TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture

Kamal Ali
TiVo, Yahoo!
701 First Avenue
Sunnyvale, CA 94089
+1 408 349 7931
kamal@yahoo-inc.com

Wijnand van Stam
TiVo
2160 Gold Street
Alviso, CA 95002
+1 408 519 9100
wijnand@tivo.com

ABSTRACT

We describe the TiVo television show collaborative recommendation system which has been fielded in over one million TiVo clients for four years. Over this install base, TiVo currently has approximately 100 million ratings by users over approximately 30,000 distinct TV shows and movies. TiVo uses an item-item (show to show) form of collaborative filtering which obviates the need to keep any persistent memory of each user's viewing preferences at the TiVo server. Taking advantage of TiVo's client-server architecture has produced a novel collaborative filtering system in which the server does a minimum of work and most work is delegated to the numerous clients. Nevertheless, the server-side processing is also highly scalable and parallelizable. Although we have not performed formal empirical evaluations of its accuracy, internal studies have shown its recommendations to be useful even for multiple user households. TiVo's architecture also allows for throttling of the server so if more server-side resources become available, more correlations can be computed on the server allowing TiVo to make recommendations for niche audiences.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms.

Keywords

Collaborative-Filtering, Clustering Clickstreams

1. INTRODUCTION

With the proliferation of hundreds of TV channels, the TV viewer is faced with a *search* task to find the few shows she would like to watch. There may be quality information available on TV but it may be difficult to find. Traditionally, the viewer resorts to "channel surfing" which is akin to random or linear search. The mid 1990's in the USA saw the emergence of the TV-guide channel which listed showings on other channels: a form of indexing the content by channel number. This was followed in the late 1990's by the emergence of TiVo. TiVo consists of a television-viewing service as well as software and hardware. The client-side hardware consists of a TiVo set-top box which records TV shows on a hard-disk in the form of MPEG streams. Live TV and recorded shows are both routed through the hard-disk, enabling one to pause and rewind live TV as well as recordings. TiVo's main benefits are that it allows viewers to watch shows at times convenient for the viewer, convenient digital access to those shows and to find shows using numerous indices. Indices include genre, actor, director, keyword and most importantly for this paper: the probability that the viewer will like the show as predicted by a collaborative filtering system.

The goal of a recommender system (an early one from 1992 is the Tapestry system [10]) is to predict the degree to which a user will like or dislike a set of items such as movies, TV shows, news articles or web sites. Most recommender systems can be categorized into two types: content-based recommenders and collaborative-based recommenders. Content-based recommenders use features such as the genre, cast and age of the show as attributes for a learning system. However, such features are only weakly predictive of whether viewers will like the show: there are only a few hundred genres and they lack the specificity required for accurate prediction.

Collaborative filtering (CF) systems, on the other hand, use a completely different feature space: one in which the features are the other viewers. CF systems require the viewer for whom we are making predictions - henceforth called the "active viewer" - to have rated some shows. This 'profile' is used to find like-minded viewers who have rated those shows in a similar way. Then, for a show that is unrated by the active viewer, the system will find how that show is rated by those similar viewers and combine their ratings to produce a predicted rating for this unrated show. The idea is that since those like-minded viewers liked this show, that the active viewer may also like it. Since other viewers' likes and dislikes can be very fine-grained, using them as features rather than content-based features leads to recommendations not just for the most popular shows but also for niche shows and audiences. We use a Bayesian content-based filtering system to overcome the "cold-start" problem (making predictions for new shows) for shows for which we do not have thumbs data from other users.

The rest of the paper is organized as follows. In Section 2, we describe previous work on collaborative filtering systems, specially those making TV or movie recommendations. We outline the issues and the spectrum of approaches used to address

KDD'04, August 22-25, 2004, Seattle Washington, USA.

those issues. Section 3 gives more detail on the flow of data starting with a viewer rating a show and culminating in the TiVo client making a recommendation to the viewer for a new show. Section 4 explains the performance algorithm: how the server computes correlations between pairs of shows and how each client uses those correlations to make show suggestions for its viewer. Section 5 details the server-side learning needed to support computation of the correlations. Section 6 outlines future challenges and directions and we conclude by summarizing the state of this massively fielded system.

2. PREVIOUS WORK

Recommender systems have been around since the Tapestry [10] system from 1992 which relied on a small community of users to make recommendations for each other. However, the first system we know of for making movie recommendations was the Bellcore Video Recommender system from 1995 [11]. Recommender systems can be categorized into content-based filtering systems and collaborative-filtering systems. In turn, collaborative filtering systems can be categorized along the following major dimensions:

1. "User-user" or "item-item" systems: In user-user systems, correlations (or similarities or distances) are computed between users. In item-item systems (e.g. TiVo and [15]), metrics are computed between items.

