User Engagement Modeling with Deep Learning and Language Models

Maksims Volkovs
Layer 6 AI
Canada
maks@layer6.ai

Felipe Perez*
Layer 6 AI
Canada
felipe@layer6.ai

Zhaoyue Cheng*
Layer 6 AI
Canada
joey@layer6.ai

Jianing Sun*
Layer 6 AI
Canada
jianing@layer6.ai

Sajad Norouzi*
Layer 6 AI
Canada
sajad@layer6.ai

Anson Wong*
Layer 6 AI
Canada
anson@layer6.ai

Pawel Jankiewicz
Logic AI
Poland
p.jankiewicz@gmail.com

Barum Rho
Layer6 AI
Canada
barum@layer6.ai

ABSTRACT

Twitter is one of the main information sharing platforms in the world with millions of tweets created daily. To ensure that users get relevant content in their feeds Twitter extensively leverages machine learning-based recommender systems. However, given the large volume of data, all production systems must be both memory and CPU efficient. In the 2021 ACM RecSys challenge Twitter simulates the production environment with a large dataset of almost 1 bilion user-tweet engagements that span a 4 week period. The goal is to accurately predict engagement type, and all models are subject to strict run-time constraints during inference. In this paper we present our approach to the 2021 ACM Recsys challenge. We use a hybrid pipeline and leverage gradient boosting, neural network classifiers and multi-lingual language models to maximize performance. Our approach achieves strong results placing 3'rd on the public leaderboard. We further explore the complexity of language model inference, and show that through distillation it can be possible to run such models in highly constrained production environments.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Recommender Systems, Deep Learning, Language Modeling

*Contribute to this work equally

RecSysChallenge 2021, October 1, 2021, Amsterdam, Netherlands

ACM ISBN 978-1-4503-8693-7/21/10.
https://doi.org/10.1145/3487572.3487604

ACM Reference Format:
Maksims Volkovs, Felipe Perez, Zhaoyue Cheng, Jianing Sun, Sajad Norouzi, Anson Wong, Pawel Jankiewicz, and Barum Rho. 2021. User Engagement Modeling with Deep Learning and Language Models. In *RecSysChallenge '21: Proceedings of the Recommender Systems Challenge 2021 (RecSysChallenge 2021), October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3487572.3487604

1 INTRODUCTION

The volume of information is increasing at an exponential rate so recommender systems that surface relevant content to users have become critically important for any online platform. This is particularly relevant for Twitter where millions of tweets are created and shared daily, and most tweets have very short relevance time span. The recommender system must thus be able to surface relevant tweets in users' feeds in near real-time. Given the immense volume of data, all models deployed in production need to satisfy strict run-time constraints to be practical. The 2021 ACM RecSys challenge [1][3][4] organized by Twitter is focused on production requirements of a large-scale real-life recommender system. This challenge is a continuation of the 2020 challenge [2] with the same goal of predicting four types of user-tweet engagements: Reply, Retweet, Retweet with Comment and Like. In contrast to the 2020 challenge, the dataset is significantly larger consisting of almost 1 billion engagements from 45M unique users and 350M unique tweets. The dataset spans 28 days with roughly 40M engagements per day. All submitted models must adhere to strict run-time requirements and complete inference on ~14M test engagements in under 24 hours on a cloud docker environment with 1 CPU and 64GB RAM. These constraints simulate Twitter's production environment where more expensive models become highly challenging to deploy. To account for potential bias where tweets from users with large following get wider exposure and engagement regardless of their relevance or quality, the challenge introduces a fairness metric. Tweet creators are binned into five group based on their follower count and separate evaluation is done on each bin. The final score is the average of scores for each bin and thus equalizes

