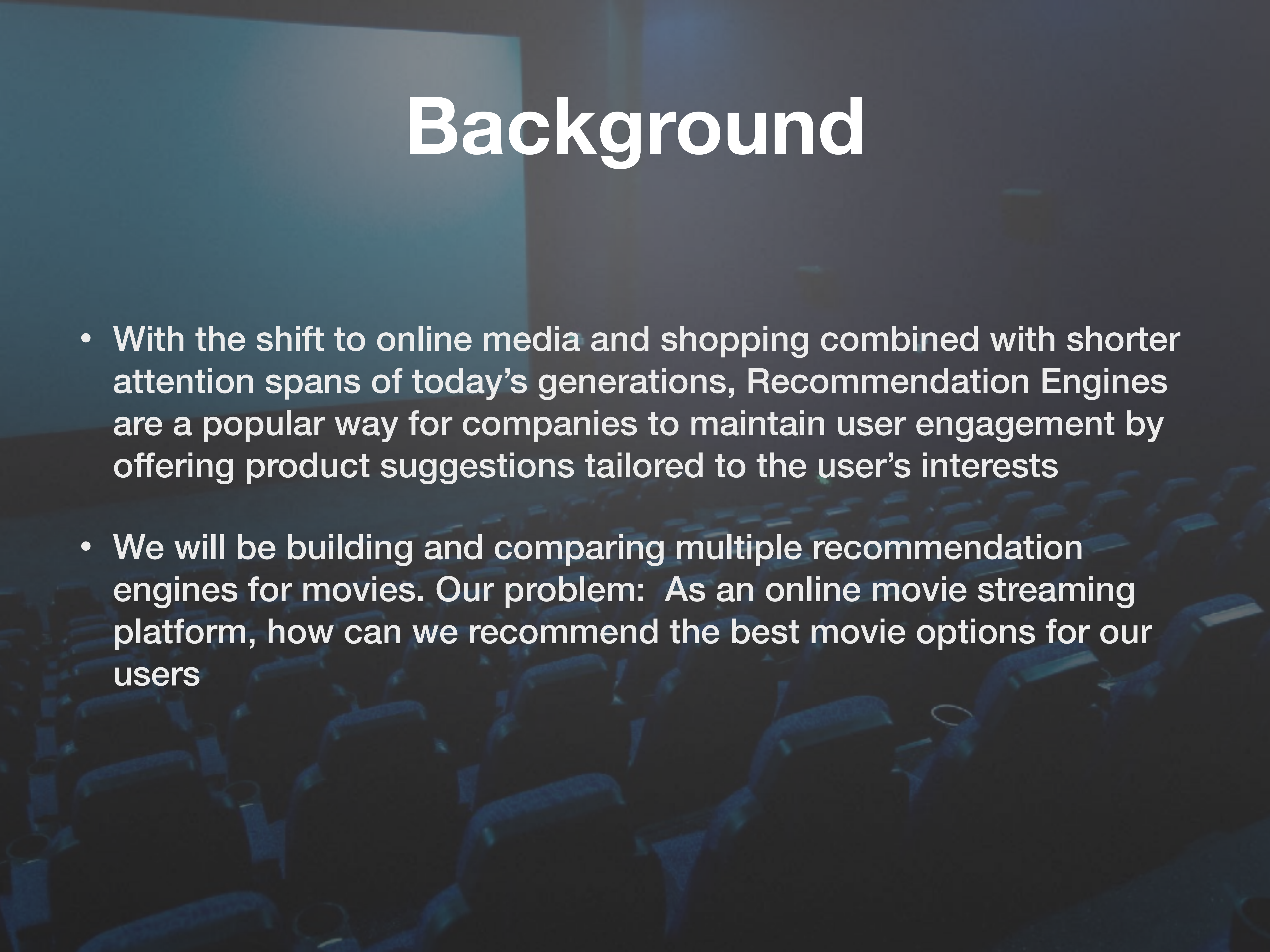
Movie Recommendations

An exploration of recommendation engines for movies

Full code implementations and Jupyter Notebook is available at
github.com/amungale

