Recommending Beers

Milestone Report

The intent here is to provide an app user with a few informed recommendations for purchase, in a situation where that user is unfamiliar with the menu or shelf of beers he or she is facing. The recommender isn't sponsored by anyone trying to give any product a leg up in the sales or ratings charts (other than if the recommender makes such good recommendations, it becomes something the app users want to pay for or to tolerate ads to use!). It's no fun making predictions/recommendations unless we have some empirical way to measure our results, so we'll state our scoring system clearly as we go. To begin with, we're building a predictor based on past ratings of users of this app. Rather than using all the ratings we have by those users to learn how each user rates beers, and then going out and waiting to test how each user rates the recommended beers tonight, it's much more effective to set aside the most recent several ratings by each user, and see how they compare to what a model would've recommended, based on the user's ratings up to that point. Even with that as a starting point, how exactly do we measure the goodness of the model's predictions?

A common way to measure predictions like these is to use the Root Mean Squared Error (RMSE). For each prediction, we take the square of how far off the prediction was from the true rating, and then we take the average of all those squares and then the square root of that average.

	predicted rating	actual rating	squared error
beer A	3	4	1
beer B	5	3	4
beer C	3.5	3.5	0
			avg = 5/3 RMSE = $\sqrt{(5/3)}$

This measure is used by many standard machine learning models as a cost to minimize when they fit their data, so it's convenient to use as a gauge of performance. But we aren't playing an accuracy game with the app user, we're tasked with recommending one beer or maybe a few beers to her, and we need a way to measure how she rated the beers we recommended vs. how she rated the beers we *didn't* recommend. To simulate this situation, it's useful to imagine that this user is looking at a menu of a set number of choices, say 10 beers. Then say we want to recommend the 3 of those 10 that we think she'll rate highest, in order. Note that we just moved the target: First we spoke of attempting to predict the ratings as closely as possible, and now our aim is to see how well we ranked the beers, and in particular how well we ranked the top 3. So how do we measure this?

An example would help:

BEER NAME	USER RATING
A	5
B	4
C	4
D	4
E	3

Suppose the user has just these 5 beers to choose from, instead of 10, and she rates them in this order.

BEER NAME	PREDICTED
D	4.266
E	4.102
A	4.085
B	3.844
C	3.766

If the recommender predicted them in this order, and we're judging it based on its top 3 picks, how should we score it?

If the user only had one chance to order, and she trusted our app, she ordered Beer D, which turned out to be her second-least-favorite one, so the recommender has failed, it appears. But upon closer inspection, it was also her second favorite, because of her tied ratings. There are generally a lot of tied ratings in this app, so we need a way to deal with them when scoring. To be fair here, a choice of Beer D as the top pick should be considered about as good as picking blindly: There were as many better beers (just Beer A) that could've been chosen as there were worse beers (just Beer E), from the user's standpoint. The choice of Beer E as a second rec, however, is an indicator of poor performance. Even after Beer D had been selected first, there were still 3 better choices than E and 0 worse ones left. This system's score should definitely be penalized, to the point that no matter how well it does on its third pick, it scores worse than just picking randomly. As it turns out, it does pick the top user pick as its third pick, but in a performance sense, the damage has already been done. It would've been much better to pick the top pick second and then the worst pick third, because the user is presumably moving down the list from top pick to bottom pick, and may never get to #3.

For each recommendation, it counts up how many beers were rated higher by the user ("fails"), as well as how possibly bad it could've done. It then removes the chosen beer from the list, removes the potentially worst pick from the "worst-case" list, and repeats for as many recs as are needed (three, in the above example and most that follow). It then adds up all the "fails", divides by the sum of "worst-case fails", and subtracts the result from 1. So in the example above, it looks like this:

Pick <—- [Choices]	Fails	Possible Fails	
D <—- [A, B, C, D, E]	1 (A was higher)	4 (E had all 4 higher)	
E <—- [A, B, C, E]	3 (A,B,C were higher)	1 (Only A would be higher than the worst picks, with E gone.)	
A <—- [A, B, C]	0 (A was chosen and highest)	1 (Only A)	
Totals	**4 Fails**	**6 Possible fails**	**Score:** $1 - \frac{4}{6} = 0.333$

A perfect score is 1 and complete failure scores 0. Random guessing scores about 0.54, and a list that has all tied scores returns 0.5 by default (there will never be any fails, but neither are there any possible fails). As it turns out, this scorer yields results that mirror the movement of the RMSE very closely, which is good, since we can't train any machine learning models by optimizing the custom score (it's not differentiable). Let's use it in a typical scenario….

