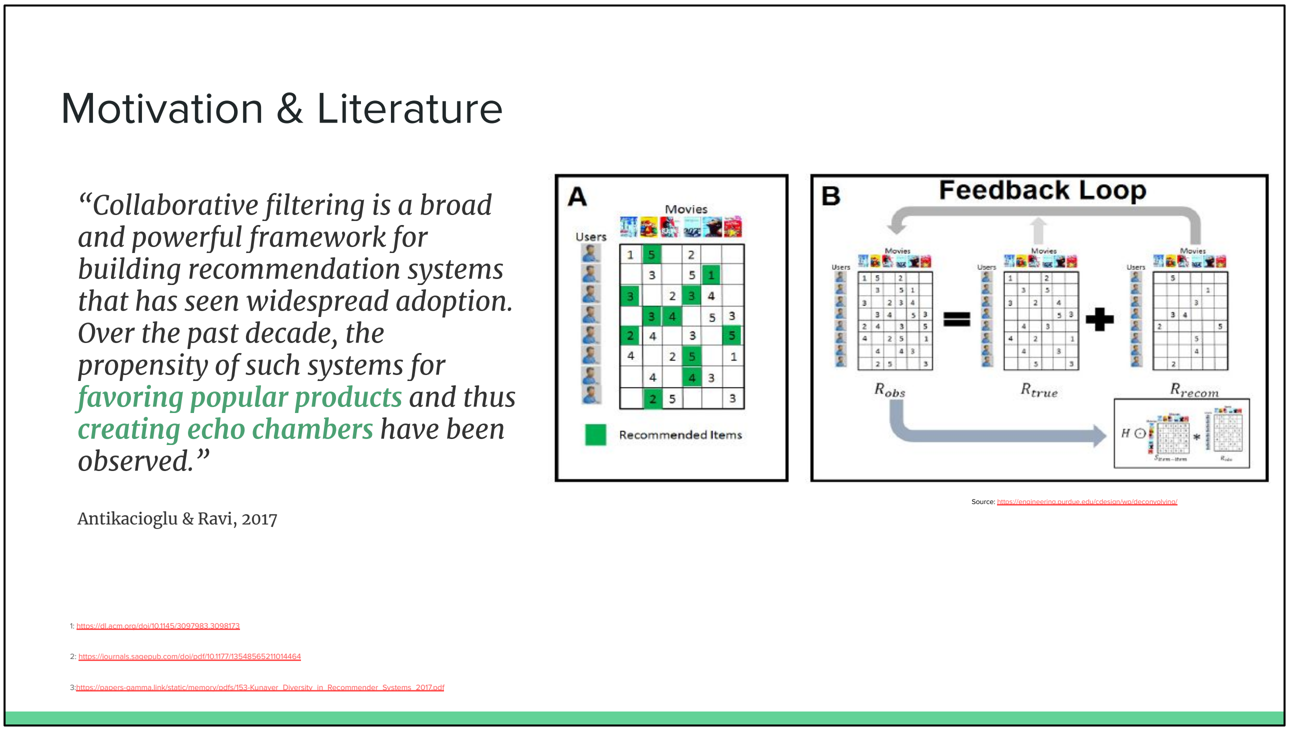
Evaluating & Diversifying Recommender Systems

Lucas Barbosa, Brittany Dougall, Rathin Bector, Jerico Johns

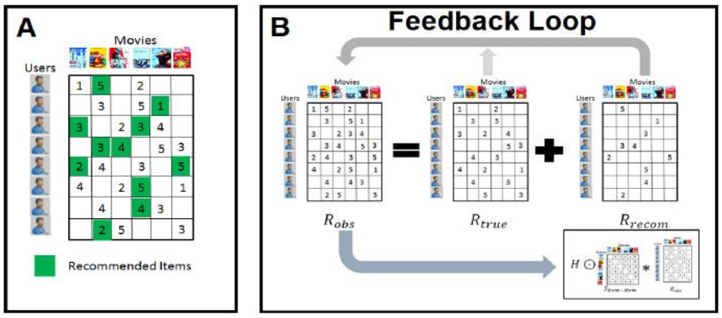
Image:
https://www.nbcnews.com/better/lifestyle/problem-social-media-reinforcement-bubbles-what-you-can-do-about-ncna1063896



Motivation & Literature

“Collaborative filtering is a broad and powerful framework for building recommendation systems that has seen widespread adoption. Over the past decade, the propensity of such systems for ***favoring popular products*** *and thus* ***creating echo chambers*** *have been observed.”*

Antikacioglu & Ravi, 2017



Source: https://engineering.purdue.edu/cdesign/wp/deconvolving/

1: https://dl.acm.org/doi/10.1145/3097983.3098173

2: https://journals.sagepub.com/doi/pdf/10.1177/13548565211014464

3:https://papers-gamma.link/static/memory/pdfs/153-Kunaver_Diversity_in_Recommender_Systems_2017.pdf

- The concern with this kind of consumption is that feedback loops reinforce a user’s pre-existing preferences, diminishing their exposure to a diverse range of cultural offerings and denying art, aesthetics and culture of its confrontational societal role. Furthermore, there is concern that NRS-generated feedback loops threaten to homogenize film and television culture, particularly considering that Figure 5. Four of the first five titles offered in the Disruptor’s ‘Exciting Movies’ row belonged to the ‘Fast & Furious’ film series. 12 Convergence: The International Journal of Research into New Media Technologies XX(X) the individual feedback loops of each user can ‘iteratively influence the collaborative filtering algorithm’s predictions over time’ (Sinha et al., 2016: 1). What this means is that the individual feedback loops each user is caught in (e.g. the Die-Hard Sports Fan’s endless loop of sports-related content) can contribute to large-scale, even global loops, because user consumption patterns are extracted by the NRS and contribute to recommendations made to other users across the world (Source: https://journals.sagepub.com/doi/pdf/10.1177/13548565211014464)
-

Contribution

Current State

1. User similarity and content preference models are **inherently biased towards popular items** and this reinforces popular items[1]

2. State-of-the-art models achieve high accuracy (HR@10 of 70%+), but **there is no direct feedback mechanism for user satisfaction or user "surprise"** when receiving recommended items[2]

3. Attempts at diversification often involve **computationally expensive operations** utilizing pairwise comparisons or complex intra-model optimization problems[3]