2. **Form of the learned model:** Most collaborative filtering systems to date have used *k*-nearest neighbor models (also referred to as Memory-based systems or Instance-based systems) in user-user space. However there has been work using other model forms such as Bayesian networks [3], decision trees [3], cluster models [3], "Horting" (similarity over graphs) [1] and factor analysis [4].

3. **Similarity or distance function:** Memory-based systems and some others need to define a distance metric between pairs of items (or users). The most popular and one of the most effective measures used to date has been the simple and obvious Pearson product moment correlation coefficient (e.g. [9]) as shown in Equation 1 below. As applied for item-item systems, the equation computes the correlation *r* between two items (e.g. shows) : *s1* and *s2*. The summation is over all *N* users *u* that rated *both* items (shows). t_{s1u} is the rating given by user *u* to show *s1*. T_{s1} is the average rating (over those *N* users) given to *s1*. σ_{s1} is the standard deviation of ratings for *s1*. Similar definitions apply for the other show *s2*.

$$r = \frac{1}{N}\sum_{u=1}^{N}\frac{t_{s1u} - T_{s1}}{\sigma_{s1}} * \frac{t_{s2u} - T_{s2}}{\sigma_{s2}}$$

Equation 1. Pearson correlation metric

Other distance metrics used have included the cosine measure (which is a special case of the Pearson metric where means are zero) and extensions to the Pearson correlation which correct for the possibility that one user may rate programs more or less harshly than another user. Following in the track of TFIDF (Term-Frequency Inverse Document-Frequency) as in [14], another extension gives higher weight to users that rate infrequently; this is referred to as the 'inverse frequency' approach. Another approach is 'case amplification' in which a non-linear function is applied to the ratings so that a rating of +3, for example, may end up being much more than three times as much as a rating of +1.

4. **Combination function:** Having defined a similarity metric between pairs of users (or items), the system needs to make recommendations for the active user for an unrated item (show). Memory-based systems typically use the *k*-nearest neighbor formula (e.g. [8]). Equation 2 shows how the *k*-nearest neighbor formula is used to predict the rating t_s for a show *s*. The predicted rating is a weighted average of the ratings of correlated neighbors $t_{s'}$, weighted by the degree $r_{s,s'}$ to which the neighbor *s'* is correlated to *s*. Only neighbors with positive correlations are considered.

$$t_s = \frac{1}{\sum_{s \in S} |r_{s,s'}|}\sum_{s \in S} r_{s,s'} * t_{s'}$$

Equation 2. Weighted linear average computation for predicted thumbs level

In item-item systems, the *k-nearest neighbor*s are other items which can be thought of as points embedded in a space whose dimensions or axes correspond to users. In this view, equation 1 can be regarded as computing a distance or norm between two items in a space of users. Correspondingly, in user-user systems, the analog of equation 1 would be computing the distance between two users in a space of items.

Bayesian networks have also been used as combination functions; they naturally produce a posterior probability distribution over the rating space and cluster models produce a density estimate equivalent to the probability that the viewer will like the show. *K-nearest neighbor* computations are equivalent to density estimators with piecewise constant densities.

5. **Evaluation criterion:** The accuracy of the collaborative filtering algorithm may be measured either by using mean absolute error (MAE) or a ranking metric. Mean absolute error is just an average, over the test set, of the absolute difference between the true rating of an item and its rating as predicted by the collaborative filtering system. Whereas MAE evaluates each prediction separately and then forms an average, the ranking metric approach directly evaluates the goodness of the entire ordered list of recommendations. This allows the ranking metric approach to, for instance, penalize a mistake at rank 1 more severely than a mistake further down the list. Breese *et al.* [3] use an exponential decay model in which the penalty for making an incorrect prediction decreases exponentially with increasing rank in the recommendation list. The idea of scoring a ranked set is similar to the use of the "DCG" measure (discounted cumulative gain [12]), used by the information-retrieval community to evaluate search engine recommendations. Statisticians have also evaluated ranked sets using the Spearman rank correlation coefficient [9] and the kappa statistic [9].

Now we summarize the findings of the most relevant previous work. Breese et al found that on the EachMovie data set [6], the best methods were Pearson correlation with inverse frequency and "Vector Similarity" (a cosine measure) with inverse frequency. These methods were statistically indistinguishable. Since Pearson correlation is very closely related to the cosine approach, this is not surprising. Case amplification was also found to yield an

added benefit. On this data set, the clustering approach (mixture of multinomial models using AutoClass [5]) was the runner-up followed by the Bayesian-network approach. The authors hypothesize that these latter approaches worked relatively poorly because they needed more data than was available in the EachMovie set (4119 users rating 1623 shows with the median number of ratings per show at 26). Note that for the EachMovie data set, the true rating was a rating from 0 to 5, whereas the data sets on which Bayesian network did best only required a prediction for a binary random variable.

