Recommender systems

Jose Pedro Esteves - m11674 – MEI – DI - UBI

Abstract— Recommender systems play a huge role in our lives nowadays. We are presented to them on advertising campaigns through ad targeting, search engines response list and even on supermarket item distribution. Either way, one of the most common applications of recommendation engines is toward the streaming industry. In this document we will look at how to build, train and evaluate a small recommendation engine based on the use of collaborative filtering. At the end of the project, for each user in the system, we should be able to get a list of 20 movies that the user has not seen yet and a rating for any movie within the list that indicates the prediction (likelihood) of the user to enjoy that movie.

Index Terms—Recommender, Netflix, Ad Targeting, Collaborative Filtering, ModeBased, RMSE, MAE, Machine learning

1 INTRODUCTION

It is impossible to speak about recommender systems without mentioning Netflix and the Netflix $1M contest prize.

The year was 2006 and Netflix, a company that rented DVDs and shipped them physically by mail, wanted to improve, by at least 10%, the recommendations they sent to their customers. In order to achieve their goal, they launched a contest with a big money prize to pay to the person/team/company that could bring they collaborative filtering algorithm, called Cinematch to a hole new level.

For that, "Netflix provided a training data set of 100,480,507 ratings that 480,189 users gave to 17,770 movies. Each training rating is a quadruplet of the form <user, movie, date of grade, grade>. The user and movie fields are integer IDs, while grades are from 1 to 5 (integer) stars. The qualifying data set contains over 2,817,131 triplets of the form <user, movie, date of grade>, with grades known only to the jury. A participating team's algorithm must predict grades on the entire qualifying set, but they are informed of the score for only half of the data: a quiz set of 1,408,342 ratings. The other half is the test set of 1,408,789, and performance on this is used by the jury to determine potential prize winners. Only the judges know which ratings are in the quiz set, and which are in the test set—this arrangement is intended to make it difficult to hill climb on the test set. Submitted predictions are scored against the true grades in the form of root mean squared error (RMSE), and the goal is to reduce this error as much as possible. Note that, while the actual grades are integers in the range 1 to 5, submitted predictions need not be. Netflix also identified a probe subset of 1,408,395 ratings within the training data set. The probe, quiz, and test data sets were chosen to have similar statistical properties. " [26]

The contest closed down in 2009 and had the "BellKor's Pragmatic Chaos" as the prize winner (a Test RMSE of 0.8567).

2 STATE OF THE ART

2.1 Leraning Models

Today, we have many models that provide API interfaces to algorithms that, in an abstract way, provide us with many options to address this machine learning problems and provide the machines with some level of atifcial intelligence through machine learning.

In this chapter we will make a high-level description of the existing models and what they are used for.

2.1.1 Decision Tree

Decision trees are structures in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represent the classification rules. Just about everyone has used the decision tree technique. Either formally or informally we decide on a single course of action from many possibilities based on previous experience. A decision tree offers a great deal of flexibility in terms of the input values it can receive. Also, the tree output can take the form of a binary, a value or a category. On the other hand, they force that any decision has to be either a yes or a no, one or zero. Moreover, the decision criteria can consider only a single variable at a time. There cannot be a combination of more than one input variable.

Decision trees cannot be updated incrementally, ie, once a tree is trained on a training set, if we get a new training data, a new tree must be created.

2.1.2 Linear Regression

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.

The premise of linear regression methods rests on the assumption that the output (numeric value) may be expressed as a combination of the input variable set (also numeric).

A weakness of the linear regression model is the fact that it assumes linear input features, which might not be the case. Inputs must be tested mathematically for linearity.

2.1.3 K-Means Clustering Algorithm

It's an unsupervised algorithm from cluster analysis. It's an iterative, non-deterministic method. It operates on data sets using predefined clusters (think about grouping into categories).

K-Means clustering algorithm can be used to group results that talk about similar concepts.

These clusters are usually formed by a set of rules and/or shared properties, for each property the K-Nearest algorithm is often applied. One of the most common uses for this algorithm is to provide users with search results.

2.1.4 Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.

Unlike other algorithms, neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

A neural network works similarly to the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis. A neural network contains layers of interconnected nodes. Each node is a known as perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

2.1.5 Bayesian Network

Bayesian Networks produce probabilistic relationships between outputs and inputs. This type of network requires all data to be binary. The strengths of this model include high scalability and support for incremental learning.