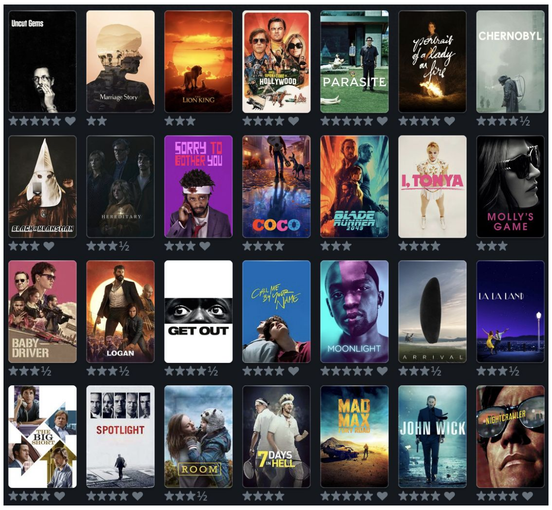
Our Contribution

1. Define **item novelty** as the diversity metric most counter to this issue and provide a definition and methods for it's calculation and implementation.

2. **Showcase the trade-off between novelty and accuracy** to demonstrate that a recommender system can maintain a threshold of accuracy, while catering to a user's explicit preference for novel items.

3. Provide a **post-processing** diversification method that can be layered onto any recommender system in an **effective** and **efficient** manner.

Data Source and Description



45,115 films

270,896 users

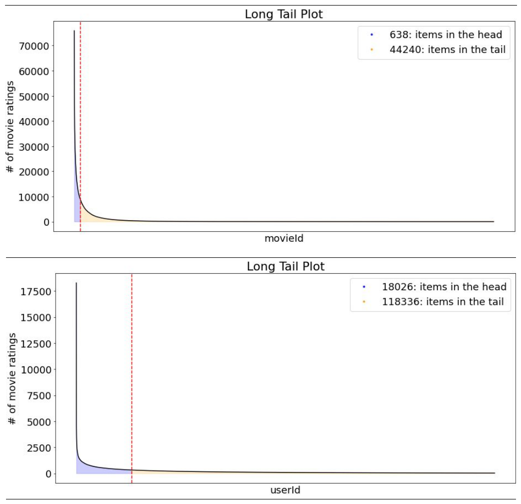
26,024,289 ratings

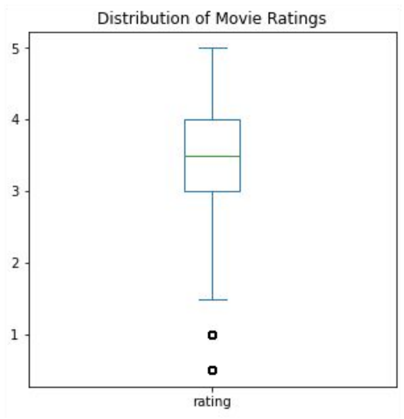
For our recommender systems, we used the popular collection of movies' datasets on Kaggle, representing over 45k films, over 270k unique users, and more than 26 M ratings. For both of our main models, we used the ratings dataset, which contains user ids, movie ids, ratings on a 0 - 5 scale, and a timestamp for the rating.

For the content recommender model, in addition to ratings, we used movies' metadata, which contains a number of features such as film id, budget, release date, title, original language, and average rating. We also used the following: credits containing the film id, cast names, and crew names, keywords containing film id and keywords describing each film, and links, containing a mapping of movie id to Imdb id.

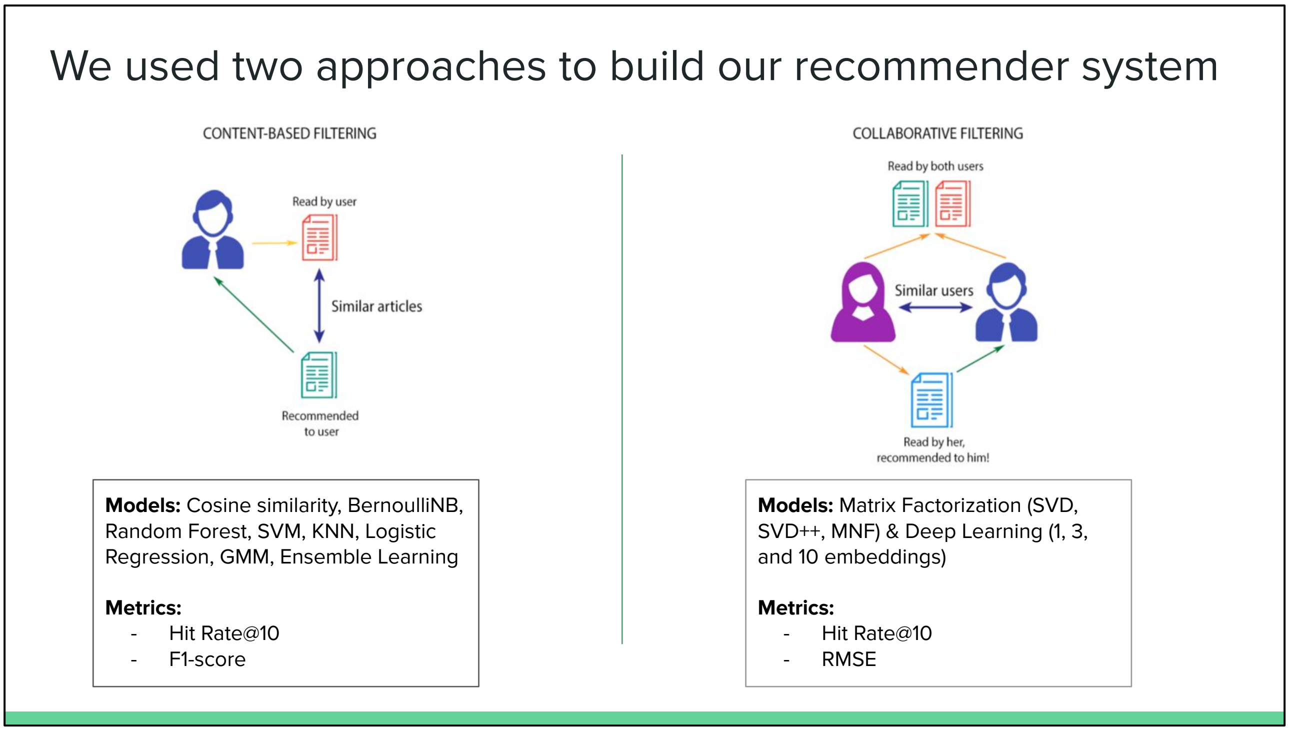
Image:
https://www.google.com/imgres?imgurl=https%3A%2F%2Fletterboxd-recommendations.herokuapp.com%2Fstatic%2Fimages%2Fpreview_image.png&imgrefurl=https%3A%2F%2Fletterboxd.samlearner.com%2F&tbnid=hRsNCzKrKKqTEM&vet=12ahUKEwjJpPulwZjyAhVdXDABHd7IAKEQMygDegUIARDHAQ..i&docid=Ob-qCLzkaERGFM&w=1118&h=1026&q=movie%20recommendations&ved=2ahUKEwjJpPulwZjyAhVdXDABHd7IAKEQMygDegUIARDHAQ

