TUTORIAL

(Recommender System using multiple ML techniques)

Introduction

The present industry makes extensive use of recommender systems in a variety of industries, including books, movies, and products. This has led to an increase in the R&D of methods to help and extract better outcomes in recent years. In the modern world, a lot of businesses use recommendation engines to suggest films, TV shows, books, documents, websites, conferences, tourist destinations, and educational resources. These businesses include those in the online business sectors. The lack of high-quality Big Data is a significant obstacle for artificial intelligence (AI), notwithstanding its tremendous advancements too far. Data in many real-world applications take the shape of isolated islands. Serious concerns about user security and privacy are contributing to the growing difficulty of data integration efforts.

A recommendation system generates suggestions based on a user's previous purchases. Any firm can benefit from our suggestions. Recommendation engines may improve product interaction and cross-selling potential in addition to fostering customer relationships, which would boost business returns. Recommender systems are used by Amazon, Tinder, Netflix, and YouTube. Depending on the user's preferences, the applications provide relevant recommendations. Recommender systems are useful for news websites. AI is being used by many businesses to reduce costs, increase productivity, increase income, and enhance customer service. Organisations should integrate all smart technologies, such as natural language processing, machine learning, and others, into their processes and products for the greatest possible benefits.

To create a recommendation system for consumers, we mined the TMDB 5000 Movie dataset (Kaggle, 2017) for this case study. The two filtering methods that we used to create the recommendation system are content-based filtering and collaborative-based filtering, which are covered in more depth in the following sections.

Types of Recommender Systems:

1. Content-based filtering suggests products based on characteristics such as keywords, categories, or summaries of previously engaged goods.

2. Collaborative filtering makes item recommendations without depending on extensive item information.

By examining preparing data, model implementation, along with evaluation strategies, this tutorial offers both theoretical understanding and useful skills for developing and improving recommender systems.

Important Libraries

Python modules such as matplotlib and seaborn are necessary for visualising data insights, pandas for data processing, and numpy for numerical calculations while building a recommender system. While sklearn and scipy offer strong capabilities for learning algorithms, feature extraction, similarity computing, and managing sparse matrices, libraries like ast assist in converting texts to Python objects.

Important utilities include NearestNeighbors for collaborative filtering, cosine_similarity and linear_kernel for similarity figures, and TfidfVectorizer for extracting pertinent phrases. A well-structured workflow that is prepared for data analysis and preparation is made possible by implementing these libraries.

Data Cleaning and Preparation

The dataset used has been taken from IMDB 5000 Movie Dataset (Kaggle, 2017). We have two datasets one is about movies rating and the other one is about movie credits.

The movie rating dataset includes features like User ID, Movie ID, budget, original language, title, genres, popularity, overview, vote average, vote counts, and additional details like production companies, countries, release date, runtime, and tagline. The movie credits dataset provides information on Movie ID, title, cast (actors), and crew (including directors and producers). Together, these datasets offer comprehensive metadata for building a recommendation system.

We need to preprocess, clean, and combine these data to use it efficiently. The title column is used as a key for connecting the two databases. Missing data are filled in or removed to ensure error-free analysis, and irrelevant columns like budget and language are eliminated to simplify the study.

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	[{"id": 1463, "name": "culture clash"}, {"id":...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...

Content-Based Recommendation System

A recommendation system based on content makes suggestions for products that are like what a user has previously expressed interest in. The algorithm will suggest additional films with comparable genres, casts, or details, for example, if a user enjoyed a certain film. This method depends less on user interactions and more on the objects' information.

Data Preparation

Key elements including cast, crew, genres, and keywords are retrieved in order to provide information needed for a recommender program. JSON data that has been stringified is transformed into useful lists by filtering pertinent columns. The metadata is then searched for actors, directors, genre names, and keywords. Lastly, to develop useful features for content-based filtering, all textual data is merged into a single "tags" column.

Overview-Based Recommender

TF-IDF (Term Frequency-Inverse Document Frequency)

Term Frequency (TF), which measures a word's relative frequency, with Inverse text Frequency (IDF), which reduces the weight of frequently used terms, the TF-IDF assesses the significance of words in a text. The result of multiplying TF by IDF is a matrix in which every row denotes a document (such as a movie) and every column represents a distinct word. By lessening the impact of often used terms, this technique improves the calculation of similarity scores.

```
tfidf=TfidfVectorizer(stop_words='english',
                                    ngram_range=(1, 1),)

tfidf_vector = tfidf.fit_transform(content_df['overview'])
```

Cosine Similarity:

Cosine similarity calculates the similarity between the items by measuring the cosine of the angle between their vectors, with values ranging from 0 (no similarity) to 1 (identical). (Polamuri, 2015).

