
Project Iris: Recommendations Models for Films

Technical Documentation Prepared by:
Win K. Phyo
MS Applied Data Science Candidate '22
Email: wkpwinphyo@gmail.com, win.phyo@mymona.uwi.edu

For:
COMP 6830 - Data Science Capstone Project II
Professor Ricardo Anderson
February 27, 2022

CONTENTS

| | |
|---|----|
| OVERVIEW | 3 |
| DATA DESCRIPTION | 4 |
| 👉 Movie Data | 4 |
| 👉 Emotion Data | 7 |
| EXPLORATORY DATA ANALYSIS/CLEANING & PRE-PROCESSING | 8 |
| 👉 Movie Data | 8 |
| 👉 Emotion Data | 12 |
| MODELS | 14 |
| 👉 Memory-based Collaborative Filtering with KNN | 14 |
| 👉 Model-based Collaborative Filtering SVD | 15 |
| 👉 Emotion Classification Model | 16 |
| DEPLOYMENT | 20 |
| 👉 Deployment Strategy for v1 of App | 20 |
| 👉 Deployment Strategy for v2 of App | 23 |
| REFERENCES | 24 |

OVERVIEW

The original intention of this project was to develop a movie recommendations engine, similar to those employed by Netflix, or YouTube, but with a focus on colour and mood as aspects or features of films that have yet to be explored in the realm of traditional recommendations systems.

In thinking about how to define “colour” in the context of the model, there a variety of approaches that could be taken. We can analyze colour as the average hue of a single frame of a movie, and then express this average hue in terms of blocks of time. For example, we could take the average hues of all the frames that make up the first minute of the movie, and define the average colour of each minute of the movie based on this logic. It is also possible to find the average hue of all the frames throughout the runtime of the film, and then add this average hue as a colour dimension to our movie metadata.

There were attempts to locate sources that could provide any data at all on the subject of colour in films, with some being identified as useful enough to be considered for future projects, most notably [The Colors of Motion](http://thecolorsofmotion.com) (thecolorsofmotion.com) and HappyCoding.io. It quickly became evident, however, that there would be a need to independently acquire movie files and conduct processing and analysis in order to develop a large enough dataset to be considered for this project. As such, it was decided that this aspect of the model would be scrapped.

The mood of a film can similarly be analyzed in a variety of ways. For the scope of this project, the idea was always to utilize a text-based emotion classification system, considering text data gathered from movie overviews/descriptions/plot summaries, and review data published by regular movie-goers as well as film review publications. This data did end up being included in the project, albeit the initial prototype does not use this emotion classification as a feature in the model. This document does describe the intention and plan to incorporate this data in a future release, in order to develop a more robust model that is closer to the original project and business goals.

DATA DESCRIPTION

👉 Movie Data

The movie related data used to build the models can be divided into 2 categories:

1. Movie metadata

The MovieLens 25M dataset, published by GroupLens (<https://grouplens.org/datasets/movielens/25m/>), contains over 25 million rows of ratings data assigned to over 62,000 unique titles. We will specifically be using the *ratings.csv* and *movies.csv* files, with the latter representing movie metadata. The raw structure of the movie metadata can be seen in the sample below:

| movieId | title | genres |
|---------|------------------------------------|---|
| 1 | Toy Story (1995) | Adventure Animation Children Comedy Fantasy |
| 2 | Jumanji (1995) | Adventure Children Fantasy |
| 3 | Grumpier Old Men (1995) | Comedy Romance |
| 4 | Waiting to Exhale (1995) | Comedy Drama Romance |
| 5 | Father of the Bride Part II (1995) | Comedy |

For the purpose of this project, we will not be using the genres data; both because the current prototype does not consider this in calculating recommendations, but also because this data is captured in the IMDb data - which should be a more robust set of metadata that can be worked into future updates to the engine.

The raw IMDb dataset (<https://www.imdb.com/interfaces/>) contains 8,999,075 rows of data; each corresponding to a different film, television episode, or other type of media as described below:

```
array(['short', 'movie', 'tvEpisode', 'tvSeries', 'tvShort', 'tvMovie',
       'tvMiniSeries', 'tvSpecial', 'video', 'videoGame', 'tvPilot'],
      dtype=object)
```

However, because the scope of this project only includes movies, we can already see that a significant number of titles will be dropped from the final dataset. The structure of can be seen in this sample:

| | tconst | titleType | primaryTitle | originalTitle | isAdult | startYear | endYear | runtimeMinutes | genres |
|---|-----------|-----------|------------------------|------------------------|---------|-----------|---------|----------------|--------------------------|
| 0 | tt0000001 | short | Carmencita | Carmencita | 0 | 1894 | \N | 1 | Documentary,Short |
| 1 | tt0000002 | short | Le clown et ses chiens | Le clown et ses chiens | 0 | 1892 | \N | 5 | Animation,Short |
| 2 | tt0000003 | short | Pauvre Pierrot | Pauvre Pierrot | 0 | 1892 | \N | 4 | Animation,Comedy,Romance |
| 3 | tt0000004 | short | Un bon bock | Un bon bock | 0 | 1892 | \N | 12 | Animation,Short |
| 4 | tt0000005 | short | Blacksmith Scene | Blacksmith Scene | 0 | 1893 | \N | 1 | Comedy,Short |

| Feature | Description |
|----------------|---|
| tconst | Unique IMDb ID used to identify the title |
| titleType | The category of content type to which the title belongs |
| primaryTitle | The title of the content as is commonly known |
| originalTitle | The original title of the content, in its original language |
| isAdult | A binary value indicating whether the content is an adult film; 0: non-adult, 1: adult |
| startYear | Corresponds to the year of release |
| endYear | For tv series, this indicates the year the series ended. For movies, this value is \N. |
| runtimeMinutes | The runtime of the content in minutes |
| genres | The genres to which the title belongs |

The primary fields we'll be using from this dataset are *tconst*, *primaryTitle*, and *startYear*. For future models, the remaining features may be useful.