Data Stats (EDA)

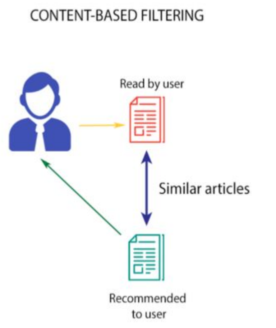




Prior to designing our recommender models, we needed to understand the distribution of our ratings data, to avoid data sparsity concerns (if users rated too few movies, it would be difficult to find similar users for collaborative filtering or predict item labels for content-based recommendations). The two long tail plots on the left illustrate potential issues in terms of novelty and data sparsity. As seen in the top left graphic, a small number of movies, popular movies, are responsible for the majority of ratings provided. There are a large number of movies in the right tail that have comparatively few ratings; these may be hidden gems that we may want to incorporate as we seek to improve recommendation novelty. In the bottom left graphic, we see a similar phenomenon among users - a relatively small subset of users provide the majority of ratings. When a rating is provided, as seen on the right, scores are fairly moderate, with the median score around 3.5.

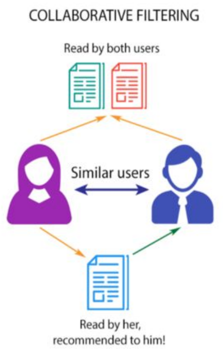


We used two approaches to build our recommender system



Models: Cosine similarity, BernoulliNB, Random Forest, SVM, KNN, Logistic Regression, GMM, Ensemble Learning

Metrics:
- Hit Rate@10
- F1-score

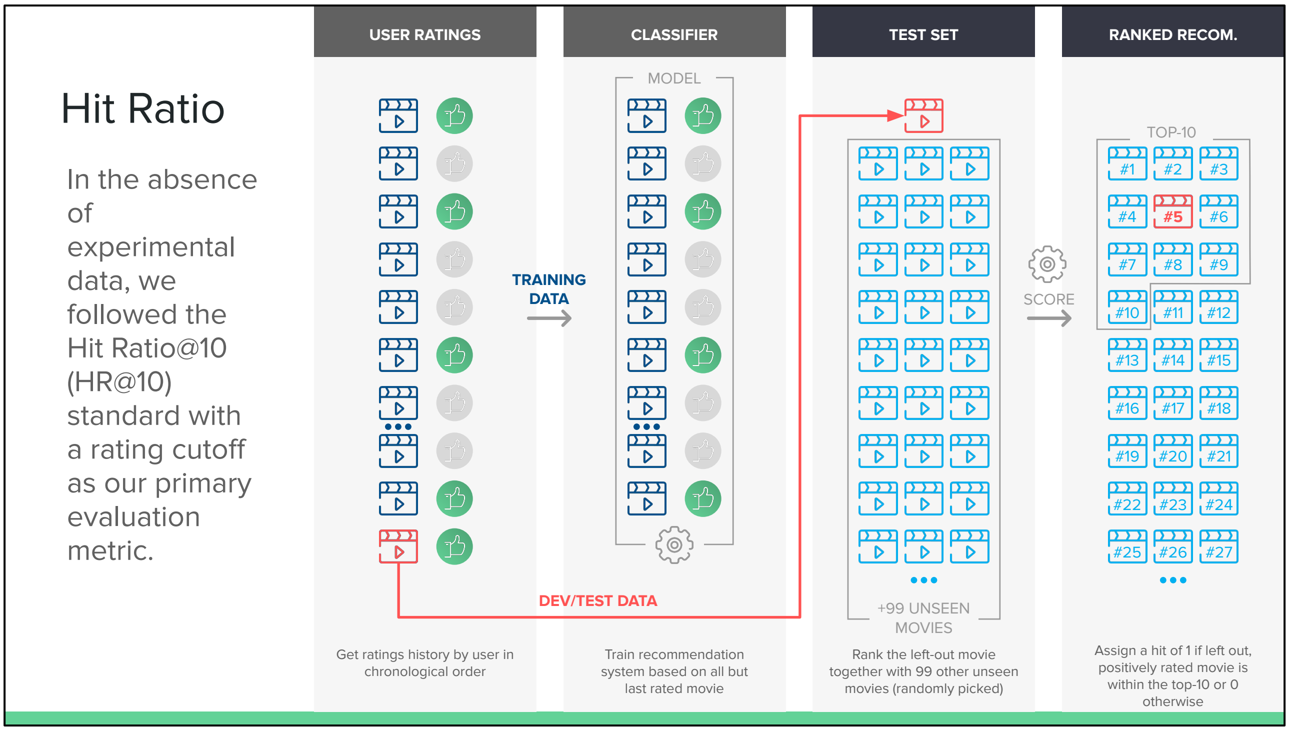


Models: Matrix Factorization (SVD, SVD++, MNF) & Deep Learning (1, 3, and 10 embeddings)

Metrics:
- Hit Rate@10
- RMSE

We used two complementary approaches to build our recommender systems. The first was a content-based approach, where we trained several classifiers in order to predict the likelihood of a given user rate well an unseen movie based on similarity of that movie metadata with other movies that the user rated positively in the past. We binarized ratings so that we ended with a binary classification problem - so the user could either like or dislike any single movie. We tested different classifiers - basically all we`ve learned in w207 - and we measured the performance using F1-score and hit rate@10, which is specific metric for recommender systems that will explain better in the next slide.

