Mining Recommendations From The Web

Guy Shani
Microsoft Research
One Microsoft Way
WA, USA
guyshani@microsoft.com

Max Chickering
Microsoft Live Labs
One Microsoft Way
WA, USA
dmax@microsoft.com

Christopher Meek
Microsoft Research
One Microsoft Way
WA, USA
meek@microsoft.com

ABSTRACT

In this paper we study the challenges and evaluate the effectiveness of data collected from the web for recommendations. We provide experimental results, including a user study, showing that our methods produce good recommendations in realistic applications. We propose a new evaluation metric, that takes into account the difficulty of prediction. We show that the new metric aligns well with the results from a user study.

Categories and Subject Descriptors

H.4 [**Recommender Systems**]: Miscellaneous

General Terms

Algorithms

Keywords

Recommender Systems, Web Mining, Evaluation Metrics

1. INTRODUCTION

Recommender systems — systems that suggest items to users — are used on many web sites to help users find interesting items [19]. There are many different approaches to creating recommender systems [13], but the most common systems fall into two broad classes: *collaborative-filtering* systems [2] and *content-based* systems [15]. Content-based systems make recommendations based on an item similarity measure, based on item features. On the other hand, Collaborative-filtering systems use patterns in user ratings to make recommendations. Both types of recommender systems require significant data resources. In this paper, our focus is on collaborative-filtering systems in which one uses a *user-item rating matrix* to generate recommendations.

Many organizations, such as e-commerce web sites, collect private datasets which can be used to create user-item ratings matrices. This data is usually not available freely, or is available for a restricted use only. We use the term *private dataset* to denote such a dataset. However, similar data is often publicly available on the internet. For example, many bloggers publicly express their opinion about movies or electronic gadgets. These opinions are often available without usage restrictions. For such datasets we use the term *public datasets*.

In this paper, we compare the efficiency of recommendation systems built using public web datasets with a recommender system constructed using private datasets. We investigate two different methods for mining the web to build a collaborative-filtering recommender system. The first method uses a search engine to extract aggregate counts of item names using a search engine, and the second method extracts a full user-item rating matrix by mining item lists from a crawl of web pages. We show that in two real-world domains, each of these two methods leads to a recommendation system that is competitive with or better than a system constructed using a private user-item ratings matrix.

A separate contribution of this paper is a new evaluation metric for recommender systems. Our new metric addresses shortcomings of standard metrics for evaluating recommendations, such as Precision / Recall or the Exponential Decay Score, pointed out by other researchers. Such methods focus only on the predictive power of the algorithm, assuming all predictions are equally important. We argue that such an approach does not align well with recommendations that benefit the user. We derive a new evaluation metric that takes into account the difficulty of prediction.

We use the new metric to compare the recommendations that were generated from the public and private datasets. We show that the new metric computes scores that align well with a user study, where users rated lists of recommendations created from public and private datasets.

The paper is organized as follows. In Section 2, we describe the details of the collaborative-filtering system we use. In Section 3, we describe two internet-mining methods that can extract the data necessary to build our system. We discuss the shortcomings of standard evaluation metrics and suggest a new evaluation method in Section 4. In Section 5, we describe an experimental evaluation of our methods. We describe a user study demonstrating that the recommendation systems constructed using the two mining methods are competitive with the system constructed using a private dataset in the domain of movie recommendations. We show that standard evaluation metrics for recommendation systems are biased towards systems that recommend popular items, and show that our new metric yields scores that are

RecSys'08, October 23–25, 2008, Lausanne, Switzerland.

better correlated with the results of our user study. We overview related work in Section 6, and then suggest directions for future research (Section 7).

2. BACKGROUND

Collaborative-filtering algorithms attempt to find patterns within a dataset of user-item ratings. Such methods usually find users that provided similar ratings, and predict ratings based on those similar users. In many cases, collaborative-filtering methods detect interesting correlations between items, that are not obvious given the item properties.

A collaborative-filtering algorithm usually requires a user-item rating matrix. We use $V_{i,j}$ to denote the rating (vote) of user i for item j in the matrix. This ratings matrix typically contains any categorical values, but for this paper we consider only binary ratings (like/dislike).

There are many other well known collaborative-filtering algorithms. Breese et al. [2] suggest creating a decision tree for each item i in the dataset. The nodes of the tree are labeled by items and the edges are labeled by ratings. Thus, a path through the tree corresponds to a set of ratings over items. The leaves contain probability distributions over the ratings for the item i. Other methods for recommendations work directly with the user-item ratings matrix. Such methods take as input a set of ratings by a user, and find other users that have similar tastes. Then, the ratings for other items that these "similar" users liked are aggregated to create a list of recommendations[17]. Algorithms that directly work with the user-item rating matrix are known as *memory based* while algorithms that construct some model (e.g. a decision tree) are known as *model based*.

