

# Capstone Project Report

## Movie Similarity

David Pinto

2020-11-03

## Contents

<b>I. Definition</b>	<b>2</b>
Project Overview . . . . .	2
Problem Statement . . . . .	2
Metrics . . . . .	3
<b>II. Analysis</b>	<b>4</b>
Data Preparation . . . . .	4
Data Exploration . . . . .	4
Exploratory Visualization . . . . .	6
Algorithms and Techniques . . . . .	13
Benchmark . . . . .	14
<b>III. Methodology</b>	<b>17</b>
Code Organization . . . . .	17
Data Preprocessing . . . . .	17
Implementation . . . . .	19
Refinement . . . . .	23
<b>IV. Results</b>	<b>23</b>
Model Evaluation and Validation . . . . .	23
Justification . . . . .	28
<b>V. Conclusion</b>	<b>28</b>
Free-Form Visualization . . . . .	28
Reflection . . . . .	29
Improvement . . . . .	29
<b>References</b>	<b>30</b>

# I. Definition

## Project Overview

The main goal of this project is to develop a **Movie Similarity Recommender** over the full version of the famous [MovieLens](#) dataset. The developed solution aims to recommend similar items given a particular movie, so that users that already watched the later are likely to enjoy the recommended movies. It involves **Recommender Systems**<sup>1</sup> theory and **Similarity Matching**<sup>2</sup> techniques.

To properly achieve this we combine two datasets, one that provides movie ratings to be used as interaction data, and another that provides movie metadata to be used as content features.

## Movie Ratings

The full [MovieLens](#) dataset, which contains 27M ratings given to 58K movies by 280K users.

## Movie Metadata

Data retrieved from the [TMDB Open API](#) and available as a [Kaggle Dataset](#). It contains information on 45k movies featured in the full MovieLens dataset. Features include title, description, genre and cast&crew information.

## Problem Statement

Recommending similar items is a very common task in many online selling platforms<sup>3</sup>. It provides recommendations like: **related to items you've viewed, recommended for you based on your previous purchase, customers who bought this item already bought**, so on. The main challenges involved are:

### (a) The cold start problem

How to recommend recent added products with few or no associated interaction data?

One way to cope with this consists in take advantage of product metadata (content features): **name, description, category, price level**, so on. Then we can link a product with others by content similarity. We call this as **Content Based** approach.

### (b) The popularity bias problem

According to the **Long Tail phenomenon**<sup>4</sup>, selling niche products has many advantages. Users are likely to pay more for niche products because they cannot find it everywhere. However, best seller products have more interaction data available to train a recommender system so that the fitted model may be subjected to popularity bias. In other words, the model will be more likely to recommend popular items.

Again, using content features to find similar items reduces popularity bias. But interaction data is very important to understand user behavior and cannot be discarded. So, in this project we apply a **Collaborative Filtering** factorization that takes into account item and user popularity.

## Movie Embeddings and Similarity Matching

First of all, we use both interaction and content data to extract item embedding<sup>5</sup> features, which are low-dimensional dense vectors that captures some of the semantics of the item by placing semantically similar items close together in the embedding space. It allows efficient nearest neighbors search, so that we can find similar movies fast in application time.

- **Content Embedding:** embedding features extracted from text fields like title and description, and from categorical fields like genre and cast&crew.
- **Collaborative Filtering Embedding:** embedding features extracted from user-movie interactions given by the ratings.

All the details about the feature extraction process will be discussed later.

After computing the embedding features, similar movies will be recommended using **Approximate Nearest Neighbors** (ANN) search with cosine similarity. The cosine similarity is the most common affinity metric applied for similarity search in the embedding space.

## Metrics

### Users Intersection as a Similarity Measure

Let's consider every rating (from 1 to 5) above 3.5 as an explicit feedback that the user liked the movie. Then we can measure the similarity between a pair of movies  $(x, y)$  as:

$$S(x, y) = \frac{N_{xy}}{\sqrt{N_x N_y}}$$

where  $N_x$  is the number of users that liked movie  $x$ ,  $N_y$  is the number of users that liked movie  $y$ , and  $N_{xy}$  is the number of users that liked both  $x$  and  $y$ .

So,  $S(x, y)$  close to 1 is a strong evidence that users who liked  $x$  will like  $y$  too. We call this as **User Correlation**.

Despite providing good recommendations, this similarity is very sparse, resulting in a discrete recommendation. So, if we try to recommend the top 30 similar to each movie we will get less than 30 for many movies. In other words, we will have many movies with few or none similar items to be recommended.

The embedding features solve this problem. Since we are computing distances between pairs of movies in a dense and low-dimensional space, the embedding features provide a continuous recommendation. We are able to find similar items to any movie, with high evidence (low distance) or low evidence (high distance).

### Evaluating the Embedding Features

How close the similarity match with embedding features can get to the similarity match given by the user's likes (user correlation)?

#### (a) Ranking correlation

In an attempt to answer this question, we can randomly select  $n$  test movies and get their top  $N$  similar using the  $S(x, y)$  measure. Then, for each test movie, we get the distances  $D(x, y)$  from the top  $N$  similar using the embedding features. Finally, for each test movie, we apply the Spearman's Rank-Order Correlation Coefficient ( $\rho$ ) between  $S$  and  $1/D$ , which is equal to the Pearson's correlation between the rank values of them.

For all test movies, we get an average rank correlation:

$$\frac{1}{n} \sum_{i=1}^n \rho(S_i, 1/D_i)$$

#### (b) Precision and Recall at $K$

Given a threshold  $t_S$  for  $S(x, y)$ , we will assume that  $S(x, y) \geq t_S$  indicates a strong/relevant similarity. So, given the  $K$  nearest neighbors from embedding features of a test movie:

$$\text{precision@k} = \frac{\# \text{relevants@k}}{K}$$

$$\text{recall@k} = \frac{\# \text{relevants@k}}{\# \text{relevants}}$$

For all test movies, we get an average Precision@K and an average Recall@K.

## II. Analysis

### Data Preparation

First of all, take a look at `data/raw` to get instructions on how to download the datasets.

Data Preparation was a quite hard extra work in this project. As a previous step, we had to apply some manipulations in order to prepare data for analysis:

1. We set MovieLens ID as the primary key in both datasets, MovieLens and TMDB.
2. We transformed Cast&Crew, movie Genre and movie Keywords data from a JSON format to a [Tidy Data](#) format.
3. We concatenated all movie Keywords in a single text field.
4. We concatenated all movie Tags assigned by the users in a single text field.

Then, we applied some filters in an attempt to discard weak relations in the data:

1. In the Cast data we selected actors that appeared in at least 10 movies.
2. In the Crew data we considered just **director**, **producer** and **writer**, and selected those that appeared in at least 10 movies.
3. From the movie metadata we selected just the text fields: **title**, **overview** and **tagline**.
4. From the ratings data (interaction data) we removed users which voted less than 3 times and movies with less than 30 votes. Items and users with poor interaction information can induce Collaborative Filtering to learn wrong correlations in the data.

All code is well organized and documented in the `01-data-preparation.ipynb` notebook.

### Data Exploration

From the Data Preparation step we finished up with 7 datasets in tidy format:

- `data/movie_info.csv`:

id	title	overview	tagline
18	Four Rooms	It's Ted the Bellhop's first n (...)	Twelve outrageous guests. Four (...)
479	Judgment Night	While racing to a boxing match (...)	Don't move. Don't whisper. Don (...)
260	Star Wars	Princess Leia is captured and (...)	A long time ago in a galaxy fa (...)
6377	Finding Nemo	Nemo, an adventurous young clo (...)	There are 3.7 trillion fish in (...)
356	Forrest Gump	A man with a low IQ has accomp (...)	The world will never be the sa (...)

- data/movie\_keywords.csv:

keywords	id
underdog prison factory worker (...)	4470
salesclerk helsinki garbage in (...)	61724
hotel new year's eve witch bet (...)	18
chicago drug dealer boxing mat (...)	479
android galaxy hermit death st (...)	260

- data/movie\_tags.csv:

id	tags
110	epic Medieval overrated Oscar (...)
260	sci-fi space action classic sc (...)
318	imdb top 250 justice story lin (...)
480	Dinosaurs dinosaurs action thr (...)
593	psychothriller brilliant explo (...)

- data/movie\_genre.csv:

genre_id	genre_name	id
80	Crime	18
35	Comedy	18
28	Action	479
53	Thriller	479
80	Crime	479

- data/movie\_actor.csv:

actor_id	actor_name	id	num_movies
4826	Matti Pellonpää	4470	14
4826	Matti Pellonpää	61724	14
5999	Kati Outinen	61724	18
4828	Sakari Kuosmanen	61724	14
3129	Tim Roth	18	60

- data/movie\_producer:

person_id	job	person_name	id	num_movies
16767	Director	Aki Kaurismäki	4470	63
16767	Director	Aki Kaurismäki	61724	63
54767	Producer	Mika Kaurismäki	61724	38
3110	Director	Allison Anders	18	16
3111	Director	Alexandre Rockwell	18	11

- data/movie\_ratings:

user	id	rating
1	307	3.5
1	481	3.5
1	1091	1.5
1	1257	4.5
1	1449	4.5

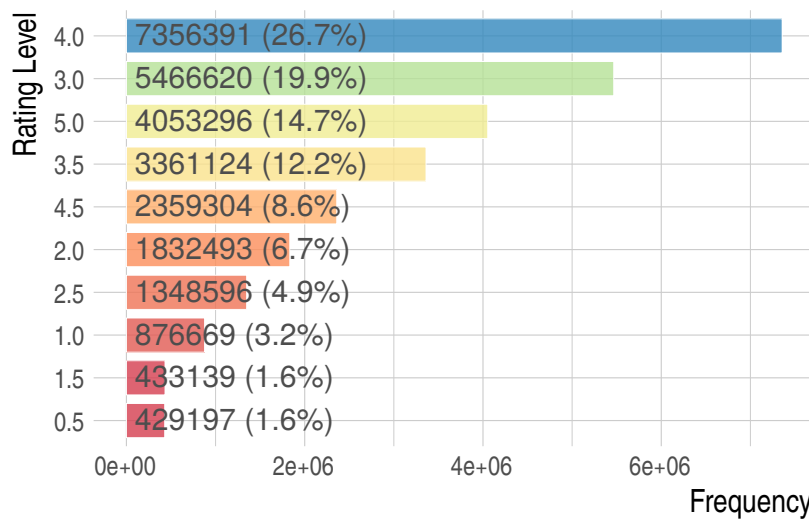
After all cleaning process we got text description for 20400 movies, keywords for 31116 movies, tags for 45981 movies, genres for 20136 movies, 10465 actors for a total of 35649 movies, 4106 director/producer/writer for a total of 24325 movies, and 27516829 ratings for a total of 15967 movies given by 273643 distinct users.