For collaborative filtering, the recommendations are based on movies liked by users that liked the same movies that my target user has liked in the past. In that case, the ratings were preserved as a continuous variable, ranging from 0 to 5, and we experimented with matrix factorization and deep learning algorithms. The metrics we used to evaluate performance were hit rate@10 again and RMSE.



Hit Ratio

In the absence of experimental data, we followed the Hit Ratio@10 (HR@10) standard with a rating cutoff as our primary evaluation metric.

Get ratings history by user in chronological order

Train recommendation system based on all but last rated movie

Rank the left-out movie together with 99 other unseen movies (randomly picked)

Assign a hit of 1 if left out, positively rated movie is within the top-10 or 0 otherwise

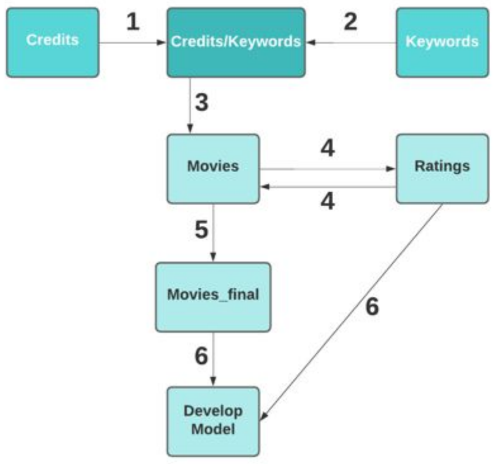
On an ideal setting, we would have ran A/B tests and we would have collected experimental data to measure the effectiveness of our recommender system in terms of increasing user engagement and satisfaction. But since all we had were historical preferences, we had to adapt and chose to follow a very used metric in the literature which is the hit ratio.

How does this work?

For each user, we take all the rated movies except by the last one to train our classifier. In the content-based approach, for instance, there is a different classifier for each different user. We then take the left out movie and complement it with other 99 randomly selected unseen movies. This became our test data. We then run the predictions for all of the 100 movies and generate a ranking based on the probability of the user to assign that movie a positive rating. If our left out movie is ranked within the top-10 recommendations, we assign that user a hit of 1. If not, we assign it a hit of 0. We follow this procedure for all the users and quantify at the end the percentage of users with a rate of 1. That`s the hit ratio for our classifier.

This metric is a good shortcut in a sense, but it has proved to be a very harsh metric as well, because as all of the users just watch and rank a very sparse set of movies - we may end up with low hit ratios even if our ranking of recommendations is good. We don`t know if users would have liked the movies unless they have actually watched or ranked them. That`s why in many real cases, user metrics engagement collected through experimental data are preferred, because they connect better with the business value of good recommender systems.

Model: Data Preprocessing



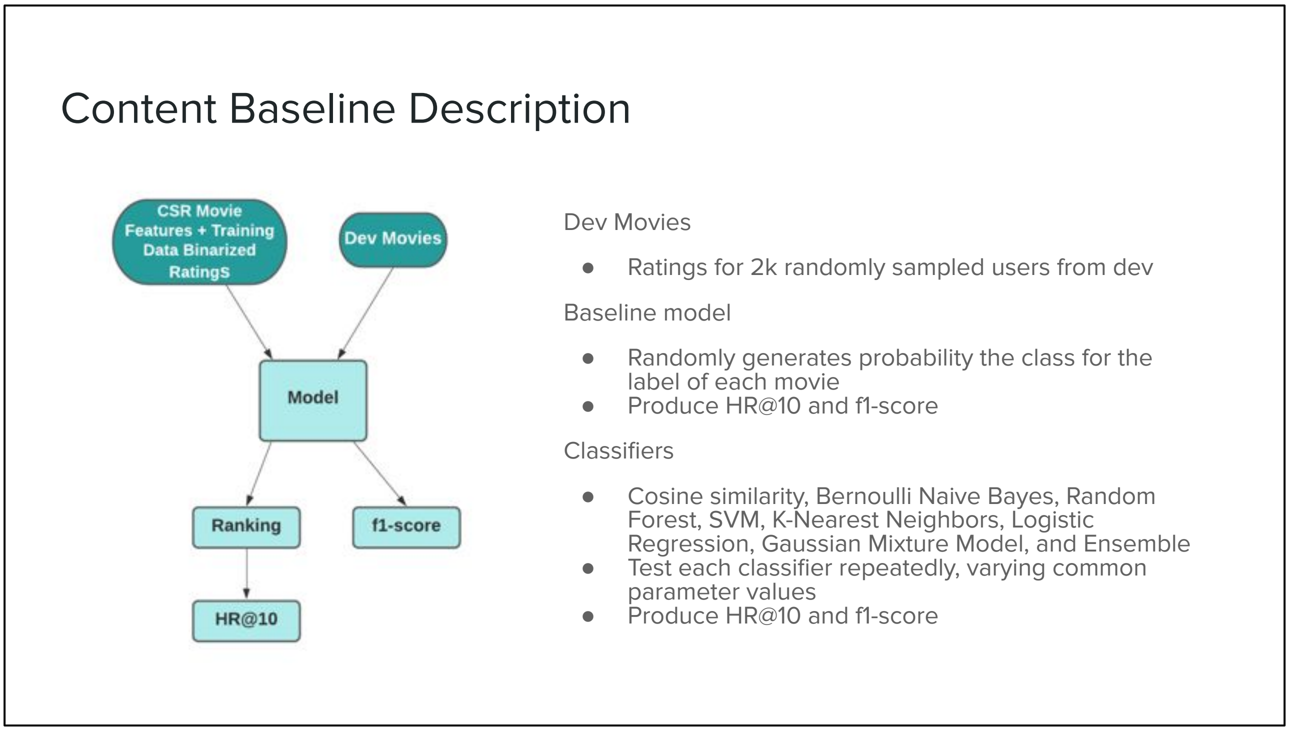
Collaborative Filtering

- Filtered ratings (users with 30+)
- Only used ratings

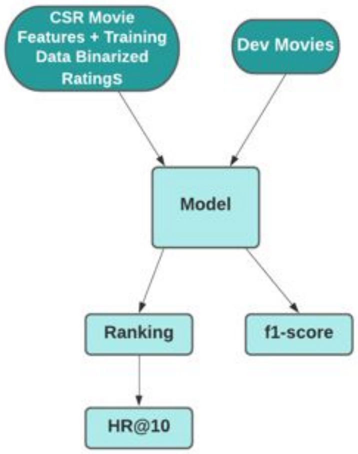
Content Recommender

- Data extraction, creation of new features, feature transformation, drop duplicates
- Dropped some features & converted to sparse matrix
- Filtered ratings and movies by IMDB ids present in the other dataset
- Filtered ratings (users with 30+) & binarized scores