Sarwar *et al.* [15] explore an item-item approach. They use a k-nearest neighbor approach, using Pearson correlation and cosine. Unfortunately for comparability to TiVo, their evaluation uses MAE rather than a ranking metric. Since TiVo presents a ranked list of suggestions to the viewer, the natural measure for its recommendation accuracy is the ranking metric, not MAE. Sarwar *et al.* find that for smaller training sizes, item-item has lower predictive error on the MovieLens data set than user-user but that both systems asymptote to the same level. Interestingly, they find the minimum test error is obtained with k=30 in the *k*-nearest neighbor computation on their data set. They find again that cosine and Pearson perform similarly but the lowest error rate is obtained by their version of cosine which is adjusted for different user scales. The issue of user-scales is that even though all users may be rating shows over the same rating scale, one user may have a much narrower variance than another, or one may have a different mean than another. User-scale correction will normalize these users' scores before comparing them to other users.

Another issue facing recommendation systems is the speed with which they make recommendations. For systems like MovieLens which need to make recommendations in real-time for users connected via the Web, speed is of the essence. The TiVo architecture obviates this by having the clients make the recommendation instead of the server. Furthermore, each TiVo client only makes recommendations once a day in batch for all upcoming shows in the program guide whereas some collaborative systems may change their recommendations in real-time based upon receiving new ratings from other users.

Lack of space prevents us from discussing work on privacy preservation (e.g. [4]), explainability[1] and the cold-start issue: how to make predictions for a new user or item.

3. BACKGROUND ON TIVO

In this section we describe the flow of data that starts from a user rating a show to upload of ratings to the server, followed by server-side computation of correlations, to download and finally to the TiVo client making recommendations for new shows.

Every show in the TiVo universe has a unique identifying series ID assigned by Tribune Media Services (TMS). Shows come in two types: movies (and other one-off events) and series which are recurring programs such as 'Friends'. A series consists of a set of *episodes*. All episodes of a series have the same series ID. Each movie also has a "series" ID. Prediction is made at the series level so TiVo does not currently try to predict whether you will like one episode more than another.

The flow of data starts with a user rating a show. There are two types of rating: explicit and implicit. We now describe each of these in turn.

Explicit feedback: The viewer can use the thumbs-up and thumbs-down buttons on the TiVo remote control to indicate if she likes the show. She can rate a show from +3 thumbs for a show she likes to -3 thumbs for a show she hates (we will use the notation -n thumbs for n presses of the thumbs-down button). When a viewer rates a show, she is actually rating the series, not the episode. Currently (Jan. 2004), the average number of rated series per TiVo household is 98. Note that TiVo currently does not build a different recommendation model for each distinct viewer in the household – in the remainder of the paper we will assume there is only one viewer per household.

Implicit feedback: Since various previous collaborative filtering systems have noted that users are very unlikely to volunteer explicit feedback [13], in order to get sufficient data we decided that certain user actions would *implicitly* result in that program getting a rating. The only forms of explicit feedback are pressing the thumbs-up and thumbs-down buttons: all other user actions are candidates for implicit feedback. Currently, the only user action that results in an implicit rating happens when the user choose to record a previously unrated show. In this event, that show is assigned a thumbs rating of +1. Currently the prediction algorithms do not distinguish the +1 rating originating from explicit user feedback (thumbs-up button) from this implicit +1 rating. Note that it is not clear if the +1 rating from a requested recording is stronger or weaker evidence than a +1 rating from a direct thumbs event. Viewers have been known to rate shows that they would like to be known as liking but that they don't actually watch. Some viewers may give thumbs up to high-brow shows but may actually schedule quite a different class of show for recording. One user action that could be an indicator even better than any of these candidates is 'number of minutes watched.'
Other possible user actions that could serve as events for implicit feedback are selection of a 'season pass', deletion, promotion or demotion of a season pass. A 'season pass' is a feature by which the viewer indicates that she wants to record multiple upcoming episodes of a given series.

The following sequence details the events leading to TiVo making a show suggestion for the viewer:

1. **Viewer feedback**: Viewer actions such as pressing the thumbs buttons and requesting show recordings are noted by the TiVo client to build a profile of likes and dislikes for each user.
2. **Transmit profile**: Periodically, each TiVo uploads its viewer's entire thumbs profile to the TiVo server. Indeed, the *entire* profile rather is sent, rather than an incremental upload because we designed the server so that it does not maintain even a persistent anonymized ID for the user. As a result the server has no way of tying together incremental thumbs profiles and so it becomes necessary to upload the entire profile each time. We have found uploading the entire

[1] An example of a collaborative-filtering system employing explainability is the Amazon.com feature: "Why was I recommended this". This feature explains the recommendation in terms of previously rated or bought items.

profile does not constitute an onerous burden on the upload time.

