Movie Recommendation System

Özgün Yargı

Mutlu Soruklu

1. Introduction

Recommendation systems are methods for data extraction based on the past activities of users in a system. Depending on the likes and dislikes of users or customers, these systems recommend items to users with the hope that the user will buy this recommended item. Movie recommendation systems are a subset of the recommendation systems. These systems use ratings of the users for the movies as their input and output a list of recommended movies. The most commonly used approach to this problem is collaborative filtering. This algorithm is being used by many popular streaming companies like Netflix and Amazon in order to attract their customers. In this project, we examine the phenomenon of movie recommendation. We will try to create a movie recommendation system using a collaborative filtering algorithm. However, we will update this algorithm by using visual features extracted from movies with the aim of getting similarities between movies.

2. Methods

2.1 Getting the Data

The datasets that we will use in this project are from different sources. List of the movies that we used is from Kaggle website that contains 1,000 most popular movies on IMDb in years between 2006 to 2016. In default, this dataset has 12 features. Ratings for those movies were taken from MovieLens dataset. When we applied visualization to the dataset, we saw some correlation between features. To ensure this, we also wanted to create a correlation matrix and saw that there is a correlation between metascore and rating. This makes sense because both of

them actually show a score about how much do people love that movie. As the score increases, it indicates that that movie is loved more. Since this is true for both metascore and rating, they actually point out the same so we may remove one of those features while constructing a movie recommendation system.

We also wanted to use movie reviews for an additional feature. However, that kind of dataset does not exist and because of this, we decided to pull that data via IMDB by ourselves. To do this, we used Selenium, Requests and IMDbPY libraries. We managed to pull related reviews by using these libraries. We pulled 25 reviews for each movie and saved them as JSON files to use later.

Additionally, we also want to use visual content for our recommendation system. To do this, we also need to scrape IMDB to pull visual content for each movie. To do this, we first need to reach the IMDB ids of movies. We filtered the related movies from MovieLens dataset and took those IMDB ids. After we have taken the ids, we started to scarpe the posters from the urls and save those poster in our local drive. We collected 993 posters. Generally, taken images were in size 100x105 pixel. You may see an example of collected posters in Figure 1.

a.Harry Potter

b.Transformers

c.Hairspray

Figure 1. Movie Posters Scraped from IMDB

2.2 Creating Similarity Matrix based on Posters

We believe that movie posters contain necessary information for a movie whether it can be suggested or not. If you check posters, you are going to see that there is actually a similarity between the movies that are related to each other. For example, if you look at the posters of movie series, you are going to see that the placements of each poster share similarity between each other. If we somehow can detect these similarities, then we can use them to create a better recommendation system.

To do this task, we used deep convolutional neural network VGG16 to extract CNN features. CNN features are basically the content of the image; mainly explaining what the image is. It is a pre-trained model on Image-Net which is a database used for image classification. But in this task, we actually do not want to do classification but we only want to extract the features of the image. Because of this we remove the output layer which is designed for image classification. This technique is widely used to extract image features (Chen, et al. 2019). You can see the architecture of VGG16 in Figure 2.

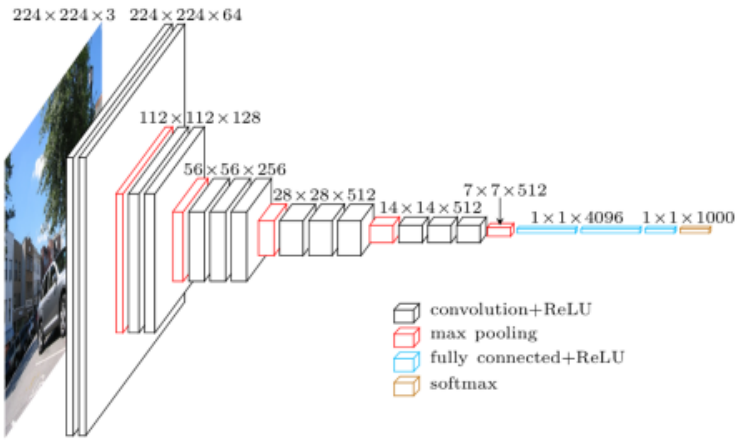


Figure 2. Architecture of VGG16

VGG16 requires a strict input size which is 224x224 with 3 channels. Because of this, we resized our images to 224x224 and then we put them into the architecture one by one. As the output, we took a vector with size 4096 which carries shrinked information about the image. The distribution of values inside of the output vector can be seen in Figure 3.