2. Ratings/feedback data

The entirety of this data comes from the `reviews.csv` portion of the MovieLens dataset. This file contains 25,000,095 rows consisting of ratings assigned to a particular movie by a single user, as well as the timestamp at which the rating was submitted. See sample below:

| userId | movieId | rating | timestamp |
|--------|---------|--------|------------|
| 1 | 296 | 5.0 | 1147880044 |
| 1 | 306 | 3.5 | 1147868817 |
| 1 | 307 | 5.0 | 1147868828 |
| 1 | 665 | 5.0 | 1147878820 |
| 1 | 899 | 3.5 | 1147868510 |

| Feature | Description |
|-----------|---|
| userId | Unique MovieLens ID used to identify user associated with rating |
| movieId | Unique MovieLens ID used to identify movie associated with rating. Foreign key of <code>movies.csv</code> . |
| rating | Value from 1.0 to 5.0 with increments of 0.5 corresponding to rating submitted by a user for a particular title |
| timestamp | Seconds since UTC corresponding to submission time of rating |

It may be interesting to analyze how a user's rating of a movie may change overtime using the timestamp dimension, however, for the scope of this project, we will not be using timestamp data.

👉 Emotion Data

The data concerning the training of the text-based emotion classification model can divided into 2 categories:

1. Training data

Though there exists public datasets of full-length movie reviews labeled for sentiment analysis, these datasets exclusively categorize review texts by binary sentiment (i.e. positive or negative). Because the aim of this project is to analyze films in more specific categories of emotion, the training set for our model is based on the Emotions dataset as published by Hugging Face (<https://huggingface.co/datasets/emotion>). This dataset contains 20,000 labelled text strings, each assigned to one of six classes: anger, fear, joy, love, sadness, and surprise. A sample of this dataset is provided below:

| Text | Emotion |
|---|---------|
| im feeling rather rotten so im not very ambiti... | sadness |
| im updating my blog because i feel shitty | sadness |
| i never make her separate from me because i do... | sadness |
| i left with my bouquet of red and yellow tulip... | joy |
| i was feeling a little vain when i did this one | sadness |

2. Movie description/overview data

Movie descriptions/overviews are retrieved from TMDB (<https://www.themoviedb.org/>) using their API. We need the IMDb ID of the desired movie in order to fetch its data; this requires a bridge between the MovieLens and IMDb datasets which we will explore later on. The TMDB API provides a find method to search by external id. See an example call: https://api.themoviedb.org/3/find/tt5726616?api_key=85fc3c49845e51a05cbaadc87fa820a8&language=en-US&external_source=imdb_id

```
▼ movie_results:
  ▼ 0:
    adult:          false
    backdrop_path: "/640tTrXNyLwbqWBcd9JBsJmZb7w.jpg"
    id:            398818
    title:         "Call Me by Your Name"
    original_language: "en"
    original_title: "Call Me by Your Name"
    ▼ overview: "In 1980s Italy, a relationship begins between seventeen-year-old teenage Elio and the older adult man hired as his father's research assistant."
    poster_path: "/mZ4gBdfkhP9tvLH1D04m4HYtiyi.jpg"
    media_type: "movie"
    ▼ genre_ids:
      0:        10749
      1:          18
    popularity: 107.297
    release_date: "2017-09-01"
    video:       false
    vote_average: 8.204
    vote_count:   10447
    person_results: []
    tv_results: []
    tv_episode_results: []
    tv_season_results: []
```

EXPLORATORY DATA ANALYSIS/CLEANING & PRE-PROCESSING

👉 Movie Data

movies.csv contains the full titles (and release year) of all films represented in *ratings.csv*. No null values are present in the *movies.csv* data. We will drop the genre field, leaving us with just *moviedb*, and title. The number of unique values by feature is as follows:

| | |
|----------------|--------------|
| moviedb | 62423 |
| title | 62325 |

The *moviedb* field ranges from 1 to 62,423, while there are 62,325 unique entries in title, meaning there are 98 duplicated movie titles. Upon further inspection, it was found that these duplicate titles refer to the same film but with slightly different values for the associated genres. The inclusion of the year of release in parentheses in the title field removes ambiguity in dealing with films that have identical titles but different years.

ratings.csv comprises the bulk of the data that we will be working with. It is related to *movies.csv* through *moviedb*, and contains 25,000,095 rows of data. As previously stated, we will not be using the timestamp dimension for the purposes of this project. This leaves us with the following number of unique values for each feature:

| | |
|----------------|---------------|
| userId | 162541 |
| moviedb | 59047 |
| rating | 10 |

