Machine Learning Engineer Nanodegree

Capstone Project

Gabriel Martelloti

April 10th, 2019

I. Definition

Project Overview

If anyone have already searched for info and rating about an Anime on the internet, he was probably sent to the MyAnimeList website. It is like the IMDB of japanese content, and has a lot of different ways for you to find something suitable to you as "Top Airing animes", "Top Upcoming animes" and so on. The score of a certain anime or manga are actually the global mean of all the ratings given by the users. Rates go from 0 to 10, and whenever you create your account you can score pretty much anything you want.

Ever since I've learned how Spotify created and developed their awesome recommendation system I got more and more interested about how all of this stuff works in practice and started to be more curious about it. A while ago, I was looking for something to watch and thought "Hey, maybe I can go to MyAnimeList, give my scores on previous watched animes and maybe find something good recommended to me, kind of like Netflix and Spotify do". But the main issue is that, even though the MyAnimeList portal is huge and they store all of the users’ scores (which is a lot), they don’t have a user-based or an item-based recommender system. So, in this project I created this user-based and an "extra" item-based recommender system for their website.

Project Statement

The goal is to develop the best suitable collaborative filtering user-based recommender system for the MyAnimeList portal and so we can have an idea of how the system is performing. Here are the following steps:

1. Download the MyAnimeList dataset on Kaggle
2. Exploring, cleaning and munging the data so it is suitable for training
3. Training and testing the available user-based collaborative filtering methods and choosing the best performing one
4. Choosing randomly any item-based recommender for testing purposes
5. Creating a function that brings the top-k recommendations for a specific user in the dataset or the top-k neighbors of a specific item

Metrics

Two metrics will be taken in to consideration for evaluating the performance of the algorithms:

$$\mathrm{RMSE} = \sqrt{\sum \frac{(y_{pred} - y_{ref})^2}{N}}$$

The **RMSE** is very useful to acknowledge if a system is doing good or badly considering the variance between the predicted values and the real values. It subtracts the difference between them, squares the result (this is done for two reasons, the first one is that it wants to only have positive differences and the other is giving more importance to bigger differences), calculates the mean and then root them because of the previous squaring.

Although this metric is awesome for regression algorithms such as linear regression, it has flaws for recommendation systems, since we are not interested in the difference between all values but between the highest scored ones. Because of this, the critical metric for the system will be the precision@k.

$$Precision = \frac{tp}{tp + fp}$$

Precision is a well known metric, mainly for categorical algorithms that can't be evaluated through accuracy (an example of such a case would be a machine learning algorithm trying to prevent credit card fraud. If the algorithms simply let all transactions pass it will have an accuracy of 99%, but this doesn't really means it is well trained). Continuing the case before, the precision would measure the proportion of the the transactions flagged as fraud that are truly frauds between all of the transactions flagged as frauds. But as mentioned in the RMSE, recommendation systems are mostly interested in the highest scores, so an alternative is utilized.

Precision@K has the same logic as it's antecessor but focus only in recommended items. It calculates **how many of the recommended items are indeed relevant**. So, it is the proportion of all the recommended and relevant items by all the recommended **k** items (averaged by all users).

