Evaluating Sparse, Dense, and Hybrid Retrieval for Movie Recommendations

Abbas Alubeid
abbal179@student.liu.se

Abstract

Collaborative filtering often dominates modern movie recommendation systems but requires extensive user data, while simpler content based methods rely on shallow metadata matching like ratings and keyword matching. This project investigates a content based alternative using only movie overview summaries. The goal is to automatically recommend K relevant movies given a single movie from a dataset. The project compares sparse retrieval (**TF-IDF**, **BM25**), dense retrieval (**Sentence Transformers**), and a hybrid approach. The results demonstrate that dense retrieval outperforms both sparse and hybrid methods in precision, though at the cost of lower content diversity.

1 Method

All code for this project can be found here.

1.1 Data Preprocessing

This project used the **TMDB 5000 Movie Dataset** (Kaggle, Inc.). The dataset required no major preprocessing steps. The primary preprocessing involved excluding non-English movies, merging the relevant files into a single dataset, and preparing text-specific fields for retrieval.

The text data was consolidated into a single field used for retrieval. Two configurations were prepared to measure the value of metadata:

1. **Basic Content**: A combination of the *Title* and *Overview* of a movie.

2. **Basic Content + *Keywords***

1.2 Sparse Retrieval Models

Sparse retrieval works by finding exact keyword matches between movies. Two standard algorithms were used.

1.2.1 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical method that judges how important a word is to a document (Manning et al., 2008). TF-IDF works by multiplying two components: term frequency (how often a word appears in a document) and inverse document frequency (how rare the word is across all documents). Words that appear frequently in one document but rarely in others get higher scores, making them more useful for distinguishing documents. Before TF-IDF can compute word weights, the raw text must be preprocessed into tokens. This involves splitting text into words, normalizing them, and filtering out noise. This preprocessing can be done in different ways, and different preprocessing steps lead to different results depending on the data. For example, lemmatization normalizes word forms (running $\rightarrow$ run) which improves matching but may lose meaningful distinctions, stopword removal reduces noise from common words like "the" and "is" but may remove meaningful words like "it" in the movie title *It*, and n-grams capture multi-word phrases like *Star Wars* but increase vocabulary size and may create noise from unrelated adjacent words like "the man". When working with movie overview descriptions, it was unclear which preprocessing choices would matter most. To evaluate this, 11 different setups were tested, including toggling lemmatization, stopword removal, bigrams, minimum token length, and alphabetic filtering. Although a complete factorial study would be more thorough, to save time and resources and see general trends, individual factors were varied one at a time from a random baseline configuration.

1.2.2 BM25

BM25 (Best Matching 25) is a probabilistic ranking function that builds on TF-IDF (Robertson, 2025). Similar to TF-IDF, BM25 weighs terms by their frequency and rarity, but adds two improvements:

term frequency saturation (controlled by parameter k_1) which prevents very frequent terms from dominating, and document length normalization (controlled by parameter b) which adjusts for varying document lengths. Since BM25 operates on tokens like TF-IDF, the same preprocessing decisions apply. Using the same 11 preprocessing configurations tested for TF-IDF with additional testing of different k_1 and b values.

1.3 Dense Retrieval Models

The main limitation of sparse methods is the "vocabulary mismatch" problem (Onal et al., 2016). If a query mentions "space" but the document says "galaxy," sparse methods will miss the match since they rely on exact term overlap. This issue is usually addressed with **embeddings**, which are vector representations of text. In the embedding vector space, similar ideas are mathematically close to each other. One famous example is **Word2Vec** which showed that vector arithmetic could capture semantic relationships between words (e.g., "Brother" - "Man" + "Woman" $\approx$ "Sister") (Mikolov et al., 2013).

This project uses four embedding models. Three of them are different variants of **Sentence Transformers** (also known as SBERT) models (Reimers and Gurevych, 2019). These models are based on the BERT architecture (Devlin et al., 2019) that are built based on the encoder part of the **Transformer** architecture (Vaswani et al., 2017). Unlike BERT, SBERT produces sentence-level embeddings optimized for semantic similarity between complete sentences rather than words. The fourth embedding model, **EmbeddingGemma**, is a newer model from Google built on the **Gemma 3** architecture, designed specifically for text retrieval tasks (Schechter Vera et al., 2025).