## Exploratory Visualization

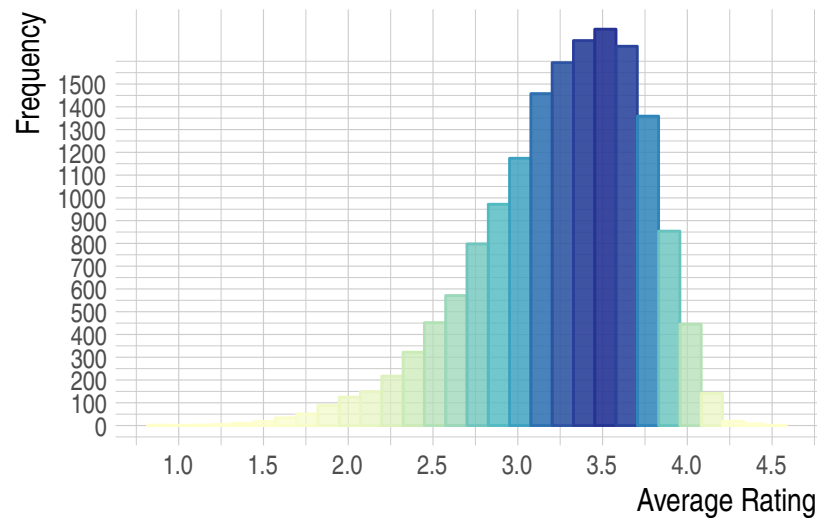
Now we invite you to take a brief tour over the data. For more details, the `02-exploratory-analysis.ipynb` notebook contains all code for the Exploratory Data Analysis we present here.

### Ratings Distribution

First of all, let's observe the distribution of ratings. We have 10 possible values for the user rating: [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]. In which proportion each one of these values appear in the data?



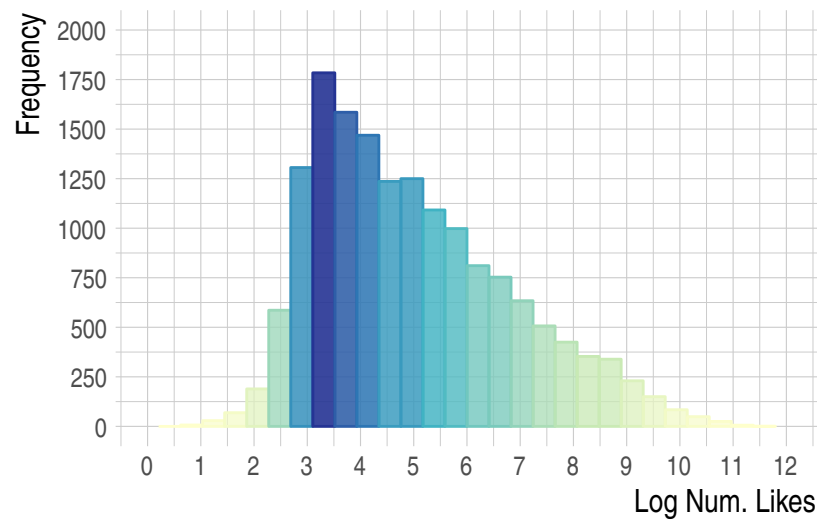
Just a small part of the ratings fall below 3.0. If we look at the movie average rating we will observe a similar behavior:



In general, users tend to give good ratings to the movies they vote. It may be an evidence that users are most likely to vote in those movies they like. It makes sense, since users usually watch movies with actors and/or directors they already know and like, and in the genre they like most.

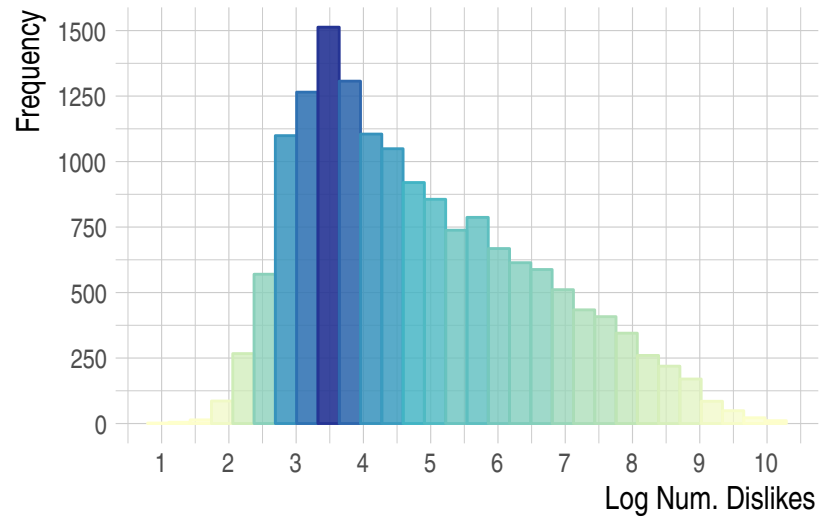
### Ratings as Likes and Dislikes

In this project we will convert ratings to likes ( $\geq 3.5$ ) and dislikes ( $< 3.5$ ). Let's take a look the the distribution of number of likes:

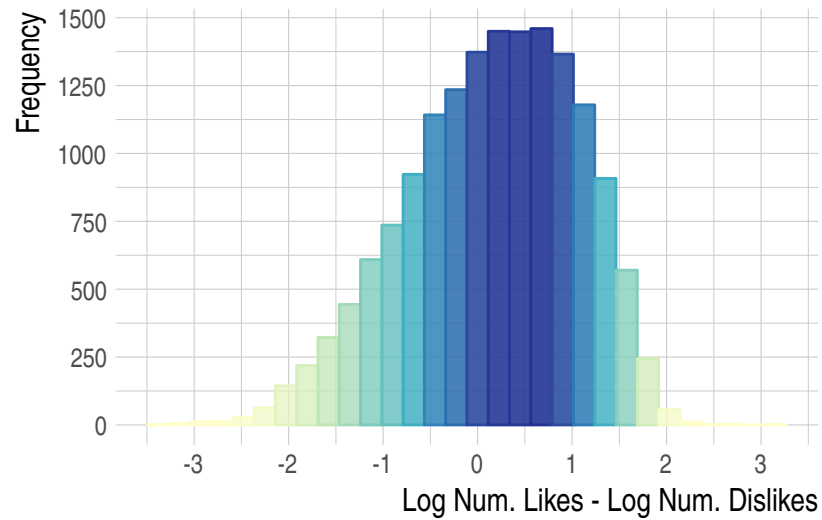


We got a very asymmetric empirical distribution, so we applied the  $\log$  transformation to make it easy to visualize the full distribution. Just to have an idea, 10 in the log scale corresponds to about 22025 likes.

Now, let's see how the number of dislikes behave:



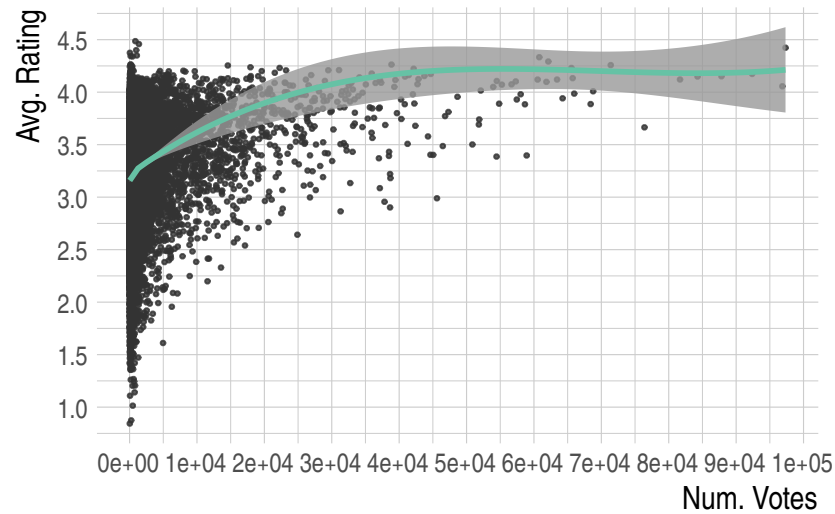
Is there any movie with more dislikes than likes?



Since we have negative values, the answer to the last question is yes.

Is there a consistent bias in the average rating to popular movies?





Yes, we can see a popularity bias. Movies with more than 20k votes have average rating above 3.0. Movies with more than 50k votes tend to have average rating above 4.0.

### Most Liked and Most Hated Movies

Which movies are the most liked by the users?

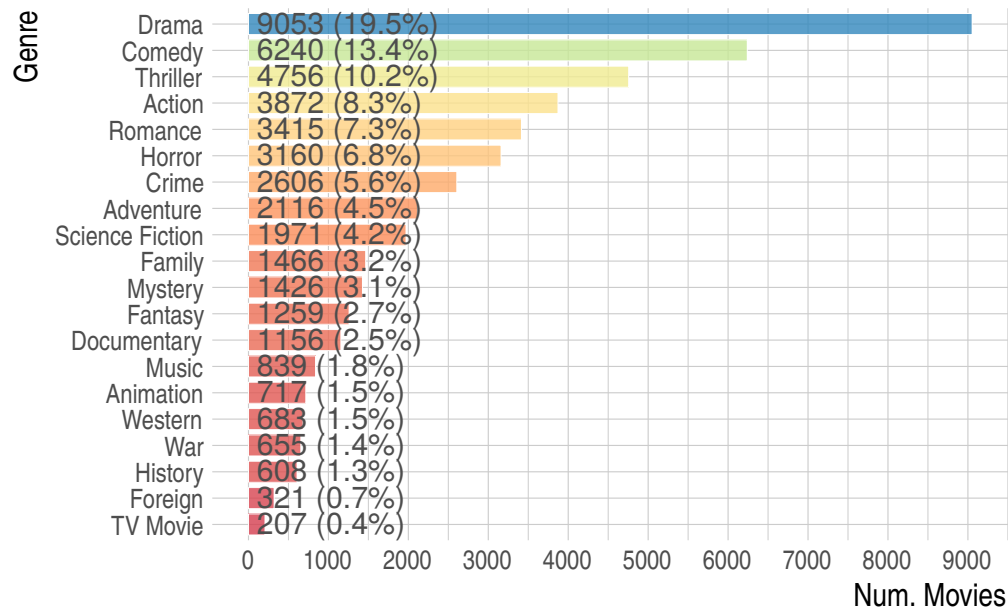
title	num_likes
The Shawshank Redemption	88846
Forrest Gump	77143
Pulp Fiction	76334
The Silence of the Lambs	74607
The Matrix	71266
Star Wars	66945
Schindler's List	62012
Fight Club	57479
The Usual Suspects	55297
The Empire Strikes Back	54653

How about the most disliked movies by the users?

title	num_dislikes
Ace Ventura: Pet Detective	29083
Batman	28837
Batman Forever	28704
Independence Day	27731
Jurassic Park	27542
Dumb and Dumber	23697
True Lies	23032
Twister	22670
The Mask	22388
Waterworld	22342

## Genre Distribution

Now, let's check the distribution of movie genres:



The Drama genre is present in about 20% of the movies, it's very frequent. On the other hand, now I understand why it's so hard to find new Western movies to watch. Just 1.5% of the movies are Western.

