CROC: A New Evaluation Criterion for Recommender Systems [1]

Andrew I. Schein[2], Alexandrin Popescul and Lyle H. Ungar

University of Pennsylvania

Dept. of Computer & Information Science

Levine Hall

3330 Walnut Street

Philadelphia, PA 19104-6389

U.S.A.

{ais,popescul,ungar}@cis.upenn.edu

David M. Pennock[3]

Overture Services, Inc.

74 N. Pasadena Ave, 3rd floor

Pasadena, CA 91103

U.S.A.

david.pennock@overture.com

[1]This work to appear shortly in Eletronic Commerce Research. This copy is a draft version. Portions of this work appeared earlier

[2]To whom correspondence should be addressed. Phone: 215.735.9167 Fax: 215.898.0587

[3]This work conducted while at NEC Laboratories America, Princeton, NJ.

Abstract

Evaluation of a recommender system algorithm is a challenging task due to the many possible scenarios in which such systems may be deployed. We have designed a new performance plot called the CROC curve with an associated statistic: the area under the curve. Our CROC curve supplements the widely used ROC curve in recommender system evaluation by discovering performance characteristics that standard ROC evaluation often ignores. Empirical studies on two domains and including several recommender system algorithms demonstrate that combining ROC and CROC curves in evaluation can lead to a more informed characterization of performance than using either curve alone.

Keywords: Recommender systems, collaborative filtering, performance evaluation

1 Introduction

Recommender systems (RS) suggest items of interest to users based on their explicit and implicit preferences, the preferences of other users, and user and item attributes. Methods for conveying recommendations to users are virtually unlimited. Recommendations can be sent through Email solicitation (push mechanism), through explicit user request (pull mechanism) or as a suggestion in a web page the user is viewing (push mechanism). In many applications the number of recommendations made to each user must be limited to a handful in order to prevent information fatigue among the user base. However, traditional evaluation metrics for recommender systems–including ROC curves and mean absolute error (MAE) statistics–do not take this phenomenon into account, and may reward recommender systems that skew the number of recommendations to the users who rate most frequently or most highly.

In this paper we will explore a new performance plot, called the CROC curve, which evaluates system performance in situations where each user receives the same number of recommendations. While traditional ROC curves uncover the raw performance of an RS algorithm, our CROC curve measures the ability of the RS to provide reasonable recommendations across a wide user base; thus we find significant advantages in using both CROC and ROC metrics together. ROC curves are formed by creating one large ranked list of recommendations (p_i, m_j), indicating user i likes/rates/purchases item j. Optimizing performance on an ROC curve can mean dramatically skewing recommendations to the most active users or the users who rate more highly on average. In contrast, the CROC curve creates one ranked list for each user, interleaving recommendations evenly among users. In this way recommendations are constrained in the CROC evaluation so that each user receives the same number of recommendations, though different users will receive a separate set of recommendations according to their predicted preferences.

We demonstrate our two-metric evaluation strategy by recommending web pages and movies: predicting both explicit ratings in the form of a rating in the scale $1-5$ in addition to implicit preference information such as a web site visitation. In order to ascertain the value of the CROC curve, we evaluate three machine learning algorithms while including baseline popularity-ranked recommendations. We evaluate in settings where we must recommend items nobody has rated before (*i.e.* recommending from a *cold-start*) as well as in the *hot-start* setting. In all of these situations we pinpoint advantages and disadvantages of the different recommender system algorithms tested, leading to some surprising conclusions about RS performance. Our insights into algorithm performance demonstrate the advantage of combining ROC and CROC metrics in evaluation, compared to current practices that ignore the consequences of recommendation constraints in performance summarization.

2 Background and Related Work

To date, most comparisons among algorithms have been empirical or qualitative in nature [13, 27], though some worst-case performance bounds have been derived [8, 20], some general principles advocated [8], and some fundamental limitations explicated [21]. Techniques suggested in evaluating recommender system performance include mean absolute error, receiver operator characteristic (ROC) curves, ranked list techniques [3, 13, 31] and variants of precision/recall statistics [27]. Our CROC curve defined and developed in this paper can be viewed as a ROC curve created through constraints on the recommendations. Sarwar *et al.* [27] define analogous constraints on precision/recall statistics. Our contribution adds to the work of Sarwar *et al.* by demonstrating *why* traditional metrics such as ROC curves and precision/recall must be adjusted for the recommender system domain; we show that ROC curves and CROC curves frequently have conflicting statements about which recommender systems are best. Breese *et al.* [3] propose another method to combine performance information about individual users. Their R metric averages performance over the entire user base using an exponential decay in the user's attention as the user peruses further and further down a list of recommendations. Unlike the R rank metric, the ROC and CROC methods employed in our evaluation have no parameters that must be set prior to evaluation, such as a decay rate.

Recommender system algorithms can be classified into collaborative filtering, content filtering, or methods that combine both forms of information. Pure *collaborative filtering* methods [3, 14, 17, 25, 34] base their recommendations on community preferences (*e.g.*, user ratings and purchase histories), ignoring user and item attributes (*e.g.*, demographics and product descriptions). On the other hand, pure *content-based filtering* or *information filtering* methods [19, 26] typically match query words or other user data with item attribute information, ignoring data from other users. Several hybrid algorithms combine both techniques [1, 4, 6, 10, 23, 32]. Though "content" usually refers to descriptive words associated with an item, we use the term more generally to refer to any form of item attribute information including, for example, the list of actors in a movie.

Early recommender systems were pure collaborative filters that computed pairwise similarities among users and recommended items according to a similarity-weighted average [24, 34]. Breese *et al.* [3] refer to this class of algorithms as *memory-based* algorithms. Subsequent authors employed a variety of techniques for collaborative filtering, including hard-clustering users into classes [3], simultaneously hard-clustering users and items [36], soft-clustering users and items [16, 23], singular value decomposition [28], inferring item-item similarities [29], probabilistic modeling [3, 6, 12, 22, 23, 31, 32], machine learning [1, 2, 20], and list-ranking [5, 8, 21]. More recently, authors have turned toward designing hybrid recommender systems that combine both collaborative and content information in various ways [1, 4, 6, 10, 23, 31, 32].

One difficult, though common, problem for a recommender system is the *cold-start* problem, where recommendations are required for items that no one (in our data set) has yet rated.[1] Cold-start recommending is particularly hard since pure collaborative filtering approaches fail: new users have no history, yet history is the sole training information of a pure collaborative filtering system. In this work we will benchmark recommender systems in hot-start settings in addition to the new-item cold-start setting. Benchmarking in the cold-start setting requires slight changes in the experimental design, but exposes convenient theoretical properties of the CROC metric.

In the sections that follow we define the ROC and CROC curves for recommender system evaluation, and then describe the algorithms used in the empirical evaluation and the experimental setup. Results and discussion follow.

3 Evaluation Metrics

In this section we define constrained and unconstrained modes of allocating recommendations and explore their consequences for evaluation of algorithms. Formally, we define a recommendation as a pair (p, m) interpreted to mean "recommend m to p." We can imagine creating a ranked list out of all possible recommendations (a $|P| \cdot |M|$ sized list), and pulling off the top n to actually recommend. There are no constraints introduced as of yet; theoretically a recommender system may choose to recommend all items to a particular user before recommending items to any other user. In fact, we will find in practice that certain algorithms will recommend more often to some users than to others if unconstrained.