As our goal is to compare the effectiveness of public and private datasets, we choose to use a simple collaborative-filtering method. This allows us to focus our attention on the datasets rather than on the features of the recommendation algorithms.

One of the simplest ways to provide collaborative-filtering recommendations is the pairwise setting, where given a single item, we try to recommend other related items [18, 10]. A well known method to evaluate the relevance of two items i_1 and i_2 is the cosine score:

$$cosine(i_1, i_2) = \frac{\sum_j V_{i_1,j} V_{i_2,j}}{(\sum_j V_{i_1,j})(\sum_j V_{i_2,j})} \quad (1)$$

that measures the angle between the vectors of the item's ratings. In the case of binary ratings, the cosine score is computed using:

$$cosine(i_1, i_2) = \frac{count(i_1, i_2)}{count(i_1) count(i_2)} \quad (2)$$

where $count(i)$ is the number of users who liked item i and $count(i_1, i_2)$ is the number of users who liked both item i_1 and item i_2.

Given an item i_1, we recommend items i_2 with the largest $cosine(i_1, i_2)$. The cosine score has the attractive property that for a set of items with similar co-occurrence counts with i_1, the least popular items will tend to have higher values due to their low counts in the denominator. Recommending unpopular items often provides the most added benefit to the user because he is less likely to know about these items a-priori[5]. Given that we are only allowed a limited number of recommendations, it is important to try to maximize the value the user gains from the recommendations.

This item-to-item recommendation scheme is simple, but it has been employed successfully in large scale commercial recommender systems (e.g., [10]).

3. MINING THE WEB FOR RECOMMENDATIONS

The World Wide Web can be thought of as a huge database of documents for which a search engine[6] such as Microsoft Live Search[1] provides indexing to allow us to rapidly search for documents that contain specific keywords and phrases. In this section, we take this view of the internet and describe how to extract the data needed to build a recommendation system.

We use as our running example the problem of recommending movies, where a user has expressed an interest in a movie and we wish to present a valuable list of other interesting related movies. As we later show, our method can be applied to artist recommendations. We believe that the methods generalize to other domains including recommending books, gadgets — domains where many people publicly express their opinions on the web.

3.1 Data Mining Using a Search Engine

There are many pages on the web where people publicly describe their preferences over movies. Such pages often have lists that specify the movies the page author likes the most, or movies that the author recommends to other people. Pages that review movies contain in many cases references to other movies commenting on their relevance to the reviewed movie. Also, there are many recommendation systems for movies online, and these too post lists of movies that are related.

The cosine method, in the case of item-to-item recommendations and binary ratings, requires the number of users who liked the pair of items and the number of users who liked each item individually. We can use the number of pages that list an item (a pair of items) as a proxy for the number of users who liked the item (the pair of items). These aggregated counts are sufficient for producing item-to-item recommendations.

The simplest way to gather such information is by executing a query that includes the item name on a search engine such as Live Search. This will result in a set of pages where the item name appears. To compute the cosine score, we execute a query for each movie separately, and then execute a query for each pair of movies. Since we are only interested in the number of pages where the search term appears, we use the number of results returned by the search engine as the counts of the users required by the cosine score. We call the resulting method — executing queries on a search engine and using the counts of results to compute cosine scores — the *WebCount* method.

Arguably, many pages returned by searching for occurrences (co-occurrences) of movie names contain occurrences (co-occurrences) which are not relevant for the movie recommendation task. For instance, many pages containing the word "seven" do not refer to the movie of that name. Even when two names on a page refer to actual movies, the co-occurrence might be due to a shared property that is not useful for movie recommendation such as being released in the same month. One approach to mitigating

[1] www.live.com

these problems in the WebCount method is to use a more refined search query. In our experiments we append the following refinement to each query (i.e., movie name or pair of movie names): *("Movie Recommendations" OR "Recommended Movies" OR "Related Movies")*. At first this might seem quite restrictive, but issuing this refinement (with no movie name) as a query to the Live Search engine resulted in more than 1.4 million pages.

Using a refined query only partially addresses the relevance issue. In particular, count estimates for movies that are common English words or phrases are inaccurate. For example, since "big" is a commonly used English word, it appears in many movie descriptions. The co-occurrences counts for the movie "Big" will therefore be over estimated. As a result, our method is more accurate when the movie name is more unique (e.g. movie names with more words).