## Most Liked Western and Horror Movies

Let's take a look at my personal favorite genres, Western and Horror. Which movies are the most liked by the audience?

### Western Movies:

title	num_likes
Dances with Wolves	33822
Django Unchained	17188
The Good, the Bad and the Ugly	16676
Butch Cassidy and the Sundance Kid	14481
Unforgiven	13387
Tombstone	10269
Blazing Saddles	9751
Maverick	9694
A Fistful of Dollars	6666
True Grit	6113

### Horror Movies:

title	num_likes
Alien	32325
Aliens	27645
The Shining	26290
Psycho	19887
Jaws	18761
Interview with the Vampire	17297
Shaun of the Dead	16134
28 Days Later	13346
The Exorcist	11961
Zombieland	10524

I totally agree with the ranking!

### Most Liked and Most Hated Directors

Can we find the public opinion about the movie directors based on the number of likes and dislikes given to the movies they worked on? To answer this question, we subtracted the number of likes by the number of dislikes considering all movies each director worked. Here it follows the ranking of the most liked and most disliked directors:

Director	Delta Likes	Most Liked Movie
Steven Spielberg	269758	Schindler's List
Christopher Nolan	168547	The Dark Knight
Quentin Tarantino	164889	Pulp Fiction
Peter Jackson	133523	The Lord of the Rings: The Fellowship of the Ring
David Fincher	123548	Fight Club
Robert Zemeckis	121917	Forrest Gump
Martin Scorsese	116307	GoodFellas
Joel Coen	111031	Fargo
Ridley Scott	110320	Gladiator
Stanley Kubrick	108636	The Shining
James Cameron	104873	Terminator 2: Judgment Day
Frank Darabont	103952	The Shawshank Redemption
Francis Ford Coppola	101317	The Godfather
Alfred Hitchcock	100372	Rear Window
Terry Gilliam	82620	Twelve Monkeys

The **Schindler's List** list is in the top 10 most liked movies and helped **Steven Spielberg** to be the first in the ranking.

Director	Delta Likes	Most Hated Movie
Tom Shadyac	-23201	Ace Ventura: Pet Detective
Joel Schumacher	-17924	Batman Forever
Frank Marshall	-16645	Congo
Renny Harlin	-16136	Cliffhanger
Brian Levant	-14454	The Flintstones
Chuck Russell	-14196	The Mask
Steve Barron	-12396	Coneheads
Paul W.S. Anderson	-12178	Mortal Kombat
Simon Wincer	-11879	Free Willy
Steve Oedekerk	-11715	Ace Ventura: When Nature Calls
Wes Craven	-11201	Scream
Joe Johnston	-11114	Jumanji
Jan de Bont	-10355	Twister
Peter Hyams	-10345	Timecop
McG	-10081	Charlie's Angels

On the other hand, **Ace Ventura**, the first in the list of the most disliked movies, put two directors in the list of the 10 most disliked ones.

### Most Liked and Most Hated Actors

How about the actors?

Actor	Delta Likes	Most Liked Movie
Philip Ettington	393308	The Shawshank Redemption
Harrison Ford	313376	Star Wars
Samuel L. Jackson	246385	Pulp Fiction
Brad Pitt	243918	Fight Club
Tom Hanks	234244	Forrest Gump
Morgan Freeman	228909	The Shawshank Redemption
Matt Damon	225037	Saving Private Ryan
Hugo Weaving	217845	The Matrix
James Earl Jones	207276	Star Wars
Bruce Willis	191274	Pulp Fiction
John Ratzenberger	187389	The Empire Strikes Back
John Rhys-Davies	185541	Raiders of the Lost Ark
Kevin Spacey	181275	The Usual Suspects
Robert De Niro	174799	The Godfather: Part II
Steve Buscemi	171934	Pulp Fiction

The **The Shawshank Redemption** put two actors in the list of the 10 most liked actors!

Actor	Delta Likes	Most Hated Movie
Rob Schneider	-36982	Judge Dredd
Pat Hingle	-29543	Batman
Sylvester Stallone	-25823	Cliffhanger
Jim Carrey	-24639	Ace Ventura: Pet Detective
Courteney Cox	-24107	Ace Ventura: Pet Detective
Bridgette Wilson	-23983	Last Action Hero
Jean-Claude Van Damme	-23523	Last Action Hero
Demi Moore	-22888	Ghost
Chris O'Donnell	-22578	Batman Forever
Stuart Pankin	-21566	Congo
Adam Sandler	-21471	Coneheads
Phil Hartman	-21429	Coneheads
Udo Kier	-21099	Ace Ventura: Pet Detective
Michael Gough	-20600	Batman
Troy Evans	-19226	Ace Ventura: Pet Detective

The `Ace Ventura` definitely didn't fall in popular taste.

## Algorithms and Techniques

### Building Content Embeddings

For text features (title, description, keywords and user comments/tags) we concatenate them all and extract a sparse TF-IDF matrix using the `TfidfVectorizer`.

For the categorical features (genre, cast and crew) we expand them into sparse binary features.

Both representations result in high-dimensional and sparse matrices. So, we combine them and apply the sparse `TruncatedSVD` algorithm to get low-dimensional and dense representations, the embedding features.

The `implicit` ([link](#)) package provides some fast factorization algorithms more powerful than SVD. They are great alternatives and may improve the representations.

### Building CF Embeddings

The `surprise` ([link](#)) package provides many classical factorization algorithms for explicit feedback problems. We can apply them to the user-movie ratings matrix to extract latent factors for the users and movies. Then, the movie factors can be used as embedding features.

However, we decided to try an approach based on likes and dislikes. Ratings above 3.5 will be considered likes (+1), while ratings below 3.5 will be considered dislikes (-1). From this assumption, we can build a user-movie matrix with (-1,+1) values and use a logistic loss to factorize this matrix. The `LighFM` ([link](#)) package provides a logistic loss factorization algorithm with additive bias. So, it takes into account the user and the movie bias, abstracting popularity away from the embedding features. This is an attempt to solve the item popularity problem.

### Similarity Match

After computing the embedding features we recommend similar movies using an approximate nearest neighbors search with cosine similarity. The `faiss` package is our choice, since it provides an extremely fast and scalable ANN algorithm.

## Hybrid Approach

First, we evaluate the recommendations given by the content embedding features and compare the result with the performance for the CF embedding features. Then, as a final attempt, we test a hybrid approach which works as follows:

1. Select a target movie for which we will recommend similar ones.
2. Compute the cosine similarity between the target and the candidates using the content embedding features.
3. Compute the cosine similarity between the target and the candidates using the CF embedding features.
4. Order the candidates by the sum between the computed similarities in the last 2 steps.

It's the same as combining the embedding features (content based + collaborative filtering) and taking the cosine similarities over them all. Since this approach combines content information with interaction information, it may provide better recommendations.

## Benchmark

To evaluate the embedding features in the similarity search task we established the user correlation as our benchmark. Basically, the similarity between a pair of movies (1, 2) is proportional to the number of users that liked both, as follows:

$$S_{1,2} = \frac{n_{12}}{\sqrt{n_1 n_2}}$$

where:

- $n_1$  is the number of users that liked movie 1.
- $n_2$  is the number of users that liked movie 2.
- $n_{12}$  is the number of users that liked both movie 1 and movie 2.

To find the number of users that liked each pair of movies we need to compute a cartesian join with the ratings dataset. It's a very expensive operation over a dataset with 27M rows. The `pandas` library is not able to perform this task. It requires lots of RAM and may take several minutes to complete. So, we decided to try the `vaex` library. It allows to split data into chunks and perform operations chunk by chunk. It is very efficient and its API is very similar to `pandas`.

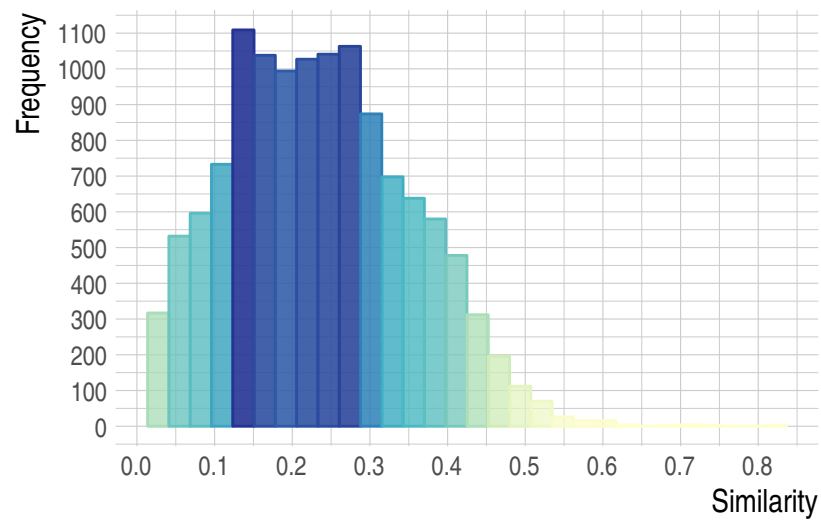
To reduce time and memory consumption even more we randomly selected a subset of 500 movies. It's enough to evaluate the embedding features. In order to keep the most famous movies, we assigned the number of likes as the probability to select the movie.