3. **Anonymization**: The server anonymizes the thumbs profile log, resetting the file transfer date to Jan. 1 1970 so transfer time cannot be used to trace which TiVo the file came from.
4. **Server-side computation**: The server computes pair-wise correlations between *series* for series "of interest". Next, it prepares a package consisting of correlations pairs for download to TiVos. See Section 5 for more detail on which series are deemed "of interest".
5. **Correlation download**: Correlation pairs are downloaded to certain TiVos. To reduce download stress for the network, the download is round-robin'd across all TiVos so that each TiVo only receives new correlation pairs every N days. Each TiVo currently receives a download of 320KB (uncompressed) consisting of twenty-eight thousand correlation pairs, averaging 11.5 bytes per pair.
6. **Client-side computation**: The collaborative-filtering algorithm on each TiVo iterates through its program guide, using the download package to make predictions for unrated shows. Note that each TiVo makes recommendations over a different set of programs, determined by which channels are received at that TiVo and which series have not been rated at that TiVo. Whereas most collaborative filtering systems have to worry about server response time for making recommendations to all connected users, because the TiVo clients make all the recommendations, this task does not tax the server.
7. **Suggestions List:** The collaborative predictions are combined with the content-based predictions and already-rated shows to assemble a list of shows that TiVo thinks the viewer would like to watch (Figure 1).
8. **Inferred-recordings:** If there is enough space on the hard disk and if there are not many shows explicitly requested by the viewer to be recorded in the short-term, TiVo will use that spare space and time to record the show highest in the suggestions list that will fit in the available space and time. The actual policy for recording is complex and ensures that such an inferred-recording will never replace an explicit recording on disk.

Privacy is preserved by this architecture in a number of ways. There is no persistent (upload to upload) storage of any user object on the server since we are computing show to show correlations instead of the usual user-user approach. Not even an anonymized version (hence necessitating the upload of full thumbs profile) is maintained at the server. The communications between the server and client are encrypted. Once the TCP-IP connection is broken between server and client, the log file name is anonymized so that the server no longer knows which TiVo the log came from. The time and date-stamp of thumbs logs are also reset to Jan 1, 1970 to preclude anyone from correlating thumbs logs to error logs. All these measures made it quite difficult for us to debug upload errors during development!

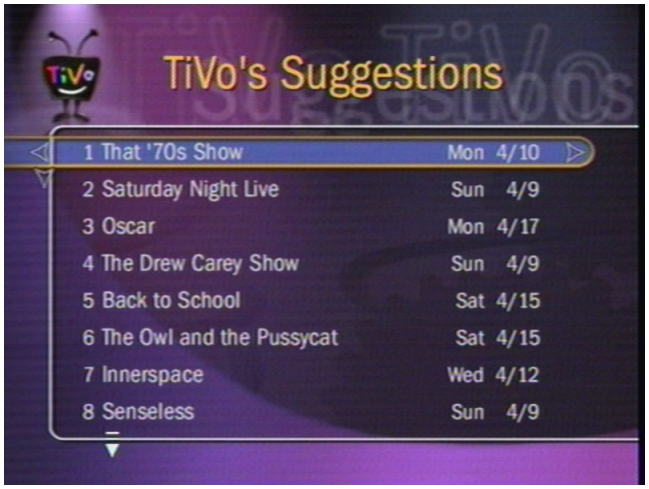

Figure 1. Sorted list of recommended shows

4. PERFORMANCE TASK

In this section we will describe the prediction performance task: to create a list of upcoming shows sorted by the predicted degree to which the user will like the show. This list is used to populate the Suggestions List that appears in the TiVo interface (Figure 1). It is also used by the program scheduler to decide which suggested shows should be recorded if there is unused space on the disk.

TiVo uses two algorithms for predicting how much the viewer will like the show: a Bayesian content-based filtering algorithm and the collaborative-filtering algorithm described in this paper. It is necessary to augment collaborative-filtering with such a content-based approach to address the cold-start problem [Schein]: namely the situation in which for new users and shows there is insufficient correlation data. Features for content-based filtering include the genres of the show and the cast: actors, directors etc.

In order to construct this list, TiVo first removes from consideration episodes that the user has explicitly requested either as a single recording or as a season pass since obviously we do not want to recommend a show already scheduled for recording. The remaining episodes are considered as 3-tuples as follows: <seriesID, thumbs, confidence>

Note that we have mapped from episode to series here. If there is more than one upcoming episode for a series, the first upcoming one will be used as the representative for the series. In the three-tuple, thumbs is an object with value from -3 to +3 and the confidence is a integer from 0 to 255. The series are sorted first by thumbs and then by confidence. In Figure 1, series that have rated by the user at +n thumbs appear with n thumbs-up icons. Series that are *predicted* by the TiVo to have +3 thumbs appear between those that are rated at +3 and those rated at +2. This ordering reinforces to the user that shows TiVo has more confidence in bringing to her attention shows she has rated versus shows it has predicted.

