Movie recommender system app based on natural language processing

Diana Iulia Bleoancă[1], Costel Ionașcu[2], Marian Cristian Mihăescu[1]
[1]Department of Computers and Information Technology,
University of Craiova, Romania
bdianaiulia@yahoo.com, cristian.mihaescu@edu.ucv.ro
Faculty of Economics and Business Administration,
University of Craiova, Romania
costel.ionascu@edu.ucv.ro

ABSTRACT
Movie recommender systems have evolved significantly by harnessing the power of NLP. Recent research has focused on leveraging deep learning models, sentiment analysis, and multimodal approaches to enhance recommendation accuracy and relevance. In this paper, a new movie recommender system app that is based on NLP is presented. The classical recommender system is extended by processing all the available information about a movie, including text reviews, descriptions, and user behaviour.

Author Keywords
technology, web application, recommender system, natural language processing, user interface, responsive interface

ACM Classification Keywords
I.2.6: Artificial Intelligence: Learning, I.2.7. Natural Language Processing (Text analysis), H.5.m: Miscellaneous.

General Terms
Human Factors; Design; Applied Computing.

DOI: 10.37789/rochi.2023.1.1.5

INTRODUCTION
The current film market is complex and diversified, with various trends and developments that influence how people choose the movies they want to watch. Moreover, people tend to focus on different categories of cinema, a clustering that can be achieved based on many film characteristics: genres, directors, cast, etc. In these circumstances, there is a need for an application that assists the user in managing the movies they have watched or would like to watch, but more than that, it provides other possible perspectives of interest based on various interactions: a particular movie, a history, a search for a keyword, or even choosing a film based on a genre. From the perspective of languages and frameworks used, they were chosen to provide an easy and rapid development mode.
Movie recommender systems play a pivotal role in assisting users in navigating the vast landscape of movies available today.
In order to quickly process information about movies, the use of artificial intelligence algorithms becomes necessary [10, 15, 16].
Leveraging the advancements in NLP, these systems have witnessed significant progress in recent years.
There are more approaches developed by other authors by using Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), in capturing complex patterns and latent features from movie-related textual data, including plot summaries, reviews, and user feedback [12].
Another approach uses a personalised recommendation method to address traditional algorithms' limitations. This approach mixes a deep belief network (DBN) with SoftMax regression to improve the precision of recommendations and better approaches to challenges such as sparse data and cold-start scenarios. The DBN fulfils the task of acquiring complex user and item representations optimising the user-item rating matrix. At the same time, SoftMax regression is employed to predict the likelihood of user-item interactions. The methodology uses negative sampling, resulting in enhanced recommendation efficacy and the generation of precise recommendations. This approach underwent rigorous evaluation on DOUBAN and various MOVIELENS datasets, demonstrating superior performance when contrasted with reference models such as singular value decomposition (SVD). It achieved a mean absolute error (MAE) value of 98%, attesting to its very good accuracy and generalisation capabilities [5]. In addition, a personalised multimodal movie recommendation system was developed to address the challenges like sparse data and cold-start scenarios. This system uses the power of multimodal data analysis and deep learning techniques to highlight latent features of both movies and users. A deep-learning network algorithm model was trained to predict movie ratings, and its accuracy was assessed using MOVIELENS datasets. The results exhibited increased accuracy, with root mean squared error (RMSE) scores of 0.9908 and 0.9096 for the MOVIELENS 100K and 1M datasets, respectively. Remarkably, this multimodal movie recommendation system, empowered by deep learning, surpasses traditional collaborative filtering algorithms like User-CF, Item-CF, and SVD, delivering superior personalised recommendations and mitigating the issues associated with sparse data. This

investigation underscores the potential for enhancing recommendation systems through the fusion of deep learning and multimodal data analysis [6].

