A Probabilistic Definition of Item Similarity

Oliver Jojic
Comcast Corporation
1110 Vermont Ave NW
Washington, D.C.
703-449-1315
Oliver_Jojic@cable.comcast.com

Manu Shukla
Comcast Corporation
1110 Vermont Ave NW
Washington, D.C.
703-430-6849
Manu_Shukla@cable.comcast.com

Niranjan Bhosarekar
Comcast Corporation
1110 Vermont Ave. NW
Washington, D.C.
202-456-1166
Niranjan_Bhosarekar@cable.comcast.com

ABSTRACT
In item-based collaborative filtering, a critical intermediate step to personalized recommendations is the definition of an item-similarity metric. Existing algorithms compute the item-similarity using the user-to-item ratings (cosine, Pearson, Jaccard, etc.). When computing the similarity between two items A and B many of these algorithms divide the actual number of co-occurring users by some "difficulty" of co-occurrence. We refine this approach by defining item similarity as the ratio of the actual number of co-occurrences to the number of co-occurrences that would be expected if user choices were random. In the final step of our method to compute personalized recommendations we apply the usage history of a user to the item similarity matrix. The well defined probabilistic meaning of our similarities allows us to further improve this final step. We measured the quality of our algorithm on a large real-world data-set. As part of Comcast's efforts to improve its personalized recommendations of movies and TV shows, several top recommender companies were invited to apply their algorithms to one year of Video-on-Demand usage data. Our algorithm tied for first place. This paper includes a MapReduce pseudo code implementation of our algorithm.

Categories and Subject Descriptors
Recommender Systems, Similarity Computation.

General Terms
Algorithms, Measurement, Performance, Experimentation, Verification.

Keywords
Recommender Systems, Similarity Computation, Video on Demand Recommendations, Usage-based Recommendations

1. INTRODUCTION
This paper describes work at Comcast to provide personalized recommendations to cable subscribers. Personalized recommendations are computed from customers' usage history. The history includes users watching Video-on-Demand titles from Comcast set-top boxes or from Comcast's XFinity website.

RecSys'11, October 23–27, 2011, Chicago, Illinois, USA.

As consumers are increasingly faced by a proliferation of new video content choices and different ways to watch, video recommendations have become an important tool for helping users discover content that they are likely to enjoy.

A common method to determine a list of recommendations personalized to a user is to compute the ratings the user would give to all items and then rank the items by rating. That raises the question of whether this approach will produce optimal recommendations, even with a perfect rating detection algorithm. It is actually easy to find examples where ranking by rating does not produce optimal recommendations. For example, a fan of science fiction might occasionally watch a detective movie. Let's imagine such a fan would give four stars to an excellent detective movie, and only three stars to some new science fiction show. Even though the rating of the science fiction show is lower (three vs. four), the fan will not want to miss it and might not bother watching the recommended four-star detective movie. This example illustrates the correlation between the rating and the quality of an item, as seen by a user. What should be more relevant to some recommender systems is the likelihood that a user will consume an item. Throughout this paper we use the common verb *to like* to indicate a positive relationship between a user and an item. But for us *user u likes item x* represents the willingness of the user to consume the item.

Our algorithm does not handle ratings, but directly determines the evidence that a user will like an item. We first compute item similarity using a novel probabilistic definition of similarity. Then we apply the usage history of a user to the similarity matrix to determine the accumulated evidence for the user to like an item.

The rest of this paper is structured as follows: in the second section we describe some related work; in the third section we describe the algorithm. In section four we give details of the dataset used in the experiments described in section five. In section six we describe the distributed implementation followed by conclusions and future work.

2. RELATED WORK
We briefly describe work that has been done in the fields of recommender systems and similarity calculations.

2.1 Recommender Systems
As recommender systems become more common, they become pivotal in presenting users with programs of interest based on the videos they have viewed in the recent past, thereby driving up video consumption [1-3]. Recommender systems have been actively researched for several years. There are several examples of movie recommender systems, some from usage analysis and others from the user interface perspective[4]. There are

recommender systems for video-on-demand services with similar scope as ours[5]. Recommender systems are often classified as collaborative based[6, 7] or content based recommendation systems[8, 9]. Hybrid approaches of collaborative and content based filtering have been used for movie recommendations[10]. Collaborative recommender systems can employ item-based collaborative filtering [11, 12], user-based filtering algorithms or hybrid algorithms. User tags can combine with user and item information to generate recommendations[13]. Item clustering is used for collaborating recommendation algorithms[14]. Pagerank based collaborative filtering algorithms have been proposed[15].

