Breaking the News Content Bubble: A Hybrid Recommendation System Approach

Kaushika Uppu, Miranda Billawala
Computer Engineering Department
San Jose State University
San Jose, USA
kaushika.uppu@sjsu.edu, miranda.billawala@sjsu.edu

***Abstract*—News recommender systems often prioritize maximizing user engagement, leading to a push on content that users are highly likely to agree with and reflect their perspectives or points of view. However, that tends to lead to a news content bubble where users are not exposed to conflicting perspectives. In this project, we aimed to alleviate that issue by first predicting the probability of a user clicking on an article and then using that to recommend atypical articles that would still catch their interest. We found that XGBoost performed better than NCF for predicting probability, with an AUC of 0.644 and a recall of 0.414. For the recommendations, due to the models not predicting click probability well, accuracy was very low.**

Keywords—recommendation system, news, content bubble, bias

I. Introduction

In a world where conflicting perspectives lead to very strong divides between individuals, social media and news outlets only further bias through extreme user personalization. The rapid development of machine learning and artificial intelligence systems geared towards building robust recommendation systems that push only content users are likely to agree with plays a role in the creation of a news content bubble as well.

The main goal for most news recommender systems is to learn user preferences and recommend content that will increase user engagement [1]. However, with that as a priority, these systems tend to create a filter bubble of news content that aligns with user perspectives and leads to an echo chamber. With collaborative filtering algorithms, there is also the possibility that since similar users are used to recommend articles, users will most often see content that people with similar views and/or interests also agree with or enjoy. This, in turn, also contributes to further polarization of views, and it is algorithmically being pushed by news recommender systems.

The objective of this project is to develop a news recommender system such that users are recommended articles that they are interested in and are likely to click on, but to also incorporate articles that fall outside of their typical readings and/or point of view. Achieving this goal required meeting the following milestones:

1) Build a model to accurately predict whether a user would interact with an article
2) Use our predictions and other measures of article content to recommend atypical articles for a user that would still get their interest

Building this system would help alleviate the content bubble issue as users would be exposed to multiple perspectives rather than just their own.

II. System Design and Implementation

A. Algorithms

One of the algorithms that we chose for this project is XGBoost. XGBoost utilizes decision trees with boosting and parallel processing to provide fast and accurate results. Boosting is the process of incrementally improving decision trees in each step to minimize loss. The algorithm often performs well with complicated datasets, with minimal overfitting. We chose this algorithm knowing our dataset would be large and required a fast model, and that we would have a large number of features from embeddings.

The other algorithm we decided to use is Neural Collaborative Filtering and the two-tower model. This algorithm was chosen because it combines regular matrix factorizations with neural networks to better predict interactions between users and articles, especially non-linear relationships. We decided to extend the base model with just user and article IDs by adding article metadata that was feature engineered from the news metadata given in the dataset. However, this algorithm is fairly computationally intensive, and would take a long time to run when training.

B. Technologies and Tools

For feature engineering on building a model that could predict if a user would interact, we applied a number of different tools, which included multiple Python modules: OneHotEncoder, SentenceTransformer, Principal Component Analysis, and TextBlob.

OneHotEncoder is used to convert categorical data into machine readable formats. Simply giving each category a number from 1 to the number of unique values may indicate relationships between categories with similar encodings that do not exist. Instead, OneHotEncoder creates a column for each unique categorical variable and the value is 0/1 if the category is not associated or associated with the row. Part of

the metadata for the articles included category and subcategory. We used this tool to transform the category and subcategories into numerical format.

SentenceTransformer [4] takes text and returns a vector of dimension 384 that best represents the contents. We used the 'all-MiniLM-L6-v2' sentence embedding model due to its versatility with a variety of topics and its speed, since we are iterating through a large dataset. The transformer was run on both title and abstract for the model to understand the text in a numerical format.

Principal Component Analysis (PCA) works to reduce the dimension of a matrix by identifying the most relevant features. Considering our use of OneHotEncoder and SentenceTransformer, PCA was necessary to decrease the number of features input into the model to lower runtime and to avoid extreme complexity, which may confuse the model.

TextBlob is a Python module that is used for text processing. In our case, we utilized it for sentiment analysis on the titles and abstracts of news articles. TextBlob takes in text input and returns a polarity score between -1 and 1, ranging from highly negative to highly positive in terms of sentiment.

