Development Track: News Recommendation System

Rachanon Petchoo
petchoo2@illinois.edu
University of Illinois Urbana-Champaign

Mayank Umesh Shetty
mshetty4@illinois.edu
University of Illinois Urbana-Champaign

Shuning Zhang
sz31@illinois.edu
University of Illinois Urbana-Champaign

Yushi Li
yushili2@illinois.edu
University of Illinois Urbana-Champaign

Abstract

We present a web-based personalized news recommendation system designed to deliver relevant and timely articles tailored to individual user interests. Our system addresses the challenge of recommendations by analyzing user interaction patterns and leveraging topic modeling to represent article content. News data is fetched from external APIs and processed using the BERTopic model to extract the content's coverage in each of the topics (as a probability) [1]. User profiles are dynamically updated based on reading behaviors and are matched with articles via similarity scoring. The backend is built with Next.js and connected to a MySQL Database on AWS Relational Database Server (RDS), while the frontend offers an interactive interface for both personalized and trending news. Evaluation plans include offline testing across several condition-based scenarios. Our approach aims to balance personalized relevance with content diversity, ensuring users receive both interest-aligned and globally trending news in an intuitive, responsive interface.

Keywords: News Recommendation, BERTopic, Topic Modeling, User Profiling, React, Next.js

ACM Reference Format:
Rachanon Petchoo, Mayank Umesh Shetty, Shuning Zhang, and Yushi Li. 2018. Development Track: News Recommendation System. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

1.1 Motivation

Users often face frustration when navigating between multiple platforms just to find news relevant to their interests. Irrelevant content, excessive advertisements, and a lack of personalization make news consumption inefficient and overwhelming. Our motivation is to solve this problem by building a lightweight, user-friendly news recommendation system that curates content based on user interests and reading behavior, with the added benefit of showing trending topics outside their usual recommendations.

1.2 Target Users

Our target users are individuals who seek personalized, up-to-date news without the clutter. They range from tech enthusiasts and business professionals to politically engaged readers or anyone with specific content interests. This includes both, casual browsers and heavy readers who want to optimize their reading time.

1.3 Project Objectives

The primary goal is to develop a web-based news recommendation system. The system will:

- Continuously fetch and store news articles from NewsAPI.
- Extract the main topics of our news collection with BERTopic, and refine its interpretability with OpenAI's GPT.
- Represent articles using BERTopic-based embeddings.
- Authenticate users and record their activity.
- Build user profiles from users' interactions with the articles.
- Recommend personalized news based on user profiles.
- Include a simple, usable front-end to display news and capture feedback.

2 System Overview

Our system follows a modular three-tier architecture consisting of a web-based frontend, a RESTful backend service, and a relational database. The frontend, built with React and Next.js, provides an interactive interface for users to view news and receive recommendations. The backend handles authentication, news fetching mechanism, user interaction

