Dual News Recommender System using MIND Dataset

Sonia Borsi
University of Trento
Trento, Italy
sonia.borsi@studenti.unitn.it

Filippo Costamagna
University of Trento
Trento, Italy
filippo.costamagna@studenti.unitn.it

Joaquin Lopez Calvo
University of Trento
Trento, Italy
joaquin.lopezcalvo@studenti.unitn.it

***Abstract*—In this project, we develop and evaluate two personalized news recommendation systems using the Microsoft News Dataset (MIND), focusing on Collaborative Filtering and Content-Based Filtering approaches. The Collaborative Filtering model leverages the Alternating Least Squares (ALS) algorithm to handle implicit user feedback, while the Content-Based Filtering system integrates BERT embeddings with the FAISS library for efficient vector similarity searches. A comprehensive exploration of the dataset and its textual features guided the design of these models, addressing challenges such as implicit feedback, scalability, and diversity of recommendations. Both models were implemented in a modular architecture using Apache Spark, MongoDB, and Docker for scalability and efficiency. Evaluation metrics such as RMSE and ranking-based measures demonstrate the efficacy of the proposed systems in capturing user preferences. The findings highlight the strengths of both approaches, with ALS excelling in user-item interaction modeling and FAISS providing robust content-based recommendations. This work provides insights into the implementation, challenges, and trade-offs of these systems for real-world news recommendation applications. More about the code can be found in the GitHub Repository of the project .**

I. Introduction

Recommender Systems (RS) were born in the early 90s as a way to combat the information overload intrinsic to the flow of data on the Internet. RS are meant to filter the vast amount of information online to improve user experience, these applications predict user's responses to options with the goal of offering a new set of personalized items for each user.

There are two main groups in RS [1]:

- Content-Based systems: focused on the features of the recommended items. So, if a user has been interested in a certain type of item, then the system will recommend more similar items to that user. For this, the system has to determine the similarity between items and suggest those with the highest similarity to the ones already liked.

- Collaborative filtering: is based on relationships between users and items. In this case, the similarity of items is determined by the similarity of ratings of those items by the users who have rated both. That is, if User A and User B have had a similar rating history, and User A liked Item X, a Collaborative System would be inclined to recommend Item X to User B.

In this paper, we present a comparison of both types of recommendation systems over a single dataset: Microsoft's MIND. This system leverages both an ALS model for Collaborative Filtering and a FAISS disk-based vector search with BERT embeddings for Content-Based recommendations. Our goal was to compare the results (both from a quantitative as well as a qualitative point of view), and the implementation requirements for each of these systems.

II. Dataset

For our project, we've used the MIND-small dataset, a subset of the Microsoft News Dataset (MIND) [2], which was developed for research on personalized news recommendation systems. MIND-small contains around 300,000 interactions coming from 50,000 users and approximately 65,000 unique news articles. In comparison, the full MIND-large dataset includes about 160,000 English news articles and more than 15 million impression logs generated by 1 million users. Due to hardware restrictions, we opted for the smaller version, which remains sufficient for our project objectives.

A. News Dataset

The news dataset contains detailed information about the news articles, providing metadata for each article. The columns include:

- **News ID:** A unique identifier for each news article, allowing it to be matched with user interactions in the behaviors data.
- **Category:** The broader topic of the news article, which can be used to group and analyze trends.
- **Subcategory:** A more granular topic classification within the category
- **Title:** The headline of the news article.
- **Abstract:** A short summary providing an overview of the article's content.
- **URL:** The web address of the news article.
- **Title Entities and Abstract Entities:** Structured information about named entities mentioned in the title and abstract of the news articles.

B. Behaviors Dataset

The behaviors dataset provides implicit feedback from users' interactions with news articles. The columns include:

- **User ID:** A unique identifier for each user, allowing interactions to be grouped and analyzed by user behavior.

- **Impression ID** A unique identifier for each impression, which represents a single recommendation session where multiple news articles are shown to a user.
- **Impressions:** This column records the news articles shown to a user during a session, along with whether each article was clicked (coded as 1) or not clicked (coded as 0).
- **History:** Historical news click behaviors of a certain user before a specific impression.
- **Timestamps:** These indicate when the interactions occurred, providing a temporal sequence of user behavior. However, analysis of the time gaps between article clicks revealed anomalies in the data. For instance, some users appeared to spend several hours or even days reading a single article before moving to the next one. As a result, while we initially explored weighting interactions by "reading time," the unrealistic patterns in this column made such an approach impractical and we ended up discarding this variable.

One specificity of news recommenders is implicit feedback. While traditional RS relies on users rating items, this type of explicit rating is very rare on news platforms [3]. Consequently, the users' interest in news tends to be inferred from their click behaviors, considering positively those articles clicked by the reader, and negatively those ignored. To deal with this binary nature of the responses, specific algorithms such as ALS should be used.

Through implicit feedback, the MIND dataset better reflects how users typically engage with online content. Implicit feedback, such as clicks, is a more common and natural indicator of user preferences. On the one hand, this makes MIND particularly relevant for modeling real-world recommendation systems on content where explicit ratings are rare, but on the other hand, it closes the door to all the algorithms based on rankings, as well as certain solutions for hybridization that couldn't be implemented due to the nature of the dataset.