The operation over the `ratings` dataset works as follows:

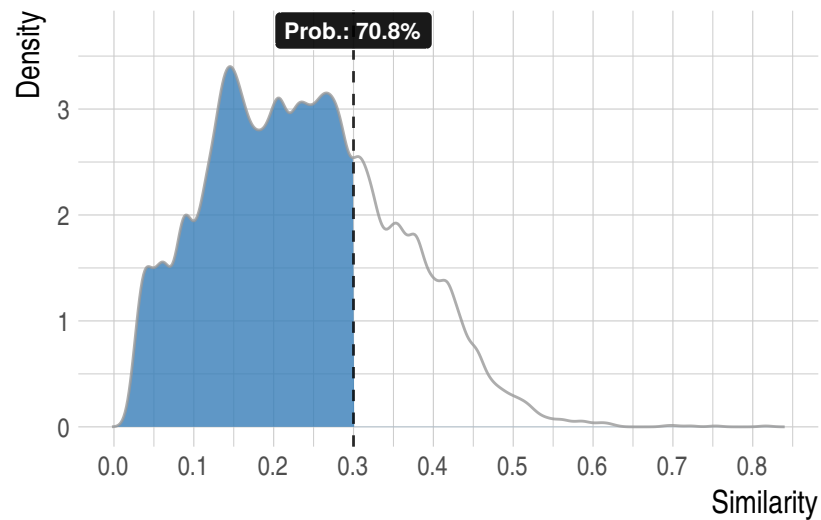
1. Filter the ratings for the 500 randomly selected movies.
2. Filter rating values greater than or equal to 3.5 (likes).
3. Create 2 copies of the resulting dataset with the following column names: `movie_x`, `user` and `movie_y`, `user`.
4. Join the 2 dataframes by `user`. It results in a dataframe with 238,593,639 rows. Pandas took several minutes and resulted in memory overflow. Vaex easily solved the problem in just 10 seconds!
5. Group by (`movie_x`, `movie_y`) and count number of rows in each group. It gives us the  $n_{1,2}$  term.
6. Summarize the dataframe on step 2 to get the number of likes for each movie.
7. Join the number of likes to the dataframe on step 5.
8. Compute the user correlation for each pair (x,y) of movies.

All code is well organized and documented in the `03-user-based-similarity.ipynb` notebook.

Let's check the distribution of the similarity values:



Considering 0.3 as an evidence of correlation/similarity between a pair, let's check the empirical cumulative probability:



Which are the top correlated movies based on user likes?

title_left	title_right	similarity
The Lord of the Rings: The Ret (...)	The Lord of the Rings: The Fel (...)	0.816
The Lord of the Rings: The Fel (...)	The Lord of the Rings: The Ret (...)	0.816
The Empire Strikes Back (...)	Star Wars (...)	0.753
Star Wars (...)	The Empire Strikes Back (...)	0.753
Return of the Jedi (...)	The Empire Strikes Back (...)	0.722
The Empire Strikes Back (...)	Return of the Jedi (...)	0.722
Return of the Jedi (...)	Star Wars (...)	0.702
Star Wars (...)	Return of the Jedi (...)	0.702
The Bourne Supremacy (...)	The Bourne Identity (...)	0.693
The Bourne Identity (...)	The Bourne Supremacy (...)	0.693
Finding Nemo (...)	Monsters, Inc. (...)	0.625
Monsters, Inc. (...)	Finding Nemo (...)	0.625
Back to the Future Part III (...)	Back to the Future Part II (...)	0.619
Back to the Future Part II (...)	Back to the Future Part III (...)	0.619
The Terminator (...)	Terminator 2: Judgment Day (...)	0.616
Terminator 2: Judgment Day (...)	The Terminator (...)	0.616
The Silence of the Lambs (...)	Pulp Fiction (...)	0.613
Pulp Fiction (...)	The Silence of the Lambs (...)	0.613
Fight Club (...)	The Matrix (...)	0.608
The Matrix (...)	Fight Club (...)	0.608
The Lord of the Rings: The Fel (...)	The Matrix (...)	0.606
The Matrix (...)	The Lord of the Rings: The Fel (...)	0.606
The Terminator (...)	Alien (...)	0.592
Alien (...)	The Terminator (...)	0.592
The Lord of the Rings: The Ret (...)	The Matrix (...)	0.591
The Matrix (...)	The Lord of the Rings: The Ret (...)	0.591
The Shawshank Redemption (...)	Pulp Fiction (...)	0.590
Pulp Fiction (...)	The Shawshank Redemption (...)	0.590
Pulp Fiction (...)	The Usual Suspects (...)	0.586
The Usual Suspects (...)	Pulp Fiction (...)	0.586

Which are the most similar movies to *The Lord of the Rings*?

title_left	title_right	similarity
The Lord of the Rings: The Fel (...)	The Lord of the Rings: The Ret (...)	0.816
The Lord of the Rings: The Fel (...)	The Matrix (...)	0.606
The Lord of the Rings: The Fel (...)	Fight Club (...)	0.541
The Lord of the Rings: The Fel (...)	Shrek (...)	0.530
The Lord of the Rings: The Fel (...)	The Empire Strikes Back (...)	0.526
The Lord of the Rings: The Fel (...)	Star Wars (...)	0.503
The Lord of the Rings: The Fel (...)	The Dark Knight (...)	0.502
The Lord of the Rings: The Fel (...)	Finding Nemo (...)	0.492
The Lord of the Rings: The Fel (...)	Monsters, Inc. (...)	0.485
The Lord of the Rings: The Fel (...)	The Bourne Identity (...)	0.472

Which are the most similar movies to *Star Wars*?



title_left	title_right	similarity
Star Wars (...)	The Empire Strikes Back (...)	0.753
Star Wars (...)	Return of the Jedi (...)	0.702
Star Wars (...)	The Matrix (...)	0.558
Star Wars (...)	Indiana Jones and the Last Cru (...)	0.530
Star Wars (...)	Back to the Future (...)	0.508
Star Wars (...)	The Lord of the Rings: The Fel (...)	0.503
Star Wars (...)	The Terminator (...)	0.496
Star Wars (...)	Alien (...)	0.487
Star Wars (...)	The Lord of the Rings: The Ret (...)	0.483
Star Wars (...)	Monty Python and the Holy Grai (...)	0.472

Which are the most similar movies to *Pulp Fiction*?

title_left	title_right	similarity
Pulp Fiction (...)	The Silence of the Lambs (...)	0.613
Pulp Fiction (...)	The Shawshank Redemption (...)	0.590
Pulp Fiction (...)	The Usual Suspects (...)	0.586
Pulp Fiction (...)	Fight Club (...)	0.558
Pulp Fiction (...)	Reservoir Dogs (...)	0.558
Pulp Fiction (...)	Fargo (...)	0.519
Pulp Fiction (...)	American Beauty (...)	0.517
Pulp Fiction (...)	The Matrix (...)	0.509
Pulp Fiction (...)	Schindler's List (...)	0.494
Pulp Fiction (...)	Terminator 2: Judgment Day (...)	0.482

## III. Methodology

### Code Organization

We organized the solution in Jupyter Notebooks:

- [1. Data Preparation.](#)
- [2. Exploratory Analysis.](#)
- [3. User Based Similarity.](#)
- [4. Content Based Embedding.](#)
- [5. Collaborative Filtering Embedding.](#)
- [6. Similarity Match with ANN.](#)
- [7. Performance Evaluation.](#)
- [8. Hybrid Approach.](#)

They were carefully refactored and documented.

Moreover, we used a Python package structure to organize the functions applied in the notebooks. All code can be find in the `src` folder.

### Data Preprocessing

In the *Data Preparation* step we transformed data to a tidy format. Now we preprocess features to put it in a numerical format for embedding extraction.

## Text Features

First of all, we combined all text fields (`title`, `overview`, `tagline`, `tags` and `keywords`) into a single one. Then we applied some text cleaning:

- Lower case.
- Remove numbers, punctuation and special chars.
- Remove single char words.
- Remove stopwords.

To encode text into numerical features we used the `TfidfVectorizer` method with the following parameters:

- Uni-grams and bi-grams.
- Vocabulary of 30k terms.
- Minimum document frequency of 5.
- Maximum document frequency of 50%.
- Scale rows by their l2-norm.

We got a Tf-Idf matrix with dimension 20624 x 30000 and 99.79% of sparsity.

Since the cosine similarity between two vectors is their dot product when their l2-norm is equal to 1, we always apply this normalization. The `faiss` package requires normalized vectors to find nearest neighbors.

All code is available in the `04-content-based-embedding.ipynb` notebook.

## Categorical Features

`Genre` and `Cast&Crew` are categorical features. To improve content representation, we need to combine them with the Tf-Idf features. We may try a classical one-hot-encoding process, but these features have high cardinality. Representing them by a dense binary matrix may consume lots of RAM and make the embedding extraction very slow. So we represent them using sparse matrices and apply l2-normlization on rows.

For `Crew` we got a 20624 x 3635 sparse matrix with 99.81% of sparsity. It means that we are consuming just 0.19% of the memory required by a dense matrix. For `Cast` we got a 20624 x 10269 sparse matrix with 99.94% of sparsity. Finally, after combining the binary features with the Tf-Idf ones we got a 20624 x 43904 sparse matrix with 99.84% of sparsity.

So, we have a high dimensional and high sparse content matrix to work with.

We decided to not include genre in the content matrix. We already have many information on text features about the movie, more specific than genre. Genre gives just a few binary features and may bring redundant information.

All code is available in the `04-content-based-embedding.ipynb` notebook.

## Ratings

For the ratings dataset we need to build an *Utility Matrix*. In other words, we need to put data into an user/movie matrix format. Users will be represented as rows, movies as columns and like(+1)/dislike(-1) as values. Since we have a large number of movies and the greatest part of the user/movie interactions are missing (a user votes in a small part of the movies, and a movie is evaluated by a small part of the users), a sparse representation is the better choice.

We transformed ratings into likes (rating  $\geq 3.5$  to 1.0) and dislikes ( rating  $< 3.5$  to -1.0) and got a 273627 x 10698 sparse matrix with 99.13% of sparsity.

All code is available in the `05-collaborative-filtering-embedding.ipynb` notebook.

## Implementation

Now we describe how we extracted embedding features from the sparse matrices generated in the previous section.

### Content Based

We extract dense features from the sparse content matrix using the [TruncatedSVD](#) algorithm. The parameter `algorithm="arnpack"` is necessary to deal with a sparse matrix. It took less than 20 seconds to factorize the content matrix.

We use 300 as the embedding dimension. It's a very common value in Word Embedding representations.

