BlackBird

A New Recommender Service

Contents

Note: text that is highlighted in yellow means it's up for debate.

Current recommender system status

The current recommender system (RS) is Chat. It presents some dilemmas for the product and engineering teams:

1. It's based primarily off of Classic usage data which is:
    a. difficult to source
    b. doesn't map cleanly to the New Safari library
    c. may not reflect the behavior of the New Safari readership
2. It's running an old version of Django that we don't support anymore (1.6)
3. It's based on a software stack (MySQL, Solr, Redis, Celery, Mahout) that isn't really in our wheelhouse and can be maintenance-intensive in the case of Celery.
4. Its methodology for deriving rankings from user behavior doesn't have a strong mathematical or statistical basis
5. The opinion of SBO employees towards our current recommendations is almost uniformly poor.
6. Users are a bit more optimistic, but opinion is still pretty low. In a recent survey of ≈ 400 New Safari users, 39% of users were satisfied with the quality of recommendations, 52% don't make use of them or haven't seen them[1] and 8% actively don't like them.

Blackbird as a replacement RS

Based on these concerns, there doesn't appear to be a lot that is salvageable in Chat. Therefore, this document describes the design of a recommender system which—although new—will still share some points in common with the existing RS:

- It will compute recommendations (initially and for the medium-term) using only collaborative filtering (CF)
- It will only source user ratings from "New" Safari, not Safari Classic[2]
- It will provide a RESTful api that duplicates the chat API so that it can easily be integrated with Heron and X.

[1] This clearly indicates that there are also some HCI factors that need to be addressed with our current rankings presentation

[2] Among new Safari data, the mean ratings-per-user is 53.27, the mean ratings-per-title is 5.31. This is a fairly reasonable level of sparsity and should provide adequate data for collaborative filtering.

However, it will differ from chat under the hood:

- The volume of data doesn't yet require distributed computing (à la Hadoop or Spark) so the large operational overhead these systems introduce via something like Mahout is not needed.
- Python (via numpy/scipy) is capable of fairly high-performance computing on a single-node basis and should meet our needs for the foreseeable future
- Recommendations can be precomputed and stored in Postgres and/or Redis. (rather than Solr, as they are in Chat)

The recommender system will consist of two components:

1. A python package with code for loading data, computing recommendations, and storing them in Postgres
2. A Django-based web application that will serve rankings stored in Postgres through a RESTful API that duplicates Chat's.

In the future, other types of RS besides CF can be implemented within this overall system.

Open Questions: User Ratings

Collaborative filtering requires three entities to be modeled within the input data:

1. Users
2. Items (as of now, our titles)
3. Ratings (some numeric value indicating a user's affinity for a given item)

Rating can be explicit (e.g. a user rates a video with five stars) or implicit. Although users can now rate items explicitly in New Safari, the feature is very new and there aren't nearly enough ratings in it yet. so we need to derive an implicit ranking for a title based solely on a user's interaction with it (reading, queueing, annotating, etc.)

Let's lay out some parameters for our implicit ratings:

1. Our set of ratings R is $R = U \times T$ (i.e. the cartesian product of all users U and titles T). In other words, for every user/title pair, there will be one rating.
2. Ratings must be numeric
3. Ratings must be bounded. In this case, we want all ratings to be between 0 and 1, inclusive.
4. A rating of zero indicates the user has had no interaction with the title (i.e. it's unrated).

5. A rating of one indicates what we believe to be the highest level of affinity for that user and that title.

Ratings Data Source Candidates

Candidate data sources for deriving ratings:

1. Whether the user has queued the book
2. Whether the user has annotated the book and how many times
3. Units viewed
4. Session time

Items one and two are problematic for several reasons. Practically speaking, our data is incomplete for both of these for technical reasons that won't soon be resolved. Secondly, the dataset is much more sparse: most users view more titles than they queue or annotate. This suggests that on their own these two are not good candidates. Could they be used in conjunction with three and/or four?

Units viewed is a potential measure that could be used on its own. *Pros*: It's a data source that's heavily vetted. We also know the total number of units for a particular book or video, so we can say something like "user A has read 20% of Book B." *Cons*: Users scrolling quickly through lots of pages will have a high number of units viewed, but that may not indicate an affinity for the book; in fact it may indicate the opposite. The meaning of a unit is also different for books and videos, so *if* we're treating books and videos the same for recommendation-calculation-purposes, that could cause problems.