Alternatively, we may wish to know the performance of our system under circumstances where we recommend the same number of items to each user. We refer to these two models of recommender system usage as unconstrained and constrained recommending respectively, and develop evaluation metrics for each case. In the constrained mode of recommending, we create $|P|$ separate ranked lists of recommendations: one list per user. To make n total constrained recommendations we pull $|P|/n$ recommendations off of the top of each of the $|P|$ lists. The following sections describe metrics for both unconstrained and constrained recommending.

[1] The phrase *cold start* has also been used to describe the situation when almost nothing is known about customer preferences [11] (*e.g.*, a start-up company has little or no purchase history for its customers). The problem of making recommendations for new users can also be thought of as a cold-start problem, falling within the framework of pure information filtering or information retrieval [26].

3.1 Unconstrained Evaluation: The ROC Curve

For unconstrained recommending, we employ the receiver operator characteristic (ROC) curve [35] advocated for recommender system evaluation by Herlocker *et al.* [13]. ROC curves are suited for tracking performance in binary classification tasks while varying a classification threshold. RS applications are cast as binary classification when we classify a person/movie pair as like/does-not-like (rating prediction) or purchased/did-not-purchase (implicit rating prediction). Instead of varying a threshold, we vary the number of top (p, m) tuples in a ranked list that we use as recommendations. ROC curves plot the false-alarm rate on the x-axis against the hit rate[2] on the y-axis where we have:

$$\begin{aligned} \text{hit rate} &= \frac{\text{tp}}{\text{tp} + \text{fn}} \\ \text{false alarm rate} &= \frac{\text{fp}}{\text{fp} + \text{tn}}. \end{aligned}$$

The number of true positives, denoted tp, are the number of positive examples we correctly identify as such. The number of false positives, denoted fp, are the number of negatives that we miss-classify as positive. The definitions for true negatives (tn) and false negatives (fn) are analogous. ROC curves have the convenient property that random recommendation is characterized by an expected plot of a forty-five degree line from the lower-left to upper-right corners of the graph, and perfect performance is characterized by a horizontal line one unit above the origin. A useful statistic for summarizing the curve is the area under the curve. Here, perfect performance is characterized by area 1.0 and random performance with expected area 0.5. ROC curves are constructed in the following manner:

1. `Order the predictions` $\mathbf{pred}(p_i, m_j)$ `in a list by magnitude, imposing an ordering:` $(p, m)_k$.
2. `Pick` k'`, calculate hit/false-alarm rates caused by predicting the top` k' $(p, m)_k$ `in the list, and plot the point.`

By selecting different k' (*e.g.* incrementing k' by a fixed amount) we draw a curve on the graph.

3.2 Constrained Evaluation: The CROC Curve

For the constrained mode of recommending we introduce the CROC, or Customer ROC curve. The curve plots hit rates and false-alarm rates as defined in the standard ROC curve setting, however a constraint is imposed so that each user gets the same number of recommendations. This is achieved by creating one separate ranked list for each user in the data set and plotting points along the curve by recommending the

[2] *false alarm* is a synonym for false positive while *hit* is a synonym for a true positive.

top k items on each list. Note that the $|P|$ ranked lists can have independent orderings of items, so users are potentially recommended separate sets of items.

In most forms of recommender system evaluation, users have different numbers of observations in the testing set; it is common practice to remove training set data from consideration in testing. In this case the $|P|$ lists have different lengths. There are exceptions to this rule, for instance when performing certain types of cold-start evaluation where none of the test set events occur in training. Also, there may be alternative scenarios where a data set contains information about repeated purchases of an item, and therefore it should be permitted to recommend all person/item pairs in testing. However, the evaluations in this paper assume that we do not want to recommend an item if we know that a user has previously seen, purchased or rated that item.

Let $n(p)$ be the number of items available to user p in the test set observations. According to the CROC curve evaluation, the recommendation problem is essentially to guess which k of the $n(p)$ items *each* user p will like/purchase. When $n(p)$ varies by p (*i.e.* the $|P|$ lists have different lengths), we make at most $n(p)$ recommendations for user p. The algorithm for generating the CROC curve is:

1. For each person p_i, order the predictions $\mathbf{pred}(p_i, m_j)$ in a list by magnitude imposing an ordering: $(m)_k$.
2. Pick k', calculate hit/false-alarm rates caused by recommending the top predicted $\min(k', n(p))$ movies to each person p from their own list and plot the hit rate against the false-alarm rate.

By varying k' we generate the different points along the curve.

There are some important differences between the CROC curve and the ROC curve. For example the perfect recommender in a CROC curve is not necessarily a horizontal line one unit above the x-axis. To see why, imagine evaluating an omniscient (*i.e.* perfect) recommender on a data set with three people each with six observations: person a likes only four out of six movies, person b likes only two out of six movies, and person c likes all six movies. When we recommend four movies to each person, we end up with two false-positives from person b, lowering the area of the curve. However, for any particular data set, we can plot the curve and calculate the area of the omniscient recommender in order to facilitate comparison.

Random performance in the CROC curve is another important behavior to understand. A natural question to ask is whether a random recommender gives an expected plot consisting of a forty-five degree

line as in the ROC curve. When each user has the same number of test set observations this quality holds for the CROC plot, as we will prove. An experiment where k items are sampled without replacement from a population of size N with s successes in the population follows the *hypergeometric distribution* [7]. Consequently, the expected number of successes is sk/N. Assume for the moment that each user has the same number of observations (items) for recommendation in a test set: $n(p) = N$. Further, we denote the number of items each user has rated highly (success events) in the entire test set pool as $s(p)$. In what follows, $|P|$ is the total number of users we make recommendations for, and S is the total number of successes, summed over all users.

Theorem 1 (CROC Random Recommendation) *The expected CROC curve of a random recommender on a test set where each user has the same number of observations is a forty-five degree line.*

Proof: *Making k recommendations to user p leads to expected number of hits $ks(p)/N$. Summing over all users we obtain expected number of hits:*

$$\frac{k}{N}\sum_{p} s(p) = \frac{kS}{N} \tag{1}$$

leading to expected hit rate $\frac{k}{N}$. Making k recommendations to user p leads to expected number of false alarms $k(N - s(p))/N$. Summing over all users we obtain:

$$\frac{k}{N}\sum_{p} [N - s(p)] = \frac{k(|P|N - S)}{N} \tag{2}$$

leading to expected false-alarm rate $\frac{k}{N}$. The expected CROC point coordinates after k random reccomendations is therefore $(\frac{k}{N}, \frac{k}{N})$, leading to a forty-five degree line. □

The theorem applies for cold-start implicit rating and cold-start explicit rating prediction, as defined in Section 5. Section 6.2 provides opportunity to confirm that the forty-five degree line described in the theorem holds up well in practice. Our other evaluations do not invoke this theorem due to having different test set observation counts for different users. In cases where the number of observations varies for each user, we can provide no theoretical guarantee on the random recommender at the moment. With a little thought it becomes clear that in circumstances where having fewer observations is correlated with having a greater number of positive outcomes, the expected curve will be above the forty-five degree line. The lack of theoretical guarantee has not been an impediment to other performance measures such as mean absolute error or precision/recall curves. In cases where we can not invoke the theorem, we can simulate random recommendation and plot the CROC curve to provide this baseline.

