The Movie Analyzer

CMPT733 Final Project
Zhicheng Xu, Rong Li, Chao Zhang

Motivation and Background

Building a sophisticated recommendation system has been an influential topic in many business domains. It is crucial to give users informative suggestions and recommendations based on their preferences. Our team was interested in two of the most trending recommendation systems: Context-based recommendation and Collaborative filtering recommendation. We are looking to practice and improve these systems through our project scope. We chose the film domain as our context to build our recommendation systems upon. Along with building a rating-prediction feature as an add-on, we hoped to create a well-rounded platform for users to explore their favourite movies.

Problem Statement

There are four main questions addressed in this project:

1. Given a user's favourite movie, what other movies can we recommend based on the content of the selected movie?
2. Can we give recommendations based on similar users' preferences?
3. Is it better to combine the previous two approaches and make a hybrid recommendation model?
4. Given some information about a new movie or a movie that is not yet released, can we predict a rating for it?

Data Science Pipeline and Methodology

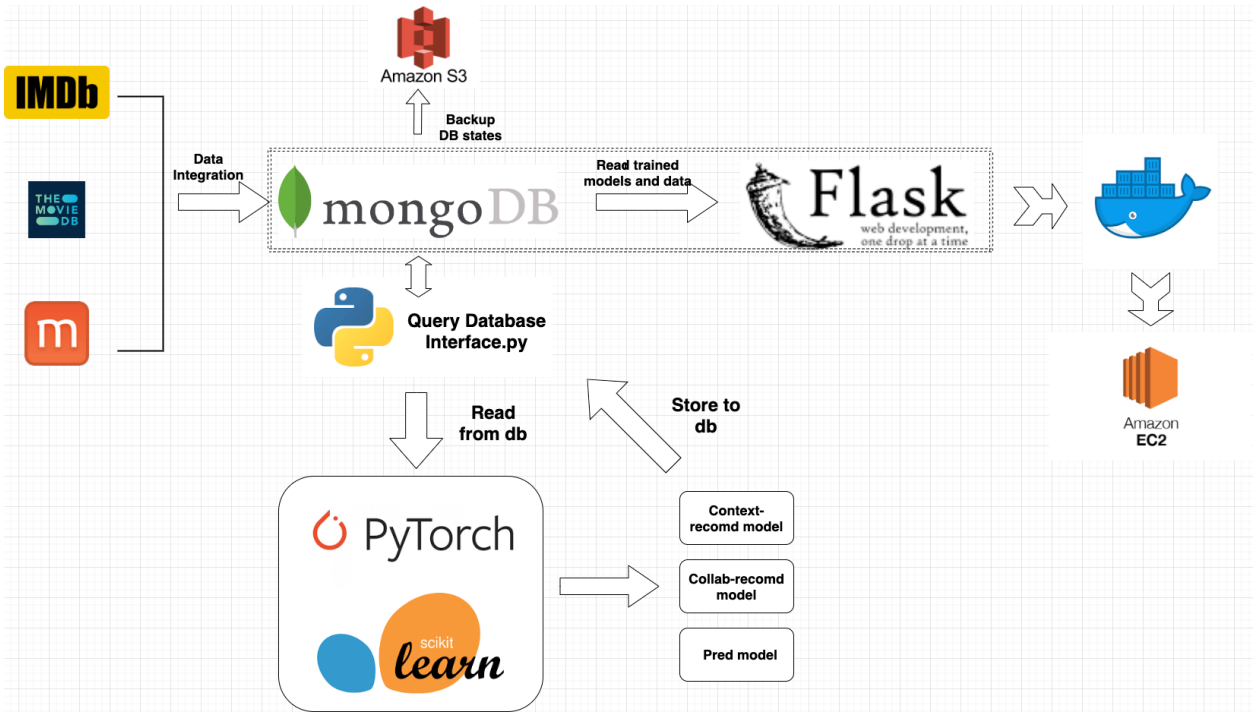


Figure 1: Project workflow

Data Collection

As we are aiming to integrate and combine movie-related data from multiple movie websites, our data are collected from three different sources in this project: IMDB [1], TMDB(The Movie Database) [2], and MovieLens [3]. These sources contain 7720893, 803605, and 58098 movies respectively. IMDB's 2GB data can be retrieved as TSV files, MovieLens 1.3GB data can be retrieved as CSV files, and TMDB data can be obtained through their API. There are 27753445 individual user ratings from MovieLens and 30000 individual user ratings from TMDB available for our Collaborative Filtering model.

Data Cleaning and Data Integration

One special feature of our data is that the information about a movie is coming from three different sources, which means there would be a lot of data aggregation needed when querying a movie. Since these data will be frequently fetched together, we plan to put them in the same place for easier and faster querying by using NoSQL (MongoDB to be exact) instead of relational databases. Another reason to use MongoDB is that the data coming from three different sources are having different formats and schemas. For example, at the end of the data integration, some movies may not have certain features like other movies do. We do not want a fixed schema in our database because of the inconsistency of the data from three sources.

To implement our integration procedure, we planned to do three small integrations first to clean and store the data from three sources into three MongoDB collections. The final integration will happen afterwards to put everything into a single collection. After achieving these procedures, querying one collection would be sufficient to know everything about a movie, rather than looking for information from three sources.

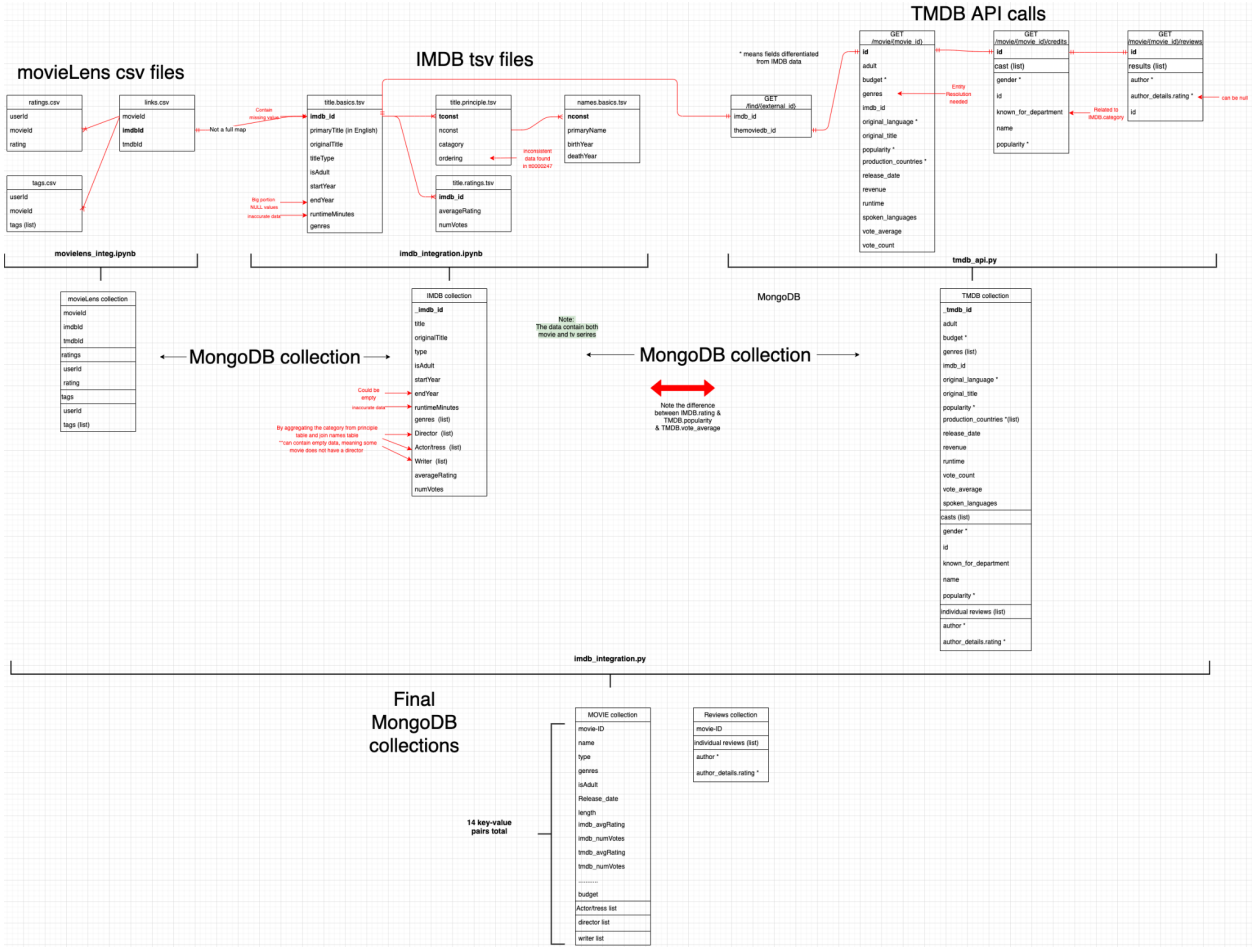


Figure 2: An overview of the data integration process

Due to the fact that data integration is not the biggest selling point of MongoDB, we used Pandas for the early stage of the data preparation procedure. Luckily, our IMDB and MovieLens datasets were close but not yet past the limit that Pandas can handle in memory at once. The procedures were written as notebooks in “imdb_integration.ipynb” and “movielens_integ.ipynb” respectively. The main focus here is to join all the tables together, remove the rows that contain lots of null values and store the results to MongoDB. TMDB data preparations are written in “scripts/tmdb_api.py” and “create_tmdb.py”, the prior takes an IMDB movie id as input and integrates data from multiple API calls, the latter calls the prior to iterate through IMDB collection and store corresponding TMDB movies to TMDB collection. As one may have noticed, we only retrieved the TMDB movies which also exist in the IMDB collection. This was for simplicity, the

IMDB movies had great coverage, TMDB movies that do not have an IMDB link were removed from consideration.

