SAPIENZA
UNIVERSITÀ DI ROMA

MOVIE RECOMMENDATION SYSTEM

Rokshana Ahmed (1994927)
Firdaous Hajjaji (2006406)
Elena Martellucci (1988602)

Sapienza University of Rome
Applied Computer Science and Artificial Intelligence
Deep Learning

ABSTRACT

This report delves into the development of an inclusive movie recommendation system. Our exploration involves experimenting with various embedding techniques, such as Word2Vec, Doc2Vec, TF-IDF Vectorizer, Count Vectorizer, Hash Vectorizer, and GloVE, with the goal of discerning the embedding method that offers superior accuracy and diversity in movie recommendations. The ultimate objective is to enhance the overall user experience.

***Index Terms*—** Word2Vec, Doc2Vec, TF-IDF Vectorizer, Count Vectorizer, Hash Vectorizer, GloVE, Movie Recommendation

1. INTRODUCTION

1.1. Background and Motivation

In today's digital era, the proliferation of movie options presents users with an overwhelming load of possible choices, creating a need for advanced recommendation systems which play a pivotal role in this context by offering tailored suggestions based on user preferences, ensuring a more seamless and enjoyable content discovery process. The motivation behind our study lies in addressing the practical challenges users face in navigating this expansive digital movie ecosystem.

1.2. Objectives of the Study

Firstly, it aims to design and implement a movie recommendation system capable of providing valuable suggestions to users. Secondly, it wishes to conduct a comparative analysis of various embedding techniques to understand their efficacy in enhancing movie recommendations, comparing accuracies and diversities, contributing to an overall improved user experience.

2. LITERATURE REVIEW

2.1. Movie Recommendation and its Application

A movie recommendation system is an ML-based approach to filtering or predicting the users film preferences based on their past choices and behavior.

The primary goal of movie recommendation systems is to filter and predict only those movies that a corresponding user is most likely to want to watch, based on their preference.

2.2. Overview of Word2Vec, Doc2Vec, TF-IDF Vectorizer, Count Vectorizer, Hash Vectorizer and GloVE

In this project, we explore the potential of several embedding techniques. These techniques have been successful in various deep learning tasks and hold promise for enhancing the accuracy of movie suggestions.

- **Word2Vec**: Word2vec is a two-layer neural net that processes text by "vectorizing" words. Its input is a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus, While Word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand. The purpose and usefulness of Word2vec is to group the vectors of similar words together in vectorspace. [1]

- **Doc2Vec**: Doc2Vec is a popular technique in Natural Language Processing that enables the representation of documents as vectors. This technique was introduced as an extension to Word2Vec. It is an unsupervised learning technique that maps each document to a fixed-length vector in a high-dimensional space. [2]

- **TF-IDF Vectorizer (Term Frequency-Inverse Document Frequency Vectorizer)**: The process of transforming text into a vector is commonly referred to as text vectorization and TF-IDF is one of the popular approaches. It consists of two parts:

Term Frequency (TF) measures how frequently a term occurs in a document. It helps in identifying the importance of a term within a specific document.

$$\text{TF}(t, d) = \frac{\text{N° of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d} \quad (1)$$

Inverse Document Frequency (IDF) measures how important a term is across the entire corpus of documents. Terms that occur frequently across many documents will have a lower IDF, while terms that occur rarely will have a higher IDF, indicating their importance.

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total n° of documents in corpus } N}{\text{N° of documents containing term } t}\right) \quad (2)$$

Once both TF and IDF are calculated, TF-IDF is computed by multiplying TF and IDF:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \quad (3)$$

Overall, TF-IDF assigns higher weights to terms that are frequent within a specific document but relatively rare across all documents, aiming to capture the distinctive characteristics of each document in the corpus. [3]

- **Count Vectorizer**: Count Vectorizers is yet again another numerical representations of words. It transforms a string into a frequency representation. The text is tokenized and very rudimentary processing is performed. The objective is to make a vector with as many dimensions as there are distinct words. Each unique word has its own dimension, which will be represented by 1 in that dimension and 0 in all others. It differs from TF-IDF as it only examines the appearance of a word in a single document instead of the whole document. [4]

- **Hash Vectorizer**: Hash Vectorizer does the same thing as the count vectorizer but the key difference is that it doesn't store the resulting vocabulary (i.e. the unique tokens). With Hash Vectorizer, each token directly maps to a column position in a matrix, where its size is pre-defined. For example, if you have 10,000 columns in your matrix, each token maps to 1 of the 10,000 columns. This mapping happens via hashing. The hash function used is called Murmurhash3.

