Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

[MOVIE RECOMMENDER]

NLP Project

Under Supervision of:

Dr. Sally Saad

TA. Eman Mahmoud

[Team ID: 2]

Team Members:

1st Team Member Name:

Kareem Sherif Fathy

1st Team Member ID:

2018170283

2nd Team Member Name:

Kareem Saeed Ragab

2nd Team Member ID:

2018170282

3rd Team Member Name:

Abanoub Asaad Azab

3rd Team Member ID:

2018170001

4th Team Member Name:

Nada El Sayed Anies

4th Team Member ID:

2018170430

5th Team Member Name:

Nada Mohamed Abdelhamed

5th Team Member ID:

2018170434

Movies AI-Recommender System

Introduction:

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 systems** as they improve the quality of search results and provide items that are more relevant to the search item or are 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. Almost every major tech company has applied them in some form or the other: **Amazon** uses it to suggest products to customers, **YouTube** uses it to decide which video to play next on autoplay, and **Facebook** uses it to recommend pages to like and people to follow. Moreover, companies like **Netflix** and **Spotify** depend highly on the effectiveness of their recommendation engines for their business and success.

In order to apply the learned **Natural Language Processing (NLP)** techniques and methodologies, and also keep knocking in the AI field, we are building a baseline Movie Recommendation System using a well-known dataset. For a start, this project will pretty much serve as a foundation for recommendation systems.

Methodology:

There are basically three types of recommender systems [1]:

- **Demographic Filtering**: Offers generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different, this approach is considered to be too simple. 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.
- **Collaborative Filtering**: The method matches persons with similar interests and provides recommendations based on this matching. Collaborative filters do not require item metadata like their content-based counterparts.
- **Content-Based Filtering**: Suggests similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person liked a particular item, he or she will also like an item that is similar to it.

In our case, we are using the **Content-Based Filtering,** in this recommender system the content of the movie (overview, cast, crew, keyword, etc.) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

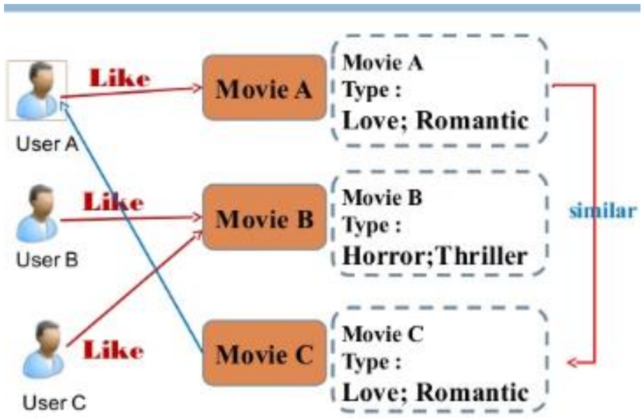


Figure 1: Content-based ontology

We are computing pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score but, there were some critical steps before this step:

1. Data Analysis

1.1. Features Statistical Relation [2]:

The aimed dataset was containing a set of numeric features, we used the statistical formulas and graphs to study the relationship between every two features, and using (Pie Chart, Bar Chart, and Box Plot) also WordCloud was used.

1.2. Features Correlation:

The matrix correlation was implemented to zoom out the relation between all the existing features e.g. (budget, popularity, revenue, runtime, release_year).

2. Data Preprocessing (Data Cleaning):

2.1. Features Selection:

After the data analysis, we gathered valuable information about every feature in the dataset and here we select the features that are going to help us in our modelling neglecting the other features as it seems useless in our case. The Features we considered in our implementation process were: (**movie_id**, **title**, **overview**, **genres**, **keywords**, **cast**, and **crew**)

2.2. Solving Confusing Values:

Some of the selected features were having missing values, for example, the overview column was having 3 missing values, and as we are having up to 5,000 samples these 3 samples won't be big harm, so we removed these samples, and the duplication of data was also checked, there were not any duplicated data exist.

2.3. **Keywords Extraction:**

Some of the dataset columns were in the shape of a complex object that hard to be read or to be used as a direct feature, here is a simple example of the **genres** column:

```
# Displaying genres sample
movies_data.iloc[0].genres
```

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

We are interested in the name tag value, we don't need it's ID in any further work, so in this step we are extracting these keywords and collect them into simple list object to be easily readable and easy-data fetch.