Multiple evaluation metrics were used throughout this project for the models that were predicting probability of clicking. Basic accuracy was calculated, along with precision, recall, and F1 to evaluate model performance. Finally, area under the curve (AUC) was used, which gives the probability that the model ranks a positive example (clicked) higher than a negative one (not clicked). This was the main metric used to evaluate how the models were performing, especially because it is particularly useful in comparing the performance of two different models [5].

For the recommendation part of our project, we used cosine similarity. Cosine similarity calculates how close two vectors are by looking at the angle between them, essentially checking if they are going in similar directions. Using cosine similarity on articles, we were able to differentiate between articles to decide what to recommend to a user.

C. Architecture

In terms of the structure and workflow of our project, it was split into two parts: one for predicting click probabilities, and the second for recommending articles to users (Fig. 1).

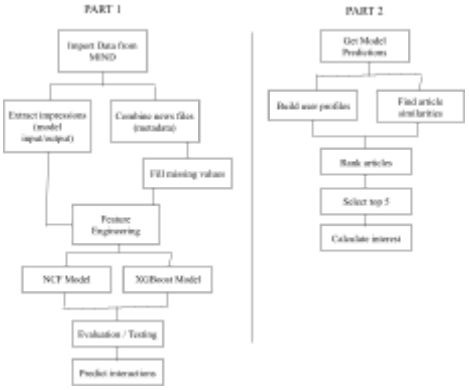

Fig. 1. Project architecture for predicting click probability (Part 1) and recommending articles (Part 2).

In Part 1, we first imported the dataset, performed preprocessing, and then engineered features from the data. Then, we used either XGBoost or the NCF model to predict probabilities of a user clicking for an article, and finally evaluated how the models did. More in-depth explanations of data preprocessing and feature engineering stages are under sections *B* and *C* of the Experiments component of this paper.

In Part 2, we used the outputs of the model from the previous step for user recommendation. First, we built user profiles and found article similarities using cosine similarity, and then utilized the model predictions to rank articles and select the top five for a user. Finally, we examined the user's actual interest in the articles that we recommended. An in-depth breakdown of this process is shown in section *E* of the Experiments component.

For the NCF model, Fig. 2 reveals the structure of the model. There were three input layers, for users, articles, and article metadata, which were converted to embedding layers and flattened. Dense layers were added in order for the model to learn non-linear relationships in the data, which is why the 'relu' activation function was used. L2 regularization was also used throughout the model to try and prevent overfitting. User and item towers were concatenated and put through further dense layers as well as batch normalization to try and decrease training time. A dropout layer was used to also further prevent overfitting and improve generalization. Finally, the output layer consists of a 'sigmoid' activation function so the model will predict probabilities ranging from 0 to 1.

Layer (type)	Output Shape	Param #	Connected to
item_input (InputLayer)	(None, 1)	0	–
user_input (InputLayer)	(None, 1)	0	–
embedding_7 (Embedding)	(None, 1, 16)	1,666,432	item_input[0][0]
metadata_input (InputLayer)	(None, 30)	0	–
embedding_6 (Embedding)	(None, 1, 16)	4,095,856	user_input[0][0]
flatten_7 (Flatten)	(None, 16)	0	embedding_7[0][0]
dense_9 (Dense)	(None, 64)	1,984	metadata_input[0…
flatten_6 (Flatten)	(None, 16)	0	embedding_6[0][0]
concatenate_2 (Concatenate)	(None, 80)	0	flatten_7[0][0], dense_9[0][0]
dense_8 (Dense)	(None, 64)	1,088	flatten_6[0][0]
dense_10 (Dense)	(None, 64)	5,184	concatenate_2[0]…
concatenate_3 (Concatenate)	(None, 128)	0	dense_8[0][0], dense_10[0][0]
dense_11 (Dense)	(None, 64)	8,256	concatenate_3[0]…
batch_normalization (BatchNormalizatio…	(None, 64)	256	dense_11[0][0]
dropout (Dropout)	(None, 64)	0	batch_normalizat…
dense_12 (Dense)	(None, 1)	65	dropout[0][0]

Fig. 2. NCF model architecture showing all of the layers.