Session time is another potential measure that could be used on its own. It's compiled from raw units viewed reporting data and—as might be expected—is strongly correlated with units viewed on a per-title basis. *Pros*: The units of measurement for books and videos is the same: minutes. It solves for the problem of users scrolling quickly through books because they can't find what they want rather than because they like the book. *Cons*: it's not as well vetted as units viewed. For books (but not videos), we don't have a way to correlate session time with how much of the book a user has read (as a percentage value). In other words, we know how many units (approx.) are in a book, but not how many minutes.

Conclusion

Although a composite rating that, for example, takes queuing, units, and session time into account might be a worthwhile task for the future, for a minimum-viable product, it's probably best to stick with one measure for the sake of simplicity. For version 1.0, let's use session time to derive our ratings. To-Do units/minute histogram

Challenges in deriving ratings

Basic Formula

The first challenge is to interpret what the amount of time spent with a title means in terms of a user's affinity for it. Distribution of reading time varies extensively among users: User A may have a mean session time of 2 minutes while user B has a mean session time of 2 hours. For that reason, we should probably stay away from trying to establish some universal standard for all users, and constrain our interpretation of the data on an inter-user basis. For a given user, we get a set of "total time spent" for each title they've consumed, ordered from highest to lowest. We specify that the title with maximum total session time (1st title) is rated as a 1.0, and scale all others accordingly. Given a set of total session times for a user's titles, S we can derive a set of ratings R with:

$$f(s) = \frac{s}{max(S)} \text{ where } f : S \rightarrow R$$

Mean Centering and Normalization

Another option that's frequently adopted to deal with intra-user differences is mean-centering: determining affinity for a book by comparing it to the mean rating. This is designed to convert individual ratings (which might be more or less idiosyncratic) to a more universal scale. This can be done in a user-based fashion or item-based fashion. In a user-based scenario, we would get the mean-centered rating $h(r_{ui})$ by taking a raw session time value r_{ui} and subtracting the mean session time value for that user $\overline{r_u}$:

$$h(r_{ui}) = r_{ui} - \overline{r_u}$$

This will produce a rating that is unbounded, and would still require the basic formula described in the previous section to be applied. An interesting property of mean-centering is that one can see right-away if the appreciation of a user for an item is positive or negative by looking at the sign of the normalized rating. Item-based mean-centered normalization for r_{ui} is given by:

$$h(r_{ui}) = r_{ui} - \overline{r_i}$$

In this case, we compare the user's session time to the mean session time of all the viewers of that title. This has the interesting effect of putting a user's session time within the context of all users' affinity for that particular title.

Let's look at an example of this in practice for *user* mean-centering:

Title	Session time	Basic Formula	Mean-centered session time	MCST + Basic Formula
A	15	0.085	-60.25	-0.598
B	176	1.000	100.75	1.000
C	12	0.068	-63.25	-0.628
D	98	0.556	22.75	0.226

Let's look at an example of this in practice for *item* mean-centering (we omit the application of the basic formula here since it makes no sense in this context):

User	Session time	Mean-centered session time
A	12	-28.75
B	68	27.25
C	120	79.25
D	15	-25.75

Another approach to normalization that build on this is z-score normalization, which linearly transforms the data in such a way that the mean value of the transformed data equals 0 while their standard deviation equals 1. An additional benefit of z-score normalization is that it takes the variance in the ratings of individual users into consideration.

Ways it can break

In some cases, rating normalization can have undesirable effects. For instance, imagine the case of a user that reads the majority of the books they view in their entirety. Mean-centering would consider this user as a "heavy reader" and any rating below these high session times would be considered as negative. However, it is possible that this user is not really a heavy reader, but is just picky about books they look at. Furthermore, normalizing on a few ratings can produce unexpected results. For example, if a user has read just one book, their rating standard deviation will be 0, leading to undefined prediction values. Similarly, if they've read just two

books for one minute each, the same problem will occur. Nevertheless, if the rating data is not overly sparse, normalizing ratings has been found to consistently improve the predictions [5].

