IRE Final Project Report

Team Blank

Team Members:

- Ainesh Sannidhi - 2019101067
- Anirudh Kaushik - 2020111015
- Tanishq Chaudhary - 2019114007

Code Link:

```
https://github.com/inesane/Movie-Recommendation-System
```

Presentation Link

Project Description

Project Number 4: Classification/Regression and Recommendation

Given a set of movie metadata, along with plot summary, predict overall rating of the movie.
Given a set of users rating and a list of movies, predict a user specific rating for the movie.

1. Abstract

The rapid growth of data collection has led to a new era of information. Data is being used to create more efficient systems and this is where Recommendation Systems come into play. Recommendation Systems are a type of information filtering system, i.e.

they improve the quality of search results by providing results more relevant to the search query and/or related to the search history of the user.

They are used to predict the rating or preference that a user would give to an item based on their previous searches and the other information that the system has related to the user. Most tech companies implement Recommender Systems to a certain extent, such as Amazon for suggesting products to customers, YouTube, Netflix and Spotify to decide which user-specific content to recommend to increase user retention, and any social media platform such as Instagram, Facebook, etc. which use these systems to recommend pages, accounts and ads.

2. Introduction

This project is broadly separated into 2 main tasks:

1. Predict rating of a movie given the movie's metadata, plot summary and movie reviews (this was our novel improvement)
2. Predicting user-specific rating for a movie given that user's ratings for a limited set of movies.

This project is essentially a Recommender System, where Task 1 is a general recommender system where we do not have any information regarding the user who has provided the search query whereas Task 2 will use Recommender Systems that allow us to get query results more specific to the user based on the information we already have on them from their previous movie reviews.

There are three types of recommender systems that we will look at in this project:

- **Demographic Filtering** - They offer generalized recommendations to every user, based on general movie popularity. The System recommends the same movies to users with similar demographic features. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. This system will be used as a default to give a rating solely based on the information of the movie.
- **Content-Based Filtering** - They suggest similar items based on a particular item. This system uses item metadata, such as descriptions, actors, director, genre, etc.

for movies, to make these recommendations. The general idea behind these recommender systems is that if a person liked a particular item, they will also like an item that is similar to it.

- **Collaborative Filtering**- This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata, unlike their content-based counterparts.

3. Literature Review (Related Work)

Matrix Factorization for Collaborative Prediction is a method that uses matrix factorization (multiplying two types of entities to get latent features) in order to determine the relationship between the information of users and the item we are relating this information with (movie ratings in this case). These latent features are in turn used to find the similarity between users and make predictions. [1]

Similarity-Based Collaborative Filtering Model for Movie Recommendation Systems is another method that was introduced in order to find similarities between multiple users' reviews for the purpose of Collaborative Filtering. Various similarity metrics such as Pearson correlation, Euclidean distance, cosine similarity and Jaccard distance were tested in this paper in order to determine the closeness between the ratings of users. [2]

The paper, "Attention is all you need." by Vaswani, Ashish, et al. was a ground-breaking paper introducing the transformer model. This model improved upon a lot of the issues of the previous models like LSTM, which suffered from vanishing gradients and long training times. The transformer model parallelized and thus reduced the training time by a lot - allowing a lot more data to be trained. [3]

Soon after, the paper, "Bert: Pre-training of deep bidirectional transformers for language understanding." was released, which was a pre-trained model with bi-directional encodings, allowing for denser meaning representations. BERT has now become the SOTA for any task requiring vector representations. [4]

4. Implementation (Baseline Methodologies, Architecture, Evaluation and Results)

4.1. Task 1: Movie Rating Prediction

4.1.0. Feature Engineering

- Our dataset consists of 45,000 movies. Additionally, we are provided with a lot of information about each movie such as cast, production crew, revenue (amount of money earned by the movie), budget, run time, plot overview language, genre, etc.
- Our task is to figure out which of these features can potentially affect the rating of the movie while the rest of the features can be ignored.
- Thus, we utilise our knowledge and experience to perform feature engineering to optimize model performance and prevent learning of redundant relations.
- The features selected include: Runtime (in minutes), genre (categorical), budget, revenue, cast, and production crew.
- Another problem is that all of these features are of a different type and some amount of preprocessing is required before merging them.

