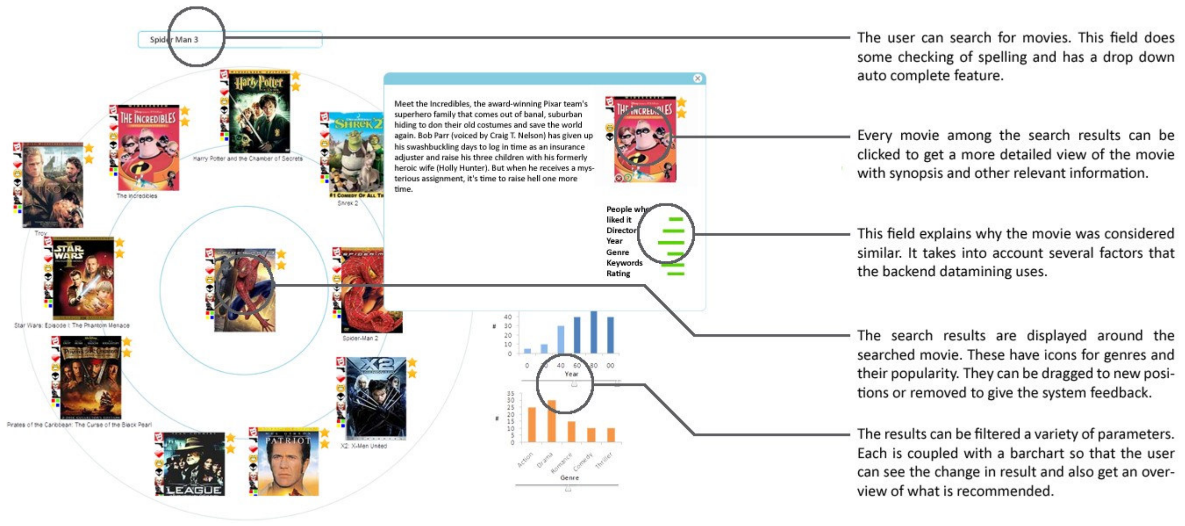
Shiniphy - Visual Data Mining of movie recommendations

Filip Gruvstad, Nikhil Gupta and Shireesh Agrawal



Meet the Incredibles, the award-winning Pixar team's superhero family that comes out of banal, suburban hiding to don their old costumes and save the world again. Bob Parr (voiced by Craig T. Nelson) has given up his swashbuckling days to log in time as an insurance adjuster and raise his three children with his formerly heroic wife (Holly Hunter). But when he receives a mysterious assignment, it's time to raise hell one more time.

The user can search for movies. This field does some checking of spelling and has a drop down auto complete feature.

Every movie among the search results can be clicked to get a more detailed view of the movie with synopsis and other relevant information.

This field explains why the movie was considered similar. It takes into account several factors that the backend datamining uses.

The search results are displayed around the searched movie. These have icons for genres and their popularity. They can be dragged to new positions or removed to give the system feedback.

The results can be filtered a variety of parameters. Each is coupled with a barchart so that the user can see the change in result and also get an overview of what is recommended.

Fig. 1. Shiniphy - Visualization & Interaction techniques highlighted in mockup

Abstract— An interactive visualization system for browsing results of a movie recommendation system is presented. The movie recommendations are based on collaborative filtering data mining techniques on Netflix Challenge dataset. The visualization allows for two way feedback, from the recommendation system to the user, and from the user to the system. The user can intuitively change the results of recommendations over time. Dynamic querying filters are also provided.

Index Terms— visualization, data mining, interaction, collaborative filtering, recommendation system, feedback, Netflix challenge, search visualization.

1 Introduction

There are many online recommendation systems, like Netflix.com and Amazon's similar-product recommendations, which show top results of a data mining back end. They generate thousands of recommendations, but are constrained to show only the top 3 or 4 out of them. Other sites like Jinni and Pandora have more interactive and visually appealing results but these use datasets specifying the genome of the song/film which are painstakingly made by hand. Many a times these results are good, but they do not give any feedback as to why a movie is being recommended. Also, a user has no way of letting the site know that two movies do not seem related to her.

We wanted to build a data mining system without going through the painstaking process of building a genome. We finally built a movie search and recommendation system which finds similar movies to the movie being searched for using Netflix prize's dataset (something like Amazon's product recommendation system). This system uses both user ratings and metadata from IMDb to give its results. The visualization of the results enables the user to find out more about the results, filter them by several parameters (such as genre and production year). By having a strong data mining element coupled with usable visualization techniques we can enable effective and quick exploration of the results, by selectively boosting and dropping scores based on user queries. This report will focus most of its attention on the data visualization element.

One very interesting possibility is giving the user the possibility to modify the results. Our system gives recommendations where the best matches end up closest to some point, if the user does not agree with this he/she can change it and the map is updated. Thus the results of the data mining will improve over time. We also wanted the user to be able to participate in the mining process and the system to give feedback to the user why a certain movie was recommended (were they by the same director?).

Our hope is that this system will make the users more interested in watching movies and can help them find titles they have never heard of.