C. EDA

Exploratory data analysis was conducted on both the news and behavior datasets to gain a better understanding of their characteristics, in particular the distribution of news categories, subcategories, and user behavior, as well as missing and duplicate data. The news dataset contains 17 unique categories and 264 subcategories. Figure 1 provides a visualization of the distribution of news across categories, with *news* and *sports* being the most common across the dataset. At the subcategory level 2 there is a greater degree of diversity, with the most common subcategories being *newsus*, *footballnfl* and *newspolitics*. The dominance of just a few categories and subcategories in the dataset reflects real-world trends.

The behaviors dataset contains 50,000 unique users with varying levels of interaction, some with very short histories and others with more extensive records of articles clicked. Missing data is present in the datasets, in particular, the abstract columns of the news dataset contain 2,666 missing entries. Some minor missing values are also observed in the

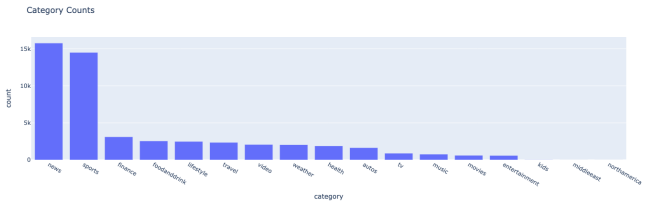


Fig. 1: *News Category Distribution: This figure shows the distribution of articles across different categories in the dataset. The majority of articles belong to the "news" and "sports" categories, with significantly fewer articles in categories such as "finance," "lifestyle," and "entertainment." This imbalance in the dataset highlights the dominance of the major categories and the presence of a long tail of minor categories.*

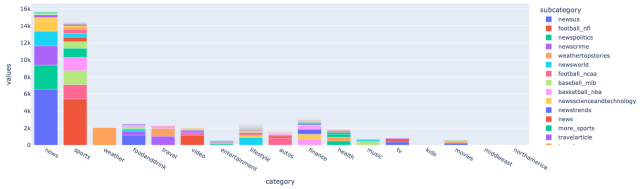


Fig. 2: *News Categories and Subcategories: This figure shows the distribution of articles across main categories, with a breakdown into subcategories. The "news" and "sports" categories dominate the dataset, with subcategories such as "newsus" and "football_nfl" contributing significantly to their totals. Smaller categories like "finance" and "entertainment" are more evenly distributed across their respective subcategories. The visualization highlights the dataset's category imbalance and subcategory diversity.*

title and abstract entities columns, but these represent a small proportion of the data. We observed 848 duplicate article titles and 3,972 duplicate abstracts.

III. Algorithms

A. Collaborative Filtering - ALS

When dealing with implicit feedback in recommender systems, traditional collaborative filtering approaches often fall short because they don't distinguish between missing data and negative feedback. Also, most models are designed for a range of ratings (e.g. 1 to 5), but in the case of implicit feedback, the user's responses are measured in zeros and ones. The Alternating Least Squares (ALS) algorithm addresses this challenge by introducing a specialized optimization framework that accounts for both user preferences and confidence levels [4].

The core idea behind ALS for implicit feedback is to decompose the user-item interaction matrix into two lower-dimensional matrices: user factors (X) and item factors (Y).

Unlike in explicit feedback scenarios, we need to consider all possible user-item pairs, not just the observed ones. The algorithm transforms raw interaction data r_{ui} into two components:

$$p_{ui} = \begin{cases} 1 & \text{if user } u \text{ interacted with item } i \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = 1 + \alpha r_{ui}$$

Here, p_{ui} represents the inferred preference, while c_{ui} indicates our confidence in that preference. The parameter α determines how quickly confidence grows with observed interactions.

The model learns latent factors by minimizing:

$$\min_{X,Y} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(||X||_F^2 + ||Y||_F^2)$$

where λ is a regularization parameter to prevent overfitting.

The optimization proceeds by alternating between solving for X and Y. When updating user factors, we solve:

$$x_u = (Y^T C_u Y + \lambda I)^{-1} Y^T C_u p(u)$$

For item factors:

$$y_i = (X^T C_i X + \lambda I)^{-1} X^T C_i p(i)$$

The algorithm continues alternating between these updates until convergence. This approach is particularly efficient for implicit feedback because it handles the dense nature of implicit data (where all user-item pairs are potential interactions) while maintaining linear scaling with the number of observed interactions.

B. Content-Based Filtering - FAISS

The second algorithm implemented in this project is FAISS for Content-Based Filtering (CBF). CBF focuses on the content of the news data a user has interacted with in the past to recommend new news. Content-based systems attempt to recommend articles similar to those that a given user has liked in the past [5]. Content-based recommendation systems include a content analyzer that extracts important features from the text of news items, a profile learner that creates a user profile for each user based on the features of the news items they have interacted with in the past (history), and a filtering component that matches the user profile to candidate news items using a similarity metric. In this project, cosine similarity:

$$\text{cosine similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Specifically, Facebook AI Similarity Search (FAISS) was the algorithm used to perform CBF of news articles. FAISS is an open-source library designed to perform vector similarity searches in a fast and scalable way. We chose this library because it represents one of the latest advancements in the field, with its foundational research [6] being published as recently as 2024. The library implements an approximate nearest neighbors (ANN) search, which is a much less computationally intensive version of the exact nearest neighbor search. Instead of searching the entire dataset as a standard nearest neighbor search does, ANN limits the search to a subset of vectors that are most likely to contain the nearest neighbors, making it really effective in high-dimensional spaces. In particular, the key functionality of FAISS is that it works by indexing the vectors to enable fast similarity searches by creating an Inverted File Index (IVF). This clustering-based approach involves three steps:

- **Clustering**: during the indexing phase, the vector space is divided into clusters, and each vector (news) is assigned to a cluster.
- **Querying**: during the retrieval phase, the user's profile embedding is compared to the centroids of the clusters, so that only the most relevant clusters are searched for the nearest vectors.
- **Nearest Neighbor Search**: within the selected clusters, FAISS applies cosine similarity to retrieve the most relevant messages for the selected use.

