# GPU Accelerated Feature Engineering and Training for Recommender Systems

BENEDIKT SCHIFFERER[*], NVIDIA, United States
GILBERTO TITERICZ[*], NVIDIA, Brazil
CHRIS DEOTTE[*], NVIDIA, United States
CHRISTOF HENKEL[*], NVIDIA, Germany
KAZUKI ONODERA[*], NVIDIA, Japan
JIWEI LIU[*], NVIDIA, United States
BOJAN TUNGUZ[*], NVIDIA, United States
EVEN OLDRIDGE[*†], NVIDIA, Canada
GABRIEL DE SOUZA PEREIRA MOREIRA[*], NVIDIA, Brazil
AHMET ERDEM[*], NVIDIA, Netherlands

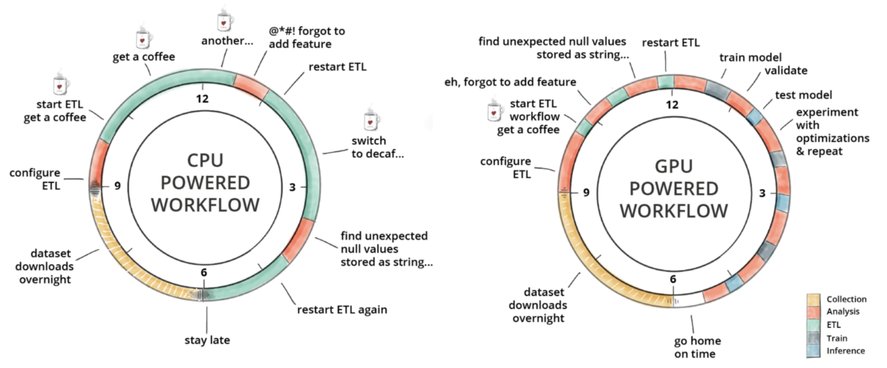
get a coffee
another...
@*#! forgot to add feature
restart ETL
start ETL get a coffee
12
switch to decaf...
configure ETL
9
CPU POWERED WORKFLOW
3
find unexpected null values stored as string...
dataset downloads overnight
6
restart ETL again
stay late
find unexpected null values stored as string...
restart ETL
train model
validate
eh, forgot to add feature
test model
start ETL workflow get a coffee
experiment with optimizations & repeat
configure ETL
GPU POWERED WORKFLOW
dataset downloads overnight
go home on time
Collection
Analysis
ETL
Train
Inference

Fig. 1. Workflow without GPU acceleration (left) and with GPU acceleration (right). Showing the reduced Extract-Load-Transform (ETL) time with GPU acceleration. [7].

In this paper we present our 1st place solution of the RecSys Challenge 2020 which focused on the prediction of user behavior, specifically the interaction with content, on this year's dataset from competition host Twitter. Our approach achieved the highest score in seven of the eight metrics used to calculate the final leaderboard position. The 200 million tweet dataset required significant computation to do feature engineering and prepare the dataset for modelling, and our winning solution leveraged several key tools in

[*]Authors contributed equally to this research.
[†]Corresponding Author

order to accelerate our training pipeline. Within the paper we describe our exploratory data analysis (EDA) and training, the final features and models used, and the acceleration of the pipeline. Our final implementation runs entirely on GPU including feature engineering, preprocessing, and training the models. From our initial single threaded efforts in Pandas which took over ten hours we were able to accelerate feature engineering, preprocessing and training to two minutes and eighteen seconds, an end to end speedup of over 280x, using a combination of RAPIDS cuDF, Dask, UCX and XGBoost on a single machine with four NVIDIA V100 GPUs. Even when compared to heavily optimized code written later using Dask and Pandas on a 20 core CPU, our solution was still 25x faster. The acceleration of our pipeline was critical in our ability to quickly perform EDA which led to the discovery of a range of effective features used in the final solution, which is provided as open source [16].

CCS Concepts: • **Information systems** → **Recommender systems**; Content analysis and feature selection; • **Computer systems organization** → Single instruction, multiple data.