1.4 Hybrid Approach

The hybrid approach combines the exact keyword matching of sparse retrieval from BM25 with the semantic understanding from EmbeddingGemma.

The different models operate on different scales. BM25 scores are theoretically unbounded and depend on document length and term frequency, while semantic scores are based on cosine similarity and are bounded between -1 and 1 (usually 0 and 1). To address this, both score distributions are normalized using Min-Max normalization to map them to a common range of [0, 1] before combination.

Once normalized, the scores are combined using a linear weighted average:

$$Score_{hybrid} = \alpha \cdot Score_{BM25} + (1-\alpha) \cdot Score_{semantic} \tag{1}$$

Here, α is a hyperparameter that controls the balance between the two methods. An α of 1.0 results in pure BM25 retrieval, while an α of 0.0 results in pure semantic retrieval. During the project, α was set to 0.5 to give equal weight to both keyword precision and semantic context.

1.5 Recommendation Logic

Before recommendations can be generated, the system must process the movie dataset to build a searchable index. For sparse models (TF-IDF and BM25), this process builds a vocabulary from the entire corpus and calculates necessary statistics like term frequency and inverse document frequency. For dense models, this process generates a fixed-size vector (embedding) for each movie. In all cases, a similarity matrix is pre-computed and stored in memory to ensure that looking up recommendations is instant.

Once the fitting and encoding phase is complete, the recommendation process is identical across all methods. It starts by locating the query movie (with title) in the dataset and retrieving the pre-computed list of similarity scores between that movie and every other movie. The top-scoring movies are then returned as recommendations. The methods differ only in how these similarity scores are calculated.

TF-IDF and Semantic Scores: Both the TF-IDF and Semantic models use **Cosine Similarity** to measure the relationship between movies. This metric calculates the cosine of the angle between two non-zero vectors in a multi-dimensional space. For a query vector A and a movie vector B, the score is:

$$\text{Score}(A, B) = \frac{A \cdot B}{\|A\| \|B\|} \tag{2}$$

This results in a score between -1 and 1 in general, but in practice both TF-IDF and sentence embeddings yield scores between 0 and 1, where 1 indicates identical orientation (maximum similarity).

BM25 Score: The BM25 model calculates scores using the BM25 ranking function. Unlike Cosine Similarity, this is a probabilistic retrieval framework that sums the weights of query terms found in a document (Robertson and Zaragoza, 2009).

Hybrid Score: The Hybrid model calculates a composite score by taking the weighted average of the normalized BM25 and semantic scores, as defined in Equation (1).

1.6 Evaluation Framework

To measure how good the recommendations are, the movie genres were used as the ground truth. The assumption is that a good recommendation should share at least one genre with the query movie. For example, if a user queries an action movie, recommending another action movie is considered relevant.

Three metrics were used to evaluate the recommendations:

Precision@K: Precision@K measures the fraction of recommended movies that share at least one genre with the query movie. For a set of K recommendations, it is calculated as:

$$\text{Precision@K} = \frac{1}{K}\sum_{i=1}^{K} I(G_{r_i} \cap G_q \neq \emptyset) \quad (3)$$

where G_q is the set of genres for the query movie, G_{r_i} is the set of genres for the i-th recommended movie, and $I(\cdot)$ is an indicator function that equals 1 if the intersection is non-empty and 0 otherwise. This answers: "of the movies recommended, how many are actually relevant?". A higher Precision@K score is better, as it indicates a greater number of relevant recommendations in the top results.

