Final Report: Louis's Film Recommender System Introduction

Gaoyuan Huang

Table of Contents

Data introduction

The data is from Kaggle.(https://www.kaggle.com/rounakbanik/the-movies-dataset/data) .

Ratings_small.csv : I used a subset of the data which includes ratings from 700 users on 9,000 movies. But when I actually import the data, there are 671 users.

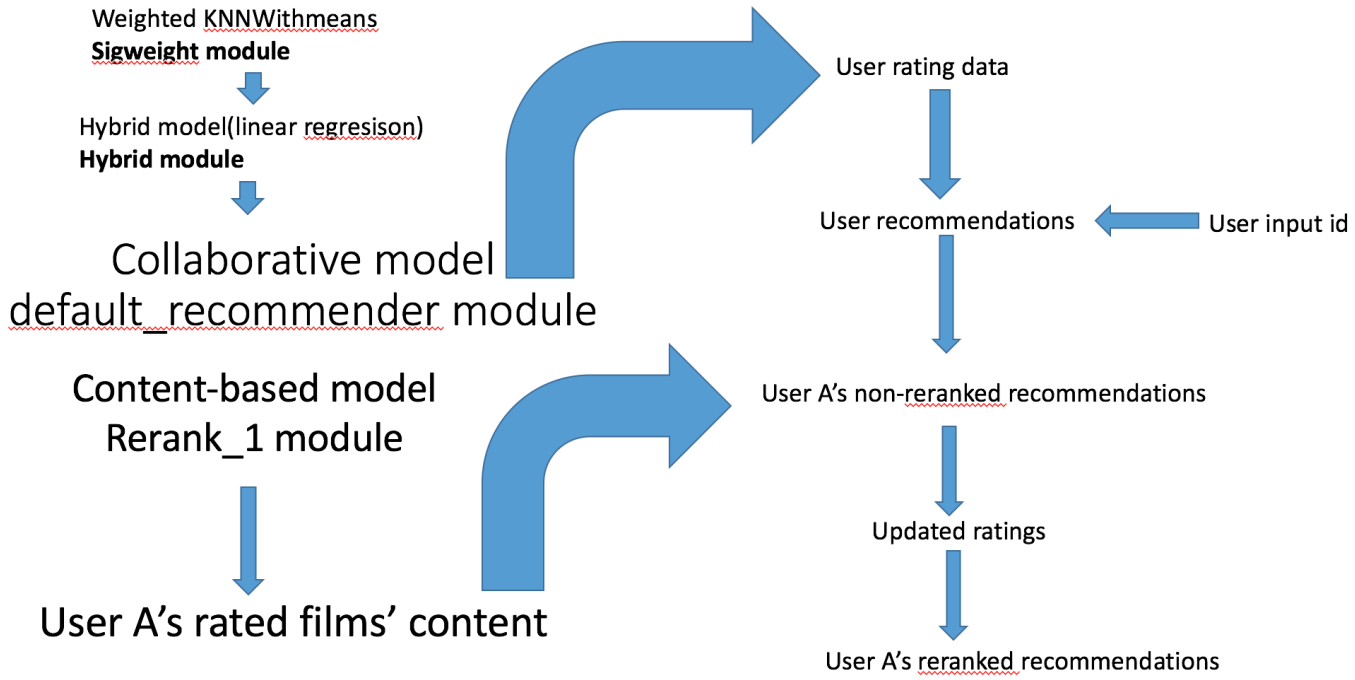
Credits.csv: Consists of Cast and Crew Information for all movies in the dataset. I need to user actors and directors information from this file.

Links_small.csv: The file that contains the TMDB and IMDB IDs of all the movies in the subset I will use. This file can help me map the movieId in ratings_small.csv with TMDB and IMDB.

keywords.csv: Contains the movie plot keywords for our MovieLens movies. This file provide very useful content information of these films.

movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies. I used this file to extract popularity and IMDB rating for future use.