The next step was to integrate these three collections into one. Since they are all stored as MongoDB collections, we have written a script “imtm_dbintegration.py” which utilized the PyMongo library to manage our database. First, the script will go through all the collections and unset the empty or null key-value pairs, then it will iterate through all the IMDB movies, and look for key-value pairs to add from the other two collections. Some keys are considered the same, such as “adult” in TMDB and “isAdult” in IMDB. Some keys are considered two different attributes that both are needed to store in the final collection, such as “IMDB_rating” and “TMDB_rating”. Due to the flexible schema nature of MongoDB, the same key set can store different data types, however, this would become a problem later when querying the data and feeding the ML model. Therefore, a consistent data type needs to be preserved in the same key set.

Entity resolutions are needed in some key-value pairs, such as director names, writer names, or cast names. Because the data are coming from three different websites, two seemingly different names may refer to the same person. We are using the “difflib” Python library to decide whether two strings are actually the same by setting a cutoff of 0.7. If 70% of the characters are the same, then we say these two names are referring to the same person.

Both MovieLens and TMDB datasets contain individual user ratings. Our second model, Collaborative Filtering, values individual user inputs heavily rather than other features. More than often, only individual reviews are retrieved from the database. Therefore, to maintain our database design philosophy that only data that are fetched together are stored together, we decided to store the individual reviews as a separate collection. The final integrated movie collections and the user rating collections are all stored in AWS S3 as backups.

Other Database Design Aspects

To query MongoDB contents, MQL is used. In order to ease the querying process, we wrote a Python interface to encapsulate the MQL queries we often use into Python functions. Such interfaces were written in “scripts/query_database.py” and “web/query_db_web.py”. One can refer to “dummy.ipynb” for a demonstration of how to use these interfaces. Because querying the whole collection can result in huge returned JSON objects, we wrote MQL queries to limit the returning result by setting a cut-off release year of a movie, a sample size, and an option to only query certain columns.

At the end of the machine learning model training process, the trained models are stored in MongoDB as well. We will have four trained models that need to be deployed, we store these four models in the database with corresponding keys “model-type”, which indicates which model this record is. Considering each group member may have needed to revert back to the previously trained model, we developed a way for us to store different versions of models. For accurate and fast model querying, the “Description” key is set to indicate the version. During the querying process, if multiple models have the same model_type and description, the latest one

would be retrieved instead. In terms of data format, the models were stored as bytes. The size of each model was approximately 30MB-50MB. Due to the size limitation of a MongoDB document being 16MB, we chopped the model bytes into chunks using the “gridfs” library before inserting them into our database.

Mongo Express Database: aoligei Collection: fs.files

_id	model_type	Upload_time	Description	md5	chunkSize	length	uploadDate
606f8ef6d6be74b602dcdd24	New Film Rating Prediction	Thu Apr 08 2021 16:17:10 GMT+0000 (Coordinated Universal Time)	Rating prediction: Max Xu Version 1	612d4bacb84c44e79fb88f1a52d291d6	261120	53246882	Thu Apr 08 2021 23:17:51 GMT+0000 (Coordinated Universal Time)
6070d796579d015e9cb1873a	Context-based Recommendation System	Fri Apr 09 2021 15:39:18 GMT+0000 (Coordinated Universal Time)	Context-based: Chao Zhang Version 1	0a32140b21fa98cfd9747d286a26d1d5	261120	8655761	Fri Apr 09 2021 22:39:24 GMT+0000 (Coordinated Universal Time)
6070d817579d015e9cb1875d	Context-based Recommendation System	Fri Apr 09 2021 15:41:27 GMT+0000 (Coordinated Universal Time)	Context-based: Chao Zhang Version 2 more weight o…	03a94a5a2b52c745b5a09468913e8e8d	261120	8655761	Fri Apr 09 2021 22:41:33 GMT+0000 (Coordinated Universal Time)
6070d8d9579d015e9cb18780	Collaborative Recommendation System	Fri Apr 09 2021 15:44:41 GMT+0000 (Coordinated Universal Time)	Collaborative filtering: Chao Zhang Version 1	0e7008638b1f34017486f0dad9542f79	261120	8656311	Fri Apr 09 2021 22:44:48 GMT+0000 (Coordinated Universal

Figure 3: A snapshot of how we store our models

Analysis/Modeling

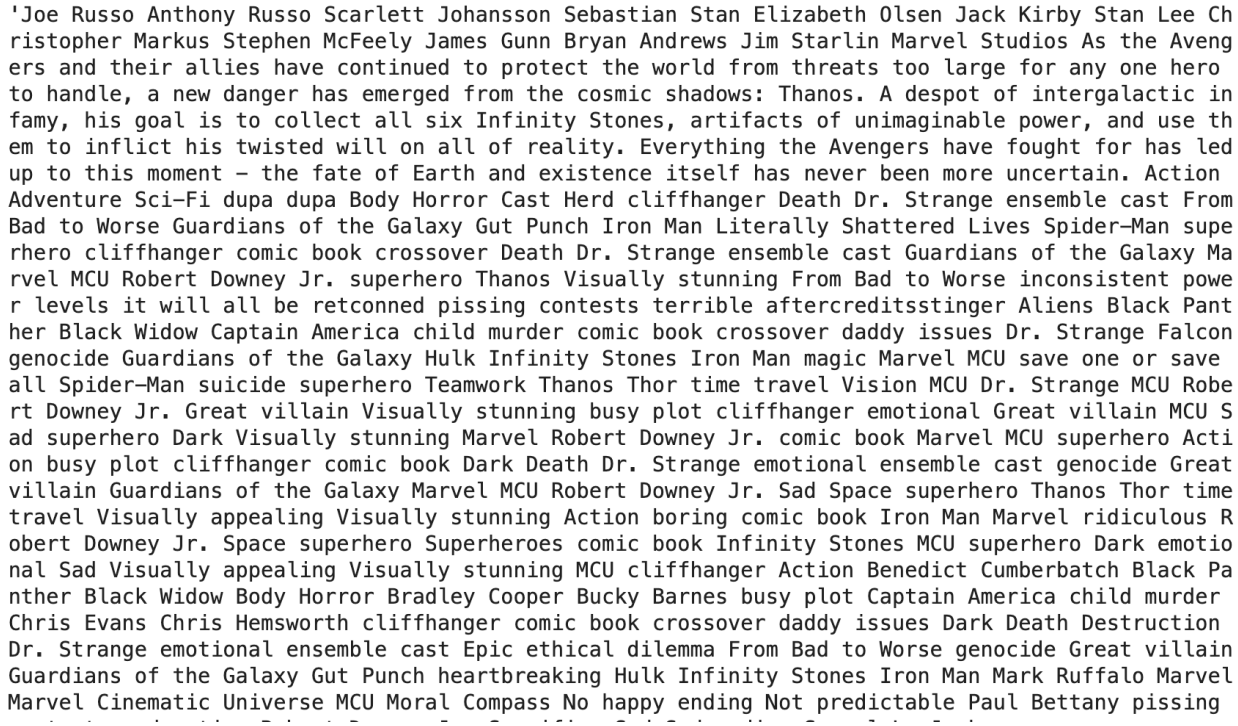
1. Recommendation system

We use deep learning to build three different recommendation systems which are content-based, collaborative filtering, and hybrid models. Our approach tackles the content and collaborative model separately at first and combines the efforts to produce a hybrid system with the advantages of both.

As mentioned above, our data are coming from three different sources. After a lot of data integration and data cleaning, we chose to use a common subset from three sources which contain 181664 users and 21639 movies as our dataset.

Content-based:

The approach we take is to first build a document for each movie using features such as directors, casts, movie overview, and tags users provide. Below is an example document for “Avengers: Infinity War”. As you can see, words like “Sci-Fi”, “Marvel”, “superhero” appear frequently.



```
'Joe Russo Anthony Russo Scarlett Johansson Sebastian Stan Elizabeth Olsen Jack Kirby Stan Lee Ch
ristopher Markus Stephen McFeely James Gunn Bryan Andrews Jim Starlin Marvel Studios As the Aveng
ers and their allies have continued to protect the world from threats too large for any one hero
to handle, a new danger has emerged from the cosmic shadows: Thanos. A despot of intergalactic in
famy, his goal is to collect all six Infinity Stones, artifacts of unimaginable power, and use th
em to inflict his twisted will on all of reality. Everything the Avengers have fought for has led
up to this moment - the fate of Earth and existence itself has never been more uncertain. Action
Adventure Sci-Fi dupa dupa Body Horror Cast Herd cliffhanger Death Dr. Strange ensemble cast From
Bad to Worse Guardians of the Galaxy Gut Punch Iron Man Literally Shattered Lives Spider-Man supe
rhero cliffhanger comic book crossover Death Dr. Strange ensemble cast Guardians of the Galaxy Ma
rvel MCU Robert Downey Jr. superhero Thanos Visually stunning From Bad to Worse inconsistent powe
r levels it will all be retconned pissing contests terrible aftercreditsstinger Aliens Black Pant
her Black Widow Captain America child murder comic book crossover daddy issues Dr. Strange Falcon
genocide Guardians of the Galaxy Hulk Infinity Stones Iron Man magic Marvel MCU save one or save
all Spider-Man suicide superhero Teamwork Thanos Thor time travel Vision MCU Dr. Strange MCU Robe
rt Downey Jr. Great villain Visually stunning busy plot cliffhanger emotional Great villain MCU S
ad superhero Dark Visually stunning Marvel Robert Downey Jr. comic book Marvel MCU superhero Acti
on busy plot cliffhanger comic book Dark Death Dr. Strange emotional ensemble cast genocide Great
villain Guardians of the Galaxy Marvel MCU Robert Downey Jr. Sad Space superhero Thanos Thor time
travel Visually appealing Visually stunning Action boring comic book Iron Man Marvel ridiculous R
obert Downey Jr. Space superhero Superheroes comic book Infinity Stones MCU superhero Dark emotio
nal Sad Visually appealing Visually stunning MCU cliffhanger Action Benedict Cumberbatch Black Pa
nther Black Widow Body Horror Bradley Cooper Bucky Barnes busy plot Captain America child murder
Chris Evans Chris Hemsworth cliffhanger comic book crossover daddy issues Dark Death Destruction
Dr. Strange emotional ensemble cast Epic ethical dilemma From Bad to Worse genocide Great villain
Guardians of the Galaxy Gut Punch heartbreaking Hulk Infinity Stones Iron Man Mark Ruffalo Marvel
Marvel Cinematic Universe MCU Moral Compass No happy ending Not predictable Paul Bettany pissing
```