Scenario 1 —

The user is new to the app, and hasn't rated anything yet. How does the recommender proceed?

Fortunately, Untappd has hundreds of millions of ratings, for hundreds of thousands of beers. Unless a beer is brand new, it has an average rating that serves as a very good indicator of how the current user will rate it. To see just how good, let's take all our users for whom we have exactly 10 ratings, pretend that those 10 beers are the menu in the user's hand, recommend the top 3 globally rated of those beers, in order, and see how those recs do with our custom scorer.

Results =====>> 413 users with exactly 10 ratings: Mean score = 0.754

Let's look at a specific case that scored 0.75, to judge those results. Here are the top recs:

beer name	global rating
Cone Wars: Idaho 7	4.03846
Wicked Haze	3.91730
The Emergent IPA	3.91379
Melon Rye IIIPA	3.90668
Citra IPA	3.81579
Humulus Unum Amarillo	3.77632
Capo Blood Orange IPA	3.71512
Fraud Alert	3.62619
Hophoria IPA	3.58289

beer name	user rating
Wicked Haze	4.25
Cone Wars: Idaho 7	4.00
Melon Rye IIIPA	3.75
Citra IPA	3.75
Fraud Alert	3.50
Humulus Unum Amarillo	3.50
Capo Blood Orange IPA	3.50
Hophoria IPA	3.50
Bullseye Pale Ale	3.50
The Emergent IPA	3.50

…and here's how the user rated the same menu.

The top 2 recommendations were the user's 2 favorites (in reverse order). The 3rd pick was actually tied for 5th, although tied for last as well.

Again, this is a user from the 413 who were scored, and it's shown here because it scored 0.75, which was approximately the mean result for the 413.

These are very useful recommendations which show how well a global mean rating can rank beers for new users.

Scenario 2 —

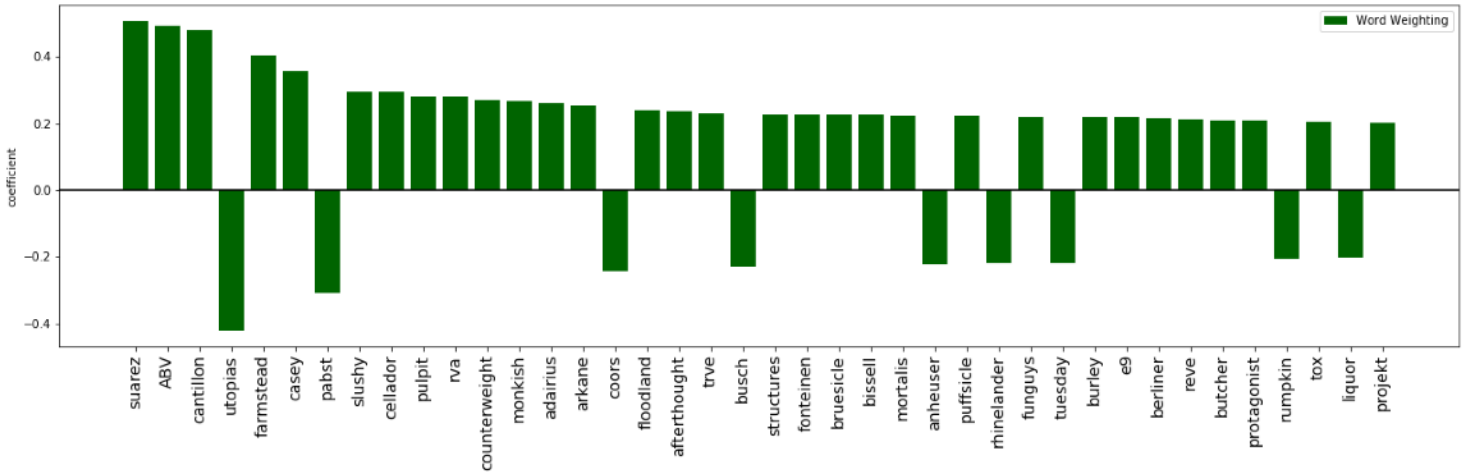
A new beer doesn't even have any ratings yet, or not enough to have converged to a stable global mean, and we'd like to predict what its mean will eventually be, so that we can know where to rank it on its menus.