Project Structure



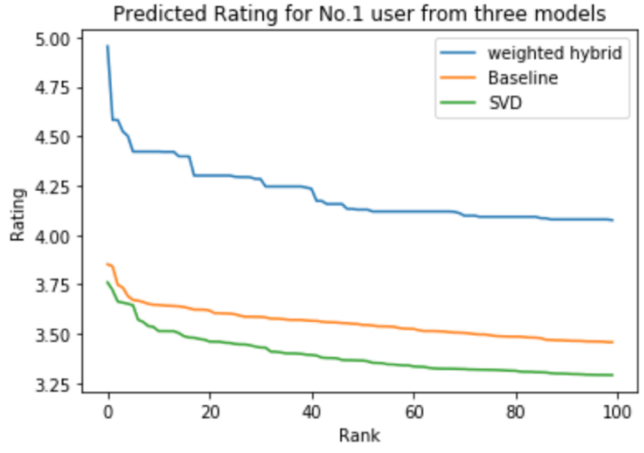
Data Preprocessing and feature extraction

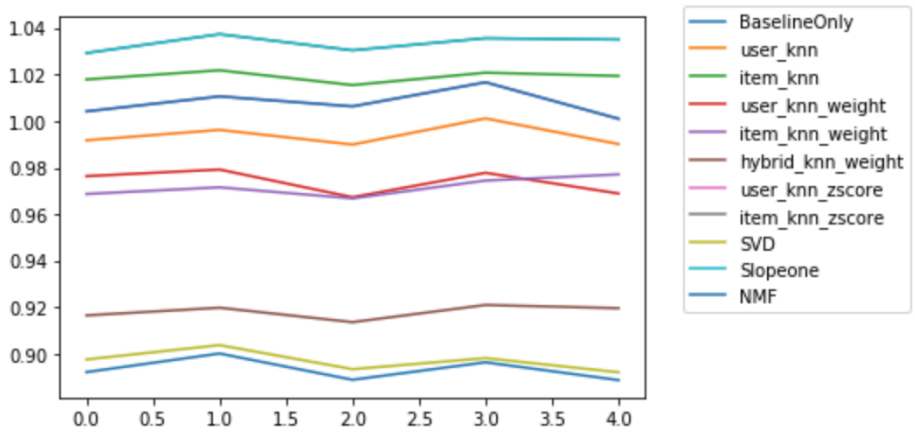
In order to make a content-based model, I have to extract useful content about each film from the dataset. I decided to extract directors, actors, keywords and genres to represent a film. So I extracted these four features of each film as content features from the dataset for those films the user rated. I also extract some variables which can help me show the recommendation result like title, popularity, vote average and vote count. Then I merge all the film information data together as a dataframe and made it as a csv file for future use. Please refer to preprocess+film+data.html for codes.

For keywords specifically, I use NLTK to help me stem all keywords and replace them with the root. Then I grouped synonyms (replace words with frequency lower than 5 to to the most frequent synonym) and later I deleted low keywords with low frequency (smaller than 3). Then I remove duplicated films. In the end, I merged the film features dataframe and the user rating dataframe so we can filter out those films who don't have any content in our data and we don't need to consider them in our content-based model. Film_merged.csv is my final data for content based model. Please refer this part to content_based.py.

Weighted Hybrid collaborative recommendation model

I first check all model performance on our data. I found out actually the baseline model has very good result which is strange. So I check the predictions it created, I found the predictions for films are around their global mean value in the dataset. So if we just minimize the RMSE, the recommendation will be very bad and it will just predict the rating to be close to mean value so it can minimize the RMSE like baseline model and SVD model did here. But if I assign weights to each algorithm in my models it can help differentiate film ratings. Besides it also has very good RMSE. So it did well in terms of bias variance tradeoff. Please refer to model+explore.html.





Here is how I did:

I created a class named KNNSigWeighting which inherits KNNWithMeans function in surprise by computing the overlap between users and films then update their similarity matrix so that in prediction part the more similar films and users will have more impact in predictions. This is the weighted part of the model. Please refer to sigweight.py.

Then I think we can also use hybrid model to again make the model better and they can make up each other's disadvantages. For this part, I modified the create a class named WeightedHybrid which inherits AlgoBase. Then I used linear regression to find the optimal weights for each algorithm in our model. This is the hybrid part of the model. Please refer to hybrid.py.

I saved the top 100 films as candidate films for reranking. And the model is saved as default_recommender.py for updating recommendations.

Content-based model for reranking

So I will rerank based on the method below:

Updated_rating = user_film_predicted_rating * User_film_similarity

User_film_similarity = director_sim + actor_sim + genre_sim + keywords_sim

Category_sim =
$\sum \text{film}[\text{Category}[feature]] \cap \text{user}[\text{Category}[feature]] * user[\text{Category}[feature]]$

I will first accumulate weighted frequencies according to ratings the user gave for all four features from films the user have rated. But for genres, we need to do one more thing. Since the genres have some dominant value like Drama, I created a tfidf to transform it.

