Enhancing News Recommendation with Transformers and Ensemble Learning

Kazuki Fujikawa
DeNA Co., Ltd.
Tokyo, Japan
kazuki.fujikawa@dena.com

Naoki Murakami
DeNA Co., Ltd.
Tokyo, Japan
naoki.a.murakami@dena.com

Yuki Sugawara
DeNA Co., Ltd.
Tokyo, Japan
yuki.a.sugawara@dena.com

Abstract

News recommendation is an important task in the digital media landscape, challenged by the rapid decline of article relevance and the need for personalized content delivery. The RecSys Challenge 2024, organized by Ekstra Bladet, focuses on this problem using the EB-NeRD dataset. This paper presents the solution developed by team ":D", which secured the first place in the challenge. Our approach combines Transformers, Gradient Boosting Decision Trees (GBDT), and ensemble techniques in a three-stage recommendation pipeline. We introduce time-aware feature engineering methods and effective data-splitting strategies to address the temporal nature of news articles and improve model generalization. Through extensive experiments and ablation studies, we evaluated our system's performance, achieving an Area Under the ROC Curve (AUC) of 0.8924. Our analysis also examines the effects of data leakage, offering considerations for the practical implementation of news recommendation systems.

CCS Concepts

• **Information systems** → **Recommender systems**; **Personalization**.

Keywords

News Recommendation, RecSys Challenge, Transformer, Gradient Boosting Decision Tree

ACM Reference Format:
Kazuki Fujikawa, Naoki Murakami, and Yuki Sugawara. 2024. Enhancing News Recommendation with Transformers and Ensemble Learning. In *ACM RecSys Challenge 2024 (RecSys Challenge '24), October 14–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3687151.3687160

1 Introduction

Online news recommendation systems have become crucial technology in the age of information overload, significantly influencing user information access and consumption behavior. These systems aim to improve user engagement by understanding user interests and providing the most relevant news in a timely manner. However, there exist various technical challenges, including modeling user preferences based on behavior, addressing the cold start problem, and considering the time-decaying nature of news articles [5, 11, 15].

To address these challenges, RecSys Challenge 2024 established a news article recommendation task using EB-NeRD (Ekstra Bladet News Recommendation Dataset), a large-scale news recommendation dataset provided by the Danish news publisher Ekstra Bladet[1]. This dataset includes user click history, session information, personal attributes, and detailed information about news articles. The objective of this challenge is to predict the article that a user is most likely to click on in a specific impression, using the user's past click history, session information, personal attributes, and a list of candidate news articles.

The main goal of our research is to develop a high-accuracy recommendation system that considers the time-dependency of news articles. To achieve this goal, we made the following key contributions:

(1) Proposal of an effective data splitting strategy considering time-dependency and feature engineering techniques to suppress overfitting.
(2) Construction of a high-performance ensemble model combining Transformer-based models and Gradient Boosting Decision Trees (GBDT).
(3) Quantitative analysis of the impact of data leakage on model performance.

Our code is publicly available on github.com[2].

2 Data Splitting And Feature Engineering

In news recommendation systems, the rapid decline in article value over time significantly impacts the model's generalization performance. This section explains our approach to addressing this issue through data-splitting strategies and feature engineering. We also describe feature extraction methods that include data leaks reported in the official forum [3].

2.1 Data Splitting

Our research focused on applying effective data splitting strategies to mitigate the impact of article value decay over time, aiming to build a recommendation system that is resilient to temporal effects. Specifically, we designed three splitting patterns to effectively utilize the most recent data close to the test period (June 1-7, 2023):

- **Standard Split (S1)**: Using May 18-24, 2023, and May 25-31, 2023, as training and validation data, respectively.