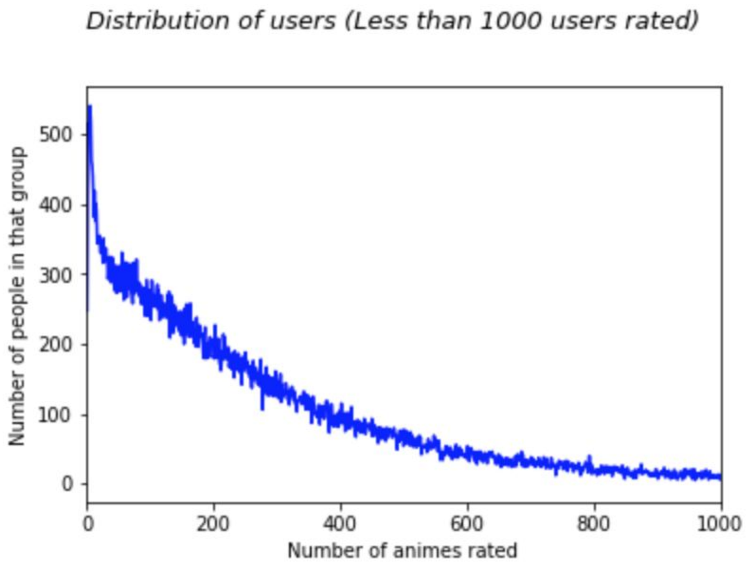
II. Analysis

Data Exploration

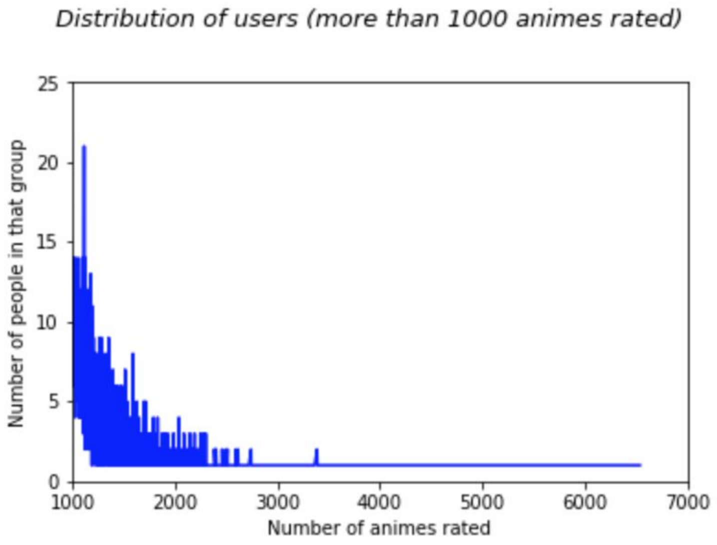
For the dataset, we will use the MyAnimeList dataset posted on Kaggle. The datasets are mainly divided in three. One with info about the MyAnimeList customers such as location, gender, birth date and so on. Another one for info about the anime, which brings data like the title of the show, what genre is it inside and even the opening themes. The last one (and also the biggest one) is all the scores that a certain user had with a show (that have 31M rows of interactions). To make the cleaning data process easier, only the CSVs that do not contain any null values (which have the cleaned suffix on them) will be used. Since only collaborative filtering techniques are applied, the User info will be irrelevant for the training and results (but could be used in the future to understand the demographics of a certain recommendation).

Exploratory Visualization

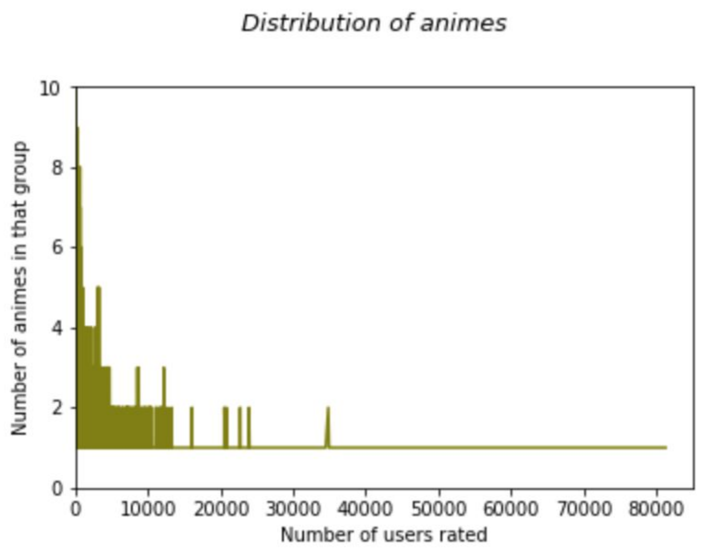
The following explorations were made so that the we could understand how the common MyAnimeList user interacts with shows he likes and dislikes and already planning how the system will address the *user cold start problem* (will talk about it later).



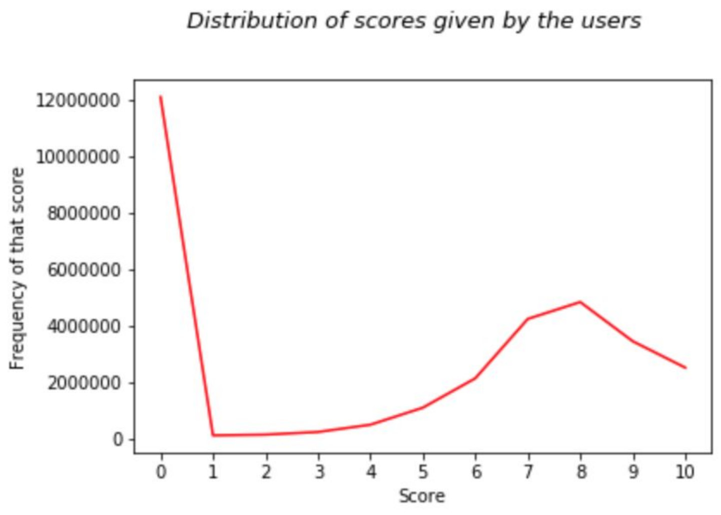
As already expected, as we increase the "Number of animes rated" info by user, the number of customers decrease.