Do we want to go down a data normalization road? Should it be applied before or after the basic normalization formula? Usually it's after.

Negative Rankings

Explicit ratings systems can sometimes provide a method for users to supply negative ratings for an item (e.g giving it a □ or □). It's possible that we could interpret very short session times as a negative affinity for the book (in practice, this is sort of what mean-centering would do). There are some problems with this approach:

1. This requires a degree of interpretation that may not be warranted
2. Users do not perceive having immediate benefits from giving negative feedback to the system [1]
3. Negative rating correlations are less reliable than positive ones. A strong positive correlation between two users who have both rated a book highly is a good indicator of their belonging to a common group (e.g., data scientists). But while a negative correlation may indicate membership to different groups, it does not tell how different these groups are.
4. Some quantitative analyses have found that negative correlations do not provide a significant improvement in the prediction accuracy of the RS [2, 3]
5. Items with polarized recommendations (i.e. all bad or all good) tend to average out to a middle-of-the-road rating, which isn't necessarily accurate.

For these reasons, we'll avoid negative ratings for the time being since their meager benefits don't outweigh the danger of making assumptions about what "negative affinity" means in this context.

Temporal Decay

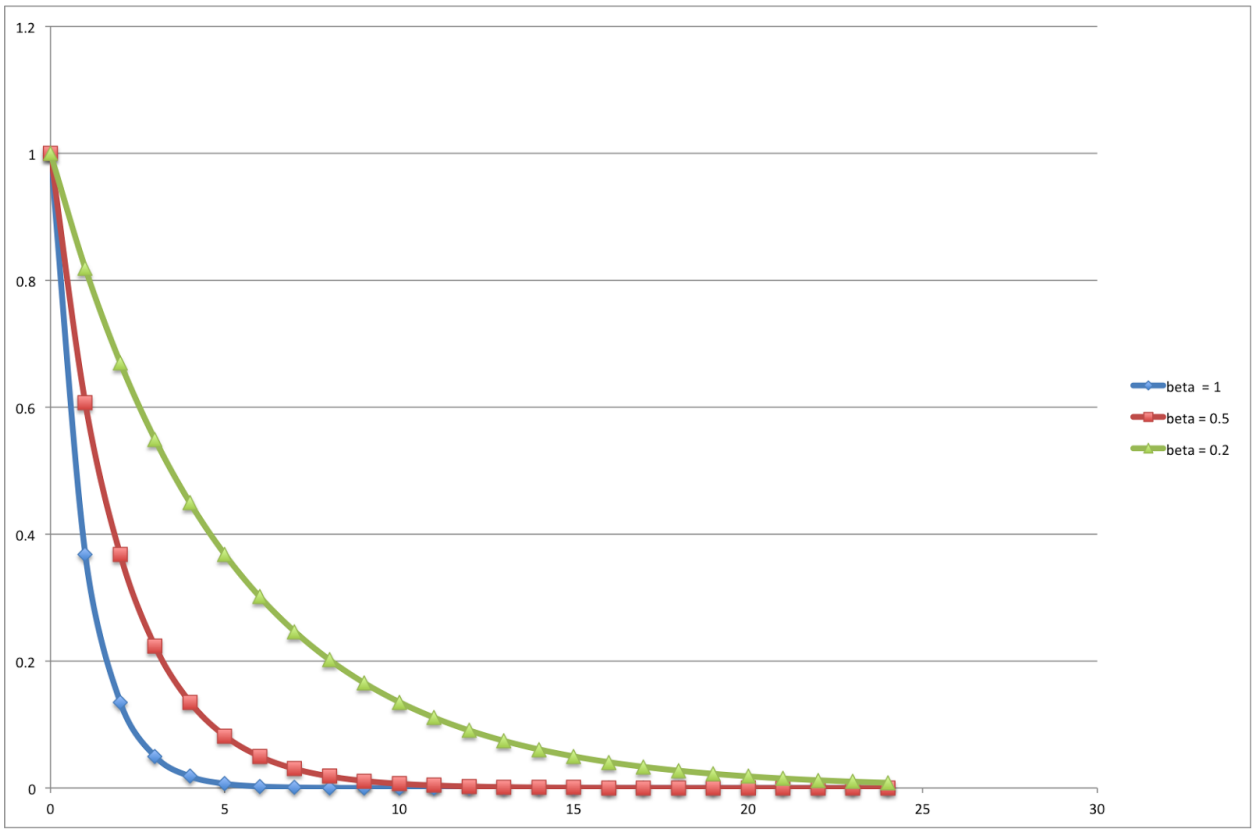
The formula in the "Basic Formula" section assumes that the value for a given member of S is just the sum of all the individual session times for that user and title. However, let's consider a situation where a user spends 50 minutes viewing a title on 3/12/2014 and then 5 additional minutes viewing the same title on 1/31/2015. If we calculate a rating 2/1/2015 we could simply say that this user has spent 55 minutes with the title and calculate rankings accordingly. But does the fact that 90% of the time spent with this title happened more than 9 months ago affect how much affinity the user has for the title now? User preferences drift over time, so it's important to introduce temporal aspects into CF models.

