EDI Project Report

Members: 1. Sachin Iyer (GR. No. 1710575)
2. Himanshu Kale (GR. No. 1710073)
3. Shreyash Palresha (GR. No. 1710067)
4. Syed Nuaym Asfan (GR. No. 1710715)

Guided by: Prof. A. A. Bhosale
Topic: Smart Movie Recommendation System

Problem Statement: To predict user-specific movies according to past experience and movie detail.

Abstract: Movie recommendation is critically important for modern technology users. It carries out comprehensive aggregation of user's preferences, reviews, and emotions to help them find suitable movies conveniently. However, it requires both accuracy and timeliness. In this report, a movie recommendation framework based on a hybrid recommendation model on Spyder platform is proposed to improve the accuracy and timeliness of mobile movie recommender system. In the proposed approach, we first use a hybrid recommendation method to generate a preliminary recommendation list. Then sentiment analysis is employed to optimize the list.

Finally, the hybrid recommender system method makes it convenient and fast for users to obtain useful movie suggestions.

Motivation:

- Movie prediction will help in increasing user time spent on websites.

- It will attract more users to the website.
- Movies are a very vast field to describe having several details in several dimensions.

Method Used:

We have broken the problem statement in 3 parts.

- **Content based Recommendation**
- **Collaborative filtering based Recommendation**
- **Hybrid Recommendation System**

1. Content based Recommendation : A Content-based recommendation system tries to recommend items to users based on their profile.
   In the first part, we will build Content Based Recommender System. System will be able to recommend films that are similar to a chosen one, based on two contents:

- **Description and taglines of the movies**
- **Cast, director, keywords and genres of the movies**

File **metadata.csv** contains all information about movies in the dataset.

```
metadata = pd.read_csv("data/metadata.csv")
metadata = metadata.drop(['Unnamed: 0'], axis=1)
```

adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	inal_langu	original_title	overview	popul
False	{'id': 10194, 'n…	30000000	['Animation', 'Comedy', 'Family']	http://to…	862	tt0114709	en	Toy Story	Led by Woody…	21.94
False	nan	65000000	['Adventure', 'Fantasy', 'Family']	nan	8844	tt0113497	en	Jumanji	When sibling…	17.01
False	{'id': 119050, '…	0	['Romance', 'Comedy']	nan	15602	tt0113228	en	Grumpier Old Men	A family wed…	11.71
False	nan	16000000	['Comedy', 'Drama', 'Romance']	nan	31357	tt0114885	en	Waiting to Exhale	Cheated on, …	3.859
False	{'id': 96871, 'n…	0	['Comedy']	nan	11862	tt0113041	en	Father of the Bride Part II	Just when Ge…	8.387
False	nan	60000000	['Action', 'Crime', 'Drama', 'Thriller']	nan	949	tt0113277	en	Heat	Obsessive ma…	17.92
False	nan	58000000	['Comedy', 'Romance']	nan	11860	tt0114319	en	Sabrina	An ugly duck…	6.677
False	nan	0	['Action', 'Adventur…	nan	45325	tt0112302	en	Tom and Huck	A mischievou…	2.561
False	nan	35000000	['Action', 'Adventur…	nan	9091	tt0114576	en	Sudden Death	Internationa…	5.231
False	{'id': 645, 'nam…	58000000	['Adventure', 'Action', 'Thriller']	http://ww…	710	tt0113189	en	GoldenEye	James Bond m…	14.68
False	nan	62000000	['Comedy', 'Drama', 'Romance']	nan	9087	tt0112346	en	The American President	Widowed U.S.…	6.318

File **links_small.csv** contains related movies ids from other files. But only *tmdbId* is needed. As a result, we get 9099 films, that are in metadata dataframe. Let **avail** dafaframe be metadata films, that are available to process in the next step.

```
links_small = pd.read_csv('data/links_small.csv')
links_small = links_small[links_small.tmdbId.notnull()].tmdbId
avail = metadata[metadata.id.isin(links_small)]
```

adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	inal_langu	original_title	overview	popularity	
False	{'id': 10194, 'n…	30000000	['Animation', …	http://t…	862	tt0114709	en	Toy Story	Led by Woo…	21.9	/rh
False	nan	65000000	['Adventure', …	nan	8844	tt0113497	en	Jumanji	When sibli…	17	/vz
False	{'id': 119050, '…	0	['Romance', 'Comedy']	nan	15602	tt0113228	en	Grumpier Old Men	A family w…	11.7	/6k
False	nan	16000000	['Comedy', 'Dr…	nan	31357	tt0114885	en	Waiting to Exhale	Cheated on…	3.86	/16
False	{'id': 96871, 'n…	0	['Comedy']	nan	11862	tt0113041	en	Father of the Bride Part II	Just when …	8.39	/e6
False	nan	60000000	['Action', 'Cr…	nan	949	tt0113277	en	Heat	Obsessive …	17.9	/zM
False	nan	58000000	['Comedy', 'Romance']	nan	11860	tt0114319	en	Sabrina	An ugly du…	6.68	/jQ
False	nan	0	['Action', 'Ad…	nan	45325	tt0112302	en	Tom and Huck	A mischiev…	2.56	/sG
False	nan	35000000	['Action', 'Ad…	nan	9091	tt0114576	en	Sudden Death	Internatio…	5.23	/eo