1.1 Motivation

In the following paragraphs, we talk about problem visualizing large data sets, data mining, why a visualization system would be helpful, and how collaborative filtering (a type of data mining) can leverage such visualizations for item-item recommendations to exploit the "long tail".

The exploration of large data sets is a difficult problem. Though clever visualization techniques partly help to solve the problem, still sometimes it is easier for humans to look at computed patterns and a good visualization of the data to find more intricate patterns in the data. Thus, involving the user in the data mining process by pre-processing the data using simple statistical techniques and then presenting the results to the user, visually, is a better idea. Integration of interactive visualization and data mining algorithms combines the intuitive power of the human mind with computational capabilities of the computer, thus improving the quality and speed of the data mining process. There is a company working towards this specific goal Palantir.

In simple terms, data mining is the process of extracting important relationships and patterns from large amounts of raw data. For example, which two items are bought together the most (DVDs + Popcorn? Then give some offer on the combo). It is used in a number of different fields nowadays including marketing, detecting fraud and in scientific discovery like bioinformatics. However, data mining is not perfect, and many a times cannot beat human intuition.

Also, it is not easy for a poorly informed end user, who is not a mathematician or computer scientist to interact with data mining system parameters and get useful results. They might even get results that are misleading or misinterpreted. Having a system which indicates why a particular pattern is being shown, and what the different parameters contributed towards that pattern would help the end user a lot, and make the system much more usable. Other factors like taking users' feedback in an intuitive manner and allowing for dynamically querying of results would help to solve this problem a lot.

A modified definition of Collaborative filtering (CF) from Wikipedia is – it is a form of data mining system. The underlying assumption of CF approach is that those who agreed in the past tend to agree again in the future. It can produce personal recommendations by computing the similarity between users or items themselves, in a large set of data. It is the method of making automatic predictions (filtering) about the interests of a user by collecting taste information from many users (collaborating). For Shiniphy, we concentrated on item-item based collaborative filtering recommendation system.

Enabling the user to explore a substantial number of recommendations quickly with dynamic filtering would allow a service provider to exploit the "Long tail" that exists in their products. "Long tail" suggests that 20% of the products account for 80% of the sales. The remaining 80% of the products ("non-hits" or "long tail") account only for 20% of the sales, and it will be highly beneficial for the service providers if users start buying more from amongst the non-hits.

1.2 Related Work

In [6] the importance of having a visualization system for a data mining system is described. In [4], an approach for getting online data mining/CF is described, however, given the static data that we have, we realized that having a "tunable" backend would be better.

The project that is most related to our work is [2] "Netflix Movie Explorer". It can show posters of movies related to a particular keyboard. It is also based on the Netflix prize data set. However, it does not provide any way of giving feedback to or receiving any feedback from the system. There are several sites online that recommend movies and music; some interesting examples are [12]-[16]. They all use some kind of recommendation system coupled with a visualization system. However, [13]-[16] lack do not give any feedback to the user about why a particular item is being recommended. Jinni [12] and Pandora [17] do that, but they use metadata created by hand. (See Fig 2)

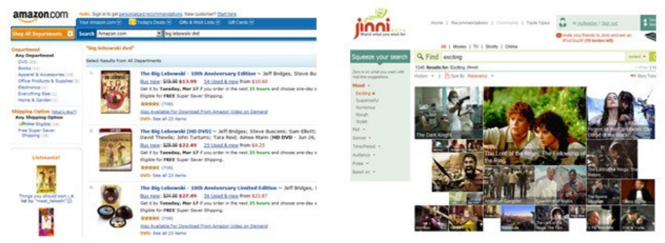


Fig. 2. Amazon and Jinni screenshots

1.3 Data Available

We got movie ratings from the Netflix dataset which contains over 100 million ratings (along with the data when the rating was given ranging from Oct-98 to Dec-05) of 17770 movies by over 480,000 different users. Furthermore this data has information about when the ratings are given and by which user. It does not however, contain any data about the movies other than which year they are produced, therefore we also chose to use IMDb's extensive database to find additional information like genre and director. Since we could not find any publicly available dataset containing both Netflix and IMDb data we had to try matching these by title somehow.

We ran some basic queries on it to get a better sense of what we were dealing with. We observed that there was a long tail in where we expected, like in the number of ratings per movie and the number of ratings per user. This data set is interesting because there were also some outliers in it, like 2 users who rated over 17,000 movies each. We could have removed such extreme cases and pruned the dataset to achieve incrementally better results, but we decided to focus more on the algorithms to get rid of such cases. Matching IMDb and Netflix titles turned out to be a difficult problem. We tried several things to find as many title matches as possible. We start of by doing some preprocessing like putting "The" at the end of a title. We then use IMDb's "also known as" aka-titles list (which contains alternative titles for different movies). We removed the titles which had fewer than 100 ratings (to improve quality and speed since the aka list is very large). We then compared all the titles to the ones in the Netflix list and replaced possible aka titles in it with what IMDb perceived as an original. To compare titles we use Levenshtein Distance (counts number of deletions, insertions, or substitutions required to transform one string into another), length, year and some homemade shingles like comparison to see how many pairs of letters both strings contain. Then we compared this new list of titles with the full list of movies from IMDb and kept the pairs found (making sure the date is correct). This didn't give good enough results so we figured Google might be able to help. We made a simple web crawler that searches Google with site:www.imdb.com and all the titles that where untouched by the above algorithm. It then chooses the first result, scrapes it to get the date and other data and checks the date and string comparing and if the result is favorable adds it to the list. We got banned a couple of times for being a bot but got it to work after some time. Once the titles were compared and ready, the rest was easy. We finally were able to correlate around 70% of Netflix and IMDb data.