3. NLP Techniques:

3.1. **Stemming:**

On displaying the words in the collected text query in a way of showing the words summary, we noticed that there are many words of the same base or root as shown

```
In [55]: vectorizer.get_feature_names()
 'admit',
 'adolesc',
 'adopt',
 'ador',
 'adrienbrodi',
 'adult',
 'adultanim',
 'adulteri',
 'adulthood',
 'advanc',
 'adventur',
 'adventure',
 'adventures',
 'advertis',
```

The existence of such an issue can weaken the training process leading to un-accurate prediction by the model, so we fixed it using the **PorterStemmer** [3] which gives us a pleasant good results.

3.2. **TF-IDF [4][5]:**

After establishing the text queries we needed to convert the word vector of each movie. We computed The **Term Frequency-Inverse Document Frequency (TF-IDF)** vectors. This gave us a matrix where each column represents a word in the overview vocabulary (all the words that appear in at least one document) and each row represents a movie, as before. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

3.3. **Cosine Similarity [5][8]:**

We saw that over **25,000** different words were used to describe the **5,000** movies in our dataset.

With this matrix in hand, we could then compute a similarity score. There were several candidates for this; such as the **Euclidean**, the **Pearson** and the **Cosine Similarity** Scores. We used the Cosine Similarity to calculate a numeric quantity that denotes the similarity between two movies. We used the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate. Mathematically, it is defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Data Set Summary:

The system was built using the **TMDB 5000** [6] Movie Dataset, it was generated from The Movie Database [7] API. Their API also provides access to data on many additional movies, actors and actresses, crew members, and TV shows. It attracted more than **203,500** data scientists to use The Dataset as the entry trusty dataset in the recommendation systems. It Consists of **23** Columns and **4809** Entries:

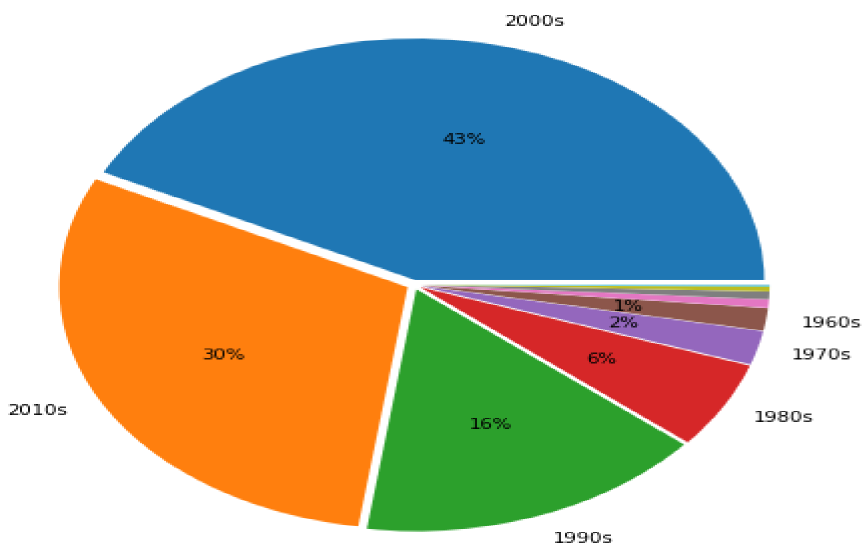
```
In [16]: movies_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4809 entries, 0 to 4808
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4809 non-null   int64
 1   genres                4809 non-null   object
 2   homepage              1713 non-null   object
 3   id                    4809 non-null   int64
 4   keywords              4809 non-null   object
 5   original_language     4809 non-null   object
 6   original_title        4809 non-null   object
 7   overview              4806 non-null   object
 8   popularity            4809 non-null   float64
 9   production_companies  4809 non-null   object
 10  production_countries  4809 non-null   object
 11  release_date          4808 non-null   object
 12  revenue               4809 non-null   int64
 13  runtime               4807 non-null   float64
 14  spoken_languages      4809 non-null   object
 15  status                4809 non-null   object
 16  tagline               3965 non-null   object
 17  title                 4809 non-null   object
 18  vote_average          4809 non-null   float64
 19  vote_count            4809 non-null   int64
 20  movie_id              4809 non-null   int64
 21  cast                  4809 non-null   object
 22  crew                  4809 non-null   object
dtypes: float64(3), int64(5), object(15)
memory usage: 901.7+ KB
```

The movies collected in the dataset are age-variant, it holds movies from **1960s** till **2019.** Here are some columns' summaries that we were interested in:

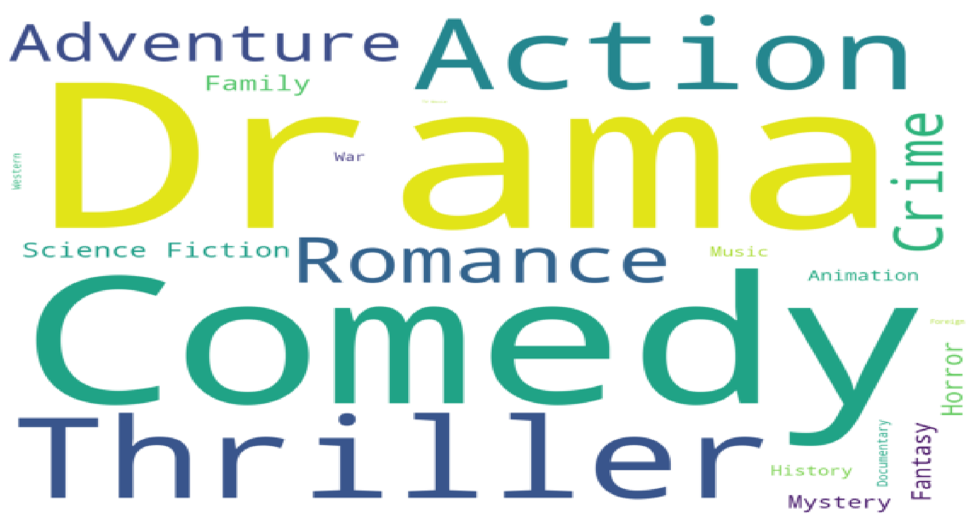
- **'year_of_production':**

```
In [7]: count_decade_pie(movies_data, filename="pie_decade.png")
```



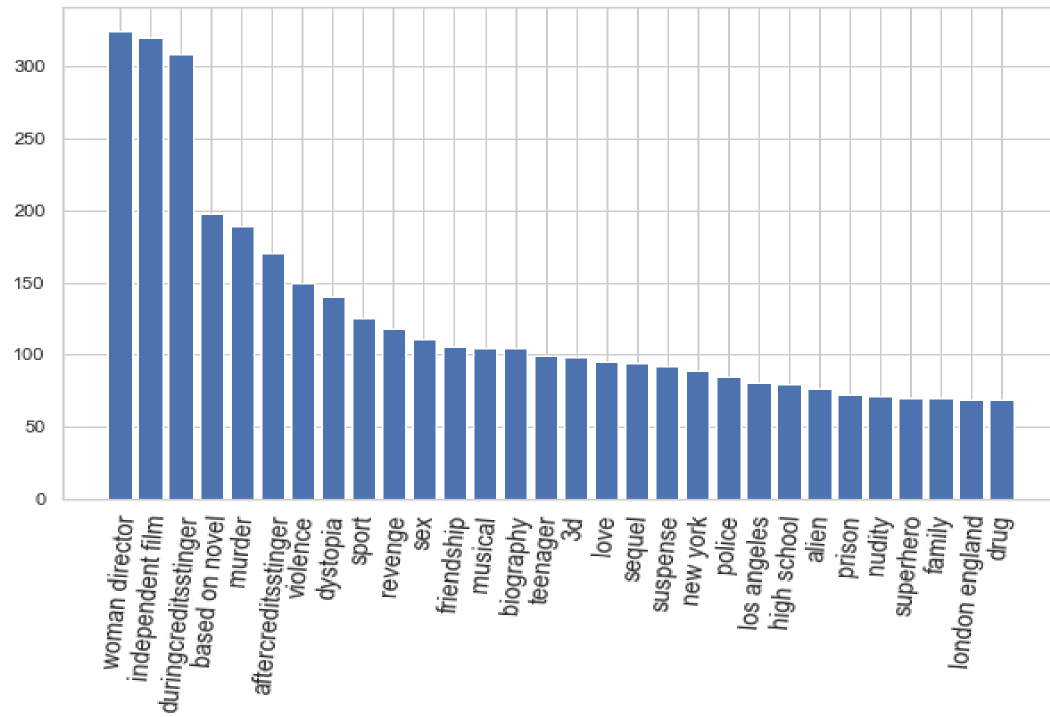
- **'genres':**

```
In [8]: multi_wordcloud(movies_data.genres,filename="wordcloud_genres.png")
multi_bar(movies_data.genres,filename="bar_genres.png")
```