Figure 4: example document for “Avengers: Infinity War”

Next, we transformed the corpus of documents into a Term Frequency - Inverse Document Frequency (TF-IDF) representation. To put it simply, TF-IDF takes how often a word appears in a document, over how often it appears in a collection of documents. It measures how relevant and unique a word is to a document in a collection of documents.

By using TF-IDF tokenizes provided by sklearn, we convert a collection of raw documents to a matrix of TF-IDF features. In TF-IDF space, each dimension represents how important a certain word is to a movie. Our matrix is (21639, 16049), which is very large and makes it hard to do computations on it. We would like to compress this TF-IDF matrix into a lower dimension while still preserving most of the information. To perform this compression, we choose to use an autoencoder. Autoencoder is a type of feedforward neural network where the input and output dimensions are the same and used to learn efficient data codings in an unsupervised manner. It compresses the input data into a lower dimension and then reconstructs the output from this representation. In our case, the TF-IDF matrix input is compressed into a 100-dimensional encoding layer and the output is reconstructed using this hidden layer. By using a mean squared error as a loss function, the network learns a function to map input data into a lower dimension while preserving most of the information.

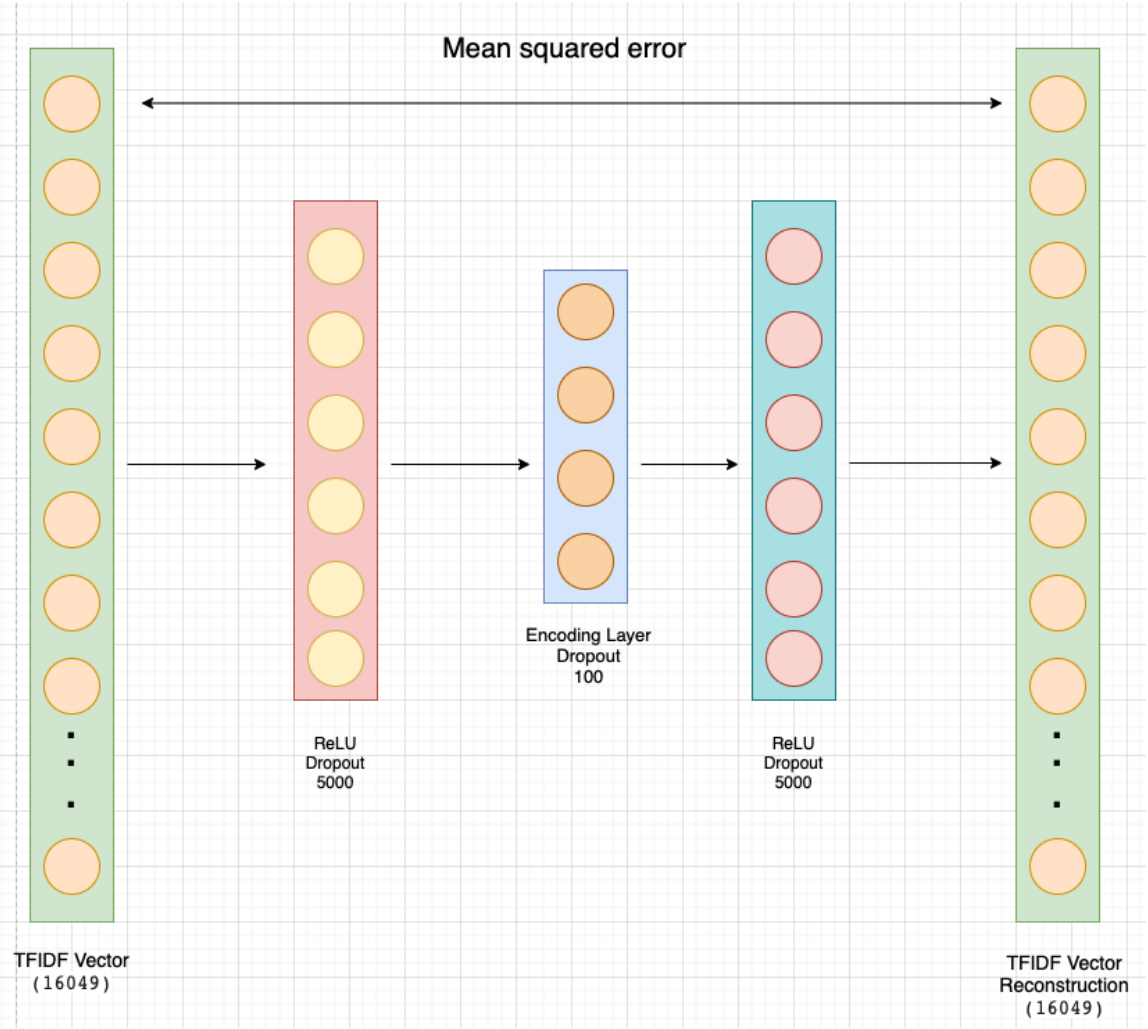


Figure 5: Neural network for content-based model

Finally, we get the encoding layer by feeding our input data to the trained autoencoder and compute the similarity matrix for all movies. Using the content-based movie encoding, the top 10 most similar movies to “Avengers: Infinity War” are shown below.

	primaryTitle	Directors	tmdb_id	genres	ratings_count	avg_rating
0	Ant-Man	Peyton Reed	102899	Action Adventure Sci-Fi	5806	3.590424
1	Fantastic 4: Rise of the Silver Surfer	Tim Story	1979	Action Adventure Sci-Fi	3868	2.668821
2	X2: X-Men United	Bryan Singer	36658	Action Adventure Sci-Fi Thriller	21404	3.613297
3	X-Men: The Last Stand	Brett Ratner	36668	Action Sci-Fi Thriller	11171	3.283144
4	Iron Man	Jon Favreau	1726	Action Adventure Sci-Fi	26682	3.848194
5	Spider-Man 2	Sam Raimi	558	Action Adventure Sci-Fi IMAX	21989	3.453454
6	Iron Man 3	Shane Black	68721	Action Sci-Fi Thriller IMAX	8028	3.536996
7	Thor	Kenneth Branagh	10195	Action Adventure Drama Fantasy IMAX	9311	3.385619
8	The Amazing Spider-Man 2	Marc Webb	102382	Action Sci-Fi IMAX	2585	3.112379
9	The Avengers	Joss Whedon	24428	Action Adventure Sci-Fi IMAX	17700	3.799011

Figure 6: top 10 most similar movies to “Avengers: Infinity War” using content-based model

Collaborative filtering

We use the same set of movies to train our collaborative model. The difference is we include users’ ratings to each movie as shown below.

	userId	movieId	rating	timestamp
0	0	0	2.0	1256677210
1	0	1	3.5	1256677486
2	1	2	3.5	1113766176
3	1	3	4.5	1113766820
4	1	4	3.5	1113766824

Figure 7: rating data set

The main technique we applied here is embedding. An embedding is a low-dimensional space that captures the relationship of vectors from a higher-dimensional space. We assume the user's interest in movies can be roughly explained by d dimensions. Therefore, each movie becomes a d-dimensional point where the value in dimension d represents how much the movie fits that aspect. For example, one dimension may represent action movies, and a point having a higher value in this dimension means it has a high probability to be an action movie.
With the main concept, we built two different collaborative models with explicit feedback and implicit feedback. Before diving into the implementation, let go over the difference between the two feedbacks.

Explicit Feedback

In the context of recommendation systems, explicit feedback is direct and quantitative data collected from users. For example, in our case, each movie has received at least one user's rating scale from 1 to 5. These ratings are provided directly by users.

However, one issue with explicit feedback is that they are rare. Because users tend not to easily provide such explicit ratings.

Implicit Feedback

On the other hand, implicit feedback is collected indirectly from user interactions. For example, movies that users watched are used as implicit feedback to make recommendations for them. Or an item you bought on Amazon will be used to recommend similar items to you.

The advantage of implicit feedback is that it is easy to get. Every click we make, every webpage we go can be used as implicit feedback. Nowadays, most recommendation systems are built using implicit feedback, which allows them to give real-time recommendations.

However, implicit feedback also has its shortcomings. One would be every user interaction is assumed to be positive and we are not able to capture the negative feedback.

For both models, a fully connected neural network is used to find movie embeddings. In this architecture, the movie embedding is of size (21639, 100), which means we have 21639 movies, and each movie is explained by d dimensions.