Additional Key Words and Phrases: Recommender Systems, RecSys Challenge, Boosting, Feature Engineering, GPU Acceleration

**ACM Reference Format:**
Benedikt Schifferer, Gilberto Titericz, Chris Deotte, Christof Henkel, Kazuki Onodera, Jiwei Liu, Bojan Tunguz, Even Oldridge, Gabriel De Souza Pereira Moreira, and Ahmet Erdem. 2020. GPU Accelerated Feature Engineering and Training for Recommender Systems. In *Proceedings of the Recommender Systems Challenge 2020 (RecSysChallenge '20), September 26, 2020, Virtual Event, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3415959.3415996

## 1 INTRODUCTION

Recommender systems play a critical role in driving user engagement and revenue on online platforms. The data structure is often tabular, and requires extensive feature engineering to find features that provide the best performance when modeling. Although deep learning architectures are able to automatically extract good data representations in other domains such as images and text, they still depend on expert knowledge and feature engineering in tabular data. Top solutions in previous RecSys challenges [10] [18] [19] depended on custom features, which were fed into boosted decision trees or neural networks.

For the RecSys 2020 Challenge, Twitter provided a dataset [3] containing 200 million user-tweet combinations, requiring participants to predict the user's behavior given a user-tweet combination: Does the user like, reply, retweet and/or retweet with comment the tweet? The Relative Cross-Entrop (RCE) and Precision-Recall Area Under the Curve (PRAUC) metrics of each behaviour type are averaged and the sum of these two ranks provides the final ranking score.

Our success in this challenge was heavily dependent on our GPU accelerated data pipelines, which enabled quick iteration when trying features and performing hyperparameter tuning. We accelerated our initial feature engineering pipeline from nine hours down to 57 seconds through a combination of RAPIDS cuDF [5], Dask [6] and UCX [15]. In addition, we accelerated training of four models in our final solution from two hours to 56 seconds using 4x NVIDIA Tesla V100 GPUs[1]. This design enabled us to create and explore hundreds of different features and to run thousands of experiments over the course of the competition. Figure 1 illustrates a typical workflow without and with GPU acceleration.

In Section 2 we describe our EDA process including the feature engineering techniques and highlight some considerations we discovered regarding this dataset. Section 3 covers our final model. We provide a comprehensive overview about alternative approaches, we tried, in Section 4. Finally, in Section 5 we demonstrate how to accelerate the pipeline, providing a comparison of several different methods.

[1]For our CPU training baseline we used LightGBM, as it is much faster than XGBoost [4] on the CPU. The GPU benchmark uses XGBoost.

## 2 DATA PROCESSING

The competition dataset from the RecSys Challenge 2020 has a tabular structure, requiring feature engineering to make information more accessible to the models. In our experiments, building features improved our final score significantly, whereas model design played a secondary role. In addition to the 20 provided raw features in the original data, we have designed and experimented with an additional 200 features, using 120 in our final models. Accelerating our processing pipeline with GPUs enabled us to iterate quickly and to try out many experiments.

bắt đc

**Dataset split**: We tried multiple approaches to create a validation set that mimics the test set, to be able to have a high correlation of the accuracy of experiment results on our validation set and test set. In the end, we had two different validation approaches: (1) separating the validation set as the last two days of the train set (based on Tweet's published date) and (2) splitting train/validation sets (50%/50%). In both cases, grouped by tweetid in order to have all instances related to the same tweet in each fold. Even with these approaches, our results for validation set and test set were not perfectly correlated, forcing us to tune some hyperparameters based on public leaderboard scores for the test set.

**Adversarial Validation**: As part of the feature selection and exploration process, we used adversarial validation [1] [2] to build a model that tries to predict whether a given data point belongs to the train or the test set. The idea is to analyze (1) if training and test sets have different distribution and (2) how to split a validation set. Feature importance from the adversarial model can then be used to give insight into which features differ the most. The validation model that we built had an AUC of almost 1.0, which suggested a very high level of difference between the two sets.

### 2.1 Feature engineering and EDA

This section describes the different main methods for feature engineering. A list of top 15 feature per target based on importance score is provided in Appendix C.