They are particularly good at classification tasks, such as detecting if an email is or is not spam.

2.1.6 Support Vector Machine

Support Vector Machine (SVM) are supervised machine learning algorithms that can be used for classification and regression analysis, although they are most often used for classification.

SVM works by dividing categories using a hyperplane (basically, a flat sheet in 3 dimensions). On one side of the hyperplane will be one category, on the other side, another category. This works well if the categories to be to be separated are clearly divisible.

SVM training times are usually high, so this algorithm isn't as well suited to very large datasets, as the larger the dataset, the longer it takes to complete the training.

2.1.7 Model-Based Collaborative Filtering

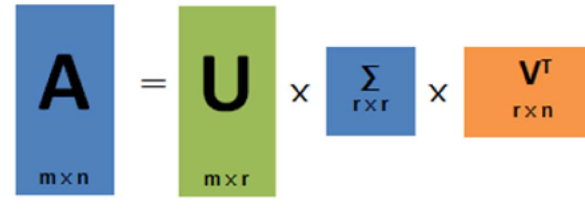
Model-based Collaborative Filtering is based on matrix factorization (MF) which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF:

The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items.

When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization, you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector.

You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

A well-known matrix factorization method is Singular value decomposition (SVD). At a high level, SVD is an algorithm that decomposes a matrix **A** into the best lower rank (i.e. smaller/simpler) approximation of the original matrix **A**. Mathematically, it decomposes **A** into a two unitary matrices and a diagonal matrix:



where **A** is the input data matrix (users's ratings), **U** is the left singular vectors (user "features" matrix), **Σ** is the diagonal matrix of singular values (essentially weights/strengths of each concept), and **VT** is the right singluar vectors (movie "features" matrix). **U** and **VT** are column orthonomal, and represent different things. **U** represents how much users "like" each feature and **VT** represents how relevant each feature is to each movie.

To get the lower rank approximation, we take these matrices and keep only the top **k** features, which can be thought of as the underlying tastes and preferences vectors.

2.3 Applications

Along with this document, we had previously identified a big set of possible applications for this algortithms, but there are many, many more that we can point out and that we use in our day-to-day applications.

- Virtual assistants, like Google Assistance, Siri, Alexa and Cortana;
- Online fraud detection.
- Data Security
- Commuting Predictions for maps;
- Personal Security eg, in airports;
- Video survilance;
- Social Media Services;
- Online Customer Support, via chatbots;
- Search Engine Results and Refinement;
- Product Recommendations;
- Market Personalization;
- Finatial Trading;
- Healthcare;
- Natural Language processing;
- SmartCards;
- IOT;
- ….

3 PRATICAL WORK PRODUCED

At the beginning of the semester, we have defined a set of requirements for this assignment. During the implementation, we have focused on meeting the requirement to provide the list of recommendations for a user.

The other requirements defined are eventually, either not related with this subject as they can be achieved with system architecture design or, with other types of implementations, not necessarily related to Machine Learning, eg, the json response.

We have decided to descope the amount of decision helpers and the weight of that, BUT, many of them are already included in the dataset, as we will be showing later on the document.

3.1 The Dataset

The dataset files consist of 3 different files. Those files were extracted from real usage databases and are anonymously kept.

movies.csv - Contains all the assets on the database. It includes not only the playables, but also the series and seasons. Extracted using the following SQL code:

```
-- Movies.csv
select a.id as movie_id,  case
    when a.assetable_type='movies' then m.original_title
    when  a.assetable_type='series' then s.original_title
    when  a.assetable_type='seasons' then s.original_title
    when a.assetable_type='episodes' then e.original_title
    when a.assetable_type='channels' then c.original_title
    end as title,
    group_concat(vag.slug separator '|') as genres
from assets a
    left join movies m on a.assetable_type='movies' and
a.assetable_id=m.id
    left join series s on a.assetable_type='series' and
a.assetable_id=s.id
    left join seasons se on a.assetable_type='seasons' and
a.assetable_id=se.id
    left join episodes e on a.assetable_type='episodes' and
a.assetable_id=e.id
    left join channels c on a.assetable_type='channels' and
a.assetable_id=c.id
    join vw_asset_genres vag on a.id=vag.id
where
    a.deleted_at is null
    and a.assetable_type in
('movies','series','seasons','episodes','channels')
group by vag.id
```