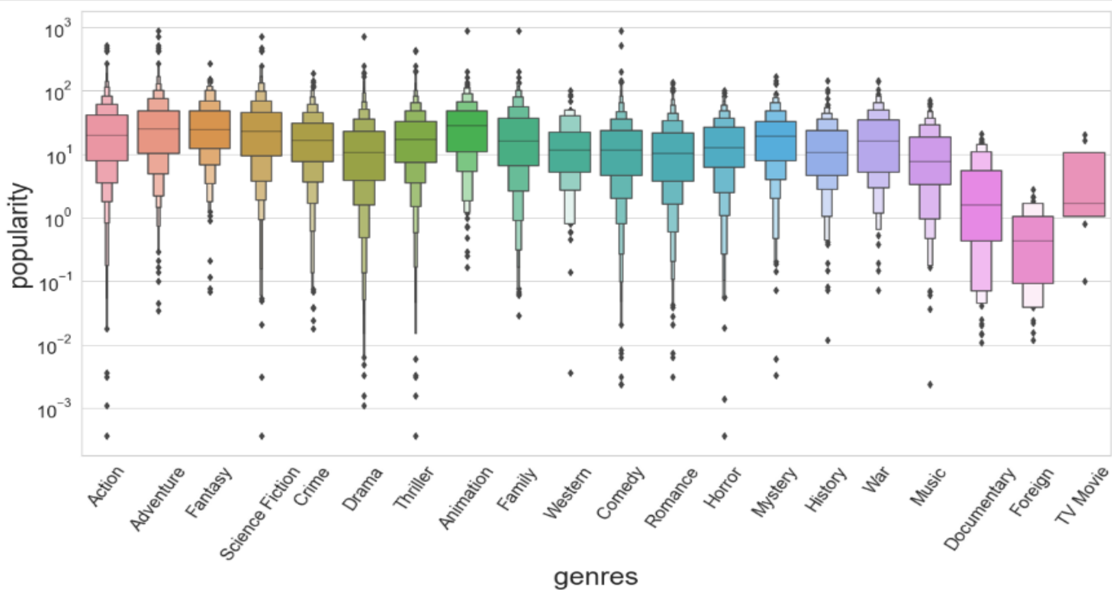
- **‘keywords’:**

```
In [9]: multi_bar(movies_data.keywords,filename="bar_keywords.png")
```



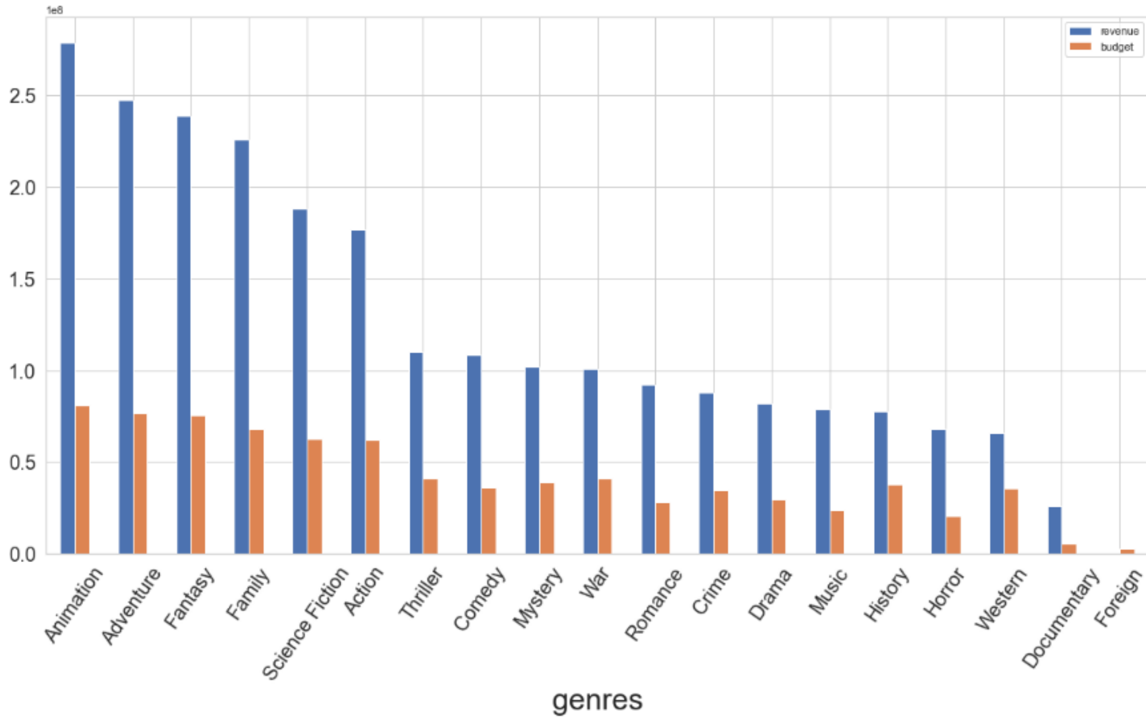
- **‘popularity’ vs ‘genres’:**

```
In [10]: plotby_box(movies_data, "genres", "popularity", yscale="log", filename="genres_popularity.png")
```



