RecSys Challenge 2021

User Engagement Modeling with Deep Learning and Language Models

Maksims Volkovs, Felipe Perez*, Zhaoyue Cheng*, Jianing Sun*,
Sajad Norouzi*, Anson Wong*, Pawel Jankiewicz, Barum Rho

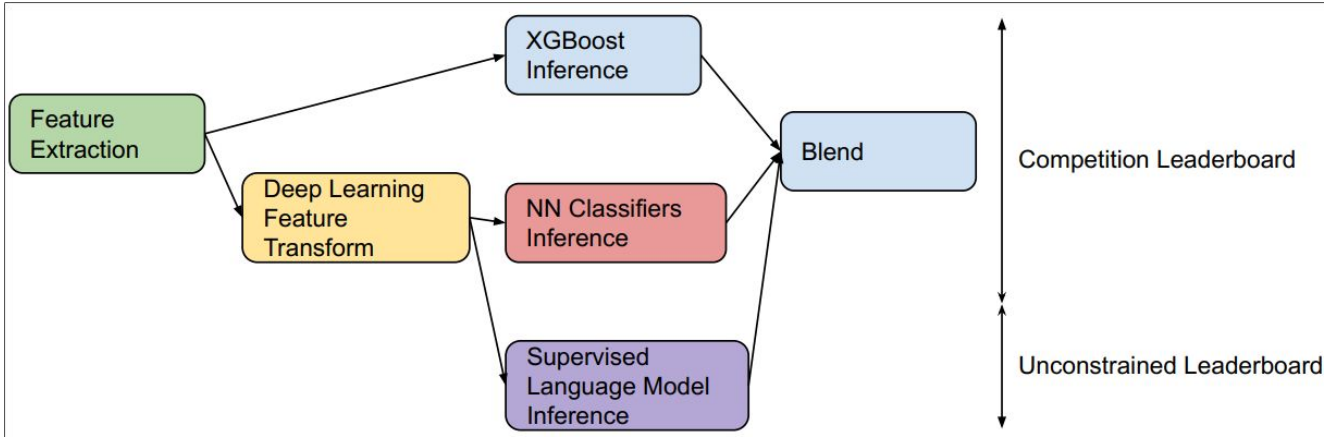
layer 6
AI at TD

Agenda

- Data Partitioning
- Feature Extraction
- XGBoost Model
- NN Classifier Model
- Multi-Lingual Language Model
- Experiments

layer 6
AI at TD

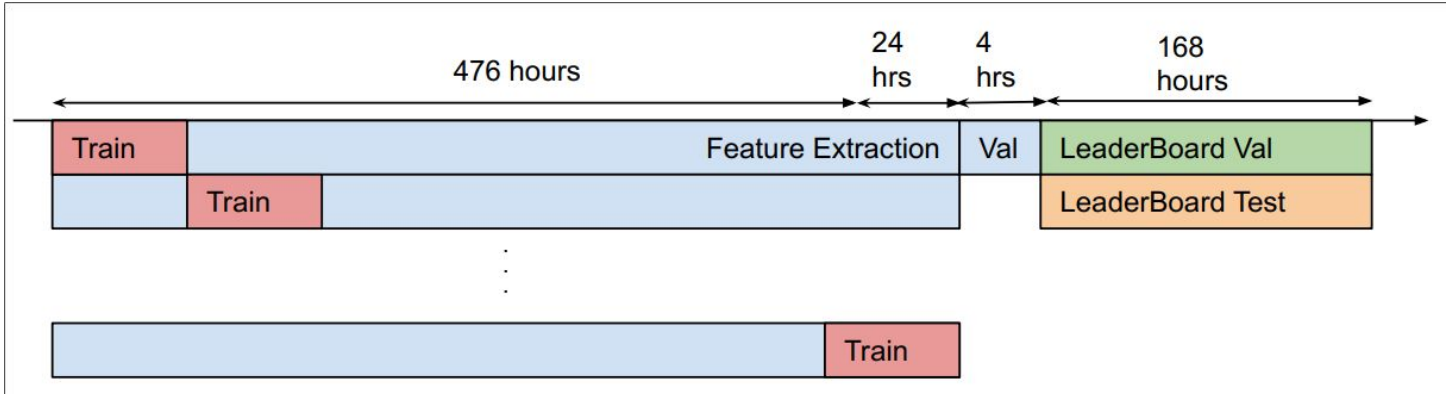
Approach Overview



- We adopt a hybrid gradient boosting (XGBoost) and Deep Learning Pipeline
- We first extract extensive features that summarize information about tweet, creator, user
- Then we pass the features to XGBoost and neural net (NN) classifiers
- In the unconstrained leaderboard, we use a multi-lingual language model fine-tuned for tweet data to extract tweet representations to help the prediction

Data Partitioning

Data Partitioning



- We split four engagements by time and use the last 4 hours as forward in-time validation set
- This split simulates a real-life production environment
- To generate training data, we slide a 24-hour non-overlapping window through the training set

Feature Extraction

Feature Extraction

In total we extract 443 features that can be organized into 4 main groups

- Tweet Content Features
- User Features
- Creator Features
- User-Creator Features

Feature Extraction - Tweet Content Features

- We summarize content and metadata (tweet text tokens, hashtags, present media, present links, and present domains) of the tweet and encode them as one-hot categorical features
- We also include the number of categorical values for each metadata field that contain multiple values, we compute target encoding (TE) for each categorical feature
- 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

Feature Extraction - User Feature

- User features summarize the engaging user
- We include base statistics such as 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
- To capture the type of content that user typically engages in, we average categorical one-hot features from unique tweets in past engagements.