4.1.1. Linguistic Features

Idea:

In the IMDb movie review dataset, one textual feature was present - the plot overview. Our initial hypothesis was that movies that have high ratings, would have interesting plot summaries.

Process:

To explore this idea, we first clean the data. Case folding was done, along with stopword removal. Then, we first used a TF-IDF vectorizer to convert the text data to a vector representation. Using TF-IDF allows us to look at the weighted importance of each word, thus allowing for a robust statistical analysis. The final vector length was 20,978.

Experiment:

Using this vector for each movie, we passed this vector through a feed-forward-neural network. After hours of fine-tuning attempts, we are able to produce a R2 score of -22.68. R2 score is a measure of how good a regression model is, giving a score between 0 and 1. A model giving the expected value of the dataset (the mean) gives a score of 0.

Discussion:

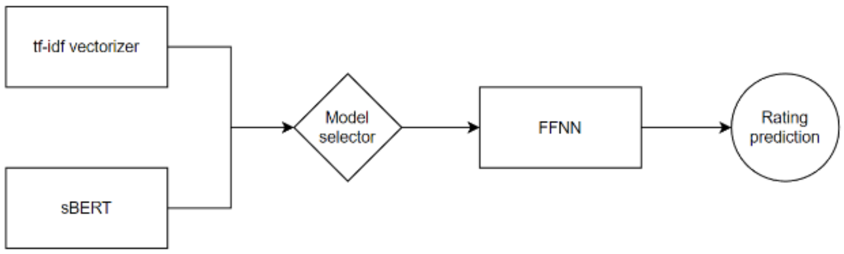
This clearly proved that our model was performing poorly. We thought it was because the TF-IDF vectorizer is not able to capture dense semantic knowledge, so we experimented further with BERT.

Improvements:

To compare it with another approach, we also used the sBERT model (all-MiniLM-L6-v2) to convert the textual data to dense feature representations. We got a feature vector of 384 dimensions.

Experiment:

After spending even more time fine-tuning the FFNN, we got the R2 score: -19.32.



We compared two different models to convert text to vectors, then applied a FFNN to get the ratings. Both resulted in poor predictions.

Discussion:

At this point, we realised something else is going on. We then looked at the dataset, and found an interesting observation. Consider the two following movie plot overviews:

1. A tax investigator chasing a tax evader stumbles over a series of bloody murders and gets involved in an investigation with a rookie cop despite his boss' orders to stay out of the way.
2. Stone (the Antichrist) becomes President of the European Union and uses his seat of power to dissolve the United Nations and create a one-world government called the World Union.

People have varied opinions on which movie is more interesting than the other. In fact, both the movies are pretty similar in terms of rating - both being < 5 rated on IMDb.

This hints at the bigger issue at hand. The movie overview of any movie is not a good indicator of its rating, simply because it is not the movie summary that makes a movie good. The plot, twists and turns, the characters, the director, cinematography, etc are a few of a multitude of factors that contribute to a movie being rated highly or poorly.

Further improvements:

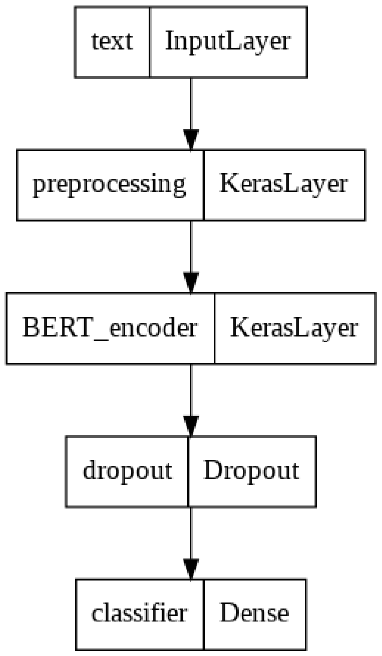
As a further suggestion to this work, we were told to use movie reviews as another feature, instead of the movie overview. To do this, we make use of two more datasets. We use the IMDb movie reviews dataset (present here) and the 400k Rotten Tomatoes dataset (present here).