Another problem is the evaluation of movie names which are substrings of other movie names, such as "American Pie" and "American Pie 2". In this case, it is also difficult to estimate which of these movies was discussed in a specific web page.

Finally, the count of results returned by a search engine is inaccurate [11]. Search engines do not explicitly count the number of pages containing a search phrase, but rather estimate these counts [9] based on the frequency of the search phrase in the index and the estimated coverage of the web pages space by the search engine[1]. In fact, the fewer results the search engine reports, the more accurate the counts are. As we later discuss, we are more interested in these less popular items, so this aligns well with our goals.

We later discuss approaches for addressing these problems, but we claim that noisy datasets are a key feature for recommendations that rely on an aggregation of web pages. Therefore, recommender systems that mine the web for recommendations must be robust to such noise in the datasets.

3.2 Crawling Web Pages

A more sophisticated approach to gathering data is finding lists of movies that users like. Indeed, many people put in their websites lists of "the movies I like" or "my top 10 movies". Generally speaking, each time a list contains a number of movies, it is reasonable to assume that these movies are correlated. In this paper we assume that the occurrence of a movie on a page is a positive rating. This needn't be the case. For instance, a list of "the digital cameras you should avoid" or a list of "the worst cell-phones" contains lists of negative ratings. Such negative ratings are also beneficial for a recommender system, but we do not consider mining of such lists in this paper.

Finding lists of items, or 'entities' is the subject of current research[4, 21]. In this paper, we limited ourselves to a single well structured domain in which lists can be extracted easily using a deterministic algorithm. As a result, we can evaluate the benefit of extracting entities in a web crawl to building a recommendation system without confounding our results by difficulties in the entity extraction algorithm.

MySpace[2] is a web site that hosts many user pages. The website offers users a standard template they can use to specify, among other things, lists of movies and lists of musicians they like. Given this standard template, identifying the list of movies is relatively easy. However, this data is still somewhat noisy. First, people often misspell movie names. Second, people sometime use only a part of the official movie name, such as writing "The Two Towers" instead of "The Lord of the Rings: The Two Towers". We used some name reduction techniques in order to handle these problems. Many people also say that they like movie series such as the "Lord of the Rings Trilogy", or "The Godfather Series". When observing such statements, we expand the user profile with all the movies from the series. Also, many people make general statements such as "I like all Tom Hanks movies". These statements usually do not resemble any movie name and are easy to ignore.

Although we did not estimate the false-positive rate of the inexact item-name identification methods described above (e.g., by hand-labeling a large set of pages), we noticed no false positives on the pages we examined. That is, our method seems not to detect that a movie appeared in the list even though it did not. On the other hand about 30% of the strings in the lists are not mapped to movies and are therefore discarded.

Many pages in MySpace (about $1,800$ of the $44,000$ pages that we crawled), contain a suspiciously large number of movies. After inspecting many of these lists, we concluded that most were likely automatically generated (e.g. they had no misspellings and were alphabetically ordered). As a result, we consider any list containing more than 70 movies to be SPAM, and we remove them from the dataset.

We obtain a user-item rating matrix by treating each web page that was crawled as a user and extracting the list of items appearing on that page. We call this algorithm the *WebCrawl* method. In contrast to the WebCount method of the previous section, any standard collaborative-filtering algorithm can be applied to the resulting user-item matrix.

4. EVALUATING RECOMMENDATIONS

User studies, where users provide feedback about recommendations, are a particularly good way for evaluating recommender systems. Unfortunately, they are also difficult to conduct. In particular, the amount of effort, time, and cost required to evaluate the performance of an algorithm over a large scale dataset makes a complete study unrealistic. Most research therefore uses statistical evaluation methods that automatically evaluate the performance over the dataset. In this setting, the users are randomly split into a training and test sets. The algorithm is trained over the users from the training set and evaluated over the users in the test set.

4.0.1 Prediction Accuracy

The standard method for the evaluation of the accuracy of a recommender system is to check its ability to predict what users liked. Evaluation methods such as the Root Mean Standard Error (RMSE), Prediction/Recall, F1[17, 5], or Exponential Decay (ED) [2], attempt to measure the accuracy of the ratings that the algorithm predicts for a user-item pair (RMSE) or a user-list pair (Prediciton/Recall, F1, ED) from the test set.