**Target Encoding**: Target Encoding (TE) calculates statistics from a target variable segmented by the unique values of one or more categorical features. For this competition we encoded the four targets as the conditional probability of the interaction $I$ given a categorical feature $C$: $P(I|C)$. On the training dataset, we used a 5-fold strategy to generate TE features and avoid overfitting. In addition, we smoothed the TE features to avoid overfitting for low frequency categories as shown in Equation 1,

$$TE_{target}([Categories]) = \frac{count([Categories]) * mean_{target}([Categories]) + w_{smoothing} * mean_{target}}{count([Categories]) + w_{smoothing}} \tag{1}$$

Where $count(\cdot)$ is the number of observations for that category and $w_{smoothing}$ is a smoothing hyperparameter, which we set to 20. The $[Categories]$ is a list of categorical features, whose values are tuples with combinations of those features that occur in the dataset. For example, we used average target encoding for the target variable *like* given the categorical feature *engaging_user*. An *engaging_user* who occurs frequently has its TE value close to their mean while users with a few observations have their TE values close to the global mean of the target.

**Tweet Features**: In addition to the supplied tweet features we developed several custom features related to its content including the number of words, characters and occurrences of the @ character. In addition, we carefully engineered a categorical feature from a hash of the first username mentioned in a tweet that identifies whether an engaging userid was regularly mentioning another (hashed) username, indicating a relationship. Finally, we extracted the two most

popular words per tweet and two least popular words based on word frequency in tweets. We considered only words, which had at least 2 and at most 100000 occurrences to filter out irrelevant or filling words.
**Time Series Features**: We used the order of the tweets to calculate multiple time-dependent features, such as the number of engagements of the engaging users in the last 5, 60, 240, 480 and 1440 minutes, the time past since an engaging user posted a tweet or since he/she interacted with another tweet.
**Graph Feature**: The dataset contains information regarding whether the engaging user (A) follows the engaged user (B). Based on this, we extract from the follower graph: Does A follow B (2nd degree)? Does B follow A (1st & 2nd degree)? This graph can only be partially constructed, as the data does not contain follower information for all users.

## 2.2 Dataset considerations

Throughout our experiments, we faced several difficulties with the dataset. The windowing of the dataset on both user and time prevented us from engineering features we initially wished to explore. For example we were unable to extract if the engaging user was mentioned in the tweet itself or extract a full follower network graph, which could lead to more interesting deep learning models such as Graph Neural Networks. The short period covered by the train dataset (1 week) also prevented us from building a good validation dataset. Furthermore, the low number of interactions per user (Median of 2 and 90th percentile of 8) and per item (median of 1 and 90th percentile of 3) prevented participants from modeling users preferences and tweets embeddings meaningfully in order to provide personalized recommendations.

The dataset creation process also introduced some data leakage, likely resulting from negative samples which were sampled from a different distribution. Comparing the frequency of engaged and engaging users, we observed that it is possible to separate positive and negative samples based on that feature alone. Data collection also stopped at midnight on the last day meaning that user interaction probability continuously decreased during the last 4 hours in training and test datasets. Even if a model would correctly predict that a user would engage with a tweet, the missing data could penalize correct predictions in that time frame.

Sampling interactions to create a recommendation dataset is a challenging task, particularly at the scale of competition host Twitter. Understandably a subset of users, a narrow time window, and a selection of negative samples had to be chosen by the competition host in such a way, that participants can explore a range of solutions without requiring hardware on the scale of what is used in production. Still, for future competitions it would be interesting to have a dataset covering a longer period of time, so that we could model long-term users preferences, and learn how short term users preferences change depending on seasonal patterns, trending topics/hashtags or breaking news. This wider time window would also provide more positive samples per user, allowing more complex solutions that do better modeling preference. Striking that balance in a competition meant to be accessible to both academia and industry is difficult.

## 3 OUR FINAL SOLUTION

For the RecSys Challenge 2020, we experimented with Gradient Boosted Decision Trees, ALS matrix factorization and several deep learning based models on the features described above. Our best single model was XGBoost [4], and our final model was a blend of three independent XGBoost models trained on different features sets and on a different subset of the training set. While two models were validated using the last two days of training data, the third was validated using two folds with no tweets in common. The hyperparameters were tuned using a simple grid search algorithm that was possible due the time optimized algorithms used. The metrics used in the competition were the PRAUC and the RCE. There are some flaws regarding PRAUC metric, like predicting all zeros scores high, so we decided to use only RCE metric to track local and leaderboard performances.

$$PRAUC = AreaUnderCurve(precision, recall) \tag{2}$$

$$RCE = 100 * (1 - \frac{log_{loss}(y_{true}, y_{pred})}{log_{loss}(y_{true}, mean(y_{true}))}) \tag{3}$$

