Photo by rawpixel on Unsplash

Evaluation Metrics for Recommender Systems

Claire Longo Follow
Nov 23, 2018 · 5 min read

Recommender systems are growing progressively more popular in online retail because of their ability to offer personalized experiences to unique users. Mean Average Precision at K (MAP@K) is typically the metric of choice for evaluating the performance of a recommender systems. However, the use of additional diagnostic metrics and visualizations can offer deeper and sometimes surprising insights into a model's performance. This article explores *Mean Average Recall at K (MAR@K)*, *Coverage*, *Personalization*, and *Intra-list Similarity*, and uses these metrics to compare three simple recommender systems.

If you would like to use any of the metrics or plots discussed in this article, I have made them all available in a python library recmetrics.

```
$ pip install recmetrics
```

Movielens Dataset

The data used in this example is the popular Movielens 20m dataset. This data contains user's ratings of movies, as well as movie genre tag.

(To increase training time, this data was downsampled to only include ratings from users who have rated over 1000 movies, and ratings of less than 3 stars were removed.)

userId	movieId	rating
156	1	5.0
156	2	5.0
156	4	3.0

Example of user movie ratings

Models

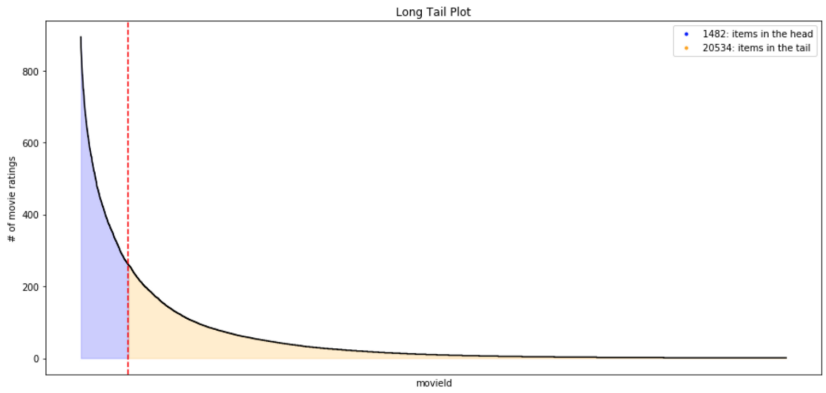
Three different recommender systems are tested and compared.

1. Random recommender (recommends 10 random movies to each user)
2. Popularity recommender (recommends the top 10 most popular movies to each user)
3. Collaborative Filter (matrix factorization approach using SVD)

Let's dive into the metrics and diagnostic plots and compare these models!

Long Tail Plot

I like to start off every recommender project by looking at the *Long Tail Plot*. This plot is used to explore popularity patterns in user-item interaction data such as clicks, ratings, or purchases. Typically, only a small percentage of items have a high volume of interactions, and this is referred to as the "head". Most items are in the "long tail", but they only make up a small percentage of interactions.

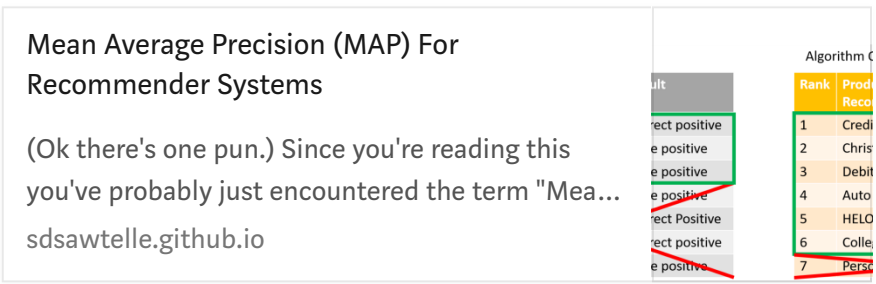


Long tail plot. (Sample of Movielens 20m ratings data)

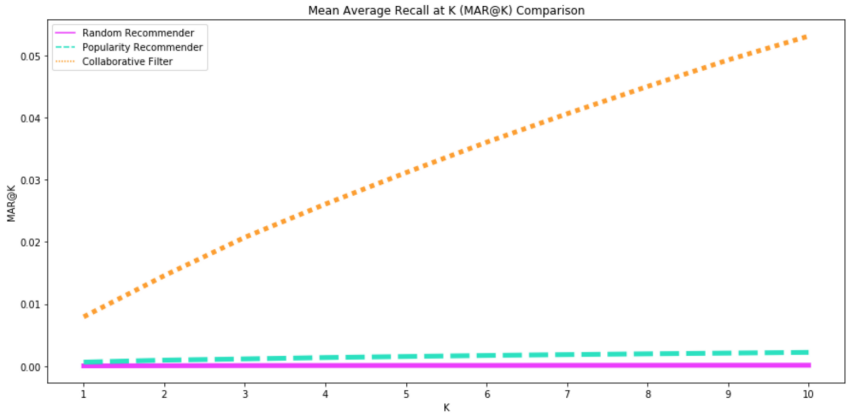
Because there are many observations of popular items in the training data, it is not difficult for a recommender system to learn to accurately predict these items. In the movie dataset, the most popular movies are blockbusters and classics. These movies are already well-known to most users, and recommendations of them may not offer a personalized experience or help users discover new, relevant movies. Relevant recommendations are defined as recommendations of items that the user has rated positively in the test data. The metrics identified here provide methods for evaluating both the relevancy *and* usefulness of recommendations.

