MERGING CONSUMPTION DATA FROM MULTIPLE SOURCES TO QUANTIFY USER LIKING AND WATCHING BEHAVIOURS

Sashikumar Venkataraman, Craig Carmichael

Rovi Corporation

Abstract

We describe an integrated platform that aggregates consumption data from multiple sources towards building a unified model for collaborative filtering for more accurate user and content representations. The goal is to provide a framework that combines various signals spanning explicit ratings, implicit information of watching behaviors and meta-content information in a single model that potentially goes beyond the usual goal of maximizing consumption and incorporates metrics that capture "likeness" and "discovery". We also feed the usage data back into meta-content to determine more accurate content representations that aid in targeting content-based recommendations more effectively.

INTRODUCTION

Recommendation systems are becoming an increasingly key component in many media and entertainment content retrieval systems by providing a powerful, efficient method for users to easily sift through large catalogs of media content and finding valuable programming [1-3]. Within the last couple of decades, several algorithms have been developed that leverage metadata, such as information on programs, casts and genres, and user consumption data to provide users with more targeted content recommendations through these recommendation systems [4-5]. However, in most systems, much of the viewership data is typically implicit in nature, and models that are based on over-simplified estimations of viewing behavior are often not wholly comprehensive and intuitive. A user is further predisposed to watch certain shows and channels often based on pure habit, content they are already aware of, or the sheer popularity of a show, and these aspects are usually unaccounted in the recall score of a recommender system.

In this paper we discuss a unified framework that aggregates consumption data from multiple sources and fuses them with meta-content to create an efficient recommendations framework capable of both significant discovery and high precision. First, we begin with the cold start problem where no consumption data is available, and create a baseline recommendation model that applies word-to-vector [6] factors solely from metadata content. An aggregation process is used to accumulate these word-level vectors to content level factors for each show and channel potentially consumed. Next, we derive the usage factors for each media asset considering both explicit and implicit ratings from various sources. While the explicit ratings provide a more direct notion of "likeness", the implicit signals only provide a measure of watching behavior. We discuss methods to correlate these notions of likeness and watching attributes in a more formal way.

A central part of the framework for merging consumption data from multiple sources is the Rovi Knowledge Graph that incorporates factual information of all 'known' or 'named' things. This includes all movies and TV shows, music albums and songs, as well as all known people such as actors, musicians, celebrities, music bands, known companies and businesses, places, sports teams, tournaments and players, etc. All the facts pertaining to these entities are synthesized from multiple publicly available sources such as Wikipedia, Freebase, and many others and correlated so as to create a unique smart tag (with a unique identifier) to represent each entity in the Rovi Knowledge Graph. The main utility of the knowledge graph for our problem is in aiding to merge information across different data sources. We have a merge system that can take any data source and does a best-effort to merge the entities in that data source with the underlying knowledge graph. Such a system makes the aggregation easier since different data sources have variations in referring to entities and carry different level of meta-information.

One often encounters rich meta-content associated with media assets, such as genre, keywords, and description. However, the relevance or weight of each individual piece of meta-content (for finding similar movies or recommendations) is often lacking, missing or wrong due to multiple sources, algorithms, and/or manual entry. For example, a show is a comedy but exactly how funny and how it impacts other comedies is more of a viewing sentiment. Usage data, on the other hand, provides a different kind of information in conveying what programs co-occur in watching behavior across users. Analysis of usage data involves very different techniques and algorithms from those for analyzing meta-content. There has been some effort in the context of recommendations wherein one uses meta-content to filter in the post-process of the collaborative filtering algorithm or mixes results coming from collaborative filtering and meta-content algorithms. However, no prior efforts make use of usage data to better understand metadata relevance and enrich meta-content directly. It would be desirable if usage data along with the implicit/explicit ratings of users could be leveraged to determine the relevant weights of different pieces of meta-content.

OVERALL ARCHITECTURE

We begin by describing the key steps involved in merging deep and dynamic metadata with usage data across various data sources. These steps are described in brief below and will be discussed in further detail in the following sections.

Step 1: We first merge the assets from the data source into the central knowledge graph, to enable infusing of usage and meta-information to and from the knowledge graph. This step serves a dual purpose. Firstly, it helps in augmenting the meta-content of the assets in terms of keywords, genres and deep-descriptors from the knowledge graph and hence aids in getting more accurate factors derived from meta-content representations. Secondly, it aids in merging the usage factors from this data source with usage factors from other data-sources that are also merged with the central knowledge graph.