3.3 Interpreting ROC and CROC Curves

On both ROC and CROC curves the portion of the curve of special interest in evaluation is the left-hand side of the curve. The left-hand side corresponds to making k recommendations where k is small in comparison to the total number of recommendations we could make. In most applications we have only the opportunity to recommend a few items before a user gets tired of looking at the list of suggestions, and so performance on the left-hand side of the curve is critical. In our present experiments we plot the entire curve and calculate the area under the entire curve. Alternatively, we might have truncated the curve at a low false-alarm rate such as 0.3 and calculated the area under this portion of the curve to lend greater emphasis to the region of interest.

4 Recommender System Algorithms Tested

In this paper, we apply and evaluate three probabilistic methods for describing the relationship between people and items: a logistic principal component analysis, an aspect model, and a naïve Bayes model. We will also employ *popularity ranked* recommendation methods that effectively recommend the most popular items or recommend only to users that seem to like everything. Popularity rank methods are interesting because they are the simplest to implement and as we will see in the evaluations sometimes outperform other algorithms. For consistency's sake, the models are denoted in the form of a movie recommendation task with the symbol m standing for a particular movie. However, in our evaluation we will recommend web pages as well.

4.1 Logistic Principal Component Analysis

Logistic principal component analysis (LPCA) [33] is a dimensionality reduction technique for binary data that is derived from a generalization of standard principal component analysis in the same way that logistic regression is derived from the generalized linear model framework. In the movie recommendation domain we use $|P|$, $|M|$ and $|L|$ to denote the number of people, movies and the latent dimensions, respectively. Given a Person×Movie matrix X of binary occurrence data indicating whether person p sees movie m, we hypothesize a latent space of dimension $|L|$ where $|L| << |M|$. We reduce the original data X to a set of coordinates ($U = |P| \times |L|$ matrix) in a lower dimensional axis ($V = |L| \times |M|$ matrix). We calculate coordinates and axis (U and V) by maximizing the likelihood:

$$\mathcal{L} = \sum_{p=1}^{|P|} \sum_{m=1}^{|M|} [X_{pm} \log \sigma(\Theta_{pm}) + (1 - X_{pm}) \log \sigma(-\Theta_{pm})] \quad (3)$$

where Θ factors into U and V as described in Table 1 and the logistic function $\sigma(\theta)$ is defined as:

$$\sigma(\theta) \quad = \quad \frac{1}{1+\exp(-\theta)}. \tag{4}$$

In our benchmarking experiments we estimate the parameters through the alternating least squares (ALS) algorithm described in Schein *et al.* [33], but fit the dimensions in a stagewise fashion (*e.g.* determine dimension $k-1$, fix it and then determine dimension k). We set the number of dimensions by performing validation on a portion of the training data, using a separate test set of observations only at final evaluation.

4.2 The Two-Way Aspect Model

The aspect model, a latent class variable framework designed for contingency table smoothing [15], appears natural for the task of predicting an association between p and m. Figure 1 (a) shows a graphical model description of the aspect model for a person/movie contingency table and Table 2 explains our notation used in the graphical model as well as in other descriptions of the aspect model applied to the movie recommendation task.

4.2.1 Pure Collaborative Filtering Model

The aspect model of Figure 1 (a) encodes a probability distribution over each person/movie pair. In other RS domains we replace the movie symbol M with the item of interest. Observations consist of tuples (p, m) recording that person p has seen/rated movie m. We store observations in a *count matrix* or *contingency table* with rows ranging over people and columns ranging over movies (or *vice versa*). In some domains the data may include multiple observations that are identical (*e.g.*, Lyle saw *Memento* twice). With each observation we increment by one the count of the appropriate contingency table *cell* (or matrix entry). A naïve probability estimate for each cell is simply the observed frequency of events in that cell. However, notice that using this method of assigning probabilities, an empty cell implies that there is zero probability of the corresponding person seeing the corresponding movie, clearly an unrealistic inference.

An *aspect model* hypothesizes the existence of a hidden or *latent* variable z (*e.g.*, an affinity for a particular style of movies) that motivates person p to watch movie m. According to the generative model semantics, person p chooses a latent class z, which in turn determines the movie m watched. The choice of movie m is assumed independent of p given knowledge of z. Since z is hidden, we sum over possible choices to define the distribution over (p, m):

$$\mathsf{P}(p, m) \quad = \quad \sum_z \mathsf{P}(p)\mathsf{P}(z|p)\mathsf{P}(m|z). \tag{5}$$

Parameters $\mathsf{P}(z|p)$ and $\mathsf{P}(m|z)$ correspond to the processes of p stochastically choosing a latent class z, and z stochastically choosing m. The $\mathsf{P}(p,m)$ values can be thought of as smoothed estimates of the probability distribution of the contingency table. The latent variables perform the smoothing in a manner that maximizes the model likelihood (by keeping estimates of $\mathsf{P}(p,m)$ close to the empirical distribution). The model also creates smoothed estimates for the values $\mathsf{P}(p)$ and $\mathsf{P}(m)$, both taking their interpretations from contingency table analysis. The parameters are calculated using the tempered EM algorithm as described in [15]. We choose the number of latent classes using performance on a partition of training data as the criterion. Our own source code for fitting the two-way aspect model is available online [30].[3]

4.2.2 Adding Content Information

The recommender system described so far is a pure collaborative filtering algorithm developed by Hofmann and Puzicha [16]. In our benchmarking, we will employ the pure collaborative filtering aspect model to the prediction of web page visitation. In addition, we will employ a similar model to cold-start recommendations of movies. Since it is impossible to make cold-start recommendations with any pure collaborative filtering model, we seek a way to implant content information into the aspect model. The person/actor aspect model of Figure 1 (b) combines collaborative with content data in one model by exploiting lists of actors starring in a movie as "content":

$$\mathsf{P}(p,a) \quad = \quad \sum_z \mathsf{P}(p)\mathsf{P}(z|p)\mathsf{P}(a|z). \tag{6}$$

Person/content models of this form have been shown to improve performance when the pure collaborative techniques suffer from sparse data [23].

We generate a dataset from the collaborative filtering model by taking the collaborative observations (p,m) and creating a set of observations (p,a_i) for each actor i in movie m. These newly formed observations are not independent of each other, explicitly breaking a modeling assumption of the aspect model. In practice the broken assumption does not prevent us from obtaining good results from this model.