  Not having to save the vocabulary is very efficient for a large dataset because the resulting HashingVectorizer object when saved, would be much smaller and thus faster to load back into memory when needed.

  The downside of doing this is that it will not be possible to retrieve the actual token given the column position. [5]

- **GloVE (Global Vectors for Word Representation)**: GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well. For this reason, the resulting word vectors perform very well on word analogy tasks. [6]

3. DATA COLLECTION AND PREPROCESSING

3.1. Movie IMDB Database

The dataset, obtained from Kaggle, is labeled as the "TMBD 5000 Movie Dataset", which actually includes 4803 items. Spanning 24 features, this dataset offers a comprehensive range of details, including movie ID, title, cast members, producers, release year, and assorted attributes. [7]

3.2. Preprocessing Techniques Applied

During the preprocessing phase, we extracted only the most relevant columns/features for our analysis. Out of the numerous attributes available, we specifically selected five key features: 'genres', 'keywords', 'tagline', 'cast', and 'director'. Once identified, we took measures to handle missing values by replacing them with null strings, ensuring data completeness and consistency and we combined them into a single cohesive text representation, which served as the basis for embedding techniques and it enabled us to capture rich semantic information.

Overall, the preprocessing phase was instrumental in refining our dataset, focusing on key features relevant to our objectives, and preparing the data for subsequent embedding techniques and recommendation system development.

4. TRAINING MODELS; THEIR CHALLENGES AND SOLUTIONS

4.1. Text to Vector Techniques

4.1.1. Vectorizer: TF-IDF, Count,Hash

Using a vectorizer like TF-IDF (Term Frequency-Inverse Document Frequency), Count Vectorizer, or Hashing Vectorizer can greatly enhance a movie recommendation system by converting textual features (such as movie titles, descriptions, genres, etc.) into numerical representations. These representations can then be used to calculate similarity scores between movies, enabling more accurate recommendations.

1. **TF-IDF Vectorizer:** TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. In a movie recommendation system, TF-IDF can be applied to movie titles, descriptions, or even user reviews. Each movie is represented by a vector where each dimension corresponds to a term in the vocabulary, and the value represents the TF-IDF score of that term in the movie. TF-IDF tends to assign higher weights to terms that are unique to a movie or are important discriminators. This can be seen in figure 1, wherein which it demonstrates the distribution. As we can observe, the graph is exponentially decreasing, meaning that it will give us a few movies that are very similar to the one we input, some that may not be the same but may have some features in common (i.e. 2% <similarity <10%) and some that are completely different (i.e. similarity = 0%). Overall this vectorization scheme helps in capturing the semantic meaning of movie descriptions or reviews, thus improving the accuracy of recommendations.

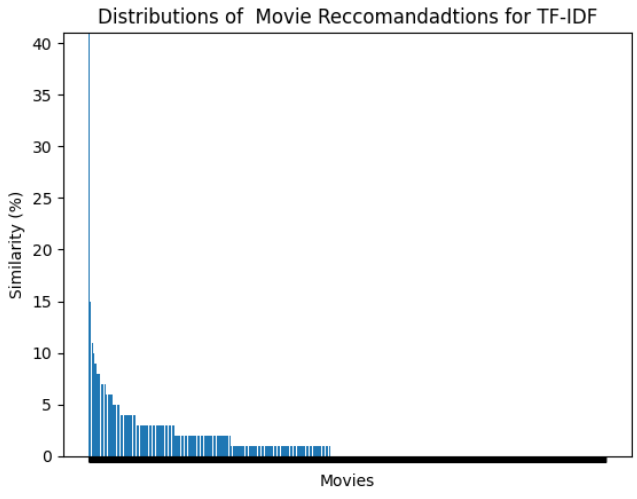

Fig. 1. Distribution of the Tf-IDF Vectorizer model for All the movie recommendations with the input "Iron Man"