A mechanism for temporal decay (i.e. one that ages the interests as expressed by the user) is common in many recommender systems. This is frequently done by applying a cost function that is focused at the current state of the user (at time *t*), while de-emphasizing past actions, frequently by applying some exponential multiplier. A good baseline method adapted from [4] is to use exponential decay formed by the function

$$e^{-\beta_u \cdot \Delta t}$$

Where Δt gives the time period that has elapsed since the session occurred. Meanwhile $\beta_u > 0$ controls the user-specific decay rate, and "should be learned from the data". Some users are more consistent than others, allowing us to relate their longer term actions to current preferences more strongly. This number, then, should model the consistency of the user's behavior over time, approaching 1.0 for users who are the least consistent and 0 for those who are the most consistent. Our goal here is to distill accurate values for the item-item weights, despite the interfering temporal effects.

The chart below illustrates how changing values for β_u affects the output of the cost function:



How do we derive a value for β_u? Frequency of usage? Frequency of session time? Spread of the session times (δ) within the session for u? Do we decay at all?

Variation in Title Length

One problem with using session time is that titles—whether videos or books—will take varying lengths of time to consume. Consider the following example: title A is a mere 50 pages, while title B is 500 pages. A typical rule of thumb is that a person can read 250-300 words per minute, or one page per minute. The majority of users will max out at 50 minutes total reading time for title A, unless they've read the entire thing more than once. Title B on the other hand requires 500 minutes of a user's time to complete. If a user U reads 50 minutes worth of title A and 60 minutes of title B, would we be justified in saying their preference for title B, of which they've read approximately 12%, is greater than title A, which they've likely read in its entirety?

