UDC 004.8

# RECOMMENDATION SYSTEM FOR VIDEO HOSTING

***Anastasiia Antonova***[1], *Ganna Sydorenko*[2]

[1] *master student of the department of Computer Science, NTU "KhPI", Kharkiv, Ukraine*
[2] *Associate Professor of Systems Analysis and Information and Analytical Technologies, Ph.D. tech. Sciences, NTU "KhPI", Kharkiv, Ukraine*
*ganna.sydorenko@khpi.edu.ua*

For any given product, there are sometimes thousands of options to choose from: streaming videos, social networking, online shopping; the list goes on. The recommender systems help to personalize a platform and help to find the user something they like. The easiest and simplest way to do this is to recommend the most popular items. However, to really enhance the user experience through personalized recommendations, we need to dedicate recommender systems. Various sources say that as much as 35 – 45% of tech giants' revenue comes from recommendations alone.

We are interested in doing this because people like to watch films ourselves and quite often find themselves faced with problem such as "nothing to watch". To solve this problem, people need someone, or as in this situation, something who can tell them what to choose to watch, based on examples of what them like to watch. This is important because millions of people watch movies. This problem is solvable, because the person concerned needs a hint for choosing, and this solution can give that hint with our program. Simply put a Recommendation system is a filtration program which prime goal is to predict the "rating" or "preference" of a user towards a domain-specific item. In this case, this domain-specific item is a movie, therefore the main focus of this recommendation system is to filter and predict only those movies which a user would prefer given some data about the user.

The Simple Recommender offers generalized recommendations to every user based on movie popularity and, sometimes, genre. The basic idea behind this recommender is that movies which are more popular and more critically acclaimed will have a higher probability of being liked by the average audience. This model does not give personalized recommendations based on the user. We use the TMDB Ratings to come up with our Top Movies Chart. We will use IMDB's weighted rating formula (1) to construct a chart. Mathematically, it is represented as follows

$$Weighted\ Raiting(WR) = (vv + m.R) + (mv + m.C), \quad (1)$$

where:

- $v$ is the number of votes for the movie;
- $m$ is the minimum votes required to be listed in the chart;
- $R$ is the average rating of the movie;
- $C$ is the mean vote across the whole report.

The next step is to determine an appropriate value for $m$, the minimum votes required to be listed in the chart. We will use 95% as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list. Therefore, to qualify to be considered for the chart, a movie has to have at least 434 votes on TMDB. We also see that the average rating for a movie on TMDB is 5,244 on a scale of to 10. 2274 movies qualify to be on our chart.

For the Movie Description based Recomender let's try to build a recommender using movie descriptions and taglines. We do not have a quantitative metric to evaluate the performance of our machine, so it will have to be done qualitatively. We will use Cosine

similarity (2) to calculate a numerical value that denotes the similarity between two movies. Mathematically, it is defined as follows:

$$cosine(x, y) = \frac{x \cdot y^T}{\|x\| \cdot \|y\|}. \quad (2)$$

Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine Similarity Score.

In this part, we build a simple Hybrid Recommender that brings together techniques we have implemented in the content based and collaborative filter-based engines (Fig. 1-2). This is how it will work:

Input: User ID and the Title of a Movie;

Output: Similar movies sorted based on expected ratings by that particular user.

```
[ ] def convert_int(x):
        try:
            return int(x)
        except:
            return np.nan

    id_map = pd.read_csv('/content/sample_data/links_small.csv')[['movieId', 'tmdbId']]
    id_map['tmdbId'] = id_map['tmdbId'].apply(convert_int)
    id_map.columns = ['movieId', 'id']
    id_map = id_map.merge(smd[['title', 'id']], on='id').set_index('title')
    #id_map = id_map.set_index('tmdbId')

    indices_map = id_map.set_index('id')
    def hybrid(userId, title):
        idx = indices[title]
        tmdbId = id_map.loc[title]['id']
        #print(idx)
        movie_id = id_map.loc[title]['movieId']

        sim_scores = list(enumerate(cosine_sim[int(idx)]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
        sim_scores = sim_scores[1:26]
        movie_indices = [i[0] for i in sim_scores]

        movies = smd.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year', 'id']]
        movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
        movies = movies.sort_values('est', ascending=False)
        return movies.head(10)
```