IV. Implementation

A. Pipeline Architecture

Our system was implemented in Python, leveraging Apache Spark through the PySpark API where applicable. It uses MongoDB for data storage, Power BI for visualization, and Docker for containerization. The architecture is divided into six separate containers, as in Figure 3, each responsible for a specific service:

- **data-fetch**: Handles dataset retrieval, basic processing, and insertion into the database.
- **als**: Implements the Alternating Least Squares CF-Based Recommender System.
- **cbrs**: Generates embeddings for the news articles and performs the vector search for recommendations using FAISS
- **mongodb**: Stores dataset, embeddings and final recommendations.
- **clustering**: Performs clustering after reducing dimensionality through PCA.
- **mongoexpress**: Provides a web-based interface to interact with the MongoDB database.

The following sections will describe how the main components work.

B. Data fetching

The **data-fetch** container automates the process of downloading, extracting, and loading the MIND dataset into MongoDB for use in the system. The pipeline is designed to handle both training and validation datasets. Below are the key steps and features of the pipeline:

- **Configuration and Initialization**:

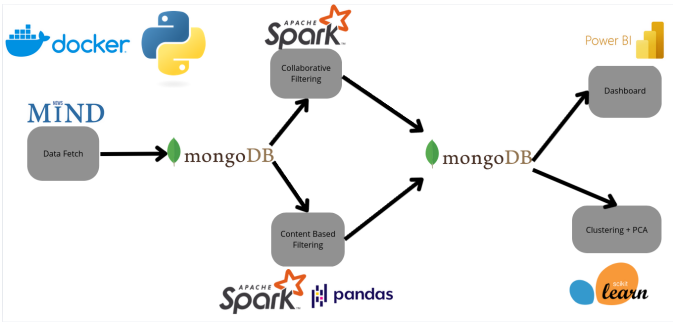


Fig. 3: *Overview of the data pipeline integrating the MIND dataset with MongoDB, Spark, and Python for collaborative and content-based filtering, clustering, and PCA analysis. Results are visualized in Power BI for actionable insights.*

  - Uses a configuration file (`config.yaml`) to determine the dataset size (`demo`, `small`, or `large`).
  - Sets up MongoDB connection details and initializes collections for storing training and validation data.
- **Dataset Management**:
  - Downloads the MIND dataset if not already available locally.
  - Extracts the downloaded dataset and identifies TSV files for user behaviors and news articles.
- **Data Loading into MongoDB**:
  - Reads TSV files into Pandas DataFrames, using predefined headers for behaviors and news data.
  - Converts the DataFrames into dictionaries and inserts them into MongoDB collections.
  - Implements checks to skip loading data into MongoDB if the collections already contain sufficient records.
  - Provides optional functionality to clear existing data in collections before reloading.

This container streamlines the preparation of the MIND dataset, ensuring that data is readily available in MongoDB for downstream tasks like embedding generation and recommendation system evaluation. The data model utilized in the database remains the same used in the original dataset, with the addition in each collection of a unique id for each record ("_id").

C. Collaborative Filtering

The **als** container implements the training and recommendation pipeline for an Alternating Least Squares (ALS) model, designed for a news recommendation system. It processes user interaction data, trains the ALS model, and generates personalized recommendations. Below are the key steps and features of the pipeline:

- **Configuration and Initialization**:
  - Loads configuration settings from `config.yaml`, including ALS hyperparameters and database connection details.
  - Configures a Spark session to handle distributed data processing and MongoDB interactions.
- **Data Loading and Preprocessing**:
  - Fetches user interaction data from MongoDB collections (`behaviors_train` and `behaviors_valid`).
  - Preprocesses the data to prepare it for ALS training, including negative sampling using a configurable `npratio`.
- **ALS Model Training**:
  - Creates an ALS model with the specified rank, regularization, and maximum iterations.
  - Iteratively trains the ALS model on the training data, evaluating its performance on the validation data at each step.
  - Logs metrics such as Root Mean Squared Error (RMSE) to both the console and Weights and Biases (WandB) for tracking.
- **Recommendations Generation**:
  - Uses the trained ALS model to generate top-N recommendations for all users.
  - Explodes and formats the recommendations for further analysis or storage, including user IDs, recommended news IDs, and their predicted ratings.
- **Results Storage**:
  - Saves the trained ALS model to a specified path for future use.
  - Writes the generated recommendations back to MongoDB in a dedicated collection (`recommendations_als`) for downstream tasks.

To ensure optimal performance, we conducted hyperparameter tuning using Spark's built-in cross-validation functionality in the `ALS_hyperparam_optimization.ipynb` notebook. The hyperparameter search space included the rank of the latent factors, the regularization parameter, and the alpha parameter controlling confidence scaling. The optimization process employed a 3-fold cross-validation strategy, evaluating models using Root Mean Square Error (RMSE) as the performance metric. The hyperparameters in the final implementation were set according to the results of this cross-validated search.