$User_i[Category_j[feature]] = \sum u\_ri$

$User_i[genre[feature]] = \sum u\_ri$* tfidf[feature]

$User_i[genre[feature]]$ is named profile_dict in in build_model function in reranl_1.py.

Now we have a dictionary of dictionary called profile_dict. The inner dictionary has key for a specific director, actor, genre or keyword and the value is its (tfidf) frequency. The outer dictionary has key for those four features and the value is the inner dictionary. So now we can compute the similarity over four features for all candidate films. For a specific category, if a film has a feature that the user has in its dictionary, then we add the value of that feature to the total similarity of this category. Then I normalized the similarity in each category to 1-5 scale. Then we can compute the new rating for each film as an updated rating for future ranking.

At the same time, I used classification model to train each user using all these features because I want to pick out the films user will like in the candidate films. I set the thresh hold value for a user to like a film as 3. So now we have a dataframe contains film's four features and a y value that indicate whether the user like it or not. Next, I used cross validation to test seven models and we would like to have more true positive in our result so we should measure the precision as our evaluation metrics. After running all seven models, I chose the model with the highest precision value and use it to predict the candidate films.

Sometimes, the classification model will give positive predictions lower than the amount of recommendations user asks, then I retrieve films with high updated rating from the rest of reranked films. Now we finally have the final recommendation. Please refer this part to rerank_1.py.

Building Recommender System

recommender_1 module

I created a class called Recommender for using my recommender system. The recommender has three main functions:

1. Recommend films for users already in our dataset. (10min)
2. Recommend films for new user(you) by recording your film ratings.(2min)
3. Update all recommendations if there are new data added to the dataset. (Not automatically, can be done offline)(10min)

Attribute of the class:

Self.isnew: Is the user rating data new?
Self.random_rated: Did user use random_rate function?
Self.my_recommend: Did user run recommend_me?
Self.my_id: User id created once the user run the code.

Important Functions:

1:test.recommend_user(user_id = 11, rerank_ = True, n_reco = 10, show_rating = False)

This is the function to get a specific user's recommendation. This function will retrieve a user's non-ranked prediction which has been created offline by calling recommend_getter() function. Then it will choose to run rerank function from rerank_1 module depending on parameters user gives. Then after calculating, we will have new recommendations and then print result.

Parameters:

user_id: From 1 to 671. Should be integer.
If you have run recommend_me, you can use your own user_id.
Default is 10.
rerank_: True or False
Decide whether to rerank the result or not.
Default is True.
n_reco: From 10 - 90. Should be integer.
Decide how many results you want.
Default is 10.
show_rating: True or False.
Decide whether or not to show the user's all ratings.
Default is False.

2:test.random_user(rerank_ = True, n_reco = 10, show_rating = False)

Function to randomly retrieve a user's recommendation. This function will first randomly create a user_id in our user id set and then call recommend_user() function to print recommendations.

Parameters:

rerank_: True or False
Decide whether to rerank the result or not.
Default is True.
n_reco: From 10 - 90. Should be integer.
Decide how many results you want.

Default is 10.
show_rating: True or False.
Decide whether or not to show the user's all ratings.
Default is False.

3:test.recommend_me(rerank_ = True, n_reco = 10, show_rating = False):

This is the function to get a your recommendations. If the user is new, this function will call random_film() to retrieve some random popular films and then call random_rate() to record user's ratings. After having new data, the function will call predict() function in default_recommender() which is the collaborative model to predict my recommendation and save it as my_predictions.pickle. Then it will retrieve the saved predictions and call recommend_user() to rerank and print. If the user has already run recommend_me and there are no new data entered, then it will directly retrieve my_predictions.pickle and call recommend_user().

Parameters:

rerank_: True or False
Decide whether to rerank the result or not.
Default is True.
n_reco: From 10 - 90. Should be integer.
Decide how many results you want.
Default is 10.
show_rating: True or False.
Decide whether or not to show the user's all ratings.
Default is False.

The recommender will choose some films for you to rate until you rated 15 films we will move on to learning your data. Please enter your ratings from 1 to 5 (including 1 and 5) and press return(enter) on your keyboard. We accept float numbers.
If you haven't seen this film, you can type -1 and press return(enter) to move on to next film.
The test has limited number of films so if you haven't watched all of them, please watch them and then come back to rate!
If you did wrong in entering the ratings. We will not save your previous data and we have to start over.
If you are a new user or you rate new films and save the data, this process will take a while, please be patient.