Fig. 1 – Code of Hybrid Recommender.

```
[ ] hybrid(1, 'Avatar')
```

| | title | vote_count | vote_average | year | id | est |
|---|---|---|---|---|---|---|
| 522 | Terminator 2: Judgment Day | 4274.0 | 7.7 | 1991 | 280 | 3.182332 |
| 974 | Aliens | 3282.0 | 7.7 | 1986 | 679 | 3.059914 |
| 8401 | Star Trek Into Darkness | 4479.0 | 7.4 | 2013 | 54138 | 2.967379 |
| 1011 | The Terminator | 4208.0 | 7.4 | 1984 | 218 | 2.957057 |
| 8658 | X-Men: Days of Future Past | 6155.0 | 7.5 | 2014 | 127585 | 2.929055 |
| 2014 | Fantastic Planet | 140.0 | 7.6 | 1973 | 16306 | 2.836898 |
| 1668 | Return from Witch Mountain | 38.0 | 5.6 | 1978 | 14822 | 2.817311 |
| 4017 | Hawk the Slayer | 13.0 | 4.5 | 1980 | 25628 | 2.799295 |
| 344 | True Lies | 1138.0 | 6.8 | 1994 | 36955 | 2.788043 |
| 922 | The Abyss | 822.0 | 7.1 | 1989 | 2756 | 2.726234 |

```
[ ] hybrid(500, 'Avatar')
```

| | title | vote_count | vote_average | year | id | est |
|---|---|---|---|---|---|---|
| 8401 | Star Trek Into Darkness | 4479.0 | 7.4 | 2013 | 54138 | 3.452798 |
| 974 | Aliens | 3282.0 | 7.7 | 1986 | 679 | 3.342840 |
| 1621 | Darby O'Gill and the Little People | 35.0 | 6.7 | 1959 | 18887 | 3.330212 |
| 4017 | Hawk the Slayer | 13.0 | 4.5 | 1980 | 25628 | 3.241260 |
| 1011 | The Terminator | 4208.0 | 7.4 | 1984 | 218 | 3.236047 |
| 8658 | X-Men: Days of Future Past | 6155.0 | 7.5 | 2014 | 127585 | 3.181210 |
| 522 | Terminator 2: Judgment Day | 4274.0 | 7.7 | 1991 | 280 | 3.172731 |
| 4347 | Piranha Part Two: The Spawning | 41.0 | 3.9 | 1981 | 31646 | 3.169526 |
| 2014 | Fantastic Planet | 140.0 | 7.6 | 1973 | 16306 | 3.109247 |
| 2132 | Superman II | 642.0 | 6.5 | 1980 | 8536 | 3.014034 |

Fig. 2 - Examples of using the recommender using different parameters.

In the conclusion, for Hybrid Recommender Engine we brought together ideas from content and collaborative filtering to build an engine that gave movie suggestions to a particular user based on the estimated ratings that it had internally calculated for that user.

**The list of references:**


**1**. *Z. A.Aston* Dive into Deep Learning / *Z. A.Aston, Z. C. Lipton, M. Li, A. J. Smola.*, 2020.

**2.** *G. James* An Introduction to Statistical Learning: With Applications in R / *G. James, D. Witten, T. Hastie.*, 2014.

**3.** *M.Ekstrand* Letting Users Choose Recommender Algorithms: An Experimental Study / *M.Ekstrand, D. Kluver, F. Harper, J. Konstan*. // RecSys'15. – 2015. – Pages 11–18.

**4.** *Raschka S.* Python machine learning / *S. Raschka*., 2015.