Jaccard@K: While Precision@K only checks for any genre overlap, Jaccard@K measures the degree of similarity. For each recommendation r_i, the Jaccard similarity between its genres (G_{r_i}) and the query genres (G_q) is calculated as:

$$\text{Jaccard}(G_{r_i}, G_q) = \frac{|G_{r_i} \cap G_q|}{|G_{r_i} \cup G_q|} \quad (4)$$

The final Jaccard@K score is the average Jaccard similarity across all K recommendations. This metric rewards recommendations that match more of the query's genres, not just one. This is because movies are often not clearly defined by a single genre, but by a combination that creates a distinct theme. For example, if the query movie is an action–comedy such as *Rush Hour*, recommending another action–comedy will score higher than a purely action film like *Die Hard*, even though both share the action genre. A higher Jaccard@K score is better, as it shows that the recommended movies align more closely with the specific genre combination of the query.

Content Diversity@K: High relevance alone is not enough if all recommendations are nearly identical. For example, a system that recommends near-duplicate movies or multiple sequels of the same series (e.g., *Iron Man 1* and *Iron Man 2*) may achieve high precision, yet provide limited opportunity for user discovery. Content Diversity@K addresses this by measuring how different the recommended movies are from one another. For each pair of recommendations, their semantic similarity is computed using sentence embeddings from the **all-MiniLM-L6-v2** sentence transformer model. The diversity score is the average pairwise cosine distance:

$$\text{Diversity@K} = \frac{1}{\binom{K}{2}}\sum_{i<j}(1 - \text{sim}(e_{r_i}, e_{r_j})) \quad (5)$$

where e_{r_i} and e_{r_j} are the sentence embeddings for recommendations r_i and r_j, $\text{sim}(\cdot,\cdot)$ denotes cosine similarity, and $\binom{K}{2} = \frac{K(K-1)}{2}$ is the number of unique pairs. A higher diversity score indicates more varied recommendations.

This approach is inspired by BERTScore (Zhang et al., 2020), which uses contextual embeddings to measure semantic similarity. BERTScore is typically applied to evaluate text generation and similarity, whereas the same principle is applied here to measure recommendation diversity in original text.

Finally, all metrics were evaluated at K=5 and K=10 on a test set of 2000 randomly sampled movies.

2 Result

2.1 Data overview

Genre distribution The dataset shows a significant class imbalance in movie genres. As shown in Figure 1, `Drama` (2,098 movies) and `Comedy` (1,647 movies) are the most dominant genres, accounting for a large portion of the dataset. In contrast, genres like `War` (122), `Western` (76), and `TV Movie` (8) are significantly underrepresented.

Overview summary lengths The length of overviews varies across the corpus, ranging from as few as 4 words to 175 words as shown in Figure 2. The distribution is relatively broad, with frequent counts around 25 to 60 words.

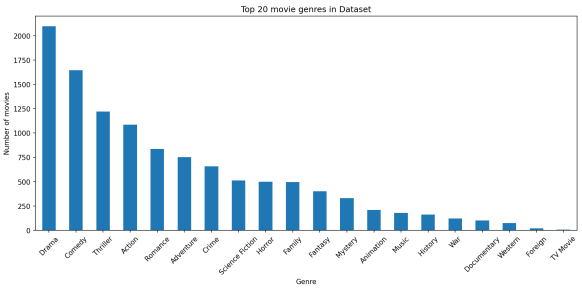

Figure 1: Top 20 movie genres in the dataset. `Drama` and `Comedy` significantly outnumber other categories.

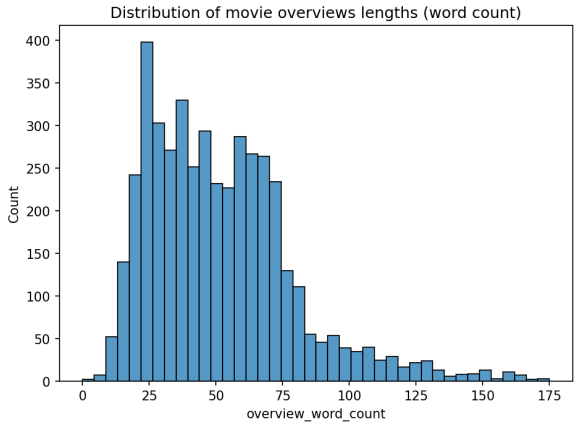

Figure 2: Distribution of movie overviews lengths (word count).