2. **Count Vectorizer:** Count Vectorizer simply counts the occurrences of each word in a document. It creates a sparse matrix where each row represents a document (or movie) and each column represents a word, with the cell values indicating the frequency of each word in the corresponding document. In the context of movie recommendation, Count Vectorizer can be applied similarly to TF-IDF, but without considering the importance of words. Figure 2, demonstrates its distribution. As you can see, the graph is exponentially decreasing, meaning that it will give us a few movies that are very similar to the one we input, some that may not be the same but may have some features in common (i.e. 5% <similarity <15%), but it is higher than the TF-IDF for the equivalent movies and this is, as mentioned above, because the Count Vectorizer doesn't take into account the importance of words. Then there are some movies that it considers completely different (i.e. similarity = 0%). This vectorizer is useful when the frequency of words is important for similarity calculation, such as identifying movies with similar plots or themes.

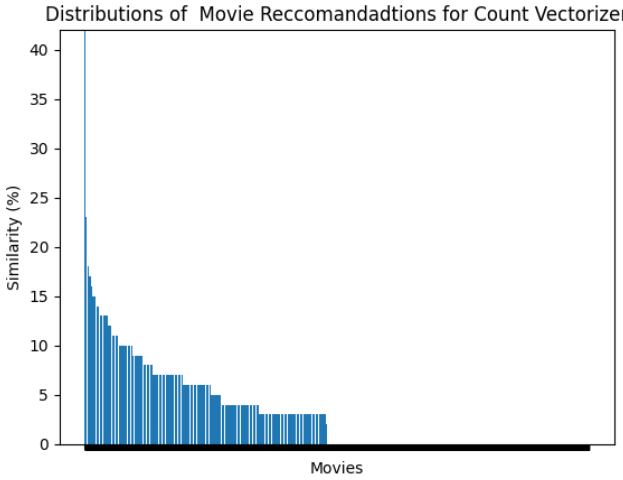

Fig. 2. Distribution of the Count Vectorizer model for All the movie recommendations with the input "Iron Man"

3. **Hashing Vectorizer:** Hashing Vectorizer applies a hashing function to tokenized text, converting them into fixed-size vectors. Unlike TF-IDF and Count Vectorizer, Hashing Vectorizer doesn't require storing a vocabulary dictionary in memory, making it memory-efficient. Each word is hashed into a fixed number of buckets, and the resulting hash values are used as indices in the vector representation. This can be seen in figure 3, which shows its distribution. As you can see, the graph is exponentially decreasing, meaning that it will give us a few movies that are very similar to the one we input, some that may not be the same but may have some features in common (i.e. 5% <similarity <15%).

This is very similar to the Count Vectorizer. But, while a Hashing Vectorizer is memory-efficient and fast, it may lead to collisions where different words are hashed to the same index, potentially losing some information. Overall, in a movie recommendation system, Hashing Vectorizer can be useful when dealing with a large vocabulary size or limited memory resources.

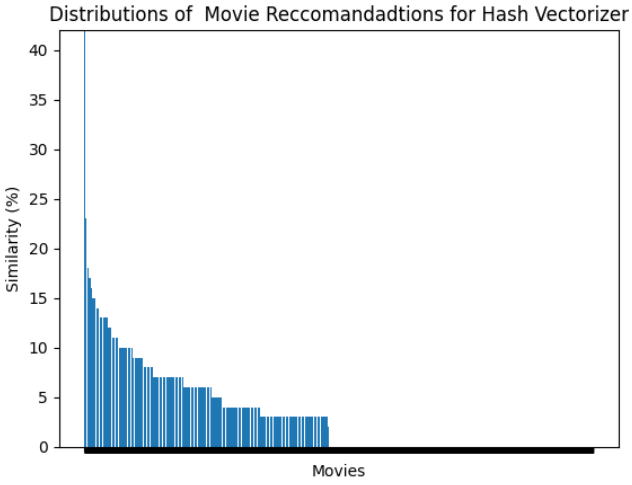

Fig. 3. Distribution of the Hash Vectorizer model for All the movie recommendations with the input "Iron Man"

4.1.2. Word2Vec, Doc2Vec

1. **Word2Vec:** Word2Vec is a technique that learns dense vector representations (embeddings) for words in a continuous vector space. By training on a large corpus of text data (e.g., movie titles, synopses, reviews), Word2Vec can capture semantic relationships between words. Words with similar meanings or usage contexts tend to have similar vector representations. But as we can see Within figure 4, all the movies are considered identical (i.e. similarity= 100% for all movies). This is likely because as mentioned in section 3.2 we already perform preprocessing, where we filter what data for our analysis. Thereby removing a large chunk of our dataset where Word2Vec can capture the relationship between words. Also due to how our dataset is set up the features get combined into one long string (e.g. Avatar gives the string "Action Adventure Fantasy Science Fiction culture clash future space war space colony society Enter the World of Pandora.") This is not ideal for Word2Vec due to repetitive phrases like "space war" and "space colony society." Repetitive phrases can bias the model and skew the learned embeddings, leading to less diverse or meaningful representations.