Feature Extraction - Creator Feature

- These features are similar to user features
- We compute statistics on follower and following counts, verified status and whether the user is following creator
- We also compute features from creator’s past engagements both as engaging user and as creator

Feature Extraction - User-Creator Feature

- For these features we focus on summarizing the relationship between engaging user and creator
- We compute neighbor-based collaborative filtering similarity between user and creator based on the binary interaction matrix R
- Similarity between rows/columns of R indicates the degree to which the corresponding users have similar engagement patterns for engaged/created tweets
- 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

XGBoost Model

XGBoost Classifier

- Using the features described before, we train our XGBoost and NN classifiers
- For XGBoost we train 4 separate binary models, one for each engagement type

NN Classifier

NN Classifier - Normalization

- Since our input feature contains features that vary significantly in scale and distribution
- We find that in practice it's nearly always advantageous to apply pre-processing to re-scale the input before it is passed to the NN.
- We examine the following popular transformation:

Min-Max scaling: $\mathbf{x}' = (\mathbf{x} - \mathbf{x}_{\min})/(\mathbf{x}_{\max} - \mathbf{x}_{\min})$

Standarization: $\mathbf{x}' = (\mathbf{x} - \mu)/\sigma$

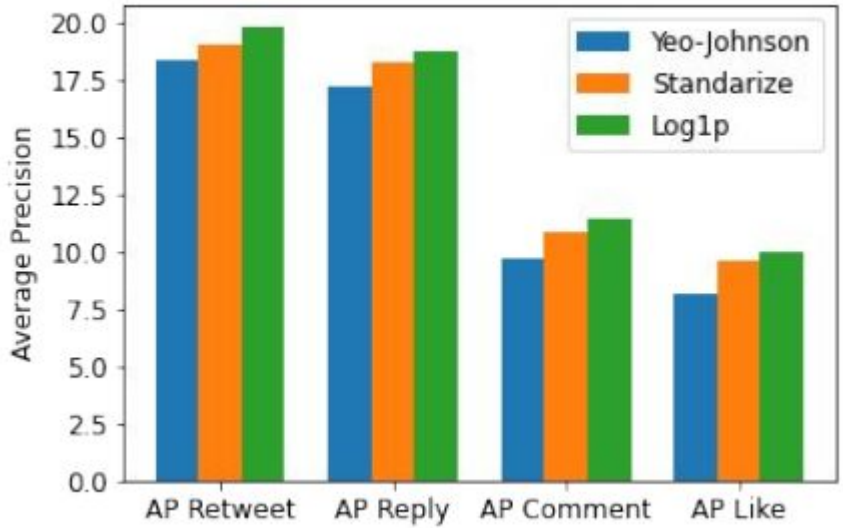
Yeo-Johnson power transform [11]

Log1p transform [7]: $\mathbf{x}' = \log_e(1 + |\mathbf{x}|) \odot \text{sign}(\mathbf{x})$

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

NN Classifier - Normalization

- NN Performance after using different Transformations



NN Classifier - Architecture and Training

- 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

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

NN Classifier - Architecture and Training

- 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

Multi-Lingual Language Model

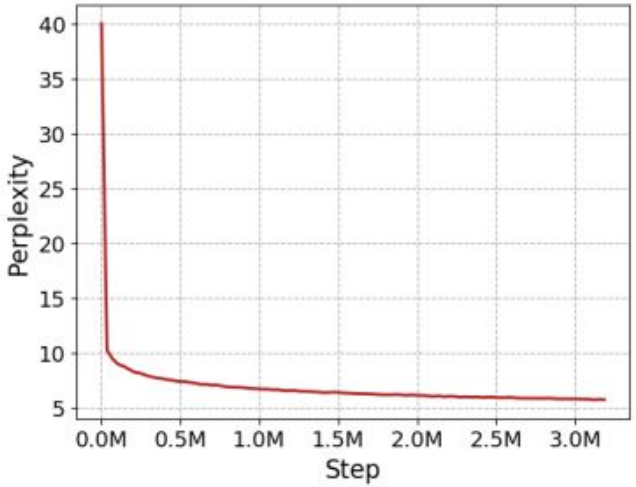
Multi-Lingual Language Model

- we explore the pre-trained multilingual Bert (MBert) model, and demonstrate that it can be used to achieve a significant accuracy boost
- 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 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