A. Recommendations based on taglines and film describing

We will be using the Cosine Similarity to calculate a numeric quantity that denotes the similarity between two movies. For that We will be using Tf-Idf Vectorizer matrix. The dot product of these matricies will give us cosine similarity matrix.

Each *i-th* row of cosine_sim corresponds to similarity of *i-th* movie with each movie.

```
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(avail.description)
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0.0166…	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0.0166…	1	0.0432…	0	0.006…	0.0411…	0	0	0.08786…	0	0.01137…	0	0	0.01183…	0.0117…
2	0	0.0432…	1	0	0.023…	0	0	0.0062…	0	0	0	0	0	0	0
3	0	0	0	1	0	0.0059…	0	0.0169…	0	0	0.00609…	0	0	0.00634…	0
4	0	0.0064…	0.0231…	0	1	0	0.0238…	0	0.03132…	0	0.01443…	0	0.00809…	0.01501…	0
5	0	0.0411…	0	0.00595…	0	1	0	0	0.04098…	0	0.00582…	0	0	0.01988…	0.0271…
6	0	0	0	0	0.023…	0	1	0	0	0	0	0	0	0	0
7	0	0	0.0062…	0.01699…	0	0	0	1	0	0	0	0	0.02163…	0	0
8	0	0.0878…	0	0	0.031…	0.0409…	0	0	1	0	0	0.08001…	0	0.02591…	0.0274…
9	0	0	0	0	0	0	0	0	0	1	0	0	0.04349…	0	0.0299…
10	0	0.0113…	0	0.00609…	0.014…	0.0058…	0	0	0	0	1	0	0	0.05394…	0
11	0	0	0	0	0	0	0	0	0.08001…	0	0	1	0	0	0
12	0	0	0	0	0.008…	0	0	0.0216…	0	0.0434…	0	0	1	0	0

Now, let's get 10 recommendations for the film **The Terminator**

```
def get_recommendations(title, number):
    try:
        idx = indices[title]
    except:
        print("Film (%s) does not exist in the dataset" % title)
        return

    if type(idx) != np.dtype('int64') and len(idx) > 1:
        print("There are several films called (%s)" % title)
        print("Their indices are: ", avail[avail.title == title].index)
        idx = sorted(idx, key=lambda x: avail.iloc[x].popularity, reverse=True)
        idx = idx[0]
        print("For recommendation, I will take the most popular one with id ", avail.iloc[idx].id)

    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:number+1]
    movie_indices = [i[0] for i in sim_scores]
    return titles.iloc[movie_indices]


get_recommendations('The Terminator', 10)
```

```
Out[12]:
582               Terminator 2: Judgment Day
13693                    Terminator Salvation
14917                          Teenage Caveman
14631                          The Book of Eli
6388      Terminator 3: Rise of the Machines
25864                      Terminator Genisys
5868                              Just Married
19669                              Cloud Atlas
10228                          Must Love Dogs
3428                                The Hunger
Name: title, dtype: object
```

B. Recommendations based on cast, crew, keywords and genres

At first, merge credits and keywords dataframes with our metadata. Then, reassign the value of **avail**, available films. Now, it's **9663** movies.

```
credits = pd.read_csv('data/credits.csv')
keywords = pd.read_csv('data/keywords.csv')

metadata = metadata.merge(credits, on='id').merge(keywords, on='id')
metadata = metadata.drop(['Unnamed: 0.1'], axis=1)
avail = metadata[metadata.id.isin(links_small)]
```

Index	title	video	vote_average	vote_count	nnamed:	cast	crew	keywords
0	Toy Story	False	7.7	5.42e+03	0	['Tom Hanks', 'T…	[{'name': 'John Lass…	[{'id': 931, 'name'…
1	Jumanji	False	6.9	2.41e+03	1	['Robin Williams…	[{'name': 'Larry J. …	[{'id': 10090, 'nam…
2	Grumpier Old Men	False	6.5	92	2	['Walter Matthau…	[{'name': 'Howard De…	[{'id': 1495, 'name…
3	Waiting to Exhale	False	6.1	34	3	['Whitney Housto…	[{'name': 'Forest Wh…	[{'id': 818, 'name'…
4	Father of the Bride Part II	False	5.7	173	4	['Steve Martin',…	[{'name': 'Alan Silv…	[{'id': 1009, 'name…
5	Heat	False	7.7	1.89e+03	5	['Al Pacino', 'R…	[{'name': 'Michael M…	[{'id': 642, 'name'…
6	Sabrina	False	6.2	141	6	['Harrison Ford'…	[{'name': 'Sydney Po…	[{'id': 90, 'name':…
7	Tom and Huck	False	5.4	45	7	['Jonathan Taylo…	[{'name': 'David Lou…	[]
8	Sudden Death	False	5.5	174	8	['Jean-Claude Va…	[{'name': 'Peter Hya…	[{'id': 949, 'name'…

We will do preprocessing of crew, cast and keywords Series. From the crew, We will only pick a director of the movie as a feature since others don't contribute that much to the feel of the movie. As a result, for a director to have more influence then for a regular keyword, for example, repeat a director 3 times. Keywords will be converted to the lists of stemmed words. We will take only keywords that have occurred more then 3 times in the dataset. From the cast, We will take only three main actors.

Director & Cast Preprocessing