Let's take advantage of the [Embedding Projector](#) tool to visualize the embedding features:

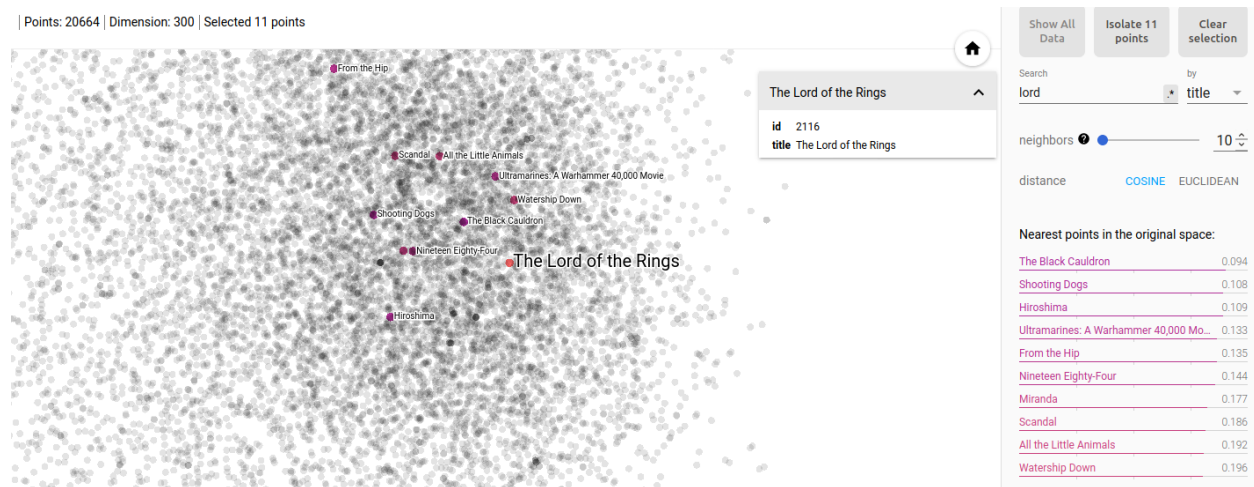


Figure 1: The Lord of the Rings.

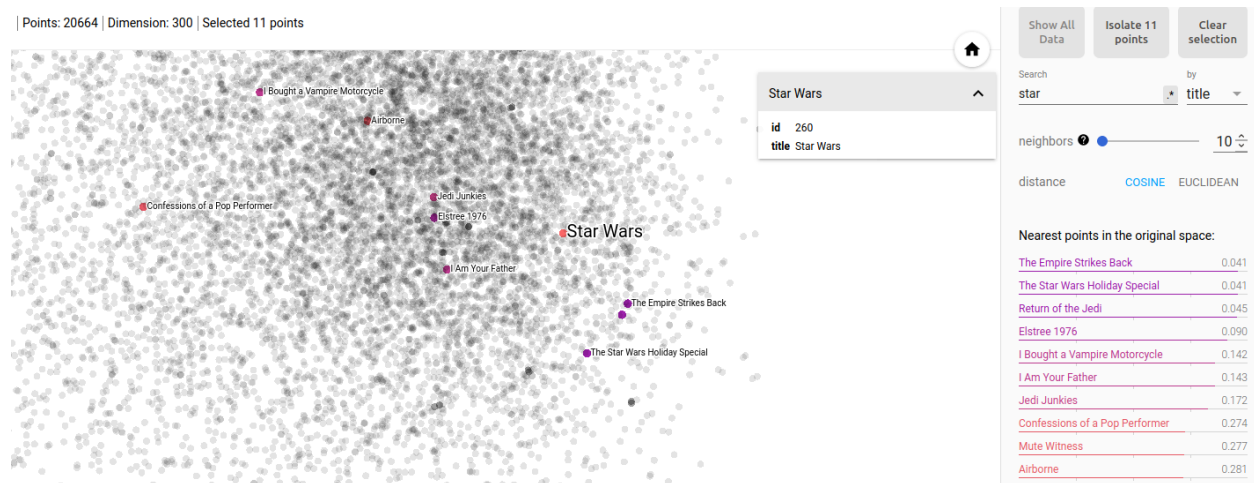


Figure 2: Star Wars.

The nearest neighbors shown make sense.

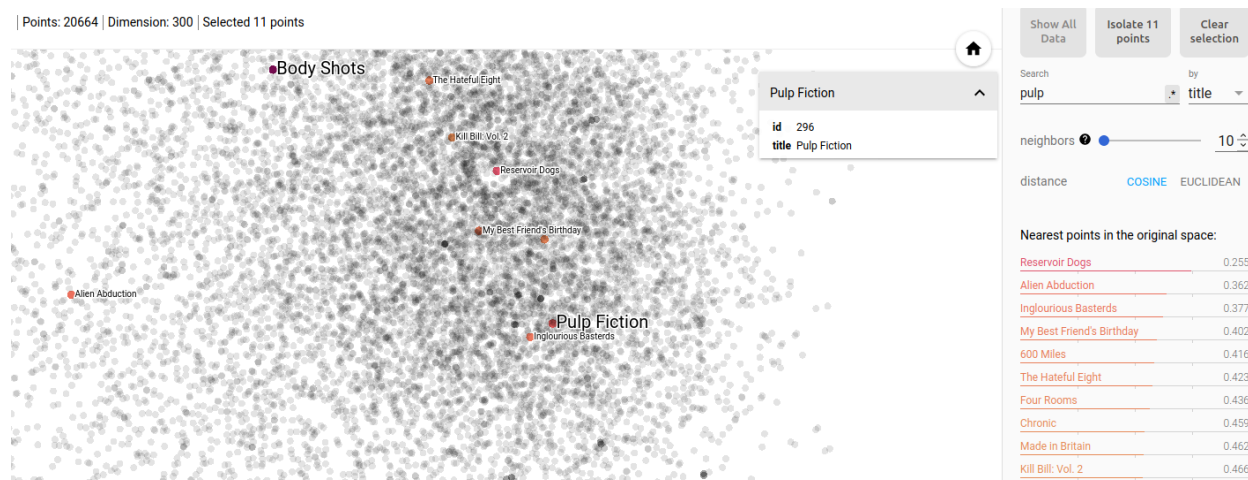


Figure 3: Pulp Fiction.

For more details, check the `04-content-based-embedding.ipynb` notebook.

## Collaborative Filtering

We extract dense features from the utility matrix using the [LightFM](#) algorithm with `logistic` loss (to deal with +1/-1 values). It took about 4 minutes to extract the user and item factors. The last are our embedding features. As in the content based approach, we set 300 as the embedding space dimension.

The LightFM algorithm learns the user and item factors with a bias term. It's important to overcome the popularity bias problem. Let's check how it relates to the movie average rating:

We see that the bias term captures the movie rating information. This way we remove the popularity bias from the embedding features.

Now, let's visualize the embedding features using the *Embedding Projector*:

We have different neighbors when compared to the content based, but they make sense too.

## Hybrid Approach

It consists on combining content based with collaborative filtering embedding features. Computing the cosine similarity over this combination is the same as computing the cosine similarity individually over content based and collaborative filtering and taking the average between the two values.

All code is organized and documented in the `08-hybrid-approach.ipynb` notebook. In the evaluation section we will show that content based and collaborative filtering are complementary solutions, which may be a evidence that the hybrid approach is more powerful.

## Similarity Match with ANN

Using the [faiss](#) package we can efficiently find similar movies based on the cosine similarity.

In the `06-similarity-match-with-ann.ipynb` notebook we experimented with the embedding features using the `faiss` package. First we got the 30 nearest neighbors to each movie, which we will use to compute precision and recall at k. Then, we computed the cosine similarity between each movie and its 30 nearest neighbors given by user correlation. They will be used to compute ranking correlation in the next section.

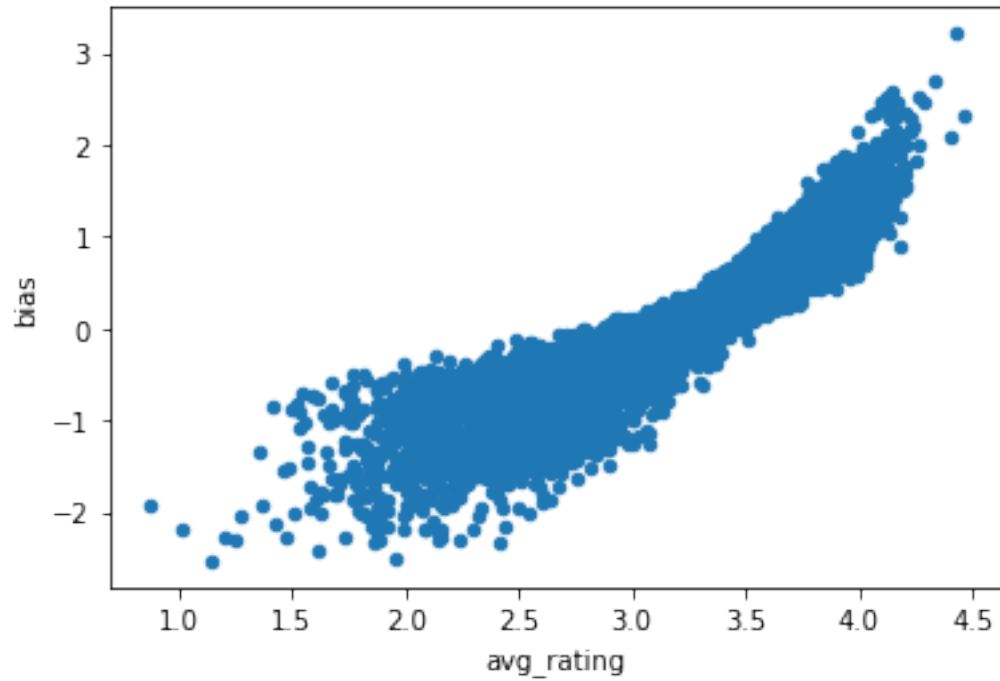


Figure 4: Item bias fitted by the LastFM algorithm against the movie average rating.