October 2017

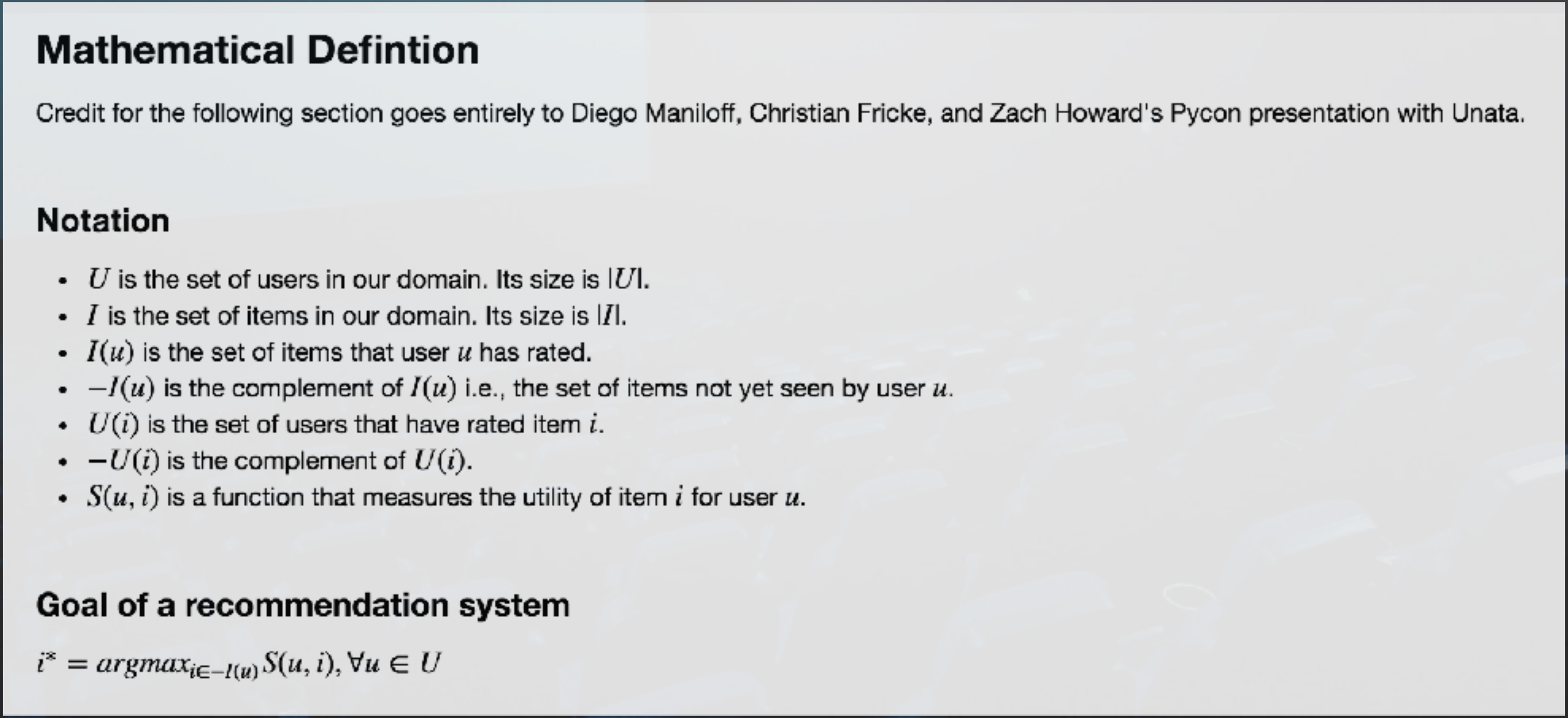


Background

- With the shift to online media and shopping combined with shorter attention spans of today's generations, Recommendation Engines are a popular way for companies to maintain user engagement by offering product suggestions tailored to the user's interests

- We will be building and comparing multiple recommendation engines for movies. Our problem: As an online movie streaming platform, how can we recommend the best movie options for our users

Background (cont.)



Mathematical Defintion

Credit for the following section goes entirely to Diego Maniloff, Christian Fricke, and Zach Howard's Pycon presentation with Unata.

Notation

- U is the set of users in our domain. Its size is $|U|$.
- I is the set of items in our domain. Its size is $|I|$.
- $I(u)$ is the set of items that user u has rated.
- $-I(u)$ is the complement of $I(u)$ i.e., the set of items not yet seen by user u.
- $U(i)$ is the set of users that have rated item i.
- $-U(i)$ is the complement of $U(i)$.
- $S(u, i)$ is a function that measures the utility of item i for user u.

Goal of a recommendation system

$$i^* = argmax_{i \in -I(u)} S(u, i), \forall u \in U$$

Approach

We will be building our recommendation engine using the MovieLens data set.

