations on the computing resources we had available and the time needed to implement an algorithm that would use a database or some other efficient storage mechanism to reduce our need for RAM.
Initially, to make the data set viable for quick testing and experimentation, we created a “toy” dataset, that only included the ratings of the top 100 most active raters in the country music genre. Since this set of users should have generally similar tastes for the most part, this is also a way to set ourselves up for success: if we could not find similarity in this set of users, we knew we had a problem with our algorithm. Once we were up and running on this data set, we decided we needed a better data set for testing.

A new combination data set was created, which we have used throughout the rest of this paper, called the “reduced” data set. This data set is a union of the top 300 raters from the rap and country genres. The size of the reduced data set is about 100MB. This was created using a custom Perl script. The output of this script was one training file, with over 7 million unique ratings, and one test file, with about 5800 unique ratings. There are no duplicate ratings in the training and test files.

Finally, the training file was split into one primary training file and one holdout set, with 2/3 and 1/3 of the 7M training ratings in each file respectively. Thus, we used the holdout set for annealing while training, and then used our test file to produce our final numbers.

Based on the size of this data set, we could not run FMM on it on a typical desktop computer, because a typical memory capacity of 2GB of RAM is not even close to enough memory for the algorithm to run a single iteration of the EM algorithm.

Developing Baseline Predictors

In order to measure the effectiveness of our Flexible Mixture Model predictions, we needed to develop some baseline numbers.

Simple Baseline: Always Guess 3
The first baseline predictor we wrote was a python script that simply always guesses a rating of 3 for unseen products, for all users. Running this script on the reduced data set gave an MAE of 1.513644 and an RMSE of 1.674249. Clearly, useful predictors must have a better error rate than this mindless prediction method.

Viable Baseline: Combination of User Avg. Rating & Item Avg. Rating
Due to the relative uselessness of the “Always Guess 3” method, we decided to write another python script to implement a baseline predictor that seemed more viable and useful. We call this baseline predictor the “Average Predictor”. The Average Predictor calculates the user’s average rating over all of his/her ratings, calculates the item’s average rating, and combines the two numbers based on a weighting. A User Bias is chosen from the range [0.0, 1.0], and the algorithm predicts a rating for an unseen product using the following formula:

Rating = UserBias * UserAverage + (1.0 – UserBias) * ItemAverage
On our data set, giving a 1.00 weight to the user's own predictions gives a lower MAE than giving any weight to the item's average rating. However, the RMSE shows some convexity, and seems to indicate that a user bias of 0.90 is ideal. To ensure these trends were not a result of data pre-processing, this method was run on the entire 10GB unreduced data set. The process took about two hours on a typical desktop computer. Most importantly, a similar trend appeared in the resulting data, so we know that these characteristics were not a result of the pre-processing step.

In short, on the reduced data set, for a user bias of 1.00, we got an MAE of 1.010708 and an RMSE of 1.381846. On the unreduced data set, for a user bias of 1.00, we saw an MAE of 1.145393 and an RMSE of 1.418729.

This seems to be a bias/variance tradeoff, and a completely biased prediction is not useful on unseen data. Thus, we would likely consider a User Bias of closer to 0.75 or so more comparable to our primary FMM prediction algorithm. However, we have still chosen to compare our FMM results to the best results obtained using this prediction method. Below is our table of results on the reduced data set.

<table>
<thead>
<tr>
<th>User Bias</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1.479447</td>
<td>1.744372</td>
</tr>
<tr>
<td>0.10</td>
<td>1.429050</td>
<td>1.673028</td>
</tr>
<tr>
<td>0.20</td>
<td>1.378653</td>
<td>1.607547</td>
</tr>
<tr>
<td>0.30</td>
<td>1.328256</td>
<td>1.548673</td>
</tr>
<tr>
<td>0.40</td>
<td>1.277927</td>
<td>1.497184</td>
</tr>
<tr>
<td>0.50</td>
<td>1.227634</td>
<td>1.453867</td>
</tr>
<tr>
<td>0.60</td>
<td>1.184249</td>
<td>1.419470</td>
</tr>
<tr>
<td>0.70</td>
<td>1.140864</td>
<td>1.394652</td>
</tr>
<tr>
<td>0.80</td>
<td>1.097478</td>
<td>1.379930</td>
</tr>
<tr>
<td>0.90</td>
<td>1.054093</td>
<td>1.375629</td>
</tr>
<tr>
<td>1.00</td>
<td>1.010708</td>
<td>1.381846</td>
</tr>
</tbody>
</table>