D. Content based filtering

The **cbrs** container implements a complete end-to-end pipeline starting from the news articles and the user behaviors. It integrates multiple modules for data management, embedding generation, recommendation computation, and evaluation. The key components of the pipeline are:

- **Data Loading and processing**:
  - Retrieves user behaviors and news articles from MongoDB

  - Ensures data consistency by preprocessing, removing duplicates, and filtering already-processed entries.
  - Normalizes text fields for clean input to the embedding generation process.
- **Embedding Generation**:
  - Leverages Spark NLP library with BERT embeddings to transform news articles into dense numerical representations.
  - Processes articles in batches using Spark for scalability and stores the embeddings back into MongoDB.
- **Recommendation Computation**:
  - Builds user profiles by aggregating news embeddings based on user interaction histories.
  - Constructs a FAISS index for efficient similarity-based search, utilizing cosine similarity.
  - Computes personalized recommendations incrementally for efficient processing and stores results in MongoDB.
- **Evaluation**:
  - Retrieves ground truth interactions from MongoDB and compares them with generated recommendations.
  - Calculates performance metrics such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) to assess accuracy.

The main module executed on container startup is `run_cbrs_pandas.py`, the embedding and text-processing part (implemented in Spark) are implemented in `clean_embed.py` while all the functions necessary for the vector search are contained in the `cbrs_utils_pandas.py`

E. Clustering

The **clustering** container implements a complete pipeline for analyzing news embeddings using Principal Component Analysis (PCA), K-Means clustering, and visualization techniques. The primary goal is to process, cluster, and analyze news embeddings while storing results in MongoDB and generating insights into category distributions across clusters. Below are the key steps and features of the pipeline:

- **Data Loading and Preprocessing**:
  - Connects to MongoDB to fetch news embeddings and associated metadata from collections.
  - De-duplicates news articles by their unique `news_id`.
  - Maps `news_id` to their corresponding categories and parses embedding strings into numerical arrays.
- **Dimensionality Reduction**:
  - Applies PCA to reduce the dimensionality of news embeddings, ensuring they are optimized for clustering and visualization.
  - Configurable number of principal components allows flexibility in analysis.
- **Clustering with K-Means**:
  - Performs K-Means clustering on the PCA-reduced embeddings to identify groups of similar news articles.
  - Configurable number of clusters enables adaptation to different datasets and use cases.
- **Visualization**:
  - Generates cluster visualizations using both PCA for dimensionality reduction.
  - Saves plots of clustered data for easy interpretation and insights.
- **Category Distribution Analysis**:
  - Analyzes the distribution of news categories within each cluster.
  - Outputs the results as CSV files, stacked bar charts, and heatmaps for comprehensive visualization.

F. Evaluation

The evaluation of our models was conducted using both regression and classification based to assess its performance. Two primary Python scripts were implemented: one focused on model training and regression metric evaluation, and the other on ranking-based evaluations. While this one was implemented, the final version of the models was only evaluated using RMSE to maintain simplicity. To evaluate regression performance, we computed the RMSE, which is defined as:

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

where y_i represents the actual value, and $\hat{y}_i$ represents the predicted value. RMSE quantifies the average deviation between predicted and true values, with higher penalties for larger errors. Submissions for the Netflix Prize, for example, were judged using the RMSE computed on the test dataset's undisclosed "true" values [7].

The ranking-based evaluation focused on metrics such as Precision@K, Recall@K, Normalized Discounted Cumulative Gain (NDCG@K), and Mean Average Precision (MAP). These metrics gauge the model's ability to rank items effectively. Predictions were grouped by user sessions, with the top-K ranked items evaluated against the ground truth.

The evaluation pipeline also incorporated iterative logging via WandB to monitor RMSE across training epochs. Although ranking-based metrics were computed, model selection was performed through RMSE, which served as the benchmark for model improvement during training. The final model was saved upon achieving optimal RMSE on the validation dataset.

V. Experiments

A. Clustering

In the initial stages of the project, we performed **K-means** clustering to validate the quality of the news embeddings generated by the BERT model, with the ultimate goal of ensuring that they capture meaningful features of the data before proceeding with the content-based filtering recommendation

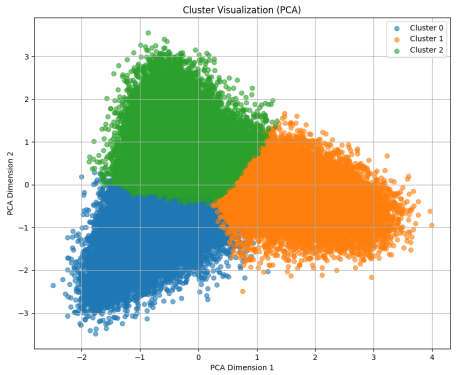


Fig. 4: *Cluster Visualization (PCA): The figure shows the two-dimensional representation of the news embeddings, reduced using PCA. The three distinct clusters represent the K-means clustering results with $k = 3$. Cluster 0 (blue) primarily corresponds to the news category, Cluster 1 (orange) to the sports category, and Cluster 2 (green) to the minor categories such as finance, lifestyle, and entertainment. The separation of clusters demonstrates that the embeddings capture meaningful semantic differences between articles.*