1. Naive Recommender - recommend the most popular movies. We can use these as the default recommendations

2. Personalized Recommenders (Content Based Filtering, Collaborative Filtering, and Hybrid Solutions)

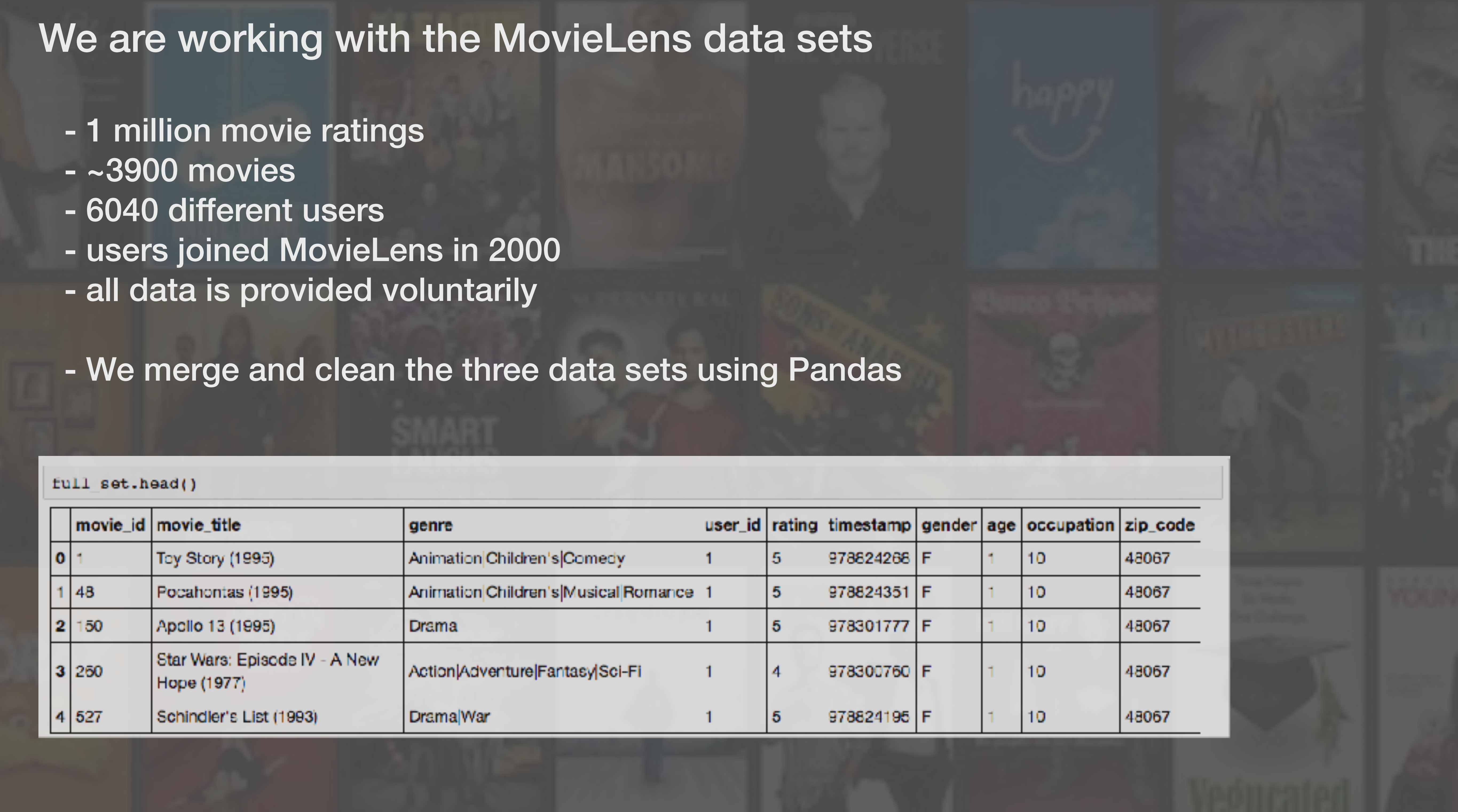
3. Improved Personalized Recommenders, continued

