MIND News Recommendation Challenge 2020
Technical Report for 4^{th} Place (3^{rd} Prize) Solution

Mathieu Ravaut
Independent, France
mathieu.ravaut@gmail.com

1 Introduction

Online news are now ubiquitous and are accessed on billions of devices daily [1]. With the abundance of news content being created everyday, it is of greater and greater importance to filter out news and recommend a selected relevant few to each user. Without news recommendation, user experience would be very poor as users would drown in the flow of articles.

In the MIND News Recommendation Challenge, the participants task is to build a model predicting whether or not a user will click on a given news, given a user history. In other words, given a user news clicking history:

$$history_i = [history_{i_1}, ..., history_{i_{n_i}}]$$

and a list of candidates:

$$candidates_i = [candidates_{i_1}, ..., candidates_{i_{m_i}}]$$

where n_i and m_i are the history length and the number of candidates for user i, respectively, we shall predict m_i probabilities, one for each candidate news article.

For this challenge, we leverage the new MIND dataset [2]. This large-scale dataset provides 24,155,470 clicks sampled from 1,000,000 users on 161,013 articles, collected from logs of Microsoft News over a six weeks period. Clicks from the last week are used for test, while those from the fifth week are used for training and validation. Among this fifth week, the last day is used for validation. Clicking history is extracted over the first four weeks. Besides, the dataset comprises multiple information on each news article: title, abstract, body, category, sub-category, and extracted entities from the title and the abstract.

Models are evaluated with Area Under the ROC Curve (AUC) computed over each impression list then averaged over users, mean reciprocal rank (MRR), and normalized discounted cumulative gain for 5 and 10 shown recommendations (nDCG@5 and nDCG@10). AUC is used for the final ranking.

2 Approach: NRMS with attentive multi-view learning

An intuitive way to model the news recommendation problem is to compute a representation of the user, a representation of the candidate news article, and then dot-product the two representations to get the click likelihood prediction. Typically, neural networks are used as encoders to get user and news representations. The news encoder can be used on the user's news clicking history as a base block for the user representation.

My main approach relies on the recent Neural News Recommendation with Multi-Head Self-Attention (**NRMS**) model [3]. In this model, the news encoder first layers consists in word embeddings on the title. The second layer is word-level self-attention, to capture interaction between the title's words. Lastly, an additive attention mechanism forms the third layer of the news encoder, to capture the importance of each word. The NRMS user encoder follows the same principles. After getting the news representation of each news in the history, a news-level multi-head self-attention layer is used to get the relatedness between news, as we can assume that news from a same user history are related. Then, an additive attention network computes the importance of each news article.

The original version of NRMS only leverages news titles. After implementing the NMRS model on news titles, as suggested by the MIND paper [2], I experimented with incorporating other sources of text data than the title. Since concatenating text channels (e.g. title, abstract, etc) makes little sense, one has to get representations for each text channel first. I used multiple NRMS news encoders, one for each text channel. Then, I leveraged attentive multi-view learning (NAML) to get a unified and meaningful representation [4], shown in Figure 1. Specifically, the last component of the news encoder in NAML is attentive pooling, which allows to weigh each text channel. Thus, with such attentive pooling, one can theoretically work with as many text channels as available.

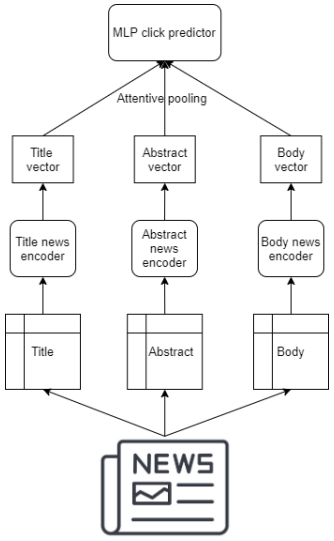

Figure 1: Overview of NRMS with attentive multi-view model applied on three text channels: title, abstract and body.

3 Models

I experimented with several choices of text channels in the NRMS with attentive multi-view learning. The best performing ones were:

- Title + abstract + category + sub-category (**TACS**)
- Title + abstract + body + category + sub-category (**TABCS**)
- Title + abstract + category + sub-category + features (**TACSF**)
- Title + abstract + body + category + sub-category + features (**TABCSF**)
- Title + abstract + category + sub-category + features + entities (**TACSFE**)

"Features" here designate manually engineered news-level features designed to capture a news article popularity. These features were built over the clicking histories only. These features are: how many users clicked on an article (during the whole six weeks period), the mean and standard deviation (over users) of the number of news that these users clicked on per session, and the mean and standard deviation (over users) of the total number of news that these users clicked on during the period.

Furthermore, since my best submissions are ensembles of multiple models, I also built models of different nature to get less correlated predictions and improve the ensembling:

- Long-and-Short-Term User Representation (**LSTUR**) [5]. This recently introduced deep news recommendation model proposes an elegant way to learn both short-term and long-term user preferences. The sort-term user representation is modeled with a GRU [6] recurrent network, while the long-term one is learned through embedding the user ID. I trained a LSTUR model on title + abstract + category + sub-category + features, and experimented with both LSTM [7] and GRU architectures as short-term user encoders. In my implementation of LSTUR, I also leveraged multiple text channels with attentive multi-view learning
- XGBoost [8]. This gradient boosting decision tree package is a classical machine learning algorithm successfully used on multiple recommender systems competitions [9]. I trained an XGBoost model on news-level features as described above, then aggregated over the user history and concatenated with news-level features from the news candidate impression list.