Step 2: Using word-representations, we next determine the meta-content factors corresponding to the media-assets for cold-start baseline with no or minimal usage data. The word2vec model developed in [1] is used to determine the word-representations corresponding to each of the meta-content information and these are aggregated to form a vector for each asset in K-dimensional vector space.

Step 3: Next, we build a model that merges implicit and explicit information from multiple data sources and fuse them into the meta-content factors to create more accurate asset representation. This naturally results in a model to estimate “likeness” from user-watching behavior even in absence of explicit information.

Step 4: Finally we feed the usage data back into meta-content in the form of coefficient weights of each individual meta-content factor involved in each asset. This enables the creation of more accurate representation of the individual meta-content factors that further increase the precision in content-based recommendations.

MERGING INTO KNOWLEDGE GRAPH

The Rovi Knowledge Graph is a dynamic system that has been created by synthesizing multiple metadata sources and evolving and refreshed continuously over time. Each smart tag in the Knowledge Graph has rich meta-content built by a combination of manual tagging and automatic aggregation from multiple sources along with several machine-learning algorithms.

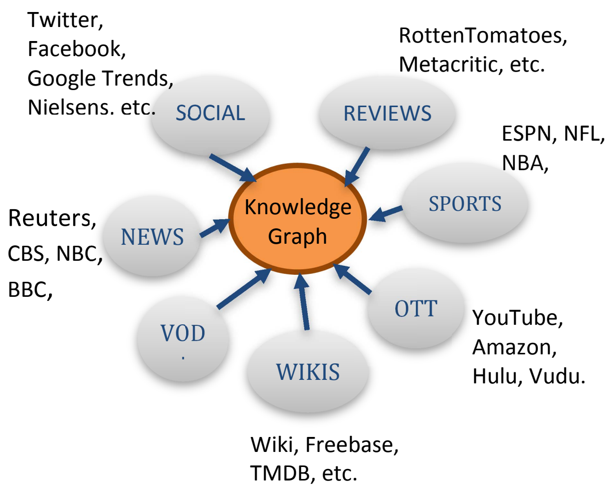

Fig 1: Merging of assets from multiple data sources into a central knowledge graph

Fig 1 below shows the merging of information from various sources into the central knowledge graph. While certain meta-content fields, such as cast-members, roles, and release year are unambiguous, while other fields such as genres and keywords are more subjective. These fields are assigned using a combination of manual tagging and automatic aggregation from multiple sites like Wikipedia and Freebase. Genres can also be assigned by tagging directly from keywords gleaned from descriptions or plots. This is done using machine-learning algorithms by first determining the set of keywords associated with a genre, using movies known to have that genre, and then predicting them on other movies that have a strong overlap in keywords with the genre keywords.

The core part of building the Knowledge Graph is a "merge" function that takes in any source and tries to map the entities in that source with the entities in the existing graph. Whenever a new entity, such as movie or personality, is merged with an existing smart tag, the metadata corresponding to the smart tag gets augmented from the entity. If the new entity does not get merged with any smart tag, then a new smart tag is created for the entity in the KG. An important aspect of the merging is the allowance of slight variations in the meta-content fields between the two assets. For example, the titles can differ slightly due to several reasons. One reason for the difference in titles is the fact that some sources put the season number and episode numbers in the title, while other sources only put the episode title in the title. Sometimes they may differ due to some lexical error or absence of common words like articles. Release year could also vary by one or two units, and sometimes cast members could be be missing. All such variations are considered during the merge process by considering all the fields simultaneously and coming with a combined match score considering all fields. Following are 2 examples of approximate matching with inexact titles which got matched due to matching of other fields:

BUILDING WORD-REPRESENTATIONS FROM META-DATA

We now describe the step of determining the baseline factors for the media-assets from meta-content. These factors are especially useful in cold-start situations with no or minimal usage information. Over time the usage information from multiple data sources play a key role in how the asset factors evolve; nonetheless the meta-content factors remain valuable to bias recommendations towards more meta-content oriented discovery without losing high precision. In our collaborative filtering algorithm, we represent meta-content information of each unique media-asset as a weighted combination of individual meta-content pieces, such as genre, category, and keywords. Each individual piece of meta-content can be represented as a vector in a K-dimensional vector space (K is usually 100-300). Each asset vector is then a weighted sum of individual vectors and hence also a vector in this space. The individual vectors could have been determined independently by other known algorithms based on co-occurrences of terms in large corpus (such as word2vec [6]).