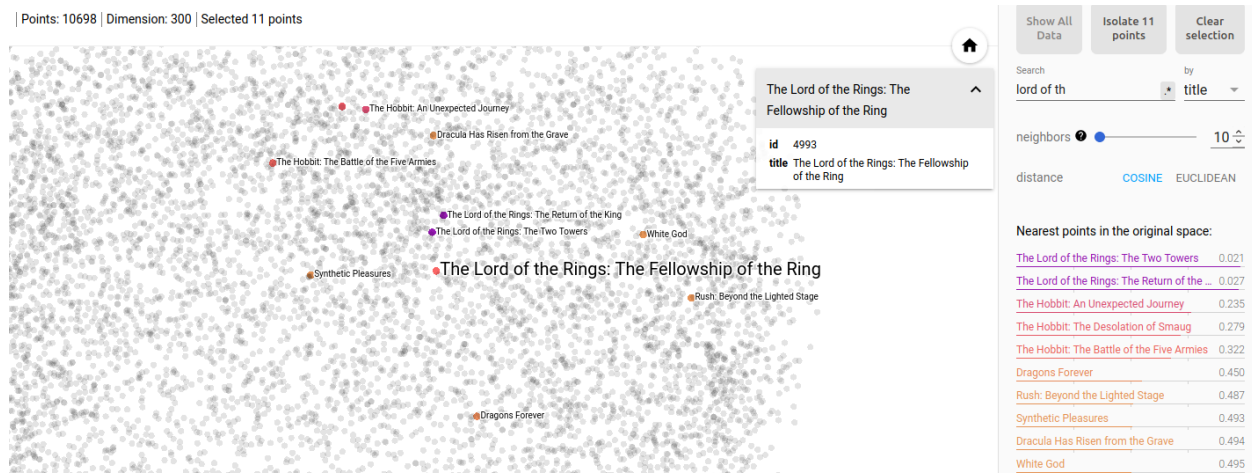


Figure 5: The Lord of the Rings.

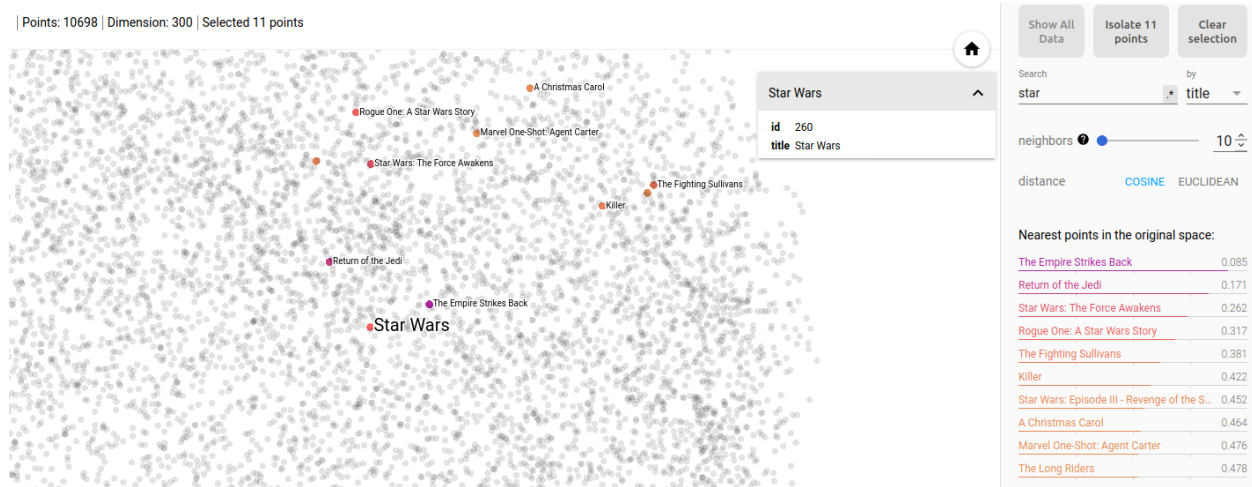


Figure 6: Star Wars.

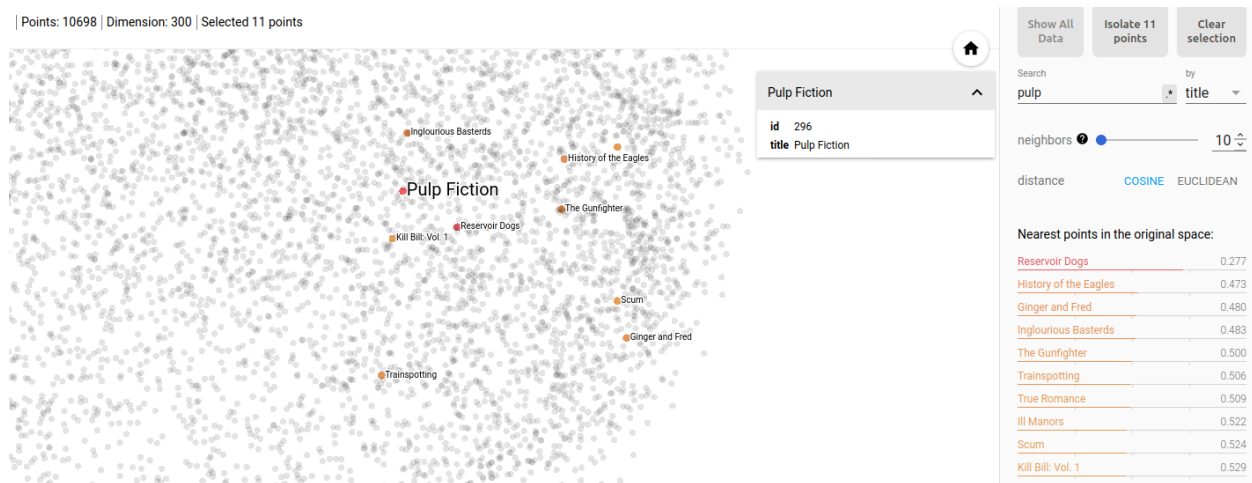


Figure 7: Pulp Fiction.



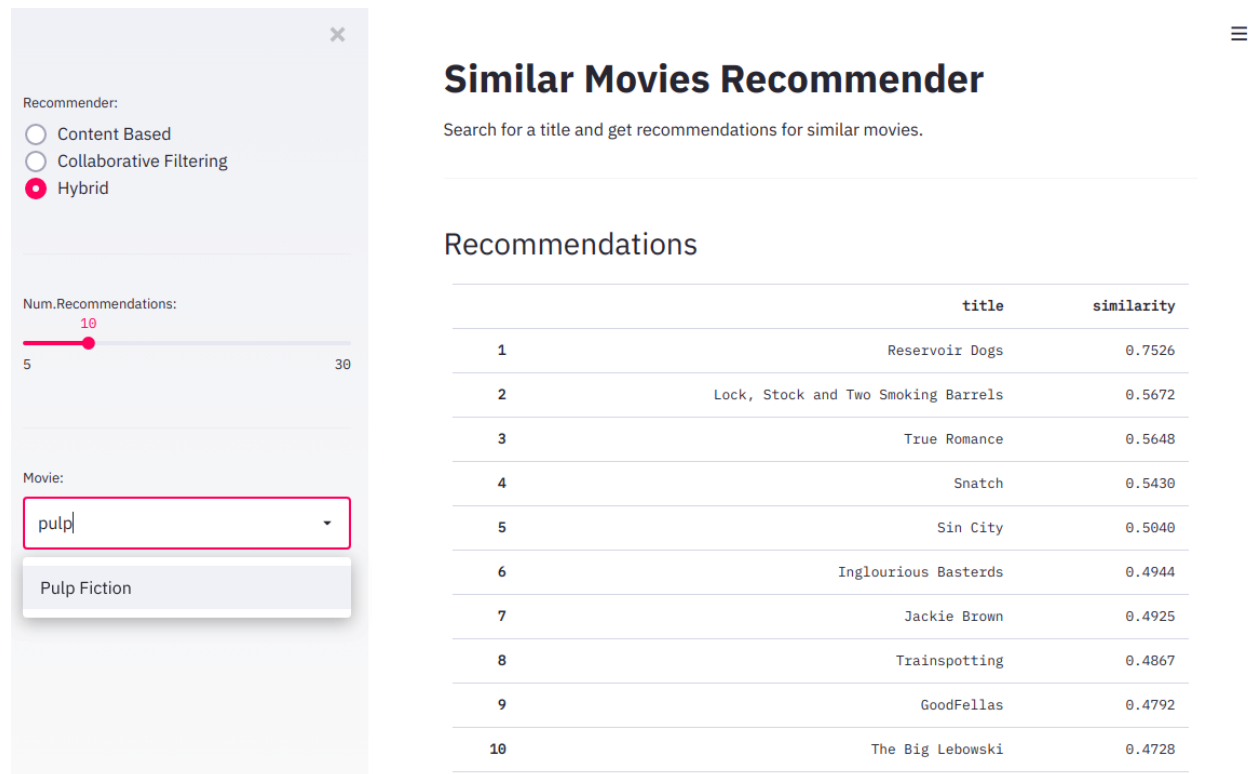
## Public Movie Embedding

We published the embedding features (content based + collaborative filtering) on the Embedding Projector. You can play with it here: [Movie Embedding](#).

More details can be seen in the `projector` folder.

## Movie Similarity Dashboard

Finally, we built a web application on which the user can choose a movie and get recommendations of similar movies, using the [Streamlit](#) ([link](#)) framework:



**Similar Movies Recommender**

Search for a title and get recommendations for similar movies.

Recommender:

- ☐ Content Based
- ☐ Collaborative Filtering
- ☒ Hybrid

Num.Recommendations:

5 10 30

Movie:

pulp

Pulp Fiction

	title	similarity
1	Reservoir Dogs	0.7526
2	Lock, Stock and Two Smoking Barrels	0.5672
3	True Romance	0.5648
4	Snatch	0.5430
5	Sin City	0.5040
6	Inglourious Basterds	0.4944
7	Jackie Brown	0.4925
8	Trainspotting	0.4867
9	GoodFellas	0.4792
10	The Big Lebowski	0.4728

We provide a `Dockerfile` to easily deploy the application in your computer. Follow the instructions in the project `README.md`.

## Refinement

Since we are not working with labeled data, hyper-parameter tuning is more complicated here. We set all parameters based on past knowledge or based on common choices in literature.

## IV. Results

### Model Evaluation and Validation

In this section we compare recommendations based on the embedding similarity match with recommendations based on user correlation. The later is just a reference. We don't have the "correct" similar movies to measure performance properly. In an e-commerce, for example, we compare algorithms using A/B testing with

business metrics like CTR and conversion. The better the business metric, the better the recommendation approach.