RecSys Challenge '24, October 14–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-1127-5/24/10
https://doi.org/10.1145/3687151.3687160

[1] https://recsys.eb.dk/dataset/
[2] https://github.com/k-fujikawa/recsys-challenge-2024-1st-place
[3] https://www.codabench.org/forums/2387/342/

- **Retrain using validation data (S2)**: Using the 7 days from May 25-31, 2023, as training data.
- **Retrain using full data (S3)**: Using the 14 days from May 18-31, 2023, as training data.

A key feature of our approach, similar to that of Schifferer et al. [12], is the separation of hyperparameter optimization and final model training. We first optimize hyperparameters using S1, then retrain the model using S2 or S3. This approach maintains hyperparameter stability while adapting to recent trends. We hypothesize that data closer to the test period capture short-term trends more effectively, while larger datasets enhance stability. Consequently, we expect S2 and S3 to outperform S1.

2.2 Feature Engineering

Preliminary experiments revealed that direct use of article IDs or textual information led to overfitting, suggesting a rapidly evolving relationship between articles and users over time. To address this problem and mitigate the impact of temporal decay on the relevance of the article, we prioritize dynamic features that evolve with user behavior and temporal factors in our feature extraction process. To address this issue, we primarily adopted the following features:

- Used time-based features (e.g., time elapsed from article publication to impression)
- Incorporated user interaction features (e.g., time since user's most recent click)
- Employed similarity features between candidate articles and user's click history, using vector representations (e.g., TF-IDF, embeddings) instead of raw content

These features help capture user preferences and article relevance while reducing the risk of overfitting to specific article identifiers or temporary popularity. This enhances the model's ability to generalize across different time periods and user segments. For a more detailed overview of key features used in our LightGBM and Transformer-based models, including descriptions and potential data leak types, refer to the Appendix A.

2.3 Feature Engineering with Data Leakage

To investigate the impact of data leakage, we designed features that use data leaks to improve the accuracy of model prediction. Although these features present operational challenges, they demonstrated high efficacy in capturing the popularity and trends of the articles. Features using data leakages are classified into three types:

- **Future Article Statistics (L1)**: Statistics included in the article dataset (e.g., total_inviews) that encompass behaviors from the test period. These features function as significant indicators of an article's general popularity, reflecting long-term trends.
- **Future Impressions (L2)**: Statistics including impressions that appear after the prediction time in the test period (e.g., total inviews of all users 5 minutes before and after the prediction time). These features were effective in capturing the short-term trends of the articles.
- **Impressions Prior to Prediction Time (L3)**: Statistics including impressions appearing prior to the prediction time (e.g., total inviews of all users for 10 minutes up to 1 minute before the prediction time) are not future data leaks. However, we define these as a type of leak because they use the test set and would require a real-time data processing infrastructure to extract features too close to the prediction time. These features, as well as L2, were highly effective in capturing short-term trends of articles.

3 Models

Figure 1 shows an overview of the solution of our team. This section explains the key components: Transformer-based models, GBDT-based models, and ensemble methods.

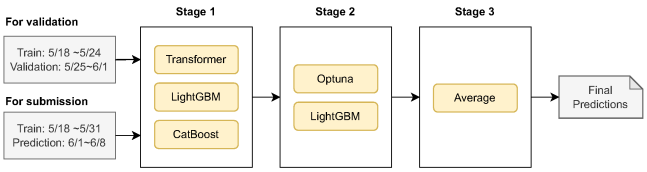

Figure 1: Overview of the solution

Our approach utilizes a three-stage ensemble strategy to predict the click-through rates of news articles. Stage 1 processes raw features using the Transformer, LightGBM, and CatBoost models. Stages 2 and 3 process outputs from previous stages using Optuna, LightGBM, and simple averaging. This structure leverages diverse model strengths while maintaining separate pipelines for validation and submission.

3.1 Transformer-Based Model

In news article recommendation tasks, it is important to capture the relationships among various elements such as impression background information, concurrently displayed article information, and users' past click histories. These interactions can be effectively modeled by adopting the Transformer [13] architecture. Related works [4, 9, 14] have proposed methods using Transformers to model users' past behaviors and estimate the click probability of a single candidate. In contrast, we adopted an approach similar to RankFormer [3], leveraging known candidate sets to simultaneously estimate click probabilities for multiple articles. Figure 2 shows the model architecture based on this approach.