While a naïve algorithm in determining the asset vector could be to just average over all the vectors of keywords and genres, there are several problems in this approach. Firstly, not all keywords are of same importance and some keywords may be noisy and unrelated to the theme of the movie. Secondly, the genre vectors are not readily available in the word2vec model and needs to be explicitly computed from the movies. To address the first problem, we process the movie keywords to form keyword clusters based on the closeness of the keyword-vectors and then filter those keywords that don't fall in a significant weighted cluster. For the second problem, we treat a genre as a collection of movies and hence cluster all the keywords across all the movies belonging to that category. The top keyword clusters found by this clustering are then used to find the final genre vectors. After we determine the keyword and genre vectors, we can then find the final asset vectors as a linear combination of the corresponding keyword and genre vectors. While these asset vectors are a good starting point in our collaborative filtering model, it may not be the most accurate representation due to the lack of knowledge of the genre and keyword cluster weighting coefficients. We will discuss techniques to optimally determine those weights in a later section.

MULTI DATA-SOURCE COLLABORATIVE FILTERING MODEL

As mentioned earlier, one of the main challenges in doing collaborative filtering is the lack of explicit ratings in dealing with usage data from many data sources. Most of this usage data results from linear TV watching behavior where the users are limited in content choice and no explicit feedback is provided that indicates how much the user liked a particular show or movie. We also get (to a lesser extent) usage data that involves VOD (video on-demand). Though the signals from VOD (and DVR) do provide a greater correlation to user liking behavior, they too are not as accurate as explicit ratings due to the limitation in the number of media assets in typical VOD catalogs. At best, they capture the kinds of genres and categories that the user typically watches, but fails to capture more fine-grained likeness or dis-likeness within a category. So the most valuable information that we get is the usage data from a couple of sources that involve explicit ratings. However, the amount of this kind of usage data is a lot less compared to the usage data relating to linear TV and VOD. The challenge then is to build a model that can unify all these forms of usage data and provide a model to carry over information from data source to augment the usage factors in another source.

There are several models existing that convert the ratings (explicit or implicit) to similarities between media assets. These include Pearson correlation coefficient, cosine similarity, log-likelihood and jaccard coefficient. Yet another interesting coefficient is a notion of probabilistic similarity as discussed in [7] referred as ProbSim in sections below. During merging ratings from multiple sources, the data sources explicit ratings are given a higher weight than data sources with implicit ratings. We also treat VOD/DVR watching as a more explicit intent and give higher weight to those data points. The final result of merging all these forms of usage data is a NxN similarity matrix where each element *(i, j)* gives the similarity between the items *i* and *j*. The similarity matrix can either be created with only explicit ratings (referred as *XSim(i, j)*) or implicit ratings (referred as *ISim(i, j)*) or combining both (referred just as *Sim(i, j)*).

Our next step is to determine the factors for the media-assets that most closely match these similarities derived from the usage data. The starting point for the asset vectors are the factors determined in the previous section from the word-representations corresponding to the meta-content. The asset vectors are then let to vary so as to match the usage similarities as much as possible in an iterative fashion. For every item-item pair, the corresponding item factors are made to come closer or away from each other based on how much the cosine distance between the asset vectors match the computed item-item similarity. These final asset vectors represent a more accurate representation that reflects usage based similarity and simultaneously remaining as close to the meta-content representation as possible. Hence, these form an ideal representation for hybrid recommendations.

We next use these similarities to create a model to determine explicit user likeness from implicit watching behavior. While explicit ratings were unambiguously absorbed in the similarity computation, implicit watching were accompanied with several signals that needed to be translated to some form of implicit rating in a systematic fashion. Some of the these signals include duration of watching, number of times/episodes watched, number of similar items user has watched around that asset, the price the user paid for watching, the average rating users have rated that particular item in other data sources, popularity of the asset, etc. The goal is then to build a model that uses these signals and uses a model to create an optimal expression for "implied rating" or "likeness". For this purpose, we create a model that translates the implicit ratings to similarities and then match them with the similarities obtained from explicit ratings. Figure 2 shows the model that is used for this purpose.