The features used in each model were built using some categorical encoding techniques due the number of high cardinality features in the dataset. Ordinal Encoding, Frequency or Count Encodings, Hash Encoding and Multi-feature Target Encoding were used. By far Target Encoding was the best feature engineering technique and was responsible for the biggest scores improvements. Table 1 (Appendix A) shows the local validation and public leaderboard (LB) scores of each model. In Appendix C, we provide the feature importance table for XGBoost 3. The weights used to ensemble the models and generate the final predictions were tuned using public leaderboard feedback per target, according to Table 2 (Appendix B). Our final weighted averaged model achieved a score of 132.42 on the public leaderboard and 132.55 on the private.

## 4 ALTERNATIVE METHODS EXPLORED

**Tree-based**: LightGBM [11] and CatBoost [12] were explored but ultimately abandoned because of its similar performance to XGBoost. Additionally, CatBoost and LightGBM on CPU had slow runtime in comparison to XGBoost on GPU.

bản chất

**Follow the Regularized Leader**: Follow the Regularized Leader (FTRL-Proximal) [9] is an online linear algorithm that is relatively fast and performs well for high cardinality variables. The total run time from scratch is approximately 3h using four cores (1 core per target). The speed is mostly limited by the hashing trick of each variable to fit the linear coefficients space. The performance is not comparable to more sophisticated algorithms like Gradient Boosting Trees and Neural Networks, but it is possible to achieve an overall score of 42.5 RCE.

**Alternating Least Squares**: For a quick experimentation, we tried to learn the entities embeddings using the Alternating Least Square (ALS) [17] matrix factorization, a traditional and scalable collaborative filtering technique for implicit feedback data. We inspected the accuracy of the ALS predictions (dot product between user and item embeddings) for one of the folds, segmented by the number of positive samples available for the engaging user in the other folds. As shown in Figure 2, the more positive samples of the user that are available for training the better the PRAUC of the model as expected.

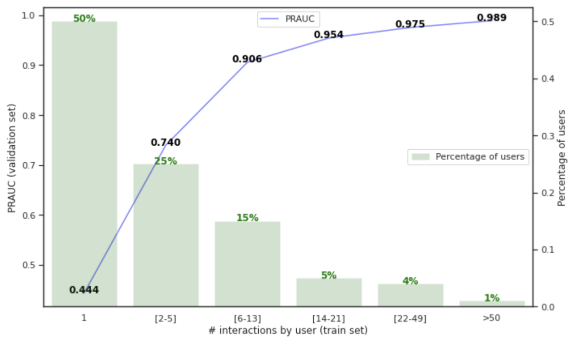

Fig. 2. Positive interactions per user and accuracy of predictions for those users using ALS.

As the item embeddings could not be reused for test set predictions (because there are no tweets in common between train and test sets), we have included hashtags and links as *artificial items*. Then, we generated features set by multiplying user embeddings with hashtags and links embeddings. ALS features improved PRAUC by 1.3 for the like target for validation, however the improvement was lower on the test set. Based on this result, we didn't try GNNs, although on a more dense dataset we would expect them to be effective.

**BERT**: Since the dataset contained a pre-calculated tokenization of tweet text using a BERT multi-language we extracted the mean embedding for each tweet using the pretrained DistilBERT transformer[8] [14] producing a 768 dimensional representation for each tweet text. In order to balance the feature dimension with our other features and to improve manageability for storage and transfer we further reduced the dimension by averaging every 24 logits. This resulted in a final 32 dimensional representation for each tweet. Adding this 32 dimensional feature to our Gradient Boosting model unfortunately did not yield an improvement.

As a second approach, we fine-tuned pre-trained DistilBERT towards the 4 targets without using any additional features. After training on 1% of the training data for 1 epoch, we achieved an overall score of roughly 10 RCE. In order to determine the benefit of combining this computationally intensive information with our other features we extracted out-of fold predictions for the complete training set resulting in a four dimensional feature. Unfortunately adding those out-of-fold predictions to our best model did not yield improvement. We concluded that not enough additional signal is contained in the tweet text to justify the computational expensive usage of text tokens for this competition.