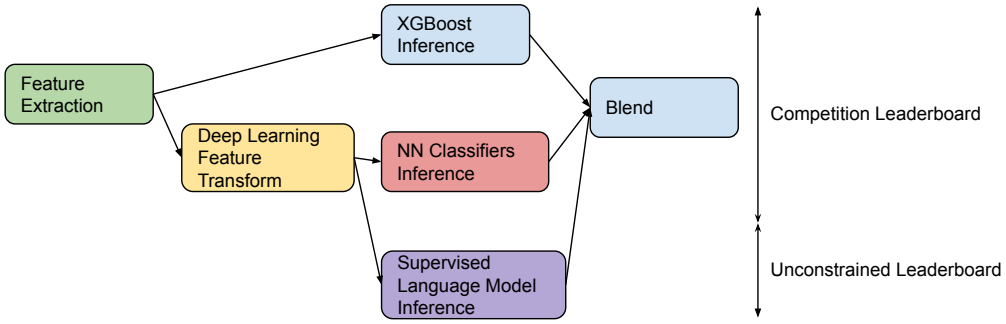


Figure 1: Inference Pipeline. Extracted features are passed to XGBoost and neural network classifiers and their scores are blended to get the final prediction. Language model is used for the unconstrained version of the challenge where CPU and RAM restrictions are lifted.

the relative contribution of each bin penalizing models that preform better on more popular users.

2 APPROACH

Our approach consists of a hybrid gradient boosting (XGBoost) and deep learning pipeline. First, we extract extensive features that summarize information about tweet, tweet creator and engaging user for each engagement. These features are then passed to the XGBoost and neural net (NN) classifiers, and the resulting scores are blended together to produce the final prediction. In addition, for the unconstrained part of the challenge where CPU and RAM requirements are removed, we add a multi-lingual language model fine-tuned for tweet data to extract tweet representations. These representations are passed to the NN classifier as additional feature input. The diagram of our pipeline is shown in Figure 1.

2.1 Data Partitioning

Following our approach to the 2020 challenge [10], we partition the data forward in time. We first order all engagements by tweet creation date and use the last four hours as the forward-in-time validation set. The remaining data from the 3 week training period is used for feature extraction and model training. Splitting by tweet creation date guarantees that our validation set only contains engagements that are strictly after the training engagements, and that tweet id are fully disjoint between the two sets. This simulates the test set-up of the challenge where the goal is to predict engagement types for user-tweet pairs where all tweets are created after the training period. Moreover, while four hours is a relatively small interval, it still contains over 4.8M engagements and we found it to generalise well to the challenge leaderboard. Our data partitioning is illustrated in Figure 2.

To generate training data, we slide a 24-hour non-overlapping window through the training set. Engagements in the each 24-hour window are taken as training targets and those outside of the window are used for feature extraction as shown in Figure 2. Note that we break the temporal structure by also extracting features from engagements that are forward-in-time from the training window. We find that including future engagements leads to a considerable improvement in model performance as it significantly increases the amount of data available for feature extraction.

2.2 Feature Extraction

We conduct extensive feature engineering to describe the tweet, engaging user, and tweet creator for each engagement, with particular focus on historical engagements and similarities between the three entities. In total we extract 443 features that can be organized into four main groups:

- ***Tweet Content Features*** These features directly summarize the content and metadata of the tweet (tweet text tokens, hashtags, present media, present links, and present domains) encoded as one-hot categorical features. We remove values that appear less that 1.5% in the training data to reduce dimensionality and improve generalisation. We also include the number of categorical values for each meta data field that can contain multiple values (e.g. tweet text token count). Following [8], we compute target encoding (TE) for each categorical feature as follows:

$$\text{TE}(C) = \frac{n_C \cdot \bar{y}_C + w_{\text{smooth}} \cdot \bar{y}}{n_C + w_{\text{smooth}}} \tag{1}$$

  where C is a tuple of categorical values, n_C is the appearance count for C, $\bar{y}_C$ is the target mean for C, $\bar{y}$ is the global target mean, and w_{smooth} is a smoothing parameter set to 20. We then use top-3 TE values and statistics such as sum and mean as features. Lastly, tweet text tokens are decoded back to text and we use a regex to generate text features such as number of characters, number of words, number of words with leading uppercase character, number of @ handles, average word length etc.
- ***User Features*** These features summarize the engaging user. We include base statistics such as user's follower and following count, follower-to-following ratio and verified status. We also summarize past engagements for the user, such as total number of engagements for each engagement type, average creator verified status, average user-creator following and average creator follower and following counts. These statistics are repeated for past engagements where user is the creator, and we summarize engaging user for those engagements. To capture the type of content that user typically engages in, we average categorical one-hot features from unique tweets in past engagements.