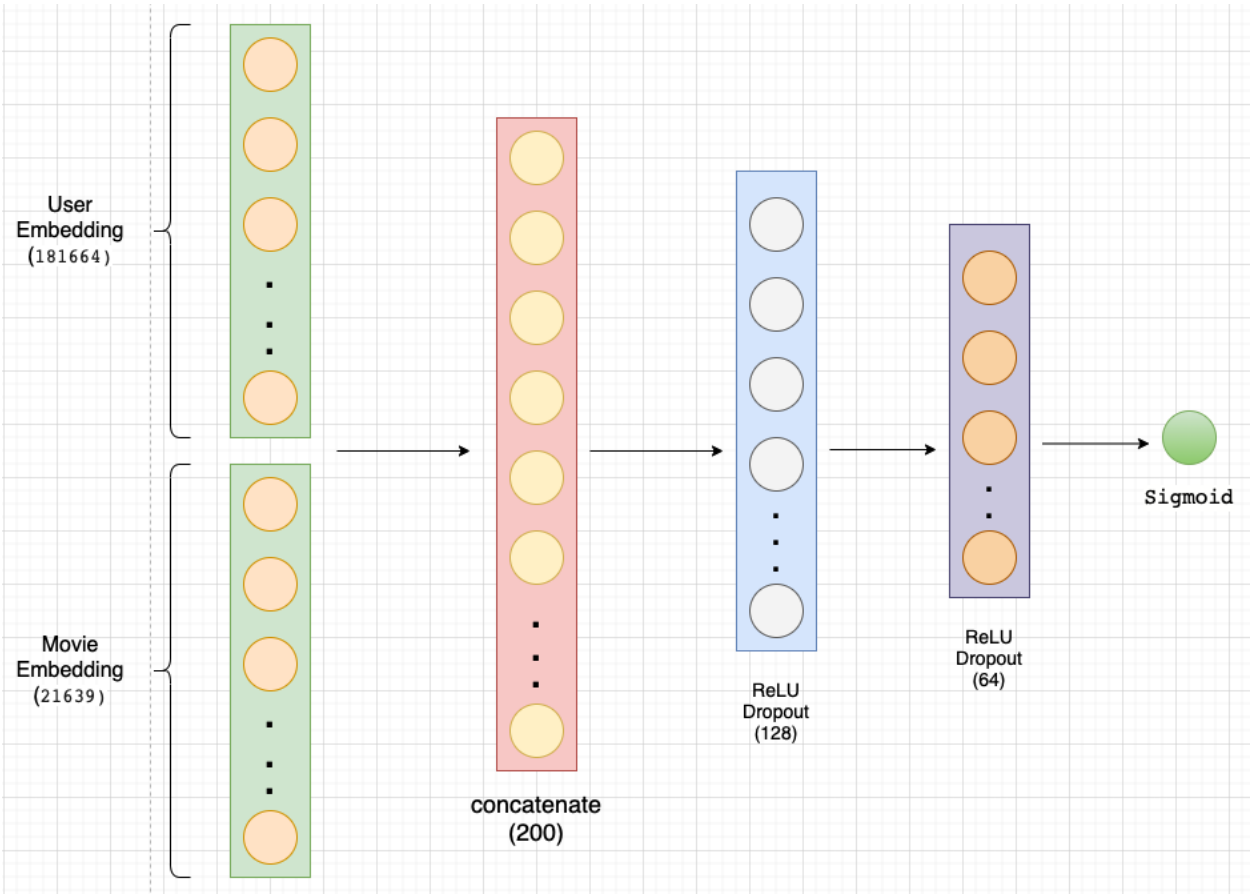


Figure 8: Neural network for Collaborative filtering

Next, we will go through the two models separately.

Explicit Feedback model

Each training data point is a user id, movie id and an explicit rating. The user and movie ids are encoded, and the rating is scaled from 0 to 1 using a min-max scale. Then we feed data into our neural network to train the embeddings. By using the binary cross-entropy as a loss function (the loss is between predicted ratings and the actual ratings), our network learns a movie embedding. We then use this learned embedding to calculate the similarity between movies. Using the explicit feedback model, the top 10 most similar movies to “The Dark Knight Rises” are shown below.

	primaryTitle	Directors	tmdb_id	genres	ratings_count
0	The Dark Knight	Christopher Nolan	155	Action Crime Drama IMAX	44741
1	Elite Squad 2: The Enemy Within	José Padilha	47931	Action Crime Drama	1673
2	The Place Promised in Our Early Days	Makoto Shinkai Yoshio Suzuki	12924	Animation Drama Romance Sci-Fi War	137
3	Poolhall Junkies	Mars Callahan	14293	Comedy Drama Thriller	361
4	Like Stars on Earth	Amole Gupte Aamir Khan	7508	Drama	316
5	Johnny Cash at Folsom Prison	Bestor Cram	126016	Documentary	25
6	Deadpool 2	David Leitch	383498	Action Comedy Sci-Fi	1633
7	Batman Begins	Christopher Nolan	272	Action Crime IMAX	32027
8	Evangelion: 1.0 You Are (Not) Alone	Hideaki Anno Kazuya Tsurumaki Masayuki	15137	Action Animation Sci-Fi	604
9	Guardians of the Galaxy Vol. 2	James Gunn	283995	Action Adventure Sci-Fi	4942

Figure 9: recommendation for Explicit Feedback model

Implicit Feedback model

The data preprocessing is a little different here. Since our data have users' explicit ratings, we need to convert that to implicit feedback. To do that, we simply binarize the ratings such that 1 represents that the user has rated the movie. The objective is also changed. Instead of trying to predict the ratings, we are trying to predict whether the user will interact with each movie. However, as mentioned before, one of the shortcomings of implicit feedback is that we are not able to capture the negative feedback. In order to overcome this problem, we assume that movies that users not rated are those that users are not interested in. Based on this assumption, we generate 5 negative samples for each row of data, and the ratio of negative to positive samples will be 5:1.

After data preprocessing, we split the training and test data based on timestamps. For each user, the most recent rating is used as the test set and the rest of the ratings are used as the training set. Doing this is better than random split which can potentially use a user's recent

ratings as the training set and earlier ratings as the test set. We then feed the data to our above neural network to get the movie embeddings and finally use the leaned embeddings to find similar movies.
Using the implicit feedback model, the top 10 most similar movies to “The Dark Knight Rises” are shown below.

	primaryTitle	Directors	tmdb_id	genres	ratings_count
0	The Bourne Ultimatum	Paul Greengrass	2503	Action Crime Thriller	21677
1	Ratatouille	Brad Bird Jan Pinkava	2062	Animation Children Drama	19803
2	Borat: Cultural Learnings of America for Make ...	Larry Charles	496	Comedy	14093
3	Harry Potter and the Goblet of Fire	Mike Newell	674	Adventure Fantasy Thriller IMAX	19404
4	Little Miss Sunshine	Jonathan Dayton Valerie Faris	773	Adventure Comedy Drama	20038
5	Snatch	Guy Ritchie	107	Comedy Crime Thriller	26552
6	The Butterfly Effect	Eric Bress J. Mackye Gruber	1954	Drama Sci-Fi Thriller	16342
7	Shutter Island	Martin Scorsese	11324	Drama Mystery Thriller	18594
8	Pan's Labyrinth	Guillermo del Toro	1417	Drama Fantasy Thriller	20509
9	Iron Man	Jon Favreau	1726	Action Adventure Sci-Fi	26682

Figure 10: recommendation for Implicit Feedback model

2. Movie rating predictor

If there is a new movie coming up in the next few days and you really want to know if the movie will be a high rated movie in the future or you are a movie producer and you would like to play around with different casts, crews combinations to best fit your movie expectation. Our movie rating predictor will do the job for you. In order to keep our model current, we only used movies that were released after 2000 and there are 110889 movies in total. From the graph below, you

can see that the movie ratings are kind of normally distributed. It will be very hard to predict the movie ratings purely based on movie metadata since the rating is a subjective score. Therefore, I used TMDB API to get the popularity scores for all the casts, crews, directors and I believe these scores could represent some sort of public opinion. Blockbusters always come with a famous director or actors!

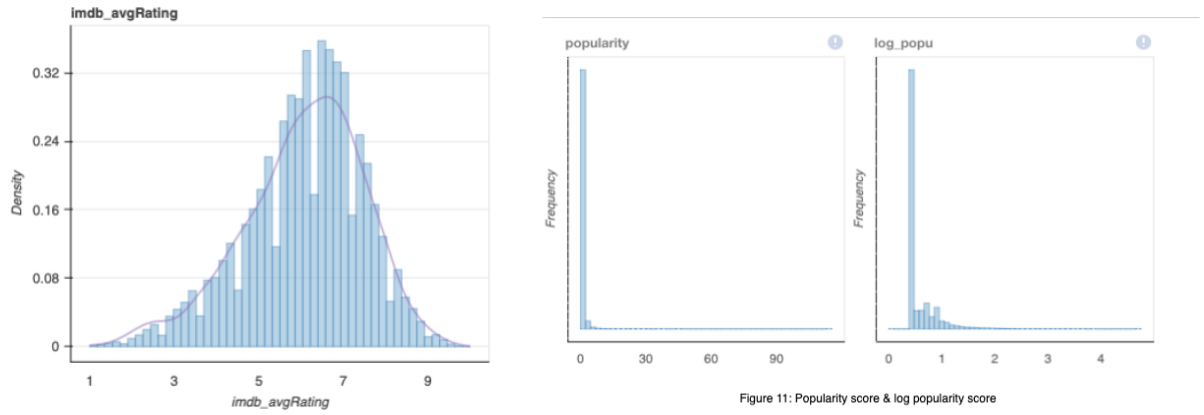


Figure 11: Popularity score & log popularity score

Feature processing:

Since we collected data by ourselves, there are many missing values and categories fields that need to be processed. Firstly, I will go through how to process the casts, crews, directors and writers. Each movie has a list of casts, crews, directors and writers. To be consistent, I only select the top 10 popular casts and crews and calculate their popularity mean respectively. First of all, we classified casts and crews into 3 different classes based on the popularity boundaries. Basically, class, A casts will be in the top 90%, class B will be the top 70% and class C will be remaining. Figure 11 shows the popularity score distribution for casts and you can see that the distribution is very skewed. This makes sense since most casts or crews are not very popular and their popular source is difficult to differentiate from each other. Then log popularity helps us to classify the casts and crews much easier. After that, for each movie, we count how many actors or crews are in class A, class B, class C respectively. The intuition behind this is that well-rated films are always star-studded. From the left graph of Figure 12, we can find that the rating is always above 4 when we have more than 3 class A actors in that movie. One interesting finding is that when we have more than 8 class A actors, the rating is unlikely to be over 8 and could be an average movie too while the movie with only 1 or 2 class A casts could sometimes achieve the highest rating in our dataset. I guess it may be because the public has higher expectations or it is hard to bring out all the popular casts' personalities when there are too many popular casts.