4.2.3 Folding In

Notice that the person/actor aspect model does not have a movie object in the event space. In order to recommend a movie, we must create a new movie object out of the set of actors that appear in that movie. This pseudo-movie is then placed in the latent space based on the content information. We use Hofmann's [15] *folding-in* algorithm (originally used to fold term-queries into a document-word aspect model). For

[3] http://www.cis.upenn.edu/datamining/software_dist/PennAspect/index.html

example, suppose we have fit a person/actor model and want to fold-in a new movie. We create a new set of parameters $\mathsf{P}(z|m)$ and use the actors in the movie $\{(a, m)\}$ as evidence for placing the movie in latent space in a manner that maximizes the likelihood of the movie. All of the original parameters from Eq. (6) are held constant during the process. The exact EM algorithm operates as follows:

E-Step:

$$\mathsf{P}(z|a, m) \propto \mathsf{P}(a|z)\mathsf{P}(z|m) \tag{7}$$

M-Step:

$$\mathsf{P}(z|m) \propto \sum_{a} n(a, m)\mathsf{P}(z|a, m) \tag{8}$$

Recommendations are made using:

$$\mathsf{P}(p|m) = \sum_{z} \mathsf{P}(p|z)\mathsf{P}(z|m). \tag{9}$$

If we desire an estimated value of $\mathsf{P}(p, m)$, we will first need to estimate or define $\mathsf{P}(m)$. To generate the experimental results in this paper, we use a uniform prior probability assumption over movies.

4.3 Naïve Bayes Recommender

In cases where we wish to predict explicit rating rather than implicit preference information, we use the bag-of-words naïve Bayes text classifier as applied to recommender systems tasks in [19], but in our case applied to person/actor data. For each person a separate naïve Bayes classifier is trained so that no collaborative information is used. Hence, this is a pure content-based method capable of cold-start recommendation. The model is trained using Laplace smoothing.

Ratings prediction is computed with the probability function:

$$\mathsf{P}(c_j|M) \quad = \quad \frac{\mathsf{P}(c_j)}{\mathsf{P}(M)} \prod_{i=1}^{|M|} \mathsf{P}(a_i|c_j) \tag{10}$$

where the class c_j is a rating. Embedded in formula (10) is the belief that all movies in the data set are rated. In other words, naïve Bayes predicts *given* a rating has occurred *which* rating (c_j) was applied. We follow the experimental design of Melville *et al.* [18] who applied this classifier to a movie dataset similar to our own; we duplicate some of their testing strategies while adding our own.

4.4 Recommendation by Popularity

The simplest strategy for a recommender system is to exploit popularity statistics. There are two types of popularity: item and user. Item popularity is the tendency of certain items to be liked/purchased by

most users. Similarly, user popularity is the phenomenon of certain users who like/purchase more items than others. Traditional ROC and MAE measures of success reward systems that take either item, user or combined popularity into account. In our evaluation of web pages, we take a combined popularity approach to popularity recommendation. We train a linear regression model to predict whether a user did/did not visit a web page URL using three terms:

1. The percentage of URLs in the data set the user visited
2. The percentage of people who visited the URL
3. An interaction term combining 1 and 2.

In cold-start settings such as our movie recommendation experiments we have no data on the percentage of users who have rated the set of movies in our test set. Popularity-based recommending reduces to user popularity recommending in the cold-start setting. The symmetrical method of item popularity is embedded in the LPCA model when the latent dimensionality is zero; the bias vector Δ of Table 1 captures item popularity.

5 Data and Experimental Methodology

Our testing data comes from two sources: web access logs from www.microsoft.com [3] and the MovieLens data set assembled by the GroupLens project [13] consisting of actual ratings of movies from a group of users. The Microsoft web log consists solely of implicit rating information in the form of *person p clicked vRoot u,* where a vRoot is a portion of the web site determined by URL prefix. In this section we define the probabilistic event space and explore the alternative definitions of success and failure that can be used in the context of testing a recommender system. Note that we distinguish between the definition of *event* from the point of view of a recommender system algorithm from that of the testing framework. For instance, the aspect model recommenders described in this paper define a probability that a person has rated a movie from which we can create a ranked list of recommendations: $(p, m)_n$. With a ranked list created in this fashion we may predict like/dislike preferences, even though notions of like and dislike do not occur in the aspect model's input.[4] Alternatively, we might train a recommender system using events that contain rating information, but predict purchases or web page clicks rather than preferences. For this reason, we must carefully define terms like *event*, *success* and *failure* separately in the context of the testing framework.

[4]It would be possible to modify the presentation of the aspect from what is described in this paper to facilitate the incorporation of ratings, but this remains a topic for later research.

5.1 Definitions

Testing in the web log domain, we define the events as *implicit rating* events since no explicit preference for a web page is indicated other than the page visitation. Formally, the events can be represented by the tuple: (p_i, m_j).[5] Analogously, implicit rating events are defined on the MovieLens movie recommendation data by removing the actual rating, and noting instead that the movie was rated. The number of possible events is the size of the cross product: $|P| \cdot |M|$. However, not all events are *observed*, i.e. constitute observations in the statistical sense. Rating information is included in the MovieLens data set, so we have the opportunity to create an alternative definition of *event* in a way that includes the rating: (p_i, m_j, r), where the variable r encodes the rating. The definitions of *positive* and *negative* outcomes in making recommendations that are used in building ROC and CROC curves vary depending on the type of testing. Three modes of testing with their accompanying definitions of positive and negative are described below and summarized in Figure 2:

1. Implicit Rating Prediction

   Implicit rating prediction refers to prediction of data such as purchase history; a purchase is not necessarily an indication of satisfaction, but a purchase is an indication of some implicit need or desire for an item. For MovieLens data evaluated in the implicit rating setting we predict that a person has rated a movie, a task that is analogous to predicting a customer purchase. When evaluating performance, we do not consider the rating itself or the positive events that occur in the training set. The remaining events in the cross-product Person $\times$ Movie are included in the test set. Implicit rating prediction is most appropriate for domains where explicit rating information is not available and we are satisfied to recommend products that the user is likely to purchase on their own. Past implicit evaluation work includes [22, 23, 32]. We will refer to MovieLens observations stripped of a rating component as implicit rating data throughout this paper. We evaluate the aspect model on the implicit rating task of the MovieLens data set for cold-start prediction. The LPCA model must be generalized to incorporate content information before it is applied on a cold-start domain such as the MovieLens evaluation.

   The Microsoft data set falls under the category of the implicit rating task as the only rating information is stored implicitly through the web site visits/clicks. We evaluate both the aspect model and the LPCA model on the Microsoft data set using the given train/test splits of positive observations. Test set events are taken from the cross product of Person $\times$ vRoot minus those events that occur in the training set: corresponding to an assumption that we only predict vRoots we don't observe in the training set.

[5] Here we continue the convention that m_j can stand for either a web page or a movie depending on the domain.

Other assumptions could be used alternatively.

2. Rating Prediction

   In rating prediction we wish to predict both implicit rating and rating components of an observation simultaneously. In MovieLens benchmarking we classify each person/movie pair that doesn't occur in the training observations into two groups:

   a) p_i rated $m_j \geq 4$.

   b) p_i did not rate $m_j \geq 4$.

   Condition b includes the case where person i did not rate movie j at all.