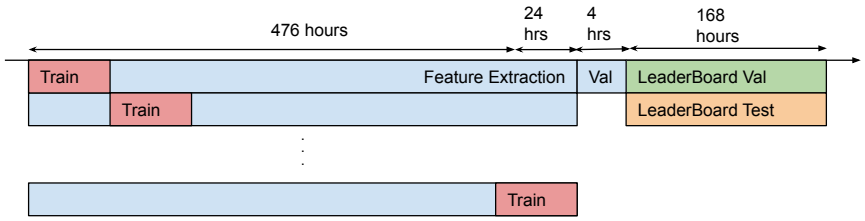


Figure 2: Data Partitioning. Last 4 hours of the 3 week released engagement data are used for validation and the rest for model training. To create the training set we slide a 24-hour non-overlapping window through the training time period. All engagements in the training window are used as training targets and those outside (including forward-in-time) are used for feature extraction. Challenge leaderboard is computed on the engagements from the week after the released data. Two weeks before the challenge ended Twitter released a portion of the engagements from the test week (LeaderBoard Val), allowing challenge participants to train and validate on this data.

- ***Creator Features*** Analogous to engaging user, we extract features for tweet creator. These features are similar to user features, and contain statistics on follower and following counts, verified status, and whether user is following creator. We also compute features from creator's past engagements both as engaging user and as a creator.
- ***User-Creator Features*** For these features we focus on summarizing the relationship between engaging user and creator. We compute the neighbor-based collaborative filtering similarity between user and creator based on the binary interaction matrix $\mathbf{R}$ where $\mathbf{R}_{uu^*} = 1$ if user u previously engaged with any tweet from creator u^*, and $\mathbf{R}_{uu^*} = 0$ otherwise. Similarity between rows/columns of $\mathbf{R}$ indicates the degree to which the corresponding users have similar engagement patterns for engaged/created tweets. We compute this similarity for user-creator and in reverse for creator-user. We also summarize the joint user-creator engagement history where we count the number of engagements of each type between the pair and various other statistics such as time since last engagement etc.

Using these features we train XGBoost and NN classifiers. For XGBoost we train four separate binary models, one for each engagement type. NN architecture is described below.

2.3 Neural Network Classifier

Our NN architecture consists of five main blocks with feed-forward layer and ReLU activation in each block. The output block consists of a fully connected layer followed by a sigmoid activation to output four probabilities that correspond to the four engagement types. Since user can have multiple engagements with a given tweet, we optimize a binary cross entropy loss:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{e} y_{ie}\log\hat{y}_{ie} + (1-y_{ie})(1-\log\hat{y}_{ie}) \tag{2}$$

where N is the number of training examples, e is the engagement type, y_{ie} is 1 if user engaged with the tweet in the form of e and 0 otherwise, and $\hat{y}_{ie}$ is the predicted engagement probability. Our feature input contains features that vary significantly in scale and distribution, e.g. number of followers vs. one-hot encoded tweet meta data. Tree-based models split features by thresholding and are invariant to monotonic transformation, so feature scaling is generally not required. However, NN models use gradient descent to update learnable parameters and are sensitive to input scale [7]. This can be illustrated with a simple 1-layer model $\hat{y}_i = \sigma(\mathbf{w}^T\mathbf{x}_i)$. The gradient step for this model is given by:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i^T\left(y_i - \sigma(\mathbf{w}^T\mathbf{x}_i)\right) \tag{3}$$

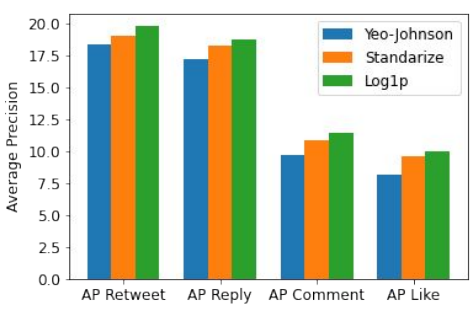
where α is the learning rate. We see that the gradient is scaled directly by the raw input $\mathbf{x}_i$. When different dimensions in $\mathbf{x}_i$ vary significantly in scale the corresponding dimensions in $\mathbf{w}$ will get disproportionally large or small updates. This can be partially mitigated with per-dimension adaptive learning rate optimizers such as Adam, but learning can still get stuck or converge to sub-optimal solutions [7]. In practice, it is nearly always advantageous to apply pre-processing to re-scale the input before it is passed to the NN. We examine the following popular transformations:

- Min-Max scaling: $\mathbf{x}' = (\mathbf{x} - \mathbf{x}_{\min})/(\mathbf{x}_{\max} - \mathbf{x}_{\min})$
- Standarization: $\mathbf{x}' = (\mathbf{x} - \mu)/\sigma$
- Yeo-Johnson power transform [11]
- Log1p transform [7]: $\mathbf{x}' = \log_e(1 + |\mathbf{x}|) \odot \text{sign}(\mathbf{x})$

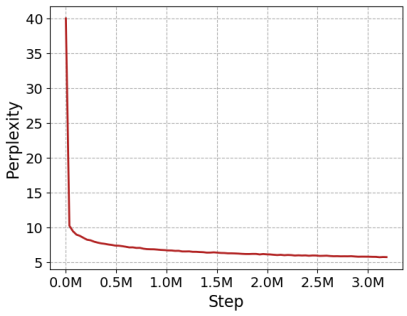
We consistently find that the Log1p transformation works best on this data and use it for all NN models.

2.4 Multi-Lingual Language Model

Tweet text can provide valuable information that is highly relevant for determining the type of engagement. However, the challenge dataset contains tweets in over 60 different languages, making it difficult to apply traditional semantic information extraction techniques such as sentiment analysis. Recent advances in NLP have produced highly accurate language models that have out-of-the-box support for multiple languages. These models are typically based on the Transformer [9] architecture and can generate embeddings that encode semantic and syntactic information across the supported languages. In this work we explore the pre-trained multilingual Bert (MBert) model [5], and demonstrate that it can be used to achieve a significant accuracy boost. MBert is pre-trained on the Wikipidea documents in over 100 different languages. The distribution of text in Wikipidea documents is very different from the highly condensed tweet text that typically contains many abbreviations, slang, emojis and other artifacts. To adapt the model to the target tweet distribution we first apply unsupervised fine-tuning where we use MBert's masked language loss training and apply it



(a)



(b)

Figure 3: (a) Validation set NN performance for different input transformations. (b) Perplexity on unique validation set tweets for the MBert model during unsupervised fine-tuning.

Stage	Model	AP Retweet	RCE Retweet	AP Reply	RCE Reply	AP Like	RCE Like	AP Comment	RCE Comment
Before Fine-Tuning	XGB	0.4004	25.71	0.2308	25.14	0.6654	15.36	0.0627	16.86
	NN	0.4007	25.46	0.2309	25.02	0.6600	13.99	0.0617	16.59
	SLM	0.4097	26.12	0.2505	26.25	0.6769	14.33	0.0658	17.26
After Fine-Tuning	XGB	0.4221	27.61	0.2397	26.21	0.7052	20.87	0.0646	17.44
	NN	0.4291	28.07	0.2442	26.57	0.7132	21.64	0.0675	17.91
	SLM	0.4411	29.34	0.2684	28.34	0.7247	23.01	0.0720	18.89
	SLM (small)	0.4401	29.06	0.2665	28.04	0.7241	26.64	0.0715	18.65
	SLM (tiny)	0.4372	28.92	0.2617	27.77	0.7213	22.41	0.0710	18.53

Table 1: LB Val Results. We show results for each model before and after it is fine-tuned on the 70% of the LB Val. All evaluation is done on the remaining 30% of LB Val. AP is average precision and RCE is relative cross entropy (log loss), see [2] for more details.