For each of the 17,700 movies, we scraped an internet site to fetch the posters using a Python script. However, again was a slow process, as the server was again blocking us progressively. Finally we received all the posters from the author of the Gflix [2] site, Chaitanya Sai. We downloaded all the movie synopsis from the same site using our own script.

2 Methods

For this project many elements were needed. We have lots of data must be accessed and processed quickly. We also have a demanding visualization with many elements. As stated in [3] & [4], we designed the system considering the following factors –

- It should be easy to implement and maintain, yet reasonably accurate - We used a simple aggregation formula and class architecture
- The schemes should not rely exclusively on batch processing of data, so we stored good results of data mining in tables and made use of them in aggregation of scores efficient at query time: we used extra storage to speed up the execution type

2.1 Visualization

We considered many different options for the visualization that would lend itself the best for such kind of a system. More specifically, we considered a linear display, a graph-node tree structure, and a radial display. We also considered different kinds of encodings possible for different data in each of these visualizations, in size, color, distance, width, length, angle, overlap etc.

We will now discuss the pros and cons of these different approaches and our final choice.

A linear tree map display, with posters of different sizes binned according to relevance of the result, somewhat like Jinni, is what we wanted to build first. The biggest plus point of such a design is that humans are very quick at assimilating linearly displayed information.