3. Conditional Rating Prediction

   Conditional rating prediction is prediction of ratings for items conditioned on our knowledge that such a rating occurred. For the MovieLens data, we ask "given that a person has seen movie x, how likely are they to rate it ≥ 4?" Our prior knowledge that the person has seen x means we have a implicit rating observation to this effect. Our goal is to guess the best rating. We implement conditional rating prediction testing by taking held out observations from the MovieLens data and predicting ratings on this set. Unlike the implicit rating and explicit rating evaluation we do not consider the the cross product Person $\times$ Movie in evaluation as all relevant information for testing are in the given observations. In real-world applications we may have data sets where implicit rating observations are available in large quantities, but the rating component is missing at random. Conditional rating prediction measures success at filling in the missing values. Conditional rating prediction has been used previously in [3, 13, 18] to evaluate recommender system performance.

The difference between the definitions for rating prediction and conditional rating prediction is subtle, but potentially critical to interpreting real-world performance of recommender system algorithms. Figure 2 shows graphically the differences between rating prediction and conditional rating prediction. Unfortunately, importance of the distinction remains untested due to lack of appropriate public data sets. The MovieLens data set is most appropriate for conditional rating prediction evaluation due to the bias in event generation. Initial movie selection shown to users are not entirely random and become less so as the user rates more movies. Though our implicit rating evaluation partially reflects the ability of our RS method to capture the training set bias in making predictions, we can still learn valuable lessons about the evaluation process itself by implicit rating prediction on this data set.

5.2 Data Preparation

The Microsoft web log data set [3] consists of 32,712 anonymous users and 286 vRoots (URL prefixes). The data set comes with predetermined train/test splits of 98,655/15,192 positive events respectively. The mean number of vRoot visits per user is 3.0.

The MovieLens data set [13] consists of 943 users and 1682 movies and events that include rating preferences. In order to evaluate recommender systems under a cold-start scenario we split the movies into two groups: 1351/331 (training/test) movies. There are 19,192 observed events for the test set movies out of a possible total of 312,133. Of the test set observations, 10,640 contain a rating of 4 or higher, where the rating scale is 1-5. In order to exploit casts of actors associated with the individual movies we downloaded relevant information from the Internet Movie Database (http://www.imdb.com). We sped up model fitting by considering only actors billed in the top ten and eliminating actors who appear in only one movie.

6 Results

The results are divided into three categories: implicit rating testing of the Microsoft data set, implicit rating testing of the MovieLens data set and conditional rating prediction testing on the MovieLens data set.

6.1 Microsoft Web Log Implicit Rating Testing

Figure 3 shows ROC and CROC plots of LPCA, the aspect model and the popularity recommender. The popularity recommender is the method described in Section 4.4 combining both user and item popularity through a linear regression. The LPCA model tested has latent dimension $|L| = 0$; it relies on the bias vector Δ described in Schein *et al.* [33]. The size $|L| = 0$ was chosen during cross-validation on a training set partition where we determined that overfitting begins at $|L| = 1$. When training LPCA in this manner, the model reduces to an item popularity recommender. The aspect model is tested with four latent variables as determined during cross-validation on the training set.

There are several results from this evaluation that differ from the two MovieLens tests. For instance, in the Microsoft evaluation the relative rank of the regression method, aspect model, LPCA and random recommender remains constant between both ROC and CROC metrics. Also, the CROC curve areas for all methods are greater than the ROC areas, indicating that on this domain we can perform strongly according to the CROC criteria. In contrast, in all the other evaluations we make fewer good recommendations when adding the CROC constraint.

The Microsoft evaluation gives us an opportunity to examine the benefit of adding user popularity infor-

mation to an item popularity recommender (embedded within LPCA). The LPCA and user/item popularity recommender perform identically on the CROC curve: as expected since user information disappears from the user/item popularity recommender when evaluating according to the CROC criteria. However, in the ROC curve the combined user/item popularity recommender is inferior to LPCA indicating overfitting on the part of the user/item popularity method. The overfitting effect is surprising given the simplicity of the user/item popularity method, *i.e.* the small number of parameters embedded in the model. We would not have expected or identified it without evaluation with the LPCA (item popularity) method.

6.2 MovieLens Implicit Rating Testing

Figure 4 shows ROC and CROC plots comparing the person/actor aspect model and a popularity recommender on the MovieLens implicit rating prediction task. The popularity recommender for the ROC plot is created by substituting the percentage of movies seen by person i for the recommender output for pair: (p_i, m_j) as described in Section 4.4. In other words, we list users by their relative activity in rating on the MovieLens site and recommend *all* movies to these users in that order. The aspect model is tested with six dimensions as determined by cross-validation on the training set. For most practical purposes, the popularity recommender performs as well as the aspect model according to the ROC curve, especially in the region of interest (the far left-hand side of the graph).

Figure 4 (b) shows the corresponding CROC plots comparing the same methods. The user popularity recommender becomes a random recommender in the CROC curve. Since this particular evaluation allows us to apply Theorem 1, the inclusion of the user popularity recommender in the CROC curve serves to test whether random performance is close to a forty-five degree line in practice. Note that the aspect model performs noticeably better than the user popularity recommender according to the CROC curve. In drawing conclusions from the ROC and CROC curves in Figure 4, we see that the aspect model and popularity recommender have nearly identical performance characteristics in the ROC case. We might expect to see the same behavior occur in the CROC case, but instead the aspect model is the clear winner. We conclude that the aspect model demonstrates the ROC curve of the popularity recommender while simultaneously recommending to a wider user base.

6.3 MovieLens Conditional Rating Prediction Testing

For the MovieLens conditional rating prediction experiments we selected the naïve Bayes model since it has been used previously on a similar data set [18]. Our preliminary experiments with naïve Bayes confirmed earlier observations [18] that naïve Bayes is sensitive to sparsity when users have rated fewer than forty

movies on this type of data (results not shown). In order to better duplicate the prior work in conditional rating prediction [18] as well as present naïve Bayes enough data to learn from, we perform conditional rating prediction only for users who have rated forty or more movies in the training set.

Figure 5 shows ROC and CROC plots comparing the naïve Bayes recommender and a popularity recommender on the conditional rating prediction task in a cold start setting. Here we predict that a user rates a movie 4 or higher (out of 5). This technique of evaluation has been called ROC-4 [27]. The popularity recommender is created by using the mean rating for each person i as the predicted rating for (p_i, m_j), for all m_j. The mean rating method outperforms all other methods by ROC standards. As in the implicit rating prediction task, the mean rating popularity recommender reduces to a random recommender when switching to the CROC curve, though Theorem 1 does not apply to this evaluation.

In drawing conclusions from Figure 5 we see that user popularity matches the naïve Bayes model by the ROC evaluation in the region of interest (the left-hand side of the curve). However according to the CROC evaluation naïve Bayes method is superior to the random recommender (the popularity method). This leads us to the conclusion that the naïve Bayes algorithm gets similar ROC performance to the popularity recommender while recommending to a wider set of users.

7 Discussion

We have advocated the use of the CROC curve in addition to the traditional ROC curve in conjunction with simple popularity-based recommenders as a means to a more complete analysis of recommender system performance. Decisions about specific recommender system deployment should be made on the basis of effort and cost balanced by the anticipated benefit. In this section we discuss how our testing methodology reveals the benefits and disadvantages of systems of varying complexity towards the goal of making informed deployment decisions.