   a. Implicit Recommender with LightFM

   b. Comparing popular algorithms with SURPRISE

4. Conclusions and next steps - big data solutions with Spark, etc.

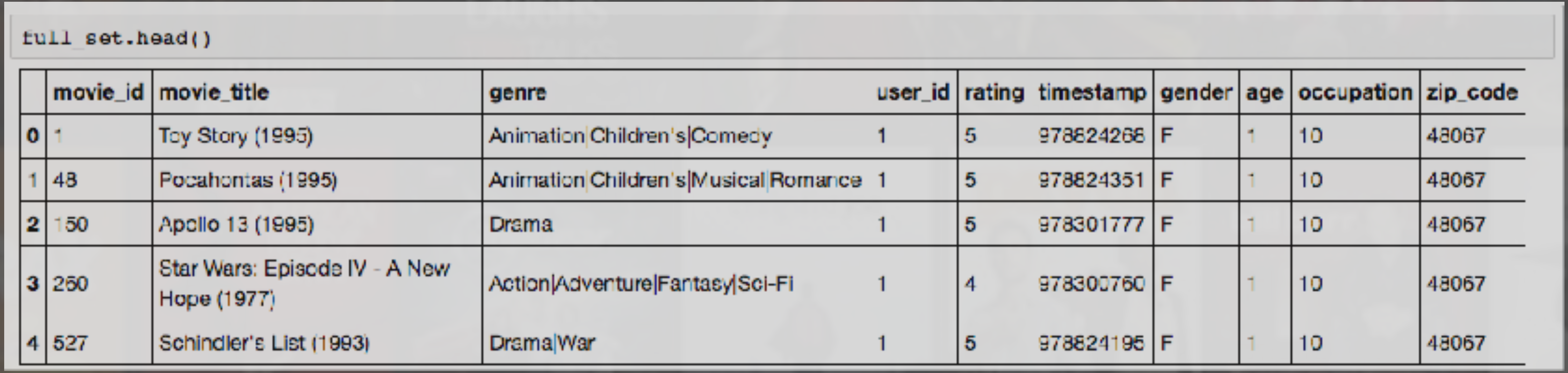
The Data



We are working with the MovieLens data sets

- 1 million movie ratings
- ~3900 movies
- 6040 different users
- users joined MovieLens in 2000
- all data is provided voluntarily

- We merge and clean the three data sets using Pandas



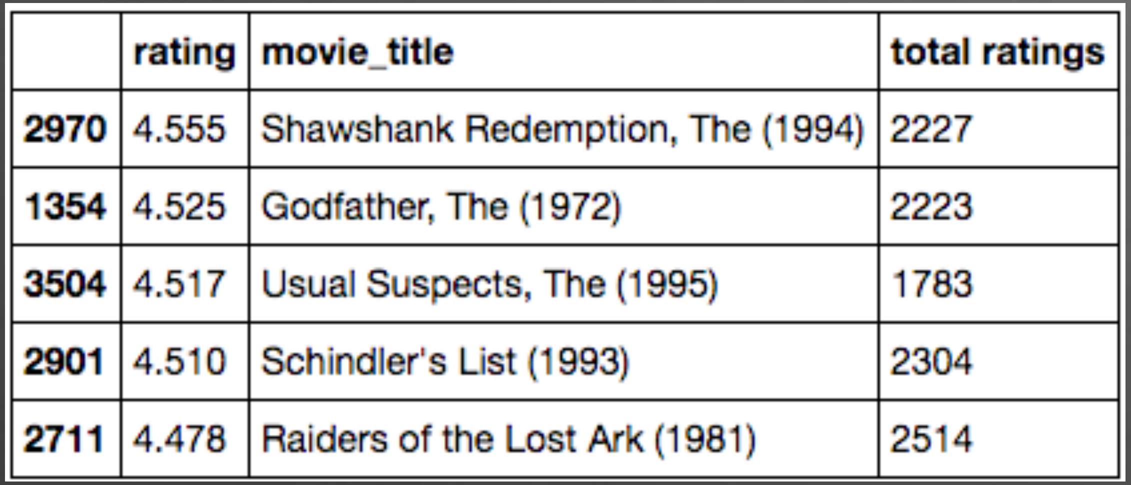
```
full_set.head()
```

	movie_id	movie_title	genre	user_id	rating	timestamp	gender	age	occupation	zip_code
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	Drama|War	1	5	978824195	F	1	10	48067

Results

Naive Recommender -

Simply find the highest rated movies with the highest number of votes. The result is a set of typically well liked films, so this is a good start to recommend to any new user:



	rating	movie_title	total ratings
2970	4.555	Shawshank Redemption, The (1994)	2227
1354	4.525	Godfather, The (1972)	2223
3504	4.517	Usual Suspects, The (1995)	1783
2901	4.510	Schindler's List (1993)	2304
2711	4.478	Raiders of the Lost Ark (1981)	2514

Personalized Recommenders -

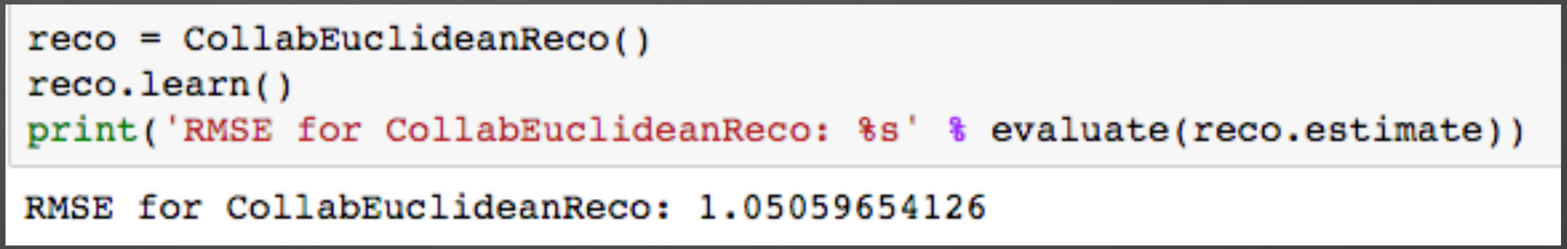
Next we will explore recommendations customized towards the user. Because our data set has 6040 unique users and 1 million movie ratings, either collaborative filtering or content bases filtering approaches would valid - we have a large enough user population to identify group preferences, and each user also has enough ratings to identify individual preferences. Therefore we hypothesize that the optimal approach is a Hybrid Filtering Solution.