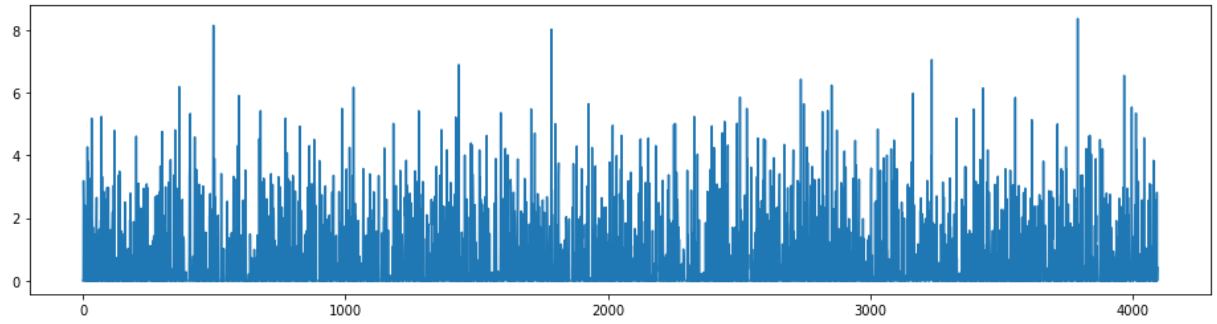


Figure 3. Weights of an image

We applied the same process for all images. After we have collected all the output vectors for all images, since the vector size is too high, we decided to use PCA to reduce the dimensionality of vectors. We took the most significant 300 dimension to represent each image.

Now, we are ready to calculate the similarities for each image. We decided to use the cosine distance metric to calculate the similarity between each image. Cosine distance is a widely used metric to figure out the similarities between two vectors by using the Euclidean dot product. Formula of the cosine distance is given below:

$$\frac{A.B}{||A||.||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

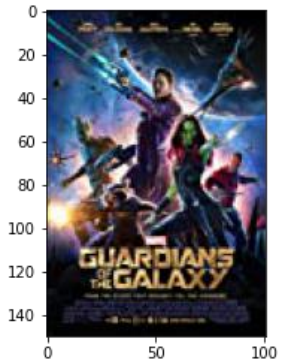
This gives us values from an interval between -1 to 1.However, we are looking for values between 0 and 1. Because of this, we normalized those distances and ranged them between 0 and

After that, since they were the distances, we distract them from 1 to create a similarity matrix. Some part of the similarity matrix can be seen in Figure 4.

	2015381	1446714	4972582	3470600	1386697
2015381	1.000000	0.278903	0.248301	0.388134	0.439046
1446714	0.278903	1.000000	0.341925	0.261093	0.188453
4972582	0.248301	0.341925	1.000000	0.201871	0.198294
3470600	0.388134	0.261093	0.201871	1.000000	0.248812
1386697	0.439046	0.188453	0.198294	0.248812	1.000000

Figure 4. Similarity matrix shaped with images

After we calculated the similarity matrix, we wanted to check its correctness. To do this, we took one movie poster from the dataset and tried to find the most similar posters with the reference poster by using this similarity matrix. The Results can be seen below.



a. Reference Image

b. Most Similar images with the reference image

Figure 5. Output results of the experiment.

As it can be seen from Figure 5. Found similarity matrix can be used to recommend people movies since the outputs of the experiment is reasonable. For the Guardians of the Galaxy movie, the Avengers was found as the most similar movie which is highly accurate. Also, the movie Transformers and Tarzan can also be recommended for this movie style. However, the movie Leap does not look similar so we can say that this similarity matrix is not fully accurate. However, it can still be used to improve further recommendation systems.

2.3 Applying Collaborative Filtering

There are two types of collaborative filtering methods. One is called the user-user algorithm and the other is called the item-item algorithm. While the main purpose of the user-user algorithm is to find similar users for a specific user, the latter one tries to find similar items. Considering the complexity of users, it is fair to say that the item based strategy is more advantageous since it is easier to find similar items rather than finding similar users. However, our dataset consists of 993 movies. Therefore, it makes sense to assume that the User-User algorithm would work more efficiently since the model will have a much bigger source to extract information from. With this in mind, we implemented both methods in which we try to find similar movies and similar users. In item-item models we find predictions with the following formula (Amer-Yahia et al. 2009). ItemSim(i,j) refers to the similarity score between item i and j. It is actually the weighted average of ratings with similarities.

$$\texttt{relevance}(u,i) = \Sigma_{i' \in \mathcal{I}} \texttt{ItemSim}(i,i') \times \texttt{rating}(u,i')$$

We implemented this algorithm by using the Lenskit library in Python. In this library, there is a function called Recommender, and we passed the item-item keyword as a parameter to this function and trained the model with the rating dataset that we get from Movielens. After training, we input a rating table of a user and get the recommendations from the model. Following figure shows the input and output of the model for one trial.