	Team	AP Retweet	RCE Retweet	AP Reply	RCE Reply	AP Like	RCE Like	AP Comment	RCE Comment
Competition Leaderboard	1. nvidia_rapidsai	0.4614	29.51	0.2649	26.61	0.7216	23.61	0.0692	17.68
	2. Synerise_v1	0.4514	28.52	0.2559	25.74	0.7046	22.09	0.0662	16.92
	3. **LAYER6_AI**	0.4317	27.42	0.2490	25.35	0.6836	19.85	0.0660	16.86
	4. test_lightgbm	0.4060	25.09	0.2118	22.64	0.6636	17.91	0.0520	14.03
Unconstrained Leaderboard	**LAYER6_AI**	0.4496	29.02	0.2741	27.28	0.7011	21.65	0.0715	18.06

Table 2: Final Leaderboard Results. We show top-4 teams from the final competition leaderboard as well as our best result from the unconstrained leaderboard where CPU and RAM restrictions are removed.

to tweet text. We incorporate this model into the NN classifier by taking MBert's output CLS token and appending it to input features. The full architecture is then trained end-to-end by propagating the gradient to update all NN and MBert layers.

3 EXPERIMENTS

3.1 Training Details

All feature extraction and deep learning experiments are conducted on Ubuntu servers with Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, 256GB RAM, and Nvidia Titan V GPUs. Language model training experiments are conducted on a server with 768GB RAM and 8 Tesla V100 GPUs. By applying the sliding window approach we generate 712,852,721 training and 4,859,770 validation instances. For XGBoost we use the gradient tree booster with 200 trees, depth 15, learning rate 0.1, and column sub-sampling 0.8. Separate XGBoost model is trained for each of the four engagements using the binary logistic loss.

For NN classifier we train three different 5-layers architectures: [400, 800, 500, 250, 100], [500, 900, 600, 250, 100], and [512, 1024, 500, 250, 100], where numbers represent hidden layer sizes from first to last layer. All NN models are trained with a large batch size of 1M engagements, learning rate of 1e-4, layer-wise dropout in $\{0.1, 0.2, 0.3\}$ and Adam optimizer. We test different input transformations outlined in Section 2.3 and the results are shown in Figure 3a. There is a clear pattern where Log1P transform leads to the best performance on each of the four engagements followed by Standardization and Yeo-Johnson. Notably with the Log1P transform we are able to nearly match the results from a well tuned XGBoost model which is challenging to achieve on tabular features [7].

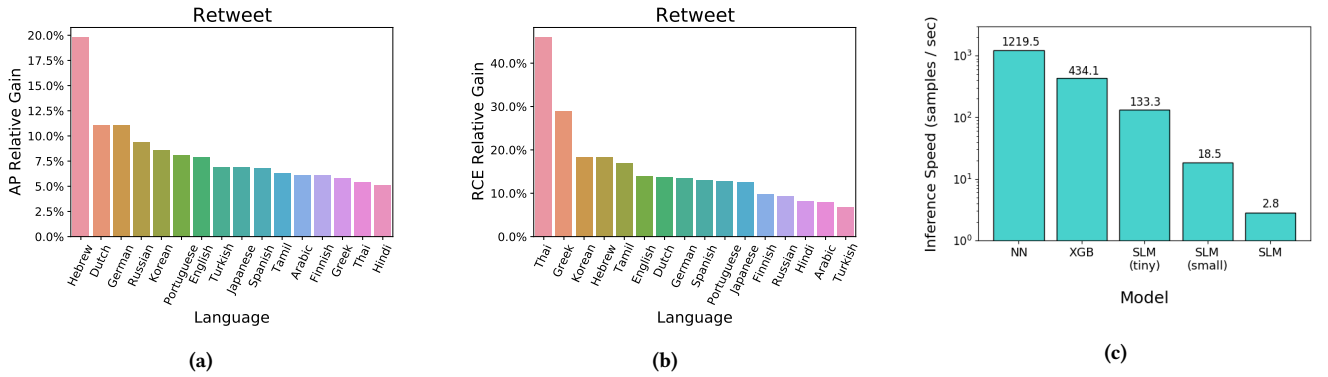


Figure 4: (a) & (b) Relative improvement for SLM over NN on Retweet engagement broken down by language. (c) Model inference speed in samples per second on the target docker image where only one CPU is available and 64GB of RAM.