In this paper we focus on using the Exponential Decay (ED) score, suggested by Breese et al. [2] to evaluate recommendations lists. In this setting, the recommender system takes as input a subset of a test user ratings, and proposes a list of recommendations, ordered in decreasing order of relevance. Then, we look at the rest of the ratings of the user. For each rated item, we look for its position in the list of recommendations. In the case of multi-valued rat-

[2]www.myspace.com

ings, the ED score for a user a and each item i in a list of recommendations ordered by decreasing relevance is:

$$R_{a,i} = \frac{max(v_{a,i} - d, 0)}{2^{(idx(i)-1)/(\alpha-1)}} \quad (3)$$

where $idx(i)$ is the position of item i in the recommendation list, $v_{a,i}$ is the actual rating of user a for item i, and d is the neutral vote. Thus, prediction accuracy is more important for items that are closer to the beginning of the list. The intuition behind this method is that when a list of recommendations is shown to the user, the user looks at the items by their order, until he finds sufficient recommendations. Therefore, the user is more likely to see a recommendation when it is closest to the beginning of the list. The half-life parameter (α) captures the index of the item that has a 0.5 probability of being observed.

In the case of binary ratings the above reduces to:

$$R_{a,i} = \frac{\delta(a,i)}{2^{(idx(i)-1)/(\alpha-1)}} \quad (4)$$

where $\delta(a,i)$ is 1 if user a has liked item i and 0 otherwise. The aggregated score for a user a is:

$$R_a = \sum_{i \in like(a)} R_{a,i} \quad (5)$$

where $like(a)$ is the holdout set of items that the test user a has liked.

The overall score over the entire dataset is:

$$ED = \frac{\sum_a R_a}{\sum_a R_a^{max}} \quad (6)$$

where R_a^{max} is the best possible score for user a.

The standard evaluation methods described above, including the ED score, fail to capture accurately the concept of the usefulness of a recommendation [12]. For example, recommending a very popular item to a user may be good in terms of a standard accuracy score, but because the user is likely to already be aware of such an item, it may be more useful to recommend a less popular item [5]. Many commercial recommender systems (such as ones using cosine similarity [10]) have a built-in bias against recommending popular items, and as a result these systems will be at a disadvantage in terms of standard evaluation methods. Below we suggest modifications to the standard evaluation methods that take into account item popularity.

4.0.2 Prediction Difficulty

As we later show (Section 5), the standard ED scores do not align well with the results that were obtained from a user study we have conducted.

Specifically, we identify two problems with the standard ED score. First, a static recommendation list based on popularity can get a very high score. Second, random recommendations can also produce a relatively high score, while one would expect random recommendations to be bad.

We suggest that these two anomalies result because some predictions are more difficult than others. Predicting that a user will like a popular item is relatively easy because popular items, by definition, are preferred by most users. Successfully predicting whether a user will like an unpopular item, however, relies on accurately computing the correlations between items.

Also, it appears that the entire lists for some users are easier to predict than others. Specifically, some users like almost any movie that they see. Since these users like almost every movie, random recommendations have a good predictive power.

4.0.3 Modified Exponential Decay Score

In view of the above two problems, we modified the ED score to take prediction difficulty into consideration. While we discuss below only the modifications to the ED score, the changes we propose can be applied to other evaluation methods such as Precision/Recall and RMSE.

First, we want to augment the method to take the popularity of the item into account. As such, we modify the above score by emphasizing items that are less popular and therefore more difficult to predict. Breese et al. suggest to emphasize less popular items when computing user similarity by:

$$f(i) = log(\frac{n}{n_i}) \quad (7)$$

where n is the number of users and n_i is the number of users who liked item i. We take a similar approach by converting Equation 4 into:

$$\tilde{R}_{a,i} = f(i) R_{a,i} \quad (8)$$

This modification gives a higher score to successful predictions of items with low n_i — less popular items. Then we modify the score for a user (Equation 5) into:

$$\tilde{R}_a = \sum_{i \in like(a)} \tilde{R}_{a,i} \quad (9)$$

R_a^{max} also needs to be redefined to $\tilde{R}_a^{max}$ given the different optimal order of items introduced by the $f(i)$ scores.

As explained above, the second problem we noticed was that some users were extremely easy to predict. We therefore propose to emphasize users who like a small number of movies:

$$g(a) = log(\frac{N}{N_a}) \quad (10)$$

where N is the number of items, and N_a is the number of items that user a liked.