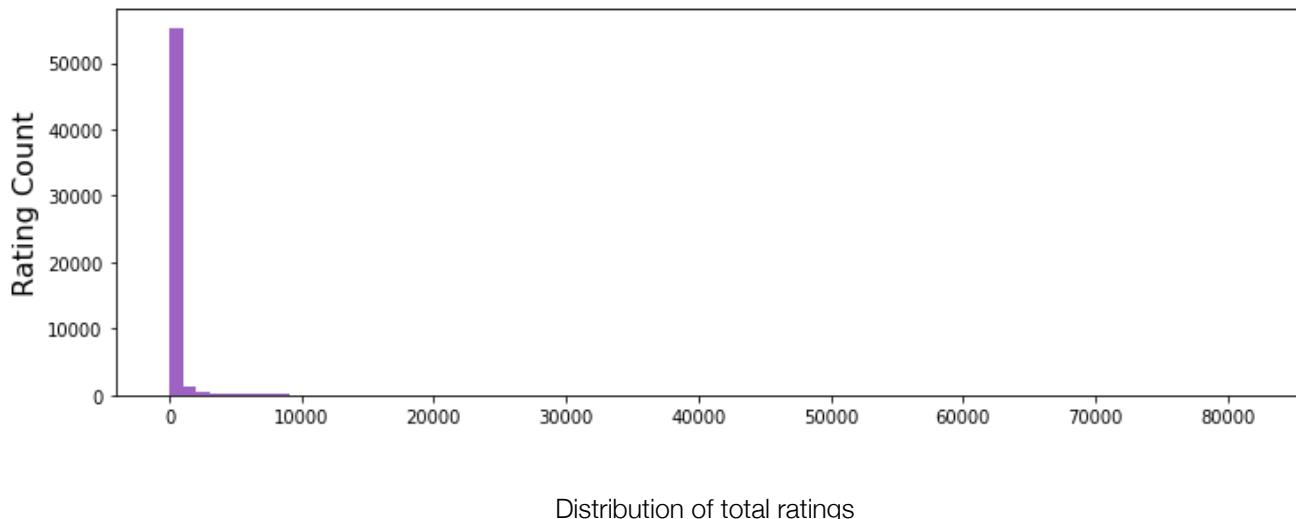
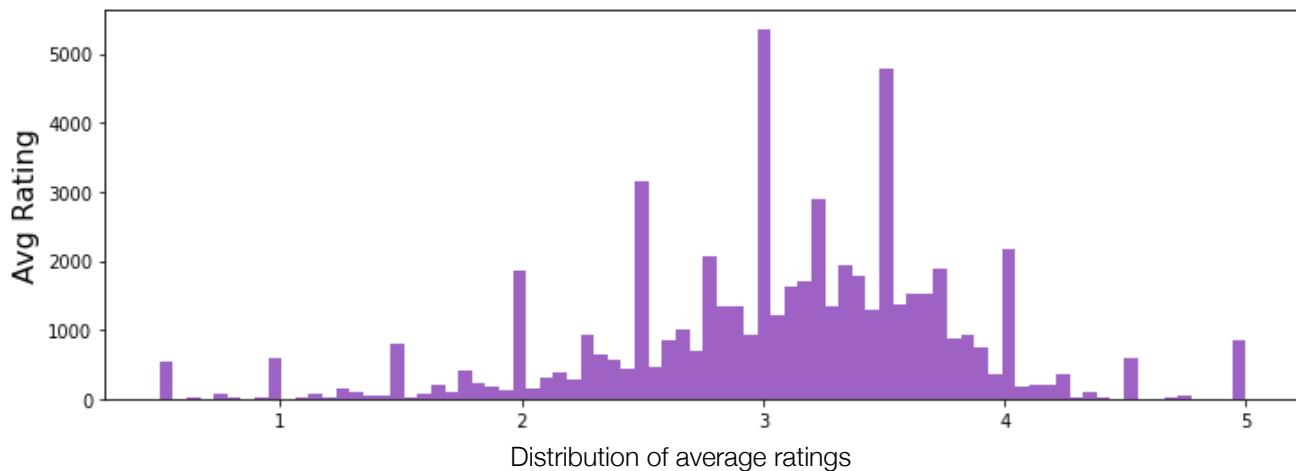
The 10 different classes of ratings include decimal numbers between range 1.0 and 5.0, with 0.5 increments in between values. *moviedb* only returns 59,047 unique values, indicating that there are titles in *movies.csv* that have no ratings.

Perform a right-inner merge of *movies.csv* and *ratings.csv* on *moviedb*.

| userId | moviedb | rating | title |
|---------------|----------------|---------------|--|
| 1 | 296 | 5.0 | Pulp Fiction (1994) |
| 1 | 306 | 3.5 | Three Colors: Red (Trois couleurs: Rouge) (1994) |
| 1 | 307 | 5.0 | Three Colors: Blue (Trois couleurs: Bleu) (1993) |
| 1 | 665 | 5.0 | Underground (1995) |
| 1 | 899 | 3.5 | Singin' in the Rain (1952) |

We continue by finding the average rating, and total number of ratings for each title, and generating a new data frame object as below:

| movielid | AvgRating | CountRating |
|-----------------|------------------|--------------------|
| 1 | 3.893708 | 57309 |
| 2 | 3.251527 | 24228 |
| 3 | 3.142028 | 11804 |
| 4 | 2.853547 | 2523 |
| 5 | 3.058434 | 11714 |



The distribution of average ratings appears fairly normal. The distribution of total ratings or ratings counts, however, is heavily skewed right. In order to effectively deal with noisy data and to truncate the current dataset to a more manageable size, we will only consider those users with at least 100 different rating submissions, and then take a random sample of 30,000 user IDs. These 30,000 user IDs will be used to filter the full ratings dataset.

```

# Calculate how many movies each user has submitted a rating for. This will be
# used later to filter out users with a low number of ratings
user_ratings_df = ratings_df.groupby('userId')['rating'].count().reset_index()
user_ratings_df.rename(columns={'rating':'NoMoviesRated'}, inplace=True)

user_list = user_ratings_df[user_ratings_df.NoMoviesRated>=100]['userId'].
to_list()
user_list = random.sample(user_list, 30000)

```

| userId | NoMoviesRated |
|--------|---------------|
| 1 | 70 |
| 2 | 184 |
| 3 | 656 |
| 4 | 242 |
| 5 | 101 |

Data frame of users and total ratings submitted

We will then only consider those films with at least 100 submitted ratings. After merging the aggregated ratings data with the main ratings data frame, we get something like looks like this:

| userId | movieId | rating | title | AvgRating | CountRating |
|--------|---------|--------|---------------------|-----------|-------------|
| 93864 | 356 | 5.0 | Forrest Gump (1994) | 4.048011 | 81491 |
| 66019 | 356 | 4.0 | Forrest Gump (1994) | 4.048011 | 81491 |
| 4039 | 356 | 4.0 | Forrest Gump (1994) | 4.048011 | 81491 |
| 16649 | 356 | 5.0 | Forrest Gump (1994) | 4.048011 | 81491 |
| 57276 | 356 | 4.5 | Forrest Gump (1994) | 4.048011 | 81491 |

This cleaned data frame represents 10,326 unique titles and 30,000 unique users.