7.1 Results Confirm Differences between ROC and CROC

A natural question to ask is whether both the ROC and CROC curves are needed in evaluation. One hypothesis is that relative performance on a ROC curve graph is sufficient to tell the whole story of recommender system performance characteristics. Our results show that this is not the case; in particular our empirical results demonstrate that ROC and CROC curves can often have no predictive power over each other: *e.g.*, they often give conflicting measures, as was the case in the MovieLens implicit rating and conditional rating prediction evaluation. Even in application domains where unconstrained recommendation is permitted, the

CROC curve provides valuable information regarding the RS characteristics. The difference in performance between ROC and CROC curves comes from a re-ranking of recommendations according to constraints; the output from the individual recommender system algorithms remains unchanged and the definitions of hit rate and false-alarm rate also remain unchanged. By looking at the CROC curve we learn which RS algorithms get good ROC performance while recommending to a wide group of users.

The Microsoft evaluation gives us two additional lessons. First we learn that there are domains where relative performance on ROC and CROC curves are stable. The Microsoft evaluation also shows a case where CROC standards produce "taller" curves than ROC curves. In general, this behavior may be an indication that item popularity is a powerful recommender in the domain. In contrast, in the MovieLens experiments the hit rate falls dramatically switching from the ROC curve to the CROC curve, when holding the false-alarm rate fixed.

7.2 The Use of Popularity Recommenders

On all ROC plots, some of the simplest methods are competitive with complex machine learning algorithms. The use of intelligently designed *popularity recommenders* like the ones employed in our benchmarking are a key ingredient to interpreting the performance of a ROC curve. Past performance results using the MovieLens and similar datasets that employ the ROC curve analysis including [13, 18, 31] should be evaluated with this observation in mind. The Microsoft data set is a case where popularity recommenders, particularly the item popularity recommenders come out well ahead by both ROC and CROC criteria. The LPCA model is convenient for such data sets as we can implement an item popularity recommender while quickly testing higher dimensional structure and ruling it out when necessary. From a recommender system implementor's point of view, the primary lesson from the Microsoft data set evaluation is that simpler recommender systems sometimes give better results.

7.3 ROC/CROC Compared to Alternative Metrics

Could our conclusions about recommender systems be drawn using other traditional performance metrics such as mean absolute error (MAE) or the ranked list statistics, such as Breese *et al.*'s [3] R? We argue that MAE and the R metric can not by themselves be used to replace our two-metric strategy. In the case of MAE, the metric is not able to adequately evaluate black box ranked list generators since the error is calculated as the absolute difference between a prediction value and the actual test value (zero or one):

$$\text{MAE} = \frac{1}{N} \sum_{p,m} |\text{pred}(p,m) - \text{test}(p,m)|, \tag{11}$$

where N is the size of the test set and test(p, m) denotes the test set outcome for p and m. The aspect model is an example of a particular model that would incorrectly suffer from MAE evaluation; the outputs define a probability distribution over pairs (p, m) that sum to one: each individual $\mathsf{P}(p_i, m_j)$ is relatively close to zero, hovering near $1/(|P| \cdot |M|)$ even for relatively large values. This effect using the aspect model is noted in González-Caro *et al.* [9], where comparing the relative performance according to MAE indicates the aspect model is among the worst methods, but according to ROC area the aspect model is among the best methods tested.

The R statistic variant for implicit ratings [23] bears similarities to the area under the CROC curve, essentially compressing the performance of all users into one value:

$$R = 100 \frac{\sum_p R_p}{\sum_p R_p^{\max}} \quad \text{where} \tag{12}$$

$$R_p = \sum_m \frac{\delta(p, m)}{2^{(m-2)/(\alpha-1)}}, \tag{13}$$

$\delta(p, m)$ is 1 for positive outcomes in the test set and 0 for negative outcomes, m denotes the index of the m'th movie in declining order of the prediction, α is the exponential decay, a tunable parameter, and $R_p^{\max}$ is the maximum achievable R_p value. The R statistic's uniform averaging over all users makes it inappropriate for unconstrained recommender system evaluation. The setting of a tunable parameter (α) and the compression of individual hit/false-alarm rate tradeoffs into a single statistic are other characteristics of this metric that distinguish it from both CROC and ROC curves.

8 Conclusions

In this paper we present the CROC diagnostic plot for use in recommender system evaluation, analyze its theoretical properties and use the plot in empirical evaluation of recommender systems. Results on the MovieLens data set demonstrate that at times the ROC plot relative rankings of methods differ substantially from the rankings generated from CROC plots. The results on the Microsoft data set using the CROC curve are uniformly stronger than those using the ROC curve, indicating that at times constrained recommending can actually improve the total number of good recommendations made, holding the data set and underlying recommender system algorithms fixed. Even when the intended application is relatively unconstrained, *i.e.* corresponding to the ROC curve, the CROC curve can identify the recommender system's coverage of the user base, leading to more accurate and specific conclusions about an algorithm's performance.

We have not addressed which of the ROC or CROC curve methods provide better evaluation in general for real-world applications. Unconstrained recommendation as measured by the ROC curve and constrained

recommendation as measured by the CROC curve define two extremes of a continuum. Allowing the recommendation of all items to a single user, as permitted by the ROC curve is usually unattractive in Ecommerce applications, since the user who receives such a long list will likely not look past the top few items, and may get annoyed. On the other hand the CROC curve can also be considered disadvantageous, since it is reasonable to assume that there will be increased opportunities to market items to more active users; for instance when they log into a web site. Our thesis in this work is that by applying *both* metrics to the evaluation of recommender system algorithms, we discover the unconstrained performance through the ROC curve and user coverage performance through the CROC curve. In most applications both user coverage and raw performance are desirable qualities in a recommender system, and since user-coverage and raw performance can be at odds, quantifying both will lead to better application-specific recommender system deployment decisions.

9 Acknowledgments

Andrew Schein was supported by NIH Training Grant in Computational Genomics, T-32-HG00046. Alexandrin Popescul was supported in part by a grant from the NEC Research Institute. We would like to thank the reviewers for their many valuable comments.

References

[1] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720, 1998.

[2] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 46–54, 1998.

[3] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[4] M. Claypool, A. Gokhale, and T. Miranda. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems—Implementation and Evaluation*, 1999.

[5] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[6] M. K. Condliff, D. D. Lewis, D. Madigan, and C. Posse. Bayesian mixed-effect models for recommender systems. In *ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*, 1999.

[7] W. Feller. *Introduction to Probability Theory and Its Application, Vol. 1.* John Wiley & Sons, Incorporated, 3rd edition, 1990.

[8] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 170–178, 1998.

[9] C. N. González-Caro, M. L. Calderón-Benavides, J. de J. Pérez-Alcázar, J. C. García-Díaz, and J. Delgado. Towards a more comprehensive comparison of collaborative filtering algorithms. In *Proceedings of the 2002 String Processing and Information Retrieval Conference (SPIRE 2002)*, pages 248–253, 2002.

[10] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. M. Sarwar, J. L. Herlocker, and J. Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 439–446, 1999.

[11] H. Guo. SOAP: Live recommendations through social agents. In *Fifth DELOS Workshop on Filtering and Collaborative Filtering, Budapest*, 1997.

[12] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for collaborative filtering and data visualization. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 264–273, 2000.

[13] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the Conference on Research and Development in Information Retrieval*, 1999.