**Feed Forward Neural Network**: We also explored feeding the same XGBoost features into a Feed Forward Neural Network (FFNN). In contrast to tree-based models, neural networks are required to normalize continuous input features. We tried multiple normalization strategies, such as Min-Max scaling, standardization, power-transformation and GaussRank and found out that Min-Max scaling had the best performance. Also, due to the shallow two layers architecture used, it's very important that all the categorical features are transformed into probabilities using Target Encoding, that way all the features become numerical. Using the same set of features from XGBoost 3 model, the FFNN performed 60.7 RCE for the LIKE target in the validation set while XGBoost 3 achieved 62.6. This model presented a good performance, diversity and a fast training time in GPUs, however it was not used in the final ensemble due to the fact that it was developed near the end of the competition.

## 5 ACCELERATION

In order to discover the most accurate model, accelerating the process of experimentation is crucial. Experimentation consisted of feature engineering, training a model, inferring a validation dataset, and computing a metric score.

Before feature engineering, we must preprocess the original dataset. This step is only performed once and then used by all future experiments. In this step, we loaded the provided TSV files, imputed NaNs, factorized categorical variables, optimized variable data type, processed lists into columns, decoded BERT tokens into text, extracted text features like word counts, and computed a dozen of count features related to engaging and engaged user. Data preprocess took 10 hours on CPU to transform about 200 million tweet user interactions.

The majority of our features are Target Encoding, Count Encoding, and Lag (variable differences in time) features. After brainstorming an idea, we would compute the feature(s), then train, infer, and compute validation score. It was essential to speed up this cycle as fast as possible. Using RAPIDS cuDF, DASK, and UCX, enabled us to use four NVIDIA V100 GPUs for this task. Using these GPU accelerated libraries it takes less than one minute to engineer all features, while using Pandas [13] on CPU takes eight hours 56 minutes, a speed-up factor of over 500x. Even when compared to optimized CPU code (Pandas+Dask) on 20 cores, our solution provided a speedup of 42x. Figure 3 visualises the speed up, excluding Pandas as it is outside the range. The full benchmark is available in Appendix B, Table 3.

Using Dask-XGBoost enabled our models to be trained on four NVIDIA V100 GPUs, which sped up model training and inference significantly compared to LightGBM on CPU. Using this multi-GPU solution we sped up training by a factor of 120x. The competition metrics PRAUC and RCE required the calculation of AUC and log loss. We accelerated these calculations with GPU and Numba JIT, reducing computation from minutes to seconds. End to end our solution was 280x faster than our initial CPU based implementation and 28x faster than the optimized CPU code.

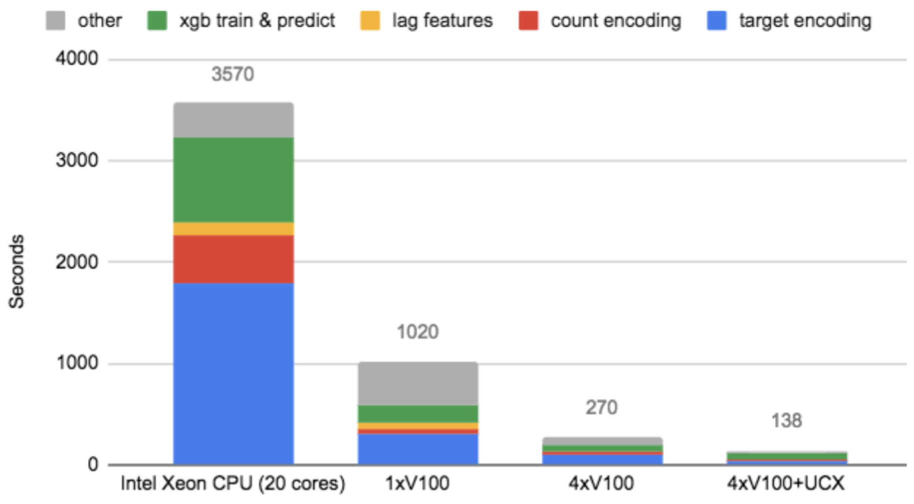

Fig. 3. Computation time in seconds for different infrastructure and libraries

## 6 CONCLUSION

In this paper we presented our 1st place solution of the RecSys Challenge 2020, which achieved the highest score in seven of the eight metrics used to calculate the final leaderboard position. Our final implementation runs entirely on GPU including feature engineering, preprocessing, and training the models for the 200M interactions in two minutes 18 seconds, a speedup of greater than two orders of magnitude over CPU. This was achieved using a combination of RAPIDS cuDF, Dask, UCX and XGBoost on a single machine with four NVIDIA V100 GPUs. We hope our final solution, which is provided as open source [16], is useful to others in the RecSys field who wish to build large scale recommender systems on GPUs.