Experiment:

We used the BERT model (small, uncased), and fine-tuned it using the IMDb movie reviews dataset. It was done for 1 epochs, with 3e-5 learning rate (suggested default), adamw optimizer and binary cross entropy loss. The model achieved an accuracy of 73%.

We thus used the trained model on the Rotten Tomatoes dataset of 400k reviews with movie name and corresponding reviews. Finally, we take the average of the scores for each movie (given my multiple raters) and we use the scores as another feature for the dataset.

We get a 0.9278 score using linear regression at the end. SGD score: 0.9279. For experimentation purposes, we also tried converting the regression problem to classification and got 0.83 and 0.788 scores for logistic regression and support vector classifiers, respectively.



4.1.2. Numerical Features

- **Budget and revenue**
  - The motivation behind their selection was that movies with larger budget and revenue earned, in general had higher ratings.
  - Budget and revenue of the movies were provided in dollars and are of the order of 10^6.
- **Runtime**
  - Movies with very long run times tend to have poor ratings as the audience tends to get tired and bored towards the end.
  - Additionally, very short movies generally do not have enough content to form a coherent plot and thus feel rushed.
- These features are normalized to keep their value in the 0 to 1 range as they have very different scales. (10^2 and 10^6)

4.1.3. Metadata Features

Motivation

- A major factor affecting a movie’s rating is the quality of direction and acting.
- Since no such information is provided in the dataset, we must find an alternative way to extract this data.
- Once again, human knowledge and experience come in handy. We create a list of top actors and directors and create two new features called top_actors and top_directors.
- The list of top actors and directors was taken from the list of top 10 highly rated actors and directors from IMDb.
- If the production crew includes any of the directors present in the above list, we make the top_directors feature 1, otherwise 0. Similarly, if the cast includes any actor present in the top 10 list, we make the top_actors feature 1.
- To avoid individual bias, we make the feature 1 and don’t assign an ordinal score within the list.
- Genre also plays an important role in a movie’s rating. Genres which place the actors in tough real-life situations and struggles such as drama tend to receive higher ratings.
- We created a list of genres of movies which receive the highest rating and corresponding feature, good_genre, similar to the previous case, if a movie’s genre belongs to the list then we make the feature 1, else 0.
- A significant amount of human insight went into the process of feature engineering.

Preliminary Results

- With Acting, Directing and genre information reduced to categorical variables, we were able to improve the r2 score of our linear regressor from 0.19 to 0.279.
- However, this score was still quite low. After taking feedback from the TAs, we tried another approach to capture more information about the performance of the actors and directors in a movie along with the popularity of the genre.

Further Improvements

- Instead of simply checking whether the movie contains a good actor or not, and changing the actor variable from a 0 to a 1 (same for director and genre) we decided to calculate the average rating for the cast and crew in the movie.

- We calculated the rating (vote average) for each actor for every movie they starred in averaged over the total number of movies they appeared in. This ensured we obtained a score between 0 to 10 for every actor and helped deal with cases where a highly rated actor only appears in a few movies but a lowly rated actor can be seen in almost every film.
- To do this, the actor's name was case folded and then we iterated over the cast for every movie to search for the specific actor and calculate his score over all the movies where he is present. We did this efficiently by using a dictionary allowing us to update an entire movie's cast in parallel.
- However, acting information alone is insufficient for rating prediction. So, we decided to do the same for Directors and Genres.
- After obtaining an average rating for every actor, director and genre, we converted this into a meaningful variable as follows. For every movie, we calculated the score of the entire cast by averaging the individual scores for each actor.
- We did the same for the directors and genres, thus reducing information about a movie's acting, directing and theme to 3 numerical values which were then used for predicting ratings.