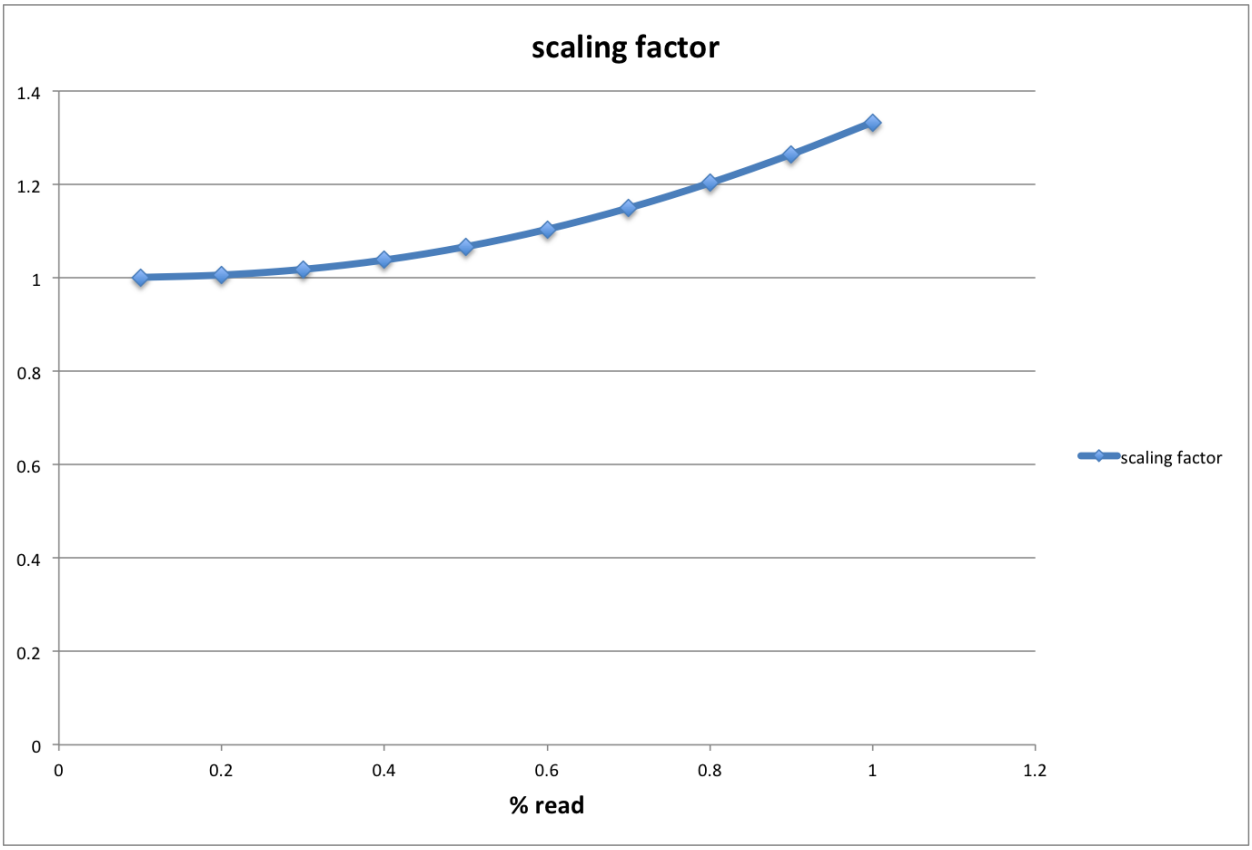
In other words, going by session time gives an advantage to longer books and videos; there is simple more of them to consume. This is actually a problem whether we're looking at units viewed or session minutes.

We might be tempted to compensate by replacing minute spent viewing or units viewed with *percentage consumed*. In this case for our hypothetical books A and B and user U, we'd feed the ratios 1.0 and 0.12, respectively to our rankings formula, rather than 50 and 500. Our set of ratings R would then be 1.0 and 0.12. However, this attempt to compensate really just flips the bias in the opposite direction, boosting shorter books at the expense of longer books (which users are less likely to finish).

We might address this by applying a scaling factor that boosts a rating as a user completes more of a book. The following formula doesn't necessarily have a valid mathematical or statistical basis, more of a proof of concept. A multiplier s for the base score is derived from the percentage of the title completed, p expressed as a decimal number.

$$s = ln(p)^3 + 1$$

Users who complete 1% of the title will have a base score multiplier of 1.00087, while a user completing 100% of a title will have a multiplier of 1.33302. We'll probably want to cap the multiplier at 100%. Otherwise, the algorithm might be open to manipulation by a malicious user.



Is this a good way to address this problem?
TF-IFD the problem?

Variations in title session time

If we see a distribution of session times like this for a title, should we be concerned? No.

User A	15 minutes
User B	5 minutes
User C	7 minutes
User D	250 minutes

Open Questions: Neighborhood or Model-Based

Neighborhood-Based CF

The first generation of CF recommenders stored user ratings in memory and used that data directly for generating the recommendations (for that reason these are also known as memory-based approaches). Neighborhood-based CF still enjoys a lot of popularity and has the advantage of being simple, efficient, and stable.

Within neighborhood-based methods, there are two approaches, user-based and item-based

User-Based

We predict the rating r_{ui} of a user u for a new item i using the ratings given to i by users most similar to u, called nearest-neighbors. We take the average rating given by these nearest-neighbors to i as our predicted rating for the u user.[3]

Another approach finds the most likely rating given by a user u to an item i, by having the nearest-neighbors of u vote on this value.