A multi-feature attention mechanism with deep neural networks and convolutional neural networks has been proposed. User and movie networks are introduced to learn respective features, and a feature attention mechanism is employed to determine the importance of different parts in movie ratings. Convolutional neural networks are used to extract text features, with an additional attention mechanism to enhance accuracy. The algorithm achieves personalised and accurate movie recommendations. Experimental results substantiate the algorithm's efficacy, unequivocally illustrating that user and movie attribute characteristics influence ratings favourably. Incorporating a feature attention mechanism contributes significantly to discerning the relative significance of these attributes. Moreover, the attention mechanism integrated into the convolutional neural network enhances the precision of text feature extraction, yielding a commendable level of accuracy across various evaluation metrics, including but not limited to Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), R-squared (R2), and Root Mean Squared Error (RMSE) [9].

DATASET

An existing database was used to analyse and include a wide range of movies (along with their data), containing details about 45,466 movies [13]. All the data is stored in .csv files, divided into data categories that will be consumed according to the algorithm's stages:

movies_metadata.csv - From this database, we use the following fields: genres, vote_count, vote_average, and release_date. The genres field helps us make genre-based recommendations based on the rating or popularity of the most-watched movies. From the release_date, we extract the day of release for additional information. Furthermore, we need to use the tagline and overview fields for the algorithm based on movie descriptions and other details.

links.csv - This table provides the movie IDs, which are used to filter the information we want to access.

credits.csv - For the third case, where we analyse movies based on credits, we are specifically interested in characters, actors, directors, etc.

keywords.csv - We use the keywords for each movie from this table, which will be processed later to influence the recommendation algorithm based on movie details.

DATA PREPROCESSING

For the data, we have to be relevant both to the machine and especially to the machine learning algorithms they need to be processed. Besides removing incorrect or incomplete values, the data must be prepared in a specific format, composed, and combined. Data preprocessing is vital in machine learning as it directly impacts our model's ability to learn from the data. Thus, it's crucial to prepare our data carefully before using it in our model.

Data preprocessing involves several tasks, including:

1. Data cleaning involves handling missing values, outliers, and incorrect data. Missing values can be imputed or removed, outliers can be detected and treated accordingly, and inaccurate data can be corrected or removed.
2. Data transformation: This includes transforming data into a suitable format for the algorithms. It may involve scaling numerical data, encoding categorical variables, or normalising the data distribution.
3. Feature engineering: This involves creating new features or selecting relevant features from the existing data that can enhance the model's predictive power.
4. Data integration: Combining multiple datasets or sources enriches the available information and provides a more comprehensive data view.
5. Data splitting: Splitting the data into training and testing sets to evaluate the model's performance on unseen data.

Through the execution of these preprocessing procedures, we can guarantee the cleanliness, uniformity, and appropriate formatting of the data, thereby enhancing the precision and efficiency of our machine learning algorithms.

Handling Null Values

In real-world datasets, null values are invariably present. Regardless of whether the problem is regression, classification, or another type, models cannot autonomously manage these NULL or NaN values. Hence, intervention is essential.

Standardisation

It is another integral step in preprocessing. In standardisation, we transform the values so that the mean of the values is 0 and the standard deviation is 1. Since we have chosen to keep the most important actors, the director will be duplicated the same number of times to not influence the group of actors compared to the film's director just because it is a more extensive list. We want the weighting of the director, as described above, to be equal to that of other significant characters.

Avoiding Multicollinearity

Multicollinearity arises within our dataset when features exhibit substantial interdependence. When multicollinearity is present, using our weight vector for feature importance assessment becomes unfeasible.

Various techniques and libraries are available to handle these preprocessing tasks, such as imputation methods for handling missing values, feature scaling techniques for standardisation, and methods for detecting and dealing with multicollinearity. By applying these techniques, we can ensure that our data is prepared in a way that optimises the performance of the machine learning models and avoids potential issues during the training and evaluation process.

DESCRIPTION OF ALGORITHMS AND INTERMEDIATE VALUES

Recommender System Based on Genre

In this approach, movies are recommended based on the genre and the ratings or popularity of the movies watched by most people. Firstly, we extract the genre values from the dictionary so that only the names are retrieved and put into a list.

The TMDB rating is used to evaluate the top movies. We will use the weighted IMDB rating to construct the graph.