2. **Doc2Vec:** Doc2Vec captures the semantic meaning of documents by learning vector representations that encapsulate the context and meaning of the entire document. But again, as we can see Within figure 5, we can see that it considers almost all movies identical (i.e. similarity= 100% for all movies). This is likely because as mentioned in section 3.2 we already perform preprocessing, where we filter what data we want for our analysis. By doing so, we are removing a large chunk of our dataset where Doc2Vec can capture the relationship between words. Also due to how our dataset is set up the features get combined into one long string (e.g. Avatar gives the string "Action Adventure Fantasy Science Fiction culture clash future space war space colony society Enter the World of Pandora.") This is not ideal for Doc2Vec due to repetitive phrases like "space war" and "space colony society." Repetitive phrases can bias the model and skew the learned embeddings, leading to less diverse or meaningful representations.

   Overall, Doc2Vec gives a slightly larger variation between different movies compared to Word2Vec, but the similarity is still too high.

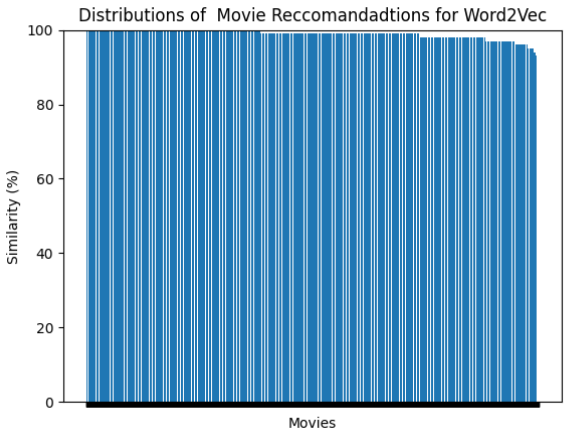

Fig. 4. Distribution of the Word2Vec model for All the movie recommendations with the input "Iron Man"

4.1.3. Problems involving Word2Vec and Doc2Vec

Overall, since both Word2Vec and Doc2Vec have high similarity for every single movie, we can conclude that they would not be adequate to use for a Movie Recommendation System because the similarity between all the movies is identical, therefore the probability that we get a random movie instead of a recommended movie is very high.

4.1.4. GloVE

GloVe embeddings enable the calculation of semantic similarity between words and phrases. Movies with similar titles,

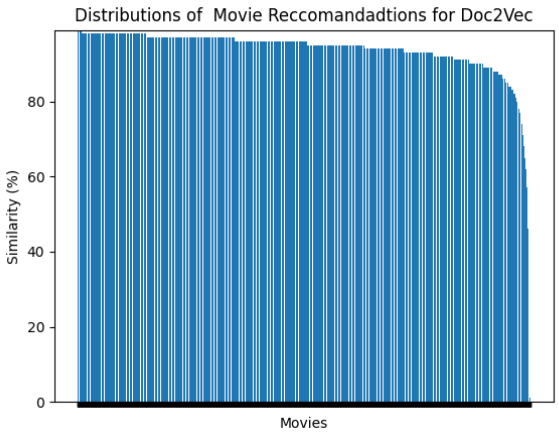

Fig. 5. Distribution of the Doc2Vec model for All the movie recommendations with the input "Iron Man"

descriptions, or themes can be identified based on the similarity of their GloVe embedding representations. Semantic relationships captured by GloVe embeddings allow the recommendation system to go beyond exact keyword matches and understand the underlying meaning and context of movie titles and descriptions. This can be seen in figure 6, wherein which it demonstrates the distribution. As you can see, the graph is decreasing gradually with only steep descents for the top and bottom 20 movies. This means that it will give us a few movies that are very similar to the one we input (i.e. similarity >50%), some that have quite a lot of features in common (i.e. 30% <similarity <50%) and some that are completely different (i.e. similarity <30%).