Then, we generated another feature for all casts, crews, directors, writers which count the number of top 10000 and lower 10000 in their list. You can see from the right graph of Figure 12 that these features really help us to identify a good movie or not. With only one top 10k cast, the minimum rating will be 4 and with more than 2 top 10k actors, the lowest rating you can get is 5.

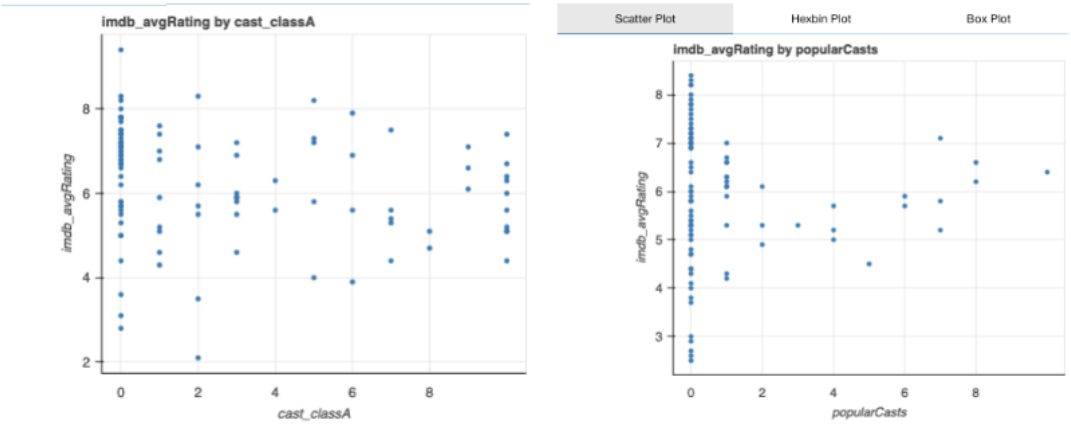


Figure 12: number of popular casts vs ratings

As I mentioned we have many missing values above, we will talk about 3 main ways of handling missing values in our model. Firstly, we used another data source to fill up the missing values. There are 13325 ratings missing in the IMDB database and TMDB is able to fill up 2226 for IMDB ratings. Secondly, we treat the missing values as a category if the missing portion is more than 50%. For example, around 96% of movies do not have homepage information, but we believe it is reasonable to consider the ones without a homepage as one category. Because a large movie producer will have a budget to launch a homepage. Lastly, we used the mean value to fill the missing value based on their category. There are many outliers and missing values for runtime. Firstly, we used the MAD method talked about in the lecture to detect the outliers and then filled up missing runtime and outliers with the mean runtime based on genres.
Still, we did one-hot encoding for some categorical variables (genres, company, etc). In order to avoid exponentially increasing the feature space, I only pick the top 5 frequent categories to do the encoding. You can see from Figure 13, a horror movie is likely to be a low-rated movie while documentary and history movies are likely high rated. I guess it is because documentary movies usually have a sufficient budget.

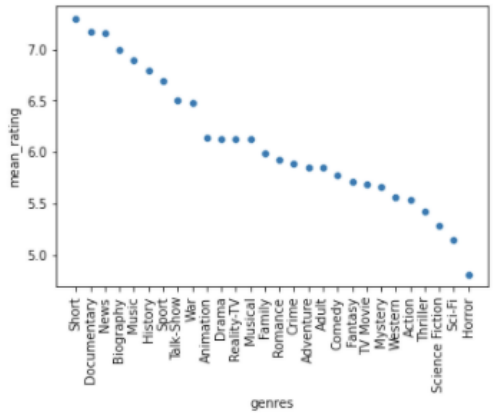


Figure 13: Genres vs Ratings

Feature selection:

We have 60 features in total and there is a greater chance of redundancy in features. The main approach we did for feature selection was RFECV (Recursive Feature Elimination with Cross-Validation). As its name suggested, REFCV will select features by recursively considering smaller and smaller sets of features. The general process of REFCV is the least important predictor(s) are removed, then the model is re-built, and importance scores are computed again. However, this method is expensive to run, and putting all features into REFCV is not reasonable and time-consuming. We first find all the feature pairs that are highly correlated (with correlation coefficient > 0.8) and keep only one of them. However, my features are independent to each other and this action did not reduce the number of features. Secondly, we were using Random Forest in REFCV. Our REFCV will remove one feature at each iteration and the optimal number of features is shown in Figure 14 and we can see that around 10 features will be the optimal number of features. Lastly, we decided to drop the company's related features since they did not help prediction too much. From the feature importance graph (Figure 15)below, it is surprising to see that directors play an important role in rating prediction since I thought casts will be the most important factor.

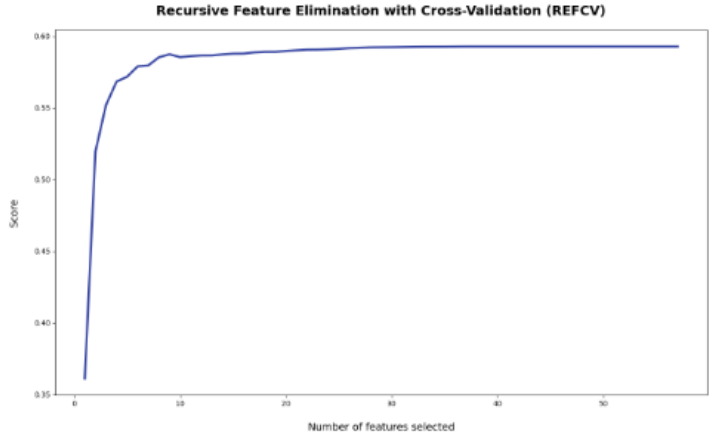


Figure 14: Optimal number of features

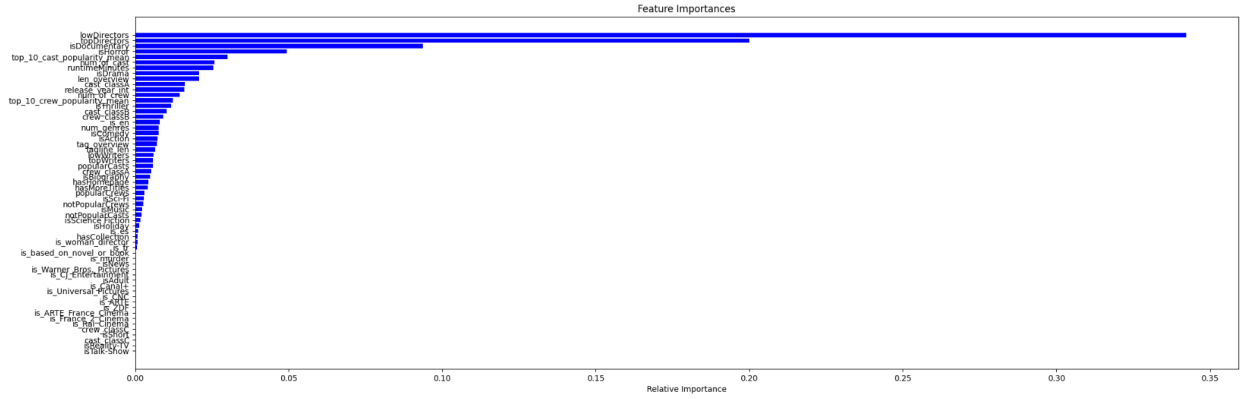


Figure 15: feature importance

Hyperparameter tuning & AutoML:

Lastly, we did a Hyperparameter tuning (random search) and AutoML (TPOT) in order to optimize our model. Both operations are very expensive to run and took us 2 days to finish on an EC2 instance. Both random search and TPOT did not improve R^2 a lot but it did improve the training time.

Evaluation

1. Model evaluation on recommendation system

Content-based:

Since the content-based model makes recommendations based on the content of each movie, we can't evaluate our model based on accuracy. The evaluation method we came up with was to check the train and validation losses for the autoencoder. Since we used an autoencoder to compress the input data into a lower dimension and use the encode layer to make recommendations, it makes sense to use its losses to evaluate our model.