After all preprocessing, a total of 4504 movies were in the final dataset

2.2 Sparse retrieval optimization

To optimize the sparse models described in Section 1.2, 11 preprocessing variants were tested against a randomly selected baseline. The baseline configuration used unigrams, lemmatization, stopword removal, a minimum token length of 3, and an alphabetic filter.

As shown in Figure 3, the performance variance was negligible across all configurations. This confirms that for this dataset, distinct preprocessing steps offer little benefit over the choice of the retrieval algorithm itself.

Across all preprocessing and hyperparameter configurations, both models maintained an average query latency of approximately 3 ms.

2.3 Semantic model selection

Four dense retrieval models were evaluated, ranging from the lightweight all-MiniLM-L6-v2 to the large-scale EmbeddingGemma-300m.

The results show a clear advantage for larger

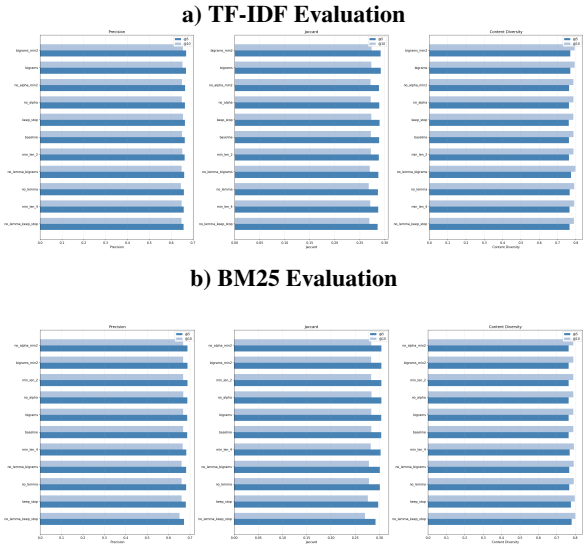

Figure 3: Performance of preprocessing variants for sparse retrieval.

models as illustrated in Figure 4. EmbeddingGemma achieved the best performance across all metrics except encoding time. The query time for all models was around 3 ms; however, the encoding time for EmbeddingGemma was much higher at around 255 seconds compared to around 10-70 seconds for the other models to encode all 4504 movies.

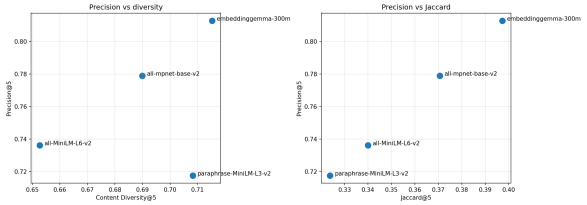

Figure 4: Performance comparison of semantic models.

2.4 Final Method Comparison

Based on the evaluations above, four methods were compared: TF-IDF and BM25 with the best preprocessing from the evaluation presented in Section 2.2, even though result shows that preprocessing had minimal impact on performance, a choice had to be made for this step. For semantic retreival EmbeddingGemma were chosen, and lastly, a hybrid combining BM25 with EmbeddingGemma ($\alpha = 0.5$). All methods used the basic content configuration from Section 1.1 in this evaluation step.

As shown in Table 1, the Semantic (EmbeddingGemma) model significantly outperforms all other methods.

Method	P@5	P@10	J@5	Div@5
TF-IDF	0.669	0.654	0.292	0.770
BM25	0.688	0.668	0.304	0.765
Semantic	**0.813**	**0.803**	**0.397**	0.715
Hybrid	0.790	0.779	0.382	0.718

Table 1: Performance comparison based on title + overview. P@K = Precision@K, J@K = Jaccard@K, Div@K = Content Diversity@K.

2.5 Impact of keywords

All methods were re evaluated using the the other data configuration from Section 1.1 which is basic content (title + overview) + keywords.