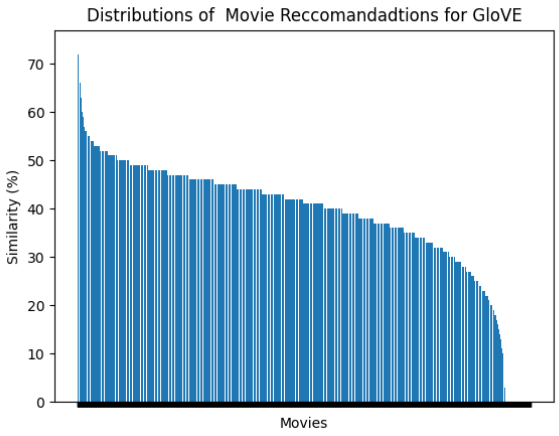

Fig. 6. Distribution of the GloVE model for All the movie recommendations with the input "Iron Man"

4.2. Validating using Cosine Similarity

Cosine similarity is a measure used to determine the similarity between two non-zero vectors in a multidimensional space, typically in the context of text mining, information retrieval, and machine learning. It calculates the cosine of the angle between the two vectors, providing a numerical value indicating how similar or dissimilar the vectors are.

Here's how cosine similarity works:

Vector Representation: Each entity (e.g., document, word, or any object) is represented as a vector in a multidimensional space. This representation can be based on various features or attributes, depending on the context of the problem.

Cosine Calculation: The cosine similarity between two vectors $\mathbf{a}$ and $\mathbf{b}$ is calculated using the formula:

$$\text{cosine similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$$

Where:

- $\mathbf{a} \cdot \mathbf{b}$ denotes the dot product of the vectors.
- $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ represent the magnitudes (or lengths) of the vectors $\mathbf{a}$ and $\mathbf{b}$, respectively.

Interpretation: The resulting cosine similarity value ranges from -1 to 1:

- If the vectors are identical, the cosine similarity is 1.
- If the vectors are orthogonal (perpendicular), the cosine similarity is 0, indicating no similarity.
- If the vectors point in opposite directions, the cosine similarity is -1, indicating dissimilarity.

5. EXPERIMENTAL RESULTS AND DISCUSSION

5.1. Model Comparison

Previously we discussed the models individually, but comparing them gives us a good idea what differing movies they recommend and their differing distributions. Figure 7 shows us the distribution of all movies for all models.

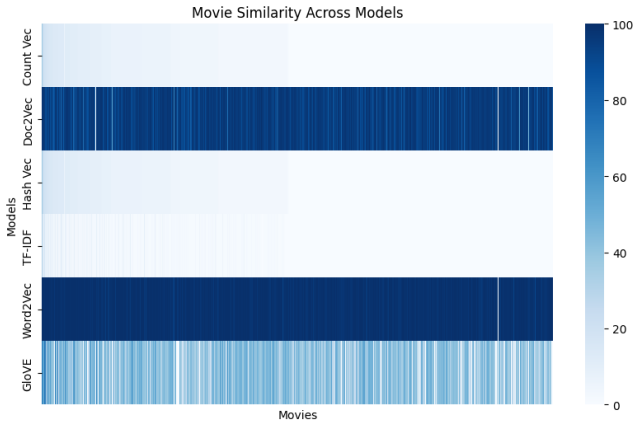

Fig. 7. Comparison of different models when asking the model to give us All the movie recommendations with the input "Iron Man"

Figure 7 demonstrates that compared to the other models Word2Vec and Doc2Vec have no difference across all the movies (i.e. similarity = 100%) But Doc2Vec fluctuates a bit more, as seen in the differing shades of dark blue. The figure also demonstrates that Hash Vectorizer, Count Vectorizer and TF-IDF are very similar both with what movies they select and their overall distribution. On the other hand, GloVE has a higher similarity across the entire dataset,but it still has the same movies (as the vectorizers) for the highest similarities.