The first of these, with it's use of a weighted average, essentially solves a regression problem. The second, with having neighbors vote on a rating, treats the prediction as a classification problem. The choice between implementing a neighborhood-based regression or classification method largely depends on the system's rating scale. For continuous ratings—*which is what we are proposing to produce*—a regression method is more appropriate. For the regression approach, as the number of neighbors used in the prediction increases, the rating r_{ui} predicted by the regression approach will tend toward the mean rating of item i. Suppose item has only ratings at either end of the rating range (very close to 0 and very close or equal to 1.0). i.e. it is either much-read or little-read. The regression approach will make the safe decision that the item's worth is average. This is also justified from a statistical point of view since the expected rating (estimated in this case) is the one that minimizes the RMSE. For this reason classification approaches may result in more serendipity, since they try to classify an item as at least one the discrete rating being voted on (e.g. if 51% of neighbors vote for a five-star rating, and 49% vote for a 1-star rating, the winner will be 5 stars).

[3] Note that this explanation simplifies a lot of things because we really need to do a weighted average, but is good enough for now

Item-Based

Here we predict r_{ui} using the similarity between i and other items that users who have rated i have also rated. This algorithm looks into the set of items the user has already rated r_u and computes how similar they are to the target item i and then selects k-NN items. This is done by looking at the ratings of other users for r_u and r_{ui} and picking the k items from r_u that are rated most similar to r_{ui}. The prediction is computed by taking a weighted average of the target user's ratings on these similar items.

Engineers at Amazon first described the item-based CF algorithm in a 1998 patent [6]. It was designed improve on user-based approaches in a number of ways. The most important one is that it improves accuracy *when the number of users is much greater than the number of items*, which is the case for us. Let's look at why this is:

Imagine we have 50,000 ratings R for 500 users U and 100 titles T. Assume the ratings are distributed uniformly over the titles[4]. Average ratings-per-user is $\frac{R}{U}$ and denoted by p. Average ratings-per-title is $\frac{R}{T}$ and denoted by q. We can compute the average number of neighboring users in a user-based method with:

$$avg.\ neighbors\ =\ (|U|-1)(1\ -\ (\frac{|T|-p}{|T|})^{p})$$

Given the numbers above, that works out to **499** users on average being available as potential nearest-neighbors. But how many common ratings are used, on average, to compute those similarities? We can get this with:

$$avg.\ ratings\ =\ \frac{p^2}{|T|}$$

The answer there is **100**. Let's run the same calculations for an item-based approach:

$$avg.\ neighbors\ =\ (|T|-1)(1\ -\ (\frac{|U|-q}{|U|})^{q})$$

Given the numbers above, that works out to **99** items on average being available as potential nearest-neighbors. But how many common ratings are used, on average, to compute those similarities? We can get this with:

$$avg.\ ratings\ =\ \frac{q^2}{|U|}$$

[4] This is hardly ever true in the real world, but an OK assumption to make in this explanation.

That works out to **500**. So for the item-based algorithm, we have fewer neighbors (99 as opposed to 499), but the average number of shared ratings for the neighbors we do have is 500 as opposed to 100. A smaller number of high-confidence neighbors (particularly when those neighbors have five times as many shared ratings) is actually much more preferable to a large number of low-confidence ones.

A second advantage has to do with efficiency: we don't have to scan nearly as many neighbors in an item-based method (in the above example, 400 less per user on average). Similarly, the item-based algorithm proves to be more stable in cases where users outnumber items, assuming the set of items is more stable than users. In our case it is; we collect users at a much faster rate than titles. This means we can pre-compute item similarity weights and only re-run those computations when new items are added *and rated.*

A third advantage is justifiability. It's well known that users like recommendations more when they understand why they are seeing them [7]. In the aforementioned SBO user survey, the response "Recommendations are show along with an explanation as to why they are being recommended to me" was ranked as the number one factor in increasing trust in our recommendations. With item-based approaches, we can display a list of neighbor-item titles to the user to explain why we are making a particular recommendation: "We're showing you title X because you liked titles A, B, C, and D." We could potentially even incorporate their similarity weights into the interface. With a user-based approach the nearest-neighbors are other users. This presents a real privacy concern of course, but even if it didn't the user would still have no idea who the other people were or why they matter.