Our model utilizes two types of input embeddings: impression and inview embeddings, as detailed in Figure 3. Impression embedding represents session-wide context information in a single node, while inview embeddings represent features of simultaneously displayed articles. We use sinusoidal embedding [13] for quantitative features and learnable embeddings for categorical features, as described in Section 2.2 and 2.3. Inview embeddings additionally incorporate similarity embeddings, computed as the cosine similarity between learnable embeddings of inview articles' topics/categories and those from the user's click history. Treated as a variable-length sequence, inview embeddings learn article importance through interaction with impression embedding in the Transformer's self-attention mechanism.

The EB-NeRD training dataset (May 18-24, 2023) contains approximately 12 million impressions from about 780,000 users, which presents a challenge in terms of user diversity. To address this, we

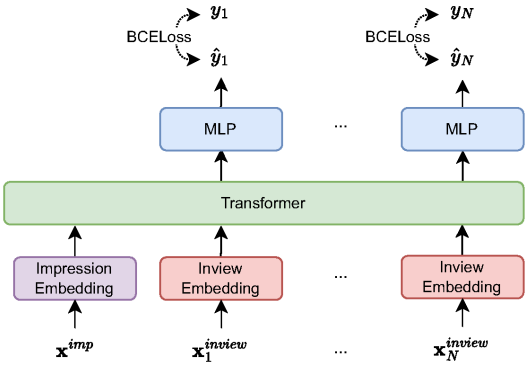

Figure 2: Transformer-based model architecture

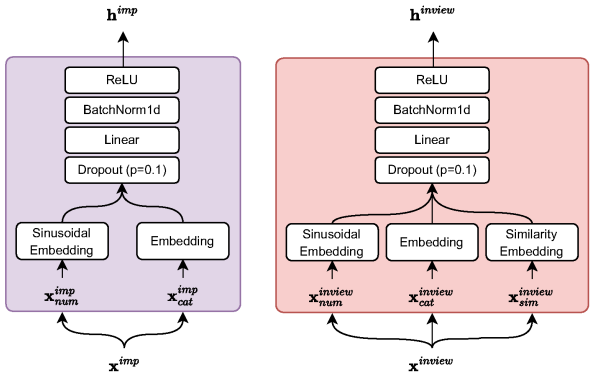

Figure 3: Details of impression and inview embeddings

implemented a dynamic sampling strategy, selecting 3 impressions per user for each epoch, which effectively utilizes approximately 25% of the total impressions. This approach balances dataset representativeness, diversity, and computational efficiency.

Our Transformer block consists of an 8-layer, 128-dimensional, 4-head Pre-LN Transformer [16]. We use binary cross-entropy loss for training as a click classification problem. The AdamW optimizer [8] is used with an initial learning rate of 1e-3 and a cosine decay schedule. Experiments are done on the g2-standard-32 instance from the Google Cloud with 1 NVIDIA L4 GPU.

3.2 GBDT-Based Model

In this study, we used two implementations of Gradient Boosting Decision Trees (GBDT) [6]: LightGBM [7] and CatBoost [10], known for their efficiency, accuracy, and interpretability. LightGBM excels at processing large-scale datasets through parallel learning, while CatBoost minimizes information loss in categorical data handling. Combining these models aims to enhance performance through ensemble methods while maintaining low implementation costs.

For model training, we randomly sample 20% of user impression data, balancing data diversity and computational efficiency. We frame the problem as a ranking task, predicting clicked articles among candidates. We utilize LambdaRank [2], a pairwise objective, to optimize article ranking.