To specifically view exactly what movies a model recommends and how it differs from other models; we show the top 30 movies for each model (seen in figure 8)

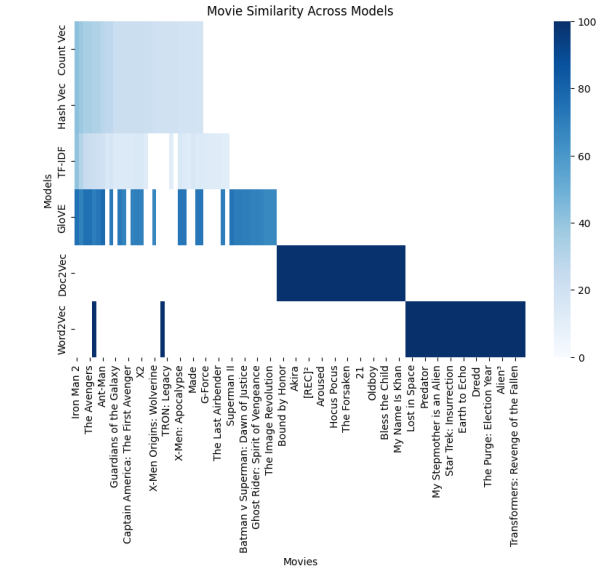

Fig. 8. Comparison of different models when asking the model to give us Top 30 movie recommendations with the input "Iron Man"

As you can see, due to their random nature Doc2Vec and Word2Vec give completely different movies compared to all the other models and can be considered outliers.

So, as mentioned in section 4.1.2 and as seen in figure 8. The models Word2Vec and Doc2Vec are not appropriate for our movie recommendation system. Therefore the following figure 9 does not include them.

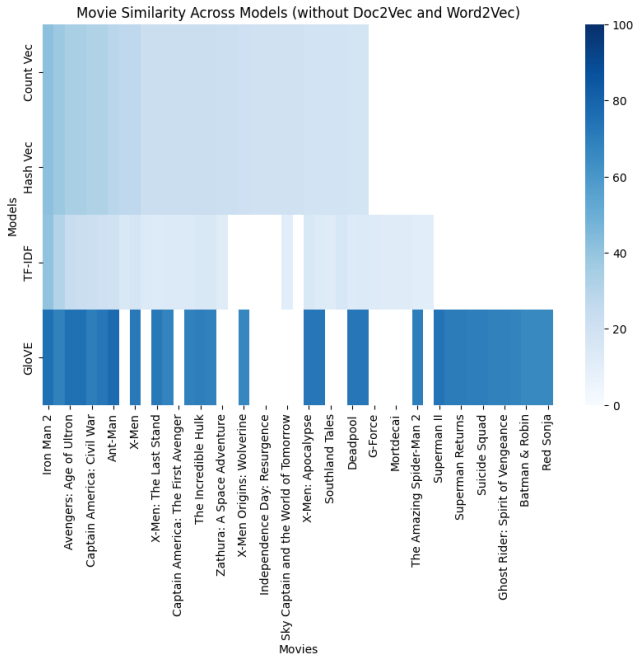

Fig. 9. Comparison of the models; GloVE, TF-IDF Vectorizer, Hash Vectorizer and Count Vectorizer, when asking them to give us Top 30 movie recommendations with the input "Iron Man"

Here we can see that there is a slight difference between TF-IDF and Count/Hash Vectorizer. Where Hash Vectorizer and Count Vectorizer are identical in their top 30 movies. Whilst TF-IDF has the same movies for the high similarities, as the similarity decreases it differs in what movies it recommends. Comparatively, GloVE selects a wider array of movies, but they still overlap with those of the other models and its movies with the highest similarity are the same as those from the vectorizers.

5.2. Example Outputs

When running the movie recommendation system, a user can ask to also visualize a graphical interpretation of their movie recommendation. Below you will see an example for both TF-IDF and GloVE.

Figure 10 shows the top 30 for TF-IDF, including the names of the movies.

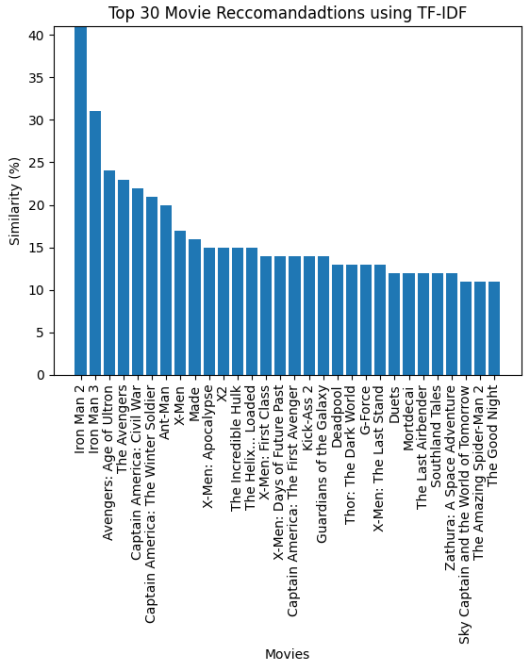

Fig. 10. Top 30 for TF-IDF when inputting "Iron Man" into our movie recommendation system