Weighted Rating (WR) = (vv+m.R)+(mv+m.C) where,

- v is the number of votes for the movie
- m is the minimum number of votes required to be listed in the graph
- R is the average rating of the movie
- C is the mean vote across the whole report
- The value of m is determined by taking the 80th percentile as the cutoff. In other words, we consider movies with more votes than 80% of the movies.

Next, we will build two content-based recommendation systems.

Recommendation for a movie based on the description

Recommendation based on the movie description is supported by both the tag lines and the overview, which we paste to create the movie description:

"A single and lonely woman finds the seemingly perfect man to date, but soon regrets it when his deranged and possessive other personality emerges, and worst still, she cannot convince anyone else of his Jekyll/Hyde true nature."

"Ray Liotta stars as a medical examiner who has been acquitted for his wife's murder but many still question his innocence. Obsessed with finding his wife's killer, a possible solution presents itself in an experimental serum designed by a neurobiology Linda Fiorentino which has the ability to transfer memories from one person to another, but not without consequences. Liotta driven to solve the case injects himself with the serum, bringing him closer and closer to finding her killer but bringing him closer to death. He loved her. He lost her. He won't let her memory die... until it tells him who killed her."

Because we have a data set in which words are influenced by their context, we chose to convert the collection of documents into a matrix with TF-IDF properties. TF-IDF stands for "Term Frequency — Inverse Document Frequency". This is a technique for quantifying the words in a set of documents. We calculate a score for each word to signify its importance in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining. We use the list of stop words provided by the library. Since all the descriptions are in English, without any special repetitions that we would like to remove, the default list is sufficient.

Stopwords are words in every language that don't add much meaning to a sentence. They can be safely ignored without sacrificing the meaning of the sentence. For example, words like the, he, have etc. Such words are already captured in the corpus.

tf = TfidfVectorizer(analyzer='word', ngram_range = (1,2), min_df = 0, stop_words = 'english')

The analyser indicates what needs to be analysed in the document, such as words, numbers, and characters. The ngram_range is used to specify the grouping in the tf-idf vectoriser - (1,2) indicates the grouping of letters into unigrams and bigrams. For example, if we consider the document "I study NLP" for bigrams, it will be grouped as "I study" and "study NLP".

tfid_mat = tf.fit_transform(smd['description'])

We thus encode the description in the TFid matrix.

cos_sim = linear_kernel(tfid_mat, tfid_mat)

We employed cosine similarity to measure the similarity between movies. This cosine similarity can be visualised by plotting the data on an XY graph, with the TF-IDF values of each document representing the X and Y axes. Given that the dot product of the cosine similarity has already been computed, we will employ the linear_kernel method to determine similarity.

len(cos_sim[0])

From the above we can see that each record has created a list of values that contains their similarity. To be able to resolve for each search the title index and vice versa, we will create a mapping between the two [Table 1]:

 # A map between titles and movies index
indices = pd.Series(smd.index, index = smd['title'])

Table 1 Mapping between titles and indexes

Movie title	Index
Toy Story	0
Jumanji	1
Grumpier Old Men	2
Waiting to Exhale	3
Father of the Bride Part II	4

Thus, if we access the index, we can request the scores of the other preprocessed films so that we have them best recommendations:

sim_scores = list(enumerate(cos_sim[idx]))
print(sim_scores[:5])[(0, 0.0), (1,0.007774131635450035), (2, 0.0), (3, 0.0), (4, 0.0)]

Since we are only interested in high scores, the next step is to sort the list in descending order and save the first n movies we are interested in.

 # Sort the movie based on the similarity score
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
 # Take first 30 movies (first one is itself)
sim_scores = sim_scores[1:31]

The first index is excluded because it represents the very own index, in which the weight will have a maximum score,

namely 1. From the above we can see a recommendation algorithm, where it found the similarity based on the description and slogan. This recommendation is especially good when we want a sequel to the movie, but let's say if the movie doesn't have a sequel or there isn't a proper description, or even the lack of it.
Then we have to recommend movies based on other criteria like cast, crew, genres and keywords.