	movieId	title	genres	rating
2	59615	Indiana Jones and the	Action|Adventure	4
3	128512	Paper Towns (2015)	Drama|Mystery|I	5
4	142420	High Rise (2015)	Action|Drama|Sc	5
5	63113	Quantum of Solace (2	Action|Adventure	4
6	169814	The Assignment (2016	Action|Thriller	5
7	76093	How to Train Your Dra	Adventure|Anima	4
8	45730	Lady in the Water (20	Drama|Fantasy|M	5
9	48043	Fountain, The (2006)	Drama|Fantasy|F	5
10	87876	Cars 2 (2011)	Adventure|Anima	4

Figure 2.2.1: Input to the model, table represents the snapshot of ratings of a user to some movies

```
RECOMMENDED FOR Mutlu:
```

	genres	title
0	Drama	Custody (2016)
1	Crime|Drama|Thriller	Departed, The (2006)
2	Drama|Mystery|Sci-Fi|Thriller	Prestige, The (2006)
3	Action|Adventure|Crime|IMAX	Dark Knight Rises, The (2012)
4	Drama	Whiplash (2014)
5	Comedy|Drama	Intouchables (2011)

Figure 2.2.2: Output of the model with the given input

Same library is used to train a User-User model. We used a test strategy for our data to get RMSE scores. The strategy is based on trying to find the rating of each user to a movie. We first eliminated one of the ratings for each user for a movie, Then we predicted the eliminated ratings. Results of these models will be included in the results section.

2.4 Combining Similarity Scores From Images and Pearson Similarities

In this part, we implemented a hybrid method to find similarity scores between movies. First side of the calculation comes from the image similarity extraction method. We got a square matrix representing similarities between movies 1 being the highest and 0 being the lowest. Another side of the calculation comes calculating similarities with Pearson correlation. First we find vectors for each movie. Length of these vectors were the total number of movies. Then we applied a SVD based transformer to reduce the dimension of each vector. After this process we got vectors with lengths 12 for each movie. At this stage we applied the Pearson correlation method to these vectors to find similarity scores between movies. At the end, we obtained a square similarity matrix. Using this similarity matrix and the matrix came from image similarity matrix, we combined them together to get a single similarity matrix. Then for a user, we used weighted averages of top 5 similar movies to the movie that we are trying to predict, weights being the rating that this specific user gave to movies. Weighted averages stored as ratings. Then we used the RMSE metric to evaluate our method. Combination process is presented in the figure below.

```
from sklearn import preprocessing
temp = pd.DataFrame.from_dict(imageDistances)


x = temp.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
x_scaled = 1-x_scaled
temp = pd.DataFrame(x_scaled,index= temp.index, columns = temp.columns)
imageDistances = temp.to_dict()
```

```
final = dict()
weight1 = 0.4
weight2 = 0.6
for movie1,val in sent_dict.items():
  ghost = dict()
  for movie2 in val.keys():
    scoreFromImg = imageDistances[movie1][movie2]
    scoreFromSVD = sent_dict[movie1][movie2]
    new_score = scoreFromImg*weight1 + scoreFromSVD*weight2
    ghost[movie2] = new_score
  final[movie1] = ghost
```

```
df_final = pd.DataFrame.from_dict(final)
df_final
```

```
df_final = pd.DataFrame.from_dict(final)
df_final
```

	1000774	1001508	1001526	1007028	1010048
1000774	1.000000	0.714995	0.363791	0.487493	0.605874
1001508	0.693865	1.000000	0.364249	0.570102	0.485955
1001526	0.365282	0.390671	1.000000	0.433150	0.383114
1007028	0.478532	0.583181	0.423630	1.000000	0.429261
1010048	0.599014	0.498985	0.370203	0.426181	1.000000

Combination of two similarity matrix

3. Results

We present the evaluation metrics' results in the table below.

MovieID- Prediction -Real

```
compare
'382': (4.424084572257647, 5.0),
'402': (3.952596291137564, 4.0),
'406': (4.765368408135878, 3.5),
'469': (2.8891906671908023, 4.5),
'489': (3.94684663108383, 4.0),
'494': (2.5007220297688337, 4.0),
'527': (3.58552412841486, 4.0),
'541': (3.8139064496211073, 5.0),
'548': (2.875346202942511, 4.0),
```

Our Model

MovieID- Prediction -Real

```
compare3
'25512': (2.8719330884163634, 2.5),
'25543': (3.9669041622910624, 2.5),
'25553': (3.9232348355613804, 3.0),
'25554': (4.574082690196178, 4.0),
'25570': (3.57368600494159, 4.0),
'25586': (4.091490754955938, 4.0),
'25596': (4.057051437352147, 4.0),
'25627': (2.9962388096989945, 3.0),
'25652': (3.6371773797984184, 4.0),
'25693': (4.052963504035711, 5.0),
'25698': (4.30633432987785, 3.5),
'25716': (4.6443608869126205, 4.0),
```