users.csv - Contains all the users in the database. Since there was the need of anonymizing the data and there was no real good reason to export data other than the user id, most of the fields are facked via the SQL code:

```
- users.csv
select
    ua.id as user_id,
    coalesce(gender, 'N') as gender,
    round(rand()*100,0) as age,
    round(rand()*10,0) as occupation,
    coalesce(address, 6200) as zipcode,
    coalesce(ag.classification, 'G') as age_desc,
    'NA' as occ_desc
from users_app ua
    left join age_groups ag on  ua.agerating=ag.id
where deleted_at is null;
```

rattings.csv - Initially, we had only extracted the ratings, but that resulted in a small dataset. This small dataset and the fact that the sparsity level was very high (99.5%) raised problems in classification as the algorithm was not able to find good -when any - matches for most of the users.

```
-- ratings
select
    user_id as user_id ,
    asset_id as movie_id,
    round(rating/20,0) as rating,
    UNIX_TIMESTAMP(ur.created_at) as timestamp,
    user_id-1 as user_emb_id,
    asset_id-1 as movie_emb_id
from users_ratings ur
join assets on ur.asset_id = assets.id and deleted_at is not null
group by user_id, asset_id
;
```

```
Number of users = 3995 | Number of movies = 415
The sparsity level of dataset is 99.5%
```

	movie_id	...	genres
0	24770	...	earthlings-love-drama
1	24772	...	animated-universe
2	24774	...	a-childs-perspective|supernatural
3	24775	...	deep-space-adventures|famous-faces
4	24777	...	earthlings-love-drama|technology
5	24778	...	aliens-monsters|famous-faces
6	24779	...	essential-action|ai-robots
7	24780	...	aliens-monsters|technology
8	24781	...	essential-action|mysteries-abound
9	24782	...	technology|the-future-of-love
10	24783	...	mysteries-abound|out-of-this-world-adventures
11	24784	...	comedy-is-universal|paradoxes
12	24785	...	earthlings-love-drama|technology
13	24786	...	earthlings-love-drama|technology|the-future-of...
14	24787	...	body-horror|drama|mystery|psychological|horror...
15	24789	...	mystery|thriller|drama|experimental|out-of-thi...
16	24790	...	out-of-this-world-adventures
17	24791	...	the-future-is-female|ai-robots|out-of-this-wor...
18	24792	...	earthlings-love-drama|paradoxes|the-future-is-...
19	24793	...	aliens|creature-feature|drama|character-driven...

[20 rows x 3 columns]

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.2312	1.2066	1.1931	1.2103	1.1942	1.2071	0.0138
MAE (testset)	0.9477	0.9340	0.9248	0.9293	0.9350	0.9342	0.0077
Fit time	0.38	0.40	0.39	0.43	0.39	0.40	0.02
Test time	0.01	0.01	0.01	0.01	0.01	0.01	0.00

In order to fix that problem, the approach was to extend the dataset to include other features that were already planned to be included as decision helpers.

On this step we could have had considered to include a ponderation on the bookmark value, but this can be scoped as future work.

As per the above, the SQL code used to export the ratings data is:

```sql
drop view if exists vw_recomendation_rattings;
create view vw_recomendation_rattings as
select
    user_id as user_id ,
    asset_id as movie_id,
    round(rating/20,0) as rating,
    UNIX_TIMESTAMP(ur.created_at) as timestamp,
    user_id-1 as user_emb_id,
    asset_id-1 as movie_emb_id
from users_ratings ur
join assets on ur.asset_id = assets.id and deleted_at is null
join users_app ua on ur.id=ua.id and ua.deleted_at is null
group by user_id, asset_id
union
select
    user_id as user_id ,
    asset_id as movie_id,
    5 rating, -- assuming that if its a favorite, then top score!
    UNIX_TIMESTAMP(f.created_at) as timestamp,
    user_id-1 as user_emb_id,
    asset_id-1 as movie_emb_id
from favourites f
join assets on f.asset_id = assets.id and deleted_at is  null
join users_app ua on f.id=ua.id and ua.deleted_at is null
group by user_id, asset_id
union
select
    user_id as user_id ,
    asset_id as movie_id,
    5 as rating, -- assuming that if its a favorite, then top score!
    UNIX_TIMESTAMP(b.created_at) as timestamp,
    user_id-1 as user_emb_id,
    asset_id-1 as movie_emb_id
from bookmarks b
join assets on b.asset_id = assets.id and deleted_at is null
join users_app ua on b.id=ua.id and ua.deleted_at is null
group by user_id, asset_id
;

select user_id, movie_id, sum(rating) as rating, UNIX_TIMESTAMP(now())
as timestamp, user_emb_id, movie_emb_id
from vw_recomendation_rattings group by user_id, movie_id;
```