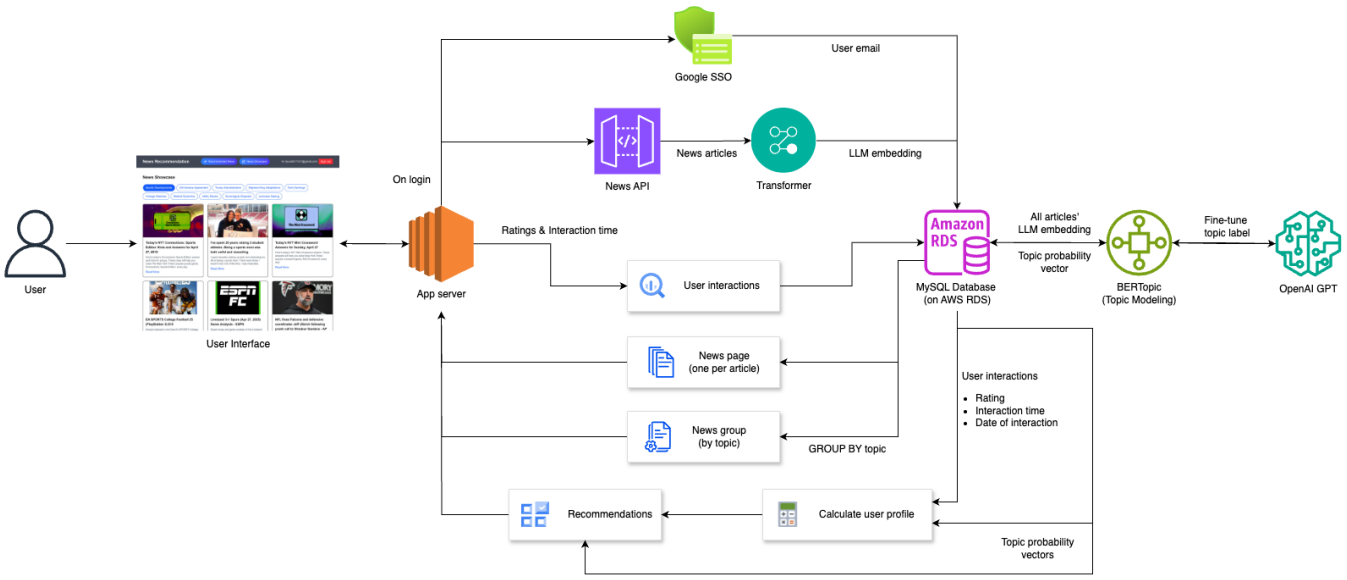

Figure 1. Software Architecture

logging, recommendation logic, and data flow coordination. MySQL serves as the persistent data layer, storing user profiles, article metadata, topic embeddings, and interaction history.

2.1 Core Components

The major components, as shown above, include: (1) the user authentication module that allow us to recognize the user and store their interactions along with building their interest profile (2) a news ingestion module that fetches articles from News API; (3) a topic modeling pipeline using BERTopic to generate article embeddings and GPT to refine interpretability of topic label; (4) a user profiling module that tracks user interactions and updates user profiles; (5) a recommendation engine that matches user profiles with articles; and (6) a news serving module that let the user visit each news article in-depth along with seeing the categorized news. These components are connected via RESTful APIs, and the backend exposes endpoints for frontend interaction.

2.2 Data Flow and Use Cases

When a user logs in, their session is initialized and their user id is saved to the database, for future references when building their profile. The system then fetches the latest news articles in the background, processes them into embeddings of probabilistic topic distributions, and stores the results. Along with this, the topic label is sent to refined with OpenAI's GPT for better human interpretability. Upon user activity, interaction data (including the explicit ratings that the user give, along with its implicit interest like time spent on each article) is recorded to the database. Once the user visit the recommendations page, their user profile is updated in real-time with their interaction histories, then recommended news are generated by comparing the user profile vector with article embeddings with cosine similarity, and results are served through the frontend. For users who are not logged in, the recommendations page will provide a message for them to login. Still, any users, with or without login credentials, can utilize the News Showcase page that pull all the articles from database and categorize them into topics, allowing users to browse the news easily by category.

2.3 Deployment and Runtime Environment

The backend of the system is developed using `Next.js`, a full-stack React framework, and connects to a MySQL database hosted on Amazon RDS for persistent data storage. The environment configuration is managed using environment variables stored in a `.env` file to separate code from sensitive credentials and deployment settings.

The `.env` file includes keys for database access (`DB_HOST`, `DB_USER`, `DB_PASS`, `DB_SCHEMA`), authentication credentials for third-party services (e.g., Google SSO via `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET`), and API keys for integrating with external services such as NewsAPI and OpenAI.

To run the system, we must setup the credentials in a local `.env` file (full instructions provided in later sections).

3 Implementation Details

This section outlines the technical implementation details of our final web server product. At a high level, our project can be divided into the following key components:

- Backend and Frontend Design
- User Data Collection and Profile Construction
- New Article Fetching and Processing

- Recommendation Logic

In the following sections, we will provide a detailed explanation of how each component was developed.

3.1 Backend and Frontend Design

To power our web application, we chose the Next.js framework, a full-stack open-source React framework, because it offers several advantages, including server-side rendering. Moreover, Next.js allows us to organize both the frontend and backend logic within the same project structure, streamlining development and reducing context switching. Additionally, one of our team members had prior experience with this framework, which made it easier for us to get started quickly.