The cosine similarity is given by

$$k(x,y) = \frac{xy^T}{\|x\|\|y\|}$$

Because the TF-IDF vectorizer was used, the cosine similarity scores may be obtained straight from the dot product calculation. Since sklearn's linear kernel() is quicker than cosine similarities(), we shall utilise it instead. Here is the linear kernel:

$$k(x,y) = x^T y$$

We then developed a recommendation engine that uses the movie's title as input and suggests ten related films considering the similarities shown by the cosine similarity.

The suggestion system appears as follows:

```
recommender_1('Thor')
```

```
126                    Thor: The Dark World
3539                               Galaxina
14                          Man of Steel
2702                               Jason X
220                             Prometheus
1477                                 Ponyo
1047    Journey to the Center of the Earth
420           Hellboy II: The Golden Army
2239                            Little Boy
476                             Surrogates
```

Credits, Genres, and Keywords-Based Recommender

Using this method, we create suggestions by combining metadata fields such as director, cast, genres, and keywords.

This strategy, in contrast to the overview-based approach, combines numerous metadata fields to form a "tag" feature. The similarity between items is calculated using these tags.

Count Vectorizer

The adoption of the Count Vectorizer in place of TF-IDF is an important change from our previous approach. This is because if an actor or director has acted in or directed a comparatively higher number of films, we do not intend to minimise their presence. (Sklearn, n.d.)

```
count = CountVectorizer(lowercase=True,stop_words='english')

count_matrix = count.fit_transform(content_df['tag'])
```

The similar cosine similarity will then be used to generate suggestions after our tag column has been converted into a matrix. We will utilise cosine similarity in this case rather than linear kernel since we haven't employed TF-IDF, therefore utilising linear kernel and computing the dot product will not yield the result of cosine similarity. (Sklearn, n.d.).

The suggestion system appears as follows:

```
recommender_2('Thor')
```

```
126                  Thor: The Dark World
182                               Ant-Man
7                Avengers: Age of Ultron
85     Captain America: The Winter Soldier
16                           The Avengers
79                            Iron Man 2
26           Captain America: Civil War
169    Captain America: The First Avenger
31                            Iron Man 3
68                              Iron Man
```

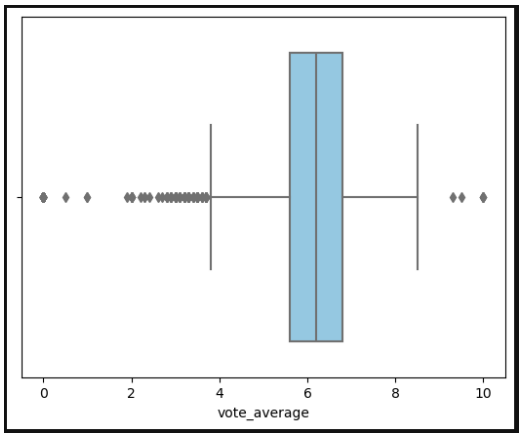
Collaborative Filtering-Based Recommender

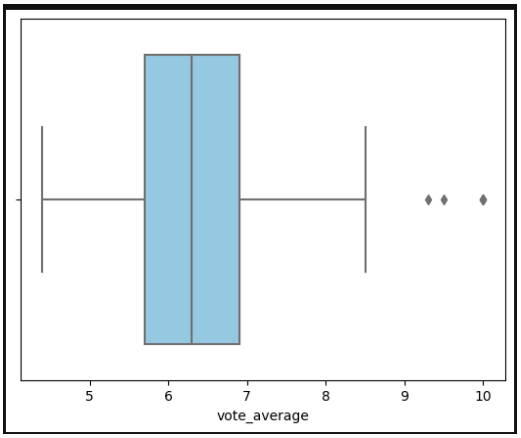
Without depending on item metadata, collaborative filtering is a popular recommender system technique that forecasts user preferences based on past interactions and similarities. It works on the premise that consumers who have previously shown a preference for a certain item are likely to do so again. There are two forms of collaborative filtering: item-based, which discovers related things based on user interactions, and user-based, which finds users with similar likes.