Recommendation for a movie based on keywords and plot

To make this recommendation, we need to incorporate extra data from our database, specifically crew information and keywords. This will centre on evaluating the resemblance between two movies based on actors involved in the storyline, directors of photography, and keywords that provide context. Initially, the movie crew data offers us a directory of individuals along with their respective roles and names.
Since we believe that the director plays one of the most important roles in a movie, we will specifically look for him to assign him a separate field.

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

From the cast we only keep a maximum of the first three names [Table 2]:

```
# Take the character names alone and make it as cast then take the first  three cast members
smd['cast'] = smd['cast'].apply(lambda cast_list:
[cast['name'] for cast in cast_list] if isinstance(cast_list, list) else [])
# If list
smd['cast'] = smd['cast'].apply(lambda cast_list:
cast_list[:3] if len(cast_list) >= 3 else cast_list)
```

Table 2 Cast list after processing by movie index

Index	Cast name
0	[Tom Hanks, Tim Allen, Don Rickles]
1	[Robin Williams, Jonathan Hyde, Kirsten Dunst]
2	[Walter Matthau, Jack Lemmon, Ann-Margret]
3	[Whitney Houston, Angela Bassett, Loretta Devine]
4	[Steve Martin, Diane Keaton, Martin Short]

We also process the keywords to get a concatenated list of sequences [Table 3]:

Table 3 List of keywords before processing according to the film index

Index	Keywords
0	[jealousy, toy, boy, friendship, friends, rivalry, boy n...
1	[board game, disappearance, based on children's book, ne...
2	[fishing, best friend, duringcreditsstinger, old men]
3	[based on novel, interracial relationship, single mother...
4	[baby, midlife crisis, confidence, aging, daughter, moth...

Because we don't want the names and surnames of the characters to weigh differently, but we want a unique identification of each person and we want to reduce the possibility of mismatch due to small differences in typography, we process the names so that they are made up only of small letters and without spaces:

```
# Process director and cast names to lowercase and no space
smd['cast'] = smd['cast'].apply(lambda cast_names:
[str.lower(name.replace(" ","")) for name in cast_names])
smd['directors'] =
smd['directors'].astype('str').apply(lambda
director_name: str.lower(director_name.replace(" ","")))
```

Because, at the moment, we have a list of characters and only one director and the weight of the director must be as significant as that of the actors, we will apply a multiplication of his name in a stack of the same size as that of the cast.

Make directors into three stacked lists to match the cast members

```
smd['directors'] = smd['directors'].apply(lambda name:
[name, name, name])
```

Next, we deal with keywords.

```
# Process most important keywords to lowercase stem words with no space
smd['keywords'] = smd['keywords'].apply(lambda keywords:
keep_most_important_keywords(keywords, keywords_movies))
smd['keywords'] = smd['keywords'].apply(lambda keywords:
[stemmer.stem(keyword) for keyword in keywords])
str.lower(keyword.replace("","")) for keyword in keywords])
```

The first important step is to keep the most important keyword corpus separately and remove the records that have fewer keywords. This is because we do not want to analyse keywords that appear uniquely or that do not have such a significant weight. This influences the possibility of false recommendations based on keywords no longer found in other movies.

The second step is extracting the word stem, a thing often encountered in NLP and which helps a lot to reduce the range of possible words encountered and to form some sets of correct words ("playing" and "game" must be part from the same beach of words).

The last step is represented by the processing of the sequences so that the keywords are made up of lowercase letters and without spaces between words [Table 4].