Finally, the score for the entire test set is computed as follows:

$$MED = \frac{\sum_a g(a)\tilde{R}_a}{\sum_a g(a)\tilde{R}_a^{max}} \quad (11)$$

5. EXPERIMENTAL RESULTS

A natural way to evaluate our WebCount and WebCrawl methods for using the web to build recommendation system datasets is to compare the recommendations that they generate to recommendations generated from a traditional private dataset of collected ratings. We hence used the NetFlix dataset[3] that contains movie ratings (one to five stars) from 480,000 users on 17,000 movies. We used the 400 most popular movies, for which each have at least 49,000 ratings. We converted the ratings to binary form by treating high ratings (4 or 5 stars) as positive votes and all others as negative votes. We make this conversion because our WebCount and WebCrawl methods currently capture only

[3]www.netflixprize.com

positive votes. We later discuss how the WebCrawl method can be augmented to provide finer ratings.

We used the two methods — counting web pages returned by movie queries over the Live Search engine (the WebCount method), and crawling 44,000 MySpace profile pages (the WebCrawl method) — to gather preferences datasets. We computed recommendation lists for all three datasets (the two datasets gathered from the web and the private ratings dataset from NetFlix) using the cosine score. Thus, we focus our attention to differences in the quality of the datasets, not the features of the recommendation algorithm.

5.1 User Study

The ultimate goal of many (if not all) recommendation systems is to bring value to a user. A particularly good way to evaluate how well a system is achieving this goal is to ask users whether the recommendations are useful [5]. As a result, we conducted the following user study. Each user was asked to rate three lists of recommendations for a single input movie, based on their relevance to the movie. The ratings were either "Good" (most movies in the list are relevant), "Ok" (some movies are relevant), or "Bad" (most movies are not relevant). Each list had 5 movies in it, sorted by decreasing scores, and the lists was presented in a random order.

To help users recognize the movies, we displayed movies details below the candidate lists. These details included the plot, the cast, and the director. If the user did not know the input movie, he could skip rating the recommendation lists and move to the next input movie. All users were required to rate at least 10 sets of recommendation lists, but some users rated more. We had 98 users participating in the user study, voting on 2054 sets of recommendation lists.

We showed users recommendations using cosine scores computed from the NetFlix dataset, the WebCount method, and the WebCrawl method. Also, a small number of users were given random recommendations in order to provide a baseline for comparison. Table 1 contains the results of the user study, showing the relative number of votes for each recommendation method. Results were verified to be significant using a Chi-square test on the contingency tables (specifically, the pairwise p-value for the WebCrawl and the NetFlix recommendations was 0.0021, and all other results were far more significant).

Rating	Random	WebCount	WebCrawl	NetFlix
Good	0.11	0.27	0.42	0.37
Ok	0.19	0.44	0.42	0.46
Bad	0.70	0.29	0.16	0.17

Table 1: Distributions of ratings of recommendation lists from the various methods during the user study.

As expected, the random recommendations provide the worst results. Still, 0.3 of the lists were deemed to be at least reasonable. We attribute this to two factors. First, our lists contained only 400 movies, which were selected based on their popularity. As such, the random lists would often contain a popular movie from the same genre as the input movie which users thought to be a reasonable recommendation. If we would have used the entire movie set as input, we would expect this effect to be reduced. Also, we speculate that people sometime like to see unexpected recommendations, and credit the recommender system for proposing movies that are not obviously related.

The remaining methods provided comparable results. Specifically, the WebCrawl method provided the best recommendations, with the NetFlix ratings close behind. This suggests that in the movie domain, the user-item matrix that we get from crawling MySpace has similar or better quality to the one that we get from the NetFlix ratings.

The WebCount method that used queries from Live Search performs somewhat worse. This is understandable, since many movies have names that are very generic, such as "Big" or "Seven". These movies are very difficult to predict using the web search. Also, search engines provide in their results only rough estimates of the number of pages that contain a key word, especially for keywords, such as popular movies, that have a high frequency. That being said, the results from the WebCount recommendations are surprisingly good, given the very simple method we have used. Over 70% of the recommendations list that were computed looked reasonable to people.

5.2 Automatic Evaluation

In this section we study the new metric we have suggested for evaluating recommendations. As explained above (Section 4), the standard ED score is biased towards popular items and towards users who like many items. We demonstrate these two problems below, and show how our modifications handle these issues. We report results over two domains — the NetFlix movie ratings, and the MSN music dataset that contains ratings over performing artists.

5.2.1 Results over the NetFlix Dataset

We split the users from the NetFlix dataset into a training set containing 90% of the 480,000 users and a test set containing the remaining 10% (about 47,000) of the users. We trained the cosine scores for the NetFlix recommendations only on the training set. Then, we evaluated all the methods over the test set. Clearly, this method of evaluation is biased towards the NetFlix recommendations. It is quite likely that people's tastes have changed since the NetFlix dataset was published in 2004, and that therefore the results mined from the web are more appropriate for an updated recommender system.