One feature of beers that correlates fairly strongly with ratings is the alcohol content ("abv"), so knowing the abv should help. Beyond that, there's not much help from the numerical features in the data, but there are textual descriptions of the beers provided by the breweries, and we can use the keywords in these descriptions to help the predictions here. I used a bag-of-words representation of the descriptions, where the entire vocabulary for all the beers was reduced to ignore words that appeared in too few or too many descriptions. For each beer, a binary vector as long as the reduced vocabulary was created, with 1's for words appearing in the beer's text and 0's for the others.

Then, using linear regression, a model fit a many-thousand-dimension line to the 82K training vectors. The test predictions from that model had a RMSE of 0.187, compared to a RMSE of 0.277 for just guessing the "mean of the means". As far as how that translated to the custom scores for ranks of user menus, it took them from 0.553 for blind top 3 picks to 0.669 for the Regression model which made its predictions by weighting the key words in the new beer's description and also its abv. Recall that predictions based on *known* global means scored 0.754.

To get a sense of how the model weighted abv and individual words when it predicted unseen beers, we can look at the terms that had the highest weightings, either positive or negative. Here are the "heaviest" 40 weights, along with the words they weigh. Since the binary vectors stop counting after 1 occurrence of each word in each description, it's easy to interpret these weights and their effect on the predictions. If the word appeared in the beer's text, the prediction moved by exactly the amount indicated by the chart's bar.

Learned Weightings for Words in Beer Descriptions



For example, we can see how “busch” and “anheuser” had approximately equal negative effect on predictions, around -0.23. The reason is that Anheuser Busch is the name of a large brewer of somewhat poorly rated beers, and so the -0.46 that the model learned to associate with their brand name was split evenly between the 2 terms. It’s worth keeping in mind here, the predictions were only made on beers whose descriptions hadn’t been seen by the model previously. The words themselves had mostly been seen in other descriptions that were trained on, and the model learned that any time the word “farmstead” appeared, for example, it was associated with a +0.40 effect on the target rating. Hill Farmstead is a brewer of highly-rated beers, but “hill” is not exclusively linked to “farmstead” in the vocabulary, so “hill” doesn’t appear in the top words.