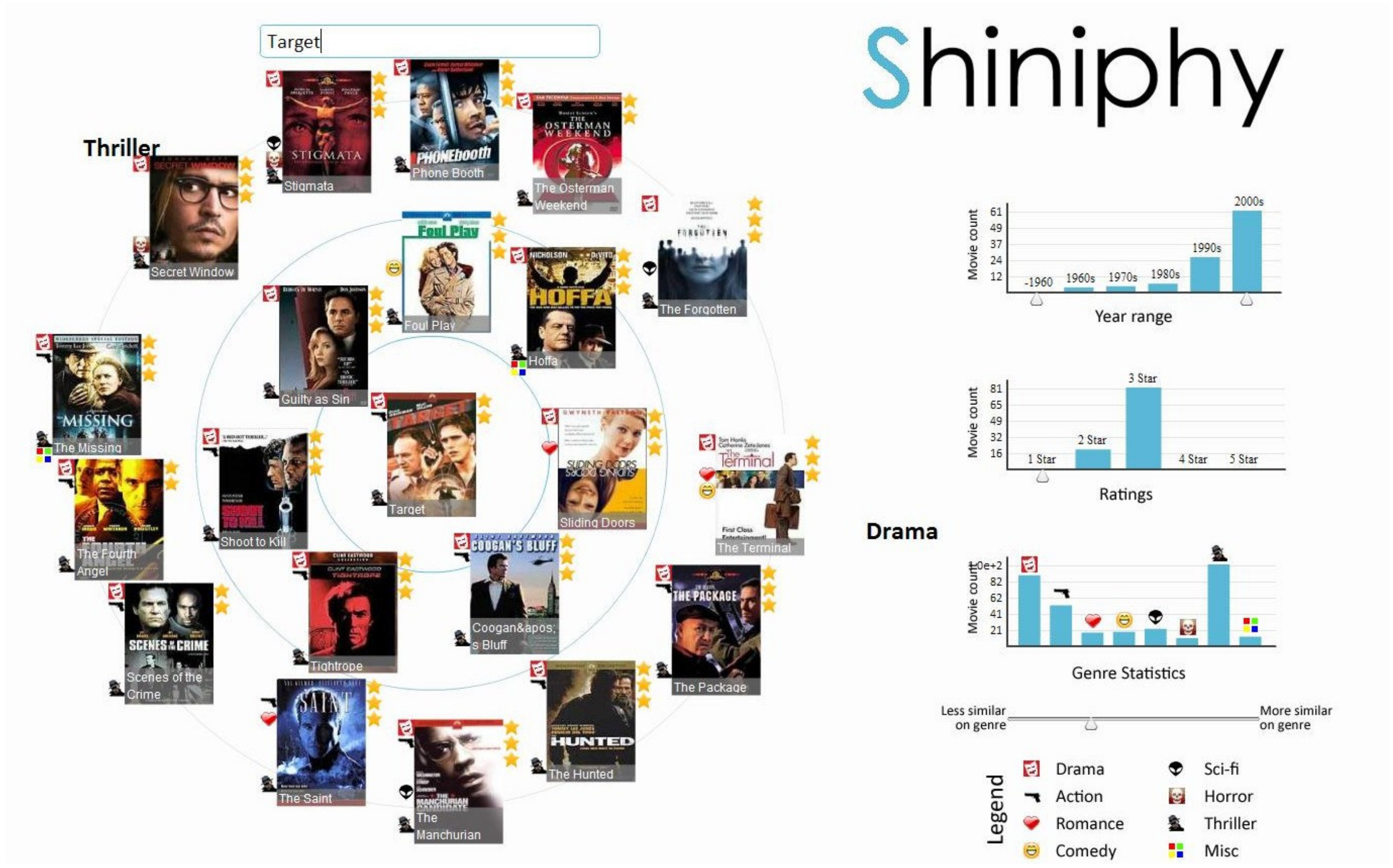


Fig. 3. Screenshot – showing the visualization when user searched for a movie "Target"

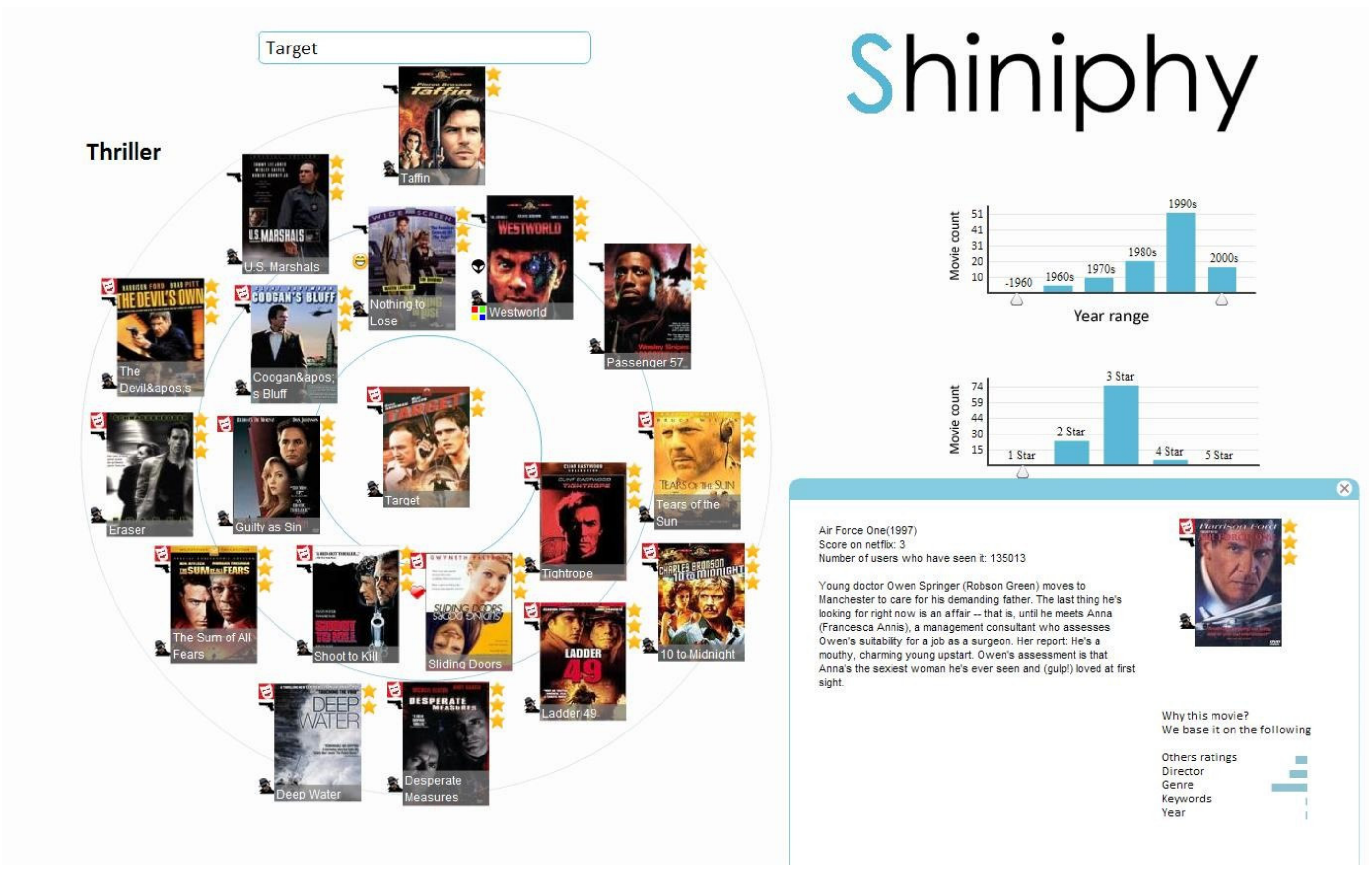
However, with this design, we soon realized that if the number of related movies in a particular bin was more than the number we could fit in, and then either we'd have to put those movies (wrongly) in a smaller poster size bin, or show it on the next page. Both would now be good, as we want the top results to be shown first and with same size. Also, if the user changed any dynamic query parameter, then the posters would not so pleasantly fly off in different directions. In case of interactive feedback from the user, we could not think of any simple way in which the user could tell the system that two movies were not so related or more related. Jinni [12] uses a linear treemap visualization, with poster size encoding the relevance of a recommendation. However we found that this visualization is not so effective for movie recommendation, as smaller posters are difficult to see, and it is very distracting to see the posters change size whenever any of the dynamic querying is applied.