Our collaborative filtering approach only used the ratings dataset, which required little preprocessing, so I will focus my discussion on the aspects relevant to the content recommender model. To obtain our final dataset of movie features,we first extracted cast and crew names from credits and descriptions from keywords, merged these datasets together, and merged this dataset with movies. We converted the form of some features using one-hot encoding, converted text features to numeric representation and normalized our features. Dimensionality reduction using PCA required a large number of features to explain variance and failed to improve model performance, as did our attempt to select the most important features to the weighted average rating using XGBoost. We did drop several fields that either provided no additional information (the movie id field) or that contained a large percentage of missing values (tagline, production countries). The result of this processing was a dataset with 769 features, which we compressed to a CSR. For ratings, we binarized ratings - scores below 4 to a 0, scores 4 or above a 1. We chose 4 as our threshold because that was the common threshold to indicate that a user liked a movie in the literature. Next, we filtered both the movies matrix and ratings to only include movies and ratings with an IMDB id in the other dataset. Finally, we filtered the ratings dataset to only retain ratings from users who'd provided 30+ ratings, choosing 30 as a our threshold since it was the common literature value. For our collaborative filtering model, we also filtered ratings to those from users with 30+ ratings, but no binarization of scores was performed.



Content Baseline Description



Dev Movies

- Ratings for 2k randomly sampled users from dev

Baseline model

- Randomly generates probability the class for the label of each movie
- Produce HR@10 and f1-score

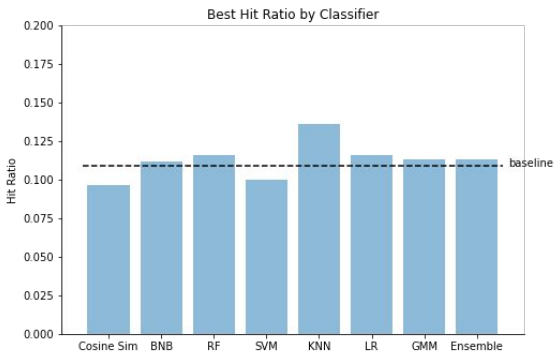
Classifiers

- Cosine similarity, Bernoulli Naive Bayes, Random Forest, SVM, K-Nearest Neighbors, Logistic Regression, Gaussian Mixture Model, and Ensemble
- Test each classifier repeatedly, varying common parameter values
- Produce HR@10 and f1-score

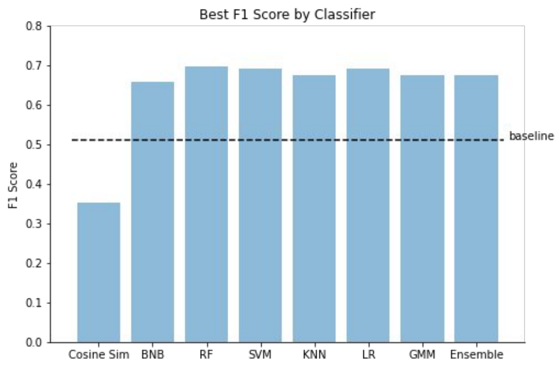
For our content recommender approach, we are building a model for each user and training that model on the user's training data ratings and the corresponding movie CSR features. To evaluate our content recommender approach across multiple users, we took a random sample of 2k unique users from the dev set since running our classifier tests on the entire dev set was too computationally expensive and built a model for the films for each user. As our baseline, we randomly generated the probabilities of the labels 0 and 1 for the test films for each user, ranked the movies in descending order of the probability of the correct label of the held out film/actual film rated, and returned the f1-score and the HR@10 across all users tested.

For our comparison to baseline, we tested this dev set using the classifiers listed on the slide. For these classifiers, instead of generating random probabilities for the labels for each test film, we fitted the model with the ratings of the training films and their corresponding movie features. We then performed the same ranking step using the model generated label probabilities, and returned the f1-score and HR@10 across all users tested.