**Implementation of FMM in Python**

Implementing the Flexible Mixture Model in Python was an interesting process. We initially chose python because of the well-known NumPy and SciPy libraries, which are typically supposed to make operations involving vectors and linear algebra easier. There are also some scientific algorithms built into SciPy. While none of us had a lot of experience with Python, we felt this would make it easier to write the FMM algorithm from scratch. Ironically, we could have written this algorithm in any other language because we did not end up using NumPy or SciPy at all in our implementation. Because the summation loops in the algorithm never sum over all ratings, but generally are quite selective about which ratings to sum over, a classical loop was easier and less memory- and CPU-intensive to utilize.
One major benefit of a language like Python for this task is the ease with which file parsing and tokenization is done, as opposed to languages like C. Additionally, the built-in Python type Dict provides for fast and easy management of hashes of data, of which we make extensive use. In fact, we use this flexible data structure for the following critical parts of the FMM algorithm.

```python
jointPosteriorProb = {}  # posterior prob. of joint clusters given an instance of data
probItemCluster = {}  # maps from item clusters -> marginal prob of those clusters
probUserCluster = {}  # same for users
probItemGivenCluster = {}  # prob of item|itemCluster
probUserGivenCluster = {}  # prob of user|userCluster
probRatingGivenClusters = {}  # prob of rating | itemCluster, userCluster
```

**Finding the right parameters for FMM**

In order to have good predictions, we had to learn how to tune the model for maximum performance. So, we ran experiments to find out how our results varied as we varied our parameters. The figure below shows runs of FMM, labeled by Eta_ItemCl._UserCl.
Our algorithm was run in total with fourteen configured parameters. We had the resource constraint of only being able to run these experiments on six individual high-performance computers in a shared environment. Each of these computers contained 16GB of RAM. We began with exploratory analysis; the first step in finding which parameters maximize performance is to try several different values for each parameters. This strategy is similar to blindly throwing darts and then exploring which stances maximized score. These graphs show the MAE calculated on the held out data at each iteration from four separate experiments. The top right corner was run with Eta, user clusters, and item clusters equal to .42, 3, and 1, respectively. The horizontal line represents the final MAE calculated on the test file. It can be seen that the most efficient experiment was achieved in the top right corner .42, 3, and 1, which converged in a quick 20 iterations. The bottom left corner, .4, 3, and 3 beat the final MAE by .05 (.92 instead of .97) but takes 232 iterations before the annealing constraint is met.

After completing this set of exploratory experiments we found that user clusters equal to three was the most important factor in having a good MAE. The two primary goals for the rest of the experiments were to achieve the best MAE and to explore the affect of different etas. Testing if a specific eta would boost performance was a priority because if we found a golden value then we would want to use that same value in our MAE experiments.

<table>
<thead>
<tr>
<th>Item Cluster</th>
<th>User clusters</th>
<th>Eta</th>
<th>MAE</th>
<th>RMSE</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>.3</td>
<td>0.973819</td>
<td>1.256500</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.5</td>
<td>0.968069</td>
<td>1.254568</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.7</td>
<td>0.972775</td>
<td>1.256018</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.42</td>
<td>0.972906</td>
<td>1.287086</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.42</td>
<td>1.167015</td>
<td>1.327152</td>
<td>**40</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>.3</td>
<td>1.450687</td>
<td>1.616877</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.5</td>
<td>1.449565</td>
<td>1.616788</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.7</td>
<td>1.450954</td>
<td>1.615893</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.42</td>
<td>1.449190</td>
<td>1.615372</td>
<td>*171</td>
</tr>
</tbody>
</table>

* terminated before annealing
** New version to avoid underflow

It can be seen from the above experiment that eta has a marginal affect on the final MAE. The effect on the number of iterations the algorithm runs through before annealing is inconclusive. Based on the above table, the experiment was designed to test two
configurations of item and user clusters with four different eta values: .3, .42, .5, .7. It can be seen that there is no direct relationship between the learning rate and any of the final metrics: MAE, RMSE, number of iterations. It is also notable that the total number of iterations has no effect on any of final metrics. In regards to this specific experiment, the best MAE was achieved with item cluster, user cluster, and eta equal to 1, 3, .5 respectively. That value of eta probably had little to do with the 0.005 increase in performance over the other experiments with the same parameters with varied etas. The conclusion from the Eta experiments is that there is no golden value for Eta that maximizes performance regardless of other parameters. The constraint on the learning rate, eta, is that it is a value less than one and greater than zero. Even when the other parameters are fixed, there is no clear advantage in terms of MAE or number of iterations prior to annealing.