Next we considered a graph node structure, however this has the problem of wasting a lot of space, and drawing edges between "movie" nodes might encode some good information (like same year, or progression etc.) but it would not make sense intuitively to a user. This visualization also suffered with the user feedback problem discussed above. Musicovery [16] has a very interesting visualization, but the placement of the different songs on screen and their "relatedness" is not encoded in the visualization at all.

Finally, we decided that a radial display, with the movie searched for at the center and recommended movies around it in a circle. Only around 10-15 movies can be shown at a time, and this is by design, as with anything more than this, the system looks too cluttered (and specifically since this is not a linear display). Also, this would in a way prompt the user to use the dynamic querying and feedback facilities provided. The distance of a movie from center of the circle encodes the similarity of the movie, so more similar movies are closer to the center. This is normalized for the movies displayed, so that the best recommendation is always at a fixed distance from the center (this also ensures that if there are no good recommendations, they stay on screen). The position of the posters is determined algorithmically. So that they do not overlap. This makes taking users' feedback very easy, as they can just drag a recommendation closer to the center to say it is more related (in a fuzzy way) and farther away otherwise. (See Fig 3 & Fig 4)

Here are some other features implemented –

- Each movie can belong to multiple genres. For all the movies shown on screen, top 5 genres and rating (on a 5 star scale) are shown as icons along with the posters, this allows to quickly see if a movie is in the genre the user is searching for. Total 28 genres available were binned into around 7 most popular categories (like "Horror" was grouped with "Thriller").
- We provide a quick statistic in microbars about the genres and Year of release of all the recommendations along with dynamic querying options on Year, Average rating of recommended movies & Genre. Based on these statistics, it is very easy to decide which year/genre the searched for movie belongs too, and which year/genre the user currently wants to focus on (by dynamic querying).
- Genre similarity search – this slider gives feedback to the data mining system and changes the weight associated with the "genre clustering" score contribution.
- To emphasize that movies closer to the center are more related, concentric circular rings have been drawn around the center with decreasing colour darkness.
- Dynamic querying interactively changes the results being shown.
- Feedback from the system, that is why a movie is being recommended is shown only on clicking an information button. This pops out an info box with a synopsis and different parameters that contributed to the movie being recommended.



Air Force One(1997)
Score on netflix: 3
Number of users who have seen it: 135013

Young doctor Owen Springer (Robson Green) moves to Manchester to care for his demanding father. The last thing he's looking for right now is an affair -- that is, until he meets Anna (Francesca Annis), a management consultant who assesses Owen's suitability for a job as a surgeon. Her report: He's a mouthy, charming young upstart. Owen's assessment is that Anna's the sexiest woman he's ever seen and (gulp!) loved at first sight.

Why this movie?
We base it on the following

Fig. 4. Screenshot – Showing the feedback from the system in the infobox when user clicks on a particular movie

- Two circular guide lines with some difference in their radius are drawn statically to aid in determining the distance of a recommendation from the center.
- Major "genre" matches are segregated and placed together with the genre name written in bold in that area. This is redundant encoding of the genre information.
- We chose not to use any other encodings in edges from center to the nodes, poster size, border size etc. to keep the system simple and non-cluttered.

2.1.1 Efficient poster placement strategy

Placing the posters efficiently on screen was an important task, so as to allow effective usage of the screen space, while also allowing the encoding of relatedness in radial proximity from center. First we tried using a force directed layout kind of structure, with posters bound to concentric circles and tangential forces placing them separately from one another. However, this did not work out to be that efficient, as FDL takes a lot of computational power on the user's machine, and also because it does not allow much algorithmic control on certain parameters like group movies of similar genre together. So we tried an algorithmic placement strategy, in which we take around 100 movies and compute their theoretical radius (r) and angle (theta) values. If a node is overlapping with some other one already placed, then a simultaneous search commences in both the (r) and the (theta) spaces. This gave us good results, and also allowed us to group movies algorithmically, and consistently.

2.1.2 Workflow & Interaction

Interaction techniques implemented in the system include –

- User searches for a movie – auto complete happen on the fly.
- Selected movie shown at center of the visualization.
- Feedback from the system in the info box and the microbars, and the distance from the center.
- Dynamic querying.
- User can drag and change radius of movie displayed to give feedback to the system.
- User can say that two movies are not related at all by clicking the close button for a particular recommendation.

2.2 Data mining

An item-item collaborative filtering recommendation system is a "non-personalized" system as it doesn't depend on the user profile. For example all users (not logged on) browsing a product on Amazon, web site see the same recommendations.