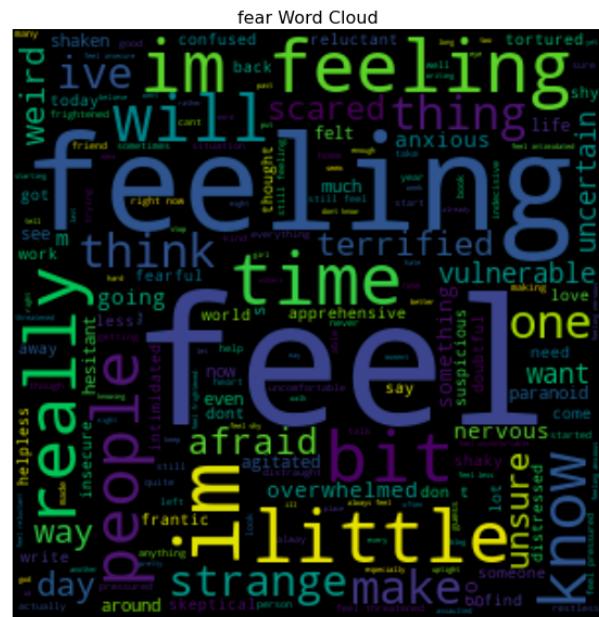
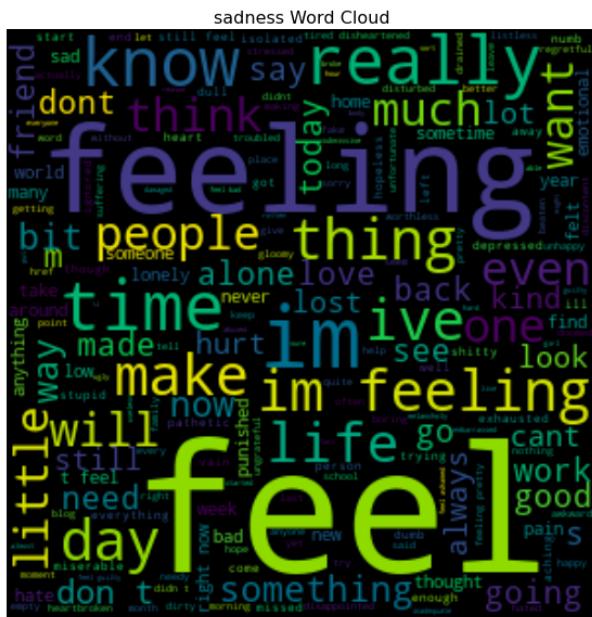
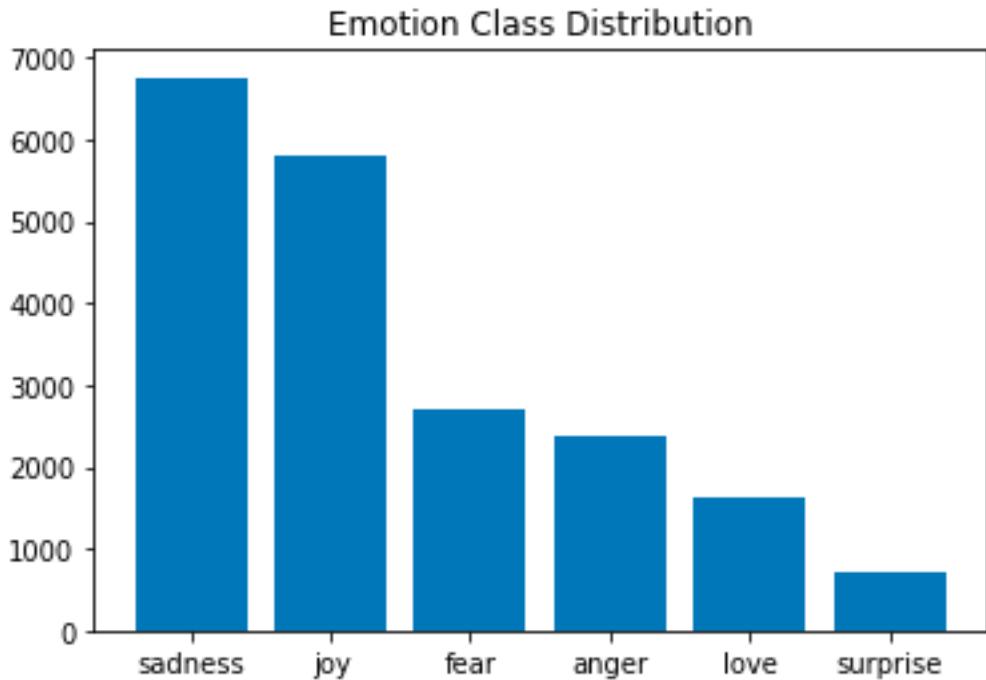
This data frame will then be transformed to produce the following matrix of user IDs, movie IDs, and ratings:

This matrix will eventually be fitted on our K-Nearest Neighbors model. The data frame used to produce the matrix will be used to train our singular value decomposition (SVD) model.

| userId | 13 | 18 | 19 | 23 | 43 | 46 | 69 | 76 | 86 | 89 | ... |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| movield | | | | | | | | | | | |
| 1 | 4.0 | 3.0 | 0.0 | 0.0 | 4.0 | 0.0 | 3.0 | 0.0 | 5.0 | 0.0 | ... |
| 2 | 0.0 | 0.0 | 3.5 | 0.0 | 3.5 | 0.0 | 0.0 | 0.0 | 3.0 | 3.0 | ... |
| 3 | 0.0 | 1.5 | 0.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 | ... |
| 4 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| 5 | 0.0 | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 203519 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| 204352 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| 204542 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| 204698 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |
| 205383 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... |

Emotion Data

The emotion data set consists of 20,000 tweet-sized strings of text with associated emotion labels from the following classes: sadness, joy, fear, anger, love, and surprise. A label encoder object is used to transform these six classes to numerical type for further work.