2.2 Similarity Measures

Numerous similarity measures have been used in recommender systems such as Cosine and its variants, Euclidean distance, Pearson's correlation, Jaccard coefficient, Tanimoto and many more. Tanimoto based similarity measure have been found useful in modeling collaborative semantics of geographic folksonomies[16]. Heuristic similarity measures that improve recommendations under cold-start conditions when only small amount of ratings are available have been proposed[17]. Similarity measures for sparse rating and user-item data use global and local user similarities[18]. Recommending items to users using collaborative filtering has been modeled as a relevance ranking problem mapping items to users[19]. Algorithms that take into account user and item neighborhoods in regularized matrix factorization model are found to be effective[20]. Other algorithms have combined latent factorization models with neighborhood models with good results[21]. Evaluation metrics for recommendation tasks that help match algorithm with the domain and task of interest have been outlined[22]. A probabilistic similarity approach similar to the one presented in the paper has never been proposed for item-based collaborative filtering.

2.3 Distributed Recommender Systems

Recommendation algorithms are usually computationally expensive and cannot run on single node in an acceptable amount of time. With the widespread use of distributed platforms like Apache Hadoop and Google's GFS with MapReduce, it has become possible to run the algorithms in a distributed manner. Recommender systems have been implemented in a distributed environment that take the collaborative filtering approach [23, 24]. There are also simpler distributed text recommender systems[25].

3. ALGORITHM

We first describe our definition of item similarity (the novel part of this paper). Then we describe how we determine personalized recommendations.

In the rest of this paper the term "ProbSim" refers to our new algorithm.

3.1 Item Similarity Algorithm

Our algorithm addresses a few issues with commonly used item similarity measures:

- Common similarities use *a priori* knowledge about items, but treat all users as equal. For the purpose of computing item similarity, should a user who likes ten items have the same weight as a user who likes a thousand? An extreme user who likes all items should have very little weight, if any.
- Common similarities are symmetrical. Non-symmetry is attractive, for example consider the case of a war movie W and a war-and-love movie WL. Many users that enjoy W will also enjoy WL because WL has a *war* component. While some of the users that enjoyed WL due to its *love* aspect might not enjoy W. Overall, W is more similar to WL than WL is similar to W.

Let's call "co-occurrences" between two items the number of users that liked both. We define the similarity between two items as the ratio of the actual number of co-occurrences to the number of co-occurrences that would be expected if user likes/dislikes were random.

Let U be the set of all users. Let A be the set of all items. Let r be the function that specifies if a user u likes an item x:

$$r: U \times A \rightarrow \{true, false\}$$

The function r can often be derived from available information:

- In a five-star rating system, one and two stars can be defined as *false,* with three, four, and five stars as *true.*
- For movies, we can use the fraction of the movie that was watched by the customer; e.g., more than 80% watched maps to *true.*
- When no information is available to make an informed decision, all items known to have been consumed by a user can be marked as *true*.

The function r is typically defined on a small subset of U x A.

We are tasked with defining the similarity of an item y to a target item x. Let $L(y)$ be the known set of users that liked item y:

$$L(y) = \{u \mid r(u, y)\}$$

Similarly, $L(u)$ is the set of items that are known to be liked by the user u:

$$L(u) = \{\, x \mid r(u, x) \,\}$$

3.1.1 *Probability of a User to Like an Item*

Let us assume that user likes/dislikes are random. Our aim is to determine the probability that a given user u likes a given item x, that is, $p(u,x)$.

If we were to only use *a priori* information about items, then the probability that a random user likes a given item x would be equal to the number of users that like x divided by the total number of users:

$$p(u, x) = p(x) = \frac{\|L(x)\|}{\|U\|} \quad (1)$$

If we were to only use *a priori* information about users, then the probability that a given user u likes a random item x is equal to the number of items liked by user u divided by the total number of items:

$$p(u, x) = p(u) = \frac{\|L(u)\|}{\|A\|} \quad (2)$$

If instead we consider *a priori* information about users and about items we will then determine a more precise probability. But what is a correct formulation of $p(u,x)$ knowing both $L(x)$ and $L(u)$? We simplify the problem by first focusing on only one user and one item. Later we take into account the impact of the surrounding density of likes.

Let us represent item *x* and user *u* in figure 1:

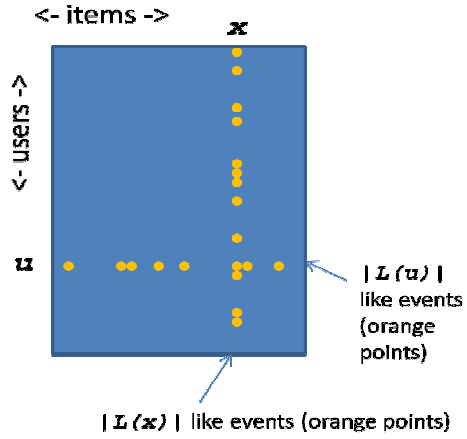

Figure 1. *Matrix of users and items. An orange point indicates that the corresponding user consumed and liked the corresponding item. For simplicity we only represent the orange points for user "u" and item "x". Consumed and disliked events are not shown because our item similarity metric does not use dislike events.*

An orange point represents a user liking an item.