Results (cont.)

Personalized Recommender -

Among the several approaches of building personalized recommender with content based and collaborative filtering, we find the best model so far is a Collaborative Filtering model using a Euclidean Similarity Function. We evaluate our models using Root Mean Squared Error, and for this approach we have
RMSE = 1.051.

$$sim(x, y) = \frac{1}{1 + \sqrt{\sum (x - y)^2}}$$



```
reco = CollabEuclideanReco()
reco.learn()
print('RMSE for CollabEuclideanReco: %s' % evaluate(reco.estimate))
```

```
RMSE for CollabEuclideanReco: 1.05059654126
```

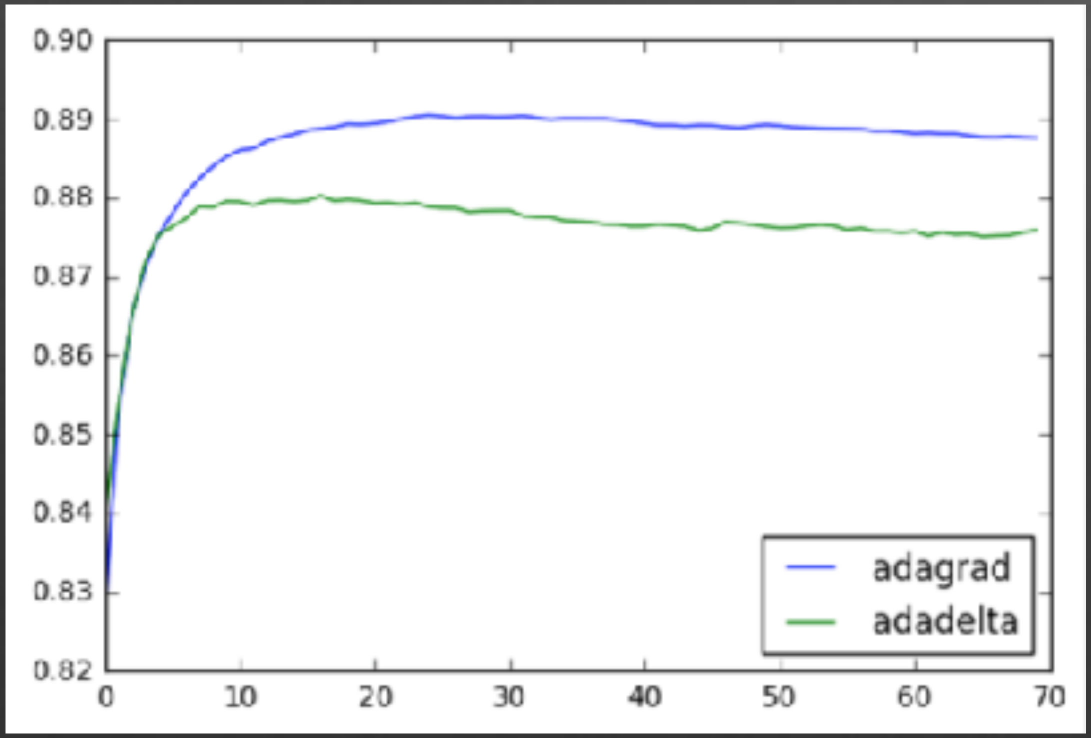
Other methods we implemented include:
- Content based filtering by user's mean ratings
- Content based filtering by user's genre preferences
- Collaborative Filtering using implicit similarity functions based on age, gender, occupation, and location
- Collaborative Filtering using other custom similarity functions like Cosine, Pearson and Jaccard