Even though that the approach did not solve the problem of the sparsity (in fact it increased to 99.6%), the biggest amount of available data made the algorithm able to provide different and accurate results per each single user.

```
Number of users = 103008 | Number of movies = 1002
```

	movie_id	...	genres
170	25228	...	mysteries-abound|technology
167	25224	...	evil-ai-assistants|ai-robots|horror
165	25154	...	ai-robots|animated-universe|stranded-abandoned
163	25131	...	ai-robots|famous-faces
160	25127	...	mysteries-abound|famous-faces
6	24779	...	essential-action|ai-robots
157	25124	...	ai-robots|famous-faces
121	25032	...	technology|earthlings-love-drama|ai-robots
158	25125	...	ai-robots|animated-universe|earthlings-love-drama
162	25129	...	mysteries-abound|horror|aliens-monsters
156	25122	...	ai-robots|famous-faces
3	24775	...	deep-space-adventures|famous-faces
166	25212	...	mysteries-abound|time-travel
171	25256	...	aliens-monsters|mysteries-abound
174	25298	...	technology|a-childs-perspective
161	25128	...	dust-original|aliens-monsters
154	25120	...	ai-robots|famous-faces
30	24878	...	mysteries-abound|technology
124	25035	...	technology|ai-robots
155	25121	...	dystopian|famous-faces

Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.0513	1.0336	1.0607	1.0390	1.0786	1.0526	0.0161
MAE (testset)	0.2554	0.2540	0.2620	0.2548	0.2681	0.2589	0.0054
Fit time	18.52	17.76	18.34	17.75	18.22	18.12	0.31
Test time	0.64	0.89	0.72	0.61	0.61	0.69	0.10

3.2 The Model

As per the above description, and after evaluating a couple of different models, due to the type of data available, and since all the data was already categorized, the decision on using SVD was clear.

3.3 The Implementation

The project is, at is core a fork from this project. Due to easy manipulation of the data structure, the easiest way to get to the objective is almost everytime, the best way. With a couple of changes, on one hand due to the deprecation of some packages on the other hand, due to the data it self, the algoritm runs as expected.

Lets take a look at some of the key pieces and transformations operated by the code:

step 1 - load the data from csv into the pandas datasets:

```
# Reading ratings file
ratings = pd.read_csv('./ratings.csv', sep='\t', encoding='latin-1', usecols=['user_id',
'movie_id', 'rating', 'timestamp'])
# Reading users file
users = pd.read_csv('./users.csv', sep='\t', encoding='latin-1', usecols=['user_id',
'gender', 'zipcode', 'age_desc', 'occ_desc'])
# Reading movies file
movies = pd.read_csv('./movies.csv', sep='\t', encoding='latin-1', usecols=['movie_id',
'title', 'genres'])
```

step 2 -creating the new ratings pivot table, one row per user:

```
Ratings = ratings.pivot(index = 'user_id', columns ='movie_id', values =
'rating').fillna(0)
```

step 3 - normalize the data by each users mean and convert it from a dataframe to a numpy array.

```
R = Ratings.to_numpy()
user_ratings_mean = np.mean(R, axis = 1)
Ratings_demeaned = R - user_ratings_mean.reshape(-1, 1)
```

step 4 - create the SVD

```
U, sigma, Vt = svds(Ratings_demeaned, k = 50)
```

step 5 - getting the Σ matrix diagonal

```
sigma = np.diag(sigma)
```

step 6 - add the user means back to get the actual star ratings prediction.

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) +
user_ratings_mean.reshape(-1, 1)
```

step 7 - build a function to recommend movies for any user. Returns the list of movies the user has already rated.