User-User Collaborative Filtering

MovieID- Prediction -Real

```
compare2
{'3': (4.104955671963698, 4.0),
 '4': (3.1336201786107187, 3.0),
 '181': (2.0188136182207685, 2.0),
 '243': (1.8623744892477019, 2.0),
 '321': (3.1339636100931134, 3.5),
 '382': (4.7853094207219256, 5.0),
 '402': (4.5120267895701, 4.0),
 '469': (3.5077624846547764, 4.5),
 '494': (3.745468779073922, 4.0),
 '527': (3.490775468601935, 4.0),
 '541': (4.566671827299137, 5.0),
 '548': (4.039738111981221, 4.0),
```

Item-Item Collaborative Filtering

	RMSE
Img+SVD	0.981948
Item-Item Collaborative	0.730899
User-User Collaborative	0.589768

The upper part of the figure shows samples from our predictions with different models. And the bottom table shows the overall RMSE scores of each model. As predicted before, the User-User algorithm outperformed the Item-Item model. We can see that our hybrid model needs to be improved in order to lower its RMSE score.

4. Discussion

When we look at the results, we have seen that User to User has the best performance and second comes Item to Item and third comes a recommendation system that was built on SVD and Image similarity. This may have a couple of reasons. One of them is that we used such algorithms that require hyperparameters (PCA, SVD, weight adjustment between SVD and image matrix etc.). Hyperparameter tuning plays a crucial role in optimization and so far, we have not been able to tune these hyperparameters. Because of this, our model may perform worse than the other two methods.

Another problem may be the image dataset. As the resolution of an image increases, VGG16 becomes more efficient which leads us to create a better similarity matrix upon images. However, since our image sizes are small (105X100), the VGG16 model may not extract great features from images. One other thing is that, to reduce the dimensionality of the vector, we used PCA that shrinked each feature vector to size 300. Rather than using the combination of VGG16 and PCA, we could have used Autoencoders to create an unsupervised feature extraction model which will be able to learn the features upon a given dataset. So, it will be much more efficient while extracting features as well as reducing dimensionality. However, in the end, it will be expensive in terms of time.

5. Future Work

For future work, rather than using VGG16 and PCA together, we may implement an autoencoder that will learn the dataset and extract more meaningful features. One other thing will be optimizing the hyperparameters and also trying to scrape the web for collecting higher resolution images. If it succeeds, this project may get published in a global environment so that anyone can try it.

References

Amer-Yahia, Sihem et al. 2009. "Group Recommendation: Semantics and Efficiency." *Proceedings of the VLDB Endowment* 2(1): 754–65. https://dl.acm.org/doi/10.14778/1687627.1687713 (March 15, 2021).

Chen, Xiaojie et al. 2019. "Exploiting Aesthetic Features in Visual Contents for Movie Recommendation." *IEEE Access* 7: 49813–21. https://ieeexplore.ieee.org/document/8688388/ (March 15, 2021).

Diao, Qiming et al. 2014. "Jointly Modeling Aspects, Ratings and Sentiments for Movie Jointly Modeling Aspects, Ratings and Sentiments for Movie Recommendation (JMARS) Recommendation (JMARS) Citation Citation Jointly Modeling Aspects, Ratings and Sentiments for Movie Recommendation (JMARS)." : 193–202. http://dx.doi.org/10.1145/2623330.2623758. (March 15, 2021).

Ekstrand, Michael D., F. Maxwell Harper, Martijn C. Willemsen, and Joseph A. Konstan. 2014. "User Perception of Differences in Recommender Algorithms." In *RecSys 2014 - Proceedings of the 8th ACM Conference on Recommender Systems*, New York, New York, USA: Association for Computing Machinery, Inc, 161–68. http://dl.acm.org/citation.cfm?doid=2645710.2645737 (March 15, 2021).

Said, Alan, Shlomo Berkovsky, and Ernesto W. De Luca. 2011. "Group Recommendation in Context." In *ACM International Conference Proceeding Series*, New York, New York, USA: ACM Press, 2–4. http://dl.acm.org/citation.cfm?doid=2096112.2096113 (March 15, 2021).

Vo, Thanh Vinh, and Harold Soh. 2018. "Generation Meets Recommendation: Proposing Novel Items for Groups of Users." In *RecSys 2018 - 12th ACM Conference on Recommender Systems*, New York, NY, USA: Association for Computing Machinery, Inc, 145–53. https://dl.acm.org/doi/10.1145/3240323.3240357 (March 15, 2021).