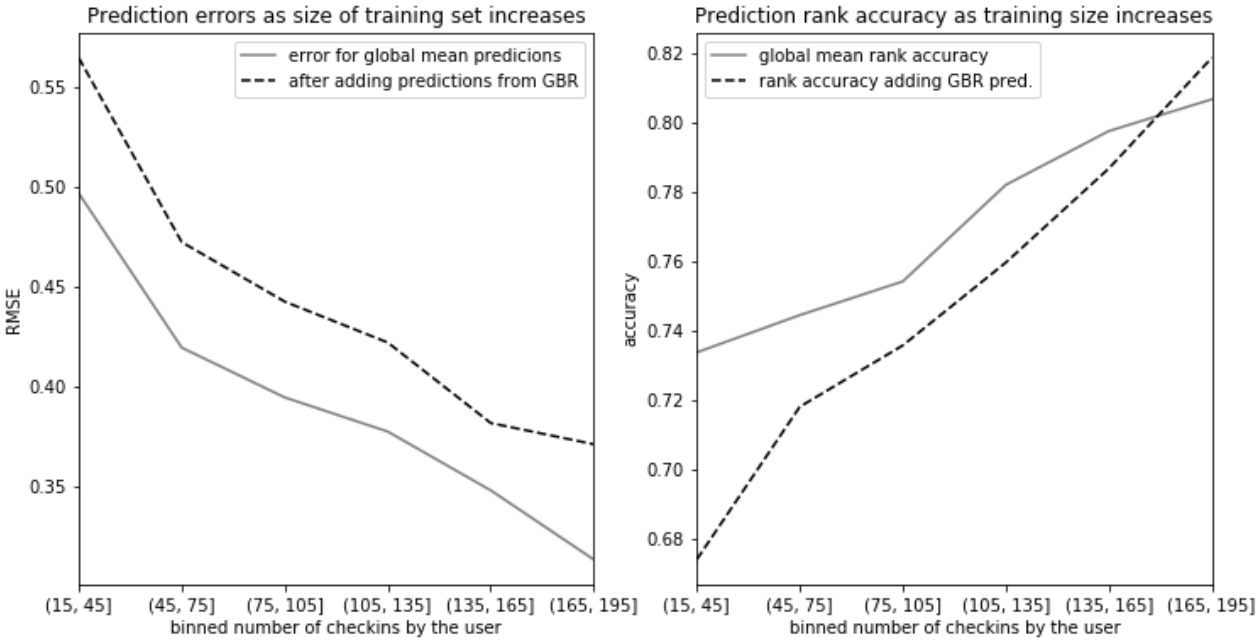
Also note that ‘ABV’ was included in the chart above to show its relative importance in predicting ratings. It’s not the word ‘ABV’, but rather the numerical column that was appended to each word vector for each beer. The beer’s abv was divided by 5.0 to scale it down to the level of the binary vector values (1’s). Almost all beers have an abv between 5 and 10, so they were scaled down to between 1.0 and 2.0 for the most part. The resulting abv weighting shown in the second bar above, approximately +0.48, tells us that a beer with 10% abv received a 0.48 higher predicted rating component than one with 5% abv, from that specific model weight. The learned intercept for the model, meaning the value it would predict for a new beer that had 0.0 abv and none of the vocabulary words in its description, was 2.99. From that starting point, beers with 5% abv got 0.48 added to their predictions, beers with “Pabst” in their descriptions got 0.3 deducted from theirs, etc.

Scenario 3 —

A user has rated a number of beers, and based on what we know about his rating history, as well as what we know about the beers on the shelf he’s staring at, we want to recommend three of the beers for him to purchase.

The first hope might be to train a model on the user’s ratings, as in Scenario 2 above. After all, it would make sense that specific users like or dislike beers by specific brewers, or with specific levels of alcohol, or with certain ingredients that are listed in the descriptions. The training is already done by the time the user is deciding, so the prediction is instantaneous. But the problem is that there’s just not a strong enough signal coming from the data for one

user, until he makes about 165+ ratings. Too many words for beers being predicted don't appear when training the model, for example. And the random noise that every rating inherently contains is not filtered out when it's just one user rating one beer, as opposed to thousands of users with varying tastes contributing to a mean rating for a beer. It's too easy for the single-user model to overfit by focusing large weights on words that just appear in one or two beers out of the user's known rating history and then appear nowhere on the testing shelf of real life. Nevertheless, as users continue to rate beer after beer, the ranking accuracy discussed in Scenario 1 above does begin to show stronger results than just picking the global favorites, as shown in the chart on the right, below:



The GBR predictions, shown by the dotted lines in both charts, refer to the predictions made by the GradientBoostingRegressor trained for each user, to fit binary word vectors that were also fit for each user individually. All users with at least 25 checkins were considered, and to keep things in line with previous scenarios, the last 10 checkins for each user were treated as the "testing menu", so that the minimum number of checkins for a training set was 15, on the left side of the 2 charts above. The sizes were binned, to smooth out individual deviations and make the trends clearer. As far as the solid grey lines in each chart, those show how predictions based solely on the global mean for the same checkins performed.

Actually, a very important point is that the globally-based predictions of the left chart, which consistently outperformed anything based on patterns learned by the GBRegressor, were in fact adjusted by amounts according to how the user's ratings were biased either higher or lower than the beers' global ratings. For example, if a user rated 20 beers whose average global mean was 3.50, and the user's average rating for those 20 beers was 3.75, we need to account for that 0.25 average "generosity" that the user seems to have. Instead of just using the global means for the 10 test beers as our predictions, we need to add 0.25 to every one of them, and that's what gives those predictions such a low RMSE, as shown by the solid grey line in the left-side chart above, which continues to strongly decrease as the number of "training" checkins increases. Now in the chart on the right, there's no need to account for that