Model performance is optimized by tuning key hyperparameters, including the number of boosting rounds, the maximum depth of the tree, the number of leaves, the feature fraction, the bagging fraction, and the regularization parameters. Specifically, for the LightGBM model, we set the number of boosting rounds to 400, applied early stopping with a patience of 40 rounds, and used a learning rate of 0.1. For the CatBoost model, we set the number of iterations to 1000, applied early stopping with a patience of 40 rounds, and used a learning rate of 0.1. We determine optimal combinations through fixed learning rate experiments. We adopted early stopping to prevent overfitting and used ndcg@10 instead of AUC, the main competition evaluation metric, for model optimization from a computational efficiency perspective. The validity of this approach was confirmed by the strong correlation observed between these two metrics in both the validation and the test data. Further implementation details can be found in the accompanying code repository.

3.3 Ensemble Multiple Models

To further improve prediction accuracy, we adopted ensemble methods that integrate prediction scores from multiple models. As Figure 1 shows, our process involves model integration in stage 2 and final prediction in stage 3.

Stage 2 uses two integration approaches: optimized weighted averaging and stacking. Optimized weighted averaging uses Optuna [1] to determine the optimal weights of the prediction score, maximizing the AUC in the validation set. The stacking approach treats stage 1 prediction scores as new features, engineering:

- Raw and relative values of prediction scores from each model
- Relative ranking of predicted articles within impressions and users

We perform a second learning stage using LightGBM with these features, employing LambdaRank to extract complex patterns through a non-linear combination of model predictions. In stage 3, we generate the final prediction score by averaging the optimized weighted averaging and stacking scores from stage 2. This method takes advantage of the strengths of both approaches without undue complexity.

4 Experiments

This section describes the experimental design and results to verify the effectiveness of our proposed method. We first compare the performance of the model across different data-splitting methods, including the impact of data leakage. Subsequently, we conducted an ablation study to assess the importance of each component of the model, followed by an evaluation of the effects of data leakage on model performance.

4.1 Results

Table 1 shows the performance of each model in different data splitting methods as described in Section 2.1. The validation set is used only in the Standard Split (S1), while for S2 and S3, we applied the hyperparameters determined from S1 without further validation. All test scores are based on the period on June 1-7, 2023. As hypothesized, we observe performance improvements from S1 to

S2 to S3 in most models. This confirms the effectiveness of our data-splitting strategy, which focuses on retraining using a validation set.

Among single models, the Transformer-based model shows the highest score. We further improved performance through pseudo-labeling (PL), a semi-supervised learning technique. We generated pseudo-labels for the unlabeled test set by computing a weighted average of predictions from our single models (Transformer, LightGBM, and CatBoost). These pseudo-labels augmented our training data, allowing models to learn from a larger dataset. Our experiments confirm that ensemble methods combining these models yield additional performance gains. These results demonstrate the effectiveness of Transformer-based models, pseudo-labeling, and ensemble methods combining diverse architectures.

Table 1: Comparison of AUC scores by data splitting method and model. S1: Standard Split, S2: Retrain using validation data, S3: Retrain using full data. PL: Pseudo-labeling.

Model	Validation	Test (S1)	Test (S2)	Test (S3)
Transformer	0.8734	0.8764	0.8824	0.8864
LightGBM	0.8668	0.8694	0.8800	0.8817
CatBoost	0.8652	0.8691	0.8787	0.8805
Transformer (+PL)	0.8738	0.8761	0.8883	0.8872
Ensemble	0.8802	0.8827	0.8911	0.8924

4.2 Ablation Study for Transformer-based Model

We conducted an ablation study to verify the effectiveness of each component in the Transformer-based model. Table 2 shows the changes in the AUC scores when key elements are excluded or modified. The "w/o Transformer" configuration replaces the Transformer architecture with a Multi-Layer Perceptron (MLP) of equivalent depth while maintaining all other components of the model.