4.1.4 Feature Normalization

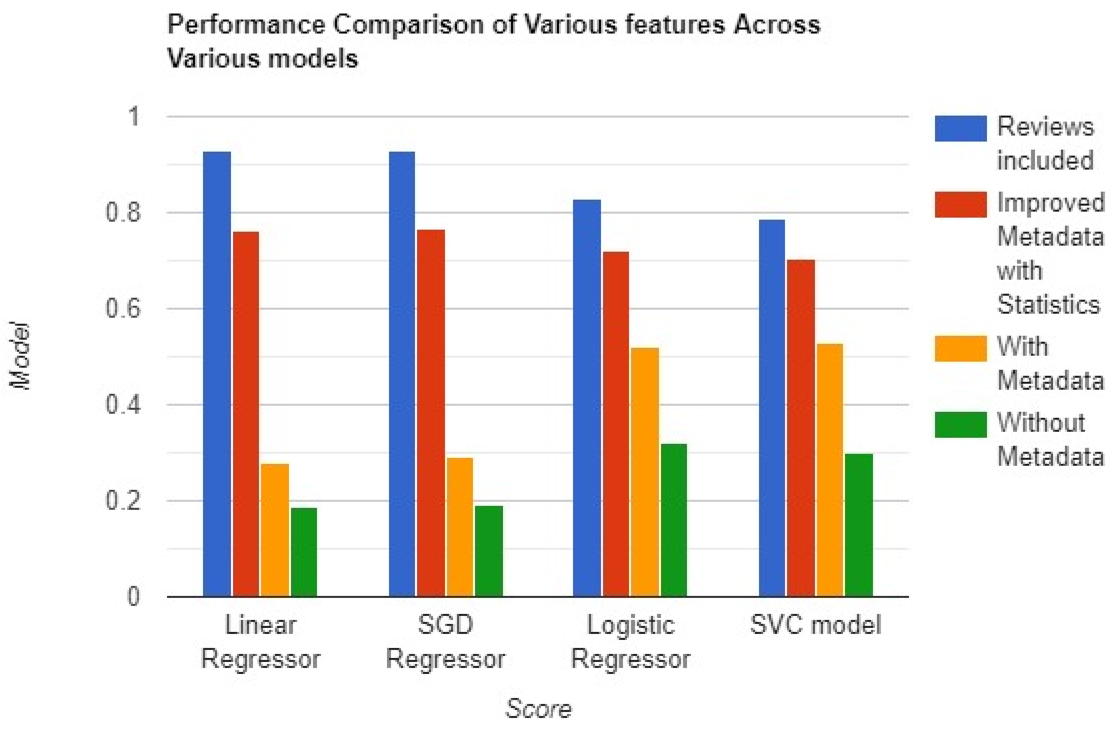
- Since we have various different types of features, ranging from categorical to numerical, in order to merge all of these features and make a prediction, we need to normalize them.
- This is done to prevent one feature from outweighing all others.
- For instance, budget is of the order of 10^6 while good_genre, top_director, and top_actors take a value between 0 to 10
- To this end, scikit_learn standard scaler was used, but the accuracy obtained was very poor.
- We needed the budget and revenue to be scaled together to preserve the relation between them, after which all of the features should be scaled together
- So, we normalize by subtracting the mean from each sample and dividing by the overall max in that category to obtain the new normalized results.

4.1.5 Problems Faced

- Some actors and directors had only appeared in a few movies whose ratings weren’t available (missing values).
- Feature normalization was a challenge since we have various different types features, ranging from categorical to numerical. In order to merge all of these features and make a prediction, they have to be normalized while retaining information between related columns such as budget and revenue.

4.1.6 Final Results

- The results were significantly better than the previous case by a large margin. The R2 score for our linear regressor increased to 0.76 while the score for the SGD regressor was 0.765.
- Thus our model was able to provide good predictions for a movie’s rating using just Numerical features like budget, revenue, runtime and metadata information like Cast, Crew and Genres.



4.2. Task 2: Recommendation System

4.2.1. Demographic Filtering

Idea:

They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. This works as a simple recommender in the case where the user is a new user or we do not have any information about them.

Experiment:

Recommendation is done on the basis of weighted rating (‘WR’) using ‘vote_count’ (number of votes, ‘v’) and ‘vote_average’ (average rating of those votes, ‘R’)

A minimum vote count (‘m’) is introduced so that movies with very low vote counts where vote average would not be an accurate representation are not considered

$$\mathrm{WR} = \left(\frac{v}{v+m} * R\right) + \left(\frac{v}{v+m} * C\right)$$

‘C’ is the mean vote across all movies

This weighted rating is useful as movies with more ratings generally would mean that that rating is more accurate and more people liked it, hence out of two movies with the same average rating the one with more ratings would have a higher weighted rating.