In simple terms, it works as follows (from [3]) – In effect, given that the user is browsing item number 1, say, we are looking for items with ID "itemID2" such then: a sufficient large number of people (defined by a threshold) rated both item 1 and "itemID2" on average, we want "itemID2" to be rated as high as possible among users who also rated item 1.

Finding similar movies requires lots of preprocessing of the data. Since this paper is more about visualization than about data mining, we just give an overview of the data mining techniques used and interested readers can read [18] for details of the data mining implementation –

Residuals - Normalization technique to handle the inherent a user to user variability in ratings, some users are harsher while giving ratings, they'll give a movie that they didn't like a "1", and some users would give such movies a "3" rating.

Pearson Correlation score - It is a correlation score which calculates a similarity score for any pair of movie based on how common users have rated the two movies.

We ran code optimized for calculating the coefficient which indexed the ratings both by user and by movie and allowed for fast in memory lookups. We limited the size of the output file to around 50Mb by calculating the Pearson correlation only when the overlap between two movies was over 50 movies, and clipped and kept the coefficients above a value of 0.5 only (determined by hit and trial on a smaller subset of the entire database). We did this because we wanted our visualization to be interactive and having a larger table of Pearson coefficients slowed down the querying too much. The results for Pearson alone are fairly good. We checked for around 5 movies and could always see sequels in the result. There were cases where for a given movie; one movie with very few users in common had slightly higher correlation than correlation with other movie with many more users in common. To boost results with more overlap higher, we recalculated the similarity coefficient as = log(support) * (Pearson coefficient).

Singular Value Decomposition – One of the most popular techniques for the matrix factorization is singular value decomposition or SVD. It decreases the dimensionality of the User X Movie matrix greatly, with little loss of information. Inherently, SVD rids us of noise and helps detecting the overall patterns in the data and helps in characterization of the structure.

Clustering – Clustering is an easy way to make sense of high dimensional data. It groups similar items together. We applied clustering to a variety of different parameters which we then used to boost the final recommendation score.

- We chose to cluster the SVD data because it was proving too slow to load all of it from the SQL database. The results were quite good, for example six out of eight Batman movies in the set ended up in the same cluster when we had 28 clusters.
- We also clustered based on movies IMDb keywords. This data is very sparse but still managed some interesting results like grouping movies with similar themes, for example we found a cluster where over 50% of the movies where about space. This gave some good results, like all the sci-fi space related flicks ended up in the same cluster.
- Finally we clustered on genres.

For the different parameters we try several different amounts of clusters. This is useful for example if two movies are more closely related then they will continue to end up in the same clusters as the clusters get smaller. To do these things we used a clustering library called Cluto which required some pre-processing of the data but was then very easy to use. We tried different clustering methods and distance functions; our final method of choice was k-1 bisections with a cosine distance function, as this gave the best results observed.

2.2.1 Blending the results

Blending multiple results is one of the biggest challenges of our project. We have different parameters on which we can our match and boost our scores. We finally used a linear combination of the different parameters multiplied by weight for each parameter. This would allow our visualization to easily interact with the backend system and change the results, as the user wishes (suppose she wants to see more famous movies, then we just boost the weight for the parameter for movie support).

The parameters we have are:

- Clusters of different sizes based on Movie matrix (from SVD), Genres and Keywords. If the two movies are in the same cluster (genre, SVD, keyword) then boost its score. Boost more if they co-occur in a more specific cluster (two movies in the same cluster when we have 200 clusters are more important than when we only have 10 clusters). SVD should be boosted higher than the other two. The user should be able to decide how important the genres are.
- Use the number of times the movie has been rated to boost up the results for more famous movies. In a similar way, adjust the Pearson score to account for the number of users in common between two movies. A movie that has fewer common users should be given a slightly lower weight.
- IMDb average rating and Netflix average rating. We divide the ratings into different blocks of 0.5 from 1 to 1.5, 1.5 to 2.0 etc. and boost the scores for higher rated movies higher (uniformly in a particular block).
- Number of times a movie has been rated (support).
- For any Movie-Movie pair, the number of users in common who rated both the movies (support).
- Year of release of the movie.
- User suggestions for any set of recommendations. Users can tell the system that two movies are related to a higher degree or not related at all. This data will be stored on a table in the backend server with slow and incremental changes (so two movies do not become unrelated when only one user says so, this will happen only when a substantial number of users indicate the same). This table will also be used to calculate the similarity scores in future.

Based on the value of these different parameters, users can learn why a particular movie was recommended to them and logic to change the weights for these parameters can be easily implemented. For example if we wanted the functionality to allow the user to control that more popular movies (with higher number of ratings) are shown higher up, then that is equivalent to providing a widget for changing the weight of the relevant parameter used for calculating the final similarity score.

2.3 System architecture

To make our visualization accessible and easy to use we want it to be available online but it must also have access to a database and be fast. Since our system needed to work interactively given any user's queries, we stored the results of the different techniques on a MySQL server. As an intermediate step to the visualization we have a Java server which handles the actual SQL queries, this is good because Java is much faster than Flash. This also means that the results of some SQL queries which are small and frequent can be stored in memory on the Java server speeding things up further.