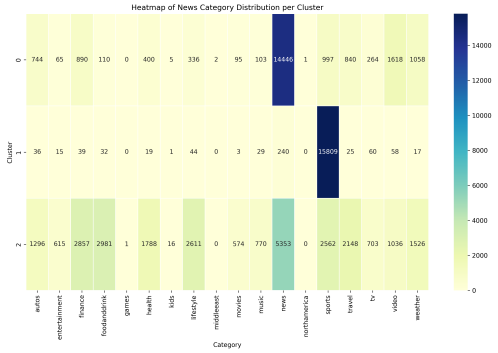


Fig. 5: *Heatmap of News Category Distribution per Cluster: This heatmap illustrates the distribution of news categories across the three clusters identified by K-means. Cluster 0 predominantly contains news articles, Cluster 1 is primarily sports articles, and Cluster 2 aggregates articles from smaller categories such as finance, lifestyle, and entertainment. The color intensity reflects the count of articles in each category, highlighting the alignment between clusters and dataset categories.*

system since it heavily relies on the embeddings to represent semantic relationships between news items. Given the results of the EDA 1, and given that the dataset is dominated by two large categories (news and sports) with a long tail of smaller categories, we chose $k = 3$ in our k-means algorithm to reflect the structure of the dataset. In addition, to reduce computational complexity and visualize the resulting clusters, we used **Principal Component Analysis (PCA)** to project the high-dimensional embeddings into a 2D space.

Figure 4 shows the final cluster visualization after performing PCA in the reduced 2D space. The plot shows three distinct clusters, with each dot representing a message. The clear and well-defined cluster boundaries confirm that the embeddings capture semantic relationships between articles.

The heatmap in Figure 5 illustrates the distribution of categories within each cluster: cluster 0 contains mainly news articles, cluster 1 contains mainly sports articles, while cluster 2 aggregates articles from the other smaller categories.

This clustering analysis acted as a sanity check on the quality of the embeddings; indeed, the results we obtained confirmed that the embeddings successfully captured the dominant themes in the dataset. This validation ensured that the embeddings were suitable for efficient and meaningful nearest-neighbor search during the FAISS-based recommendation process.

B. FAISS vector search with Spark

Vector search with disk-based indices, such as the DiskANN viewed in lecture, is highly valuable for large-scale similarity search, as it enables efficient querying of datasets that exceed in-memory limits. While we succeeded in integrating Spark with FAISS, the performance is significantly slower compared to using FAISS with Pandas. This disparity arises because Spark's distributed architecture introduces additional overhead in coordinating tasks, partitioning data, and transferring information between nodes, which can bottleneck the low-latency nature of FAISS operations. In contrast, Pandas operates in a single-node, in-memory environment, allowing FAISS to perform vector computations directly on contiguous data structures without the need for distributed communication. The result is a much faster and more efficient query process with Pandas, despite the integration with Spark providing broader scalability and distributed processing capabilities. It is also worth noting that, as FAISS is a very recent library (2024), its integration with Spark may still face challenges due to its cutting-edge nature. This could contribute to the suboptimal performance observed in a distributed Spark environment.

C. Hybridization

In the last years, hybrid RS have been growing in popularity. The motivation for a hybrid Recommendation System approach that combines Collaborative Filtering and Content-Based Filtering methods arises to cover the inherent shortcomings of each method. On the one hand, CF is vulnerable to the "cold start" problem. The cold start issue arises because a new user or item does not have a rating history within the system. Given the lack of personalized data for reference, it is difficult for CF systems to predict user interest in items or users with little to no available data [8]. On the other hand, recommendations coming from CBF systems tend to be limited in diversity. This lack of personalization affects systems' efficiency as the recommendations do not properly address user heterogeneity. A hybrid approach is expected

to avoid the disadvantages of the existing approaches while combining the advantages of them.

This is why initially we thought about implementing a hybrid RS. Initially, we thought about a switching architecture, with the CF as the main backbone of the whole system, and the CBF covering the cold-start problem providing recommendations with as little as one article read by the user. However, the reading of review literature revealed that switching systems are not very popular nowadays [9]. Instead, in the last years, the trend has followed the weighted approach [10]. In this setting, predictions are combined linearly following some formula similar to this one:

$$p_{u,i} = \alpha_u \cdot p_{u,i}^{CF} + (1 - \alpha_u) \cdot p_{u,i}^{CBF}$$

Where:

- $p_{u,i}^{CF}$: Predicted rating from Collaborative Filtering.
- $p_{u,i}^{CBF}$: Predicted rating from Content-Based Filtering.
- α_u: Weight factor that adjusts the influence of each method dynamically.

The issue we encountered while implementing this solution was, once again: implicit feedback! All these solutions rely on one way or another on ratings for their formulas. So, what happens when feedback is implicit? Honestly, we couldn't find much literature to inspire a solution. While we considered the possibility of creating one formula by ourselves, we also realized this was way out of line with respect to the course's content, so we preferred to focus our efforts on applying techniques covered during the classes.

D. Recommending brand new data

Since we wanted to create an end-to-end working product, during the first weeks we explored the possibility of recommending news unseen by our models to a hypothetical set of users. So we contacted the NewsCatcher API [11] to request an API quota to retrieve news articles for our project. Not only did we receive a generous API quota, but also we implemented the initial steps of a pipeline for fetching and cleaning the NewsCatcher data (which can be found in the `experiments/newsapi` folder).