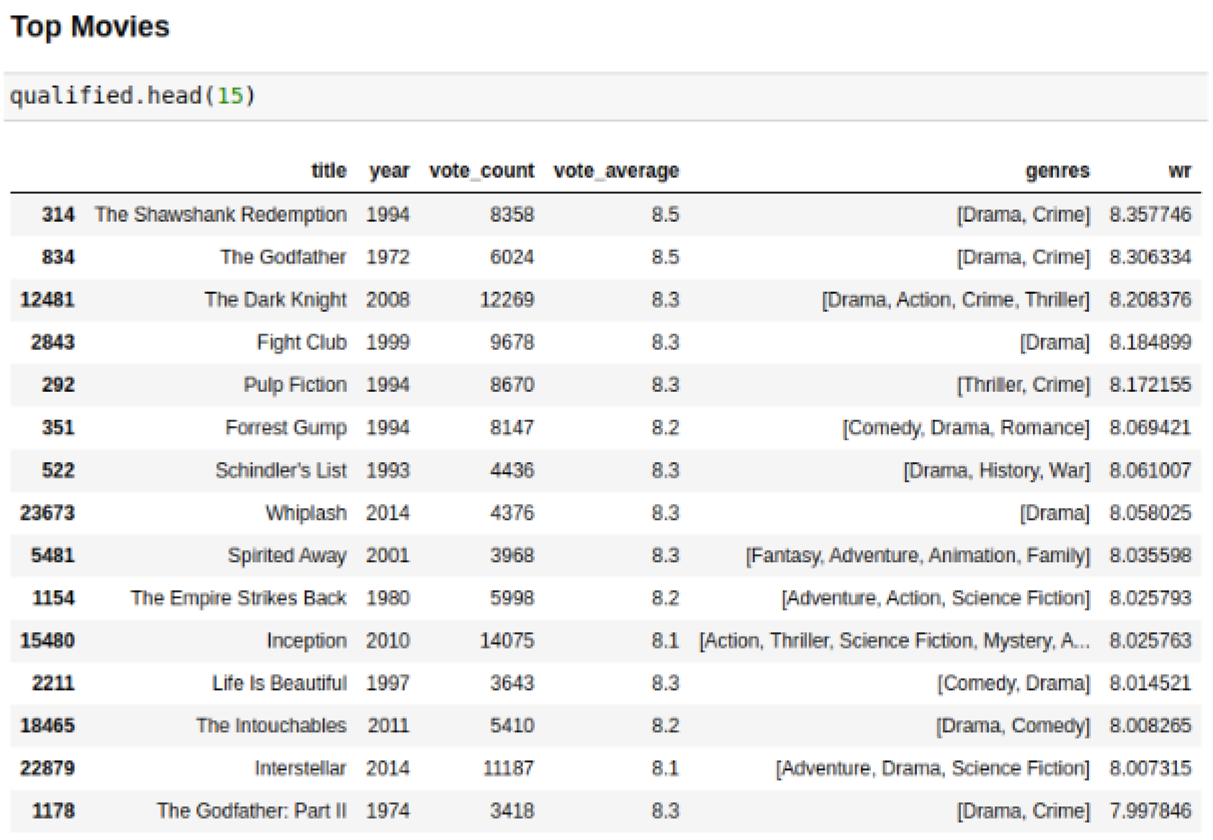
Improvements:

Weights were assigned to the vote count and vote average along with the minimum vote count were changed and tested to see what gave the best-generalised recommendations.

Results:

Top 15 movies according to the general weighted rating system

Here, you can see that some movies with lower vote average are ranked above movies with higher vote average as they have more votes