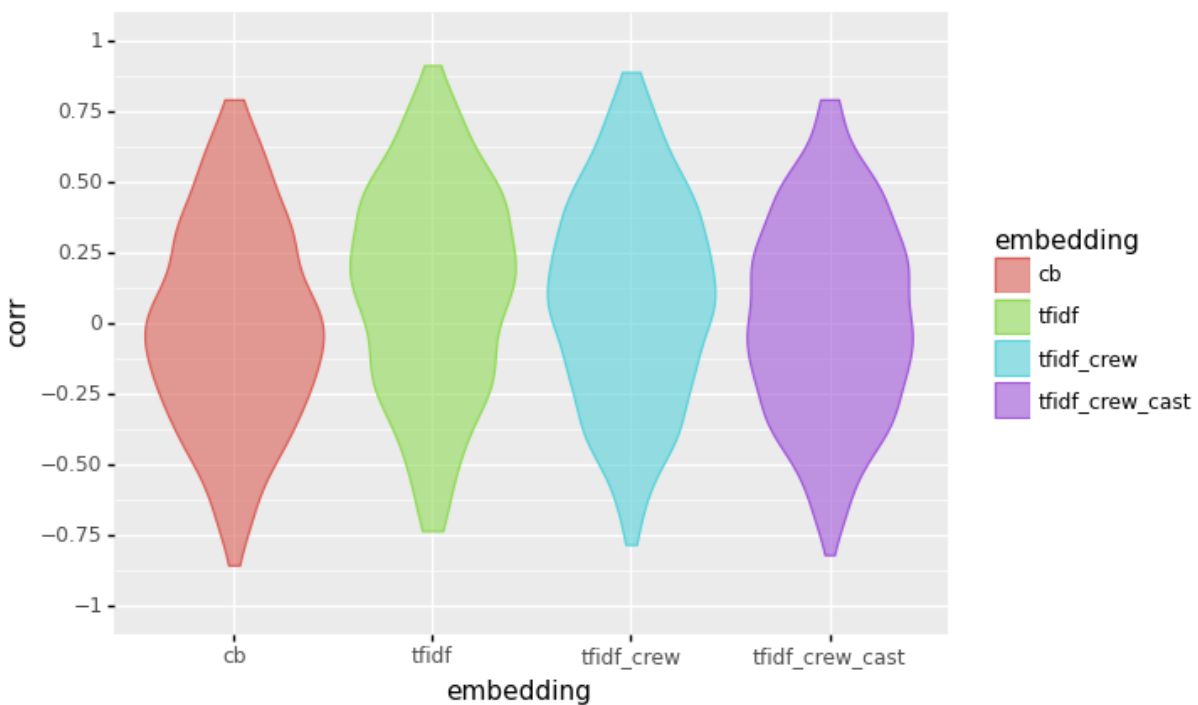
The complete analysis is provided in the `07-performance-evaluation.ipynb` notebook.

## Ranking Correlation

In the Similarity Match section we have computed the cosine similarity between pairs of movies. Now, we compare the user correlation ranking with the cosine similarity ranking based on the embedding features, using the Spearman correlation.

The Spearman Ranking Correlation Coefficient give us a value between -1 (exactly opposite order) and +1 (same order). Like the classical Pearson's Correlation Coefficient, values above 0.3 can be seen as an evidence of correlation.

For each target movie (the 500 movies selected in the Benchmark section) we computed the Spearman correlation between the user similarity values and the cosine similarity values for the movie's 30 nearest neighbors. To evaluate the performance, we can observe the distribution of the correlation measures for all 500 target movies, as follows:



The embedding features based on text data performed a little bit better than other approaches. It's a surprise, since text data does not carry any information about users. But there is a large overlapping between the distributions. So, there is no strong evidence of difference between methods.

In addition, including Cast&Crew information doesn't seem to improve the performance of the content embedding features. The Pearson's correlation between the Tf-Idf results and the Tf-Idf + Cast&Crew results is above 0.7. So, Cast&Crew seems to provide redundant information to the embedding features.

## Precision and Recall

Another way to check ranking performance is to transform the problem into a classical binary classification problem, so that we can take advantage of better known metrics. Precision and recall are the most common ones.



First of all, we must label some items as ‘relevant’. Then, we look for relevant items in the top K positions of the ranking. K in [5,10] is a common choice. Here we choose k=10.

Precision is given by the number of relevant items in the top K divided by K. In other words, it measures the proportion of the top K items that are relevant. Recall is given by the number of relevant items in the top K divided by the total number of relevant items. In other words, it measures the proportion of relevant items that are in the top K items.

We assigned as relevant all similar movies with user correlation above 0.3. Then, to make a better evaluation, we considered only those target movies with 3 or more relevant similar movies.

We got very bad results for precision and recall. In general, the relevant movies are not appearing in the top 10 given by the embedding ranking. Here it follows what we got:

- Precision for the Tf-Idf Embedding:

stats	value
mean	0.016667
std	0.057985
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	0.400000

- Recall for the Tf-Idf Embedding

stats	value
mean	0.061658
std	0.191476
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

- Precision for the CF Embedding

stats	value
mean	0.007471
std	0.030439
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	0.200000

- Recall for the CF Embedding

stats	value
mean	0.048851
std	0.199571
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

### Popularity Bias

In addition, we checked if the embedding features carry any type of popularity bias.

As expected, content based embedding is not subjected to popularity bias:

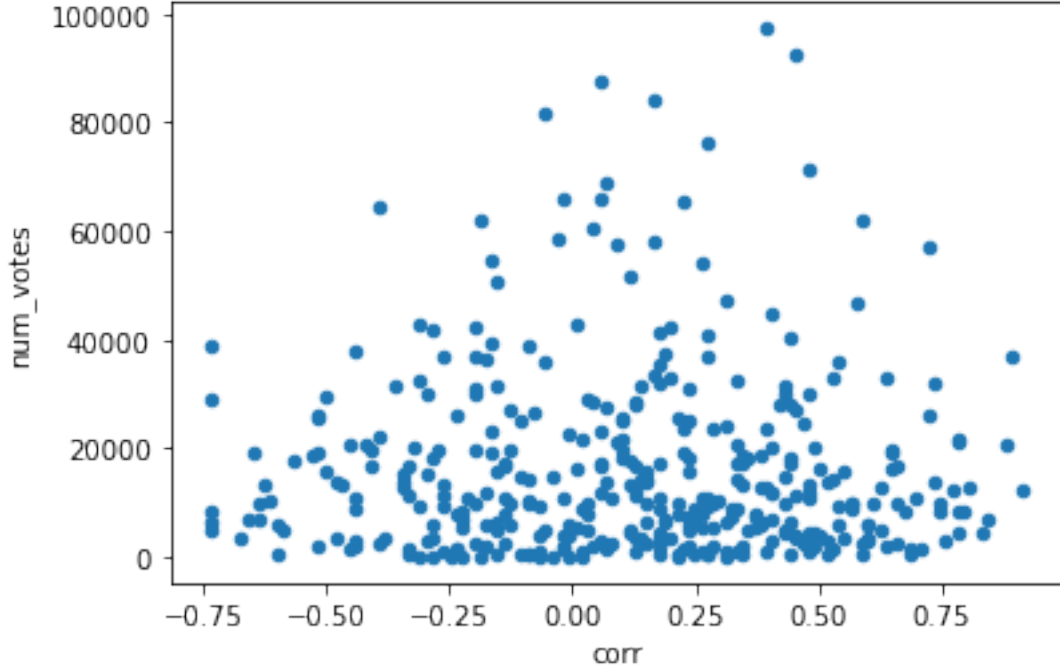


Figure 8: Movie popularity against ranking correlation values for Content Embedding

How about the collaborative filtering embedding?

No popularity bias can be observed, indicating that the bias term in factorization algorithm was fitted properly.

### Correlation between Content Based and Collaborative Filtering Embedding

We saw that Cast&Crew brings redundant information to the Content Embedding. How about Collaborative Filtering? Can we expect similar results between Content Based and Collaborative Filtering Embedding?

We can observe that there is no strong correlation between text embedding and collaborative filtering embedding. The Pearson's correlation between them is 0.2. It may be indicative that both methods are complementary. So, combining them can provide an even better performance. It is the intuition behind the Hybrid Approach.

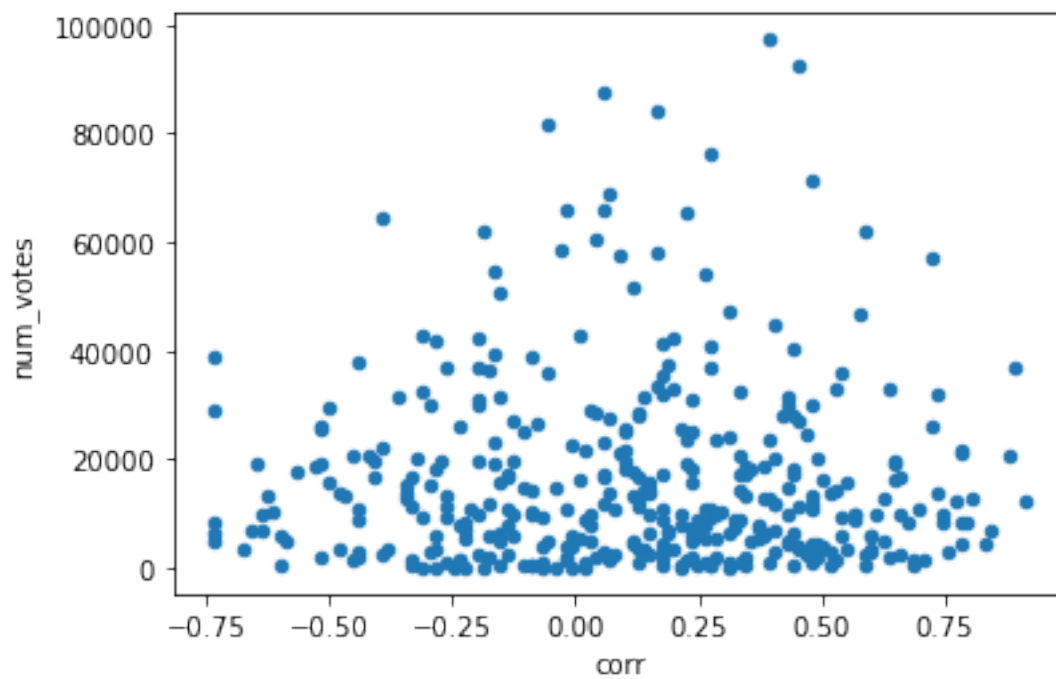


Figure 9: Movie popularity against ranking correlation values for Collaborative Filtering