So, why did we abandon this path? First of all, after deepening our research, we realized that training a recommendation model and putting it in production are two radically different tasks that require a radically different set of tools. Second, while the content vectors that we're calculating for the CBF recommender would have been useful to compare with new content vectors from new content, the CF part would have been much more difficult to implement requiring at least the creation of a synthetic set of users, and a synthetic set of interactions between synthetic users and NewsCatcher data. But the fact that closed the door for us was that even after all those implementations of synthetic user-item interaction data, the synthetic aspect of the pipeline probably would have retrieved senseless results (as a "progressive Roman 27 years old girl living in Pigneto and studying art history" reading articles on "Bollywood artists" or "F1 results"). For all this, we decided to prioritize the training of the models, which was closely aligned with the course's content.

VI. Results

The implementation of the two recommendation systems provided insights into the trade-offs between the two approaches, particularly in terms of performance, scalability, and suitability for different datasets and infrastructures.

Metric	Collaborative Filtering	Content-Based Filtering
RMSE	0.6260	0.835177

TABLE I: Comparison of RMSE between Collaborative Filtering and Content-Based Filtering.

In quantitative terms, ALS outperformed FAISS in RMSE I by a margin of around 25% measured by RMSE. However, we acknowledge that in an industrial setting both only ranking metrics (e.g. Precision@K, Recall@K, NDCG@K)) should be considered as well as business-centric metrics (in the case of news, CTR or Conversion Rate should also be considered.)

Looking at the results from a qualitative perspective over the dashboard, we did notice how CF systems retrieve much more heterogeneous recommendations. This can be seen in the donut chart counting the recommendations by categories. While the CBF recommendations aligned closely with the predominant categories (so, if "news" was the main section in the EDA, was also predominantly recommended by the CBF model), the CF did follow the book's axioms recommending much more personalized news. In our case, this increased heterogeneity led to a reduction in RMSE.

ALS was implemented in Spark, as its architecture handles large matrix factorization by distributing the computation across different nodes. On the other hand, FAISS was implemented in Pandas, so it worked on a single-node environment. Initially, we tried to implement it in Spark, but it suffered performance penalties due to Spark's overhead and proved to be less efficient for low-latency operations compared to Pandas.

Collaborative Filtering uses matrix factorization, an approach that excels at identifying relationships in large datasets by exploiting user-item interaction patterns. It can recommend a wide range of items, even those with no direct feature similarity, by finding patterns in collaborative data. However, it requires historical interaction data to make recommendations, making it ineffective for new users with no historical behavior. It also struggles with sparse interaction matrices, where many users have limited interactions, resulting in poor recommendations. Finally, it may be less suitable for rapidly changing content domains (e.g. news) as it relies on static user-item interaction matrices.

Moreover, **Content-Based Filtering** finds nearest neighbors in high-dimensional embeddings based on item features, making it suitable even in situations where we have no interaction data or data about other users' activity, as it does not rely on historical user interaction data. It is also easily adaptable to domains with frequently changing content, such as news.

However, it requires high-quality embedding to work properly and often recommends items similar to those the user has already interacted with, leading to a 'filter bubble' effect and limited diversity of suggestions.

VII. Conclusions

This project presented a detailed comparison between Collaborative Filtering (CF) using the Alternating Least Squares (ALS) algorithm and Content-Based Filtering (CBF) using FAISS with BERT embeddings. Both approaches demonstrated their respective strengths and challenges when applied to the MIND dataset, providing valuable insights into their practical applications.

ALS-based collaborative filtering demonstrated its strength in modeling user-item interaction data through matrix factorization. Its scalability, facilitated by Spark's distributed architecture, allowed it to handle the MIND-small dataset. However, its reliance on historical interactions does represent a limit in its ability to address the cold start problem, which was dropped in our implementation. Our conclusion of this model is that, while it remains a useful "backbone" for RS, will face issues adapting to the news dynamic content domain.

On the other hand, FAISS-based content-based filtering excelled in using embeddings to perform efficient vector similarity searches. The use of BERT embeddings enabled the system to capture the semantic relationships within news articles, making it highly effective in content-rich domains and capable of handling cold-start scenarios. The FAISS implementation with Pandas provided faster query times in a single-node environment, but integrating FAISS with Spark revealed performance bottlenecks, highlighting the trade-off between speed and scalability.

Ideally, a future implementation would try to convert the current FAISS implementation to a Spark distributed framework. Since the CB filtering does address effectively the biggest Achilles heel of the CF model, the cold-start problem, we wonder whether a hybridization would have increased our performance, as suggested by the literature and last year's common industry practices.

References

[1] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2nd ed., 2011.

[2] C. Wu, F. Wu, H. Ge, T. Qi, Y. Huang, X. Xie, M. Ding, W. Wei, and M. Zhou, "MIND: A large-scale dataset for news recommendation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 3597–3606, 2020.

[3] I. Ilievski and S. Roy, "Personalized news recommendation based on implicit feedback," in *Proceedings of the 2013 International News Recommender Systems Workshop and Challenge*, pp. 10–15, ACM, 2013.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *2008 Eighth IEEE International Conference on Data Mining*, pp. 263–272, IEEE, 2008.

[5] P. Lops, M. de Gemmis, and G. Semeraro, *Content-based Recommender Systems: State of the Art and Trends*, pp. 73–105. Boston, MA: Springer US, 2011.

[6] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2024.