III. Experiments

A. Dataset

The dataset that is used for this project is the Microsoft News Dataset (MIND) [2], which contains user behavior logs from Microsoft News. MIND has information for about 65,000 English news articles, as well as behavior logs from around 50,000 randomly sampled anonymous users generated over a period of six weeks.

The user behavior logs contain clicks, time of the interaction, previously clicked articles, and impressions (clicked or not clicked) for all articles shown to a user. In addition, the dataset contains news article metadata for all of the articles, including categories, subcategories, title, abstract, and entities. Finally, the dataset has embedding vectors for all entities and relationships, which were already obtained previously using the TransE method [3].

The dataset provided was first published for use in a competition to most accurately predict user interactions. Because of this, the data was already split into training and validation sets with the four aforementioned files for both.

B. Data Preprocessing

Our first step was converting given files into usable formats and combining them where necessary. User impressions were given as a list of all articles shown in an impression as well as their response (clicked or not clicked). We exploded these lists to give each news article and impression a distinct row, changed timestamp to seconds since the start of the study, and converted user ID and news ID from categorical to integer values. We additionally separated the impressions into clicked and not clicked to help with the second step of building a user profile. Despite the larger file size from expanding the lists, we decided to save as a CSV file due to long preprocessing run times. We combined the news files from training and validation sets, dropping duplicates, for ease of use when feature engineering.

After altering the format, we started cleaning up the actual values. This meant filling all NA values, which were only present in the abstract of some news articles, with empty strings. There were no other missing values to deal with.

C. Feature Engineering and Analysis

To incorporate the metadata, we applied a number of different techniques. However, although we attempted various methods and features, not all of them ended up being implemented in the final project as they decreased performance.

Categories and subcategories were provided for every news article. We started by one-hot-encoding the two. While the number of categories was small (18), there were 270 different subcategories. For this reason, we then applied PCA to significantly reduce the dimensions. For XGBoost and NCF, PCA was then performed on the combined category and subcategory one-hot-encoded columns and reduced to 18 components, for the number of unique categories.

Another important step was incorporating the titles and abstracts into model input. As mentioned previously, the dataset included 100-dimensions embeddings for certain keywords in both the title and abstract. To represent the title and abstract the sum of the embeddings were included for each article. However, about 26.2% of the titles and abstracts did not have any entities given. Around 12.5% of the articles had neither. For this reason, we attempted an independent approach to getting embeddings using SentenceTransformers. Running both through SentenceTransformers and reducing the dimension proved to be a more effective technique.

Three different values for PCA components were tested using the NCF model: 10, 20, and 50. When the model was fit on the training data and then predicted on the test set, we found that AUC and recall were highest for 10 components, while precision and accuracy were highest for 20 components. Fig. 3B shows the metrics subtracted by the lowest metric for each one, revealing the small differences between the number of components. However, precision was overall low for all tests, being under 0.05 (Fig. 3A). This is likely because the validation dataset was very imbalanced.

(A)

	ROC-AUC	Accuracy	Precision	Recall	F1-Score
10	0.559535	0.565307	0.047977	0.514630	0.087772
20	0.558686	0.682949	0.049889	0.376970	0.088116
50	0.533239	0.565212	0.043932	0.467172	0.080312

(B)

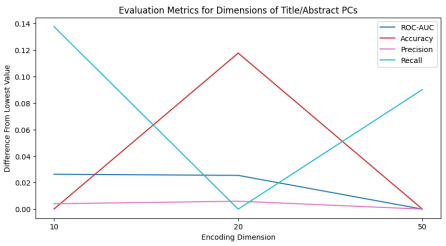

Fig. 3. (A) Metric comparison for 10, 20, and 50 principal components for title and abstract encodings. (B) Graphical representation of difference of metrics from the lowest value of each.

In addition, we visualized training and validation loss for the three different numbers of components. As shown in Fig. 4, while training loss did decrease for all three, validation loss increased over epochs except for 20 components. This further reveals that the NCF model was overfitting, as what it learned during the training phase did not generalize during validation.