But even though the number tends to drop, there are a lot of hard users that scored over 1000 animes, which leads to this huge amount of data.



For the number of times a specific anime was rated, the curve is large, counting with shows that have over 80k ratings from users.



This one chart is quite interesting. It shows that the most frequent score (which is almost equal to all other scores summed up) is the "0". There is also a curve being formed at the score "7", which could lead us to think that if a user likes something he tends to score from 7 to 10.

Algorithms and Techniques

First of all, every algorithm tested came from the Surprise lib, which is a SciPy Toolkit designed for bringing lots of different collaborative filtering algorithms in to the same environment. Here are the ones that were tested:

- BaselineOnly
- KNNWithMeans
- SlopeOne
- KNNBaseline
- KNNWithZScore
- CoClustering
- NMF
- KNNBasic
- SVD
- NormalPredictor

The entire training and testing process ran at a local machine with 8 GB of Memory RAM.

Benchmark

The primary benchmark for this project was this article from Kaggle. He uses the Recall@5 to evaluate his algorithms (which is an alternative for the Precision@k metric) and despite that other approaches were compared (Content, Collaborative, Popularity and Hybrid), the same evaluation can be done for only Collaborative algorithms.

III. Methodology

Data Preprocessing

Although the cleaned datasets were used (which didn't have any null value in their fields) some preparation was needed for the algorithms to run. The first problem addressed was the size of the dataset containing all ratings, that had over 31M rows. Since the that would be too costly for a common CPU, only 1% of the customers were sampled. The reasoning for sampling customers and not records is that it is preferable to just assume that the mining was done with less customers than losing information about specific customers.

Another argument for sampling customers and not records comes with our next preparation: the *user cold start problem*. Since recommender systems algorithms have a strong characteristic of predicting data individually, they need to have good information about the customers being evaluated. An example: Even though 1000 records about each of 1000 customers would result in the same amount of data that 5 records about each of 200000 customers, the first case would be the most valuable for a recommender system, since you can give really relevant recommendations for your customers since you know about them a lot more. To solve this issue, another step is cutting any anime that was rated less than 10 times and any user that rated less than 10 animes.

Implementation

The implementation step is quite simple: We split our preprocessed data into a **Training** dataset (75% of the records) and a **Test** dataset (25% of the records). The model learns with all the data given by the training dataset and then it tries to predict every records from the test dataset to avoid any overfitting (prediction only working for the data you are working in) in the evaluation.

Then a list containing all models that will be trained and tested is created so each one of them can be ran together. In the function created to run all algorithms contained in this list, besides the most important metrics as RMSE and Precision@K, time of executing was also recorded.