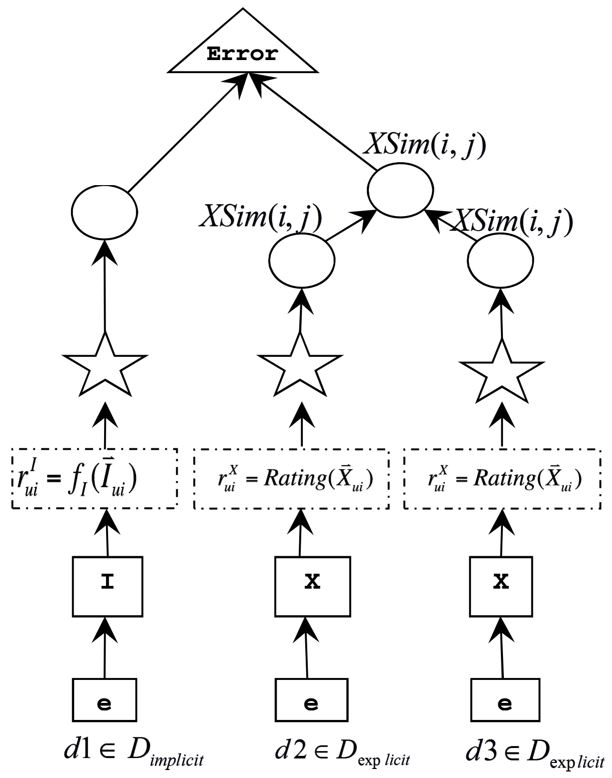

Fig 2: Merging usage from multiple data sources with explicit and implicit ratings

In the above figure, we aggregate the raw user event data in the first or bottommost layer, *L1*. Examples of explicit indicators include specific data that contain explicit information such as ratings on the scale of 1-5 in increments of a half-star, or a like/not (0/1) binary indicator. This would comprise an explicit (input) vector $X_{ui}(d)$. Example of implicit indicators for user-item interaction would include implicit watching signals such as the ones described earlier. This would comprise an implicit vector *I*. The next layer

L2 contains a mapping from consumption details provided by *L1* (both $\vec{I}_{ui}(d)$ and $\vec{X}_{ui}(d)$) to the output to a form of preference or likeness of a user *u* to item *i,* $r_{ui}(d)$ in the data source *d*. If the data set d involved is an explicit one, the mapping is often straightforward. However, for implicit data sets, the mapping is more challenging and the mapping would be in the form of a (preference) model taking various forms $f_I(\vec{I}_{ui}(d^I),\vec{W})$ that ultimately depends on a set of trainable weights $\vec{W}$. Different functions can be used for various embodiments of f_I which represent trainable $\vec{W}$ in different ways. For example, we can use a linear estimator with a sigmoid function at the output node, a general regression neural net, a random forest, or various others.

The predicted ratings are then passed to a similarity layer *L3*, that takes the preference detail estimates for each (u,i,d) across all common media assets and users, and produces a similarity estimate between media assets i and j. While different kinds of similarities (discussed earlier) can be used, we use similarities based on a weighted Pearson coefficient as mentioned below:

$$Sim(i,j,d)=\frac{\sum_{u\in(i,j)}^{U(d)}\left(r_{ui}^{X}(d)-\bar{r}_i^{X}(d)\right)\left(r_{uj}^{X}(d)-\bar{r}_j^{X}(d)\right)}{\sqrt{\sum_{u\in(i,j)}^{U(d)}\left(r_{ui}^{X}(d)-\bar{r}_i^{X}(d)\right)^2}\sqrt{\sum_{u\in(i,j)}^{U(d)}\left(r_{uj}^{X}(d)-\bar{r}_j^{X}(d)\right)^2}}$$

It is assumed that whether similarities are determined using implicit or explicit data, they should roughly equate to the same values per (i,j) pair across multiple data sets. So first we obtain an estimate *XSim(i,j)* based on the usage data among data sources with explicit ratings. This is then compared with the data sources with implicit ratings and a difference is then used in the final layer *L4* to compute the error between the two observed similarities as shown below:

$$Error=\sum_{d=1}^{Dimplicit}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(XSim(i,j)-ISim(i,j,d)\right)^2$$

Then we can take the error, and propagate it backwards, layer by layer until the derivatives are estimated across the trainable weights $\vec{W}^I$. The iterations are performed until the error is minimized to below a certain threshold. We then use those weights as the optimal coefficients to combine the implicit signals and create an implicit rating.