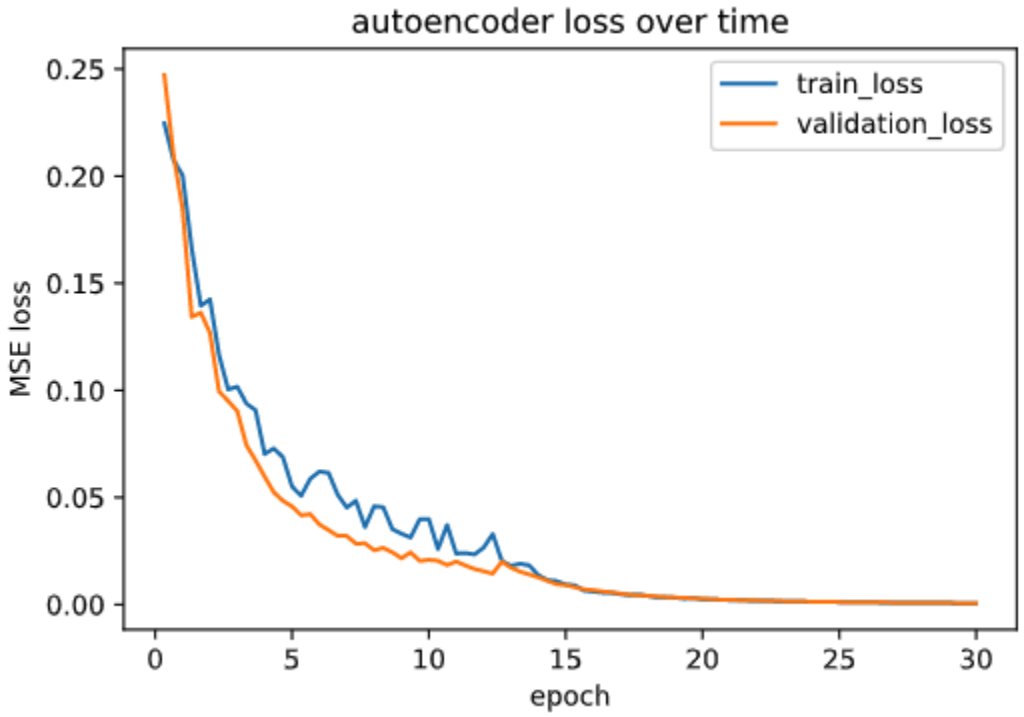


Figure 16: Autoencoder train & validation loss

As we can see, the mean squared error for both train and validation sets is close to 0 after 20 epochs, and there is no sign of overfitting.

Collaborative filtering

Explicit Feedback

For the explicit feedback model, we used binary cross-entropy as our loss function to evaluate our model prediction. Since we applied a sigmoid function at the end of our neural net, our predicted rating is between 0 and 1, binary cross-entropy is a good choice to evaluate our prediction. Our best validation loss is 0.45 which is not that good. We think some possible reasons are:

1. some movies only have few ratings (less than 5) while others have more than 7000 ratings which introduced some imbalance and biases
2. some hyperparameters should be tuned to get better accuracy
3. our neural network is not complicated enough

Implicit Feedback

The evaluation method here is a little different. We first used training loss and accuracy to see our model performance on the training set. As we can see, after 5 epochs, the training loss is 0.12 and the training accuracy is 95.3% which is very good.

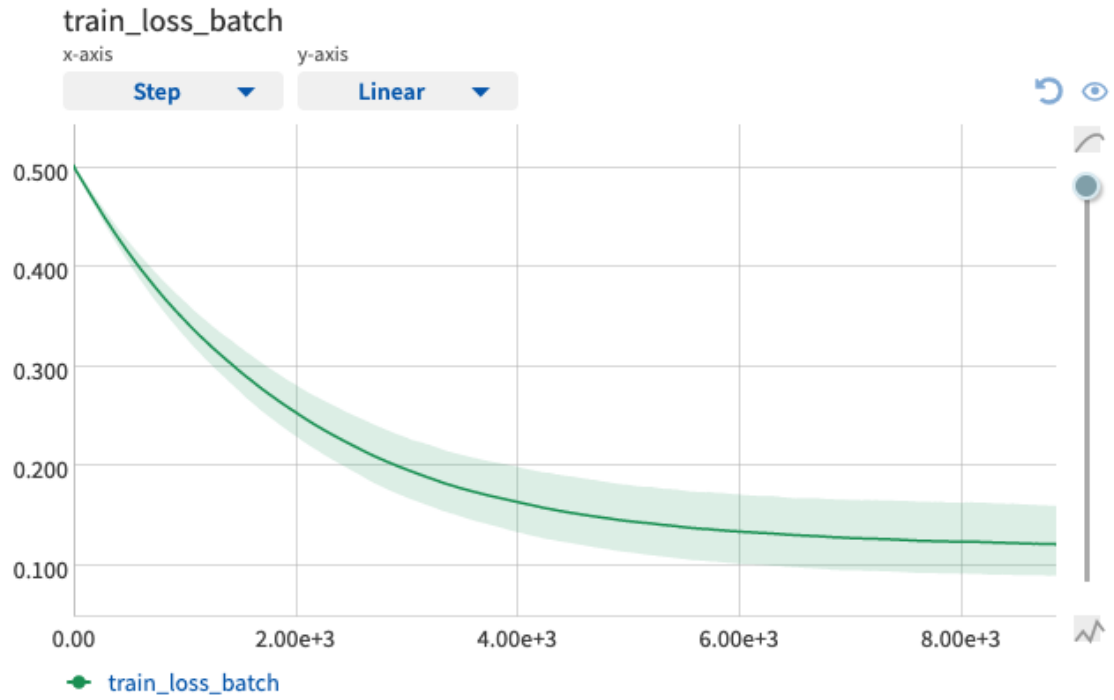


Figure 17: Implicit feedback training loss

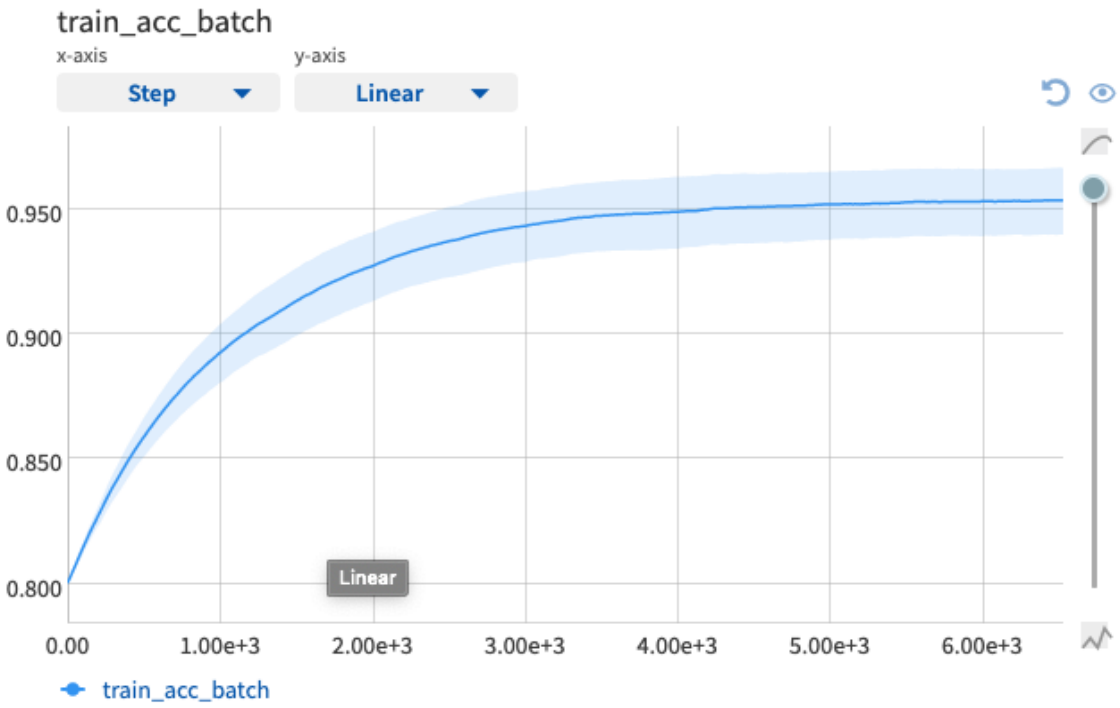


Figure 18: Implicit feedback training accuracy

In addition to this initial evaluation, we used a new method called hit ratio. In order to understand the hit ratio, let first understand how modern recommendation systems are used. For example, Amazon makes recommendations for you based on your viewing histories.