Are there any cons to the item-based approach? The rating predicted for an item is based on the ratings given to similar items. Therefore an RS using this approach will tend to recommend to a user items that are related to those usually appreciated by this user. For us that means users may tend to see recommendations for titles having the same subject (or even publisher and author) as other books. In RS literature, the term serendipity refers to the property of an RS to supply unexpected and fortuitous item recommendations. Serendipity is an important way to diversify recommendations. Item-based approaches tend to fall short here. Indeed, one of the most common criticisms of the item-based approach among Amazon users is that it too-often recommends items exactly like ones the user has already purchased. Fortunately, there are methods for introducing serendipity to item-based recommendations that can be evaluated [8]. It is probably not necessary to do this for an MVP product, but worth considering as we attempt to improve recommendations quality.

If we choose to use a neighborhood-based, it should be item-to-item.

Aggregation Function

$$r_{ui} = k \sum_{u \in U} sim(u, u') r_{u'i}$$

Where sim is the function that computes a similarity between two users and k is a normalizing factor defined as:

$$k = \frac{1}{\sum_{u \in U} sim(u, u')}$$

Similarity Weighting

Any neighborhood-based approach needs a way, when scanning neighbors, to evaluate which are the *most similar*, as these should have the most predictive power. There are a few ways to do this. Let's define some terms first. We want to find a distance d between two neighbors x, y. We represent x and y as two vectors. In an item-based approach, we consider item x as a vector $x_i \in R^{|u|}$ and do the same thing for y. $x_{ui} = r_{ui}$ if the user has rated the item, and 0 otherwise.

We'll ignore distance metrics like log-likelihood (which Chat uses) or Tanimoto similarity which are both better for binary or discrete ratings values.

Minkowski Distance

This metric describes one of several specific distance measures. They have different names depending on the degree r.

$$d(x,y) = \left(\sum_{k=1}^{n} \left| x_k - y_k \right|^r \right)^{\frac{1}{r}}$$

When $r = 1$ we have Manhattan distance. When it's 2 we have the Euclidean distance. When it's ∞ we have the Chebyshev distance.

Euclidean distance, in some ways, is a better measure of dissimilarity than similarity. The coordinates that are the same are less important than the coordinates that are different. The distances are nice in that they provide an intuitive sense of proximity. In two-dimensional vector space Euclidean distance is, in fact, our normal sense of distance as measured by a ruler. If you want distance to reflect "absolute magnitude" (for example, you are interested in using the distance to neighbors have similar mean ratings values) this is a good choice.

Cosine Distance

$$cos(x,\ y)\ =\ \frac{(x \bullet y)}{||x||\ ||y||}$$

This metric considers neighbors as document vectors of an n-dimensional space and compute their similarity as the cosine of the angle that they form. In the above formula $\bullet$ is the dot product and $||x||$ is the norm of vector x. It's important to note that objects that have a large Euclidean distance can have a very strong cosine similarity. It's more difficult to explain cosine distance intuitively, but it tends select for similarity based on direction and magnitude of the vectors.

Pearson Correlation

This is one of a family of methods (and the most common) where similarity between neighbors is measured by their correlation, which is the linear relationship between objects.

$$pearson(x,\ y)\ =\ \frac{\sum(x,\ y)}{\sigma_x \times \sigma_y}$$

In other words, the Σ of the covariance of x and y, divided by their standard deviation. Formulated in terms of correlation, two users are similar if they rate the same titles high and other titles low.

The differences in the rating scales of individual users are often more pronounced than the differences in ratings given to individual items. Therefore, while computing the item similarities, it may be more appropriate to compare ratings that are centered on their user mean, instead of their item mean. Therefore, for item-to-item recommender systems, a modification of pearson is used called adjusted cosine similarity. where x and y are vectors of user-mean-centered vectors.

Other methods in the same family include Mean Squared Difference and Spearman Rank Correlation (SRC). The principal advantage of SRC is that it avoids the problem of rating normalization by using rankings instead of ratings; it works poorly on discrete values because of ties in rank, but could work well for our continuous values.

Note that computation of the covariance matrix on very large matrices can be cost-prohibitive, especially since it can't be done on a sparse matrix.

Choosing a Metric

Cosine similarity and Pearson correlation are the most popular methods. In some cases, PC and AC have been found to outperform other metrics [9]. However, the appropriate distance metric is highly data-dependent and should really be arrived at through experimentation.

Other Considerations

These issues start to get deep in the weeds, and for version 1.0 may not be useful. Everything from here on out needs to be written down...