[14] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 194–201, 1995.

[15] T. Hofmann. Probabilistic latent semantic indexing. In *22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57, 1999.

[16] T. Hofmann and J. Puzicha. Latent class models for collaborative filtering. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 688–693, 1999.

[17] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. GroupLens: Applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[18] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, 2001.

[19] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 195–204, 2000.

[20] A. Nakamura and N. Abe. Collaborative filtering using weighted majority prediction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 395–403, 1998.

[21] D. M. Pennock, E. Horvitz, and C. L. Giles. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 729–734, 2000.

[22] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 473–480, 2000.

[23] A. Popescul, L. H. Ungar, D. M. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, 2001.

[24] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 175–186, 1994.

[25] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[26] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill, 1983.

[27] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Analysis of recommender algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pages 158–167, 2000.

[28] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender system—A case study. In *ACM WebKDD Web Mining for E-Commerce Workshop*, 2000.

[29] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference*, pages 285–295, 2001.

[30] A. I. Schein, A. Popescul, and L. H. Ungar. PennAspect: A two-way aspect model implementation. Technical Report MS-CIS-01-25, Department of Computer and Information Science, The University of Pennsylvania, 2001.

[31] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Generative models for cold-start recommendations. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems*, 2001.

[32] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25'th International ACM Conference on Research and Development in Information Retrieval*, 2002.

[33] A. I. Schein, L. K. Saul, and L. H. Ungar. A generalized linear model for principal component analysis of binary data. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[34] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 210–217, 1995.

[35] J. A. Swets. Measuring the accuracy of diagnostic systems. *Science*, 240:1285–1293, 1988.

[36] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at the Fifteenth National Conference on Artificial Intelligence*, 1998.

$\|P\|$	number of observations (people)
$\|M\|$	dimensionality of data (movies)
$\|L\|$	dimensionality of latent space

$U_{n\ell}$	coefficients	($\|P\| \times \|L\|$ matrix)
$V_{\ell m}$	basis vectors	($\|L\| \times \|M\|$ matrix)
Δ_m	bias vector	($1 \times \|M\|$ vector)

$$\Theta_{pm} = (UV)_{pm} + \Delta_m$$

Table 1: Notation for the Logistic PCA model.

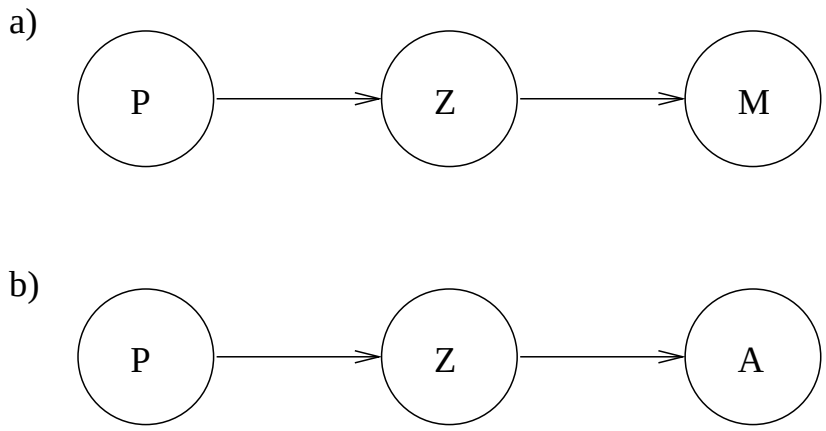

Figure 1: Graphical model of (a) person/movie aspect model and (b) person/actor aspect model. These graphs can be interpreted precisely as Bayesian belief networks.

Random Variable	Object	Interpretation
P	p	person
M	m	movie
A	a	actor
Z	z	latent class

Table 2: Notation used in our aspect model descriptions.

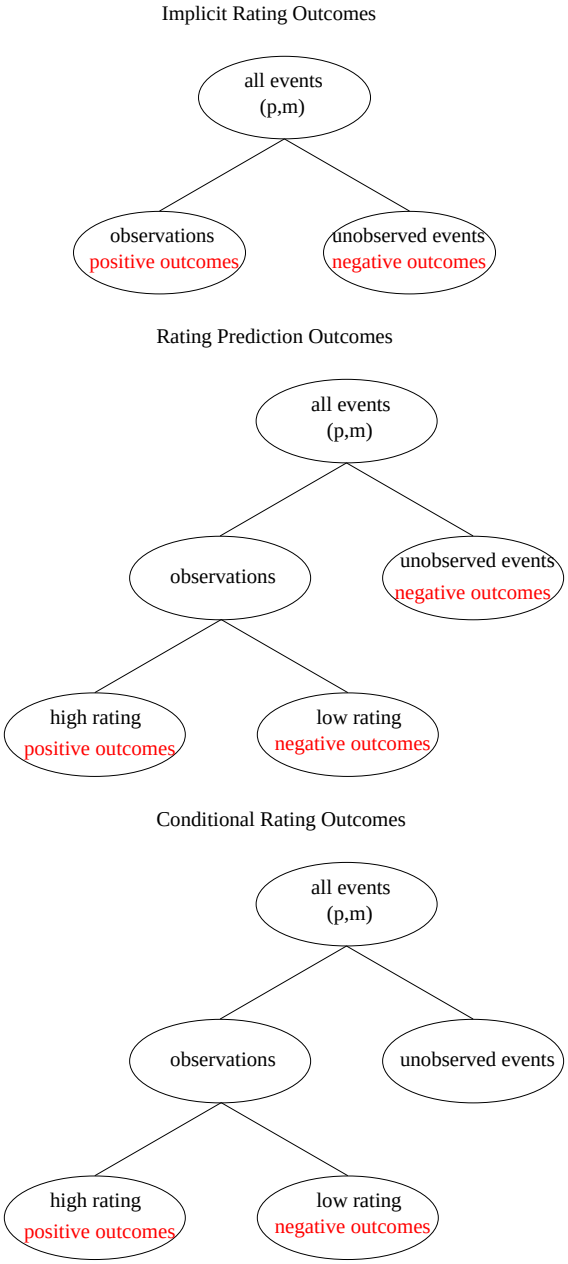

Figure 2: Division of event space into positive and negative outcomes for implicit rating prediction, rating prediction and conditional rating prediction modes of recommender system evaluation. Lower levels of the hierarchies indicate partitioning of the event space.