```
def get_director(crew):
    for member in crew:
        if member['job'] == 'Director':
            return member['name']
    return np.nan

avail['director'] = avail['crew'].apply(get_director)
avail.director = avail.director.astype('str').apply(lambda x: x.replace(" ", "").lower())
avail.director = avail.director.apply(lambda x: [x, x, x])


avail['cast'] = avail['cast'].apply(lambda x: x[:3] if len(x) > 3 else x)
avail.cast = avail.cast.apply(lambda x: [w.replace(" ", "").lower() for w in x])
```

Keywords Preprocessing

```
stemmer = SnowballStemmer('english')
stemmer.stem('films')

def filter_keywords(keywords):
    words = []
    for i in keywords:
        if i in key:
            words.append(i)
    return words

avail.keywords = avail.keywords.apply(filter_keywords)
avail.keywords = avail.keywords.apply(lambda x: [stemmer.stem(i) for i in x])
avail.keywords = avail.keywords.apply(lambda x: [str.lower(i.replace(" ", "")) for i in x])
```

We now have our preprocessed cast, director, keywords and genres, now we will just stack them into **stack** columns in **avail**.

```
avail['stack'] = avail.keywords + avail.cast + avail.genres + avail.director
avail['stack'] = avail['stack'].apply(lambda x: ' '.join(x))
```

	crew	keywords	cast_size	crew_size	director	stack
e…	[{'name': 'John …	['jealousi', 't…	13	106	['johnlasseter', '…	jealousi toy boy friendship friend riva…
o…	[{'name': 'Larry…	['disappear', "…	26	16	['joejohnston', 'j…	disappear basedonchildren'sbook newhom …
a…	[{'name': 'Howar…	['fish', 'bestf…	7	4	['howarddeutch', '…	fish bestfriend duringcreditssting walt…
a…	[{'name': 'Fores…	['basedonnovel'…	10	10	['forestwhitaker',…	basedonnovel interracialrelationship si…
n…	[{'name': 'Alan …	['babi', 'midli…	12	7	['charlesshyer', '…	babi midlifecrisi confid age daughter m…
d…	[{'name': 'Micha…	['robberi', 'de…	65	71	['michaelmann', 'm…	robberi detect bank obsess chase shoot …
l…	[{'name': 'Sydne…	['pari', 'broth…	57	53	['sydneypollack', …	pari brotherbrotherrelationship chauffe…
a…	[{'name': 'David…	[]	7	4	['peterhewitt', 'p…	jonathantaylorthomas bradrenfro rachael…
e…	[{'name': 'Peter…	['terrorist', '…	6	9	['peterhyams', 'pe…	terrorist hostag explos jean-claudevand…

We will use the same Cosine Similarity, but now with CountVecrorizer matrix. Then let's get recommendations for **The Terminator** film. As you can see, we have **Avatar** and **Titanic** films as recommendations. That's because thay all have a same director: James Kameron.

```
count = CountVectorizer(analyzer='word',ngram_range=(1, 3),min_df=0)
count_matrix = count.fit_transform(avail['stack'])

cosine_sim = linear_kernel(count_matrix, count_matrix)

titles = avail.title
indices = pd.Series([i for i in range(len(avail))], index=avail.title)


print(get_recommendations('The Terminator', 10))
```

```
In [21]: print(get_recommendations('The Terminator', 10))
582                Terminator 2: Judgment Day
1185                                The Abyss
1251                                   Aliens
15479                                  Avatar
375                                 True Lies
6697       Terminator 3: Rise of the Machines
1731                                  Titanic
5888          Piranha Part Two: The Spawning
7092                  The Matrix Revolutions
6530                     The Matrix Reloaded
Name: title, dtype: object
```

As we see, **Recommendations based on cast, crew, keywords and genres** works better than **Recommendations based on taglines and film describing**. So we will use these cosine_sim values in the next Hybrid system

2. **Collaborative Filtering based Recommendation** :

In the 2nd part, We will use a technique called **Collaborative Filtering** to make recommendations. Collaborative Filtering is based on the idea that users similar to me can be used to predict how much I will like a particular product or service those users have experienced but I have not.

For that We will use the **Surprise** library that used extremely powerful algorithms like **Singular Value Decomposition (SVD)** to minimise RMSE (Root Mean Square Error) and give great recommendations. **Surprise** is a Python scikit building and analysing recommender systems.