To ease the notation we sometime drop the norm sign and define:

- *A* as the total number of items
- *U* as the total number of users
- *L(x)* as the known number of users that like *x*, and
- *L(u)* as the known number of items that are liked by *u*

If we assume that like/dislikes are spread randomly, then all permutations of the orange points in figure 1 are equiprobable. Under that assumption, the probability that user *u* likes item *x* is equal to the number of combinations for which the point *(u,x)* is orange divided by the total number of combinations.

The "positive" cases, when *u* likes *x*, are those cases where the point *(u,x)* is fixed as orange. Therefore the number of positive cases is the number of [combinations of *L(u)-1* horizontal points among *A-1*] times the number of [combinations of *L(x)-1* vertical points among *U-1*]:

$$\#P = \binom{A-1}{L(u)-1} \cdot \binom{U-1}{L(x)-1} \tag{3}$$

where $\binom{N}{n}$ is the number of combinations of *n* among *N*:

$$\binom{N}{n} = \frac{N!}{n!\,(N-n)!}$$

The "negative" cases, when *u* does not like *x*, are those cases where the point *(u,x)* is fixed as not colored. Therefore the number of negative cases is:

$$\#N = \binom{A-1}{L(u)} \cdot \binom{U-1}{L(x)} \tag{4}$$

The total number of cases is equal to the sum of the number of positive and the number of negative cases. After some simplifications we find the probability:

$$p(u,x) = \frac{\#P}{\#P + \#N} = \frac{1}{1 + \frac{A-L(u)}{L(u)} \cdot \frac{U-L(x)}{L(x)}} \tag{5}$$

3.1.2 *Effect of Surrounding Density of Likes*

The above combinatorial expression has the disadvantage of being independent of the surrounding distribution of likes[1]. Consider the surrounding density of orange points (likes) in figure 2. Due to the high density on the right side picture, the chance is reduced that a colored point from column *x* is due to the row *u*. The opposite is true on the left side picture.

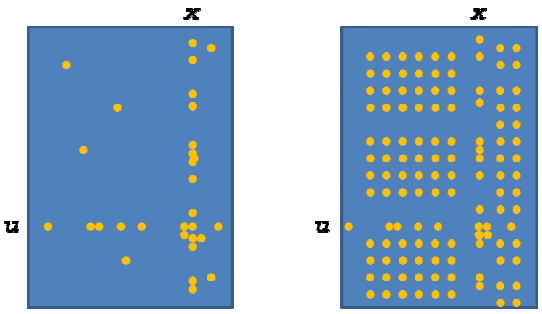

Figure 2. *The probability of a user u to like an item x is impacted by the surrounding density of "likes." The number of colored points for row u and column x in both pictures is the same. In the right picture the high density of likes reduces the chance that the intersection between u and x is painted. The opposite is true in the left picture.*

In a probability formula $p = \frac{1}{1+x}$ $(x \geq 0)$ a normalization factor $\pi >= 0$ can be introduced as:

$$p = \frac{1}{1 + \pi x}$$

In our case:

$$p(u,x) = \frac{1}{1 + \pi \cdot \frac{A-L(u)}{L(u)} \cdot \frac{U-L(x)}{L(x)}} \tag{6}$$

The effect of the normalization factor is to lower or increase all probabilities depending on the surrounding density of likes.

Let's call *V* the total number of *likes* (the total number of orange points).

When:

- User *u* has just the average number of likes per user:

$$L(u) = \frac{V}{U}$$

- Item *x* has just the average number of likes per item:

$$L(x) = \frac{V}{A}$$

Then *p(u,x)* should be equal to the average density of orange cells:

$$p(u,x) = \frac{V}{AU}$$

That is:

[1] Using the expression of *p(u,x)* in equation (5) we get a recall of 33%; compare this to the 39% recall obtained with the final expression of the probability in (9) (the experiment is described in a later section)

$$\frac{V}{AU} = \frac{1}{1 + \pi \frac{A - L(u)}{L(u)} \cdot \frac{U - L(x)}{L(x)}} \quad (7)$$

The resolution of this equation gives the value of π:

$$\pi = \frac{V}{UA - V} \quad (8)$$

Finally, the probability that user *u* likes item *x* is:

$$p(u,x) = \frac{1}{1 + \frac{V}{UA - V} \cdot \frac{A - L(u)}{L(u)} \cdot \frac{U - L(x)}{L(x)}} \quad (9)$$

To intuitively validate our formula let us analyze a few extreme cases:

- *For a user u that likes all items, p(u,x) should always be 1.* To like all items implies that *L(u) = A*. Therefore the right side of the denominator in (9) becomes zero and *p(u,x)* is *1* as expected.
- *For a user that does not like any item, p(u,x) should always be zero.* To not like any items implies that *L(u)* is empty. Therefore the right side of the denominator in (9) becomes infinite and *p(u,x)* is zero as expected.
- Similar verifications apply to items that are liked by all or by none.