To illustrate the difference between the data sets, the most rated movie in the NetFlix dataset is "Miss Congeniality", which is only the 256th on the WebCrawl popularity. The most popular movie among the MySpace users, "Fight Club", was only 86th on the NetFlix popularity list.

We use the "Given one" method of Breese et al.[2] to create the prediction tasks for our evaluations. In particular, for each user in the test set with liked items $< i_1, \ldots, i_m >$, we consider each item i_j from the list as an input, and evaluate the systems by how well they predict the remaining items.

We have evaluated five prediction algorithms; we computed random recommendations, and a fixed list of items ordered by popularity (computed from the NetFlix dataset). Next, we used the NetFlix ratings from the test set to compute cosine item-to-item scores. We also used the two new methods we propose — the WebCount method, that uses query results count from Live Search, and the WebCrawl method that crawled pages from MySpace to obtain a user-item matrix. We computed cosine scores for these two methods as explained in Sections 3.1 and 3.2.

Metric	Random	Popularity	WebCount	WebCrawl	NetFlix
ED	0.169	0.299	0.229	0.257	0.228
ED, $g(a)$	0.078	0.214	0.155	0.181	0.159
ED, $f(i)$	0.137	0.093	0.182	0.200	0.224
MED	0.048	0.079	0.131	0.148	0.165

Table 2: Evaluating the modifications to the ED score over the NetFlix train-test experiment.

The first row of Table 2 demonstrates the two problems we have identified with the ED score; First, the static recommendation list based on popularity is getting the highest score. Second, the random recommendations score is very close to the other methods, while the user study suggests that the difference should be much larger. We attribute the high scores of the random recommendations to "easy" users who like most items. Indeed, among the 400 most popular movies, about 1700 users in our test set rated over 200 movies with at least 4 stars.

To show how the changes we propose affect the ED score we computed four different scores over the NetFlix train-test set. We first compute the original ED score, then evaluate separately each of the changes we proposed: $g(a)$ and $f(i)$, and finally, we also report our Modified Exponential Decay (MED) score that contains both changes. Table 2 shows the results for evaluating the performance of the prediction methods using the different evaluation methods.

As explained above, the original ED score does not capture well user preferences obtained from the user study. The modifications that we made fix these shortcomings. Adding the $f(i)$ score reduces the evaluated effectiveness of popular recommendations, and adding the $g(a)$ score reduces the evaluated score of the random recommendations. The final MED score we propose seems to better capture the results from the user study. The NetFlix ratings superiority likely results from the NetFlix bias as explained above.

5.2.2 Results over the MSN Music Dataset

We also experimented with a second dataset. The MSN Music website[4] allows users to rate performing artists (singers and bands), and we used these ratings as a second domain for evaluating our approaches for recommendations and for evaluation. The dataset consists of ratings on the scale of 1 to 5 for about 14,000 artists, given by slightly more than 120,000 users. The dataset was gathered in 2004.

We selected from the MSN Music dataset the 200 most popular artists. We converted the ratings to binary values in the same manner as we did for the NetFlix dataset. We split the dataset into a training set consisting of 90% of the 120,000 users and a test set consisting the remaining 10% (12,000) users.

We executed the WebCount and WebCrawl methods in the same way as in the movies domain. We used again the MySpace pages since MySpace users can also list the artists they like. We also generated random recommendation lists and a recommendation list sorted by decreasing popularity. Finally, we used the ratings in the MSN Music dataset to compute cosine scores.

Table 3 shows the results over the MSN Music dataset. Again, we can see how the original ED score gives the highest score to the fixed popularity recommendations. In this domain the random recommendations did not do that well. This is because there are not many "easy" users in this dataset, since users have rated fewer items than in the NetFlix dataset. The MED metric assigns lower scores to random and popularity while preserving the results for the rest of the methods that use the cosine score.

[4]music.msn.com

6. RELATED WORK

There are several previous papers that utilized the web obtaining ratings where no publicly available user-item ratings datasets exist. For example, the Book-Crossing dataset [22] contains book ratings collected from a social network dedicated to discussions of books[5]. Users of this network provide lists of books they like. We expect that this dataset is less noisy than most publicly available datasets, such as the MySpace dataset that we crawled. The primary difference is that users of Book-Crossing do not enter item names via free text, removing the problem of correctly matching identical items.