We take the pre-trained MBert model from the Huggingface library[1], the base model has 12 Transformer blocks with 768 dimensions each. This model is fine-tuned with unsupervised masked language loss on 341M unique training set tweets for 3M steps with a learning rate of 2e-5 and batch size of 64 tweets per GPU. The perplexity on unique validation set tweets is shown in Figure 3b, we see a very significant improvement from over 40 at the start of fine-tuning to 5.72 at the end. The almost 8x reduction in perplexity demonstrates that fine-tuning is very important to adapt the language model to the tweet distribution. After unsupervised fine-tuning, we connect the CLS token to the NN classifier and do end-to-end supervised training. The NN is randomly initialized and the MBert is initialized with the average of three checkpoints from unsupervised fine-tuning. We use a learning rate of 5e-4 for all NN layers and 1e-5 for all MBert layers. We deliberately use a much larger learning rate for NN layers since they are initialized randomly and need more updates than MBert which is near convergence. To improve training stability we use gradient clipping of 1 for all layers. We refer to this model as SLM.

3.2 Results

Two weeks before the competition end Twitter released a portion of the engagement data from the target test week. We refer to this data as the leaderboard validation set (LB Val), and challenge participants were allowed to both train and evaluate models on this set. As this data contains direct information from the target test time period we focus on evaluating all models on LB Val. To maximize performance we split LB Val by unique tweet ids where 70% is used to fine-tune models and remaining 30% for evaluation. To fine-tune XGBoost we continue adding trees to the ensemble that are fitted on the 70% of LB Val. The results are shown in Table 1, we see that before fine-tuning XGBoost generally outperforms NN but the gains are relatively minor indicating that the Log1p transform is highly effective for NN training. Adding language model to NN leads to a significant boost in performance across all four engagements, with particularly large gains on Reply where SLM improves AP by close to 2 points or 8.4%.

[1]https://huggingface.co/bert-base-multilingual-cased

Results in the bottom half of the table show that, as expected, fine-tuning on LB Val is highly effective with large gains across all models and engagement types. Tweets have a strong temporal dynamic and typically contain in-the-moment information (news events, announcements etc.) that has a short life span. Consequently, training on the engagements from the test time period is highly beneficial as the model is able to correlate what information is relevant to users at that particular moment in time. However, we caution against such training if the intended use is forward-in-time prediction.

Final leaderboard results are shown in Table 2. We use an ensemble of three NN models and an XGboost model, where each model is fine-tuned on the LB Val set. Our approach, LAYER6 AI, achieves highly competitive performance placing 3'rd with a total docker inference runtime of 13 hours. We also show our best unconstrained result where CPU and RAM restrictions are removed. Here, we add the language model-based classifier SLM to the ensemble and observe a consistent gain across all engagement for both AP and RCE.

3.3 Language Model Analysis

In Figures 4a and 4b we show languages that get the largest performance boost on the Retweet engagement from adding the language model relative to the base NN classifier. The more rare languages such as Hebrew, Dutch, Thai, and Greek get the biggest boost with up to 18% and 45% relative improvement in AP and RCE respectively. Rare languages don't have sufficient engagement data for the model to accurately capture preferences for users that engage with that content. Accurately modeling tweet content can be particularly beneficial in these sparse situations since the language model is pre-trained on a large text corpus from each language.

Adding MBert to the NN classifier makes the model considerably more expensive to run, and we are unable to run it on the restricted docker environment. To investigate whether language models can be practically used in constrained production environments, we experiment with distillation to reduce model size and improve runtime. In particular, we experiment with two smaller versions of the MBert architecture: *small* [2-768-1536] and *tiny* [1-128-512], where numbers correspond to number of Transformer blocks, vocabulary

embedding dimension, and hidden dimension respectively. For both architectures we first train them with distillation on MBert using a temperature of 2 (following [6]), then attach to NN classifier and fine-tune end-to-end as in SLM. The results are shown in Table 1 as SLM (small) and SLM (tiny). We see that even with a very compact SLM (tiny) architecture we can preserve most of the SLM gains and still significantly improve over the best NN classifier.