3.1.3 *Similarity Between Two Items*

What is the expected number of users who like *y* to also like *x*? *L(y)* is the set of users that like *y*. Each of these users likes *x* with a probability *p* defined in equation (9). *T*he expected number of users who like *y* to also like *x* is:

$$e(x,y) = \sum_{u \in L(y)} p(u,x) \quad (10)$$

e(x,y) is not symmetrical: *e(x,y)* is summed over *L(y)* while *e(y,x)* is summed over *L(x)*.

The *actual* number of users who like *y* and also like *x* is ("co-occurrences"):

$$c(x,y) = \|L(x) \cap L(y)\| \quad (11)$$

We define the similarity from *y* to *x* as the ratio between the actual and the expected:

$$sim(x,y) = \frac{c(x,y)}{e(x,y)} \quad (12)$$

Therefore the similarity of an item *y* to a target item *x* is:

> **How much more than expected are you to like *x* if you liked *y*.**

This definition not only identifies similarity *(sim >> 1)* but also differentiates between "unrelated" *(sim ~ 1)* and "antagonistic" *(sim << 1)*.

Finally, the similarity of an item *y* to a target item *x* is:

$$sim(x,y) = \frac{c(x,y)}{\sum_{u \in L(y)} p(u,x)} \quad (13)$$

With *p(u,x)* defined in equation (9).

3.2 Personalized Recommendations Algorithm

The novel part of this paper is in the probabilistic definition of item similarity described in the previous section. To be able to test our algorithm, we needed to produce personalized recommendations.

There are many ways to determine personalized recommendations once an item similarity metric is available. Due to our probabilistic definition of item similarity (i.e., "how much more than expected are you to like x if you liked y"), we decided to explore new algorithms.

3.2.1 *Accumulation of Evidence*

We found in the previous section the probability that *u* likes *x* (under the assumption that likeness is randomly distributed): *p(u,x)* in equation (9).

But if we know that *u* likes some other item *y* (that is, *y* belongs to *L(u)),* then, using our definition of similarity, the expectation that *u* will like *x* becomes:

$$l(u,x) = p(u,x) \cdot sim(x,y) \quad (14)$$

That is, the "probability to like x" times "how much more than expected are you to like x if you liked y".

The question now is how to combine (14) for all *y* in *L(u).* A similarity score of 1 implies no relatedness between two items (ratio between actual co-occurrences and expected is 1). Therefore the "evidence" of similarity is:

$$sim(x,y) - 1$$

The accumulated evidence for user *u* is:

$$\sum_{y \in L(u) | sim(x,y) > 1} (sim(x,y) - 1)$$

The expectation that *u* likes *x* becomes:

$$l(u,x) = p(u,x) \cdot \left(1 + \sum_{y \in L(u) | sim(x,y) > 1} (sim(x,y) - 1)\right) \quad (15)$$

3.2.2 *Applying Dislikes*

Lastly, we want to incorporate the known dislikes into our expectation formula. Let *r(u,x)* be some rating of *u* toward *x* that is positive for likes and negative for dislikes. For example, in a 5-star rating system *r* can be defined as:

$$\begin{cases} r(u,x) = 0 & if\ no\ rating\ is\ available \\ r(u,x) = \dfrac{rating(u,x) - 2.5}{2.5} & otherwise \end{cases}$$

With our Video-on-Demand data-set (no ratings), we define the likes/dislikes based on percentage watched (discussed in the next section).

A natural generalization of the expectation using *r(u,x)* is then:

$$l(u,x) = p(u,x) \cdot \left(1 + \sum_{y | sim(x,y) > 1} r(u,y)(sim(x,y) - 1)\right) \quad (16)$$

The summation is done over all items consumed by *u* (including likes and dislikes).

The ranking of items by their expectation produces recommendations that are personalized to the user *u*.

4. VOD DATASET

The dataset used in our main experiment consisted of a subset of one year of Video-on-Demand usage data collected from Comcast set top boxes. Some statistics on our subset:

- 131 geographical markets
- 1.5 billion events
- 19 million users
- 10 thousand videos
- 100 GB uncompressed

Each data file contains events that represent a user watching a video item. More precisely, an event is made of:

- The user
- The video
- The timestamp
- The total/watched duration

The "user" is an encrypted cable set-top box ID, but because set-top boxes are frequently shared across a household, our definition of user might not represent the activity of a single person.

The video ID for an episode is actually the series ID (not the particular episode ID). For example, if a customer watched 50 episodes of the series *Dexter*, the 50 events will all have the same item ID (that is, the ID for the *Dexter* series).

The timestamp is the time, local to the user, when the video was watched.

We compute the fraction watched using the total duration of the video in minutes and the number of minutes that were actually watched. If the same video is watched multiple times by the same user, we add up the fractions.