Similar to our work, Frankowski et al. [3] collect user ratings from the web. Unlike our work, their goal is to show that these ratings can be used to identify users. This work provides evidence to the effectiveness of ratings collected from the web. Narayanan and Shmatikov [14] extended this work to the movies domain and the NetFlix dataset, reporting noise in the data gathered by crawling the web.

Li et al. [8] suggest an evaluation metric where the 3 least popular items in the basket are predicted based on the more popular items. The motivation for this approach is the higher value of recommending less popular items, and therefore the higher importance for accurate predictions over these items. Our method can be viewed as a generalization of this idea.

7. FUTURE RESEARCH

In this section we discuss potential extensions to our research. We explain how the WebCount and the WebCrawl methods can be extended to more sophisticated recommendation algorithms by learning richer models and by collecting additional data. We will also discuss how other recommendation systems can be constructed from web data.

7.1 Other Collaborative-Filtering Algorithms

In this paper we used a simple item-to-item collaborative-filtering algorithm: the cosine score. We selected this algorithm because its simplicity allows us to focus on the datasets rather than the choice of the recommendation algorithm.

The WebCount method is particularly well-suited for the cosine score. It can, however, be used as the basis of more sophisticated collaborative-filtering algorithms. For instance, we can use the WebCount method to construct decision trees. Decision trees can predict the probability of a user

[5]www.bookcrossing.com

Metric	Random	Popularity	WebCount	WebCrawl	MSN Music
ED	0.085	0.246	0.211	0.238	0.217
ED, $g(a)$	0.057	0.208	0.188	0.218	0.190
ED, $f(i)$	0.082	0.036	0.214	0.214	0.278
MED	0.021	0.031	0.197	0.200	0.252

Table 3: Evaluating the modifications to the ED score over the MSN Music train-test experiment.

rating over a specific item given a set of items he likes [2]. Such a decision tree can have nodes labeled by items, and edges labeled by ratings. A path in the tree specifies a set of ratings over items. The leaves provide a probability distribution over possible ratings.

We can use the counts that were computed in order to decide which items should be placed in the tree nodes. For example, when constructing a tree for item i we can place in the root the item $i' = \max_j cosine(i, j)$. Then, we can execute new query that contains the items i and i', and a new query that contains i and excludes i', to build the two subtrees (when i' is liked and when i' is disliked). Finally, when a sufficiently small number of results is returned, we can run query that includes i and the rest of the search path, and one query that excludes i and includes the rest of the search path to compute a probability estimation. This algorithm is an interesting case of constructing decision trees where we can only access the data through aggregation queries, rather than allowing us to directly observe the data.

The WebCrawl method has no obvious limitations on the type of collaborative-filtering algorithm we use because it gathers a standrad user-item rating matrix; this matrix is the standard input data used to train most of the collaborative-filtering algorithms from the literature.

7.2 Cold Start

As collaborative-filtering methods require previous user ratings in order to provide useful recommendations, such systems must gather sufficient ratings before recommendations for new items can be produced. This is usually known as the *cold start* problem [20].

It is possible that our methods can reduce the cold start problem, by providing recommendations for some items prior to their release, or shortly afterwards.

For example, many people post information about movies prior to their release. Many web pages, either formal or fan blogs are already posting comments about a movie prior to its release. For instance, a query for "Indiana Jones and the Kingdom of the Crystal Skull", that was issued prior to the release of the movie, returned more than 3.8 million results. In addition ratings are rapidly available on the web; Many people express their opinions online shortly after experiencing an item. As such, a day after a new movie release, the web already contains the opinions of many people who have seen it.

7.3 Collecting Better Data

Mining the MySpace web pages was a relatively easy task due to the standard layout of the profile web page. In general, however, mining item references from text is a difficult task.

When mining web pages, it is possible to extract *entities*, for example, to identify whether the web page uses "big" as an adjective or as a movie name. We can also enrich the dataset by considering the *sentiment* that was expressed for the item [4, 21]. For example, we can differentiate between movies that the user "likes" and the movies that she "loves" or "hates". This would allow us to create datasets that contain more than binary ratings. Extracting sentiment is a topic of on-going research and is already used in some commercial applications. For example, if we search for a specific digital camera on Live Search, the results show an aggregation of opinions from online reviews.

Another possibility is to try and capture the notion of *closeness*. For example, if two movies are mentioned relatively close in the web page then they are more likely to be related. As a special case of this suggestion we might want to identify lists in a web page. Since many movie recommendations in web pages are arranged in a list, identifying these lists will give us access to more robust data.