Table 2: Ablation study results for Transformer-based model configurations

Model configuration	Validation	Test (S3)
Full model	0.8734	0.8864
w/o impression features	0.8679	0.8835
w/o Transformer (replaced by MLP)	0.8280	0.8435

Excluding impression features reduced the AUC to 0.8835. This indicates that session context information, such as user browsing time and device information, contributes to improving prediction accuracy. Additionally, replacing the Transformer with an MLP with the same number of layers significantly decreased the AUC (0.8422). This suggests that the Transformer architecture effectively captures complex interactions between features in the news recommendation task. These results confirm that each component of the proposed Transformer-based model contributes to performance improvement.

4.3 Evaluation of the Impact of Data Leakage

To evaluate the impact of features using data leakage described in Section 2.3 on model performance, we performed comparative experiments using the Transformer-based model. The results are shown in Table 3.

Table 3: Impact of data leakage on Transformer-based model performance

Data usage	L1	L2	L3	Validation	Test (S3)
Full features	✓	✓	✓	0.8734	0.8864
w/o future impressions	-	-	✓	0.8495	0.868
w/o test impressions	-	-	-	0.7483	0.7699

A significant performance difference was observed between "Full features" and "w/o test impressions", with a difference of 0.1165 points in AUC score on the test data. This result strongly suggests the importance of careful data handling in recommendation systems. On the other hand, the "w/o future impressions" setting, which includes L3 (impression information from the test set more than 1 minute before the prediction time), showed limited performance degradation compared to "Full features". This confirms the importance of using short-term trends in articles as features.

5 Conclusion

This paper presented the solution by team ":D" for the news recommendation task in RecSys Challenge 2024. We introduced time-aware feature engineering methods and effective data-splitting strategies to address the temporal nature of news articles and improve model generalization. This approach achieved a maximum AUC score of 0.8924, marking a significant improvement over the baseline models. Furthermore, our analysis of data leakage effects underscored the critical role of capturing short-term trends in articles to ensure the accuracy of recommendations.

Future work includes further performance improvement when excluding data leakage and building flexible recommendation models that also estimate the set of candidate articles itself. These advances aim to further enhance the robustness and applicability of news recommendation systems in real-world settings.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019).

[2] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems* 19 (2006).

[3] Maarten Buyl, Paul Missault, and Pierre-Antoine Sondag. 2023. RankFormer: Listwise Learning-to-Rank Using Listwide Labels. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2023), 3762–3773.

[4] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. 2021. Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation. *Proceedings of the 15th ACM Conference on Recommender Systems* (2021), 143–153.

[5] Chong Feng, Muzammil Khan, Arif Ur Rahman, and Arshad Ahmad. 2020. News recommendation systems-accomplishments, challenges & future directions. *IEEE Access* 8 (2020), 16702–16725.

[6] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[7] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).
[8] Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.
[9] Yichao Lu and Maksims Volkovs. 2023. Robust User Engagement Modeling With Transformers and Self Supervision. *Proceedings of the Recommender Systems Challenge 2023* (2023), 23–27.
[10] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. *Advances in neural information processing systems* 31 (2018).
[11] Shaina Raza and Chen Ding. 2022. News recommender system: a review of recent progress, challenges, and opportunities. *Artificial Intelligence Review* (2022), 1–52.
[12] Benedikt Schifferer, Jiwei Liu, Sara Rabhi, Gilberto Titericz, Chris Deotte, Gabriel De Souza P. Moreira, Ronay Ak, and Kazuki Onodera. 2022. A Diverse Models Ensemble for Fashion Session-Based Recommendation. *Proceedings of the Recommender Systems Challenge 2022* (2022), 10–17.
[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[14] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural News Recommendation with Multi-Head Self-Attention. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (2019), 6389–6394.
[15] Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, et al. 2020. Mind: A large-scale dataset for news recommendation. In *Proceedings of the 58th annual meeting of the association for computational linguistics*. 3597–3606.
[16] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*. PMLR, 10524–10533.