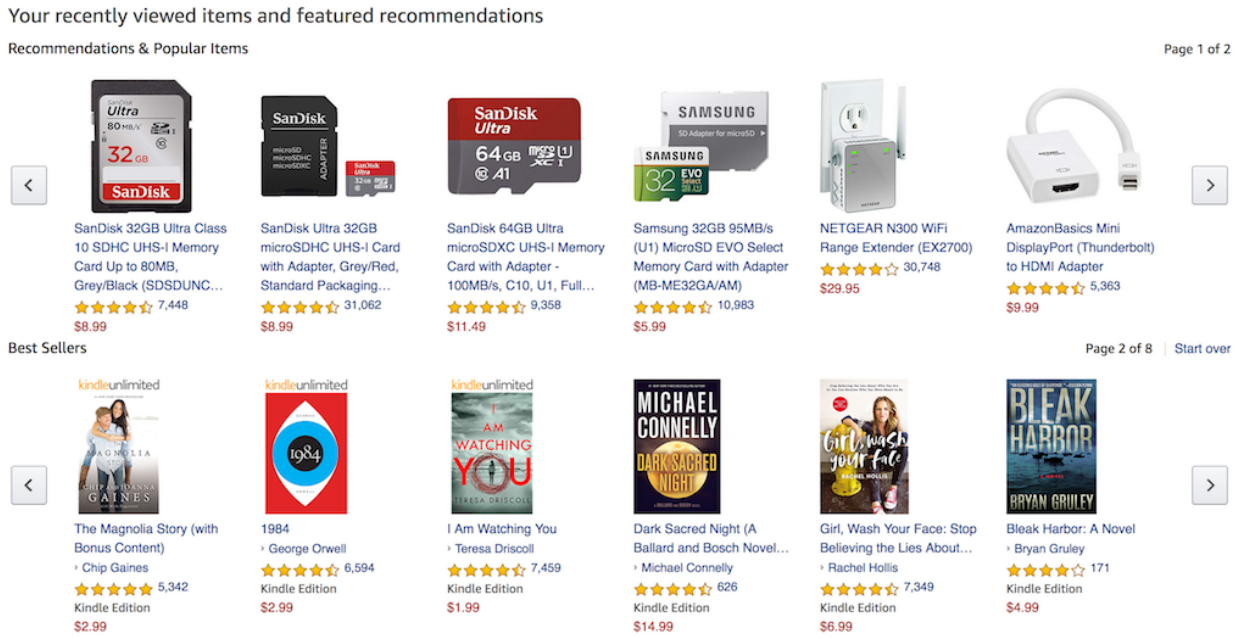


Figure 19: Amazon recommendation

Among the list of recommended items, we wouldn't expect the user to click on all of them. Instead, as long as the user clicks on one of them, our recommendations have worked. To simulate this, for each user in our test set, we randomly select 99 movies that they haven't interacted with and 1 movie that they did. We run our model on these 100 movies and rank them according to their predicted probabilities. Then, we select the top 10 movies from the ranked list and if our test movie (the one the user actually interacted with) is in the top 10 list, then we say that this is a hit. Finally, we get our average hit ratio which is 0.77.

```
print("The Hit Ratio @ 10 is {:.2f}".format(np.average(hits)))
```

```
100%|██████████| 181664/181664 [18:01<00:00, 167.92it/s]The Hit Ratio @ 10 is 0.77
```

Figure 20: Hit ratio

2. Movie rating predictor evaluation

We trained several regression models. XGBRegressor and RandomForestRegressor produced a similarly good result with R^2 value of around 60%. We got R^2 sound 38% if we predicted the rating purely based on the movie information. 38% means we are at least picking something up, but may not be good enough to use for important business decisions. After we gained the popularity score and classified casts, crews, directors into different classes, the R^2 values increased to around 60%. You may notice from the rating distribution that we have too little data for both high ratings (above 8) and low ratings (below 3). The classes are not balanced. As you can see from Figure 21, our prediction just needs to tilt in order to provide a better result. We are wondering if we can make a better prediction with more data in high and low-rated areas. Then we used data from older years (< 2000) and kept rows that are the top 10% highest and bottom 10% of the lowest rating. The R^2value gets improved by 1%. It is not a huge improvement, but you can see from Figure 22 that we did make a better prediction on the high and low rated part.

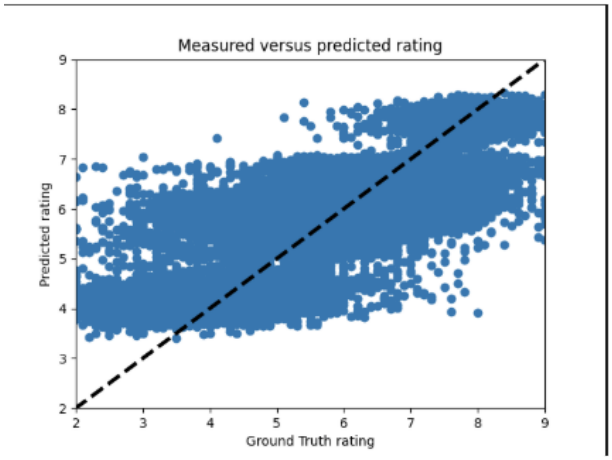


Figure 21: Truth vs Predicted

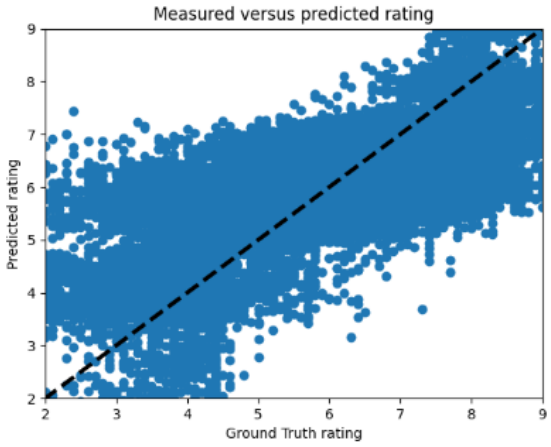


Figure 22: Truth vs Predicted

Data Product

Our data product is a website for people to find good movies that suit their tastes, and a place users can make their predictions for some movies.

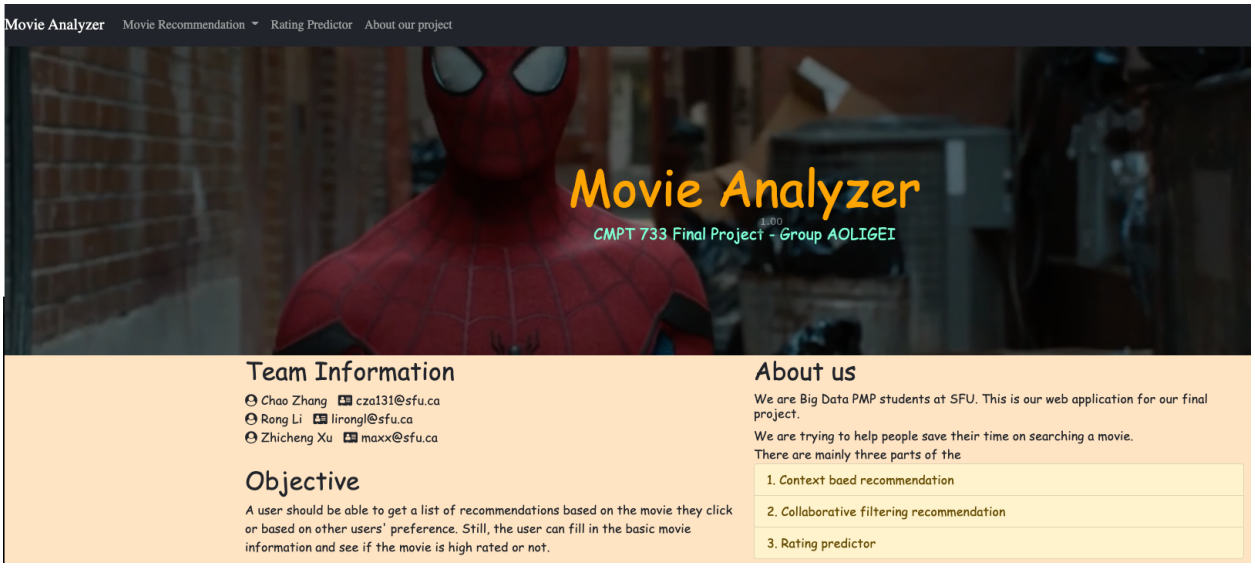


Figure 23: Movie Analyzer info page

Below is the recommendation page where you can click on a movie that you like the most, and a list of similar movies will be generated for you. There are three options you can choose from the dropdown menu, which are content-based, collaborative filtering, and hybrid. Different models will give you a different list of recommendations.

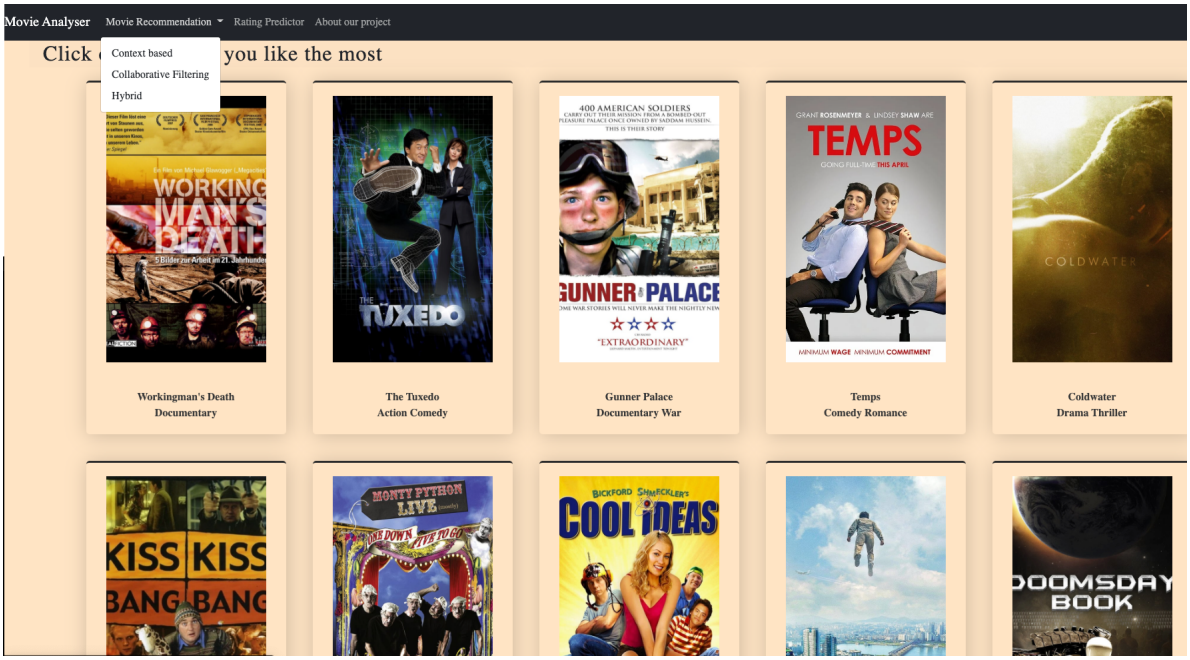


Figure 24: Movie Analyzer recommendation page

The picture below is our movie rating predictor page. It has three functionalities. The first option is to randomly generate a movie where all the movie information will be randomly generated from our database and a predicted rating will be calculated. The second option is similar where we randomly generate a good movie. The third option is to make predictions on a customized movie. Users can enter information such as movie title, actors, directors, genres, etc and a predicted movie rating will be calculated.