The backend of our application is responsible for handling core logic, data processing, and communication with both the frontend and external services.To support the features of our application, we developed several custom API endpoints, including:

- **auth**: Handles user authentication and create a user profile in our database, so we can match their profiles with their interactions and create personalized recommendations later.
- **addUserInteraction**: Use to store users' interactions with each news article, including the stars rating that users explicitly give and the time that they spent on each article (which our recommendation algorithm treats as implicit feedback), to the database.
- **fetchNews**: Retrieves news articles using the NewsAPI and processes them through the BERTopic model to generate document embeddings as the probability of being assigned to each topic. Then, we pass the "topic label" generated by BERTopic to OpenAI's GPT to refine its interpretability as well.
- **updateUserProfile & recommendations**: Updates the user profile based on interaction data (the alogirthm itself will be explained in a later section) and, using our recommendation logic, returns a personalized set of news articles based on cosine similarity.
- **groupedNews**: Get news articles from the database, and organize them by topic, to be shown in the News Showcase page. This allows user to browse all the news articles by category.
- **news/[id]**: Get a single news articles with the specified id. This is used in the News Display page to provide article's title, image, short description, and also render the publisher's website on our website natively as well.

On the frontend, we have the main interface features two primary sections for news presentation, and one for login:

- **Google SSO**: We implemented a user-friendly sign-in interface that allows users to authenticate via Google and generate a unique user ID in our database. This ID is used to track user interactions with news content, enabling us to update their profile and tailor future recommendations based on their interests.
- **News Showcase**: This section offers a broader selection of the latest news articles, categorized by topic. It is designed to provide users with a more general overview and encourage exploration beyond their usual interests.
- **Recommended News**: This section displays personalized news articles tailored to each user based on their interaction history. It reflects the topics they have shown interest in and evolves as the user continues to engage with the platform.

This dual-section approach balances personalization with discovery, enhancing user engagement and information diversity.

3.2 User Data Collection and Profile Generation

After each user logs in for the first time, we will save the user's authentication information in our database and assign a unique userID, which will act as the user identifier in our project.

As shown in Figure 2, the User Interaction table stores key interaction data for each user. Each row records the time a user spends on a given news article (news_id) and, optionally, a user-provided rating indicating their level of interest in the article's content. To update and manage this information efficiently, we utilized SQL queries along with built-in date and time functions to track and process user interactions accurately.

```sql
CREATE TABLE user_interactions (
    interaction_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT NOT NULL,
    news_id INT NOT NULL,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    time_spent_seconds INT NOT NULL,
    viewed_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    FOREIGN KEY (news_id) REFERENCES news_articles(news_id) ON DELETE CASCADE
);
```

Figure 2. User Interaction Table Creation

3.3 New Article Fetching and Processing

To fetch news in our application, we used 'get_top_headlines' endpoint from NewsAPI, a simple and developer-friendly RESTful service that provides real-time access to news articles from a wide range of sources. The get_top_headlines function allows us to retrieve the most recent and relevant news stories based on parameters such as country, category, and keyword, which we then process for topic modeling and recommendation. However, it does not include the full text of the articles. To address this limitation, we leveraged the article URLs returned in the response and used the Axios package to fetch the complete content directly from the

source websites. Once retrieved, the full-text articles were ready for processing.

For the topic modeling component, we utilized the BERTopic framework. As a preprocessing step, we embedded the article content using the "all-MiniLM-L6-v2" sentence transformer model to enable faster and more efficient calculations. We then passed the list of embeddings and corresponding article texts to the BERTopic model, which assigned each article a topic label and produced a probability distribution indicating how well each article aligned with the identified topics. To further improve the topic labeling, we added the GPT model as part of the fine-tune process, which generates more human-understandable names that will be used for displaying news in the newsroom.

3.4 Recommendation Logic

There are two API build together keep user profiles current and power personalized news recommendations: UserProfileService.ts fetches a user's recent explicit (ratings) and implicit (dwell-time) interactions, convert them into interest weights, with explicit feedback having more weights in determining user's interest (how we convert will be explained in details in the following section). Then, we applies an exponential time decay to reflect "decayed interests", meaning that if the interaction is long ago, then the users might not be interested in it anymore, and thus we give less weight to them. Finally, we upserts the resulting user profile which is a normalized "probabilistic embedding" into a MySQL table; RecommendationService.ts then loads that user profile embedding along with candidate articles' stored embeddings, computes cosine similarities between each article's embedding and user profile's embedding, and returns the top N articles with the most similarities; Finally, the /api/recommendations route exposes a single authenticated POST endpoint which, when invoked by the client, simply calls the two services in one-go and return the recommended articles.