FUSING USAGE INFORMATION INTO META-CONTENT

The collaborative filtering model used here is referred as Weighted Vector Collaborative Filtering (WVCF) where the meta-content information of each media-asset is represented as a weighted combination of individual meta-content pieces, such as genre, category, and keywords and each individual meta-content piece is treated as a vector in a K-dimensional vector space. A meta-content similarity of the two assets is then modeled as a function of these individual meta-content pieces of information (such as a dot product). However, the weight coefficients for each individual piece of information are usually not known apriori and our goal is to use usage information to best predict these weights. Once these weights are determined, we can create more accurate item-item similarities and thereby more accurate recommendations.

Importantly, the WVCF baseline model consisting of a single trained vector per asset can be broken up into two finely tuned predictors in the modeling pipeline, one that targets aspects of watching (WVCF-watch), and one that targets liking characteristics of the user (WVCF-like). Later in the results section is an example of recall performance using Gradient Boosted Trees as a local corrector to target both sides of precision (liking) and recall (watching).

As described in the previous section, the usage information is separately modeled to produce item-item similarity wherein items watched together and similarly evaluated/rated (common sentiment) across multiple users have better usage-similarity. This similarity also takes into account the user's sentiment along with viewing information (i.e. both watched and similarly liked/reviewed as opposed to only watched). All similarities are in some sense based in part on co-occurrence; usage-based similarity is the co-occurrence of users watching the same program, while meta-content similarity is the co-occurrence of some meta-content (crew or genre or keyword). The strength of co-occurrence in the meta-content case is based on the weights of the individual meta-content information within that asset, while the strength of co-occurrence in usage-based similarity is governed by the user sentiment/rating.

The system then tries to align the media asset vectors as close as possible to the usage-based similarities. An error function is then constructed that compares the modeled meta-content similarity to the usage-based similarity (based on co-occurrence combined with sentiment factors).

$$E = \sum_{ij} \left(s_{ij} - m_{ij} \right)^2$$

where s_{ij} denotes the observed "sentimental" similarity between items i and j (as determined from usage data discussed in previous section) and m_{ij} denotes the modeled similarity based on the asset meta-content vectors. The objective in this error minimization is to explain the observed sentimental similarities between many asset pairs with a model that also has deeply embedded metadata. The modeled similarity m_{ij} defined by the dot product of asset vectors $\vec{a}_i$ and $\vec{a}_j$ over all the latent factors:

$$m_{ij} = \sum_f a_{if} a_{jf}$$

We next break down the asset vectors into individual metadata components consisting of keywords and genres:

$$a_{if} = v_i^{genres} w_{if}^{genres} + v_i^{key} w_{if}^{key}$$
$$w_{if}^{genres} = \sum_{g \in i}^{genres\ in\ i} v_g^{genre} w_{gf}^{genre}$$
$$w_{if}^{keywords} = \sum_{k \in i}^{keywords\ in\ i} v_k^{keyword} w_{kf}^{keyword}$$

where w_{if} denotes the total genre and keyword factors, and v_i denotes the corresponding weights of those fields. The model continues to extend so far as metadata is available. Unknown components can also be added to address when metadata is lacking or unavailable. We then break down the genre and keyword vectors further into factors corresponding to individual genre and keywords. Next we minimize the error function using stochastic gradient descent that changes the weights of the individual meta-content components so that the net error between the meta-content similarities and usage-based similarities is minimized. After iterating over all the usage data, the individual meta-content weights are stored as the best predictors for the corresponding meta-data relevance for the media-asset.

RESULTS

Table 1 below shows 10 examples of the most similar shows based on the cosine similarity between sentiment vectors prior to training the model using only a small subset of the terms in each a_{if}. This is using only raw source combined with weighted word vectors without the use of metadata filtering based on advanced tags, moods or deep descriptors. So the starting point a_{if} used for training each media asset vector is already capable of overcoming cold-start issues.

Table 2 below and Figure 3 demonstrate the trained accuracy of our model for various number of meta-content factors. Recall@K for various values of K was computed over a VOD data set. Note the objective here was to show that precision and recall can be contained within a vector rather than a similarity matrix without significant loss, while holding metadata seamlessly across usage spaces as vectored sub-components. Further improvements are seen using WVCF-watch and WVCF-like (beyond the vector). With the vector, other training was based on 15K assets and 1 million users over the span of a year. It appears little is lost in performance when comparing the current method to a purely usage based approach such as ProbSim, which has proven to be a top method in recommender systems space in terms of precision/recall.