Using a dataset of movie ratings, we will use an item-based collaborative filtering system in this lesson.

Data Preparation

We then created a pivot table among the title columns and user id maintaining the average votes provided by each user as the values for collaborative-based filtering, using the vote average rating provided by each user.





Before and after removing Outliers.

Nearest Neighbours Algorithm

The K-Nearest Neighbours (KNN) is a machine learning algorithm to create projections utilising the average rating of the top k nearest neighbours based on their distances and identify groups of people who are similar based on their book ratings.

Parameters of Nearest Neighbours:

1. **“metric”** establishes how the distance or similarity between objects or users is computed. Because it evaluates similarity regardless of scale, cosine similarity, for instance, is often used and appropriate for sparse rating data.
2. An algorithm, like **"brute"** which calculates distances directly. It is helpful for smaller datasets or situations requiring precise results.
3. **“n_neighbors“,** which regulates how many neighbours are fetched in order to provide suggestions. The diversity and relevancy of ideas are impacted by this; more neighbours promote diversity, whereas fewer neighbours concentrate on extremely similar topics.

```
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute',n_neighbors=5)
model_knn.fit(df_col_matrix)
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

Using the K-Neighbor's distances as a guide, we created a recommendation system that uses the movie's title as input and suggests six comparable films.

The suggestion system appears as follows:

```
recommender_3('Thor')
```

```
Thor
0: Creed
1: Riddick
2: Iguana
3: Top Hat
4: Tomorrowland
5: The Adventures of Tintin
```

Evaluation of Recommendation Systems

To determine a recommendation system's efficacy and dependability, evaluation is essential. Root Mean Squared Error (RMSE), a commonly used statistic for collaborative filtering, measures the discrepancy between expected and actual user ratings. Better forecasts are shown by lower RMSE values.

The RMSE formula is as follows

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y - \hat{y})^2}$$

Where:

y : Actual rating.

$\hat{y}$: Predicted rating.

n : Total number of predictions.

In contrast to MSE, which generates squared units, RMSE provides an error measure in the same units as the data, making it a popular tool for assessing non-binary ratings. It works especially well in situations when precise rating projections for every product are needed. Determining the actual values and similarity values dot product, normalised by the sum of similarities, allowed us to compute the RMSE for our model. Whereas a high RMSE (>2) reveals notable variances and suggests a demand for model modifications, a low RMSE (<1) shows accurate predictions that are closely in line with customer preferences.

```
error_rate = rmse(pred,true_val)
print('Accuracy: {:0.3f}'.format(100 - error_rate))
print('RMSE: {:0.5f}'.format(error_rate))
```

```
Accuracy: 93.478
RMSE: 6.52200
```

Seeing the Root mean square error, this recommender system is performing well as an accuracy of 93% would predict similar movies without many errors

Conclusion

We created two main categories of recommendation systems throughout this lesson: collaborative filtering and content-based filtering. To extract valuable elements like cast, crew, keywords, and genres, we first prepared and cleaned the data. We used methods such as cosine similarity and TF-IDF vectorisation for the content-based approach to suggest movies based on combined characteristics and textual information. We converted user-item data from interactions into a sparse matrix to use in the collaboration-based filtering, then used the K-Nearest Neighbours technique to suggest things based on user behaviour.

Additionally, we used Root Mean Squared Error to assess the collaborative filtering method, which gave us information on how accurate the projected ratings were in comparison to the actual user ratings.

The recommendation system will be improved in the future by creating hybrid models that mix collaborative and content-based filtering to increase variety and accuracy. Using deep learning methods, such recurrent networks or autoencoders, can assist in modelling intricate user-item interactions. While real-time systems can allow for dynamic updates and scalability, suggestions can be further personalised by using contextual information such as time, location, and user preferences. The goal of these developments is to produce a cutting-edge recommendation system that can adapt to the changing requirements of contemporary, customised applications.

References

Kaggle, 2017. *TMDB 5000 Movie Dataset,* s.l.: s.n.

Polamuri, S., 2015. *FIVE MOST POPULAR SIMILARITY MEASURES IMPLEMENTATION IN PYTHON,* s.l.: s.n.

Sklearn, n.d. *Feature extraction,* s.l.: s.n.