CSC_51054_EP
DATA CHALLENGE REPORT

Team "Louvre Security"

December 2025

Andrey Belyaev, Claire Mytelka, Vaughn Janes

1 - Tried approaches

1.1 • Conventional Deep Learning

General idea: use a pretrained transfomer (trained on French language content) to process the natural language columns, concatenate the embeddings with some metadata, and feed that to an MLP classfier.

The metadata features used were of several categories:

- Numeric/boolean columns;
- String timestamp columns, parsed into utc time;
- Lengths of various string/array columns;
- Channel values from color columns;
- Number of occurences of various substrings in the tweet text (specifically, emojis).

The specific features were chosen based on correlation with the label field.

The text features were embedded with the `camembert-large` transformer. Several options were tried (`camembert-base`, `camembert-large`, `camembertav2-base`, `distilbert-base-cased`, `distilbert-en-fr-cased`, `flaubert_base_cased`, `flaubert_small_cased` — most being BERT-based encoders finetuned on French language texts), and this one was chosen as the most efficient. Finetuning was attempted, but appeared ineffective. The contextual embeddings were precomputed for the entire dataset and cached in order to improve speed. To combat overfitting, the embeddings for some text features were subjected to PCA dimensionality reduction, however this did not produce any positive effect. The text features were not preprocessed outside of the model's own tokenizer, but some text features augmentation was done (a text feature was generated from several smaller, semantically related string features).

For the classifier head, 2 and 3 layers were tried, with the hidden dimension ranging between 256 and 768. Best performance was eventually achieved with 3 layers, 256 `hidden_dim`. Higher values resulted in more prominent overfitting. For the activation function, ReLU was used. The first layer was split into several parallel linear layers, whose outputs were added together, which enabled independent processing of different feature embeddings in different ways. Specifically:

- Different dropout coefficients between 0.1 and 0.8 were used for the input features, depending on their estimated impact on overfitting. Eventually only values up to 0.4 were used.
- The first layer weights for useful features prone to significant overfitting (specifically, the embedding for `user.description`) were frozen after an established number of epochs. This has proven to be an efficient way to reduce overfitting. The classifier head predicted two outputs, the logits for the two classes.

1.2 • Gradient Boosting

General idea: train a gradient tree boosting model via the CatBoost library.

Most columns of the dataset were used as features. More precisely:

- Constant-value columns, as well as some ID columns considered unrelated to the user characteristic being inferred, were discarded.

- All numeric columns were used directly as numeric features.
- All boolean columns were used as categorical features (with `NA` converted into a third category).
- For most string and array-like fields, the length was used as a numeric feature.
- String columns with natural text were used as text feature (subject to basic text processing by CatBoost). This includes the full tweet text, a version of it with the usernames, urls and emojis replaced with placeholds, the user description, and the same for the quoted status.
- String columns with few possible values were used as categorical features.
- Array-like columns were not used (except for their lengths) due to complexity of handling.
- String timestamp columns were converted to unix timestamps and used as numeric features.
- String color columns were converted into numeric values of all 3 channels and used as numeric features.
- The hour, month and day of the week the tweet was posted at were introduced as separate categorical features.
- The number of uppercase letters in the tweet's text, as well as the number of occurences for a few common emojis, were used as numeric features.
- The contextual embeddings for the full text and the user description were used as numeric features in some runs.

The hyperparameters were initially chosen by grid search, which demonstrated monotonic dependency on most of them (depth, number of training iterations, learning rate). Some incremental changes were tried during further experimentation (which showed results breaking the observed monotony over the aforementioned parameters), but full grid search was not repeated due to the large required time.

1.3 • Per-user answer reconciliation

A simple trick that was applied in conjunction with both approaches described above and resulted in consistent accuracy improvements of around 0.5%. The idea is to group the posts likely belonging to the same user and, since we're predicting a property of a user, unify the answer for them via majority vote.

For estimating the user identity, three columns are considered: `user.description`, `user.created_at`, `user_profile_image_url`. The latter, when present, includes the user's internal ID, but using all three provides redundancy against missing values. When all three fields are missing, identity is instead based on the post ID to avoid erroneously conflating unidentified users. The involved fields are hashed using `hashlib.blake2s(digest_size=16)`, as a compromise between ease of use, speed and collision avoidance, to get a conservative synthetic user identifier.

The inferred identities are also used to govern the train-validation split, ensuring that there are no common users between the two datasets. This is crucial to obtaining accurate validation results since the inference dataset has no common users with the combined train dataset.

2 - Results

Deep Learning based models have achieved, at best, 82.8% accuracy on the inference dataset. Gradient Boosting based models have achieved, at best, 82.3%. For the final submission, the former was chosen.

Interestingly, around 80% accuracy could be achieved with only metadata (aka, no tweet text / user description embeddings except their lengths). The embeddings only contributed to the final 2-3% improvement.

Of all the fields, `user.description` (or rather the contextual embedding thereof) was most prone to overfitting. This can be explained by its direct correlation with user identity, which in turn is directly correlated with the label. The field has proven useful in generalizable inference, but it was challenging to find the balance between fitting on it efficiently and overfitting. Several methods were conceived to fight this specific case of overfitting, as previously described:

- PCA dimensionality reduction of the embedding (proven itself more harmful than beneficial);
- High to extremely high dropout fractions for the specific input (eventually producing a positive result, but not eliminating overfitting outright);
- Freezing of corresponding first layer weights after several epochs (seems most efficient, but still not a comprehensive solution).

Another interesting observation is the discrepancy in validation accuracy fidelity between the two kinds of models tried. With the DL-based model, the validation accuracy closely matched the inference accuracy (thanks to the careful, user identity aware split, as well as general good hygiene with isolating the validation+inferense sets from the training process). On the other hand, with CatBoost, the validation accuracy was consistently up to 4% above the inferense accuracy. We were unable to track down the exact cause, but the current leading hypothesis is one of the features available to it (but not to the DL model) having a different distribution in the combined training set compared to the inference set.

3 - Future work

A direct avenue for improvement of the DL model could be incorporating the encoder into the base model (as opposed to having it as part of the preprocessing pipeline, giving fixed precomputed embeddings) and finetuning it with a lower learning rate. The current best classifier head should be taken as a base, and care must be employed to push back the overfitting on `user.description` further than before (for example, even more aggressive dropout). This can be expected to give a small but steady improvement. Extended finetuning is not recommended because of the small size of the training dataset for a transformer model (only 10^5 strings of around 10^2 characters).

For the catboost model, the most straightforward way of improvement would be extracting more features (for example, converting list-like features into usable representations) or improving the quality of existing ones. The feature(s) supposedly suffering from a distribution shift between the train and inference datasets should be identified and either removed or somehow corrected.

Two approaches could be combined, taking the hidden layer values from the DL classifier and providing them as features to the Catboost model. This can be hoped to leverage the linear combinations of inputs identified by the DL model with the precision of a decision tree.

A different gradient tree boosting library could be tried, which might yield slightly better results by virtue of bundled methodical optimizations.

The information from different posts could be aggregated in some ways before being fed to the model.