user's bias/generosity, because it won't change the relative rankings of the 10 test beers, being that we add the same amount to each one. (A common situation wherein the user has tried and rated some of the beers on the menu would necessitate factoring in the user's bias in order to show where the previously rated beers ranked among the unrated ones, but that's beside the point here.) We ideally would like to consider the user's bias, but not divide it up evenly amongst the user's test beers, even though that works so well. We want to give more of the bias to beers that somehow are more similar to ones the user gave more of it to in previous checkins with ratings that were more above (or below, for negative biases) the global mean. Unfortunately, as the dotted line in the left-hand chart above showed, the model was never able to figure out how to divvy up the generosity of such a user. Each user's individual model was trained on vectors fit to that user's specific training beer vocabulary, and the targets were how much and in which direction each training beer deviated from the expected user rating after considering global mean and user bias. For example:

beer name	global mean	Joe's rating	Joe - world
A	3.25	3.00	-0.25
B	4.00	4.50	0.50
C	3.25	3.75	0.50
		Joe's bias—>	**0.25 avg.**

If these are user Joe's three training rows, and we aim to minimize the RMSE for any test rows, we have to account for the fact that Joe rates beers 0.25 higher than their global means, overall. But look at beers 'A' and 'C'. The global mean predictions, which do so well overall, would score the two similarly, i.e. 3.50 predicted for both (3.25 global mean + 0.25 Joe bias). Yet Joe rated 'C' much higher than he rated 'A'. With this in mind, the GBR models were fit to target the "Joe - world" column. The only other sensible option was to target the "Joe's rating" column, but that wastes too much of the model's capacity fitting itself to what it's able to learn about the global mean component of each of Joe's ratings. After fitting the GBRegressor, its predictions were added to the test beers' global ratings plus Joe's bias, yet never caught up with the lower errors for the predictions made without the model's input.

Before admitting defeat to the evil powers of overfitting and drunken ratings noise, however, let's look at the custom ranking accuracy discussed in Scenario 1, whose results for the same predictions are shown in the right-hand chart of the previous page. As pointed out before, these are the results that really matter to this recommender, given its declared purpose—to select a few beers from an available list of options for an undecided buyer. The ray of hope here is the trend on the right side of the chart, where the GBR-assisted predictions finally seem to have captured Joe's preferences in a way that will generalize to future recommendations. Not only does the dashed line cross above the global mean line around 165 checkins, it's consistently moving upwards as it does so.

This seems like a good time to ask: Why does the global mean do so well overall? It seems like a user who dislikes Anheuser Busch products should be easily identifiable by a machine learning model, such that the model could easily predict that the user would rate such a product even lower than might be predicted by its global rating combined with the user's bias. I have two suggested answers as to why this intuition isn't reflected in the RMSE results:

1. Users who don't like Anh-Busch products, for example, or low-abv beers, perhaps, are well aware of that fact. Yet their preferences never show up in the ratings, because those users avoid ordering and rating those products. Or even worse, maybe they make the bad order mistake once, and quickly learn not to repeat it, but their already-recorded dislike of beers with terms "Anheuser" and "Busch" actually worsens future predictions, since the vectors containing 1's in those corresponding indices only occur in the training checkins,