4 Experiments & Results

All experiments were run on two GPUs: a NVIDIA GTX 2060 6GB, and a NVIDIA GTX 1080 Ti 11GB. All experiments were run on an Intel Core i7-10750H CPU @ 2.60GHZ with 12 cores.

All deep neural network models were implemented in Python, in the Keras framework [10]. The Microsoft Recommenders package (`https://github.com/microsoft/recommenders`) was used as a starting point for the neural network implementations.

Models were trained with the Adam optimizer [11], and a learning rate of 0.001, which was the best among the list [0.1, 0.01, 0.001, 0.0001] of learning rates tried. Models were evaluated on the validation set three times per epoch. NRMS and LSTUR models were overfitting after training for more than a few epochs, and I used early stopping to find the best iteration to stop training at. Typically, this best iteration was between 1 and 2 epochs.

With an average abstract length of 43 words, and an average body length of 585 words [2], using titles alongside both abstracts and bodies becomes challenging for memory usage. I truncated abstracts to the first 40 words, which conveniently did not harm the performance too much compared with taking a long subset of 70 or 80 words. Regarding bodies, I first summarized them with the pysummarization Python package (`https://pypi.org/project/pysummarization/1.0.5/`), reducing the average body length to 182 words. I further capped body length to the first 300 words.

Table 1 shows results of each individual model on the validation set.

Model	Validation AUC
NRMS baseline (title only)	67.94
NRMS TACS	68.71
NRMS TABCS	69.02
NRMS TACSF	69.14
NRMS TABCSF	68.62
NRMS TACSFE	69.29
LSTUR TACSF	68.82
XGBoost	57.35

Table 1: Validation AUC for the different models used. The NRMS baseline with just the title was not included in the final ensemble.

Throughout this competition, I have found a few tricks to be of critical importance to achieve competitive results:

- **Weighted average ensembles**. Ensembling different models significantly helped push the AUC higher. Assigning higher weights to better performing models worked better than vanilla averaging, and it is worth noting that even vanilla averaging significantly improved on the best model in the ensemble. To blend models, I also experimented with ranks in the following manner: rank predictions over all models, then sum ranks for each news candidate, and rank news candidate from the lowest sum to the highest. Such process was performing slightly lower than weighted averaging.
- **Training on the validation set**. A common way to push model performance higher is to re-train the model on the concatenation of the training and validation sets. More training data may help, but training blindly also means a greater risk of overfitting. Typically, the best early stopping iteration is found by tracking the validation set performance, then one re-trains on the concatenated set for this exact number of iterations. In this competition, it may be due to the temporal nature of the training/validation/test split that training on the validation set gives an improvement.
- **Snapshot ensembling**. Lastly, I also found it slightly beneficial to include several checkpoints from a given model instead of just one in the final ensemble.

Table 2 shows results of ensembles on the test set. My final ensemble (last row of Table 2) consists of 13 models: two NRMS TACS, two NRMS TABCS (one trained on T, and one trained on T+V), four NRMS TACSF (two trained on T, and two trained on T+V), one NRMS TABCSF (trained on T+V), two NRMS TACSFE (one trained on T, and one trained on T+V), one LSTUR, and one XGBoost. This final model achieves fifth place on both the public and private test sets.

Model	Public Test AUC
Weighted avg T+V	70.26
Weighted avg T and T+V	70.40
Weighted avg T and T+V + snapshot ensembling	**70.44**

Table 2: Test AUC for my best ensembles. The last row represents my selected submission. "T" designates models trained on the training set, and "T+V" models trained on the concatenation of training and validation sets. "T and T+V" signifies that the ensemble contains models trained only on the training set, and models trained on training+validation.

Acknowledgments

I want to thank the organizers of the MIND Challenge for a great competition in a new research area, as well as the authors of the MIND paper for creating this dataset and describing thoroughly multiple experimental setups.

References

[1] Das, A. S., Datar, M., Garg, A. & Rajaram, S. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, 271–280 (2007).

[2] Wu, F. *et al.* Mind: A large-scale dataset for news recommendation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 3597–3606 (2020).

[3] Wu, C. *et al.* Neural news recommendation with multi-head self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 6390–6395 (2019).

[4] Wu, C. *et al.* Neural news recommendation with attentive multi-view learning. *arXiv preprint arXiv:1907.05576* (2019).

[5] An, M. *et al.* Neural news recommendation with long-and short-term user representations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 336–345 (2019).

[6] Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[7] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).

[8] Chen, T., He, T., Benesty, M., Khotilovich, V. & Tang, Y. Xgboost: extreme gradient boosting. *R package version 0.4-2* 1–4 (2015).

[9] Volkovs, M. *et al.* Two-stage model for automatic playlist continuation at scale. In *Proceedings of the ACM Recommender Systems Challenge 2018*, 1–6 (2018).

[10] Chollet, F. *et al.* Keras: The python deep learning library. *ascl* ascl–1806 (2018).

[11] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).