Top Movies

```
qualified.head(15)
```

	title	year	vote_count	vote_average	genres	wr
314	The Shawshank Redemption	1994	8358	8.5	[Drama, Crime]	8.357746
834	The Godfather	1972	6024	8.5	[Drama, Crime]	8.306334
12481	The Dark Knight	2008	12269	8.3	[Drama, Action, Crime, Thriller]	8.208376
2843	Fight Club	1999	9678	8.3	[Drama]	8.184899
292	Pulp Fiction	1994	8670	8.3	[Thriller, Crime]	8.172155
351	Forrest Gump	1994	8147	8.2	[Comedy, Drama, Romance]	8.069421
522	Schindler's List	1993	4436	8.3	[Drama, History, War]	8.061007
23673	Whiplash	2014	4376	8.3	[Drama]	8.058025
5481	Spirited Away	2001	3968	8.3	[Fantasy, Adventure, Animation, Family]	8.035598
1154	The Empire Strikes Back	1980	5998	8.2	[Adventure, Action, Science Fiction]	8.025793
15480	Inception	2010	14075	8.1	[Action, Thriller, Science Fiction, Mystery, A...	8.025763
2211	Life Is Beautiful	1997	3643	8.3	[Comedy, Drama]	8.014521
18465	The Intouchables	2011	5410	8.2	[Drama, Comedy]	8.008265
22879	Interstellar	2014	11187	8.1	[Adventure, Drama, Science Fiction]	8.007315
1178	The Godfather: Part II	1974	3418	8.3	[Drama, Crime]	7.997846

4.2.2. Collaborative Filtering

Idea:

This system matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata, unlike their content-based counterparts.

Experiment:

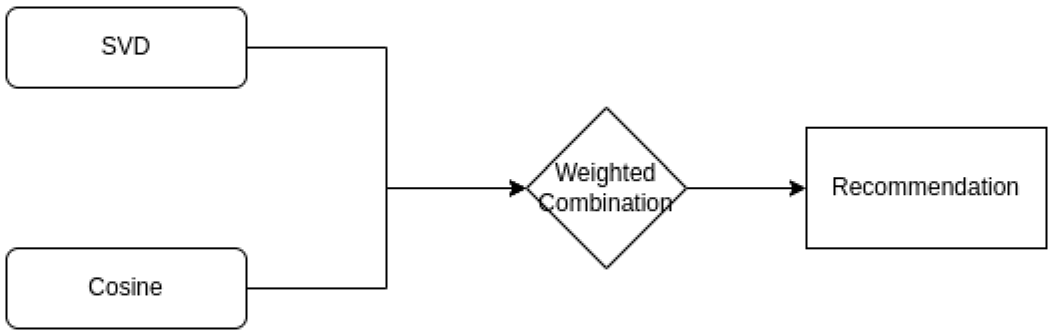
We use Singular Value Decomposition in order to compute the similarity between users and movies by constructing a utility matrix consisting of the rating for each user-movie pair. This is used to predict a rating for every single movie and we rank the movies and give recommendations based on this predicted rating.

Improvements:

Initially, we only used SVD which is the general method, however, later we combined the rating obtained along with cosine similarity between movies on the basis of the metadata features to calculate a rating on the basis of both SVD and cosine similarity to see if this worked better.

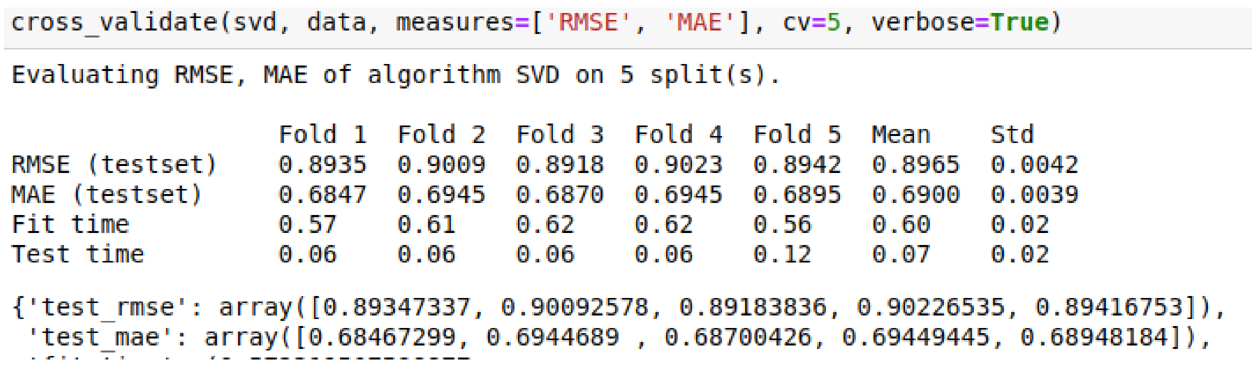
This was tested by assigning rankings to each movie based on how high up their recommendation was, i.e. the top recommended movie would get 1 and so on. A linear combination of the two rankings from SVD and cosine was taken and the movies were ranked again to get the final ranking.