[7] Netflix, "Netflix prize rules," 2009. Accessed: 2024-12-27.

[8] N. Ifada and M. Ummamah, Kautsar, "Hybrid popularity model for solving cold-start problem in recommendation system," in *Proceedings of the 5th International Conference on Sustainable Information Engineering and Technology*, pp. 40–44, Association for Computing Machinery, 2020.

[9] V. G. M. Murillo, D. E. P. Avendaño, F. R. López, and J. M. G. Calleros, "A systematic literature review on the hybrid approaches for recommender systems," *Computación y Sistemas*, vol. 26, no. 1, pp. 357–372, 2022.

[10] H.-Q. Do, T.-H. Le, and B. Yoon, "Dynamic weighted hybrid recommender systems," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, pp. 644–650, 2020.

[11] "Newscatcher api." https://www.newscatcherapi.com/. Accessed: 2025-01-06.

APPENDIX

APPENDIX A - DATA SCHEMA DEFINITIONS

The following tables outline the variables of interest for this project.

Field Name	Data Type	Example
News ID	string	N37378
Category	string	sports
Subcategory	string	golf
Title	string	PGA Tour winners
Abstract	string	A gallery of recent winners on the PGA Tour.
URL	string	https://www.msn.com/en-us/sports/golf/pga-tour-winners/ss-AAjnQjj?ocid=chopendata
Title Entities	list¡dict¿	{“Label”: “PGA Tour”, “Type”: “O”, “WikidataId”: “Q910409”}
Abstract Entities	list¡dict¿	{“Label”: “PGA Tour”, “Type”: “O”, “WikidataId”: “Q910409”}

TABLE II: Example of a news entry from the `news` collection.

Field Name	Data Type	Example
Impression ID	string	123
User ID	string	U131
Time	datetime	11/13/2019 8:36:57 AM
History	list¡string¿	N11, N21, N103
Impressions	list¡string¿	N4-1, N34-1, N156-0, N207-0, N198-0

TABLE III: Example of a behaviors entry from the `behaviors` collection.

Field Name	Data Type	Example
_id	ObjectId	ObjectId(’675e99fba31fda796ab1743a’)
news_id	string	N1
embedding_string	string	-0.1343533, 0.10080873, 0.44086725, ..., -0.37292448, 0.07811815

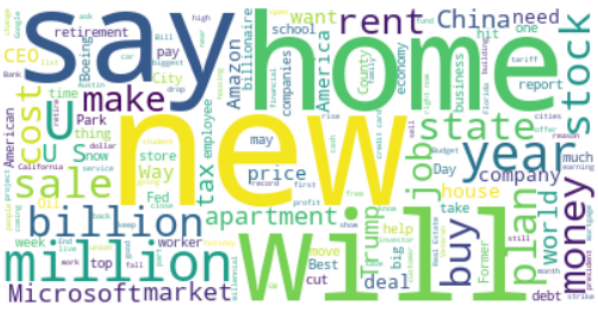
TABLE IV: Example of an entry in the `news_embeddings` collection.

Field Name	Data Type	Example
_id	ObjectId	ObjectId(’677187e8e4f2d6b817f2e2d8’)
userId	string	U100
recommendations	list¡dict¿	[{newsId: ’N36940’, rating: 0.8471}, {newsId: ’N56931’, rating: 0.8465}, {newsId: ’N12690’, rating: 0.8427}, ...]
newsId	string	N36940
rating	float	0.8471711874008179

TABLE V: Example of how recommendations are stored for both content-based and collaborative filtering recommendation systems.

APPENDIX B - EDA

In the initial exploration phase of our project, we conducted a detailed analysis of the textual data to gain insights into the structure, content, and patterns present in the dataset. This analysis helped guide the design and implementation of the two recommendation systems by identifying key features and potential challenges in the data. By examining aspects such as title length, sentiment polarity, and category-specific trends, we were able to better understand the characteristics of the news articles and user interactions. The following plots summarize the results of this exploratory data analysis.

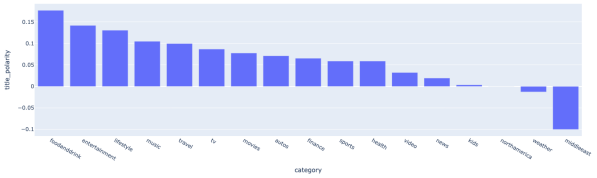


(a) Finance Word Cloud

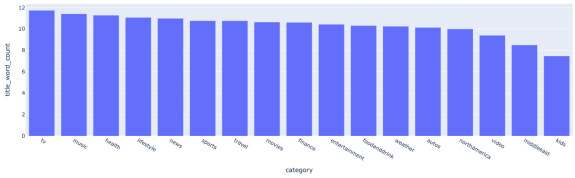
(b) News Word Cloud

(c) Sport Word Cloud

Fig. 6: Word clouds generated from the dataset. (a) represents frequent terms in the Finance category, highlighting business and economic topics, (b) shows frequent terms in the News category, reflecting general news trends and current events, and (c) illustrates frequent terms in the Sports category, emphasizing terms related to games, teams, and victories.