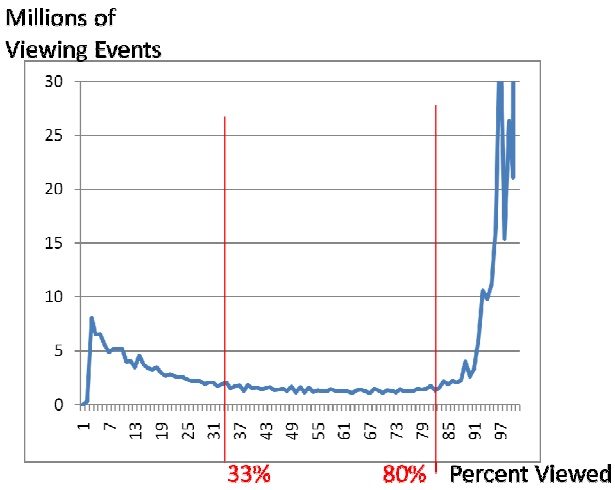

Figure 3. *Histogram of viewing events based on the fraction watched (VOD data-set). We define a like as percent viewed > 80% and a dislike as percent viewed < 33%.*

Based on the histogram of the fraction watched shown in figure 3, we define a confident dislike as viewing less than 33% of a video and a confident like as viewing more than 80% (some credit rolls are rather lengthy). We assume that the area in the middle, between 33 and 80 percent, where the line is horizontal and therefore independent of the percent watched, is mostly due to real-life events and therefore inconclusive.

We separate the usage data into a test set and a training set. Out of the 1.5 billion events, we extracted more than 600 million aggregated like and dislike events. We randomly selected 10,000 users and extracted 25% of their events for the test set then used the rest for the training set.

Our rules for measuring the quality of a personalized recommendations system were as follows:

- Each system trains with the training set and has no knowledge of the test set.
- Each system is expected to return a ranked list of top 100 personalized recommendations for each user.
- The recall is the fraction of the like events in the test set that were correctly detected.
- The false positive rate is defined as the fraction of the dislike events in the test set that were wrongly detected.
- The precision is obtained by dividing the number of known good recommendations by the sum of the number of known good plus known bad recommendations (known good = present in the test set and liked; known bad = present in the test set and disliked).

5. EXPERIMENTS

The main experiment is with the complete Video on Demand data-set and involves comparison against multiple vendors. Experiments in section 5.2 use the MovieLens data set and is aimed at a more thorough comparison against Mahout.

5.1 Main Experiment

We performed experiments with the Video-on-Demand data set described in the previous section. For this data-set, in the last phase of our ProbSim algorithm, we used the following definition for the function *r(u,x)*:

$$\begin{cases} r(u,x) = 1 & for\ a\ like \\ r(u,x) = -0.3 & for\ a\ dislike \\ r(u,x) = -0.1 & otherwise \end{cases}$$

Several top commercial providers of recommender systems were invited to apply their algorithms as well. Due to confidentiality concerns, the identities of the participating recommendations solution providers must not be revealed.

As a baseline we used the not-personalized list of the top 100 most popular items (after removing, for each user, the events that are already in the training set). We also applied the co-occurrences algorithm from the public domain Mahout system.

One co-winner, VENDOR_B, used a matrix factorization model to calculate recommendations by mapping users and items to a low dimensional space.

The other co-winner, VENDOR_A, used both usage data and high-quality metadata about items to cluster users and items, and to define semantic vectors for each.

The figures below show the results of this experiment.

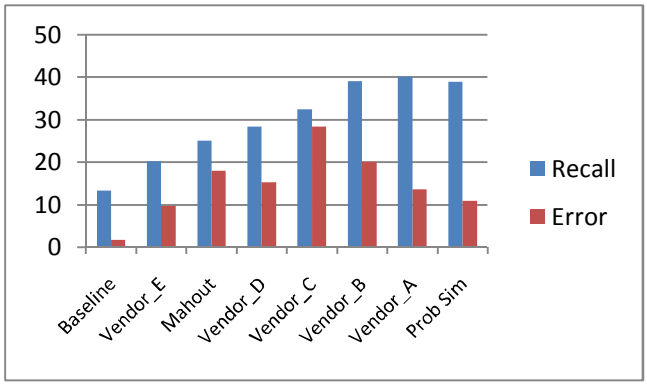

Figure 4. *Recall percentage and error percentage (false positives) for top 100 recommendations.*

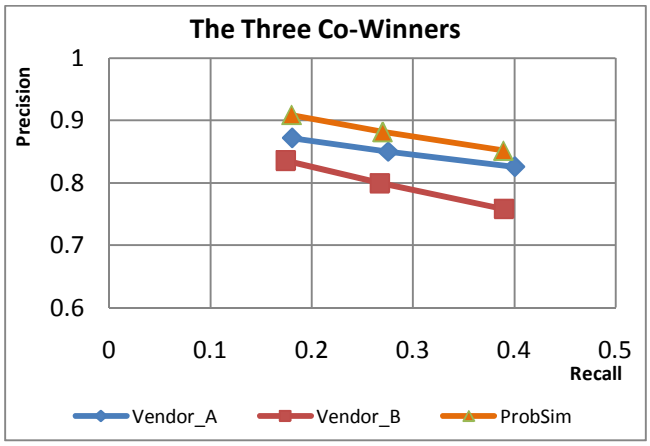

Figure 5. *Precision and recall of the three co-winners: Prob-Sim, Vendor_A (semantic vectors), and Vendor_B (matrix factorization) for top 25, top 50, and top 100 recommendations.*