## ACKNOWLEDGMENTS

The authors wish to thank our colleagues on the RecSys, KGMON and RAPIDS.AI teams for their support in this competition and in particular Joshua Patterson for his vision of a GPU accelerated data science workflow and Nicolas Koumchatzky for his guidance and recommender system expertise.

## REFERENCES

[1] adv1 2016. *Adversarial validation, part one.* Retrieved August 10th, 2020 from http://fastml.com/adversarial-validation-part-one/

[2] adv2 2018. *Adversarial validation, part one.* Retrieved August 10th, 2020 from https://www.kaggle.com/tunguz/elo-adversarial-validation

[3] Luca Belli, Sofia Ira Ktena, Alykhan Tejani, Alexandre Lung-Yut-Fon, Frank Portman, Xiao Zhu, Yuanpu Xie, Akshay Gupta, Michael Bronstein, Amra Delić, Gabriele Sottocornola, Walter Anelli, Nazareno Andrade, Jessie Smith, and Wenzhe Shi. 2020. Privacy-Preserving Recommender Systems Challenge on Twitter's Home Timeline. (2020).

[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[5] cudf 2020. *cuDF - GPU DataFrames.* Retrieved July 6th, 2020 from https://github.com/rapidsai/cudf

[6] dask 2020. *Dask: Scalable analytics in Python.* Retrieved July 6th, 2020 from https://dask.org/

[7] daydatascience 2018. *RAPIDS Accelerates Data Science End-to-End.* Retrieved July 6th, 2020 from https://medium.com/rapids-ai/rapids-accelerates-data-science-end-to-end-afda1973b65d

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[9] Chuong B. Do, Quoc V. Le, and Chuan-Sheng Foo. 2009. Proximal Regularization for Online and Batch Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada) *(ICML '09)*. Association for Computing Machinery, New York, NY, USA, 257–264. https://doi.org/10.1145/1553374.1553407

[10] Paweł Jankiewicz, Liudmyla Kyrashchuk, Paweł Sienkowski, and Magdalena Wójcik. 2019. Boosting Algorithms for a Session-Based, Context-Aware Recommender System in an Online Travel Domain. In *Proceedings of the Workshop on ACM Recommender Systems Challenge* (Copenhagen, Denmark) *(RecSys Challenge '19)*. Association for Computing Machinery, New York, NY, USA, Article 1, 5 pages. https://doi.org/10.1145/3359555.3359557

[11] 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. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[12] Liudmila Ostroumova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *NeurIPS*.

[13] pandas 2008. *pandas - Python Data Analysis Library.* Retrieved July 6th, 2020 from https://pandas.pydata.org/

[14] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv* abs/1910.01108 (2019).

[15] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. 2015. UCX: an open source framework for HPC network APIs and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 40–43.

[16] solution 2020. *NVIDIA RAPIDS.AI's final model.* Retrieved July 6th, 2020 from https://github.com/rapidsai/deeplearning/tree/master/RecSys2020

[17] Gábor Takács and Domonkos Tikk. 2012. Alternating least squares for personalized ranking. (09 2012). https://doi.org/10.1145/2365952.2365972

[18] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. 2018. Two-Stage Model for Automatic Playlist Continuation at Scale. In *Proceedings of the ACM Recommender Systems Challenge 2018* (Vancouver, BC, Canada) *(RecSys Challenge '18)*. Association for Computing Machinery, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/3267471.3267480

[19] Maksims Volkovs, Guang Wei Yu, and Tomi Poutanen. 2017. Content-Based Neighbor Models for Cold Start in Recommender Systems. In *Proceedings of the Recommender Systems Challenge 2017* (Como, Italy) *(RecSys Challenge '17)*. Association for Computing Machinery, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/3124791.3124792