The FMM algorithm is very memory hungry. Specifically, the product of user and item clusters is equal to the number of probability hashes that must be stored for each user. Additionally, the speed of the algorithm is likewise approx. linear in the product of user clusters and item clusters. Any attempt to run our algorithm on a very large dataset with a very large number of clusters resulted in a memory error, even on systems with 16GB of RAM. The results of our experiments are below.

Table: Results of running FMM with different numbers of clusters

<table>
<thead>
<tr>
<th>Item clusters</th>
<th>User clusters</th>
<th>Eta</th>
<th>MAE</th>
<th>RMSE</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>.8</td>
<td>0.995363</td>
<td>1.281100</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>.8</td>
<td>0.929219</td>
<td>1.212783</td>
<td>*59</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>.4</td>
<td>0.920588</td>
<td>1.233844</td>
<td>232</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>.42</td>
<td>0.934684</td>
<td>1.259438</td>
<td>113</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>.5</td>
<td>0.904797</td>
<td>1.231812</td>
<td>661</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>.5</td>
<td>1.003154</td>
<td>1.306884</td>
<td>208</td>
</tr>
</tbody>
</table>

* terminated before annealing

The best results were attained after running 600 iterations with three item and four user clusters with an eta value of .5, which resulted in the final MAE computed on the test file of .905, in the context of the problem that means when we predict a rating, one through five, for an item given a user, we are off by .905 on average. The VLDS we analyzed is the property of Yahoo! As as such we could not find any published results that could be used as a baseline, which is why we created our own.
CONCLUSION

Comparison with Best Baseline
In conclusion, we have shown that the Flexible Mixture Model will beat an Average Prediction-type (un-clustered) model. Final results are in the table below.

<table>
<thead>
<tr>
<th>Method</th>
<th>MAE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad Baseline (always predict rating=3)</td>
<td>1.513644</td>
<td>1.674249</td>
</tr>
<tr>
<td>Viable Baseline (user bias=0.75)</td>
<td>1.119171</td>
<td>1.356237</td>
</tr>
<tr>
<td>Best Baseline (always predict user avg. rating)</td>
<td>1.010708</td>
<td>1.381846</td>
</tr>
<tr>
<td>Best FMM (item clusters=3, user clusters=4)</td>
<td>0.904797</td>
<td>1.231812</td>
</tr>
</tbody>
</table>

Our model (FMM) outperformed predicting the user’s average rating by 0.1. We believe that if we could run our model with more clusters we would reach further improvements.

Technical Issues and Limitations
Because we ran FMM, a very high-powered algorithm, on a VLDS we had memory and computation limitations. Luckily for us based on the computers we had access to, six computers with 16 GB of random access memory, we could run our algorithm on a large chunk of data at a time. Unfortunately we would need a real super computer in order to run the entire dataset (10 GB) with a large amount of clusters. There is a direct relationship between the size of the dataset and the product of clusters in terms of what hardware is required to predict ratings. Therefore we did the best we could, selected an area of the dataset that was somewhat sparse, but certainly had genre clusters, and experimented by running the algorithm with more and more clusters until the memory limit was reached. Our best results took over a week to be computed.

Future Goals
We could like to refractor the FMM code so that it stores its results in a database. In this way, we could make use of running totals and reduce the memory footprint of our algorithm. It would be interesting to see how this algorithm could be adapted to predict well on very large data sets with reasonable memory constraints in a viable amount of computational time.

REFERENCES


Pennock, D. M., Horvitz, E., Lawrence, S., Giles, C. L. Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach. In the Proceeding of the Sixteenth Conference on Uncertainty in Artificial Intelligence, 2000


Si, L. and Jin, R., Flexible Mixture Model for Collaborative Filtering, In the Proceedings of the Twentieth International Conference on Machine Learning, ICML 2003


Thomas Hofmann, Probabilistic latent semantic indexing, Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, p.50-57, August 15-19, 1999


Netflix Prize. Netflix Movie user ratings http://netflixprize.com

ACKNOWLEDGEMENTS

UCSC Baskin School of Engineering
Professor Yi Zhang
Professor David Helmbold
Professor Hongyun Wang
Thomas Aguilar