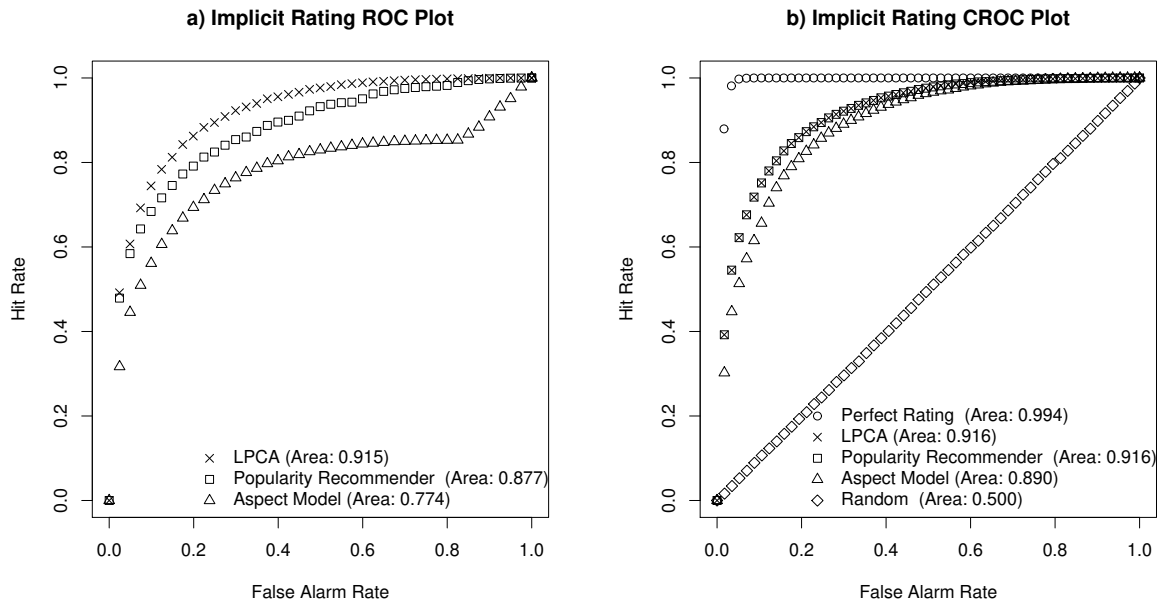

Figure 3: **Web Log Results:** ROC (a) and CROC (b) plots comparing a person/vRoot aspect model, logistic PCA and a popularity recommender (based on user and vRoot activity) on the *implicit rating* prediction task. To generate ROC plot points we increment the number of predictions by 156,898. To generate CROC plot points we increment the number of URLs predicted for each person by 5.

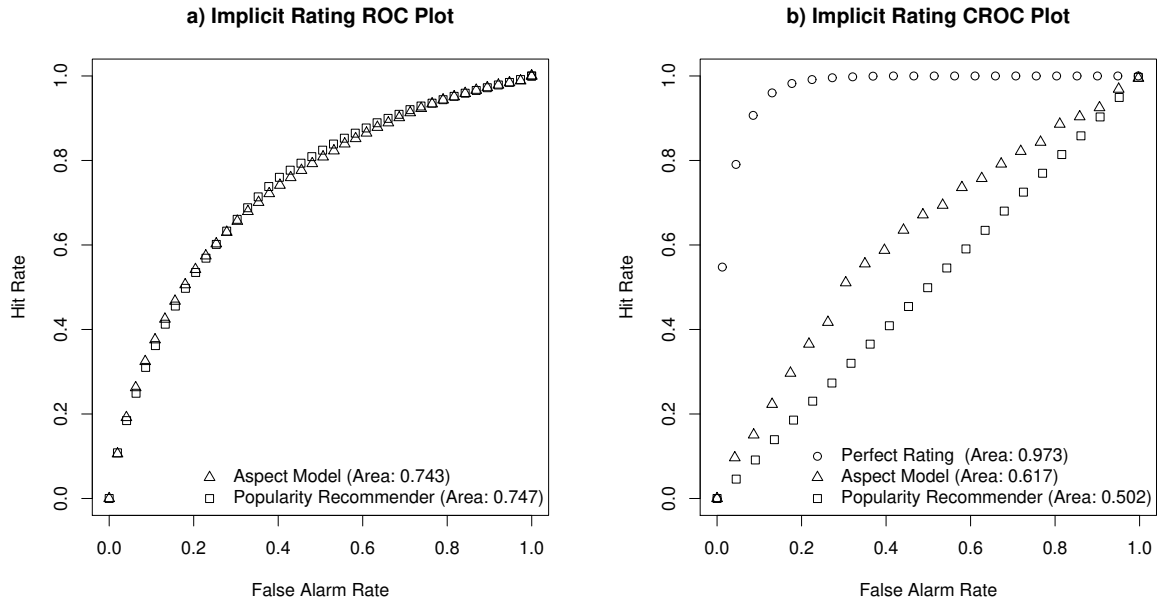

Figure 4: **MovieLens Results:** ROC (a) and CROC (b) plots comparing person/actor aspect model and a user popularity recommender on the *implicit rating* prediction task. To generate ROC plot points we increment the number of predictions by 7803. To generate CROC plot points we increment the number of movies predicted for each person by 15. The user popularity recommender becomes a random recommender when evaluating by the CROC scheme.

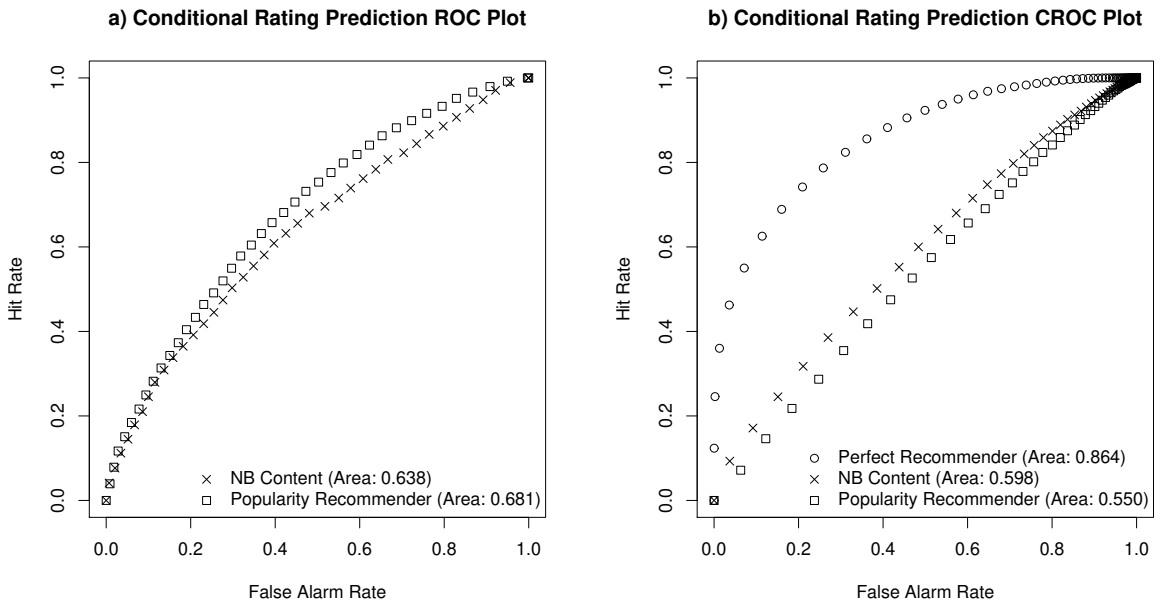

Figure 5: **MovieLens Results:** ROC (a) and CROC (b) plots comparing naïve Bayes and a popularity recommender (based on user mean rating) on the *conditional rating prediction* task. To generate ROC plot points we increment the number of predictions by 425. To generate CROC plot points we increment the number of movies predicted for each person by 2. The user mean rating recommender becomes a random recommender when evaluating by the CROC scheme.