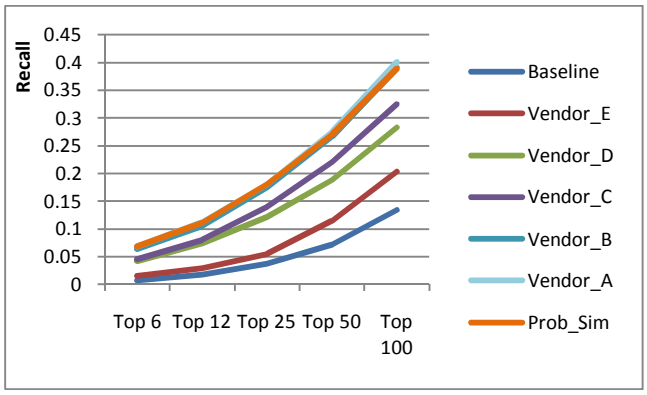

Figure 6. *Recall percentage at various top N recommendation levels.*

5.2 MovieLens Experiment

For our second experiment we selected the 10 million ratings data set from MovieLens[26]. The ratings vary from 1 to 5. This data set has 10 thousand movies and 71 thousand users. We ran their script *split_ratings.sh* and selected the files *ra.train* for training and *ra.test* for testing. The test file has 10 ratings per user (and these ratings are not present in the training file).

We used the Apache Mahout[27] project that provides free implementations of machine learning algorithms. We ran several Mahout collaborative filtering algorithms and found that the following four algorithms had the best results: Loglikelihood, Cooccurrences, Cosine, and Jaccard. We added our new algorithm Prob-Sim to the lot and followed the procedure outlined below.

- Each system trains with the training set and has no knowledge of the test set.
- Each system returns a ranked list of top 100 personalized recommendations for each user.
- The known good recommendations are the subset of test set ratings with a score >= 3 that were correctly recommended. The recall is the fraction of the test set that are known good recommendations.
- The known bad recommendations are the subset of the test set ratings with a score <= 2 that were wrongly recommended.
- The precision is obtained by dividing the number of known good recommendations by the sum of the number of known good plus known bad recommendations.

For the Movie Lens data-set, in the last phase of our ProbSim algorithm, we used the following definition for the function *r(u,x)*:

$$r(u,x) = \frac{rating(u,x) - 2.5}{2.5}$$

The comparison of Prob-Sim with the four Mahout algorithms is shown in figure 7.

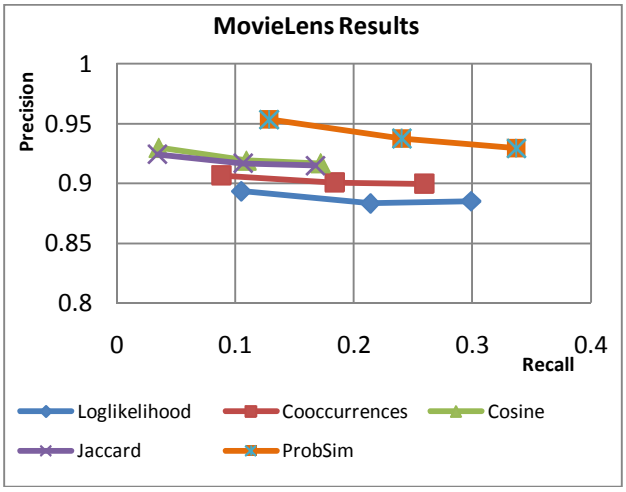

Figure 7. *Prob-Sim vs. four Mahout algorithms at top 20, top 50, and top 100 recommendations per user on the Movie Lens dataset. This diagram shows that Prob-Sim is significantly better for precision and for recall.*

6. DISTRIBUTED IMPLEMENTATION

We implemented ProbSim on a distributed platform using the Apache Hadoop MapReduce framework. MapReduce distributes the similarity computation in parallel across all nodes in a cluster.

In order to make the algorithm efficient and reduce the amount of bytes moved around by the Hadoop framework, we create indexes for items and users (a user/item is manipulated as an integer from

0 to the number of users/items – *1*). We also attempt to minimize the key space as more keys emitted by mappers result in longer execution times and lower throughput. One example of key space minimization: instead of the elegant

$$(x, y) \rightarrow sim$$

we use the optimized

$$y \rightarrow [(x, sim)]$$

Unless you have a large cluster MapReduce will likely be I/O bound. In our 4-node cluster, each node with 32GB RAM and 2 AMD Opteron 2.9GHz quad core processors, the total execution time dropped by one order of magnitude after implementing the key space minimization. The disadvantage to this approach is that the implementation code becomes more complex.