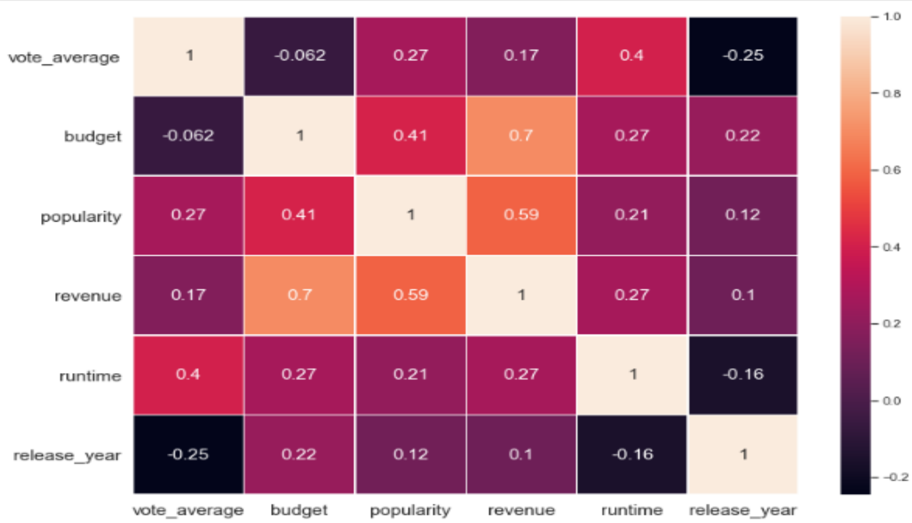
- **'budget' vs 'revenue':**

In [12]:
```
plotby_2bar(movies_data,"genres", ["revenue","budget"], filename="genres_budget_revenue.png")
```



- **Numeric features Correlation:**

In [13]:
```
plot_corr(data[["vote_average", "budget", "popularity", "revenue", "runtime", "release_year"]], filename="corr.png")
```



Results:

Evaluation of the model performance in the AI field is crucial, we evaluated our model performance on the whole train data, and also on unseen data during the training (test data) so we can totally judge fairly on its performance.

1. **Performance Evaluation on the Train Data**:

The first evaluation approach was evaluating our recommendation system exactly the same way the **Leave-One-Out Cross-Validation (LOOCV)** works. For each movie in the dataset, we calculated the number of movies which has a similarity score value greater than a specific threshold value **(Strong Similarity)**, the number of which is lower than the threshold **(Weak Similarity)** and the number of which has no similarity at all **(No Similarity)** then, calculating the **Precision** and **Recall** values for that movie given the 3 pre-calculated values.

Metric	Formula
Precision	$P=\frac{\vert\{relevant\_docs\}\cap\{retrieved\_docs\}\vert}{\vert\{retrieved\_docs\}\vert}$
Recall	$R=\frac{\vert\{relevant\_docs\}\cap\{retrieved\_docs\}\vert}{\vert\{true\_relevant\_docs\}\vert}$
F-measure	$F=2\cdot\frac{precision\cdot recall}{precision+recall}$

By looking at the **Precision** and **Recall** formulas, we could then say that the **Strong Similarity** value is considered the number of the relevant retrieved movies, and the **true_relevant_docs** or **all the relevant movies in the dataset** is considered the sum value of the Strong and Weak Similarities.