Table 4 List of keywords after processing according to movie index

Index	Keywords
0	[jealousi, toy, boy, friendship, friend, rivalri, boynex...
1	[boardgam, disappear, basedonchildren'sbook, newhom, rec...
2	[fish, bestfriend, duringcreditssting]
3	[basedonnovel, interracialrelationship, singlemoth, divo...
4	[babi, midlifecrisi, confid, age, daughter, motherdaught…

All the information processed so far is pasted into a list of representative sequences for each movie.

```
smd['soup'] = smd['keywords'] + smd['cast']
+smd['directors'] +smd['genres']
smd['soup'] = smd['soup'].apply(lambda x: " ".join(x))
```

We will use the similarity matrix to obtain weights for the word analysis part, similar to the previously described algorithm. But, this time, it is no longer necessary to apply the TF-IDF algorithm because the context no longer influences the words. Hence, a simple application of an occurrence counter is sufficient.

```
count = CountVectorizer(analyzer = 'word', ngram_range =
(1,2), min_df =0, stop_words = 'english')
count_mat = count.fit_transform(smd['soup'])
cos_sim = cosine_similarity(count_mat, count_mat)
```

If we apply as in the previous case a simple sorting, we will have results in which the common elements can be identified [Table 5]:

Table 5 Example result for plot and keyword recommendation for 'The Dark Knight'

Index	Movie title
8031	The Dark Knight Rises
6218	Batman Begins
6623	The Prestige
2085	Following
7648	Inception
4145	Insomnia
3381	Memento
8613	Interstellar
7659	Batman: Under the Red Hood
1134	Batman Returns

Recommendation based on keyword search

We currently have a description and a list of keywords attached. We want to turn them all into a list of words to create our knowledge base and dictionary.

```
def getWordList(x):
rough_wordList = re.sub("[^\w]", " ", x).split()
wordList = []
for word in rough_wordList:
if word not in stop_words:
wordList.append(word)
return wordList
gathered_md['dataset'] =
gathered_md['description'].apply(lambda
description: getWordList(description))
gathered_md['dataset'] = gathered_md['dataset'] +
gathered_md['keywords']
```

For our training model, we need a knowledge base and a dictionary.

```
words_for_dictionary = gathered_md['dataset'].tolist()
dictionary =
gensim.corpora.Dictionary(words_for_dictionary)
bow_corpus = [dictionary.doc2bow(doc) for doc in
words_for_dictionary]
```

Now we have the entire data set transformed into structures that the model we are about to enter is able to interpret in such a way as to transform the words into landmarks for our dictionary.

```
Dictionary(41348 unique tokens: ['Afraid', 'Andy', 'But',
'Buzz', 'Led']...)
Bow_corpus: [[(0, 1), (1, 3), (2, 1), (3, 3), (4, 1), (5, 1), (6,
3), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1),

(13, 1), ..]
```

The knowledge base (bow_corpus) contains the word ID and its frequency in each document. Therefore, an additional detail added to the algorithm is transforming it into the TF-IDF space.

```
# Transform bow_corpus in a tf-idf vector
tfidf = models.TfidfModel(bow_corpus)
corpus_tfidf = tfidf[bow_corpus]
```

Next comes the actual training and indexing of the trained model in matrix form.

```
lsi = models.LsiModel(corpus = corpus_tfidf, id2word =
dictionary,num_topics = 5)
# Compute a similarity matrix, which it's necessary later,
for query
indexList = similarities.MatrixSimilarity(lsi[corpus_tfidf])
```

RESULTS ACHIEVED AND ANALYSIS

The results obtained for each algorithm will be presented and analysed individually in the following section. All movie values relevant for analysis will be considered in the list of responses. Furthermore, comparisons will be made with similar functionalities provided by publicly available applications to assess the results correctly.

Genre-Based Recommendation

The genre-based recommendation algorithm aims to provide the best movies within a specific genre, considering user ratings in proportion to the quantity. To interpret the data, we have referred to reviews provided by both Google and IMDB, considering the average ratings from these platforms as well as the number of users who participated in the voting [Table 6].