There are three paths by which a series may appear in the suggestions list:

1. **Explicitly rated but not scheduled for recording**: Consider the following scenario: You have rated 'Friends' as +3 thumbs but for an upcoming episode, your spouse has instead scheduled 'Nova' to be recorded. Such 'orphaned' episodes will appear in the Suggestions list to remind you that there is an episode of a show you like that is not scheduled to be recorded. This is especially useful for movies: by perusing the Suggestions list you can see upcoming old favorite movies you have rated highly but may not have known were airing in the next few days. Here the Suggestions list is acting as a scout or agent because it is suggesting shows you already know about. Since TiVo knows for sure you like this series, it assigns the maximum confidence level (255) to this prediction.
2. **Predicted by collaborative filtering:** Collaborative filtering predicts for each unrated series, a thumbs level and a confidence. If the predicted thumbs level is +3, for instance, the series will appear between the series with +3 and +2 ratings. Collaborative filtering is allowed to use the confidence range from 160 to 250.
3. **Predicted by content-based filtering:** Content-based filtering also produces for each series, a thumbs level and a confidence. In fact, it produces a two-tuple (<thumbs, confidence>) for each thumbs level but we only utilize the two-tuple for the thumbs level with the highest predicted confidence. In Bayesian parlance, the predicted thumbs level is a ordinal random variable and the confidence corresponds to the posterior probability of that thumbs level. Since Collaborative-Filtering is using a much richer feature space than the simple features (genres and cast) used by Content-Filtering, and since we have observed Collaborative-Filtering to make better predictions than our Content-Filtering approach, it was decided to assign the lower confidence range ([0,128]) to Content-Filtering and the higher range ([160,250]) to Collaborative-Filtering.

4.1 Making Predictions

TiVo has a background Linux thread running at low priority that makes content-based and collaborative-based predictions whilst not interrupting the responsiveness of the system for the viewer. This suggestions engine runs periodically; at least once a day. The prediction algorithm is shown in Figure 2. The inputs to the collaborative subsystem - `Collaborative` - of the suggestions engine are the series to be predicted along with the set of correlation pairs objects - `Pairs`. Each object in `Pairs` is a three-tuple: <series, series, correlation>. For each series `S` that is unrated, `Collaborative` finds the subset of the correlation objects that predict for `S`; lets denote this set `Pairs(S)`. Next, `Pairs(S)` is sorted with respect to the absolute degree of the correlation, which we will denote as *r(s1,s2)*. Finally, a weighted linear average over the top *k* correlates is computed to yield the predicted thumbs level τ as a floating point number. The integer portion of this (θ) is made visible in the suggestions list (Figure 1) and the fractional part (χ) is used a as a confidence level. Unfortunately, for proprietary reasons we cannot reveal the exact value of *k* or the optimizations we applied to this apparently quadratic algorithm in order to make fast predictions. Previous work in collaborative-filtering had to deal with issues of making fast predictions at the server for many simultaneous users. This issues goes away completely for us in our distributed client architecture since in this averaging phase of the computation, each TiVo is only making predictions for its own user.

Collaborative(Series, Pairs):
  For each unrated series S in Series
    let Pairs(S) be subset of Pairs predicting for S
    consider elements <S1, S, r(S1,S)> in Pairs(S)
    sort Pairs(S) with respect to absolute r
    compute weighted linear average τ as in equation E
    Predictions := Predictions $\cup$ <S, τ>
  Output Predictions sorted by τ

Figure 2. Collaborative Filtering Algorithm

If a show is unrated and does not have any correlation objects, we invoke the Content-Filtering algorithm to produce a prediction $\tau = <\theta,\chi>$ consisting of a predicted scalar thumbs level θ and confidence level χ. Series from these three sources (Content-Filtering, Collaborative-Filtering and orphaned episodes) are merged and sorted to produce the Suggestions list.

Some comments on the scale of the operation are in order. In order to produce a suggestions list daily for each user we are required to produce approximately 100 suggestions per user and since we have one million users, this requires the system as a whole to make on the order of 100 million suggestions daily. The system of Breese et al. [3] from 1998 made 13 suggestions per second on a 266MHZ Pentium so in one day (about 10^5 seconds) it is capable of making on the order of 1 million suggestions. A modern 2.6GHZ machine might make 10 million suggestions so server-side, we would need 10 such machines running 24 hours a day just to make the suggestions that the million TiVo boxes make almost as a side-effect on the client-side. This refers to the client-side weighted average calculation. It does not take into account the more computationally challenging calculation of the correlation pairs.

5. LEARNING

In this section we will first describe the scalable server-side architecture which enables us to deal with the large volume of series pairs correlations and in the second part we will describe computational details we took to ensure statistical reliability of the series-series correlation estimates.

There are approximately 300,000 distinct series airing in the USA every week, so there are approximately 10^{11} correlations that might need to be calculated. However, we do not have user ratings for all the shows: the total number of ratings is 100 million and the number of *distinct* rated shows is much less than this. Thus the ratings are sparse with respect to the number of possible series pairs and that poses a challenge if our correlation estimates are to be accurate for the less popular series.