Method	Without KW	With KW	Change
TF-IDF	0.669	0.730	+9.1%
BM25	0.688	0.755	+9.7%
Semantic	0.813	**0.833**	+2.5%
Hybrid	0.790	0.819	+3.7%

Table 2: Impact of adding keywords (KW) on precision@5.

Adding keywords improved all methods, with sparse methods benefiting most (BM25 +9.7%, TF-IDF +9.1%) compared to semantic (+2.5%).

Figure 5 shows the final result with all methods using the basic content + keywords configuration. A trade-off is visible where the semantic model achieves highest precision but lowest diversity.

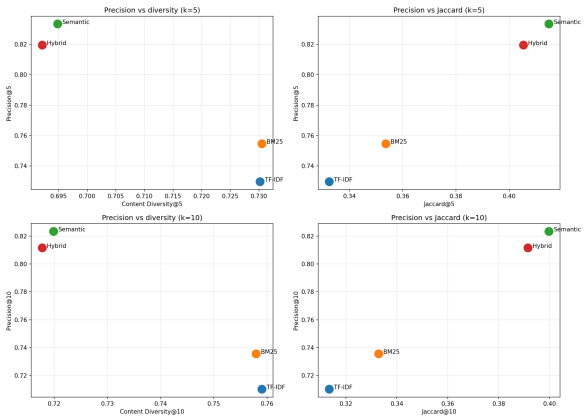

Figure 5: Precision vs diversity trade-off based on title + overview + keywords.

3 Discussion and Conclusion

The class imbalance in the dataset (Figure 1) likely biases the evaluation, as rare genres have fewer relevant candidates and are more prone to false positives, causing overall precision to be dominated by majority classes. Additionally, variation in overview length (Figure 2) may affect retrieval performance, since sparse methods depend on sufficient term frequency, while dense models are generally more robust but still benefit from richer context. Further analysis is needed to quantify these effects.

Across both data configurations (Table 1 and Figure 5), the hybrid approach fails to outperform pure semantic retrieval, indicating that lexical matching introduces noise rather than complementary information.

Figure 5 shows a clear accuracy–diversity trade-off. Semantic models achieve high precision but low diversity, while sparse methods increase variety, possibly through looser keyword matches. The preferred approach depends on the application: precision oriented systems favor semantic retrieval, whereas discovery oriented systems benefit from higher diversity. However, extremely high diversity may indicate weak relevance, making moderate diversity the desirable balance where recommendations are still thematically related to the query while offering variety within that theme.

Genre based evaluation may also advantage semantic models, as sparse methods can produce valid but cross genre recommendations not captured by the metrics.

Adding metadata (Table 2) highlights the vocabulary mismatch problem. Sparse methods benefit substantially from keywords (+9.7%), while semantic models show only minor gains (+2.5%), suggesting much of the context is already encoded in the title + overview embeddings.

In summary, dense retrieval clearly outperforms sparse methods for overview based movie recommendation. Although embedding models often have higher computational cost and resources to produce, the precision gains justify their use when relevance is critical.

Future work should include human evaluation and integrate genre information as input alongside titles, overviews, and keywords. Additionally, clustering methods could be explored to assess whether it is possible to move beyond predefined genres and group movies based on themes inferred from raw text.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep

bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Kaggle, Inc. Tmdb 5000 movie dataset. https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Sch"utze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Kezban Dilek Onal, Ismail Sengor Altingovde, Pinar Karagoz, and Maarten de Rijke. 2016. Getting started with neural models for semantic matching in web search. *Preprint*, arXiv:1611.03305.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Stephen Robertson. 2025. Bm25 and all that – a look back. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 5–8, New York, NY, USA. Association for Computing Machinery.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

Henrique* Schechter Vera, Sahil* Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, Alice Lisak, Min Choi, Lucas Gonzalez, Omar Sanseviero, Glenn Cameron, Ian Ballantyne, Kat Black, Kaifeng Chen, and 69 others. 2025. Embeddinggemma: Powerful and lightweight text representations.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. *Preprint*, arXiv:1904.09675.