However, methods that attempt to mine the web page are bound to be slower than methods that use the search engine indexing directly. It is possible that these methods should only be used when the search engine produces noisy results, as measured by some significance measurement, or when the item is of high importance and we want high confidence in our recommendations.

The notion of a "user" should also be further explored. In this paper, we assume that each web page is the equivalent of a user. In practice, bloggers, or movie critics, use a multitude of pages to express their opinions. If we properly identify the sets of pages that correspond to a single user, we can capture deeper correlations between items.

7.4 Other Recommendation Approaches

Even though this paper focuses on mining the web to get collaborative-filtering data, we can also mine the web to get data that is suitable for other recommendation types.

Mining the web to find the actors that appeared in a movie, or the writer of a book is a relatively easy task, since most search engines are designed exactly for that — providing a set of results to answer a specific query. As such, gathering a dataset of item attributes might prove to be a simple and efficient method for constructing a dataset for content-based recommendations

Another type of recommendation engine relies on the opinions of experts. Indeed, reviews for many types of items are readily available on the web. We can use data-mining techniques to extract from such reviews positive or negative sentiment, and in some cases we can even understand that the review specifies that item i_1 should be used instead of the reviewed item i_2 [16]. For example, a review of a specific digital camera might say that the product is not good, and that another digital camera that has similar attributes should be preferred [7].

Another expert-based system might use existing recommendation engines as experts. Many recommender engines are publicly available (especially in the movies domain). We

could explicitly limit our search to pages that use such recommender engines, and then aggregate the results into a single list of recommendations. This is similar to the approach used by some meta search engines, which only aggregate results from a set of search engines.

8. CONCLUSIONS

In this paper we developed two methods for generating public datasets that utilize the web. We demonstrated in a user study how recommendation systems built using our techniques produce good results in practice compared to private datasets.

We modified a standard evaluation metric—the exponential decay score—to better evaluate the performance of a recommendation algorithm, and we experimented with the new evaluation metric over two domains. The results from the new method align well with the user preferences that were obtained from our user study.

We identified the advantages and disadvantages of the data that we collected and we suggested some ways in which our methods can be extended. Our results are important as they allow researchers to create recommendation systems without the use of private datasets. This opens up research opportunities in domains where such datasets are difficult to obtain.

9. REFERENCES

[1] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. *Comput. Netw. ISDN Syst.*, 30(1-7):379–388, 1998.

[2] J Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI '98*, pages 43–52, 1998.

[3] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl. You are what you say: Privacy risks of public mentions. In *Special Interest Group on Information Retrieval (SIGIR)*, 2006.

[4] M.A. Gamon, S. Aue, S. Corston-Oliver, and E. Ringger. Pulse: Mining customer opinions from free text. In *Lecture Notes in Computer Science*, volume 3646, pages 121–132. Springer Verlag, 2005.

[5] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.

[6] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[7] D. Lee, O. Jeong, and S. Lee. Opinion mining of customer feedback data on the web. In *ICUIMC '08*, pages 230–235, 2008.

[8] M. Li, B. Dias, W. El-Deredy, and P. J. G. Lisboa. A probabilistic model for item-based recommender systems. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 129–132, 2007.

[9] P. Li and K. W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics*, 33(3):305–354, 2007.

[10] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. 7:76–80, 2003.

[11] Y. Matsuo, H. Tomobe, and T. Nishimura. Robust estimation of google counts for social network extraction. In *AAAI*, pages 1395–1401, 2007.

[12] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06*, pages 1097–1101, 2006.

[13] M. Montaner, B. López, and J. L. De La Rosa. A taxonomy of recommender agents on theinternet. *Artificial Intelligence Review*, 19(4):285–330, 2003.

[14] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets (how to break anonymity of the netflix prize dataset). In *29th IEEE Symposium on Security and Privacy*, pages 111–125, 2008.

[15] M. Pazzani and D. Billsus. Content-based recommendation systems. *The Adaptive Web*, 5:325–341, 2007.

[16] A. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *HLT '05*, pages 339–346, 2005.

[17] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Application of dimensionality reduction in recommender system – a case study. Technical report.

[18] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *World Wide Web*, pages 285–295, 2001.

[19] J. B. Schafer, J. A. Konstan, and J. Riedl. Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–166, 1999.

[20] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Generative models for cold-start recommendations. In *6th SIGIR Workshop on Recommender Systems*, 2001.

[21] P. Viola and M. Narasimhand. Learning to extract information from semi-structured text using a discriminative context free grammar.

[22] C. N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW '05*, pages 22–32, 2005.