However, this did not perform better and only performed ok when the weight assigned to either SVD or cosine was much higher, i.e. they did not perform well together and did not predict the same/similar outcomes. This is probably due to the fact that we are just directly combining both general and personal factors where both might not necessarily be linked directly.



The evaluation metrics used were Root Mean Square Error and Mean Absolute Error with k-Fold Cross Validation (k=5 in our case)

RMSE is around 90%



```
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

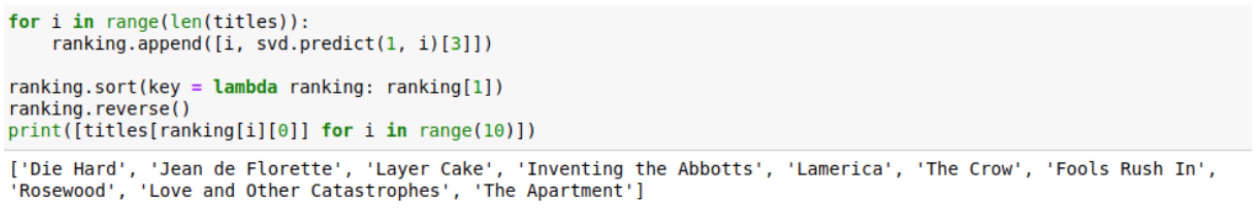
```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8935  0.9009  0.8918  0.9023  0.8942  0.8965  0.0042
MAE (testset)     0.6847  0.6945  0.6870  0.6945  0.6895  0.6900  0.0039
Fit time          0.57    0.61    0.62    0.62    0.56    0.60    0.02
Test time         0.06    0.06    0.06    0.06    0.12    0.07    0.02

{'test_rmse': array([0.89347337, 0.90092578, 0.89183836, 0.90226535, 0.89416753]),
 'test_mae': array([0.68467299, 0.6944689 , 0.68700426, 0.69449445, 0.68948184]),
```

Results:

Here are the top 10 recommendations for a specific user



```
for i in range(len(titles)):
    ranking.append([i, svd.predict(1, i)[3]])

ranking.sort(key = lambda ranking: ranking[1])
ranking.reverse()
print([titles[ranking[i][0]] for i in range(10)])
```