In more detail of the calculateUserProfile function, it takes a user's past interactions—each of which pairs an article's stored embedding vector e_i with a timestamp t_i, an explicit rating r_i, and an implicit dwell-time τ_i—and turns them into a single, up-to-date "profile" embedding by following these steps:

1. Compute recency decay for each interaction:

$$w_i^{\text{rec}} = \exp\big(-\lambda\,\Delta_i\big), \quad \Delta_i = \text{daysSince}(t_i). \tag{1}$$

2. Convert explicit feedback (ratings r_i) into a score directly. Then, convert implicit feedback into a score using time spent reading (τ_i) and WPS (words-per-second) along with document-length (ℓ_i) to calculate number of reads ($n_{\text{read},i}$) and number of subsequent reads ($n_{\text{sub},i}$), and then finally convert the number of reads into score:

$$w_i^{\text{exp}} = r_i, \tag{2a}$$

$$n_{\text{read},i} = \frac{\tau_i\,\text{WPS}}{\ell_i}, \tag{2b}$$

$$n_{\text{sub},i} = \max\big(0,\ n_{\text{read},i} - 1\big), \tag{2c}$$

$$w_i^{\text{imp}} = S_1 + n_{\text{sub},i}\,S_N, \tag{2d}$$

In the above equations S_1 denotes stars for one visit, S_n is stars awarded for each subsequent read.

3. Combine them into a single weight per interaction:

$$w_i = w_i^{\text{rec}}\big(\alpha\,w_i^{\text{exp}} + \beta\,w_i^{\text{imp}}\big). \tag{3}$$

In the equation above, α is the scaling factor for explicit feedback (w_i^{exp}), and β is the scaling factor for implicit feedback (w_i^{imp}), usually $\alpha > \beta$. Then, combining both feedback components along with the recency decay (w_i^{rec}) provide us with the interest score (w_i).

4. Accumulate the embeddings weighted with interest score and normalize to get the final user profile:

$$\hat{\mathbf{u}} = \frac{\sum_{i=1}^{N} w_i\,\mathbf{e}_i}{\sum_{i=1}^{N} w_i}. \tag{4}$$

In other words, computeWeightedProfile blends what you've explicitly rated and how long you've lingered on each article, while down-weighting older interactions, to produce a single embedding that best reflects your current interests.

These components form a clean, decoupled pipeline for ingesting feedback, updating profiles, and serving up-to-date personalized recommendations.

4 User Guide

4.1 Installation Instructions

First, clone the project from the remote repository and install dependencies:

1. Clone the repository: "git clone https://github.com/Rachanon-Andrew-Petchoo/news-recommendation-system.git"
2. Run "cd news-recommendation-system/"
3. Install Node.js:
   - macOS: "brew install node"
   - Ubuntu/Debian: "sudo apt update" "sudo apt install nodejs npm"
4. Install project dependencies:
   - "npm install"
   - "pip install -r requirements.txt"
5. Create a ".env" file using the contents from this file:
   - Link: "https://www.protectedtext.com/cs510-sp25-group9"
   - Password: cs510
6. Run the development server: "npm run dev"

You can view the app by visiting http://localhost:3000

4.2 Example Workflow

On the homepage, start by clicking "Sign in" in the top-right corner and log in with your Google account. This allows the system to store a personalized recommendation model for you in the database.

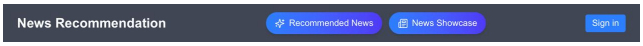

Figure 3. Top Navigation Bar

After logging in, start by visiting the News Showcase page, which contains all available news articles, categorized into categories. You can browse news by selecting different categories. The main purpose of this page is to let you explore the news that match your interests, and we'll build your personalized recommendations based on your reading habits.