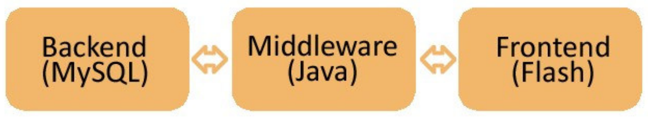
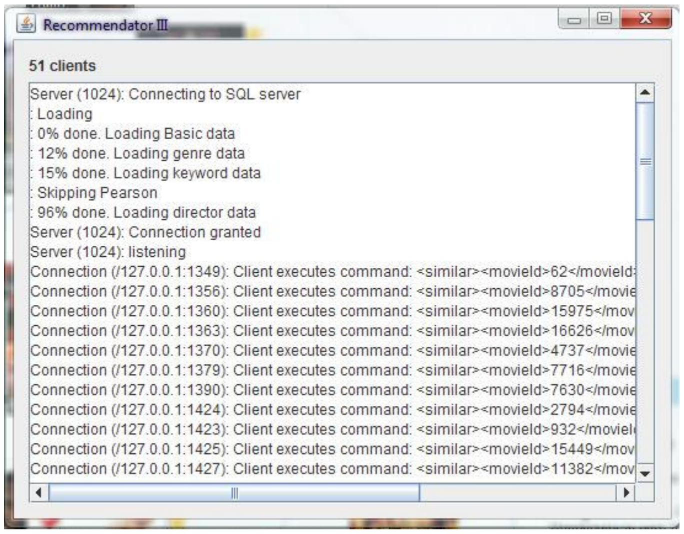


Fig. 5. System architecture

- On the front end we have a Flash/Flex based interface built using the Flare visualization library.
- An interface middle level built in Java to interact with the MySQL server. This level was needed since Flex does not allow direct coupling with database. Also, having the interface in Java allows for much faster interface with the SQL server.
- Backend MySQL server. This stores all the different results tables and allows us to access the data from front end in an efficient manner. Doing dynamic queries is much easier in this matter, as one just has to play around intelligently with SQL queries now.
- The frontend flash application is connected to the data stored in the backed MySQL server via a Java server. This server precaches lots of the information needed to recommend movies and thus offloads the front end (which due to Flash is pretty slow) and backend (where data resides on disk).
- The Flash application is connected to the middleware Java server via XML sockets. From it, it gets sorted lists of recommendations for which it then fetches movie posters, ratings etc. Each search also comes with some common statistics over common genres, ratings and production years.

These are visualized as icons and microbars using Flare, and can be filtered upon using dynamic query sliders provided.

- *Tools & Languages used:* Java, Flash, MySQL, Flex, M/R framework, Python, IMDbPy.



```
51 clients
Server (1024): Connecting to SQL server
: Loading
: 0% done. Loading Basic data
: 12% done. Loading genre data
: 15% done. Loading keyword data
: Skipping Pearson
: 96% done. Loading director data
Server (1024): Connection granted
Server (1024): listening
Connection (/127.0.0.1:1349): Client executes command: <similar><movieId>62</movieId
Connection (/127.0.0.1:1356): Client executes command: <similar><movieId>8705</movie
Connection (/127.0.0.1:1360): Client executes command: <similar><movieId>15975</mov
Connection (/127.0.0.1:1363): Client executes command: <similar><movieId>16626</mov
Connection (/127.0.0.1:1370): Client executes command: <similar><movieId>4737</movie
Connection (/127.0.0.1:1379): Client executes command: <similar><movieId>7716</movie
Connection (/127.0.0.1:1390): Client executes command: <similar><movieId>7630</movie
Connection (/127.0.0.1:1424): Client executes command: <similar><movieId>2794</movie
Connection (/127.0.0.1:1423): Client executes command: <similar><movieId>932</moviel
Connection (/127.0.0.1:1425): Client executes command: <similar><movieId>15449</mov
Connection (/127.0.0.1:1427): Client executes command: <similar><movieId>11382</mov
```

Fig. 6. The Java server – *"Recommendator"*

3 Results

Visualizations should be judged in the context of use. We found it difficult to judge the usability of this system ourselves, and even a small user study could not be conducted due to time constraints. Quantitative measures are difficult to estimate as we have not had any such test. We will present a few quantitative parameters that we could measure ourselves and informal user feedback that we could gather.

3.1 Visualization - User feedback

We made some informal observations at the final poster presentation day though, and present them over here –

- People had needed to be explained that radial proximity indicated closeness.
- Also the concentric circles were difficult for users to see. But we think it was because of: the screen angle and distance of the laptop. Also, the time spent by users on this was quite less.
- We saw that once we explained the features to the user they could easily understand it and found it useful.
- The few other users, who had issues understanding the system, were mostly related to the data mining part of it. However we feel that it was more out of curiosity about understanding the system, and we have successfully abstracted the user from knowing any details of data mining system and how it is implemented.
- People suggested have to the ability to double click on a movie poster to make it the search movie.
- People liked that we showed the genres on outer ring of (like drama, action). They didn't pay much attention to genre icons. They also suggested that we should have more filters for future.