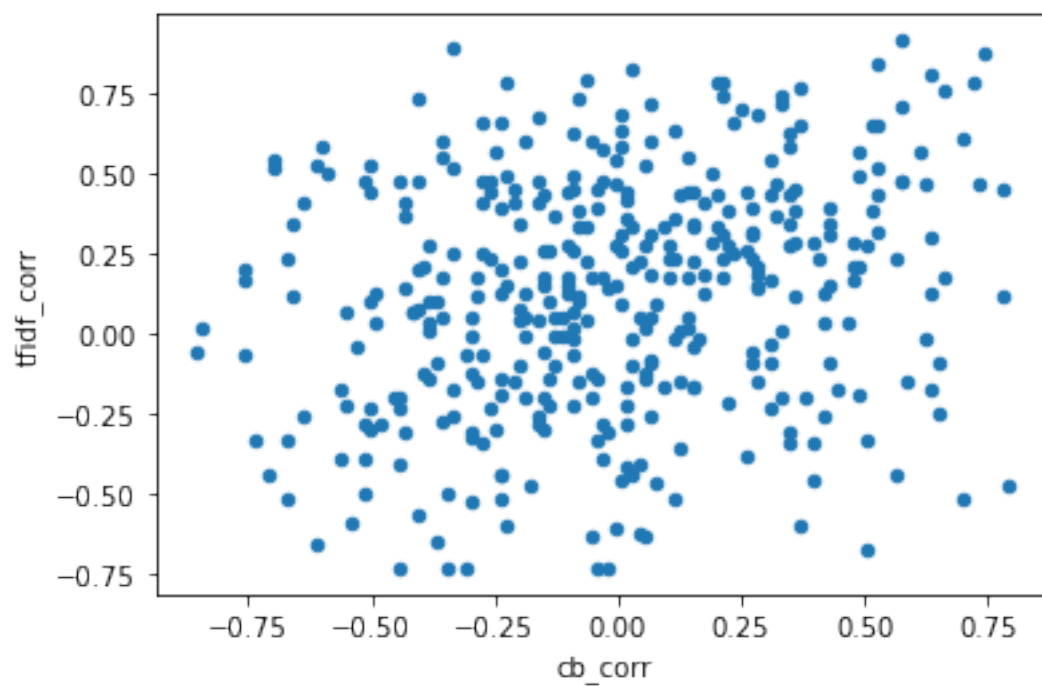


Figure 10: Ranking correlation results for Content Based Embedding against Collaborative Filtering Embedding

## Justification

We got very bad results for precision and recall. But it is not a problem, since the user correlation similarity is just a reference for us. In the previous tests we saw interesting recommendations given by embedding similarity. We had to choose a method to compare with and we tried a reference that provides good recommendations, but is sparse (you need users in common to link a pair of movies) and very expensive to compute. Maybe a method based on the most popular movies (always recommend the most popular movies) could provide a better reference.

Anyway, in the recommendation dashboard we can get really interesting recommendations for the embedding methods.

## V. Conclusion

### Free-Form Visualization

Embedding features can efficiently capture content and user-interaction information and represent than in a low-dimensional and dense space. It allows fast nearest neighbors searching and provides continuous recommendations. We are able to establish a link between any pair of items, where a distance measure provides the evidence of similarity between the pair. In addition, they allow combining information from different sources, like we did in the Hybrid approach.

Moreover, we are able to visualize the embedding features using techniques like PCA, t-SNE or UMAP. It can be very insightful!

Here it follows a screenshot from the Embedding Projector:

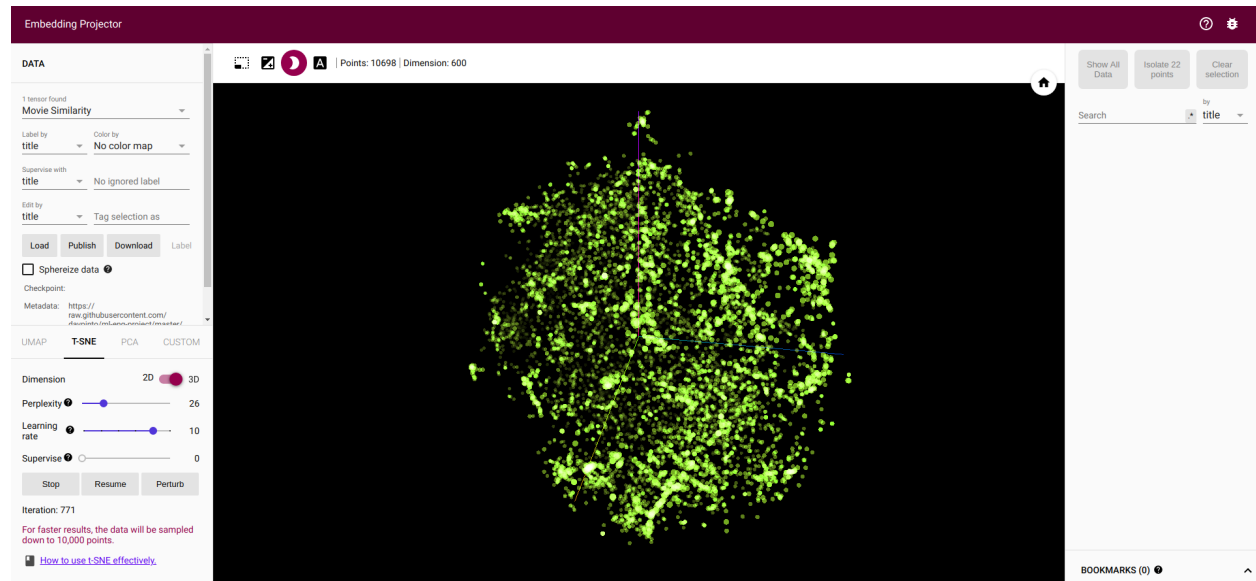
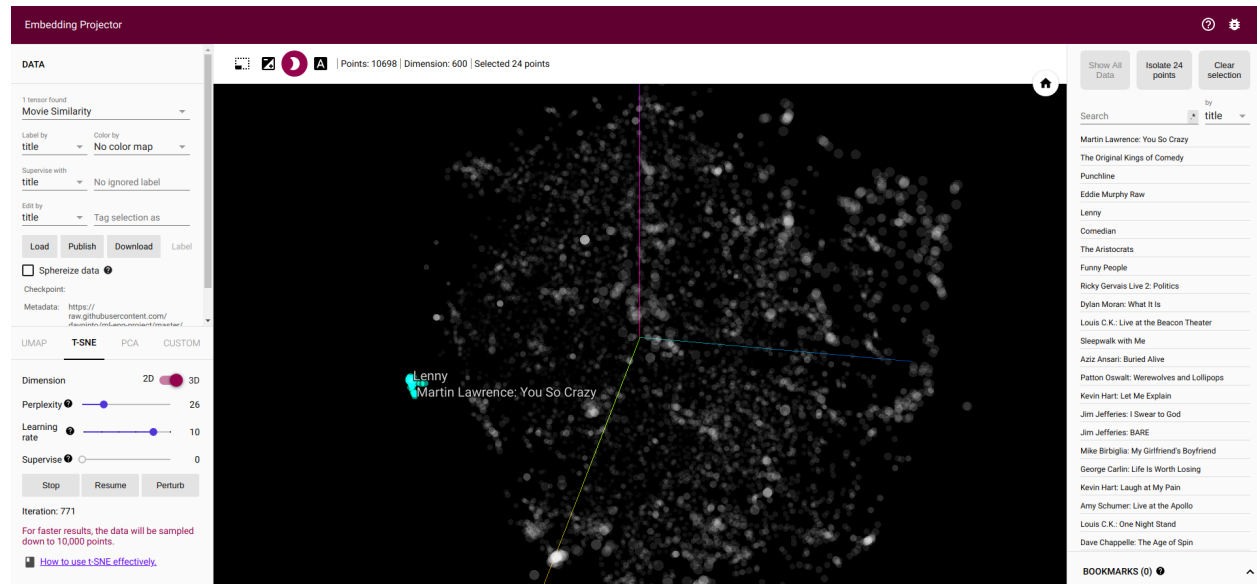


Figure 11: Embedding features in a 3D space generated with the t-SNE algorithm.

We can observe a large number of small groups. They are movie clusters, grouped by content similarity and/or user-interaction similarity.

Let's inspect one of them:



Nice! It's a Comedian cluster.

So, embedding features have a large contribution! They are very powerful for recommendation, but can be effectively applied in many other Machine Learning problems, like movie clustering, movie genre classification, movie success prediction, movie profitability prediction, so on.

## Reflection

The problem solved in this project was very challenging. We choose a large dataset to work with. It came from 2 sources, one that contains movie metadata and other that contains user-interaction data. Firstly, we had to transform data to a tidy format using a quite advanced data cleaning skills. Then, we carefully represent all features as numerical sparse vectors to allow fast and memory efficient embedding extraction. Some data manipulations required more robust tools, like `vaex`. We carried about the nearest neighbor searching performance, so we brought `faiss` to solve the problem.

All steps were new for me! I had to study every technique and every package used here.

To make the results accessible by others, we published the embedding features on the Embedding Projector. In addition, we created a web application that allows experimenting with Content Based, Collaborative Filtering and Hybrid recommendations. The application runs nearest neighbors searching in real time and provide recommendations in milliseconds. It can be deployed easily through the `Dockerfile` we provide. We tried to deploy it on Heroku, but after a entire night trying, we decided to move on.

In summary, I'm very proud about the results! I know that I'm subjected to personal bias. But I kept a version of the application deployed in my computer and I'm using it to get recommendations based on my favorite movies. I'm enjoying it. I even watched some of the recommendation I received!

## Improvement

I would like to use the `implicit` package to replace the TruncatedSVD algorithm. It provides some modern factorization algorithms that can improve embedding representation.

The `LightFM` allows to factorize the utility matrix together with item and user metadata. It is an interesting alternative to the Hybrid approach. I would like to experiment with it.

Graph algorithms are becoming so popular nowadays. The utility matrix can be represented as a graph, where nodes are the movies and edges are users that interacted with them. Then, we can apply some Graph

Embedding technique like DeepWalk, Node2Vec or VERSE to extract movie embedding features. It's a very promising solution.

## References

- [1] [Google's Introduction to Recommendation Systems](#).
- [2] [Building a real-time embeddings similarity matching system](#).
- [3] [The Amazon Recommendations Secret to Selling More Online](#).
- [4] Anderson, Chris. The long tail: Why the future of business is selling more for less. Hyperion, 2006.
- [5] [Overview: Extracting and serving feature embeddings for machine learning](#).