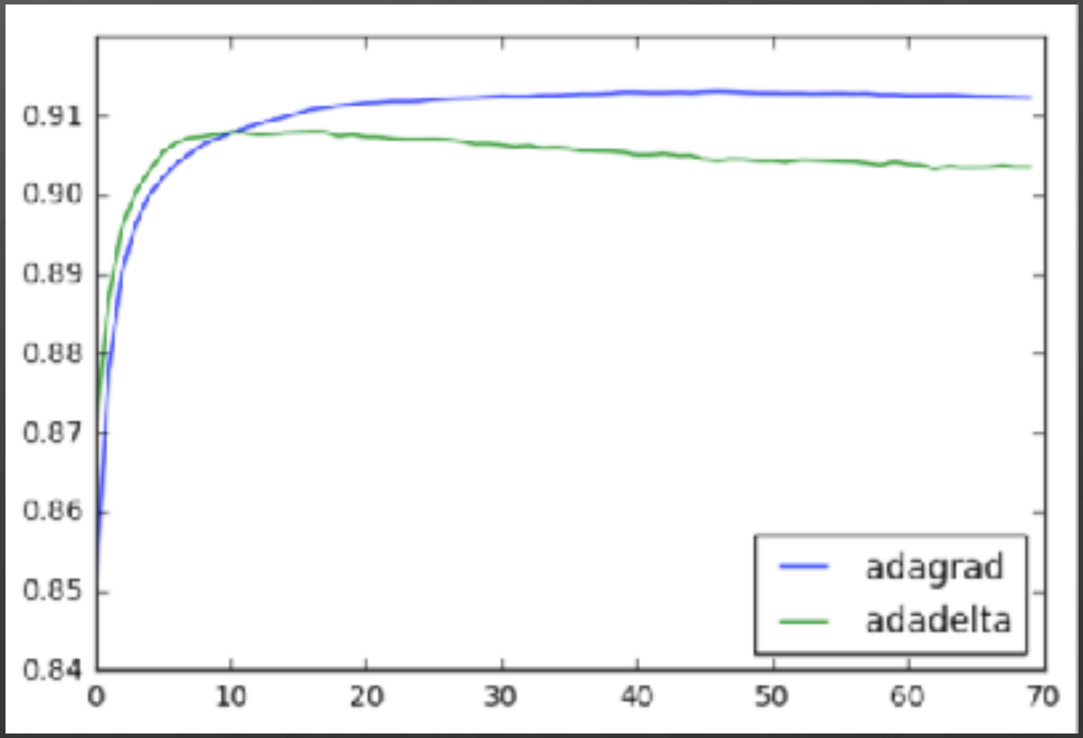
Results (cont.)

Improved Personalized Recommenders - LightFM

There are multiple recommendations engine libraries for Python that can help us quickly build powerful solutions that we will also explore, like LightFM.

LightFM allows us to build reco engines based on implicit and explicit feedback. See ROC curve screenshots (experimentation with learning rates in LightFM)

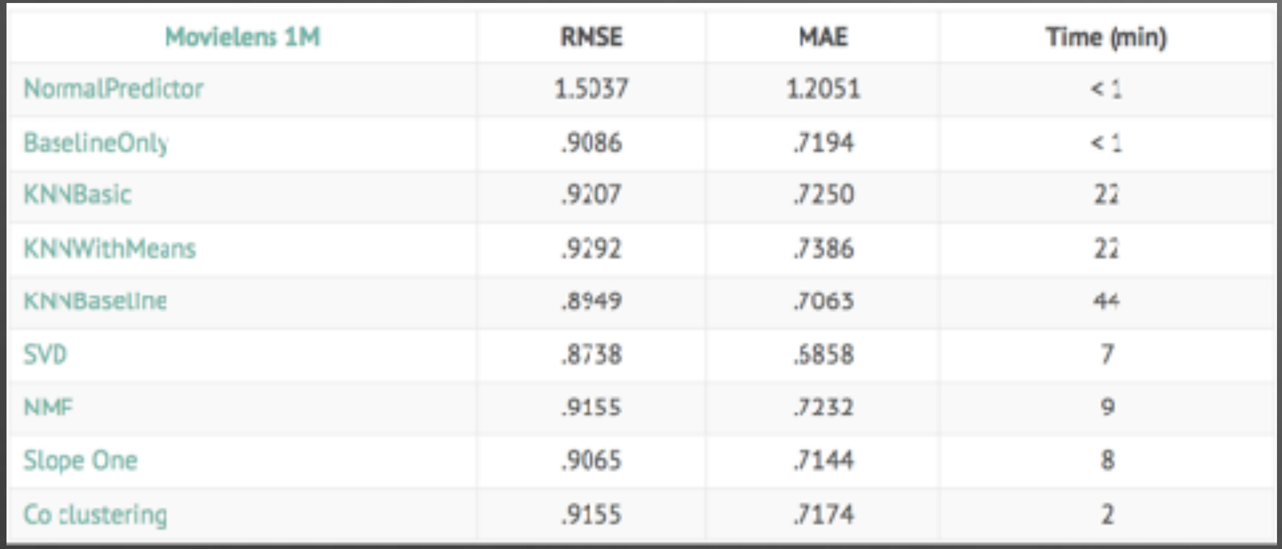




Results (cont.)