## A FINAL MODEL SCORES AND BLENDING

Table 1. RCE scores for different targets of local validation dataset and public test dataset

| | Reply | | Retweet | | Retweet Comment | | Like | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|
| Model | Local | LB | Local | LB | Local | LB | Local | LB | Sum Local | Sum LB |
| XGBoost 1 | 20.2 | 19.6 | 32.6 | 30.1 | 13.6 | 13.3 | 28.8 | 26.7 | 95.2 | 89.7 |
| XGBoost 2 | 30.0 | 20.9 | 46.5 | 34.1 | 21.2 | 16.3 | 70.0 | 40.3 | 167.6 | 111.6 |
| XGBoost 3 | 24.6 | 23.3 | 39.5 | 36.1 | 16.4 | 15.2 | 62.6 | 52.3 | 143.1 | 126.9 |

Table 2. Weights for blending different models from our final submission

| | Reply | Retweet | Retweet Comment | Like |
|---|---|---|---|---|
| XGBoost 1 | 0.12 | 0.15 | 0.00 | 0.28 |
| XGBoost 2 | 0.40 | 0.50 | 0.30 | 0.12 |
| XGBoost 3 | 0.48 | 0.35 | 0.70 | 0.60 |

## B COMPUTATION TIME

Table 3. Computation time in seconds for different infrastructure and libraries. The pandas train&predict step uses LightGBM on CPU.

| | Pandas | Intel Xeon CPU (20 cores) | 1xV100 | 4xV100 | 4xV100+UCX |
|---|---|---|---|---|---|
| target encoding | 30120 | 1785 | 302 | 108 | 46.4 |
| count encoding | 540 | 485 | 50 | 18 | 7.5 |
| lag features | 1380 | 121 | 62 | 7 | 3.3 |
| train & predict | 6900 | 844 | 180 | 56 | 56 |
| other | 0 | 335 | 426 | 81 | 24.8 |
| Total | 38940 | 3570 | 1020 | 270 | 138 |

## C IMPORTANCE TABLE

Table 4. Feature importance for the top 15 features for XGBoost 3 model predicting the target reply.

| Feature Name | Description | Importance |
|---|---|---|
| a_follower_count | Number of followers of the user (engaged with) | 6192 |
| TE_a_user_id_reply | Target Encode (TE) engaged with user id | 5614 |
| a_following_count | Number of accounts the user is following (engaged with) | 4904 |
| TE_b_user_id_reply | TE engaging user id | 4798 |
| TE_b_user_id_tweet_type_language_reply | TE engaging user id + tweet type + language | 4393 |
| TE_tw_last_word_tweet_type_language_reply | TE least popular word + tweet type + language | 4353 |
| TE_tw_first_word_tweet_type_language_reply | TE most popular word + tweet type + language | 4263 |
| TE_hashtags_media_tweet_type_language_reply | TE most popular hashtag + media + tweet type + language | 4252 |
| TE_tw_hash1_tweet_type_language_reply | TE 2nd user hash + tweet type + language | 4134 |
| TE_tw_rt_uhash_tweet_type_language_reply | TE retweet userhash + tweet type + language | 3936 |
| count_char | Number of characters in the tweet | 3898 |
| b_follower_count | Number of followers of the user (engaging) | 3729 |
| TE_links_media_tweet_type_language_reply | TE links + media + tweet type + language | 3685 |
| TE_a_user_fering_count_mode_tweet_type_langua | TE difference mode a_following_count + mode a_follower_count + tweet_type + language | 3563 |
| TE_domains_media_tweet_type_language_reply... | TE domains + media + tweet_type + language | 3504 |

Table 5. Feature importance for the top 15 features for XGBoost 3 model predicting the target retweet.