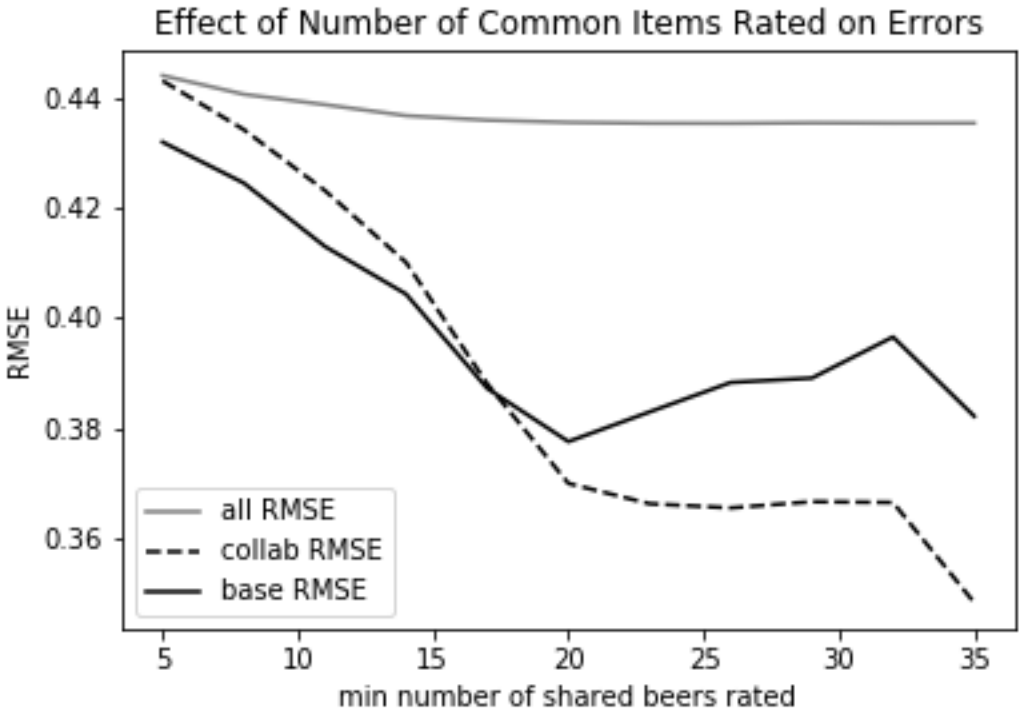
yet the same indices in the test vectors have 0's. Meanwhile, the model's learned weights for other important terms are diluted by its having been able to explain so much of a negative rating by the 1's in the "Anheuser" and "Busch" columns. This is a sort of "sample bias" or "implicit user bias," whereby the very choices the user makes in determining the training set of checkins already create a lack of randomness that skews the results in a way that's hard to account for accurately.

2. Any single checkin by any single user is subject to various types of noise— factors that are hugely important to the user's specific rating yet not seen in its data. For example, the freshness of the beer, the temperature of the beer, the type of glassware it's served in, the previous beer that was drunk by the user, the food that the user is eating with the beer, the cold that the user has, the cleanliness of the tap line through which the beer travelled on its way from the keg to the glass, the way the beer was handled and stored since being brewed, differences in how the brewery brewed that particular batch, and even the price of the beer and other elements of the checkin setting, which might influence how the user rates it. While the user-specific model can to a certain extent filter out some of the noise just by virtue of a user having checked in for 100 beers, we are only considering one checkin for each user-beer combo, so each one is subject to more variance than the collective total of all users everywhere under all possible types of noise. While our goal is to account for each user's unique tastes, it turns out that the difference between that user's preferences and the rest of the world's preferences are in fact not as great, especially once we narrow our focus to the set of users who specifically chose to use the same app and to order and rate the same beer, as perhaps are the differences between a single user's ratings due to random noise factors.

Still, in addition to harnessing the predictive power of thousands of users rating the same beer, which we've seen to be a powerful form of collaborative filtering (all these users share some sort of implicit bias which led them to order the same beer), we can attempt to sort through these users to find out which ones lean the same way as the user for whom we need recommendations, which is what the term collaborative filtering actually refers to usually. We could instead look at how the user in question rates all his beers, attempt to draw parallels between those beers and the ones on the shelf, and recommend the 3 that most resemble his highest rated beers. This is in effect what we were trying to do with the Gradient Boosting models that analyzed the abv and beer descriptions, but instead of using regression to analyze features related to each beer, we would use collaborative filtering to determine which beers were most similar. For all pairs of beers, we'd note how similarly every user who rated both beers did so, and armed with similarities for every possible pair of beers, we'd find the ones on the shelf that were closest to beers the user had rated highest.

In our specific case, we simply have too many beers and not enough users rating them, so that for any arbitrarily chosen pair of beers, a typical number of users who have rated both those beers is only 1 or 2, with a maximum of maybe 4. This is a side-effect of the way that the data were gathered; Untappd's API returned twice as many checkins per hour if you searched by user vs. by beer, so I searched by user, aiming for 200 checkins per user. What this translates to in the current project is a situation where finding the similarity between all possible pairs of beers is very ineffective, and instead we want to compute all user-to-user similarities. For any given pair of users, a typical number of shared beers might be 3 to 10, but the maximums are more commonly 20 to 30. The ~ 1 million checkins used in this project are a small percentage of Untappd's total checkins (over 900 million total, so perhaps 700 million with user ratings), so with a lot more data and computing power it seems very likely that user-user similarities would be even more effective, and beer-beer similarities should be very useful as well. Before explaining how I calculated the user-user similarities, I'll show a chart of the results, which are similar to the GBR single-user models above, where the benefits of using the method only

begin at the upper end of our user usage (in this case, when pairs of users have 20+ commonly rated beers).