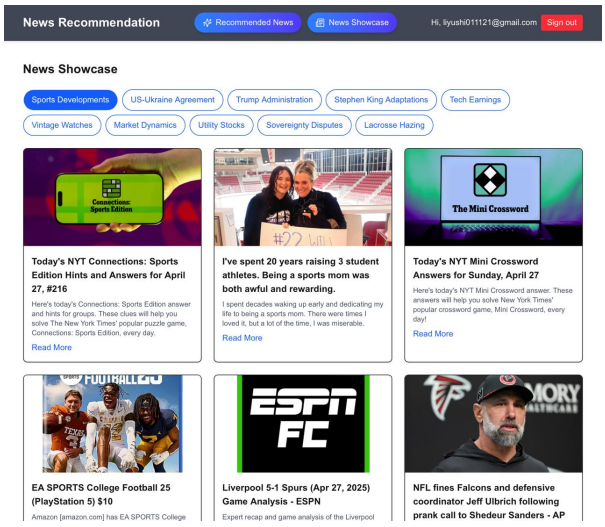

Figure 4. News Showcase Page

Once you find an article that you want to read, you can click on it and view the news in the News Display page (as shown below). You'll find the title of the articles, along with their details like author, publishing date, and short description. Moreover, you can scroll down to see the article rendered natively on our website, or if you want, you can visit the original source of the article by clicking the "Open Full Article" button as well. Finally, if you like the news, then you can provide ratings based on how interested you are in it. This will help us prioritize this kind of news in your "Recommendations" page. We also keep track of how much time you've spent on the article, the longer you stay on the page, the more "assumed interests" we'll give to your user profile. Our model uses these two types of feedback (rating and time) data to recalculate and improve the recommendations tailored for you.

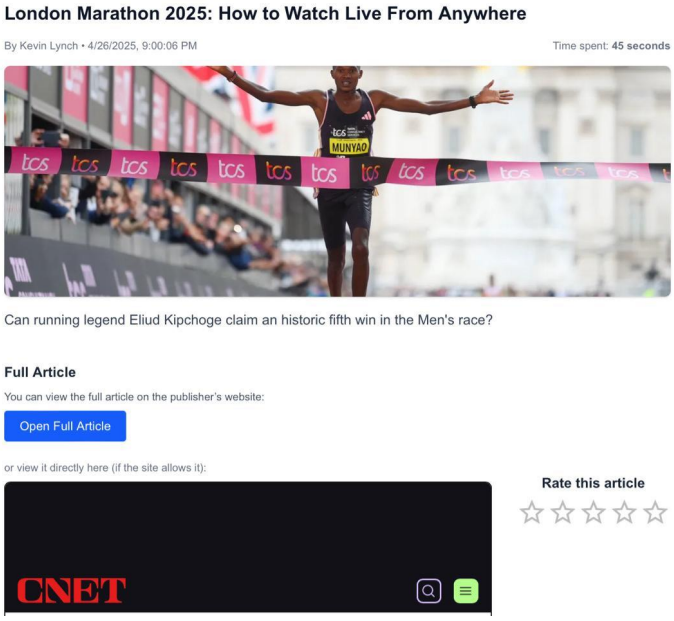

Figure 5. News Display Page

After reading some articles and providing ratings, you can visit the Recommended News page, where news tailored to your interests will be displayed.

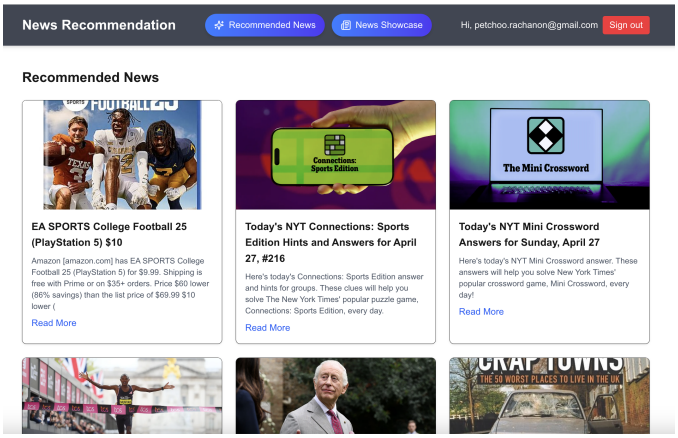

Figure 6. Recommended News Page

5 Evaluation and Results

We evaluated the recommendation algorithm through a series of usage scenarios, each designed to isolate and test a specific feature of the system. This section presents our findings.

Scenario 1 – Explicit Feedback Prioritization

Setup: A user was shown a mixture of technology, sports, and political news. They rated several technology articles with high scores (4–5 stars) and gave lower ratings (1–2 stars) to articles in other categories. They spent about 15 seconds on the technology news, and about 1 minute on each of the other categories.