Accounting for significance and variance

TBD

Selecting Nearest Neighbors

Once we calculate them, we still need to decide how many to select, or where the cutoff point is

Sparsity Concerns

….probably aren't but we'll need to evaluate.

Dimensionality Reduction

Model-based CF

•Bayesian Clustering
•Latent semantic analysis
•Latent Dirichlet Allocation
•Maximum Entropy
•Boltzmann Machines
•Support Vector Machines
•Singular Value Decomposition

Open Questions: Evaluating Recommendations

Although recommendation quality will always be subjective, relying on the perception of the individual users, there are some statistical indicators of recommendation quality that should be used in any experiments aimed at improving recommendation quality.

Measuring Ratings Prediction Accuracy

A common evaluative method for recommender systems is to predict the rating a user would give to an item. In such cases, we wish to measure the accuracy of the system's predicted ratings, since these are ideally a proxy for the quality of the recommendations overall. We can do this by taking a set of user/item ratings and sampling some of those ratings without replacement. Those samples become our test data set, with the remaining ratings serving as the training data. We can then generate predictions for the user/rating pairs we held out for our test data and then compare them against the actual user ratings using a metric like **mean squared error** or **mean absolute error**. [10]

Experiments which reduce the deviation of of predicted from actual from the baseline are good candidates for putting into production (perhaps on an A|B test basis).

Problems With Prediction

For most types of recommender systems, the goal is not to predict a user's rating of an item, but to suggest things the user will like. Although the first may be treated as a proxy for the second there are some differences. For example, a system may be able to accurately predict that for a given user/item pair, the user would give the item a low rating. However that doesn't mean the system is able to find items to recommend that the user will actually like, which is the real goal. For this reason we also need to employ online evaluation. [10]

Online Evaluation

With online evaluation, we look at the actual interactions of users with the system. When a recommendation list is shown, the user may select a number of items from the list. We can assume that the user has scanned the list at least as deep as the final selection. Imagine a user selects (i.e. reads or queues) items 1, 5, and 9 in their list. We assume they've seen the first 9

items, liked 1, 5, and 9 and found 2, 3, 4, 6, 7, and 8 uninteresting. We can evaluate the quality of predictions based on the sum of utilities of the selected items. Methods that place interesting items closer to the beginning of the list than others should be preferred.

References

1. Schwab, I., Kobsa, A., Koychev, I.: *Learning User Interests through Positive Examples using Content Analysis and Collaborative Filtering* (2001).
2. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: “An algorithmic framework for performing collaborative filtering”. In: *SIGIR* ’99: Proc. of the 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 230–237. ACM, New York, NY, USA (1999)
3. Good, N., Schafer, J.B., Konstan, J.A., Borchers, A., Sarwar, B., Herlocker, J., Riedl, J.: “Combining collaborative filtering with personal agents for better recommendations”. In: *AAAI* ’99/IAAI ’99: Proc. of the 16th National Conf. on Artificial Intelligence, pp. 439–446. American Association for Artificial Intelligence, Menlo Park, CA, USA (1999)
4. Koren, Y., Bell R.: *Recommender Systems Handbook*. “Advances in Collaborative Filtering” **5** (145-186) (2010).
5. Howe, A.E., Forbes, R.D.: “Re-considering neighborhood-based collaborative filtering parameters in the context of new data”. In: *CIKM* ’08: Proceeding of the 17th ACM conference on Information and knowledge management (1481–1482). ACM, New York, NY, USA (2008)
6. Greg Linden, Brent Smith, Jeremy York, Amazon.com.: “Recommendations: Item-to-Item Collaborative Filtering”, *IEEE Internet Computing*, **7.1(**76-80) (2003).
7. Nava Tintarev and Judith Masthoff. 2012. “Evaluating the effectiveness of explanations for recommender systems”. *User Modeling and User-Adapted Interaction*, 22 (4-5) (2012).

8. Sridharan, Suboojitha, "Introducing Serendipity in Recommender Systems Through Collaborative Methods" *Open Access Master's Theses*. University of Rhode Island. Paper 453. (2014).
9. Celma, Ò. *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space.* (72-73) Springer, Berlin. **(**2010)
10. Shani, G. and Gunawardana, A. "Evaluating Recommendation Systems". *Microsoft Research*. (2009)