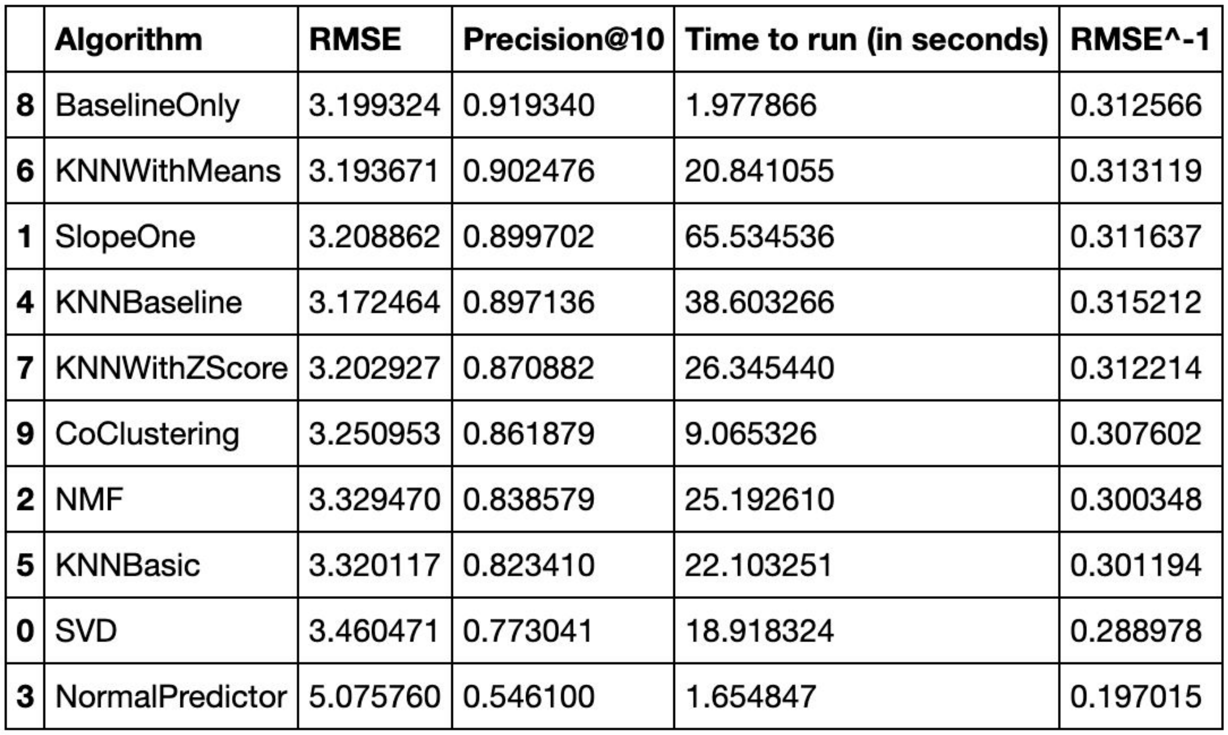
IV. Results

Model Evaluation and Validation

The reasons for choosing the best algorithm were a mix of:

- Precision@10
- RMSE
- Time needed for it to run with a robust dataset

- Logic behind the algorithm



	Algorithm	RMSE	Precision@10	Time to run (in seconds)	RMSE^-1
8	BaselineOnly	3.199324	0.919340	1.977866	0.312566
6	KNNWithMeans	3.193671	0.902476	20.841055	0.313119
1	SlopeOne	3.208862	0.899702	65.534536	0.311637
4	KNNBaseline	3.172464	0.897136	38.603266	0.315212
7	KNNWithZScore	3.202927	0.870882	26.345440	0.312214
9	CoClustering	3.250953	0.861879	9.065326	0.307602
2	NMF	3.329470	0.838579	25.192610	0.300348
5	KNNBasic	3.320117	0.823410	22.103251	0.301194
0	SVD	3.460471	0.773041	18.918324	0.288978
3	NormalPredictor	5.075760	0.546100	1.654847	0.197015

Considering all of the algorithms, the chosen was the **BaselineOnly** and the low time needed to run it comes from its simple logic. The model creates a bias for each user and each item based on the average of every score and then calculates a forecast score. An example: Let's say the model wants to predict the rating of the anime Naruto for the user named Gary. The average rating of all animes is 5.7, but since Naruto is a good show, it tends to have 2.3 score over the other shows. On the other hand Gary is a demanding customer that tends to score 1.3 points below the average of customers, so his recommendation for Naruto would be: 5.7 + (2.3) + (-1.3) = 6.7.