Result: In subsequent recommendation rounds, the top-ranked articles were overwhelmingly technology-focused, despite the presence of implicit signals (e.g., time spent) on non-tech articles. This indicates that explicit feedback was correctly weighted more heavily than implicit feedback, demonstrating that the system prioritizes direct user preferences.

Scenario 2 – Implicit Feedback via Reading Time

Setup: The user did not rate any articles, but spent significantly longer time on a few sports articles and skimmed through political ones.

Result: Sports-related articles appeared in the top 5 recommendations, showing the algorithm's ability to effectively infer preferences from reading time. Based on words-per-second (WPS) calculations and document length, these long-duration sessions were translated into multiple "virtual reads." Articles that were visited multiple times were assigned bonus scores, further boosting their category in the recommendations.

Scenario 3 – Document Length Normalization

Setup: The user read both a short 300-word article on finance and a long 1500-word article on entertainment, spending roughly 2 minutes on each without giving any ratings explicitly.

Result: Without normalization, the longer article would have been considered equivalently relevant due to time spent. However, after applying document length normalization (via WPS), the calculated "number of reads" showed higher engagement with the short article. As a result, finance articles were recommended more strongly than entertainment, validating the effectiveness of normalization.

Scenario 4 – Interest Decay

Setup: The user initially showed a strong interest in politics articles over the first day. Two days later, they began engaging heavily with sports content but ignored new politics articles.

Result: The recommendations shifted away from politics and increasingly favored sports content. Older interests naturally decayed in influence, ensuring that outdated topics no longer dominated the recommendation list. This confirms that the time-decay mechanism correctly lowers the weight of stale interests.

6 Individual Contributions

Rachanon Petchoo served as the project coordinator, managing team communication and overall progress. He wrote the API to call all of our services and tie them together, making sure that the backend services work together seamlessly. He also made the whole UI, and also integrated it with the APIs, ensuring the UI is properly connected with the backend.

Shuning Zhang focused on topic modeling using BERTopic. She fetched news articles, processed embeddings, and stored them in the database. She also collaborated on evaluation strategy design.

Mayank Umesh Shetty led the development of recommendation logic. He designed and implemented the user profiling method and handled computation of similarity scores for article ranking.

Yushi Li implemented the backend using FastAPI, including user authentication and database integration. He also deployed the initial backend environment and maintained API documentation.

7 Conclusion and Future Work

7.1 Summary

We developed a personalized news recommendation system that delivers real-time content tailored to individual user interests. The system leverages BERTopic to extract dominant themes from news articles and models user preferences based on both explicit feedback and implicit reading behavior. It features a responsive frontend, a scalable backend, and a relational database to manage user profiles, article metadata, and interaction history, enabling accurate and dynamic content recommendations.

7.2 Future Improvements

In the future, we plan to:

- **Cold Start Problem:** Addressing new users with limited interaction history remains a challenge. Incorporating demographic data or short onboarding surveys could help generate initial interest profiles.
- **Better model:** Improve recommendation accuracy using advanced neural models like BERT4Rec.
- **Diversity:** Our application only supports English readers, adding multilingual news support will help us expand our user base.
- **Deployment:** Deploy the platform for public access and gather real-world usage data.
- **A/B Testing Framework:** Introducing an A/B testing pipeline would allow systematic evaluation of different recommendation strategies, user interfaces, and feedback mechanisms.

References

[1] Maarten Grootendorst. 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794* (2022).

[2] Xiangfu Meng, Hongjin Huo, Xiaoyan Zhang, Wanchun Wang, and Jinxia Zhu. 2023. A Survey of Personalized News Recommendation. *Data Science and Engineering* 8, 4 (12 2023), 396–416. doi:10.1007/s41019-023-00228-5

[3] Chuhan Wu, Fangzhao Wu, Yongfeng Huang, and Xing Xie. 2023. Personalized News Recommendation: Methods and Challenges. *ACM Trans. Inf. Syst.* 41, 1, Article 24 (Jan. 2023), 50 pages. doi:10.1145/3530257

[4] Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. 2021. Empowering News Recommendation with Pre-trained Language Models. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 1652–1656. doi:10.1145/3404835.3463069

[5] Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, and Ming Zhou. 2020. MIND: A Large-scale Dataset for News Recommendation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 3597–3606. doi:10.18653/v1/2020.acl-main.331