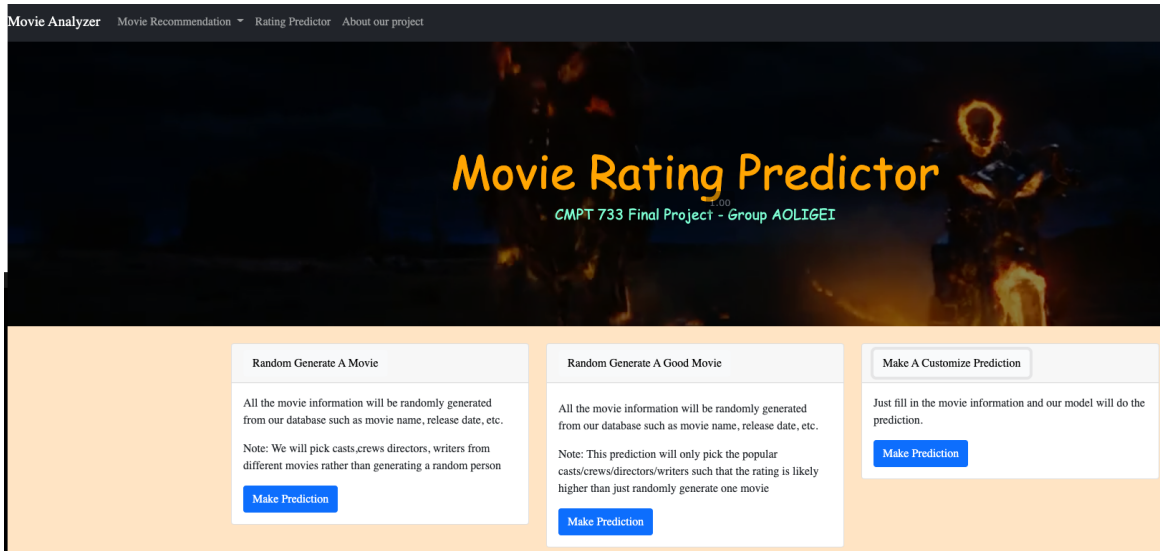


Figure 25: Movie Analyzer predictor page

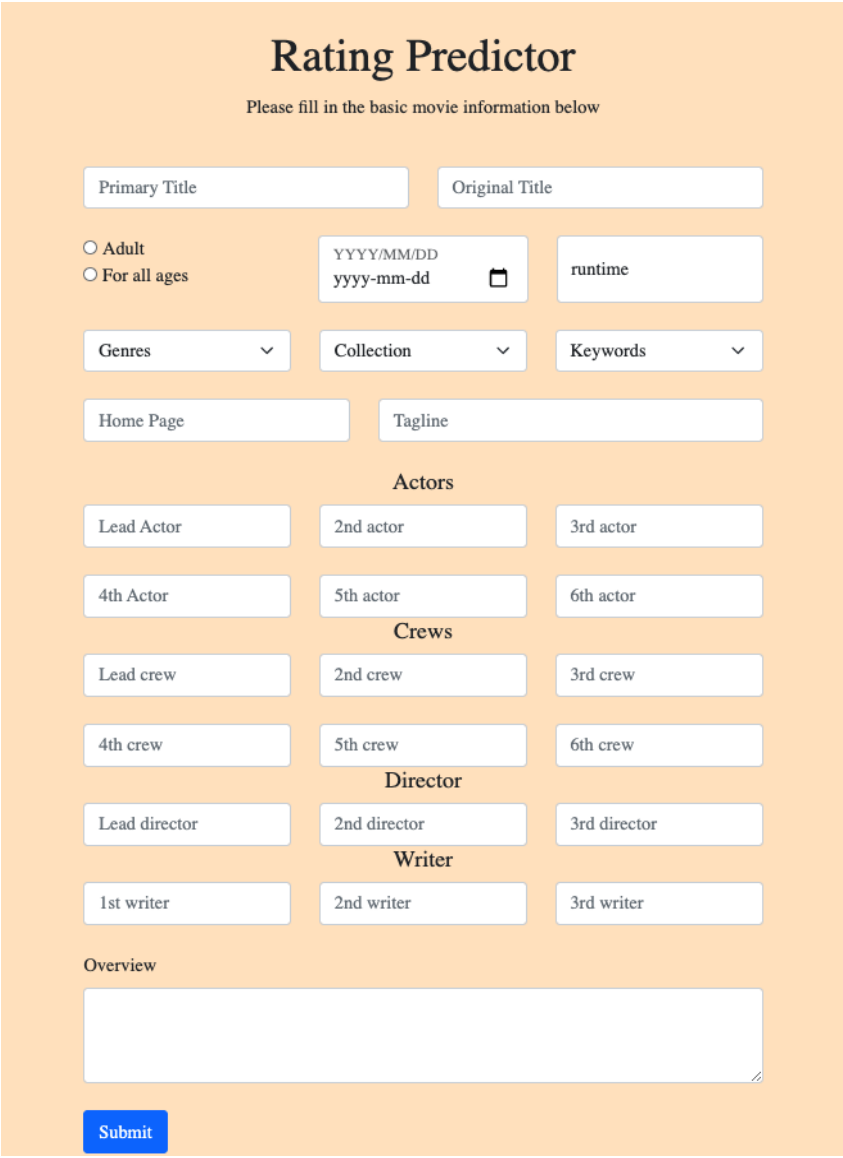


Figure 26: Movie Analyzer customized prediction

Lessons Learned

1. Network travel makes a huge impact on performance speed. During the process of pulling TMDB data through their API, it took us 10 minutes to pull a thousand data points while there were 571841 data points we needed to pull. It was impossible at the point until we tried to run our API script on an EC2 machine that was much closer to a TMDB datacenter. The time it took to pull data through API was reduced significantly by roughly 20 times.

2. Exposing the MongoDB database port to the public is dangerous. As we were serving our MongoDB database in an EC2 machine, we exposed the default 27017 MongoDB port to the public so that we could query that database anywhere we wanted. However, someone hacked our database through the exposed port 27017 and removed our data to ask for a bitcoin payment. Fortunately, we have backup our database snapshot in S3 previously.

3. Data preprocessing plays a vital role in model building. Working with cleaned data will improve model performance, training time, and makes model building much easier.

4. Working with missing values is an important skill to have. We should adopt different methods based on the types of missing values we are dealing with.
5. Public opinions really matter! Rating is a subjective score and the popularity score we got did help us to make a better prediction. We tried to use Twitter or Youtube API but it turned out it is either hard to get data on some older movies or has a request limit cap. We can definitely make a better model If we can find another new feature other than the popularity score.
6. Data balance is also important for model training and imbalanced data may lead to model bias.

Summary

Our project aims to facilitate users to select the movies by giving a list of recommendations based on the movie they clicked or based on similar users' preference, and help people such as movie producers to predict if the new movie rating will meet their expectations. In this project, we have a chance to start from collecting and cleaning data to deploying our application into production. We learned how to centralized data sources into MongoDB, use Pytorch to build deep learning models, play around with AutoML, feature selection, etc. In short, we are able to work through the entire data science workflow in this project. We are excited to go further with it in the future.

Reference

[1]. Datasets.imdbws.com. (n.d.). Retrieved April 16, 2021, from https://datasets.imdbws.com/

[2]. API docs. (n.d.). Retrieved April 16, 2021, from https://developers.themoviedb.org/3/getting-started/introduction

[3]. MovieLens latest datasets. (2021, March 02). Retrieved April 16, 2021, from https://grouplens.org/datasets/movielens/latest/

[4]. Jian Wei, Jianhua He, Kai Chen, Yi Zhou, Zuoyin Tang, Collaborative filtering and deep learning based recommendation system for cold start items, Expert Systems with Applications, Volume 69, 2017, Pages 29-39, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2016.09.040.
[5]. Collaborative Filtering and Embeddings. (Shikhar Gupta, 2017, December 28). Retrieved April 16, 2021, from https://towardsdatascience.com/collaborative-filtering-and-embeddings-part-1-63b00b9739ce
[6]. Deep Learning based Recommender Systems. (James Loy, 2020, October 18). Retrieved April 16, 2021, from https://towardsdatascience.com/deep-learning-based-recommender-systems-3d120201db7e
[7]. Creating a Hybrid Content-Collaborative Movie Recommender Using Deep Learning. (Adam Lineberry, 2018, September 10). Retrieved April 16, 2021, from https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-learning-cc8b431618af
[8]. Recommender Systems in Practice. (Houtao Deng, 2019, February 13). Retrieved April 16, 2021, from https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a
[9]. Figure 19. Retrieved April 16, 2021, https://aws.amazon.com/blogs/aws/amazon-personalize-real-time-personalization-and-recommendation-for-everyone/