4:test.rate(user_id = 672)

Function to rate new films.

Parameters:

user_id: Your user id. Default is 672. Should be greater than 671 since smaller number are original users, you should't update their ratings.

Type test.rate() or test.rate(user_id = YOUR USER ID). If you only only runs recommend me once. Then your user id is 672(default) then you don't need to enter your id.
Attention: Ple
We will not delete your user ratings when you close the recommender. And when you open the recommender next time, the system will use the latest version of user rating file.
So please be careful when you rate a film. And you might re-rate a film since we haven't created new function to help you overlap your old ratings.

5.test.save()

Function to save user's information.

6.test.rate_for_fun(novelty = 1)

Function to rate films randomly. It will call random_film() first to get the films and then ask user to rate.

Parameter:

Novelty: From 1- 30. Should be integer.
The degree of the film's novelty you want. Bigger number means more surprise!
The process is same as the cold starter rating function.
Attention: Enter -1 to skip this film. If you want to end and save the ratings, enter 11. If you don't want to save, enter 12 or close the program.

7.test.reset()

Type test.reset() to reset every thing! But I won't delete any files actually. Just reset the data to the original one

8.test.bsl_recommend_user(user_id = 10, n_reco = 10, model = "baseline"):

This is the function to get a specific user's baseline model recommendation. This function will call baseline_recommender_getter() to retrieve the baseline predictions from saved baseline_predictions.picke.

Parameters:

user_id: From 1 to 671. Should be integer.

If you have run recommend_me, you can use your own user_id.
Default is 10.

n_reco: From 10 - 90. Should be integer.
Decide how many results you want.
Default is 10.

model: “baseline” or “svd”. Should be string format.
The model you want to compare.

9.test.bsl_recommend()

Function to calculate recommendations. It calls predict function in bsl_recommender module to predict recommendations using Baselineonly and SVD model in Surprise.

Use it if you add new data.

10.test.update_all()

Update all users' recommendations. Call predict() in default_recommender module to recalculate recommendations using new user rating file.

Functions that user should not use:
random_rate()

Function to create profile for cold starters. It will use candidate_films created by random_films() to ask user to rate.

Hybrid module

This python file has a class name WeightedHybrid. It implements the collaborative model. Predict and fit function has same functionality as other recommendation algorithms. But I add testme function here to help predict a specific user’s prediction so we can improve the efficiency of the model by not calculating all predictions at once.

Sigweight module

This python file implements KNNSigWeighting function to update similarity matrix of user-user and item-item in order to improve performance.

Sig_weight() function assigns weights.

Rerank_1 module

This module is to rerank the recommendations.

rerank()

Function to rerank recommendations. It calls build_model function to build a classifier based on user rating and films' features. Then it calls build_fil_overlap function to create a dataframe contains four features' similarity between candidate films and the user. Then it applies the classifier into this dataframe to predict whether the user would like the films in the list. In the end, the function will update the ratings based on User_film_similarity.

Updated_rating = user_film_predicted_rating * User_film_similarity

User_film_similarity = director_sim + actor_sim + genre_sim + keywords_sim

The function will return the candidate films with updated rating and the index of user-like films.

build_film_overlap()

Go through all candidate films and compute the similarity over four categories' features between user and film by the following method:

Category_sim =
$\sum \text{film}[\text{Category}[feature]] \cap \text{user}[\text{Category}[feature]] * user[\text{Category}[feature]]$

build_model()

Function to create classifier.

First, it builds a profile_dict according to user's rating for each film's category and feautres:

$\text{User}_i[\text{Category}_j[\text{feature}]] = \sum u\_ri$

$\text{User}_i[\text{genre}[\text{feature}]] = \sum u\_ri *$ tfidf[feature]

Then It calls build_film_overlap() to create a dataframe as training data to learn how user's rating pattern in directors, actors, genres and keywords. It uses 5-fold cross validation to test seven classifiers and uses precision to select the best classifier.

default_recommender module

This module is to create non-ranked recommendations.

predict()

This function creates instances of KNNSigWeighting and WeightedHybrid from sigweight module and hybrid module. So now we can combine these two methods together to create a new model to help us predict. Then we can fit and predict depending on the user need. In the end, it calls get_top_n() function to get top rated 100 films for reranking and saves the predicitons for future use.

bsl_recommender module

This module is for comparing models. It will run two models, BaselineOnly and SVD. The result will be saved as baseline_predictions.pickle for future use.

Open_me module:

This is the program for user to run. It creates an instance of Recommender class.