Table 6 Results obtained for the Comedy genre

Id	Title	Year	Votes count	AVG score	Popu-larity	Weighted score
351	Forrest Gump	1994	8147	9	34.457024	8.891981
1225	Back to the Future	1985	6239	8	48.307194	7.993442
18465	The Intouchables	2011	5410	8	25.778509	7.991443
22841	The Grand Budapest Hotel	2014	4644	8	16.086919	7.990136
2211	Life Is Beautiful	1997	3643	8	14.442048	7.988516
732	Dr. Strangelove or: How I Learned to …	1964	1472	8	39.39497	7.985378
3342	Modern Times	1936	881	8	9.80398	7.964101
883	Some Like It Hot	1959	835	8	8.159556	7.940554
1236	The Great Dictator	1940	756	8	11.845107	7.937356
10309	Dilwale Dulhania Le Jayenge	1995	661	8	9.241748	7.930978

Through analysis, we can easily observe the following two essential aspects:

- All ratings given to the movies have very high scores, indicating that they can be classified as some of the best films within their respective genres.
- Even though the ratings have a decreasing trend, a strong weighting factor in the recommendation calculation is the number of votes, with the observation that this factor is also consistently decreasing.

Recommendation Based on Movie Properties using a Given Title

The recommendation algorithms based on movie properties aim to provide the best recommendations relative to a given title, analysing various properties of the items [Table 7].

Table 7 Example of a response as a recommendation for the movie "Toy story"

Recommendation based on description		Recommendation based on plot	
10754	Luxo Jr.	3833	Monsters, Inc.
3024	Toy Story 2	7629	Toy Story 3
17551	Cars 2	2522	Toy Story 2
11074	Cars	8595	The Lego Movie
2262	A Bug's Life	6968	Horton Hears a Who!
22126	Toy Story of Terror!	3016	Chicken Run
15519	Toy Story 3	1832	Antz
3336	Creature Comforts	1662	One Hundred and One Dalmatians
4797	Monsters, Inc.	7404	Cloudy with a Chance of Meatballs
1738	Meet the Deedles	1883	A Bug's Life

Although it is about two different algorithms, which analyse different data, a percentage of 40% repetition of the titles can be observed(regardless of the order in the top)[Table 8].

Table 8 Example of a response as a recommendation for the movie "Interstellar"

Recommendation based on description	Recommendation based on plot
115651 Inception	115651 Inception
2486 Following	6981 The Dark Knight
11463 The Prestige	6623 The Prestige
12589 The Dark Knight	3381 Memento

Recommendation based on description	Recommendation based on plot
4126 Memento	8031 The Dark Knight Rises
5302 Insomnia	6218 Batman Begins
18442 The Dark Knight Rises	8983 The Martian
5324 Silent Running	756 2001: A Space Odyssey
10210 Batman Begins	129 Apollo 13
30261 The Martian	8726 The Giver

In this case, a repetition rate of 60% can be observed. For validation,10 different movies were searched on the Google search engine and, by comparing the list of answers of the 2 algorithms and their list of recommendations, we managed to extract the number of movies that coincide [Table 9].

Tabel 9 Coincidence between algorithm responses and those of other platforms

Title	Coincidence algorithm 1	Coincidence algorithm 2
Toy Story	7/10	7/10
Inception	6/10	6/10
The Polar Express	5/10	4/10
Jump Street	7/10	8/10
Harry Potter and the Prisoner of Azkaban	6/10	8/10
300	6/10	4/10
Matrix Revolutions	8/10	6/10
Interstellar	7/10	8/10
Rocky Balboa	7/10	7/10
Django Unchained	8/10	6/10

Although the information is different, the percentage of repetition between the answers is relatively high (over 50% on average), strengthening the algorithms' credibility.
Moreover, we can recognise a large number of recommendations that coincide with those of the Google search platform.

Recommendations for searching by keywords

The keyword recommendation aims to serve the most relevant items based on one or more words [Table 10].

Table 10 Example of response for the keyword "Love"

Id	Title	Year
2370	Rain	1932
5041	Last Tango in Paris	1972
563	The Superwife	1996
5872	L'eclisse	1962
1026	Annie Hall	1977
6443	Dinner with Friends	2001
8255	Take This Waltz	2011
1829	About Last Night	1986
7648	Persuasion	2007
2230	Jules and Jim	1962

CONCLUSION

The current movie recommendation application that utilises recommendation algorithms brings numerous benefits to users and enhances their experience in finding suitable films. Here are some important takeaways:

1. Personalisation and relevant recommendations: The use of recommendation algorithms allows us to provide personalised recommendations based on user preferences and feedback. This significantly improves the user experience and increases the chances of discovering new and interesting movies.