Multi-Lingual Language Model - Training

- We fine tune the Multi-Lingual BERT model 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
- We see very significant improvement on validation perplexity from over 40 at the start of fine-tuning to 5.72 at the end
- After unsupervised fine-tuning, we connect the CLS token to the NN classifier and do end-to-end supervised training
- 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
- We refer to this model as SLM

Multi-Lingual Language Model - Perplexity



(b)

Perplexity on unique validation set tweets for the BERT model during unsupervised fine-tuning

Experiments

Leaderboard Results

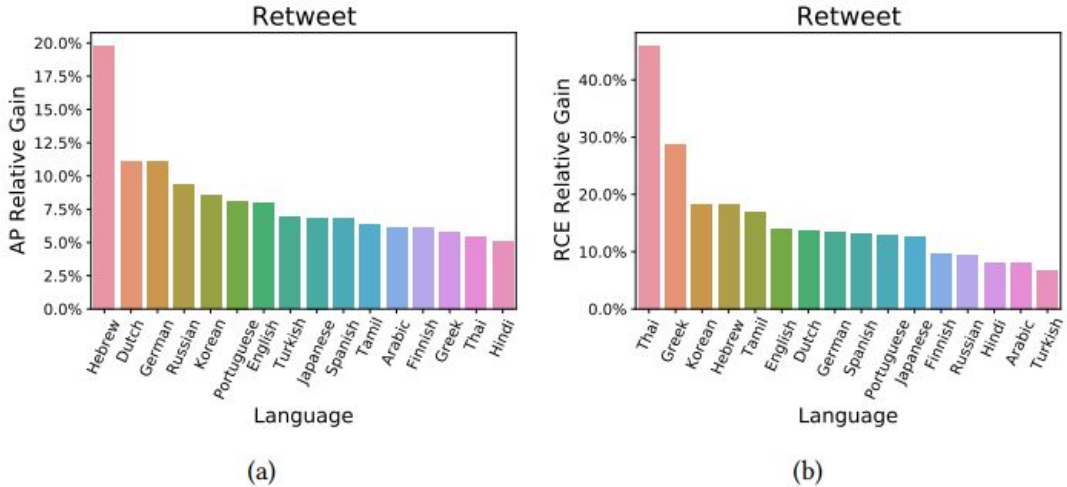
Stage	Model	AP Retweet	RCE Retweet	AP Reply	RCE Reply	AP Like	RCE Like	AP Comment	RCE Comment
Before Fine-Tuning	XGB	0.4004	25.71	0.2308	25.14	0.6654	15.36	0.0627	16.86
	NN	0.4007	25.46	0.2309	25.02	0.6600	13.99	0.0617	16.59
	SLM	0.4097	26.12	0.2505	26.25	0.6769	14.33	0.0658	17.26
After Fine-Tuning	XGB	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.

Language Model Analysis



- Relative improvement for SLM over NN on Retweet Engagement broken down by language

Distillation Experiments

- 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
- 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, then attach to NN classifier and fine-tune end-to-end as in SLM
- The results are shown 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

Distillation Results

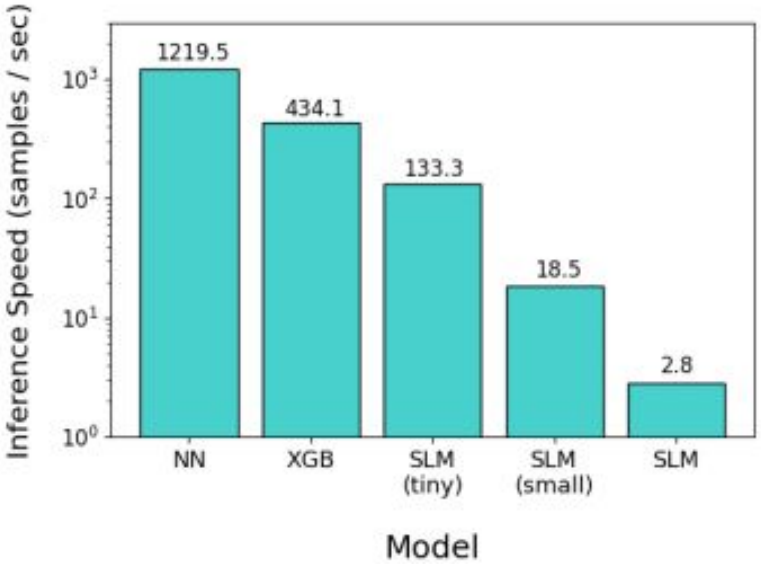
Stage	Model	AP Retweet	RCE Retweet	AP Reply	RCE Reply	AP Like	RCE Like	AP Comment	RCE Comment
Before Fine-Tuning	XGB	0.4004	25.71	0.2308	25.14	0.6654	15.36	0.0627	16.86
	NN	0.4007	25.46	0.2309	25.02	0.6600	13.99	0.0617	16.59
	SLM	0.4097	26.12	0.2505	26.25	0.6769	14.33	0.0658	17.26
After Fine-Tuning	XGB	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.

Model Inference Speed



- Model inference speed of different models in terms of samples/sec

Thank you!