```
preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
```

Step 8 - Write a function to return the movies with the highest predicted rating that the specified user hasn't already rated

```
def recommend_movies(predictions, userID, movies, original_ratings,
num_recommendations):
    # Get and sort the user's predictions
    user_row_number = userID - 1  # User ID starts at 1, not 0
    sorted_user_predictions = preds.iloc[user_row_number].sort_values(ascending=False)  #
User ID starts at 1
    # Get the user's data and merge in the movie information.
    user_data = original_ratings[original_ratings.user_id == (userID)]
    user_full = (user_data.merge(movies, how='left', left_on='movie_id',
right_on='movie_id').
                 sort_values(['rating'], ascending=False)
                 )
    print
    'User {0} has already rated {1} movies.'.format(userID, user_full.shape[0])
    print
    'Recommending highest {0} predicted ratings movies not already
rated.'.format(num_recommendations)
    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies[~movies['movie_id'].isin(user_full['movie_id'])].
                       merge(pd.DataFrame(sorted_user_predictions).reset_index(),
how='left',
                             left_on='movie_id',
                             right_on='movie_id').
                       rename(columns={user_row_number: 'Predictions'}).
                       sort_values('Predictions', ascending=False).
                       iloc[:num_recommendations, :-1]
                       )
    return user_full, recommendations
```

step 9 - Get the index of the user (918, in the example) on the dataset. On a real scenario, all values would be stored on a database for quick access. The user_id would come as a URL parameter or in the request header.

```
uindex=list(ratings['user_id'][ratings['user_id'] == 918].index)
```

step 10 - Get the list of the recommended movies for a user:

```
already_rated, predictions = recommend_movies(preds, uindex[0], movies, ratings, 20)
print (already_rated.head())
print (predictions)
```

3.4 The Evaluation

Model evaluation will be done using the Surprise library. This library is part of the scikit and provides various ready-to-use powerful prediction algorithms, (including SVD) to evaluate the RMSE (Root Mean Square Error)

```
# Load Reader library
reader = Reader()
# Load ratings dataset with Dataset library
data = Dataset.load_from_df(ratings[['user_id', 'movie_id', 'rating']], reader)
# Split the dataset for 5-fold evaluation
kf = KFold(n_splits=5)
kf.split(data)
```

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	Std
RMSE (testset)	1.0785	1.0691	1.0407	1.0418	1.0409	1.0542	0.0163
MAE (testset)	0.2704	0.2646	0.2551	0.2599	0.2543	0.2609	0.0060
Fit time	19.43	18.48	18.63	18.51	19.14	18.84	0.38
Test time	0.66	0.64	0.76	0.67	0.66	0.68	0.04

```
trainset = data.build_full_trainset()
svd.fit(trainset);
```

As we can see, as expected, the time needed to process the data is very high due the existing of near half a million users in the dataset. However, it alow us to get a TMSE of 1 which is a good number.

3.5 The gran finalle

We are now ready to bring everything together and get, not only the list of the 20 recommended movies but we can also get access to all the rattings that user made and the value for any specific movie that the algorithm estimated.

```
print (predictions)
print (ratings[ratings['user_id'] == 918])
print (svd.predict(918, 25937))
```

	movie_id	...	genres
120	25032	...	technology|earthlings-love-drama|ai-robots
3	24775	...	deep-space-adventures|famous-faces
159	25127	...	mysteries-abound|famous-faces
162	25131	...	ai-robots|famous-faces
6	24779	...	essential-action|ai-robots
155	25122	...	ai-robots|famous-faces
19	24793	...	aliens|creature-feature|drama|character-driven...
156	25124	...	ai-robots|famous-faces
170	25256	...	aliens-monsters|mysteries-abound
157	25125	...	ai-robots|animated-universe|earthlings-love-drama
153	25120	...	ai-robots|famous-faces
287	25846	...	fantasy|famous-faces

	user_id	movie_id	rating	timestamp
1265	918	24783	5	1641125086
1266	918	24810	5	1641125086
1267	918	24904	5	1641125086
1268	918	25031	5	1641125086
1269	918	25033	5	1641125086
...	...	...	...	...
1321	918	26100	5	1641125086
1322	918	26103	5	1641125086
1323	918	26105	5	1641125086
1324	918	26126	5	1641125086
1325	918	26270	5	1641125086