The grey line that goes across at about 0.44 RMSE is the global mean baseline rating for all checkins, whether or not there were other users with enough beers rated in common with the user or not, and whether or not those users met the 0.2 similarity threshold. The reason it slopes slightly downward from left to right is that the user-user similarities that get tacked onto the global mean prediction reduce the overall error.

The solid black line that begins with the lowest error on the left shows how the global mean rating performed on its own in cases where there were user-user similarities added to it. The reason it's better than the grey line is that it tends to indicate when a user has many ratings, which in general correlates with a lower error, due in part to the user's bias regressing to his "true mean" bias.

The dashed line shows how the same checkins as the solid black line performed when a user-user similarity was added to the global base for those checkins. As mentioned before, it starts becoming useful after user pairs share 20 or more beers rated in common (as evidenced by the point at which the 2 chart lines cross). The fact that the black line moves upward between about 20 and 32 hints at the fact that there are not a huge percentage of all checkins that meet the user-user similarity thresholds as they increase, since our data have "only" a million ratings. It should be noted that it's the global baseline portion of the combined predictions that is increasing in this 20-32 area, not the similarity portion; The dashed line predictions are built upon the black line predictions, to which small tweaks are added when the similarity signal is strong enough, and that line never increases more than the black does at any point in the chart. With more data it seems very likely that the similarity tweak would be more and more helpful.

Let's not forget to take a quick look at our usual custom accuracy score with the similarity-tweaked predictions.
At the far right of the last chart, where the minimum threshold for common beers rated is 35, the rank accuracy for the black line is 0.796 and for the dashed line it's 0.804. Not a dramatic improvement, but perhaps useful. To get a better idea of what it represents, there are 7238 "test menus" of 10 beers each, and only 455 of them had any predictions tweaked. There were an average of just under 3 of the 10 beers tweaked for those 455 users.

Now for how the similarities were calculated—

Start with the deviations from expectations for every checkin. This was what the GBR previously used as a target when training. Just to make it clear, let's go back to the Joe vs. World table from above, and add one more column to it to show the target value.

beer name	**global mean**	**Joe's rating**	**Joe - world**		Joe - world - Joe's bias
A	3.25	3.00	-0.25		-0.50
B	4.00	4.50	0.50		0.25
C	3.25	3.75	0.50		0.25
		Joe's bias—>	**0.25 avg.**		

This new column is how Joe's rating deviates from what we'd have expected, knowing Joe and knowing the beer. It's the part of the rating that we've been trying to squeeze a signal out of. It's also the "Error" part of our RMSE when we use global mean predictions.

Now calculate and store the Pearson Correlation Coefficients for every pair of users, so the recommender can look up which other users A) have tried the beers Joe is deciding upon and B) have similar enough ratings deviations to Joe that we can use theirs to predict his. I also used negative correlations, since it worked slightly better with them. Meaning that if user Jane tended to rate beers opposite from Joe, and the tendency was strong enough, then Jane's deviating positively on her rating of beer 'D' was used to slightly reduce the prediction of beer 'D' for Joe. I also slightly reduced the magnitude of the similarity tweak when the number of common beers rated was lower. And finally, the tweak is added to the global mean of beer 'D' and Joe's bias in order to get the prediction for him.

In Summary—

The overall mean rating of a beer is the main foundation of the best predictions. Where it doesn't exist, a model can be trained to predict it, with helpful results. Once a user has rated enough beers, there are other methods to improve upon the global mean predictions. These include user-user collaborative filtering for pairs of users who have 20+ beers rated in common, as well as single-user trained models that learn the individual user's tendencies and can contribute helpfully to predictions after 160 checkins or so.

With a denser user-beer matrix of ratings, and/or more ratings per user, it seems very safe to claim that these results could be improved upon.