The approach proceeds by calculating the average value of precision and recall for all movies and then calculating the **F1-Score** to be our **Evaluation Metric.** Below are the model evaluation results specifying a **threshold=0.20** and **N=10** (the maximum number of retrieved movies): **[F1-SCORE = 0.84]**

```
In [212]: print("Average Percision", avg_percision, "Average Recall", avg_recall)
          f1_score = 2 * (avg_percision * avg_recall) / (avg_percision + avg_recall)
          print("F1-Score = ", f1_score)

          Average Percision 0.8448886112846128 Average Recall 0.8448886112846128
          F1-Score =  0.8448886112846128
```

On specifying a **threshold=0.15** and **N=10** The F1-Score value **increases** about **10%** on **decreasing** the threshold value by **0.05: [F1-SCORE = 0.99]**

```
In [62]: avg_percision, avg_recall = evaluate_performance(sim_thresh=0.15, n_retrieval=10)
         f1_score = 2 * (avg_percision * avg_recall) / (avg_percision + avg_recall)
         tbl = PrettyTable(['Average Percision', 'Average Recall', 'F1-Score'])
         tbl.add_row([avg_percision, avg_recall, f1_score])
         print(tbl)

         +--------------------+--------------------+--------------------+
         | Average Percision  |   Average Recall   |      F1-Score      |
         +--------------------+--------------------+--------------------+
         | 0.9920674578388506 | 0.9920674578388506 | 0.9920674578388506 |
         +--------------------+--------------------+--------------------+
```

2. **Performance Evaluation on the Test Data**:

The second approach considers evaluating the model performance by introducing new data that were not seen during the training process, unfortunately for the lack of such data online we had to build the test data by ourselves.

Performance Evaluation on the Test Data

In [63]:
```
# Reading TEST Data...
test_data = pd.read_csv('tmdb-movies-5k/cleaned_merged_test.csv')
test_data
```

Out[63]:

	movie_id	title	tags
0	354912	Coco	despite his family's baffling generations-old ...
1	696806	The Adam Project	after accidentally crash-landing in 2022, time...
2	568124	Encanto	the tale of an extraordinary family, the madri...
3	646380	Don't Look Up	two low-level astronomers must go on a giant m...
4	508943	Luca	luca and his best friend alberto experience an...
5	639933	The Northman	prince amleth is on the verge of becoming a ma...

Following the same evaluation way like in (1) below is the model evaluation results on specifying a **threshold=0.20** and **N=10** (the maximum number of retrieved movies): **[F1-SCORE = 0.91]**

In [111]:
```
avg_percision, avg_recall = evaluate_performance(sim_thresh=0.20, n_retrieval=10, data=test_data)
f1_score = 2 * (avg_percision * avg_recall) / (avg_percision + avg_recall)
tbl = PrettyTable(['Average Percision', 'Average Recall', 'F1-Score'])
tbl.add_row([avg_percision, avg_recall, f1_score])
print(tbl)
```

```
+--------------------+--------------------+--------------------+
| Average Percision  |   Average Recall   |      F1-Score      |
+--------------------+--------------------+--------------------+
| 0.9166666666666666 | 0.9166666666666666 | 0.9166666666666666 |
+--------------------+--------------------+--------------------+
```

On specifying a **threshold=0.20** and **N=5000: [F1-SCORE = 0.007]**

In [110]:
```
avg_percision, avg_recall = evaluate_performance(sim_thresh=0.20, n_retrieval=5000, data=test_data)
f1_score = 2 * (avg_percision * avg_recall) / (avg_percision + avg_recall)
tbl = PrettyTable(['Average Percision', 'Average Recall', 'F1-Score'])
tbl.add_row([avg_percision, avg_recall, f1_score])
print(tbl)
```

```
+----------------------+----------------------+----------------------+
|  Average Percision   |    Average Recall    |       F1-Score       |
+----------------------+----------------------+----------------------+
| 0.006000000000000001 | 0.008276583390592297 | 0.006956776559898418 |
+----------------------+----------------------+----------------------+
```

References:

1. https://www.appier.com/blog/what-is-a-recommendation-engine-and-how-does-it-work
(Last accessed May 2022)
2. https://www.matematica.pt/en/cheatsheet/statistical-graph-type.php
(Last accessed May 2022)
3. https://www.nltk.org/_modules/nltk/stem/porter.html
(Last accessed May 2022)
4. https://en.wikipedia.org/wiki/Tf%E2%80%93idf
(Last accessed May 2022)
5. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
(Last accessed May 2022)
6. https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_movies.csv
(Last accessed May 2022)
7. https://www.themoviedb.org/
(Last accessed May 2022)
8. https://www.machinelearningplus.com/nlp/cosine-similarity/
(Last accessed May 2022)