The word “feel” and its derivatives appear frequently in the data set. We will deal with this noise shortly, but first, we will divide our data into training, validation, and testing subsets. The original file download from Hugging Face already splits the data into these categories in the ratio 16000:2000:2000, but we will go ahead and randomly select data from the full 20,000 records following these ratios for our model.

Working with the Keras library, we will use a tokenizer object to vectorize the training set text corpus. With the assistance of the Natural Language Toolkit (NLTK) and Regular Expressions (RE) libraries, we will clean the texts by removing common stop words, stemming the words to their root forms, etc.

After processing the training, validation, and test text data, the output results in this type of data structure:

```
array([[ 0,  0,  0, ..., 6737, 12724, 5015],
       [ 0,  0,  0, ..., 6484, 3089, 4430],
       [ 0,  0,  0, ..., 11511, 6075, 8546],
       ...,
       [ 0,  0,  0, ..., 6484, 6371, 14260],
       [ 0,  0,  0, ..., 11878, 8186, 13585],
       [ 0,  0,  0, ..., 9706, 2504, 14504]], dtype=int32)
```

Each line or nested array represent cleaned text strings.

MODELS

3 separate models will be built; 2 based on the ratings data, and 1 based on the emotions data. The intention is for the final recommendation engine to incorporate suggestions from all 3 models. However, the initial prototype deployment does not have a robust recommendations system that is built on the emotion data. We will first discuss the 3 models and then outline how they have been brought together in the initial prototype, and how they are planned to be used in future versions of the product.

👉 Memory-based Collaborative Filtering with KNN

There are two broad categories of recommendations systems: content-based, and collaborative-based. The former returns suggested content based on the similarity of the content itself to other pieces of content, whereas the latter suggests content based on what users with similar preferences have consumed. This project is built on user ratings/preferences data, meaning we are using the collaborative approach.

Memory-based approaches use simple measures such as cosine similarity to calculate the distance between two vectors. In this context, our KNN model will utilize cosine similarity to calculate the distance between a given movie and all other movies in the data set based on the ratings users have submitted for each movie. We take the matrix of user IDs, movie IDs, and ratings, and transform the data frame object to a scipy sparse matrix. The KNN object is instantiated, and then fitted with the sparse matrix. We can then pass an example movie title to the kneighbors method of the model, specify the number of nearest neighbors (or recommendations) we want, and it returns those movies that are shortest in distance to the inputted movie title.

```
recommend('inception')

Inception (2010)

      Title    Distance
0  Dark Knight, The (2008)  0.217647
1        Interstellar (2014)  0.277964
2  Dark Knight Rises, The (2012)  0.278636
3    Inglourious Basterds (2009)  0.282665
4       Shutter Island (2010)  0.287717
```

👉 Model-based Collaborative Filtering SVD

In the model-based approach, we use an SVD algorithm to predict the ratings a particular user would assign to a movie based on the data set of ratings, movie IDs, and user IDs. The Surprise library provides easy-to-use algorithms and methods for training and evaluating recommendation system models. We split the data in a 75:25 ratio of training to test, and then instantiate the SVD model object and fit it with the training set. We then test the model by comparing the predicted ratings calculated by the model to actual ratings submitted by users.

```
RMSE: 0.6991  
0.6991493234777875
```

Evaluating the model, we see an RMSE of 0.6991, indicating very good performance.

We define a function to return the top-N recommendations per user based on predicted ratings (in this case, we'll use N=5). Save these recommendations as a pandas series object along with a series for associated user IDs. This new data frame will be saved and loaded during deployment.

| uid | movieId | predictedRating |
|--------|---------|-----------------|
| 118670 | 356 | 5.000000 |
| 118670 | 457 | 4.977576 |
| 118670 | 527 | 4.931338 |
| 118670 | 1246 | 4.869925 |
| 118670 | 1198 | 4.796270 |

👉 Emotion Classification Model

Instantiate a Keras RNN model object and add the following layers and compile:

| Layer (type) | Output Shape | Param # |
|-----------------------------|------------------|---------|
| <hr/> | | |
| embedding (Embedding) | (None, 300, 200) | 3048800 |
| dropout (Dropout) | (None, 300, 200) | 0 |
| lstm (LSTM) | (None, 128) | 168448 |
| dropout_1 (Dropout) | (None, 128) | 0 |
| dense (Dense) | (None, 64) | 8256 |
| dropout_2 (Dropout) | (None, 64) | 0 |
| dense_1 (Dense) | (None, 6) | 390 |
| <hr/> | | |
| Total params: 3,225,894 | | |
| Trainable params: 3,225,894 | | |
| Non-trainable params: 0 | | |

Fit the model with our training sets and the following parameters:

```
callback = EarlyStopping(  
    monitor='val_loss',  
    patience=2,  
    restore_best_weights=True  
)  
  
# Fit model  
history = model.fit(X_train,  
                     y_train,  
                     validation_data=(X_val, y_val),  
                     verbose=1,  
                     batch_size=64,  
                     epochs=10,  
                     callbacks=[callback])
```