In order to compute the correlation between two series `s1` and `s2` it is sufficient to retain a 7 by 7 counts matrix where the count n_{ij} in the *ij*-th cell of the matrix is the number of viewers who gave *i* thumbs to `s1` and *j* thumbs to `s2`. Figure 3 shows the distributed, extensible server-side architecture. The basic insight is that

correlation pairs do not need to be computed for all 10^{11} pairs since we are limited by data sparsity and server-side computational resources. Data sparsity means that if we require, for example, a minimum of 100 viewers to have rated both shows in a pair, we only need to compute on the order of 30,000 pairs with our current user population of one million client TiVos. The minimum number of viewers required to have rated a show, `min-pair`, is an important parameter to the server-side computation.

The second limit is computational: our current server setup is surprisingly lean. If more server machines become available one can simply decrease the value of `min-pair` which in turn will allow us to calculate correlations for less popular shows and provide more niche recommendations. Alternatively, as our user population increases, even with our current value of 100 for `min-pair`, the amount of calculation we will have to do will increase. Our goal has always been to push out as many correlation pairs as possible within budget limits so we can cater to as many niche audiences as possible.

In Figure 3, the first layer (horizontal) consists of logs servers: these machines accept thumbs logs and diagnostic error logs from the TiVo boxes. Each log is assigned a fictitious user ID. The lifespan of these user IDs is for the duration of the correlations computation. During computation, the log of a single viewer may be split among many machines indexed by series ID (second layer). We need a temporary anonymized user ID so that when we re-join the data to do pairs computation we can align thumbs for different shows from the same user.

The second layer consists of machines which are indexed by series ID. Each machine is assigned a series ID range and it is responsible for counting the number of votes received for series in its range. A second parameter to the architecture is `min-single` - the minimum number of distinct users that have to give thumbs to a series for it to be even considered for pairs computation. Note that for a pair of series `s1, s2` to have at least `min-pair` ratings jointly, then `s1` and `s2` separately must also have also received at least `min-pair` ratings. So it is an admissible heuristic if we prune series that by themselves receive less than `min-pair` ratings. However, if we set `min-single > min-pair` then we run a possibility that we will prune series that jointly may have met the `min-pair` criterion. For example, if `min-pair` is 100 and `min-single` is 150 then we will prune away two series each of which singly received 145 ratings - yet it is possible (although unlikely) that had we retained them, we would have found that that pair had jointly received more than 100 ratings. Machines at this second layer are responsible for forwarding series IDs only of series that have been rated on at least `min-single` TiVos. They also forward, for each surviving series, the anonymized user IDs and thumbs values of all TiVos that had ratings for that series.

Machines at the 3rd and final layer - the compute layer - are partitioned over the space of pairs: series * series. For each series-series pair, the machine first does a simple quick filtering calculation: it computes the number of TiVos which yielded ratings for that pair. If this number is less than `min-pair,` the series-series pair is pruned. Otherwise, the machine can embark on a more expensive calculation to compute the 7 by 7 matrix for that pair and then to compute the linear correlation.

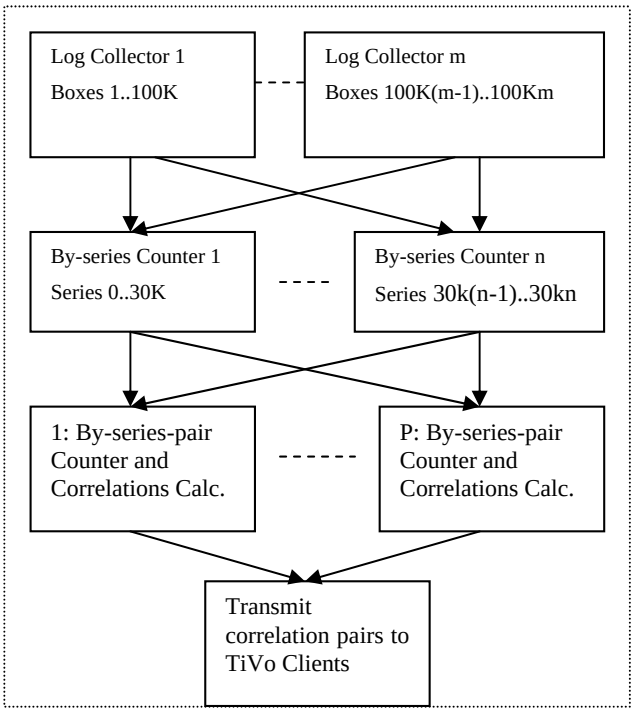

Figure 3. Parallelized, scalable server-side architecture

We use linear correlation rather than measures of association over discrete variables (e.g. Phi coefficient or Cramer's V [9]) because we treat the thumbs on each series as a discrete *ordinal* variable rather than a nominal (categorical) variable. Doing so affords greater statistical power. The output of this compute layer is a vector of 3-tuples of the form <series, series, correlation>.