The total processing time of the Video-on-Demand data set with 600 million viewing events is approximately 1 hour. The total processing time of the MovieLens data-set with 10 million ratings is approximately 10 minutes.

Table 1 lists the steps of our job pipeline.

Step	*Job*	*VOD*	*MovieLens*
1	item-popularity	2m29s	1m16s
2	item-co-occurrences	15m24s	3m41s
3	item-ucounts	2m53s	1m16s
4	item-similarities	15m45s	1m33s
5	user-topN	26m11s	2m
	Total	1h2m42s	9m46s

Table 1. *Steps of our MapReduce pipeline and shows the execution time of each step for the VOD and MovieLens data-sets.*

The sub-sections below describe the steps in Table 1.

6.1 item-popularity:

The first step determines the popularity of each item.

Map input	$u_i \rightarrow [(a_y, r_{iy})]$	The usage data set
Map output	$a_y \rightarrow 1$	$if\ r_{iy} > 0$
Reducer input	$a_y \rightarrow [1]$	
Reducer output	$a_y \rightarrow L_y$	$L_y = sum[1]$

$L_y=L(a_y)$ is described in the algorithm section (to lighten the notation we again avoid the norm sign $|L(a_n)|$).

6.2 item-co-occurrences:

For each item pair, determine the known number of users that like both.

Map input	$u_i \rightarrow [(a_j, r_{ij})]$	The usage data set
	$L = \{a_j \mid r_{ij} > 0\}$	Liked subset
Map output	$a_m \rightarrow [(a_n, 1)]$	$a_m, a_n \in L$
Reduce input	$a_m \rightarrow [[(a_n, 1)]]$	
Reducer output	$a_m \rightarrow [(a_n, c_{nm})]$	C_{nm} = co-occurrences = number of 1 associated with a_n

$c_{mn} = c(a_m, a_n)$ is described in the algorithm section.

6.3 item-ucounts:

This step concatenates the number of likes for all users that liked an item.

Map input	$u_i \rightarrow [(a_j, r_{ij})]$	The usage data set
	$L = \{a_j \mid r_{ij} > 0\}$	Liked subset
	$L_i = \|\|L\|\|$	Number of likes
Map output	$a_j \rightarrow L_i$	
Reduce input	$a_j \rightarrow [L_i]$	
Reduce output	$a_j \rightarrow [L_i]$	List of user like counts

6.4 item-similarities

For each item pair, compute the probabilistic similarity.

$L_x=L(a_x)$ for all items is made available through the Hadoop distributed cache.

Two parameters are used to minimize the size of the similarity matrix:

- *MIN* is the similarity threshold (lower similarities are not output).
- *NMAX* is the maximum number of similars per item.

Reduce input 1	$a_y \rightarrow [(a_x, c_{xy})]$	Co-occurrences
Reduce input 2	$a_y \rightarrow [L_i]$	List of user like counts
	$f_i = \frac{A - L_i}{L_i}$	Necessary for the computation of $p_{ix}=p(u_i,a_x)$
	$f_x = \frac{U - L_x}{L_x}$	Necessary for the computation of $p_{ix}=p(u_i,a_x)$.
	$p_{ix} = \frac{1}{1 + \pi f_i f_x}$	
	$e_{xy} = \sum_{u_i} p_{ix}$	Expected number of users from a_y to like a_x as well
	$sim_{xy} = \frac{c_{xy}}{e_{xy}}$	
Reduce output	$a_y \rightarrow [sim_{xy}]$	Similarity vector to all items. If $sim_{xy} > MIN$ *Top NMAX only*

6.5 user-topN

This is the final step: we compute 100 personalized recommendations per user. The candidate similarity matrix $csim_{xy}$ and L_x are made available through the Hadoop distributed cache.

Map input	$u_i \rightarrow [(a_y, r_{iy})]$	The usage data set
	$L_i = \|\{r_{iy} \mid r_{iy} > 0\}\|$	The user like count
	$f_i = \frac{A - L_i}{L_i}$	Necessary for p_{ix}
	$a_y \rightarrow [(a_x, csim_{xy})]$	Available from distributed cache
	$f_x = \frac{U - L_x}{L_x}$	Necessary for p_{ix}
	$p_{ix} = \frac{1}{1 + \pi f_i f_x}$	
	$l_{ix} = p_{ix} \cdot \left(1 + \sum_y (csim_{xy} - 1)\right)$	The expectation that u_i will like a_x
Map output	$u_i \rightarrow [(a_x, l_{ix})]$	

7. FUTURE WORK

To reduce the sparseness problem with the original usage data set, we plan to try a second round of probabilistic item similarity matrix computation, where the items take the role of users and the ratings *r(u,x)* (or like/dislikes) are replaced with similarities *sim(x,y)*.

The probabilistic item similarity computation could be attempted on video metadata instead of usage data, maybe even on a combination of both.

8. CONCLUSIONS

This new item-based collaborative filtering algorithm has proven very accurate in computing recommendations.