2. Intuitive experience and responsive interface: The current application is designed to provide an intuitive and user-friendly experience. The responsive design allows users to access the application from various devices, regardless of screen size. As a result, users can explore and discover movies anytime and anywhere.

3. Efficiency and scalability: With a dedicated server for managing the database and another server for machine learning algorithms, our application can operate efficiently and handle a large volume of data. This ensures a fast and uninterrupted experience for users, even with an increasing number of users.

4. Continuous improvement of recommendation algorithms: With a dedicated server for machine learning algorithms, we can implement and test new machine learning models and techniques. This allows us to constantly improve the recommendation algorithms, achieving more accurate results and better catering to the needs and preferences of users.

Movie recommendations can be done in various ways, and there is enough data to analyse each movie from multiple perspectives (genre, description, actors, etc.). However, user feedback is the most important missing piece of information that could have led to many improvements. This can be considered as a potential future approach. When comparing algorithms, we can say that the initial recommendations described, where users can search by genre or title, are the most common algorithms on any dedicated platform. The responses are quite accurate, and we can observe the same trend for any search engine where users cannot leave their imprint (Google, IMDB, etc.).

REFERENCES

1. Alexander Clark, Chris Fox, Shalom Lappin. The Handbook of Computational Linguistics and Natural Language Processing, 271-290, 2010.
2. Ben HE, Iadh Ounis, A study of parameter tuning for term frequency normalisation. 2003.

3. D. D. Lewis and K. S. Jones. Natural language processing for information retrieval. Communications of the ACM, 39(1):92–101, 1996.
4. Jonathan J. Webster & Chunyu Kit. TOKENISATION AS THE INITIAL PHASE IN NLP, 1992
5. Li L, Huang H, Li Q, Man J. 2023. Personalised movie recommendations based on deep representation learning. PeerJ Computer Science 9:e1448, https://doi.org/10.7717/peerj-cs.1448
6. Mu, Y.;Wu, Y. Multimodal Movie Recommendation System Using Deep Learning. Mathematics, 2023, 11, 895. https://doi.org/10.3390/math11040895
7. S. Deerwester, S. T. Dumais, T. K. Landauer, G. Furnas, F. d. L. BECK, and L. Leighton-Beck. Improving Information-retrieval with latent semantic indexing.1988.
8. T. Cvitanic, B. Lee, H. I. Song, K. Fu, and D. Rosen. Lda vs lsa: A comparison of two computational text analysis tools for the functional categorisation of patents.In International Conference on Case-Based Reasoning, 2016.
9. Yu, S.; Guo, M.; Chen, X.; Qiu, J.; Sun, J. Personalized Movie Recommendations Based on a Multi-Feature Attention Mechanism with Neural Networks. Mathematics, 2023, 11, 1355. https://doi.org/10.3390/math11061355
10. Yue Kang, Zhao Cai, Chee-Wee Tan, Qian Huang & Hefu Liu. Natural language processing (NLP) in management research: A literature review, 2020.
11. Z. Kastrati, A. Kurti, and A. S. Imran. Wet: Word embedding-topic distribution vectors for mooc video lectures dataset. Data in brief, page 105090, 2020.
12. Zhang et al.: Deep Learning for Recommender Systems: A Survey and New Perspectives, 2019
13. *** https://grouplens.org/datasets/movielens/latest/
14. *** https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6
15. *** https://www.marketingaiinstitute.com/blog/7-key-differences-between-nlp-and-machine-learning-and-why-you-should-learn-both#:~:text=Machine%20learning%20is%20primarily%20concerned,amounts%20of%20natural%20language%20data.
16. *** https://iodinesoftware.com/insights/blog-machine-learning-versus-natural-language-processing-what-is-the-difference/