```
user: 918       item: 25937     r_ui = None   est = 4.91   {'was_impossible': False}
```

5 CONCLUSIONS

Overall, this was a very exciting project to work on. Despite some challenges and growth pains, I can honestly consider that most of the objectives defined for the project were accomplished. There were some things that were unexpected, and were difficult to achieve by someone that is not an experienced developer in these modern languages (my background as a developer is mainly C, Delphy as SQL). This fact also explains why it's much easier from my perspective to manipulate the data while still on the database than to work with it inside the python environment.

As for the future, I gained a good high level view of what are the possibilities, which are the best algorithms for each case scenario and, most of all, the understanding that there are no 1 perfect solutions. Each case needs to be evaluated individually and, many times, the available data has a much bigger influence on the choice of the algorithm than the results we want to achieve.

Thanks for giving me the opportunity to produce this work and enrich my knowledge on an area that is of key importance for my professional work.

REFERENCES

[1] Mazhar Javed Awan , Rafia Asad Khan , Haitham Nobanee, , Awais Yasin, Syed Muhammad Anwar, Usman Naseem and Vishwa Pratap Singh - A Recommendation Engine for Predicting Movie Ratings Using a Big Data Approach at https://www.mdpi.com/2079-9292/10/10/1215

[2] JAKOB IVARSSON, MATHIAS LINDGREN - Movie recommendations using matrix factorization at https://www.diva-portal.org/smash/get/diva2:927190/FULLTEXT01.pdf

[3] https://nextjournal.com/berwa/surprise-movie-recommender-system-example

[4] https://mxnet.apache.org/

[5] https://www.youtube.com/watch?v=VwVg9jCtqaU

[6] CARLOS A. GOMEZ-URIBE and NEIL HUNT, Netflix, Inc. - The Netflix Recommender System: Algorithms, Business Value, and Innovation at https://dl.acm.org/doi/pdf/10.1145/2843948

[7] David Chong, Deep dive into Netflix recommender system at https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48

[8] Libby Plummer, How do Netflixs algorithms work At https://www.wired.co.uk/article/how-do-netflixs-algorithms-work-machine-learning-helps-to-predict-what-viewers-will-like

[9] Ada Tzereme, How to build a content recommendation engine with snowpolow at https://snowplowanalytics.com/blog/2020/10/26/how-to-build-a-content-recommendation-engine-with-snowplow/

[10] Amhmud Ridawn, Predicting likes inside a simple recommendation engine at https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine

[11] Pulkit Sharma, Comprehensive Guide to build a Recommendation Engine from scratch (in Python) At https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

[12] James Lee, The 4 recommentation engines that can predict your movie tastes at https://towardsdatascience.com/the-4-recommendation-engines-that-can-predict-your-movie-tastes-109dc4e10c52

[13] https://grouplens.org/

[14] Kurtis Pykes – 3 Approaches to build a recommendation System at https://towardsdatascience.com/3-approaches-to-build-a-recommendation-system-ce6a7a404576

[15] https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html

[16] Jan Teichmann, How to build an advanced recommendation engine at https://towardsdatascience.com/advanced-use-cases-for-recommendation-engines-4a420b14ab4e

[17] Jan Teichmann, How to build a recommention engine quick and simple at https://towardsdatascience.com/how-to-build-a-recommendation-engine-quick-and-simple-aec8c71a823e

[18] Spring Board India, How Netflix Recommenation engine Works at https://medium.com/@springboard_ind/how-netflixs-recommendation-engine-works-bd1ee381bf81

[19] Saurav Suman, Building an intelligent recommendation engine with collaborative filtering at https://www.velotio.com/engineering-blog/collaborative-recommendation-system-approaches-challenges-implementation

[20] Aditya Sharma, Beginner tutorial: Recommender systems in Python at https://www.datacamp.com/community/tutorials/recommender-systems-python

[21] Github repo from khanhnamle1994 https://github.com/khanhnamle1994/movielens

[22] Github repo from khanhnamle1994 https://github.com/khanhnamle1994/movielens/blob/master/SVD_Model.ipynb

[23] https://surprise.readthedocs.io/en/stable/getting_started.html

[24] Roberto Alain Berwa, Surprise: Movie recommenr System Example at https://nextjournal.com/berwa/surprise-movie-recommender-system-example

[25] TURNER, Ryan – Python Machine Learning – '

[26] https://en.wikipedia.org/wiki/Netflix_Prize