(a) Average polarity of titles across categories



(b) Average word count per category

Fig. 7: (a) **Average Title Polarity per Category:** This plot shows the average polarity of titles for each category. Positive polarity values indicate a generally positive sentiment, while negative values suggest negativity. Categories such as ”food and drink,” ”entertainment,” and ”lifestyle” exhibit higher positive sentiment in their titles, while categories like ”middle east” and ”weather” lean toward neutral or slightly negative sentiment. These insights reflect the emotional tone prevalent in different types of news content. (b) **Average Word Count per Category:** This plot displays the average number of words in titles for each category. Categories like ”TV” and ”music” have the longest titles, with an average word count exceeding 11 words, while ”kids” and ”middle east” have the shortest titles. This variance in word count could indicate differences in writing styles or the complexity of information conveyed within each category.

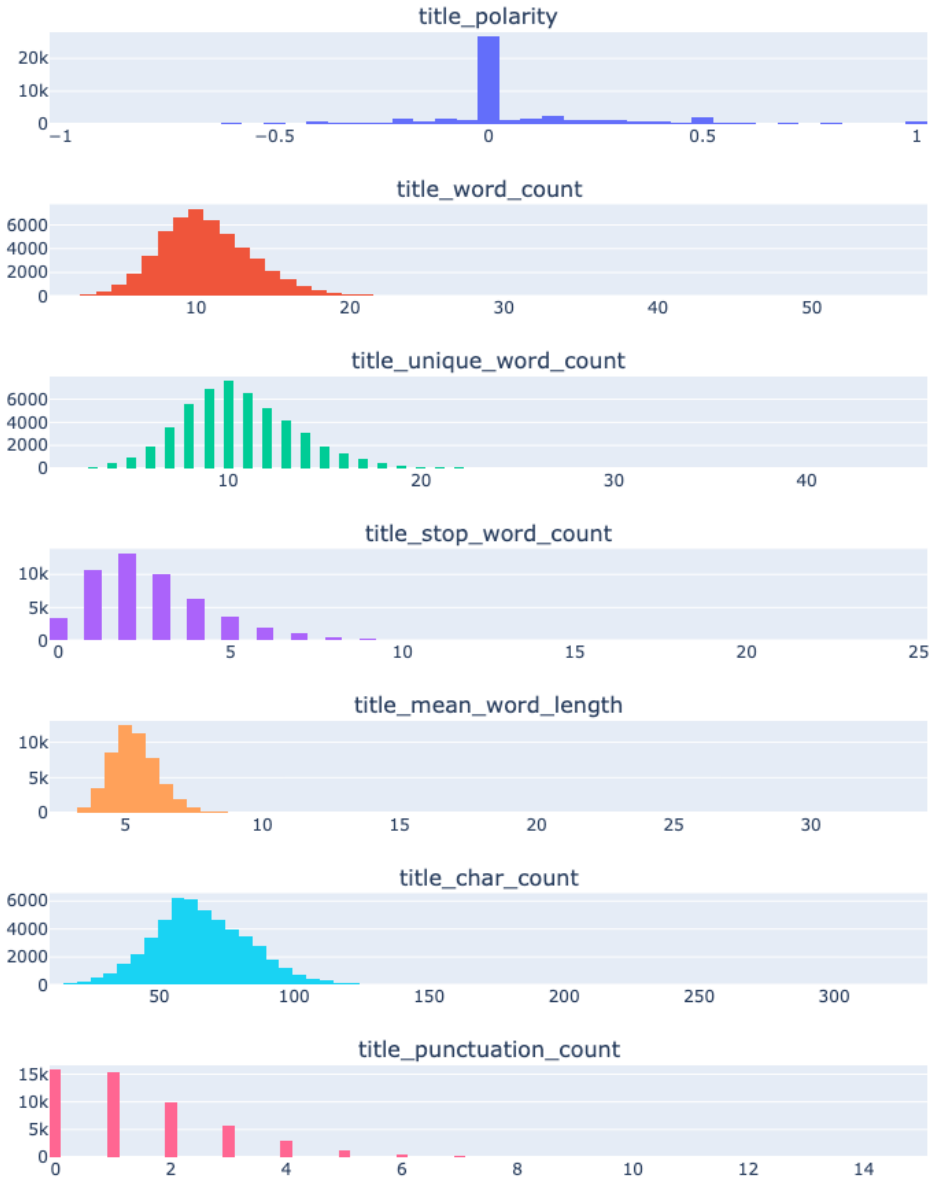


Fig. 8: Distribution of key title features across the dataset. The subplots present various aspects of news titles: (1) **Title Polarity:** The polarity score distribution highlights that most titles are neutral (polarity near 0), with fewer titles exhibiting strongly positive or negative sentiments. (2) **Title Word Count:** This histogram shows that the majority of news titles contain 10 to 20 words, with very few titles exceeding 50 words. (3) **Title Unique Word Count:** This distribution indicates the number of unique words in titles, with most titles having 10 to 20 unique words, aligning with the overall word count trends. (4) **Title Stop Word Count:** The stop word count reveals that a significant portion of titles use 5 to 15 stop words, showing the prevalence of common language constructs in news headlines. (5) **Title Mean Word Length:** This histogram shows that most words in titles have an average length of 5 to 8 characters, suggesting concise phrasing. (6) **Title Character Count:** The character count distribution aligns with the word count, showing that most titles contain 50 to 150 characters. (7) **Title Punctuation Count:** The punctuation count distribution reveals that most titles include 0 to 4 punctuation marks, indicating minimal but essential usage for structure and clarity. Overall, these visualizations provide insights into the linguistic structure of the dataset's news titles.