MAP@K and MAR@K

A recommender system typically produces an ordered list of recommendations for each user in the test set. MAP@K gives insight into how relevant the list of recommended items are, whereas MAR@K gives insight into how well the recommender is able to recall all the items the user has rated positively in the test set. I will not go into detail describing MAP@K and MAR@K because a great description can be found here:



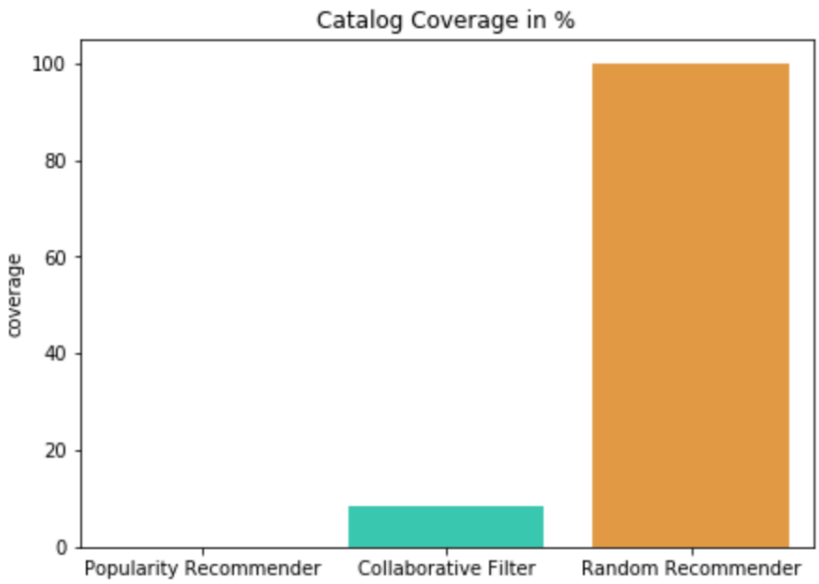
MAP@K is available in the ml_metrics library, and I have made MAR@K available in recmetrics.



By MAR@K, the collaborative filter is able to recall the relevant items for the user better than the other models.

Coverage

Coverage is the percent of items in the training data the model is able to recommend on a test set. In this example, the popularity recommender has only 0.05% coverage, since it only ever recommends 10 items. The random recommender has nearly 100% coverage as expected. Surprisingly, the collaborative filter is only able to recommend 8.42% of the items it was trained on.



Coverage comparison for three recommender systems.

Personalization

Personalization is a great way to assess if a model recommends many of the same items to different users. It is the dissimilarity (1- cosine

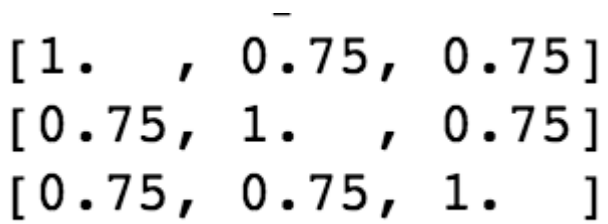
similarity) between user's lists of recommendations. An example will best illustrate how personalization is calculated.

```
example_predictions = [
    ['A', 'B', 'C', 'D'],
    ['A', 'B', 'C', 'X'],
    ['A', 'B', 'C', 'Z']
]
```

Example list of recommended items for 3 different users.

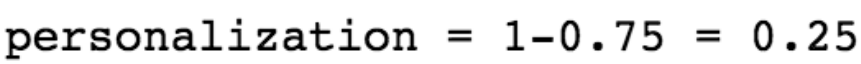
	A	C	B	D	X	Z
0	1	1	1	1	0	0
1	1	1	1	0	1	0
2	1	1	1	0	0	1

First, the recommended items for each user are represented as binary indicator variables (1: the item was recommended to the user. 0: the item was not recommended to the user).



```
[1.   , 0.75, 0.75]
[0.75, 1.   , 0.75]
[0.75, 0.75, 1.   ]
```

Then, the cosine similarity matrix is calculated across all user's recommendation vectors.



```
personalization = 1-0.75 = 0.25
```

Finally, the average of the upper triangle of the cosine matrix is calculated. The personalization is 1-the average cosine similarity.

A high personalization score indicates user's recommendations are different, meaning the model is offering a personalized experience to each user.

Intra-list Similarity

Intra-list similarity is the average cosine similarity of all items in a list of recommendations. This calculation uses features of the recommended

items (such as movie genre) to calculate the similarity. This calculation is also best illustrated with an example.

```
example_predictions = [
    [3, 7, 5, 9],
    [9, 6, 12, 623],
    [7, 894, 6, 623]
]
```

Example recommendations of movie ids for 3 different users.

movieId	Action	Comedy	Romance
3	0	1	0
7	0	1	0
5	0	1	0
9	1	0	0

These movie genre features are used to calculate a cosine similarity between all the items recommended to a user. This matrix shows the features for all recommended movies for user 1.

Intra-list similarity can be calculated for each user, and averaged over all users in the test set to get an estimate of intra-list similarity for the model.

```
recmetrics.intra_list_similarity(example_predictions, feature_df)
```

```
0.27777777777777773
```

If a recommender system is recommending lists of very similar items to single users (for example, a user receives only recommendations of romance movies), then the intra-list similarity will be high.

Using the right training data

There are a few things that can be done to the training data that could quickly improve a recommender system.

1. Remove popular items from the training data. (This is appropriate in cases where users can discover these items on their own, and may not find these recommendations useful).
2. Scale item ratings by the user's value, such as average transaction value. This can help a model learn to recommend items that lead to loyal or high-value customers.

Conclusion

A great recommender system makes both relevant and useful recommendations. Using a combination of multiple evaluation metrics, we can start to assess the performance of a model by more than just relevancy. Check out my python library if you would like use these metrics and plots to evaluate your own recommender systems.