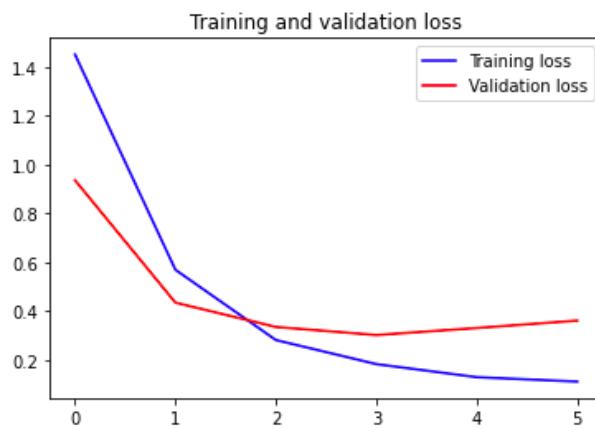
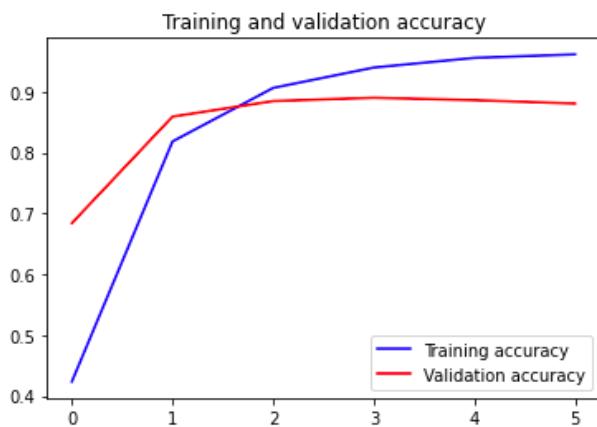
Train the model:

```
Epoch 1/10  
250/250 [=====] - 179s 709ms/step - loss: 1.4526 -  
accuracy: 0.4238 - val_loss: 0.9367 - val_accuracy: 0.6840  
Epoch 2/10  
250/250 [=====] - 177s 709ms/step - loss: 0.5691 -  
accuracy: 0.8182 - val_loss: 0.4344 - val_accuracy: 0.8590  
Epoch 3/10  
250/250 [=====] - 182s 728ms/step - loss: 0.2810 -  
accuracy: 0.9061 - val_loss: 0.3347 - val_accuracy: 0.8845  
Epoch 4/10  
250/250 [=====] - 193s 770ms/step - loss: 0.1822 -  
accuracy: 0.9396 - val_loss: 0.3016 - val_accuracy: 0.8900  
Epoch 5/10  
250/250 [=====] - 214s 855ms/step - loss: 0.1287 -  
accuracy: 0.9555 - val_loss: 0.3303 - val_accuracy: 0.8860  
Epoch 6/10  
250/250 [=====] - 205s 821ms/step - loss: 0.1107 -  
accuracy: 0.9613 - val_loss: 0.3608 - val_accuracy: 0.8805
```

After 6 epochs, we stop as the validation accuracy has declined twice since the 4th epoch. We keep the best weights, and evaluate the model against the validation and test sets:

```
model.evaluate(X_val, y_val, verbose=1)
✓ 11.7s
63/63 [=====] - 11s 177ms/step - loss: 0.3016 -
accuracy: 0.8900
[0.30155712366104126, 0.8899999856948853]

model.evaluate(X_test, y_test, verbose=1)💡
✓ 11.8s
63/63 [=====] - 12s 183ms/step - loss: 0.2763 -
accuracy: 0.9000
[0.2763170003890991, 0.8999999761581421]
```



Testing the emotion classification model with sample texts:

```
He was speechles when he found out he was accepted to this new job
1/1 [=====] - 0s 92ms/step
1/1 [=====] - 0s 38ms/step
joy : 0.4717341661453247
```

This is outrageous, how can you talk like that?

```
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 47ms/step
anger : 0.8275076150894165
```

I feel like im all alone in this world

```
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 40ms/step
sadness : 0.989284098148346
```

He is really sweet and caring

```
1/1 [=====] - 0s 61ms/step
1/1 [=====] - 0s 42ms/step
love : 0.7478350400924683
```

DEPLOYMENT

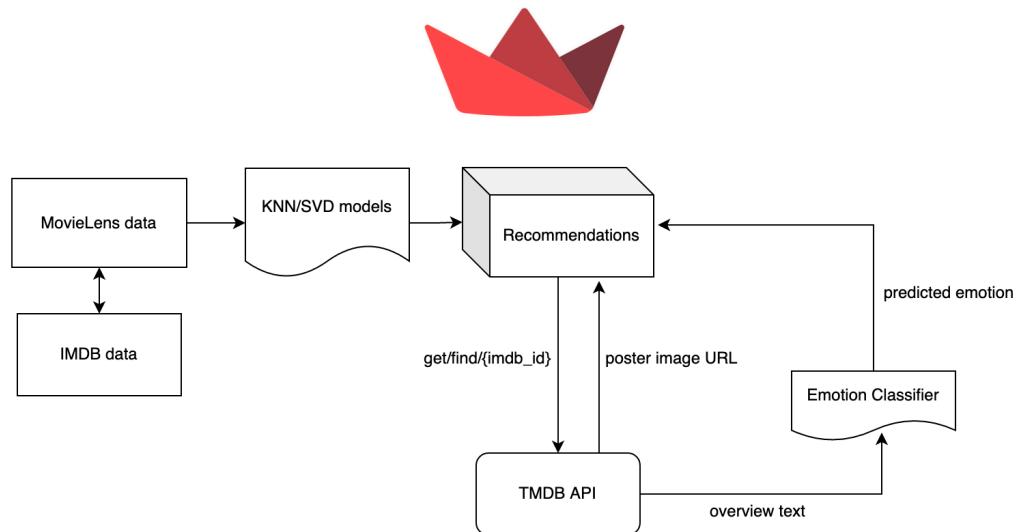
👉 Deployment Strategy for v1 of App

The v1 prototype of the movie recommendations app will host both the KNN and SVD model on separate web pages. We will take advantage of the Streamlit library to build out the front-end for our app. We need to save a few important objects relating to each of the 3 models:

- ▶ For the memory-based KNN approach, we need the model itself, the scipy sparse matrix object, and the main data frame with the aggregated ratings, movie title, and movie ID.
- ▶ For the model-based SVD approach, we save the top 10 predictions for each user in a data frame object, and then load this data frame when running the app. This is a good-enough solution at present, as it does not appear to introduce any significant strain on loading times or resource usage.
- ▶ Additionally, for the emotion classification model we need to load the model itself, as well download the NLTK English stop words

For both ratings-based models, we will need to define a relationship between the MovieLens proprietary movie ID and the corresponding IMDb ID for suggested titles. The IMDb ID is a parameter for the call to the TMBD API using the `get/find/{external_id}` method (<https://developers.themoviedb.org/3/find/find-by-id>). This response object contains the overview text and URL for the associated movie poster. The IMDb dataset contains the IMDb ID that we need, the title of the movie, as well the year of release. It is necessary to produce a clean text string that includes the year of release in a format that is consistent across both IMDb and MovieLens data. Following this, we can merge on movie title in order to get the IMDb ID needed for the API call.

This API call and subsequent application of the emotion classifier to the overview text occurs after the app user submits input. For the memory-based KNN, the input is a movie title (chosen from a drop-down list of the 10,326 unique movie titles after cleaning the dataset). For the SVD model, the input is a user ID of one of the MovieLens users with at least 100 ratings submitted.



Movie Recommendation System

Built by: Win Phy

Type or select a movie from the dropdown

Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001) ▾

Show me recommendations based on what other people are watching 😊

Main page of KNN-model based recommendation system. The model finds those titles with the lowest distance between them and the user-submitted movie in terms of cosine similarity, and returns the top 5. The IMDb IDs of the resulting recommendations are passed to the TMDB API call; from the response, the URL to the movie poster and overview text is extracted and the emotion classification function applied to it. The final result are 5 movies, posters, overviews, and the predicted emotion from the overview text.

Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)

Show me recommendations based on what other people are watching 😊

Harry Potter and the Chamber of Secrets | Harry Potter and the Prisoner of Azkaban | Harry Potter and the Goblet of Fire | The Lord of the Rings: The Fellowship of the Ring | Harry Potter and the Order of the Phoenix



Cars fly, trees fight back, and a mysterious house-elf comes to warn Harry Potter at the start of his second year at Hogwarts. Adventure and danger await when bloody writing on a wall announces: The Chamber Of Secrets Has Been Opened. To save Hogwarts will require all of Harry, Ron and Hermione's magical abilities and courage.

Emotion: anger

Year three at Hogwarts means new fun and challenges as Harry learns the delicate art of approaching a Hippogriff, transforming shape-shifting Boggarts into hilarity and even turning back time. But the term also brings danger: soul-sucking Dementors hover over the school, an ally of the accursed He-Who-Cannot-Be-Named lurks within the castle walls, and fearsome wizard Sirius Black escapes Azkaban. And Harry will confront them all.

Emotion: fear

When Harry Potter's name emerges from the Goblet of Fire, he becomes a competitor in a grueling battle for glory among three wizarding schools—the Triwizard Tournament. But since Harry never submitted his name for the Tournament, who did? Now Harry must confront a deadly dragon, fierce water demons and an enchanted maze only to find himself in the cruel grasp of He Who Must Not Be Named.

Emotion: fear

Young hobbit Frodo Baggins, after inheriting a mysterious ring from his uncle Bilbo, must leave his home in order to keep it from falling into the hands of its evil creator. Along the way, a fellowship is formed to protect the ringbearer and make sure that the ring arrives at its final destination: Mt. Doom, the only place where it can be destroyed.

Emotion: sadness

Returning for his fifth year of study at Hogwarts, Harry is stunned to find that his warnings about the return of Lord Voldemort have been ignored. Left with no choice, Harry takes matters into his own hands, training a small group of students to defend themselves against the dark arts.

Emotion: fear

Built by: Win Phy

Select a user to see their top 5 recommended movies

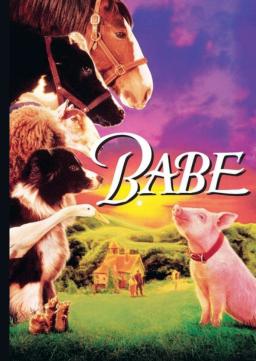
84872

Terminator 2: Judgment Day (1991)



TERMINATOR 2
JUDGMENT DAY

Babe (1995)



Babe is a little pig who doesn't quite know his place in the world. With a bunch of odd friends, like Ferdinand the duck who thinks he is a rooster and Fly the dog he calls mum, Babe realises that he has the makings to become the greatest sheep pig of all time, and Farmer Hogget knows it. With the help of the sheep dogs, Babe learns that a pig can be anything that he wants to be.

Emotion: anger

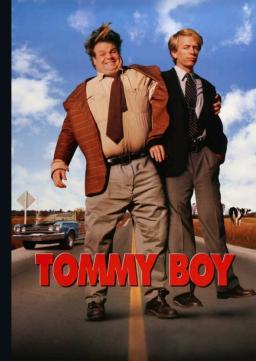
Stargate (1994)



An interstellar teleportation device, found in Egypt, leads to a planet with humans resembling ancient Egyptians who worship the god Ra.