Embedded into our model is the potential to also normalize to a space that can be consistent across multiple usage data sets. So when recommending an asset that is either completely new or not yet popular, this model will tend to significantly outperform purely usage-based approaches.

Title 1	Title 2
Top Chef	Top Chef: Texas
Scooby-Doo! Pirates Ahoy!	Scooby-Doo and the Alien Invaders
28 Weeks Later	Resident Evil: Apocalypse
Pirates of the Caribbean: On Stranger Tides	Pirates of the Caribbean: The Curse of the Black Pearl
Kansas City SWAT	Detroit SWAT
Stuart Little 2	Stuart Little
Modern Vampires	Vampire in Brooklyn
Mad Max	Death Race 2000
Godzilla vs. Mechagodzilla II	Terror of Mechagodzilla
Attack of the Killer Tomatoes	Return of the Killer Tomatoes

Table 1: Example most similar shows using meta-content tags before training usage data

Model	Recall@K				
	K=5	K=10	K=25	K=50	K=100
Probsim	0.192	0.286	0.445	0.551	0.681
WVCF, F=20	0.163	0.249	0.365	0.452	0.575
WVCF, F=50	0.163	0.266	0.392	0.495	0.626
WVCF, F=100	0.186	0.289	0.425	0.528	0.664
WVCF, F=300	0.206	0.296	0.432	0.551	0.678

Table 2: Recall performance of WVCF vs Probsim

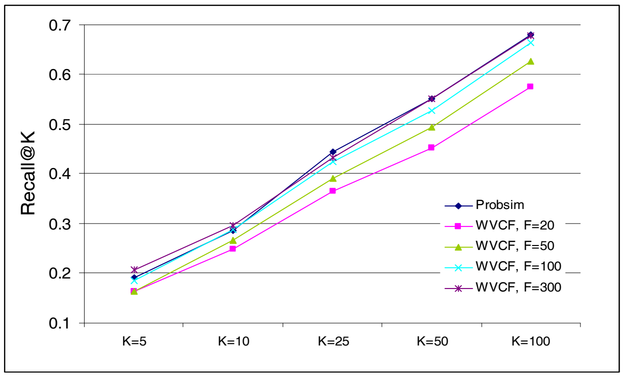

Fig 3: Recall performance of WVCF baseline vs Probsim for various factors, prior to local affects

The difference between WVCF and ProbSim becomes more prominent when we consider the “discovery” factor along with recall precision. This factor is based on how many overall programs of the corpus get recommended in recall to a set of users. It has been noted that models like ProbSim have an extremely low diversity since most of the recommendations are based from the top 1% popular programs.

Table 3 shows the top 15 programs from the bottom quartile of the MovieLens (20M, shows having less than 100 ratings discarded) dataset using the total number of positive and negative ratings and sorting by the ratio of positive ratings to the total number of ratings. While these “hidden gems” are virtually never recommended by the ProbSim model, they do have a high positive rating (though overall number of ratings is small), and are therefore appropriate for more targeted user-recommendations. This is versus at least 4% hidden gem recommended for a modified approach, and thus capable to have a better spread in recommendations w.r.t. the discovery of "long-tail" programs.

Figure 4 shows the variation among users in MovieLens [9] for their preference in watching different grades of popular shows. Recommender systems targeted to high recall and high precision tend to overshoot for the most popular shows (Q1) to score well on recall (and precision automatically since there are not many "2" or lower ratings for most popular shows), while not discovering quality shows at lower popularity.

Title(i)	N(i)	Rating Avg(i)	G(i) +ve rating	B(i) -ve rating	P(i) = G(i)/N(i)
Intimate Strangers	126	3.655	117	2	0.9832
Follow the Fleet	118	3.712	111	2	0.9823
Ax, The (couperet, Le)	114	3.807	107	2	0.9817
One Man Band	105	3.895	98	2	0.9800
Angels' Share	105	3.710	96	2	0.9796
Queen of Versailles, The	144	3.628	137	3	0.9786
Yes Men Fix the World, The	194	3.843	179	4	0.9781
Big Clock, The	162	3.756	153	4	0.9745
Hundred-Foot Journey, The	156	3.750	143	4	0.9728
Short Film About Love, A	224	4.025	214	6	0.9727
Our Man in Havana	153	3.748	142	4	0.9726
Pixar Story, The	186	3.747	177	5	0.9725
Criss Cross	115	3.639	103	3	0.9717
Advise and Consent	183	3.811	170	5	0.9714
Something the Lord Made	214	3.895	199	6	0.9707