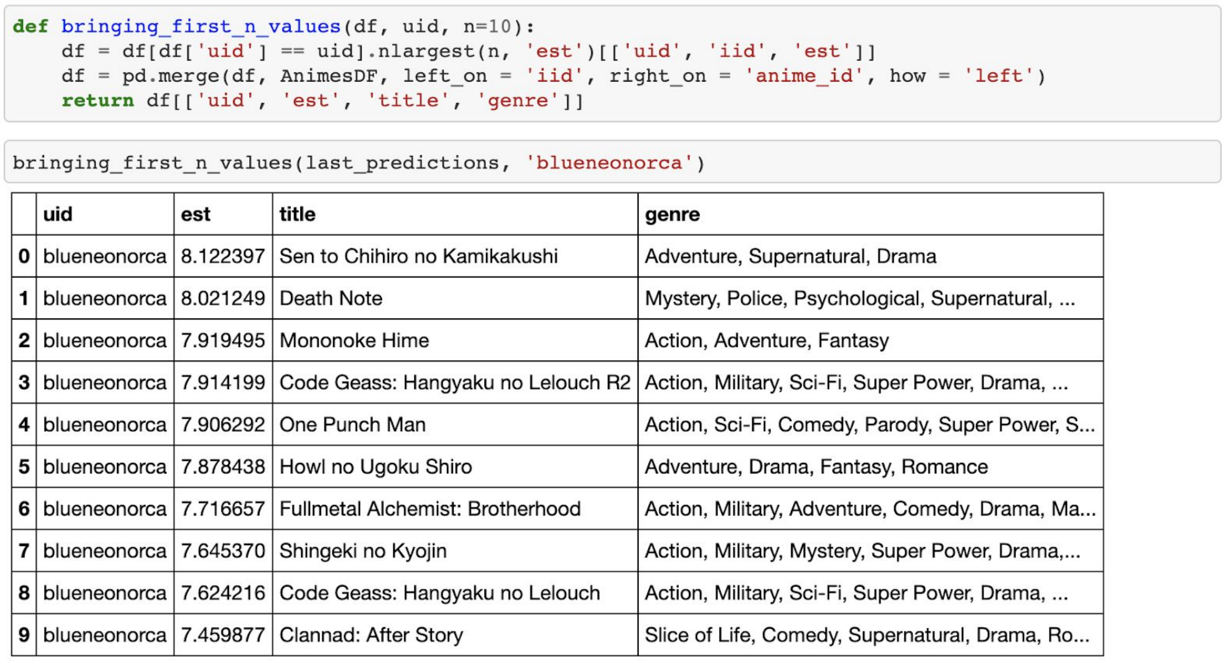
Justification

It is really hard to compare the scores of this model with any other model, since each kind of application of recommendation systems are completely different from each other, as the profile of the users will have a big impact (as an example, a MyAnimeList user, in average, rates a lot more animes than a Netflix user). Even though the benchmark application is the most similar to this one, this article is one of the fewest that utilizes Precision@K as its metric (the other issue with comparing recommendation systems is the lack of information about models, since it is a really new practice). And if you match the results of this model with the already mentioned one, it has overall a lot better results of precision.

V. Conclusion

Free-Form Visualization

For this topic, I thought of showing different recommendations for either used based and item based models. All of the system was built as a user based perspective, as it tries to recommend animes based at the profile of the user, but I thought it would be cool to create a MVP model of an item based recommender so that, empirically, we could test if our recommendations make sense and if we can extract any kind of value from it.



```
def bringing_first_n_values(df, uid, n=10):
    df = df[df['uid'] == uid].nlargest(n, 'est')[['uid', 'iid', 'est']]
    df = pd.merge(df, AnimesDF, left_on = 'iid', right_on = 'anime_id', how = 'left')
    return df[['uid', 'est', 'title', 'genre']]
```

```
bringing_first_n_values(last_predictions, 'blueneonorca')
```

	uid	est	title	genre
0	blueneonorca	8.122397	Sen to Chihiro no Kamikakushi	Adventure, Supernatural, Drama
1	blueneonorca	8.021249	Death Note	Mystery, Police, Psychological, Supernatural, ...
2	blueneonorca	7.919495	Mononoke Hime	Action, Adventure, Fantasy
3	blueneonorca	7.914199	Code Geass: Hangyaku no Lelouch R2	Action, Military, Sci-Fi, Super Power, Drama, ...
4	blueneonorca	7.906292	One Punch Man	Action, Sci-Fi, Comedy, Parody, Super Power, S...
5	blueneonorca	7.878438	Howl no Ugoku Shiro	Adventure, Drama, Fantasy, Romance
6	blueneonorca	7.716657	Fullmetal Alchemist: Brotherhood	Action, Military, Adventure, Comedy, Drama, Ma...
7	blueneonorca	7.645370	Shingeki no Kyojin	Action, Military, Mystery, Super Power, Drama,...
8	blueneonorca	7.624216	Code Geass: Hangyaku no Lelouch	Action, Military, Sci-Fi, Super Power, Drama, ...
9	blueneonorca	7.459877	Clannad: After Story	Slice of Life, Comedy, Supernatural, Drama, Ro...