In all, the user feedback was that our visualization was effective for the task at hand, and that it would decrease the time they needed to find interesting (and not so well known) movies, and make it easier for them to decide.

3.2 Visualization - Quantitative measures

The only quantitative measure that we measure effectively were running times, as this is supposed to be an interactive visualization system with three tier architecture, we needed the running time to be fast. To get our system running is a three step process, in the first step one needs to start the MySQL server, in the second the Java server and finally the visualization front end. We observed that MySQL started as soon as the server was rebooted, in around 1 minute, and the Java server took around 2 minutes to start and pre-fetch the basic information in its memory. However, these two would not need to be restarted in an ideal case, and the only time noticeable to users would be the visualization time. We accessed the visualization from two different PCs and observed that it never took more than 4-5 seconds to load the movie information and all the posters, so the visualization part is light weight, and uses minimal resources. This is a good result, as this shows our success in making the system close to real time, and providing a visual data mining system.

3.3 Data mining results

We first present the results for different methods we tried and then for the overall system –

- Clustering on basis of SVD was able to group related movies in the same cluster, like 6 out of the 8 Batman movies were clustered together.
- Keywords based clustering worked well too, while analyzing the clusters, we observed that most of the sci-fi space related movies (Star Trek, Star Wars etc) had clustered together.
- Pearson correlation worked well out of the box, for most of the genre specific movies, it was able to give good recommendations. It was also able to catch sequels without fail.

Again, it is a difficult task to evaluate something like movie recommendations; some people might like the system's recommendations, others might not. It depends a lot on the person actually using the system.

- We compared our results with Amazon's movie recommendations. (However, Amazon's database has >>17,700 movies).
- We checked recommendations manually for movies from Amazon.
- On an average 5/20 movies were common in the recommendations by our system and by Amazon.
- There were cases where:
- Amazon was not able to make any recommendations for some movies like "Khakee" (a not so well known Bollywood movie), while our system did make good recommendations (Bollywood movies in the same Genre).
- Most of the times both systems worked well, especially in comedy and action genres (Ace Ventura, Terminator). We also observed that Amazon weighs much more heavily on the Actor and Director information, while we do not use this information effectively yet.

Our system did not work well in case of a hugely popular movie Titanic, and neither did Amazon's (they recommended Forrest Gump). This is probably because so many people watch the block buster movies that they get grouped together (almost everyone we know has watched both Titanic and Forrest Gump). However, our visualization system can outperform Amazon in case the user selects to see less a lesser known movie.

4 Future Work

There are several improvements that can be made to the system –

- One is to make use of other factors like actor, director and others.
- It would be really interesting to apply our algorithm to a system like the one of Jinni.com where every movie is couple with extensive tagging.
- Trying out other layouts that use space more efficiently while meeting our interaction needs would be important.
- We can also try encoding the state of the system in the URL so that users can start over at the same point, or email recommendations to their friends.

- Perform personalized recommendations (based on a particular user's profile). Also, save their preferences for different parameters like genre similarity, average rating cutoff etc.

ACKNOWLEDGEMENTS

The authors wish to thank Professor Jeff Heer for helping them brainstorm the idea and suggesting feedback techniques. They would also like to thank Chaitanya Sai (Boston University) for providing posters for all of the 17,700 movies.

REFERENCES

[1] Chris Anderson. "The Long Tail". In Wired magazine, 2004.
[2] "Netflix Movie Explorer" - http://gflix.appspot.com
[3] Daniel Lemire, Sean McGrath. "Implementing a Rating-Based Item-to-Item Recommender System in PHP/SQL"
[4] Daniel Lemire and Anna Maclachlan. "Slope one predictors for online rating-based collaborative filtering". In Proceedings of SIAM Data Mining (SDM'05), 2005.
[5] Bruce W. Herr, Weimao Ke, Elisha Hardy & Katy Börner. "Movies and Actors: Mapping the Internet Movie Database". In Conference Proceedings of 11th Annual Information Visualization International Conference (IV 2007).
[6] Daniel A Keim. "Information Visualization and Visual Data Mining". In IEEE Transactions on Visualization & Computer Graphics, Vol 7, Jan-Mar 2002.
[7] Wikipedia. "The Long Tail" - http://en.wikipedia.org/wiki/The_Long_Tail
[8] BellKor team, AT&T Labs Research. "Improved Neighborhood-based Collaborative Filtering".
[9] Ben Shneiderman. "Dynamic queries, starfield displays, and the path to Spotfire". Feb, 1999.
[10] "Collective Intelligence", Toby Segaran, O'Reilly, 2007
[11] Netflix Prize Forum
[12] Jinni – http://www.jinni.com
[13] IMDb – http://www.imdb.org
[14] Mufin - http://www.mufin.com/
[15] Last.fm – http://last.fm
[16] Musicovery – http://www.musicovery.com
[17] Pandora – http://www.pandora.com
[18] Filip Gruvstad, Nikhil Gupta. "Netflix & IMDb for search based recommendations". For CS345a – Data Mining, 2009 at Stanford University.