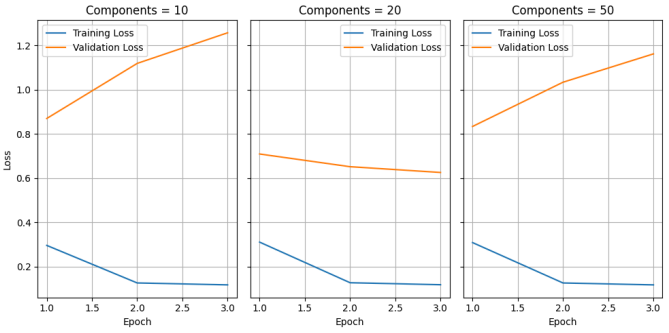

Fig. 4. Training and validation loss comparison between different components.

We did try and incorporate the embeddings that were given in the data set. To do this, we extracted the entity IDs for each article for title and abstract (if given). Then, we took all the entities in the title and abstract combined, and computed a weighted average of all of the entity embeddings for an article. The weights were based on how often an entity was seen throughout all of the articles. When this was applied, we ended up with a 100-dimensional embedding vector for every article that aggregated the entities in the title and abstract. This value was only left blank if an article had no title or abstract entities given. However, when tested using the NCF model,

performance went down, and due to the number of articles that did not have entities given, as mentioned above, we decided against using this method.

We also did sentiment analysis on the titles and abstracts of articles. For this, we used TextBlob and calculated a sentiment score ranging from -1 to 1 for the titles and abstracts separately. We added these as features to our data to incorporate more metadata that we believed would help the models understand the probability of clicking better. However, using the sentiment analysis led to very minor improvements in AUC, recall and precision.

Since the data was time-series, we tried to incorporate user history. To do this, we found all the unique timestamp and user ID pairings, then calculated the mean of all the clicked article embeddings (from SentenceTransformer). The idea was to capture the typical articles a user clicked on in the past for the model to learn off of. However, many users had no click history so the effectiveness was minor. Likely due to the number of additional columns added from the embedding, with not informative data, the model's score actually decreased.

D. Evaluation

As mentioned previously, the dataset provided was split into training and validation. It followed a roughly 70-30 train-test split. However, since users tend to only click on a small subset of the articles shown, the dataset was extremely imbalanced in the validation set.

For NCF, when run with 20 principal components for title and abstract embeddings, we got a final AUC of 0.547 and recall of 0.458. Precision was very low (< 0.05) due to the fact that the model predicted more articles to be clicked than actually were, since the validation dataset had so few articles that were clicked.

XGBoost performed better than NCF, with a final AUC of 0.644 and recall of 0.414. Once again, precision was low, similar to NCF, for the same reason. However, since XGBoost had the better performance overall, we decided to use the predicted probabilities from this model for our user recommendations.

E. User Recommendation

Our task was to pick up to five articles to recommend to the user, chosen from the validation set so we would know if they did indeed interact with the article. Since not every user was guaranteed to be shown enough articles or to be predicted to interact with five articles, some had less recommendations. In deciding what to recommend, we wanted to choose articles in categories they preferred, but that were different from what they had previously read. This provides a level of interest in the content, while exposing different perspectives to the user.

Our first step was to build user profiles based on previous clicks. We calculated top categories based on the number of articles clicked in a category. If a user had no click history, they were recommended articles in the overall popular categories. Then, we ranked each article shown to a user. Anything not predicted to be clicked was excluded from the ranking since we do not want to lose interest by showing entirely unrelated articles. When ranking articles, we sought to strike a balance between category of interest, likelihood of clicking, and dissimilarity to normal content. To do this, we determined three thresholds for probability of clicking: [0.6, 0.7), [0.5, 0.6), and [0.7, 1]. Using these thresholds, we iterated through the categories, looking at all articles within a threshold and ranking the least similar highest. After iterating through the user's preferred categories we chose recommendations in the following way:

1. Chose up to five from the [0.6, 0.7) threshold
2. If we still needed more recommendations, we looked at the articles ranked in [0.5, 0.6)
3. If we still needed more, we moved to the [0.7, 1]

If there were not enough articles in the user's top categories, we then looked at the articles in every other category, ranking again in the thresholds and selecting in the same manner. The thresholds were selected in this order because we emphasized wanting to ensure a user will still be interested in the content shown. The worst case would be exacerbating the bubble of news, which is why articles with high probability of being clicked were considered last.