Content Baseline Results

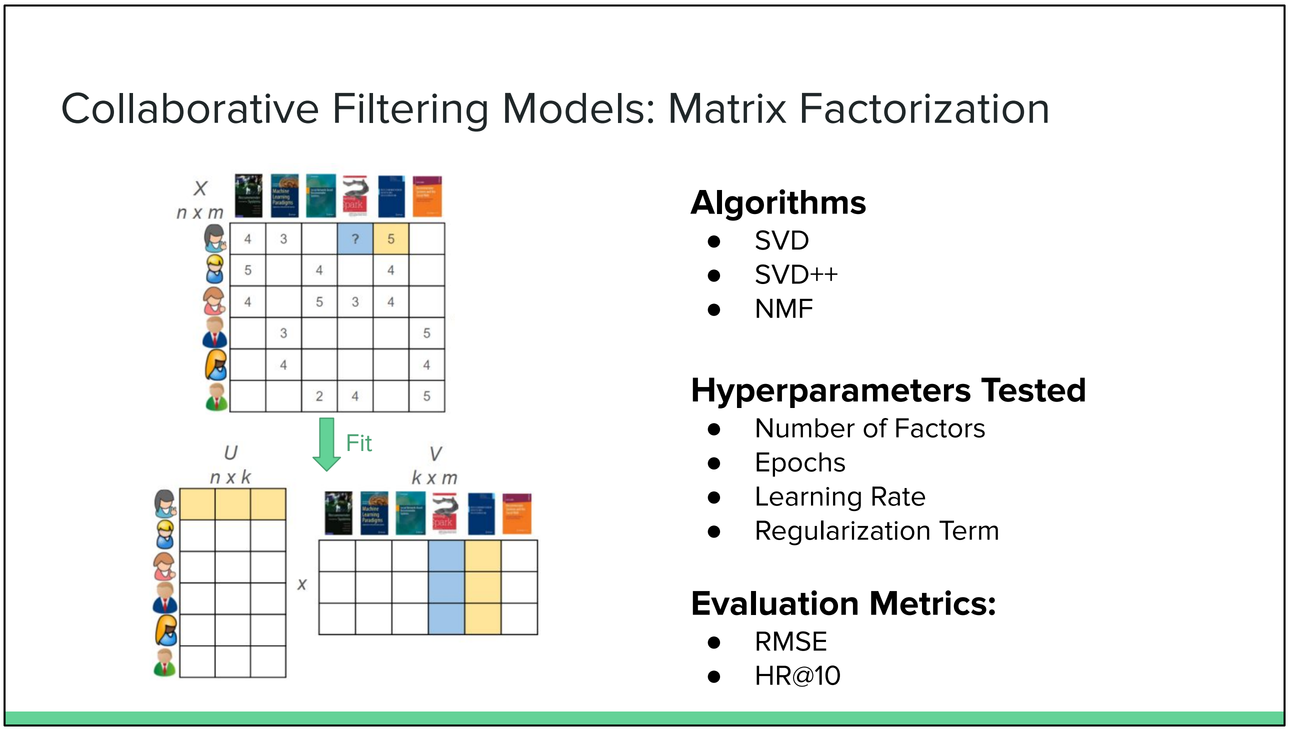


Best params (from hyperparameter tuning):	alpha	n_estim	C	k	C	n_comp
	1.0	100	1.0	25	1.0	5

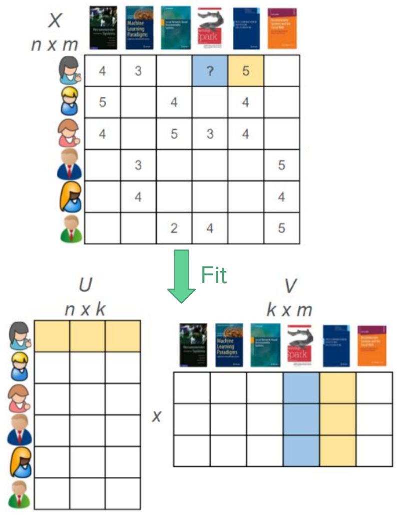


alpha	n_estim	C	k	C	n_comp
1.0	100	1.0	25	1.0	5

To determine the optimal classifier and parameters to use to against the test data, we ran the classifiers on this random sample of dev users and ratings multiple times, each time varying model parameters with common values. For BNB, we varied the alpha value, for RF the number of estimators, for SVM the cost parameter, for KNN the number of neighbors, for logistic regression the inverse of regularization strength, and for GMM the number of components. As you can see above, while the majority of models surpassed the f1-score of the baseline model, only KNN with a number of neighbors of 25 had an appreciable difference in HR@10. For that reason, we chose to use KNN with n_neighbors of 25 on the test data. I'll now pass it off to Rathin, who will be discussing development of the collaborative model.