On the real-world usage data collected at Comcast it reached a comparable level of quality as more complex algorithms such as matrix factorization. The results of the MovieLens experiment demonstrate that we perform significantly better than known item-based algorithms. These are exciting results from a simple probabilistic definition of item similarity.

With item similarity being based on probabilities, there are no parameters to experiment with. This prevents the potential tendency of over-fitting.

9. REFERENCES

1. Gediminas Adomavicius, I. and I. Alexander Tuzhilin, *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions.* Knowledge and Data Engineering, IEEE Transactions on, 2005. **17**(6): p. 734-749.
2. Melville, P. and V. Sindhwani, *Recommender Systems.* Encyclopedia of Machine Learning, 2010.
3. Resnick., P. and H.R. Varian, *Recommender systems.* Communications of the ACM, 1997. **40**(3).
4. Miller, B.N., et al., *MovieLens unplugged: experiences with an occasionally connected recommender system*, in *Proceedings of the 8th international conference on Intelligent user interfaces* 2003: New York, NY.
5. Bambini, R., P. Cremonesi, and R. Turrin, *A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment.* Recommender Systems Handbook, 2011: p. 299-331.
6. Herlocker, J.L., et al., *Evaluating collaborative filtering recommender systems.* ACM Transactions on Information Systems (TOIS), 2004. **22**(1).
7. Su, X. and T.M. Khoshgoftaar, *A survey of collaborative filtering techniques.* Advances in Artificial Intelligence, 2009.
8. Cantador, I., A. Bellogín, and D. Vallet. *Content-based recommendation in social tagging systems*. in *Proceedings of the fourth ACM conference on Recommender systems*. 2010. Barcelona.
9. Musto, C., *Enhanced vector space models for content-based recommender systems*, in *Proceedings of the fourth ACM conference on Recommender systems*. 2010: Bari, Italy.
10. Lekakos, G. and P. Caravelas, *A hybrid approach for movie recommendation.* MULTIMEDIA TOOLS AND APPLICATIONS, 2008. **36**(1-2): p. 55-70.
11. Karypis, G. *Evaluation of Item-Based Top-N Recommendation Algorithms*. in *Proceedings of the tenth international conference on Information and knowledge management*. 2001. New York, NY.
12. Sarwar, B., et al. *Item-based collaborative filtering recommendation algorithms*. in *Proceedings of WWW '01 Proceedings of the 10th international conference on World Wide Web*. 2001. New York, NY.
13. Liang, H., et al. *Collaborative Filtering Recommender Systems Using Tag Information*. in *International Conference on Web Intelligence and Intelligent Agent Technology*. 2008.
14. Gong, S., *An Efficient Collaborative Recommendation Algorithm Based on Item Clustering.* ADVANCES IN WIRELESS NETWORKS AND INFORMATION SYSTEMS, 2010. **72**.
15. Jiang, F. and Z. Wang, *Pagerank-Based Collaborative Filtering Recommendation.* INFORMATION COMPUTING AND APPLICATIONS, 2010. **6377**.
16. Schlieder, C., *Modeling Collaborative Semantics with a Geographic Recommender.* ADVANCES IN CONCEPTUAL MODELING – FOUNDATIONS AND APPLICATIONS, 2007. **4802**.
17. Ahn, H.J., *A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem.* Information Sciences, 2008. **178**(1): p. 37-51.
18. Anand, D. and K.K. Bharadwaj, *Utilizing various sparsity measures for enhancing accuracy of collaborative recommender systems based on local and global similarities.* Expert Systems with Applications, 2011. **38**(5).
19. Wang, J., et al., *Probabilistic relevance ranking for collaborative filtering.* INFORMATION RETRIEVAL, 2008. **11**(6).
20. Töscher, A., M. Jahrer, and R. Legenstein. *Improved neighborhood-based algorithms for large-scale recommender systems*. in *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*. 2008. New York, NY.
21. Koren, Y. *Factorization meets the neighborhood: a multifaceted collaborative filtering model*. in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008. New York, NY.
22. Gunawardana, A. and G. Shani, *A Survey of Accuracy Evaluation Metrics of Recommendation Tasks.* The Journal of Machine Learning Research, 2009. **10**.
23. Ormándi, R., I. Hegedűs, and M. Jelasity, *Overlay Management for Fully Distributed User-Based Collaborative Filtering* Lecture Notes in Computer Science, 2010. **6271**.
24. Zhao, Z.-D. and M.-s. Shang. *User-based collaborative-filtering recommendation algorithms on hadoop*. in *Third International Conference on Knowledge Discovery and Data Mining*. 2010. Phuket, Thailand
25. Fu, C. and Z. Leng. *A Framework for Recommender Systems in E-Commerce Based on Distributed Storage and Data-Mining*. in *2010 International Conference on E-Business and E-Government*. 2010. Guangzhou, China
26. *GroupLens Research*. Available from: http://www.grouplens.org/node/12.
27. *Apache Foundation*. Available from: http://mahout.apache.org/.