Emotion: sadness

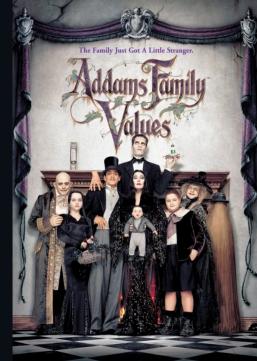
Tommy Boy (1995)



To save the family business, two ne'er-do-well traveling salesmen hit the road with disastrously funny consequences.

Emotion: fear

Addams Family Values (1993)



Siblings Wednesday and Pugsley Addams will stop at nothing to get rid of Pubert, the new baby boy adored by parents Gomez and Morticia. Things go from bad to worse when the new "black widow" nanny, Debbie Jellinsky, launches her plan to add Fester to her collection of dead husbands.

Emotion: surprise

Nearly 10 years have passed since Sarah Connor was targeted for termination by a cyborg from the future. Now her son, John, the future leader of the resistance, is the target for a newer, more deadly terminator. Once again, the resistance has managed to send a protector back to attempt to save John and his mother Sarah.

Emotion: fear

Emotion: sadness

Main page of the SVD-model based recommendation system. The model takes a particular user ID as input and returns the top 5 movies with the highest predicted ratings for that user. In exactly the same process as above, the app makes requests to the TMBD API to fetch overview and movie poster data, and the emotion classifier is applied to the overview text.

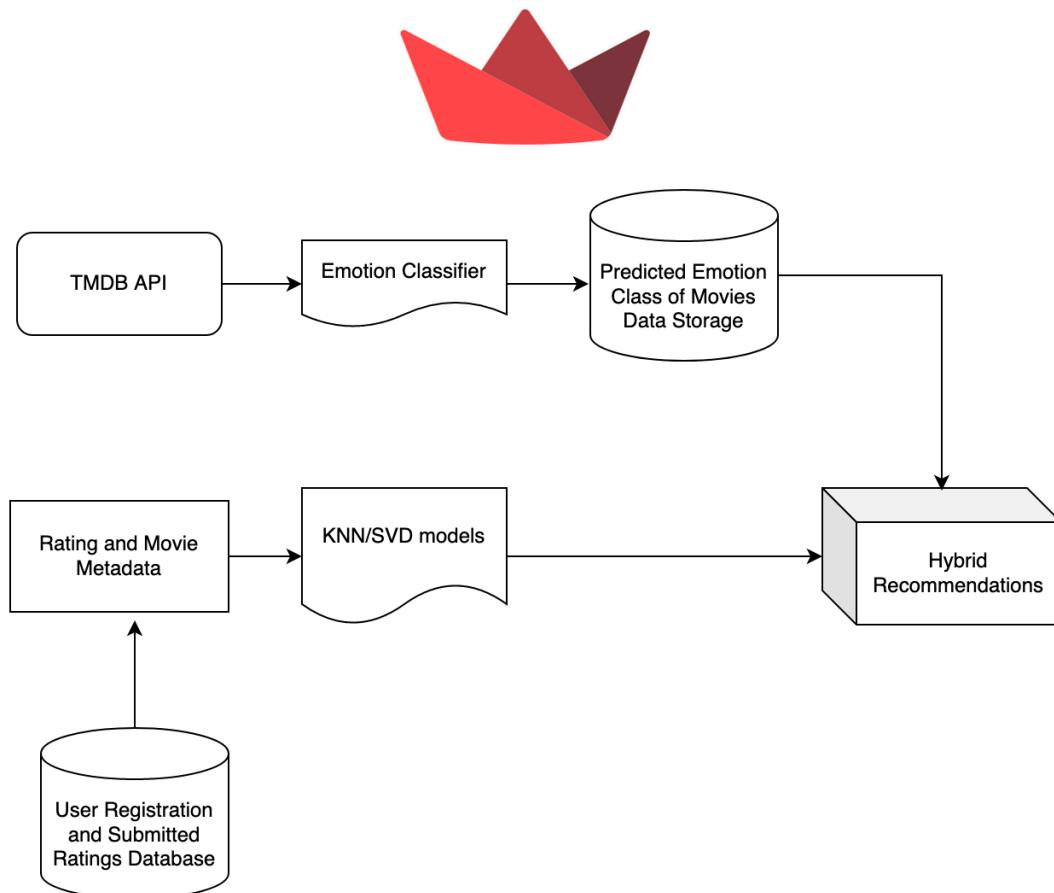
👉 Deployment Strategy for v2 of App

The primary flaws of the current prototype version of the app are: its failure to adequately incorporate emotion data in the recommendations themselves, instead simply providing the user with a predicted emotion of the recommendations generated solely through ratings data; and the inability to add new data points in the form of new users or new ratings submissions.

To address the first issue, it is planned to build a hybrid recommendation engine that returns recommendations from the ratings-based models as well as a new emotion-based model. This new model will be built using the overview texts from the TMBD API response, and the current emotion classifier. The idea is to provide content suggestions entirely based on similarity in terms of predicted emotion class. As the current model is capable of providing both a class and a probability, the simplest deployment of this would be to, given a movie title:

- ▶ find those movies of the same class,
- ▶ among those movies, select those with the closest probability to that of the given movie.

There also is a need to build out a registration feature. For this to be possible, we need to provision a database that holds user data including ratings, and allow the user to submit ratings for films.



REFERENCES

The Colors of Motion - thecolorsofmotion.com

HappyCoding.io – happyCoding.io

Internet Movie Database (IMDb) – imdb.com

The Movie Database (TMDB) – themoviedb.org

GroupLens MovieLens 25M Dataset - <https://grouplens.org/datasets/movielens/25m/>

Surprise Library - <https://surpriselib.com/>

Streamlit - <https://streamlit.io/>

GitHub Repository - <https://github.com/wkphyo/capstone-project>