Collaborative Filtering Models: Matrix Factorization



Algorithms

- SVD
- SVD++
- NMF

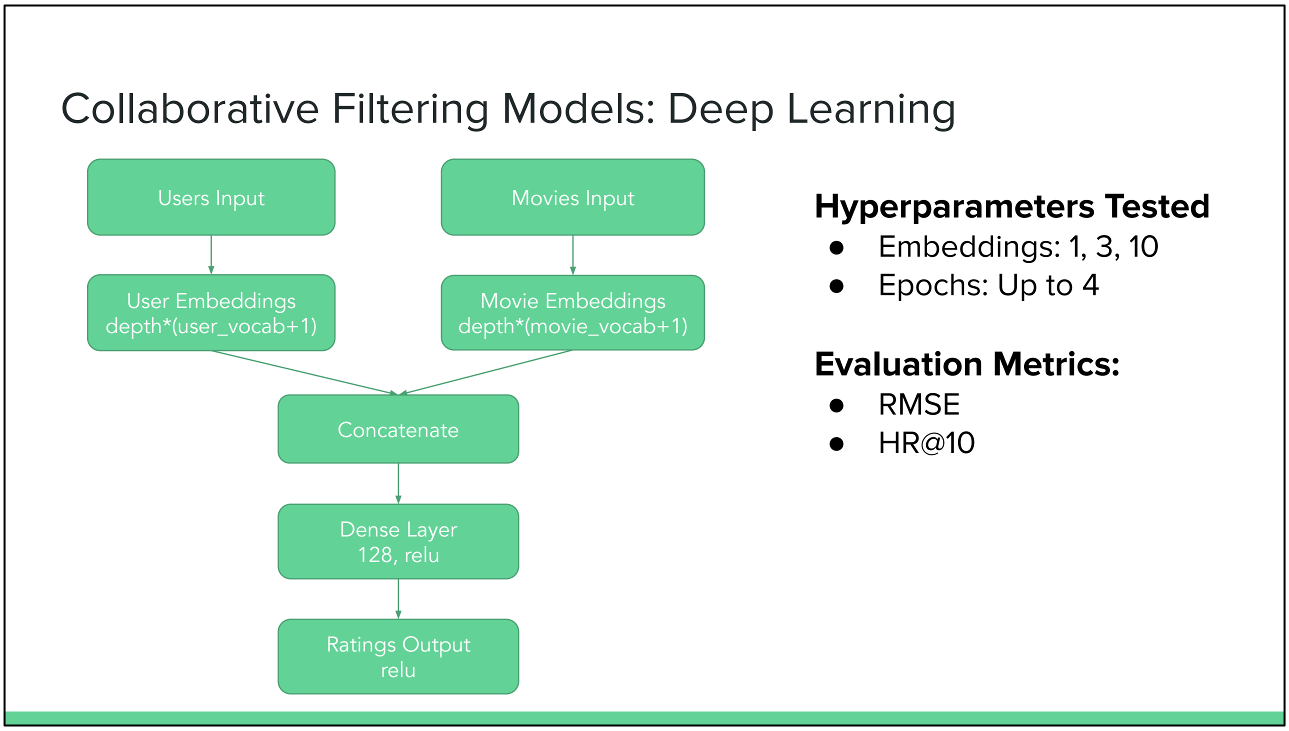
Hyperparameters Tested

- Number of Factors
- Epochs
- Learning Rate
- Regularization Term

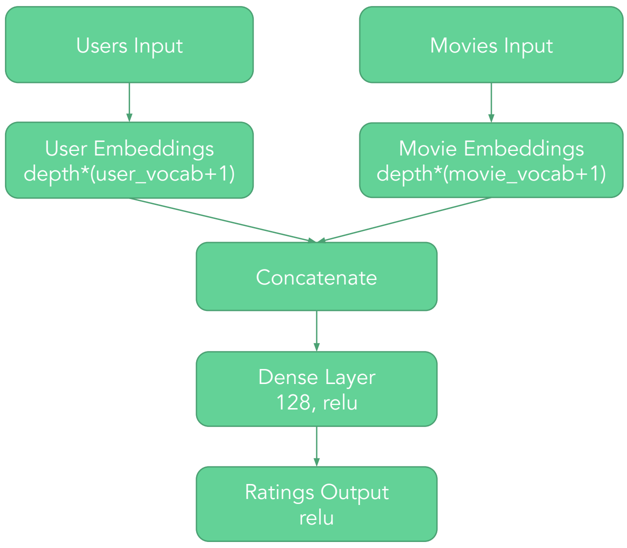
Evaluation Metrics:

- RMSE
- HR@10

Rathin



Collaborative Filtering Models: Deep Learning

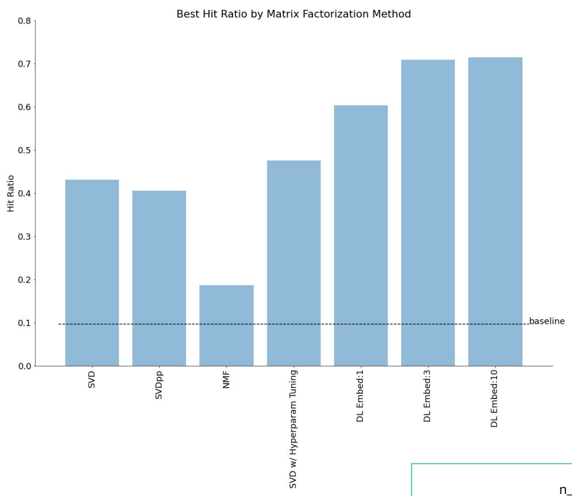

Users Input
Movies Input
User Embeddings depth*(user_vocab+1)
Movie Embeddings depth*(movie_vocab+1)
Concatenate
Dense Layer 128, relu
Ratings Output relu

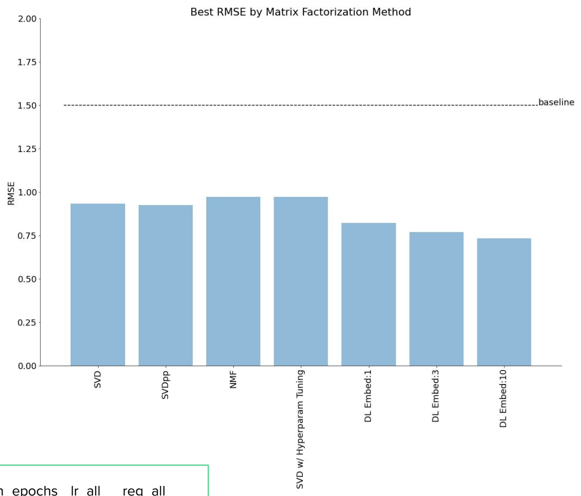
Hyperparameters Tested
- Embeddings: 1, 3, 10
- Epochs: Up to 4

Evaluation Metrics:
- RMSE
- HR@10

Rathin

Collaborative Filtering Metrics





	n_factors	n_epochs	lr_all	reg_all
Best parameters	50	30	0.001	0
(default)	*(100)*	*(20)*	*(0.005)*	*(0.02)*

Rathin

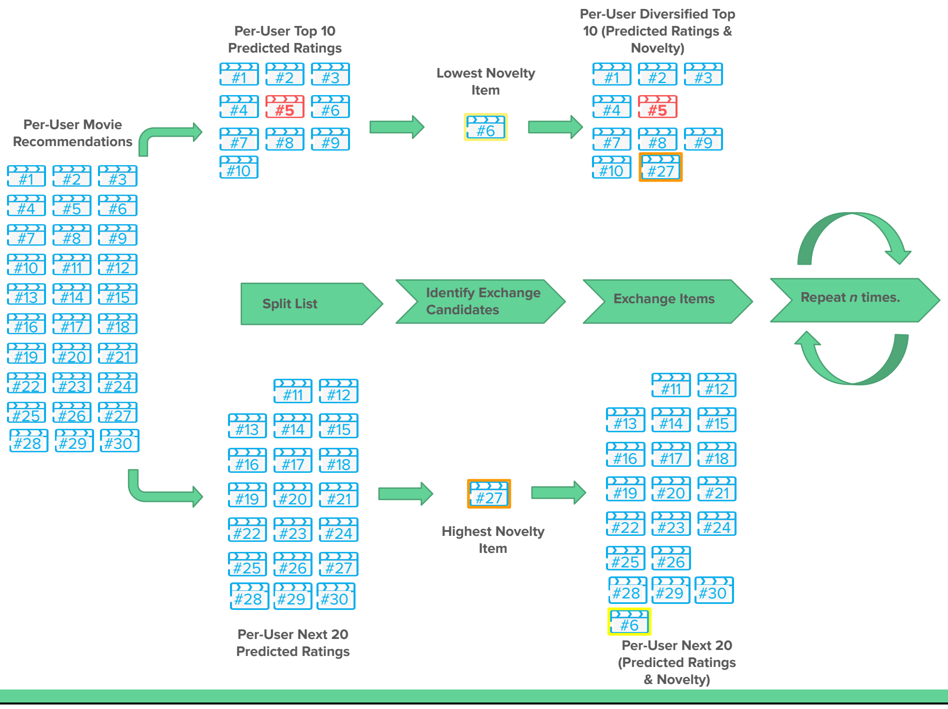
Model: Evaluation with Diversity Factor

Novelty:
Inverse Item Frequency in Training Data*

*CF approach scaled novelty to (0,5) scale to blend with ratings.