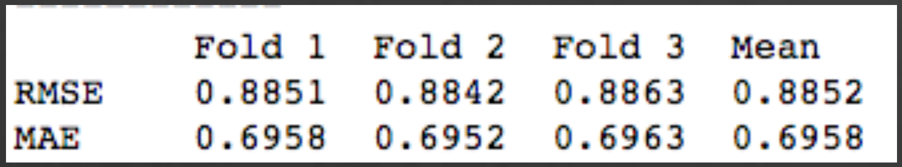
Improved Personalized Recommenders - Surprise

Another powerful recommendation engine library for Python is Surprise. Below is Surprise's average performance on this MovieLens data set using various algorithms



Movielens 1M	RMSE	MAE	Time (min)
NormalPredictor	1.5037	1.2051	< 1
BaselineOnly	.9086	.7194	< 1
KNNBasic	.9207	.7250	22
KNNWithMeans	.9292	.7386	22
KNNBaseline	.8949	.7063	44
SVD	.8738	.6858	7
NMF	.9155	.7232	9
Slope One	.9065	.7144	8
Co-clustering	.9155	.7174	2

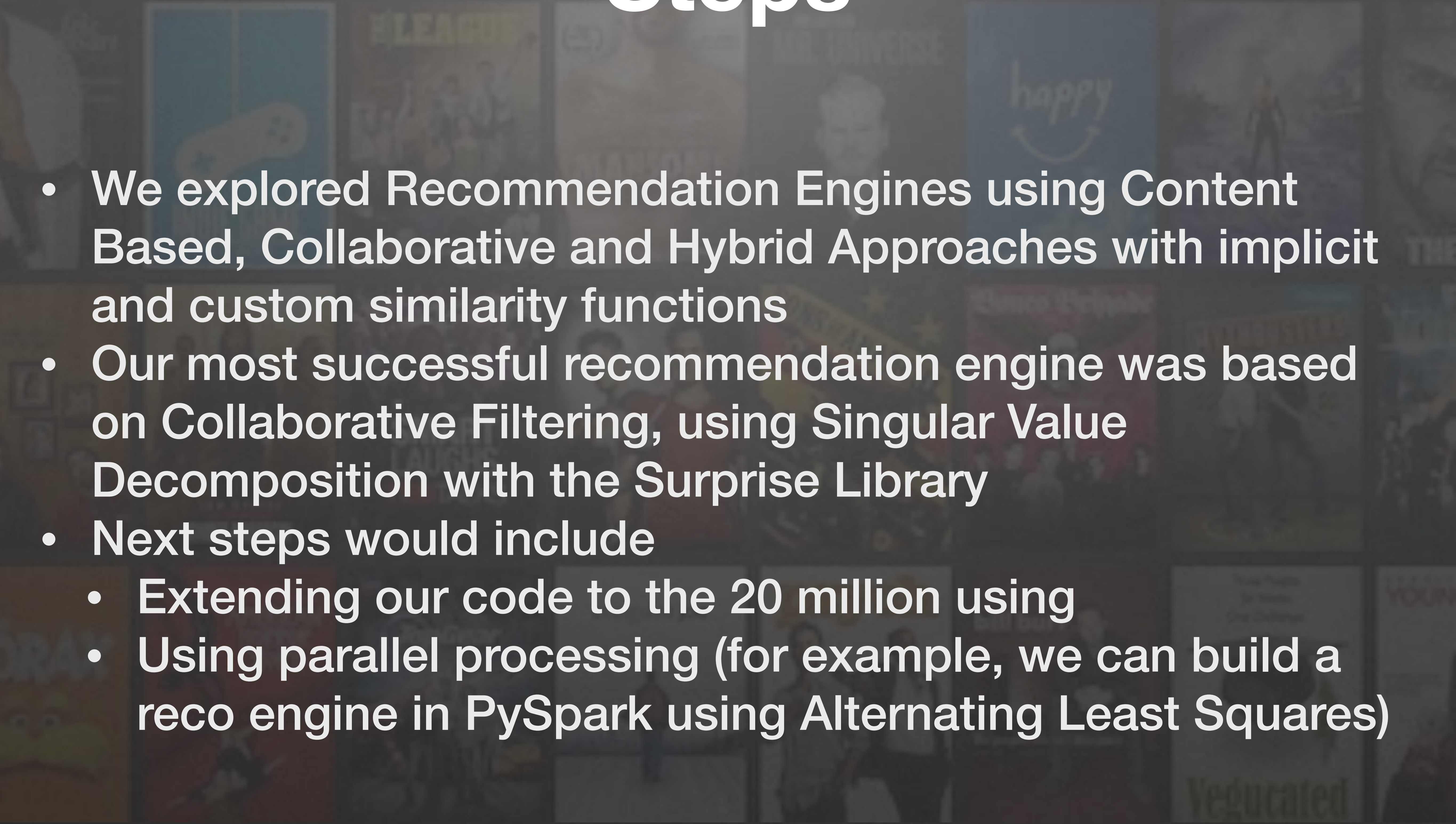
We verify this by reimplementing the most effective method for this scenario, Singular Value Decomposition. We find similar results:



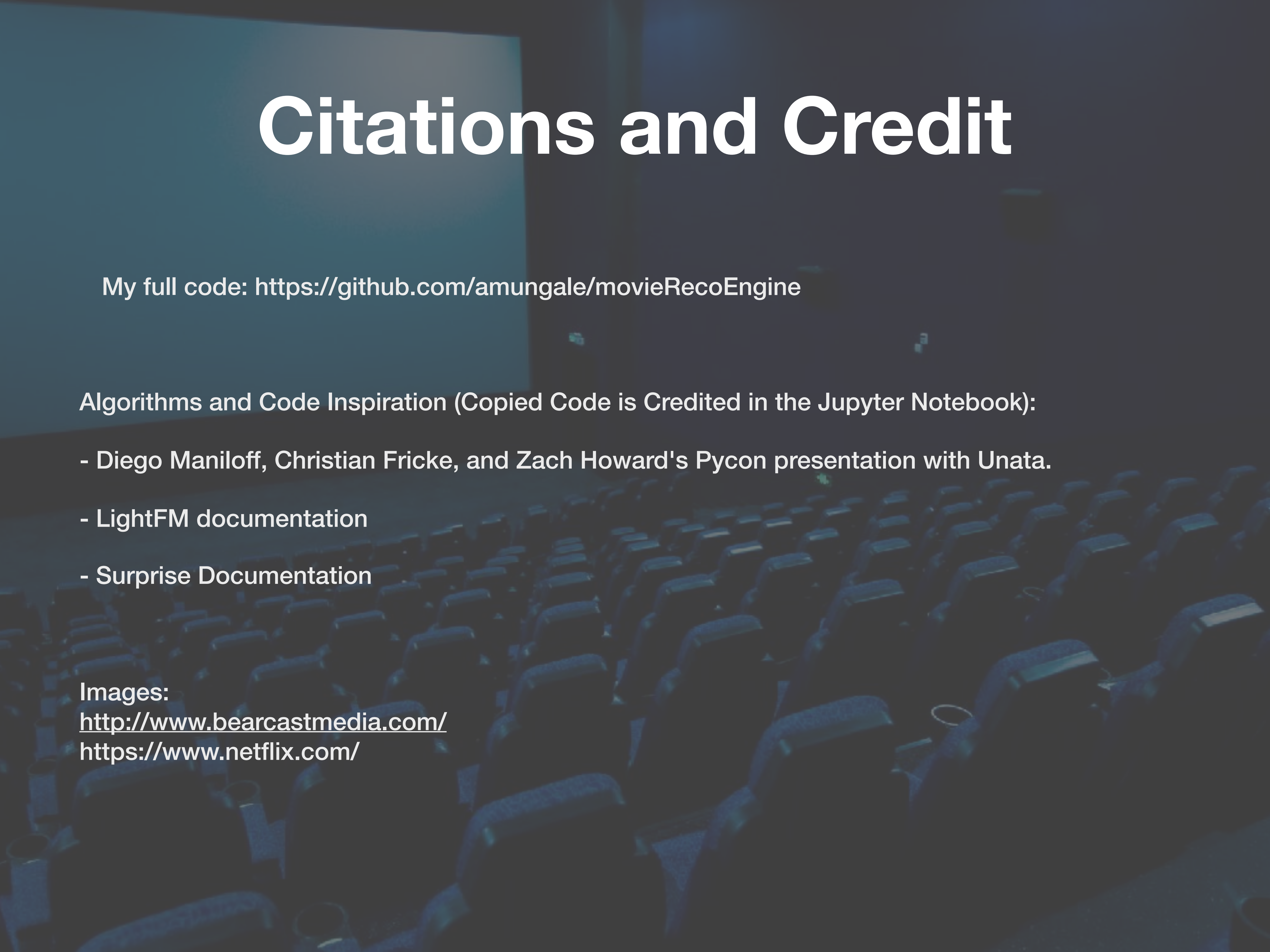
```
        Fold 1  Fold 2  Fold 3  Mean
RMSE    0.8851  0.8842  0.8863  0.8852
MAE     0.6958  0.6952  0.6963  0.6958
```

This is our highest scoring approach!

Summary, Insights, and Next Steps



- We explored Recommendation Engines using Content Based, Collaborative and Hybrid Approaches with implicit and custom similarity functions
- Our most successful recommendation engine was based on Collaborative Filtering, using Singular Value Decomposition with the Surprise Library
- Next steps would include
  - Extending our code to the 20 million using
  - Using parallel processing (for example, we can build a reco engine in PySpark using Alternating Least Squares)



Citations and Credit

My full code: https://github.com/amungale/movieRecoEngine

Algorithms and Code Inspiration (Copied Code is Credited in the Jupyter Notebook):

- Diego Maniloff, Christian Fricke, and Zach Howard's Pycon presentation with Unata.

- LightFM documentation

- Surprise Documentation

Images:
http://www.bearcastmedia.com/
https://www.netflix.com/