```
['Die Hard', 'Jean de Florette', 'Layer Cake', 'Inventing the Abbotts', 'Lamerica', 'The Crow', 'Fools Rush In',
'Rosewood', 'Love and Other Catastrophes', 'The Apartment']
```

4.2.3. Content-Based Filtering

Idea:

Unlike in the case of movie rating prediction, this time using the plot summary was helpful. This is because movies with similar themes would have higher similarities between their plots.

Experiment:

We compute TF-IDF vectors telling us the keywords that described each movie in the plot description and then compute similarity scores.

Idea:

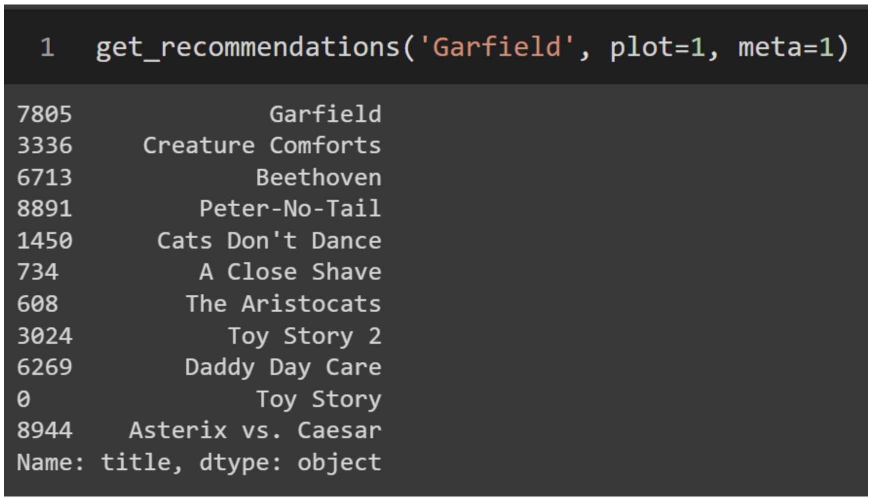
We also parse the data given to us in terms of numerical and categorical features and extract parameters such as movie runtime, actor and director.

Experiment:

To get an accurate statistical measure of the features like actor and director, we use a count vectorizer instead of TF-IDF as we will need to increase the importance of these terms as it is more likely that movies with these parameters in common will be more closely related than if the movies' plots have terms in common.

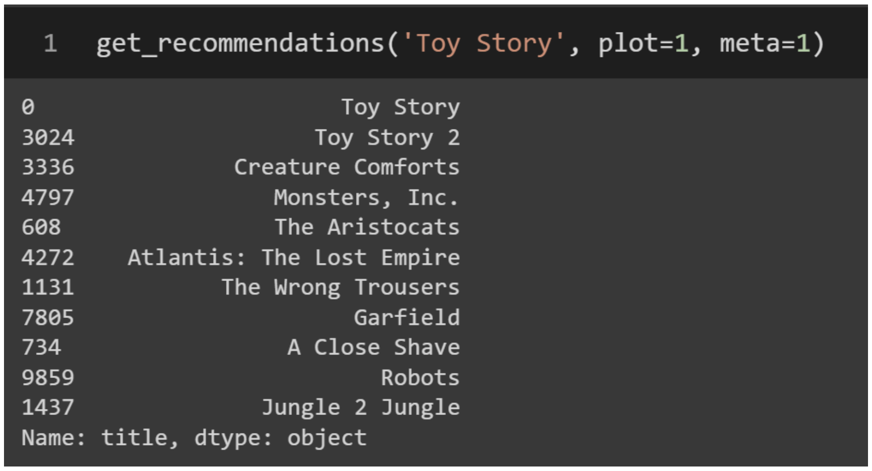
Results:

Finally using a combination of plot overview and metadata, we get the recommendations as follows.



```
get_recommendations('Garfield', plot=1, meta=1)
```

```
7805                Garfield
3336       Creature Comforts
6713               Beethoven
8891           Peter-No-Tail
1450        Cats Don't Dance
734            A Close Shave
608           The Aristocats
3024              Toy Story 2
6269           Daddy Day Care
0                  Toy Story
8944      Asterix vs. Caesar
Name: title, dtype: object
```



```
get_recommendations('Toy Story', plot=1, meta=1)
```

```
0                        Toy Story
3024                   Toy Story 2
3336             Creature Comforts
4797                Monsters, Inc.
608                 The Aristocats
4272    Atlantis: The Lost Empire
1131             The Wrong Trousers
7805                      Garfield
734                  A Close Shave
9859                        Robots
1437               Jungle 2 Jungle
Name: title, dtype: object
```

Conclusion/Analysis

For each of the standard methods, we have tried adding our own improvements on each and tested.

For the movie rating prediction problem, our scores improved from a meager 0.19 to 0.93 (r2 score of linear regressor) with comparable scores across various models. This was done by testing with a variety of features and making improvements based on the feedback provided by the TAs (statistical analysis of actors, directors and genres for better scoring as opposed to a simple categorical variable) and the professor (expanding our dataset by including reviews for better predictions).

For movie recommendation, we experimented by combining different parameters and methods and playing with the weights assigned to them in order to get the best fit. On top of normal content-based filtering with the metadata parameters, we added plot overview in order to better our results. We also attempted to combine a general recommender and personal recommender system in order to get better results which captured both the demographic and personal understanding of user in order to give better recommendations.

References

[1] Kleeman, Alex, Nick Hendersen, and Sylvie Denuit. "Matrix factorization for collaborative prediction." ICME, 2005.

[2] Raghavendra, C. K., and K. C. Srikantaiah. "Similarity Based Collaborative Filtering Model for Movie Recommendation Systems." *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)* . IEEE, 2021.

[3] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[4] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).