After selecting the articles to be recommended for each user, we found the accuracy of how many recommendations were actually clicked. As a result of poor performance of the click prediction algorithm, this accuracy ended up being around 10%. Upon further examination, we saw that nearly every article with a predicted probability of clicking in the 0.6-0.7 range ended up not clicked.

IV. Discussion and Conclusion

The dataset ended up being a lot harder to work with than initially assumed, requiring many changes in direction throughout the process.

The MIND competition itself focused on a very different task than ours. Rather than accurately predicting whether a user interacted or not, the goal was to accurately rank the articles in an impression from most to least likely to interact. This meant we could not apply the performance or metrics to our project since we did not have the same goals. While we wanted to submit to the competition, the approach ended up differing too much leading to very different results.

Additionally, despite our best efforts at feature engineering and trying a variety of approaches with the metadata given, very few features ended up improving our results. Metrics often barely increased and sometimes even went down due to the increased complexity from unhelpful data. The models were both biased towards classifying articles as not clicked due to the heavy imbalance – even after we put measures in place to alleviate the issue. The cold-start issue and users with very little data posed a big challenge, especially since we had no user-level information and our metadata features performed poorly.

As a result of our low prediction scores, it became quite hard to accurately gauge our recommendations. The articles we could recommend were limited by a couple factors. First, by the articles we had results for from the validation set, which went as low as two. And second, by the model's

predictions. Since we do not want to predict an article that a user is not likely to be interested in (< 0.5 chance of clicking), we removed any such article. That gave us less than 25% of the validation set to work with for recommending. With a large chunk of those being incorrectly predicted, the scores for our recommendations were extremely low. The recommendation aspect of our project relied heavily on high performance in the first prediction step; however, since we could not achieve that, the second step was very negatively impacted. While the idea for recommendation is strong, we need better predictions and increased flexibility in articles to truly understand the performance. Our system would strongly benefit from A/B testing.

While we believe that the project idea was strong, complications in model building and dataset limitations led to inconclusive results on the effectiveness of our approach. Future work would likely benefit from a different dataset or additional feature extraction that better captures the variation in the data.

V. Project Plan and Task Distribution

The first step of importing and formatting the data was assigned to Miranda. For models, Miranda worked with XGBoost and Kaushika worked with Neural Collaborative Filtering. Knowing feature engineering would be the toughest part of the project, we both individually worked on it. We communicated our results constantly, bouncing ideas off of each other, and implementing approaches the other proposed. Our goal was to simply build the best version of the algorithm we were assigned.

For the recommendation part, we agreed to work on it together. We got on a call and coded this part together. While only one person committed the changes to GitHub, the recommendations system was built together.

For the report and presentation, we did not explicitly assign parts. We each added information specific to what we had worked on with the models – Miranda talked about XGBoost, the results, and challenges, and Kaushika did the same for NCF. And, for the remaining sections, we filled them in as we had time. We both completed the tasks we were given and took initiative to work on the remaining parts without needing to assign parts.

References

[1] H. Han, C. Wang, Y. Zhao, M. Shu, W. Wang, and Y. Min, "SSLE: A framework for evaluating the 'Filter Bubble' effect on the news aggregator and recommenders," *World Wide Web*, vol. 25, no. 3, pp. 1169–1195, Mar. 2022, doi: https://doi.org/10.1007/s11280-022-01031-4 (accessed Apr. 30, 2025).

[2] F. Wu *et al.*, "MIND: A Large-scale Dataset for News Recommendation," ACL, 2020. Accessed: Apr. 28, 2025. [Online]. Available: https://msnews.github.io/assets/doc/ACL2020_MIND.pdf

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating Embeddings for Modeling Multi-relational Data," *Neural Information Processing Systems (NIPS)*, Dec. 2013, Available: https://www.researchgate.net/publication/279258225_Translating_Embeddings_for_Modeling_Multi-relational_Data. [Accessed: Apr. 28, 2025]

[4] "SentenceTransformers Documentation — Sentence Transformers documentation," *Sbert.net*, 2019. https://sbert.net/index.html (accessed Apr. 30, 2025).

[5] Google Developers, "Classification: ROC Curve and AUC | Machine Learning Crash Course," *Google Developers*, 2019. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (accessed Apr. 30, 2025).