At first, for the user based recommender, I've built a function where you type the username and it displays the 10 most valuable recommendations ordered by the estimated value of the anime.

```
In [29]: get_item_recommendations('Dragon Ball', k=30)
```

Out[29]:

	Anime_ID	title	genre
0	813	Dragon Ball Z	Action, Adventure, Comedy, Fantasy, Martial Ar...
1	225	Dragon Ball GT	Action, Adventure, Comedy, Fantasy, Magic, Sci...
2	528	Pokemon: Mewtwo no Gyakushuu	Action, Adventure, Comedy, Drama, Fantasy, Kids
3	527	Pokemon	Action, Adventure, Comedy, Kids, Fantasy
4	392	Yuu☆Yuu☆Hakusho	Action, Comedy, Demons, Supernatural, Martial ...
5	552	Digimon Adventure	Action, Adventure, Comedy, Fantasy, Kids
6	986	Dragon Ball Z Special 1: Tatta Hitori no Saish...	Adventure, Comedy, Fantasy, Sci-Fi, Shounen
7	2116	Captain Tsubasa	Action, Shounen, Sports
8	1674	Captain Tsubasa J	Action, Shounen, Sports
9	249	InuYasha	Action, Adventure, Comedy, Historical, Demons,...
10	901	Dragon Ball Z Movie 08: Moetsukiro!! Nessen, R...	Action, Sci-Fi, Adventure, Comedy, Fantasy, Sh...
11	896	Dragon Ball Z Movie 03: Chikyuu Marugoto Chouk...	Action, Sci-Fi, Adventure, Comedy, Fantasy, Sh...
12	550	Yu☆Gi☆Oh!	Action, Game, Comedy, Fantasy, Shounen
13	985	Dragon Ball Z Special 2: Zetsubou e no Hankou!...	Adventure, Drama, Fantasy, Shounen
14	1117	Pokemon: Maboroshi no Pokemon Lugia Bakutan	Adventure, Comedy, Kids, Drama, Fantasy
15	47	Akira	Action, Military, Sci-Fi, Adventure, Horror, S...

This is the function built for an item-based perspective, where you input the name of the anime and it represents the most similar shows. As the most relevant animes from Dragon Ball in this example are Dragon Ball Z and Dragon Ball GT, which are sequels of the inputted show, I think it did a great job recommending. But as we go to the next item we face some recommendations as Pokemon and Digimon, and if you watched them you know they are not that similar. So why would it be displayed, then? Thinking about this question, I remembered the time when I was a child. I used to wake up and turn on the TV as soon as possible so I could watch all my favourite shows at the morning, and I recalled that all these shows were broadcasted one followed by the other. So, when you want to recommend something based on an anime or a movie you don't have to think only about the genre that they belong to, but if they happen to be from the same generation, as this model shows that there is a pretty high chance that it interferes at the last score.

Reflection

Summing the project up: The ideia was to create an intelligent and simple way of recommending animes to a user of the portal MyAnimeList based on his previous scores of other shows. The process can be described as:

- Understand and gather the already mined dataset from Kaggle
- Do some exploration so the profile of the MyAnimeList users could be understood
- Make all the data cleaning and wrangling needed so the collaborative filtering algorithms could be tested
- Train and test all the available algorithms at the chosen lib and choose the best performing one
- Create an easy way of checking the recommendations for each user and even develop a MVP of an item-based recommender

Probably the hardest part for me through this project and all of the previous projects from this nanodegree was my lack of previous experience with Python. There were a lot of documentation and other kinds of sources that I couldn't simply understand and that made me stop with the course and focus simply on improving my language skills. The other difficulty, but now specific with this project, was to work with a type of machine learning modelling and with a lib that have a lot less documentation and benchmarking comparing to other kinds of applications, as Supervised and Unsupervised Learning. Recommender Systems are still little in terms of open-source applications, but with its powerful applications it has a large potential of growing in the next few years.

Improvement

There is still room for improvement in this model, but the most importants would be:

- Test not only the algorithms but do some grid search and evaluate their parameters
- Not only focusing on collaborative filtering approaches but on content filtering, where you would specifically recommend shows based on the genre and so on
- Take some more time to improve the item based recommender, as it could be another cool feature, to most importantly create value for the decision makers in the MyAnimeList
- Use a deep learning approach for recommendation, as Tensorflow is proving to be a really good alternative to solving this kind of problem

Sources

Explanations of Precision and Recall

Explanation of RMSE

The paper for the BaselineOnly method

Surprise Lib Documentation

Collaborative filtering with surprise application