Table 3: 15 Hidden Gems in the fourth quartile of popularity in MovieLens

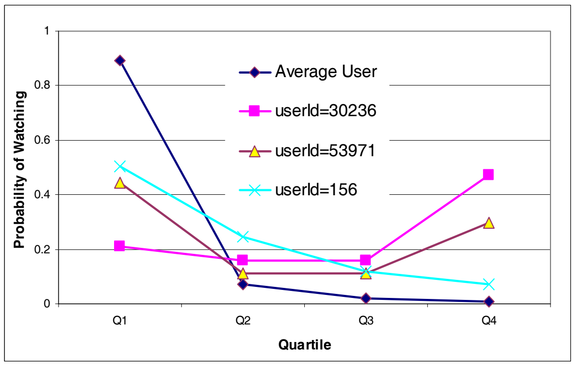

Fig 4: Popularity effect by quartile in MovieLens

To address the above issue, we introduce a penalty for not matching the user's overall popularity preferences over multiple bins using a loss such as:

$$LOSS(u) = 1 - \alpha \sum_{b=1}^{Nbins} \left| P(u,b) - p(u,b) \right|$$

Averaged over all u users, with $\alpha = 1$ and $N_{bins} = 10$ popularity bins, for example, the new recall score is normalized to better match user distribution $\vec{P}(u)$ with the recommended distribution $\vec{p}(u)$. This penalty reduces Probsim's recall score over MovieLens from roughly 0.3 to 0.1, with Q4 hidden gems still at nearly 0%. Furthermore, different shows and movies have quite different popularity characteristics across data sets, both explicit and implicit. To this extent, recall scores should be penalized even further to remove arbitrary popularity artifacts associated with each data set. Although the choice of α was arbitrary in this case, without having a "budget" to place recommendations into popularity bins which fits known user preferences, the model otherwise appears lacking in diversity and drastically under-recommends hidden gems, for the short-sighted purpose of scoring high recall.

If the model were tweaked for purely high recall, where watching is predicted rather than liking (to find hidden gems), Figure 5 below shows example performance of WVCF-watch along with liking and the combination of like/watch. Figure 6 shows the precision equivalent. Here Gradient Boosted trees take the baseline WVCF (vector only), one movie at a time, and apply the baseline predictor as an attribute for that movie. It then expands the input deck to also include many attributes in the surrounding neighborhood of ratings. This produces a very strong estimator targeted to watching. In this case, Figure 5 shows SVD (F=100) versus Probsim versus WVCF-watch for the (budgeted) popularity bin of the top 25 most popular shows. Recall/Precision points were tallied based on each user/item sample in the cross validation being somewhere in that bin (with the predictor answering which movie was watched of the 25). The important thing here is that with both WVCF-watch and WVCF-like, the underlying recommender engine can control effects such as the dial for liking and watching, compared to other models with high precision/recall that only provides a single answer (both like and watch simultaneously).

Table 4 below shows the results of applying the 208 word emotions vocabulary from [10] using pre-trained word vectors to MovieLens usage within a modified SVD framework. As these moods serve as additional metadata, the experiment here shows the power of Word Vectors fused into the model. The training process tuned the relevance of each vocabulary word vector to each movie, within SVD latent movie factors, to best model explicit likes. For top word ranking, interesting patterns emerge between overall, most popular, and gems. For example "contrary" and "bored" is universally important, "obnoxious" matters for most popular, and "powerful" really applies to gems.