| Feature Name | Description | Importance |
|---|---|---|
| TE_a_user_id_retweet | Target Encode (TE) engaged with user id | 6796 |
| TE_b_user_id_tweet_type_language_retweet | TE engaging user id + tweet type + language | 6220 |
| a_follower_count | Number of followers of the user (engaged with) | 6195 |
| TE_b_user_id_retweet | TE engaging user id | 5850 |
| a_following_count | Number of accounts the user is following (engaged with) | 4832 |
| TE_hashtags_media_tweet_type_language_retweet | TE most popular hashtag + media + tweet type + language | 4827 |
| TE_tw_hash1_tweet_type_language_retweet | TE 2nd user hash + tweet type + language | 4439 |
| TE_a_user_fering_count_mode_tweet_type_langua | TE mode a_following_count + tweet_type + language | 4367 |
| TE_a_count_combined_tweet_type_language_retwe... | TE combined a_follower_count & a_following_count differences + tweet type + language | 4356 |
| count_char | Number of characters in the tweet | 4252 |
| TE_tw_rt_uhash_tweet_type_language_retweet | TE retweet userhash + tweet type + language | 4114 |
| TE_a_user_fer_count_delta_time_media_language | TE difference mode a_follower_count + media + language | 3898 |
| TE_tw_hash0_tweet_type_language_retweet... | TE 1st user hash + tweet type + language | 3799 |
| TE_links_media_tweet_type_language_retweet | TE links + media + tweet type + language | 3779 |
| TE_a_user_fering_count_delta_time_tweet_type_ | TE difference mode a_following_count + mode a_follower_count + tweet_type + language | 3665 |

Table 6. Feature importance for the top 15 features for XGBoost 3 model predicting the target retweet with comment.

| **Feature Name** | **Description** | **Importance** |
|---|---|---|
| TE_a_user_id_retweet_comment | Target Encode (TE) engaged with user id | 4977 |
| a_follower_count | Number of followers of the user (engaged with) | 4925 |
| TE_tw_last_word_tweet_type_language_retweet_c | TE least popular word + tweet type + language | 4568 |
| a_following_count... | Number of accounts the user is following (engaged with) | 4335 |
| TE_hashtags_media_tweet_type_language_retweet | TE most popular hashtag + media + tweet type + language | 4213 |
| b_follower_count | Number of followers of the user (engaging) | 3917 |
| count_char | Number of characters in the tweet | 3883 |
| TE_tw_rt_uhash_tweet_type_language_retweet_co | TE retweet userhash + tweet type + language | 3866 |
| TE_tw_first_word_tweet_type_language_retweet_... | TE most popular word + tweet type + language | 3796 |
| TE_domains_media_tweet_type_language_retweet_... | TE domains + media + tweet_type + language | 3465 |
| TE_b_user_id_retweet_comment... | TE engaging user id | 3448 |
| b_following_count | Number of accounts the user is following (engaging) | 3387 |
| TE_tw_hash0_tweet_type_language_retweet_comme | TE 1st user hash + tweet type + language | 3346 |
| count_words | Number of words in the tweet | 3313 |
| TE_b_user_id_tweet_type_language_retweet_comm | TE engaging user id + tweet type + language | 3303 |

Table 7. Feature importance for the top 15 features for XGBoost 3 model predicting the target like.

| **Feature Name** | **Description** | Importance |
|---|---|---|
| TE_b_user_id_like | Target Encode (TE) engaging user id | 9461 |
| TE_b_user_id_tweet_type_language_like | TE engaging user id + tweet type + language | 7947 |
| TE_a_user_id_like | TE engaged with user id | 7805 |
| a_follower_count | Number of followers of the user (engaged with) | 7544 |
| a_following_count | Number of accounts the user is following (engaged with) | 4997 |
| TE_a_count_combined_tweet_type_language_like | TE combined a_follower_count & a_following_count differences + tweet type + language | 4943 |
| TE_domains_media_tweet_type_language_like | TE domains + media + tweet_type + language | 4766 |
| TE_a_user_fering_count_mode_tweet_type_langua | TE mode a_following_count + mode a_follower_count + tweet_type + language | 4714 |
| TE_a_user_fing_count_delta_time_media_languag... | TE differences a_following_count + media + language | 4412 |
| TE_a_user_fer_count_mode_media_language_like... | TE difference mode a_follower_count + media + language | 4233 |
| TE_links_media_tweet_type_language_like | TE links + media + tweet type | 4159 |
| TE_tw_rt_uhash_tweet_type_language_like | TE retweet userhash + tweet type + language | 4124 |
| TE_a_user_fer_count_delta_time_media_language | TE time difference mode a_follower_count + media + language | 3924 |
| TE_hashtags_media_tweet_type_language_like... | TE most popular hashtag + media + tweet type + language | 3917 |
| TE_a_user_fing_count_mode_media_language_like | TE difference mode a_following_count + media + language | 3867 |