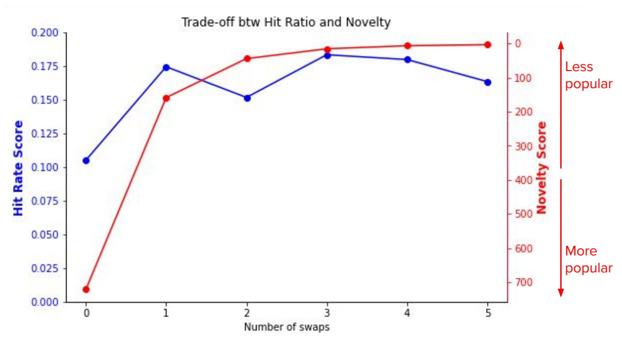
*Content approach used unadjusted movie count.

Both approaches yield the same results on novelty-only swapping.



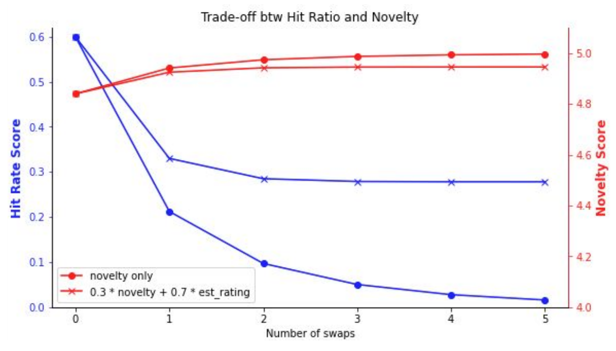
Novelty vs. Accuracy Trade-Off Varies

Content based filtering



Results for KNN model on test data with k=25

Collaborative filtering



Results for tuned SVD Model on test data.

-content based recommendation systems, despite having lower accuracy, may be more resistant to 'popular item bias' than CF models, since they are assessing user taste's as independent factors.
-while there are better diversity metrics out there (intra-list diversity scores or aggregrate diversity across users) they are too computationally intensive for our purposes. Swapping on novelty has a steep hit to HR@10, because our underlying dataset is so skewed towards popular movies, that even swap one movie often removes the test movie. This may be evidence of the feedback loop of recommender systems already at play. We do show that we can nearly max out novelty while maintaining a 30% hit rate with our blended swapping method.

-The cf novelty metric suffers from the long tail distribution we mentioned previously, since we normalize to 0,5 scale the long tail of movies get squeezed into the 5 novelty bucket. This means we quickly max novelty with a novelty-only swap. With a blended swap we are able to control the trade-off and move towards max novelty, while maintaining a HR@10 ~ 30%. We considered using log transforms and other adjustments to novelty score, but wanted to keep our interpretation of novelty pure. In this sense, the interpretation between 4.8 and 5.0 is that the top 10 movies we recommend are on average 4% less popular (more novel) across our entire test pool of over 135,000 users. Think about the exposure movies that are hidden gems can receive from even this small bump in diversification.And while the content recommender initially starts off with low novelty (-700), they increase average novelty by over 70% with just one swap, while maintaining their HR@10. These modifications help expose highly rated, low popularity movies over the long run and break the feedback loops around popular movies. If every user on Netflix had 5 new or underrated movies presented on their homepage, think about the exposure of these oft-forgotten movies would get over the long-run!

Conclusions

- Content recommender systems may have lower accuracy, but are more resistant to popularity biases.
- Properly baselining any machine learning model is critical for understanding performance.
- Deep Learning models are highly effective in RS tasks, but are not efficient.
- Novelty is a computationally efficient diversity metric, but may be less effective in some use-cases.
- Users can and should have the ability to express a preference for diversity in the recommendations they receive!

References

Data: https://www.kaggle.com/rounakbanik/the-movies-dataset?select=ratings.csv

Team Github Repo: https://github.com/bdougall/W207_movies

Sources:

1. https://dl.acm.org/doi/10.1145/3097983.3098173
2. https://journals.sagepub.com/doi/pdf/10.1177/13548565211014464
3. https://papers-gamma.link/static/memory/pdfs/153-Kunaver_Diversity_in_Recommender_Systems_2017.pdf
4. https://arxiv.org/pdf/1711.08379.pdf
5. http://ceur-ws.org/Vol-1181/pia2014_paper_02.pdf
6. https://www.netflixprize.com/assets/ProgressPrize2008_BellKor.pdf
7. https://www.researchgate.net/publication/275890626_State_of_the_Art_Recommender_System
8. https://grouplens.org/
9. https://arxiv.org/pdf/1904.12575v1.pdf

Educational Resources:

- Great intro to Matrix Factorization (Github)
- Recommender Systems -- Models and Evaluation (TDS)
- Getting Started with a Movie recommendation System (Kaggle)
- Deep Learning based Recommender Systems (TDS/Kaggle)
- Evaluation Metrics for Recommender Systems (TDS)

Appendix

A User Journey

Positively Rated Movies

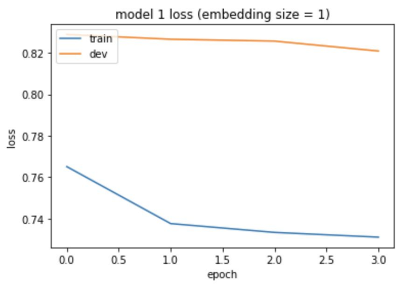
Baseline Recommendations

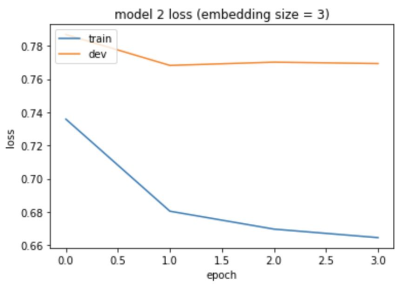
Diversified Recommendations

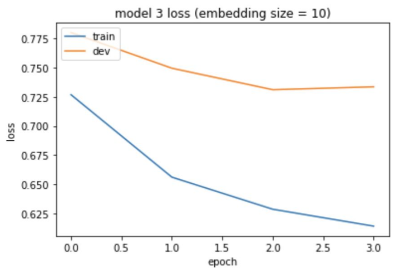
Appendix - Deep Learning Metrics

Embedding Depth	RMSE	HR@10	Training Time
1	0.821	60.3%	16.4m
3	0.769	70.9%	56.8m
10	0.734	71.5%	1hr 52m

Appendix - Deep Learning Models


model 1 loss (embedding size = 1)
train
dev
loss
epoch


model 2 loss (embedding size = 3)
train
dev
loss
epoch


model 3 loss (embedding size = 10)
train
dev
loss
epoch