```
ratings = pd.read_csv("data/ratings_small.csv")
reader = Reader()
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
```

Index	userId	movieId	rating
0	1	31	2.5
1	1	1029	3
2	1	1061	3
3	1	1129	2
4	1	1172	4
5	1	1263	2
6	1	1287	2
7	1	1293	2
8	1	1339	3.5
9	1	1343	2

```
svd = SVD(n_factors=100, n_epochs=40, lr_all=0.005, reg_all=0.2, verbose=False)
cross_validate(svd, data, measures=['RMSE', 'MAE'])

trainset = data.build_full_trainset()
svd.fit(trainset)
```

```
Out[24]:
{'test_rmse': array([0.8962562 , 0.88827792, 0.89087942, 0.89174665,
0.8880256 ]),
 'test_mae': array([0.69108176, 0.68488087, 0.68689359, 0.68977496,
0.68720805]),
 'fit_time': (29.113847732543945,
  28.843252182006836,
  29.297473669052124,
  29.874940633773804,
  35.94582390785217),
 'test_time': (0.5463240146636963,
  0.7033896446228027,
  0.48439502716064453,
  0.5303101539611816,
  0.8437826633453369)}
```

After training, we get RMSE = **0.8912** and MAE = **0.688** , which is good for our model.

3. **Hybrid Recommendation System** :

In the last part, we will build a hybrid system. How the model works: get 50 top scoring films from the cosine_sim matrix; for a particular user, sort them by predicted rating for user.

```python
def hybrid(userId, title, number=10):
    try:
        idx = indices[title]
    except:
        print("Film (%s) does not exist in the dataset" % title)
        return

    if type(idx) != np.dtype('int64') and len(idx) > 1:
        print("There are several films called (%s)" % title)
        print("Their indices are: ", avail[avail.title == title].index)
        idx = sorted(idx, key=lambda x: avail.iloc[x].popularity, reverse=True)
        idx = idx[0]
        print("For recommendation, I will take the most popular one with id ", avail.iloc[idx].id)

    tmdbId = id_map.loc[title]['id']
    movie_id = id_map.loc[title]['movieId']
    sim_scores = list(enumerate(cosine_sim[int(idx)]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:50]

    movie_indices = [i[0] for i in sim_scores]
    movies = avail.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'year', 'id']]
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId'])
    movies = movies.sort_values('est', ascending=False)
    return movies.head(number)
```

Now , Let's get Recommendation For **Inception** for 2 different Users.

```
hybrid(34, 'Inception', 10)
```

```
Out[29]:
                          title  vote_count  ...      id       est
4222                    Memento      4168.0  ...      77  8.496985
12973           The Dark Knight     12269.0  ...     155  8.358105
11847              The Prestige      4510.0  ...    1124  8.355965
2213                       Cube      1101.0  ...     431  8.277865
23952              Interstellar     11187.0  ...  157336  8.209403
7800                     Cypher       196.0  ...   10133  8.107287
1288             The Terminator      4208.0  ...     218  7.992540
1264                      Alien      4564.0  ...     348  7.990208
582   Terminator 2: Judgment Day      4274.0  ...     280  7.958035
1251                     Aliens      3282.0  ...     679  7.914204

[10 rows x 6 columns]
```

```
hybrid(10, 'Inception', 10)
```

```
Out[30]:
                          title  vote_count  ...      id       est
4222                    Memento      4168.0  ...      77  8.453886
11847              The Prestige      4510.0  ...    1124  8.316322
12973           The Dark Knight     12269.0  ...     155  8.305580
2213                       Cube      1101.0  ...     431  8.223071
23952              Interstellar     11187.0  ...  157336  8.184828
7800                     Cypher       196.0  ...   10133  8.160575
1288             The Terminator      4208.0  ...     218  7.952493
1264                      Alien      4564.0  ...     348  7.941127
582   Terminator 2: Judgment Day      4274.0  ...     280  7.913418
1251                     Aliens      3282.0  ...     679  7.872159

[10 rows x 6 columns]
```

We see that for our hybrid recommender, we get different recommendations for different users although the movie is the same.

Conclusion:

Under the guidance of Resp. Prof. A. A. Bhosale Sir, we were able to build our system under 3 layers. The system works best on hybrid model which use both collaborative and content based filtering.