Figure 11 shows the top 30 for GloVE, including the names of the movies.

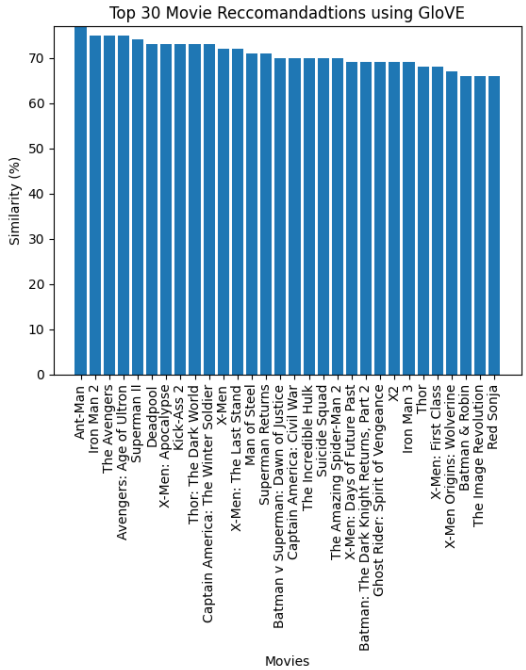

Fig. 11. Top 30 for GloVE when inputting "Iron Man" into our movie recommendation system

Comparing figure 11 and 10 we can see that, TF-IDF tends to give the sequels a higher similarity (i.e. Iron Man 2 and Iron Man 3). Whilst GloVE tends to give movies with similar themes (i.e. Ant-Man [another superhero origin story]) a higher similarity. But overall both of them tend to show the same movies within their Top 30.

5.3. The Models' Appropriate Applications According to our Results

As already hinted at in section 5.2. The two models that are most appropriate according to our results are **TF-IDF** and **GloVE**. This is because, between the vectroizers, TF-IDF applies the benefits of both Count and Hash whilst taking into account the inverse document frequency (see formula 3) and because GloVE isn't exponential, it allows us to have a wider range of movie recommendations. This means that if we want a larger number of movie recommendations, this would be the appropriate model.

Overall we can deduce that when we want a higher volume of movie recommendations we need to use GloVE and when we want few movie recommendations

6. CONCLUSION AND FUTURE WORK

In summary, in this movie recommendation project, we explored and compared six different models for representing textual features and generating recommendations: Count Vectorizer, Hashing Vectorizer, TF-IDF Vectorizer, Word2Vec, Doc2Vec, and GloVe. Each model offers unique advantages and considerations for building a recommendation system tailored to movie titles and descriptions.

In the future, we could; explore hybrid approaches that combine the strengths of multiple models to further improve recommendation accuracy and diversity. For example, integrating GloVe embeddings into TF-IDF, so that the new models could leverage both semantic meaning and term frequency information.

7. REFERENCES

[1] Chris V. Nicholson, "A beginner's guide to word2vec and neural word embeddings," *Pathmind*, 2023.

[2] ram9119, "Doc2vec in nlp," *Geeksforgeeks*, Dec 2020.

[3] Yassine Hamdaoui, "Tf(term frequency)-idf(inverse document frequency) from scratch in python .," *Towards Data Science, Medium*, Dec 2019.

[4] Shubham Shankar, "Natural language processing - understanding count vectorizer and tf-idf," *Linkedin*, July 2021.

[5] Kavita Ganesan, "Hashingvectorizer vs. countvectorizer," *Kavita Ganesan Blog*.

[6] Christopher D. Manning Jeffrey Pennington, Richard Socher, "Glove: Global vectors for word representationg," *NLP Standford Project*, 2014.

[7] Madhan singh, "Tmdb 5000 movie dataset," *Kaggle*, 2022.