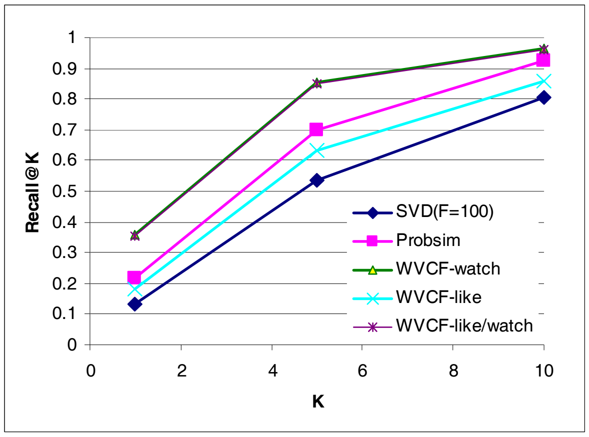

Fig 5: MovieLens Recall@K for top 25 Most Popular bin during "budgeted" recall

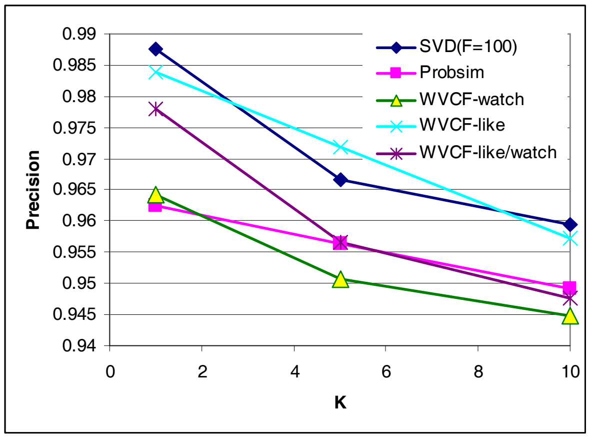

Fig 6: MovieLens Precision for top 25 Most Popular bin during "budgeted" recall

Rank	Overall	Popular	Gems
1	contrary	lazy	honest
2	angry	inconsiderate	contrary
3	brilliant	pity	cordial
4	wonder	obnoxious	powerful
5	bored	annoyed	lucky
6	insightful	woeful	wise
7	rough	haughty	aggressive
8	affectionate	uncertain	sad
9	weird	alive	nasty
10	folksy	silly	folksy

Table 4: Relevance of emotions/feeling/mood word vectors trained over MovieLens usage

CONCLUSION

In this paper, we experiment with the idea of providing powerful recommendations ranging from discovery to high precision across an arbitrary space of multimedia assets. We discuss a unified framework that aggregates consumption data from multiple sources and fuses them with meta-content to obtain more accurate content and user representations. The explicit ratings are used to convert the implicit watching user behavior to a notion of "likeness" based on ground-truth. The usage information is then used to feedback into the meta-content to determine more accurate weights of the individual meta-content factors thereby enabling richer content-content recommendations.

Methods within the unified framework were applied to MovieLens data to frame some common issues involved in the tradeoff between discovery (of hidden gems) versus recall/precision. Both watching WVCF-watch and liking WVCF-like models provide further flexibility to personalize recommendations. By budgeting recall to within the same popularity bins, hidden gems are not as likely to be ignored due to a prediction of not watching caused by a lack of awareness of the (unpopular) show's existence. Modifications to scoring functions were discussed to better personalize recommendations to best fit each user's watching preferences for maximal viewing enjoyment. An example fusion of word vectors to explicit ratings usage was provided based on a specified vocabulary, with patterns provided overall, for most popular shows, and gems. This is complementary to tag fusion, as in this case each vocabulary word was universally applied to each show.

There are several interesting directions in which our research can continue in the future. Care was taken to seamlessly convert all available data spaces, including metadata, into factored components, but integrated local effects such as [8] as well temporal effects can be better added. Even though the choice of gradient boosted trees works well for WVCF-watch and WVCF-like, other local correctors and stacking may further be beneficial. Another key issue is extensive data of explicit ratings are usually not available in linear content. Additional ways may exist, such as latent factor and nearest neighbor combinations, to close the gap between an arbitrary notion of "liking" in implicit space and explicit and/or ground truth. Another area of research is to further formalize the notion of "discovery" and relating it to likeness and precision. Further modifications of recall scores based on popularity biases between data spaces may help also. It will be interesting to see how this can then be used to create a parameter knob that can be controlled to either increase discovery vs. precision in recommendations.

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 Engg, IEEE Transactions, 2005. 17(6): p. 734-749.

[2] Melville, P. and V. Sindhwani, Recommender Systems. Encyclopedia of Machine Learning, 2010.

[3] 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.

[4] Su, X. and T.M. Khoshgoftaar, A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009.

[5] Musto, C., Enhanced vector space models for content-based recommender systems, in Proceedings of the fourth ACM conference on Recommender systems. 2010: Bari, Italy.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems 26, pages 3111–3119.

[7] O. Jojic, M. Shukla, N. Bhosarekar, *A Probabilistic Definition of Item Similarity*, Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011.

[8] Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model", Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008.

[9] *GroupLens Research*. Available from: http://grouplens.org/datasets/movielens

[10] *Vocabulary word list*. Available from: https://myvocabulary.com/word-list/emotions-feelings-mood-vocabulary