Figure 4c shows inference runtime in examples per second on the constrained docker environment with one CPU and 64GB of RAM. To score 14M test engagements in 24 hours on this environment the model has to have an inference throughput of at least 162 examples per second. From the figure we see that both NN and XGBoost are comfortably above that threshold. XGBoost is around 3x slower than NN because it contains 4 separate ensembles (one for each engagement) which can only be run sequentially on one CPU. As such XGBoost is best suited for binary classification since inference runtime scales linearly with the number of classes. We also see that adding MBert significantly slows down inference, where SLM can only process 2.8 examples per second and would require over 57 days to process the full test set. Distillation considerably improves this runtime, in particular the SLM (tiny) architecture has a throughput of 133.3 which is near the target throughput threshold. These results demonstrate that through distillation we can leverage the representational power of language models while satisfying very strict production constraints. We believe that with further engineering and experiments, both throughput and accuracy of SLM (tiny) can be further improved.

4 CONCLUSION

In this paper we present our approach to the 2021 ACM RecSys Challenge organized by Twitter. Our pipeline consists of extensive feature extraction followed by XGBoost and neural network classifiers. In the unconstrained phase of the challenge, we further explore multi-lingual language models, and show that they lead to a significant boost in performance. We then apply distillation to compress the language model architecture, and demonstrate that with sufficient engineering effort these models can be run in highly constrained production environments.

REFERENCES

[1] [n. d.]. Twitter RecSys Challenge 2021. https://recsys-twitter.com/.

[2] Vito Walter Anelli, Amra Delić, Gabriele Sottocornola, Jessie Smith, Nazareno Andrade, Luca Belli, Michael Bronstein, Akshay Gupta, Sofia Ira Ktena, Alexandre Lung-Yut-Fong, et al. 2020. RecSys 2020 Challenge Workshop: Engagement Prediction on Twitter's Home Timeline. In *ACM Conference on Recommender Systems*.

[3] Vito Walter Anelli, Saikishore Kalloori, Bruce Ferwerda, Luca Belli, Alykhan Tejani, Frank Portman, Alexandre Lung-Yut-Fong, Ben Chamberlain, Yuanpu Xie, Jonathan Hunt, Michael M. Bronstein, and Wenzhe Shi. 2021. RecSys 2021 Challenge Workshop: Fairness-aware engagement prediction at scale on Twitter's Home Timeline. In *RecSys '21: Fifteenth ACM Conference on Recommender Systems, Amsterdam, The Netherlands, 27 September 2021 - 1 October 2021*, Humberto Jesús Corona Pampín, Martha A. Larson, Martijn C. Willemsen, Joseph A. Konstan, Julian J. McAuley, Jean Garcia-Gathright, Bouke Huurnink, and Even Oldridge (Eds.). ACM, 819–824. https://doi.org/10.1145/3460231.3478515

[4] Luca Belli, Alykhan Tejani, Frank Portman, Alexandre Lung-Yut-Fong, Ben Chamberlain, Yuanpu Xie, Kristian Lum, Jonathan Hunt, Michael Bronstein, Vito Walter Anelli, Saikishore Kalloori, Bruce Ferwerda, and Wenzhe Shi. 2021. The 2021 RecSys Challenge Dataset: Fairness is not optional. arXiv:cs.SI/2109.08245

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Association for Computational Linguistics*.

[6] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[7] Zhen Qin, Le Yan, Honglei Zhuang, Yi Tay, Rama Kumar Pasumarthi, Xuanhui Wang, Michael Bendersky, and Marc-Alexander Najork. 2021. Are Neural Rankers still Outperformed by Gradient Boosted Decision Trees?. In *International Conference on Learning Representations*.

[8] 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 Workshop on ACM Recommender Systems Challenge*.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

[10] Maksims Volkovs, Zhaoyue Cheng, Mathieu Ravaut, Hojin Yang, Kevin Shen, Jin Peng Zhou, Anson Wong, Saba Zuberi, Ivan Zhang, Nick Frosst, Helen Ngo, Carol Chen, Bharat Venkitesh, Stephen Gou, and Aidan N. Gomez. 2020. Predicting Twitter Engagement With Deep Language Models. In *Proceedings of the Workshop on ACM Recommender Systems Challenge*.

[11] In-Kwon Yeo and Richard A Johnson. 2000. A new family of power transformations to improve normality or symmetry. *Biometrika* 87, 4 (2000), 954–959.