Another way of taming the scale problem is to not conduct these server-side correlations for the entire series universe every day. For example, if we only have one server machine, we can compute 1/16 of the series-series space each day and thus complete one calculation over 16 days. The disadvantage of this approach of course is that the server cannot serve "fresh" correlations each day; it can only serve new correlations every 16 days. However this not so much of a disadvantage since the correlations between series tend not to vary much from day to day, or even week to week. Over time ranges of months, however, they do vary as the flavor of a series may change. To reduce network bandwidth and cost, the size of the download from the server to the TiVo is limited, so we round-robin serve TiVos over the 16 day period so this reason also allows us to spread the computation out over a multi-day period. Thus, within any given 16 day cycle, the server is serving pairs correlations from the previous cycle and is working on completing its current cycle.

Now we examine the issue of the statistical reliability of the correlations. The correlations we compute are estimates of the "true" degree to which the two shows are correlated. We do not know the “true” degree of correlation between two shows even though we collect logs from all our viewers because viewers who like those shows may not have given thumbs to them, because our viewers are only a statistical (and non-random) subset of all viewers and because the show may be newer than other shows so fewer people have had a chance to give it a rating. For estimates

based on sparse data, there may be significant error in the estimated correlation. We need to take the confidence interval around our point estimate into consideration. For popular shows, we may have lots of evidence (support) to compute their pair-wise correlations. Mathematically, support for a pair of series is the number of TiVos from which we have thumbs data for both shows. We may have a computed correlation of 0.8 between two popular series with a support of 10000 and on the other hand we may have a computed correlation of 0.8 between two rare series based on a support of 10. Is there a principled way to differentiate these two estimates; to assign a lower number to the 0.8 correlation arising from the support of 10? Currently we attack this problem by computing a 95% confidence interval around the correlation point estimate and we use the lower boundary of the confidence interval as a pessimistic, support-penalized correlation estimate. In our example, the confidence interval around the rare shows will be wider so the lower boundary will be lower than the lower boundary for the popular shows. Hence, after this support-based correction, the two pairs will no longer be assigned the same correlation number.

The Pearson correlation measure, *r*, we are using does not have a normal distribution but if we apply Fisher's r-to-z transform, we have a normally-distributed quantity *z* with mean μ and standard-deviation σ and 95% confidence interval $[\mu - 1.96\sigma, \mu + 1.96\sigma]$.

Therefore, we can use the quantity "$\mu - 1.96\sigma$" and convert it back to a r-value and use that r value as our estimate of correlation.

$$z' = \frac{ln(1+r) - ln(1-r)}{2}$$

Equation 3. Transform linear correlation to get normally distributed variable z

For example, the 95% confidence interval around the popular show pair may be [0.75, 0.85] and hence we will use the 0.75 number as its correlation. For the rare show pair, the confidence interval may be [0.65, 0.90] so we will use the 0.65 correlation estimate. Therefore, whereas the two pairs both had a 0.80 correlation estimate to begin with, now the show pair which has more thumbs evidence and support receives a higher estimate than the rare pair with less support.

6. FUTURE WORK

The number one item on our agenda for future work is a thorough empirical evaluation of the quality of suggestions. So far we have only evaluated the suggestions engine among TiVo employees by setting up an internal website. This set of suggestions was also used to bootstrap TiVo's server-side computations in 2000.

Other future work falls into three categories: user interface, server-side, and client-side new features. In the user-interface, suggestions can be used in a number of ways. Currently, the TiVo Suggestions screen serves the dual functions of a Scout (finding orphaned episodes of rated series) as well as suggesting unrated shows that may be new to the user. We would like to separate these in the user-interface. It would also be interesting to allow programs to be sorted in the live-guide by predicted thumbs value. So that, while watching live TV, one could list shows on other channels sorted not by channel number, but by the probability

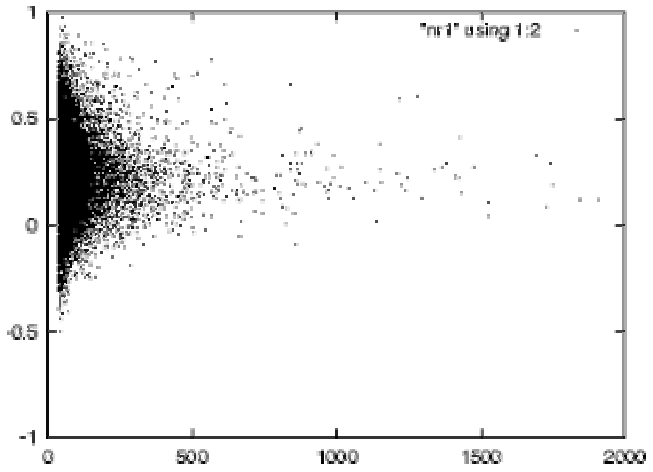

Figure 4. Plot of support (x-axis) versus correlation (y-axis) for 100,000 shows

that you would like that other show. Undoubtedly, this would pose some legal issues because networks pay more to obtain lower numbered channels which are more likely to be channel surfed and watched than some obscure network at channel 1000.