A Feature details

This section provides detailed information on the features used in our LightGBM and Transformer-based models, complementing the feature engineering discussion in Section 2.2.

Table 4 lists the top 10 most important features identified by our LightGBM model, together with their descriptions and data leak types (L1, L2, L3) as defined in Section 2.3. The importance of these features, as determined by the LightGBM model, reflects the key factors contributing to our news recommendation system's effectiveness.

Table 5 presents an overview of the main feature categories used in our Transformer-based model. Although not exhaustive, this table provides a representative sample of the most significant feature types and their descriptions. It highlights the diverse range of information captured by our Transformer model, including temporal aspects, user behavior, and article characteristics.

Table 4: Description of top 10 important features in LightGBM model and their data leak types

#	Feature name	L1	L2	L3	Description
1	c_time_hour_diff				Time difference in hours between the impression time and the publication time of the candidate article.
2	c_common_read_time_sum_past_and_future_5m_rank_ascending		✓	✓	Rank of the candidate article based on the sum of (read_time / number of candidate articles) for impressions 5 minutes before and after, sorted in ascending order.
3	c_time_min_diff				Time difference in minutes between the impression time and the publication time of the candidate article.
4	c_user_count_past_all_rank_descending			✓	Rank of the candidate article based on the sum of (read_time / number of candidate articles) for all past impressions viewed by the recommended user, sorted in descending order.
5	c_common_read_time_sum_future_5m_rank_ascending		✓		Rank of the candidate article based on the sum of (read_time / number of candidate articles) for impressions in the next 5 minutes, sorted in ascending order.
6	c_common_read_time_sum_future_1h_rank_ascending		✓		Rank of the candidate article based on the sum of (read_time / number of candidate articles) for impressions in the next hour, sorted in ascending order.
7	a_inviews_per_pageviews	✓			Ratio of total inviews to total pageviews for the candidate article.
8	c_time_sec_diff				Time difference in seconds between the impression time and the publication time of the candidate article.
9	c_topics_count_svd_sim				Cosine similarity between the average vector of articles previously clicked by the user and the vector of the candidate article, using CountVectorizer and SVD on topics.
10	c_common_read_time_sum_past_5m_rank_ascending			✓	Rank of the candidate article based on the sum of (read_time / number of candidate articles) for impressions in the past 5 minutes, sorted in ascending order.

Table 5: Description of features used in Transformer-based model

#	Feature type	Feature name	L1	L2	L3	Description
1	Impression	elapsed_ts_from_history				Time-related features (click history, most recent impression)
		elapsed_ts_from_future		✓		
		elapsed_ts_from_past			✓	
2	Impression	read_time				Access information included in the impression
		scroll_percentage				
		device_type				
3	Impression	num_history_articles				Click history, number of future and past impressions for the same user
		num_future_articles		✓		
		num_past_articles			✓	
4	Inview	article_sentiment_score				Article features (sentiment score, time since publication, rank within inview)
		article_published_ts_diff				
		inview_article_ts_ranks_desc				
5	Inview	history_article_log_counts				Number of articles, rank, and log rank in click history
		history_article_ranks				
		history_article_log_ranks				
6	Inview	history_matched_topics_normed_counts_rank				Rank of topic/category overlap count with articles in click history, embedding similarity
		history_matched_category_normed_counts_rank				
		article_history_topics_embedding_similarity				
		article_history_category_embedding_similarity				
7	Inview	global_article_-5m_5m_normed_counts		✓	✓	Article statistics for specific time windows across the entire dataset (e.g., frequency, total read time, rank within inview)
		global_article_-5m_5m_counts_rank		✓	✓	
		global_article_-5m_5m_readtime_sum_rank		✓	✓	
		global_article_-5m_5m_readtime_mean		✓	✓	
8	Inview	article_total_inviews	✓			Article statistics (inviews, pageviews, read_time)
		article_total_pageviews	✓			
		article_total_read_time	✓			