Another feature we have prototyped is 'Teach TiVo'. This feature would allow the user to explicitly rate genres and cast members. So, for example, you could directly tell TiVo that you like (all) documentaries. It would also allow you to look at your complete set of thumbs so if someone in your household or a visitor inadvertently rated a program, you could see it and remove it. For example, a visiting child may thumbs lots of children's shows causing TiVo to suggest children's programming in an otherwise adult household.

On the client, we might make improvements in the following ways:

- **Aging**: Since TiVo has been fielded for over four years, some of the thumbs ratings in the profile may be several years old and it would be nice to be able to decay or remove the influence of such older thumbs.
- **Separate user-profiles**: If there are multiple viewers with disparate tastes, it might be beneficial to have them 'log in' so that suggestions personalized to them could be served. There are other advantages of 'logging in'. Advertisers might be interested in who within the household is watching. The viewer might be incented to log in if they get a favorites channel lineup just for them, or suggestions just for them.
- **Portfolio effect:** Consider a viewer with two favorite genres: science fiction and boating. Assume that the number of shows and ratings for science fiction far exceeds those for boating. TiVo may recommend plenty of science fiction shows but very few boating shows. A portfolio maximizing approach seeks to deliver the best *set* of suggestions. By contrast, a probability maximizing approach seeks to deliver suggestions, each of which individually maximizes probability that the viewer will like the show.

On the server, we might take TD-IDF steps so that shows that got fewer votes would be more heavily weighted. Following Breese

[2] we could also give differential weights to users so that users that only have a few votes would receive greater user weights.

7. CONCLUSION

TiVo has afforded us a rare opportunity to try collaborative filtering on an very large scale. The collaborative filtering system described here has been fielded in one million TiVo client boxes and is used daily by millions of users. Each of these viewers has rated approximately one hundred shows on average leading to a total set of one hundred million ratings. TiVo uses an item-item form of collaborative filtering with strong provisions for privacy preservation. It uses k-nearest neighbor with Pearson correlation to make show recommendations. Correlations computed over less support are penalized in a principled way using the pessimistic estimate from a 95% confidence interval. The collaborative filtering system is augmented on the client by a content-based Bayesian recommendation system to address the cold start problem for new users and shows. The server architecture is highly scalable with capacity for many more users and can be throttled to provide more correlations to cover niche recommendations. TiVo's novel distributed collaborative-filtering approach reduces load on the server by having each client in parallel make recommendations for its own user.

ACKNOWLEDGMENTS

Thanks to Howard Look and Jim Barton at TiVo. Thanks to Mike Pazzani and Cliff Brunk for feedback..

8. REFERENCES

[1] Aggarwal C.C., Wolf J.L., Wu K-L. and Yu P.S. Horting. (1999). Hatches an Egg: A New Graph-Theoretic Approach to Collaborative Filtering. In KDD'99, Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 201-212. ACM Press.

[2] Billsus, D. and Pazzani, M. (1998). Learning Collaborative Information Filters, ICML 1998: 46-54.

[3] Breese J.S., Heckerman D and Kadie C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI. Morgan-Kaufmann.

[4] Canny J. (2002). Collaborative Filtering with Privacy via Factor Analysis, ACM SIGIR, Tampere, Finland.

[5] Cheeseman, P. and Stutz, J. (1995). Bayesian Classification (AutoClass): Theory and Results. In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R., ed.s, *Advances in Knowledge Discovery and Data Mining,* 153-180. AAAI Press.

[6] Digital Equipment Research Center. http://www.research.digital.com/SRC/EachMovie/.

[7] Cohen, W.W. and Fan, W. (2000). Web-Collaborative Filtering: Recommending Music by Crawling the Web. In Proceedings of WWW9.

[8] Duda, R.O. and Hart, P.E. (1972). *Pattern Classification and Scene Analysis.* Wiley, New York.

[9] Everitt B. S. (2002). *The Cambridge Dictionary of Statistics*. Cambridge Press.

[10] Goldberg, D., Nichols, D., Oki, B.M. and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM,* 35, 12 pp. 61-70.

[11] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, 194-201.

[12] Järvelin, K. and Kekäläinen J. (2000). IR evaluation methods for retrieving highly relevant documents. In Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 41-48.

[13] Nichols D. (1997). Implicit rating and filtering. In *Proceedings of the Fifth DELIOS Workshop on Filtering and Collaborative Filtering*, Budapest, Hungary.

[14] Salton, G. and Buckley, C. (1988). Term weighting approaches in automatic text retrieval. In *Information Processing and Management,* 24, 5, pp. 513-523.

[15] Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. Item-Based Collaborative Filtering Recommendation Algorithms. In Proceedings of WWW10, May 2001.