3rd Place Solution of Feedback Prize - Evaluating Student Writing

Ahmet Erdem, Shujun He, and Darek Kleczek*

1 Background of our team

We participated in the recent competition "Feedback Prize - Evaluating Student Writing", which was hosted during the first quarter of 2022. Our team name is bestoverfitting. Our public leaderboard score is 0.731 and our private score is 0.74.

Our team consistes of Ahmet, who is a senior data scientist at NVIDIA in İstanbul Turbkey, Shujun, who is a PhD student in chemical engineering at Texas A&M University in College Station, TX, US, and Darek Kłeczek, who is a data scientist at P&G in Warsaw, Poland.

2 Solution Overview

Our solution is a combination of token classification models using transformers and a stacking framework that classifies spans for 7 discourse types separately. Credit to Chris Deotte and Chase Bowers for sharing their amazing notebooks (https://www.kaggle.com/cdeotte/tensorflow-longformer-ner-cv-0-633 and https://www.kaggle.com/chasembowers/sequence-postprocessing-v2-67-lb). A graphic of our solution can be seen in FIG 1.

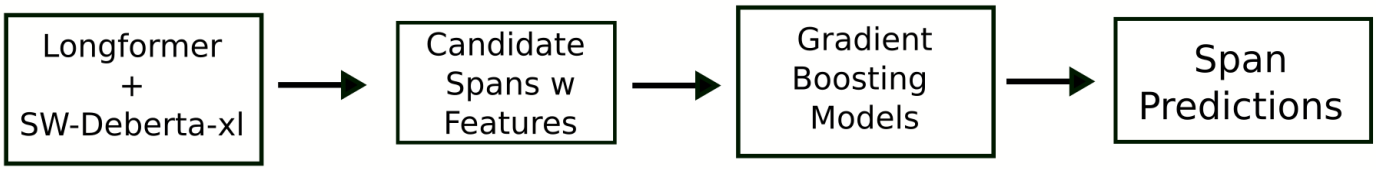

Figure 1: Overview of our solution.

3 Deep learning models

Our best ensemble consists of 6 folds of Longformer [1] and 6 folds of sliding windows Deberta-xl [2] models (weighted by discourse type based on cv). Due to the sequential nature of the problem, we add a 2-layer GRU on top of the transformer hidden states before outputting predictions for each token. We find it beneficial to increase max_len during training/inference to 2048, beyond which we actually saw performance degradation. In our case, deberta-l outperforms longformer large by quite a bit, and deberta-xl is even better, although deberta-xl is more difficult to train.

Below we will discuss some specificities of training these models.

3.1 Longformer

Training longformers is relatively simple because longformers can take sequences longer than 512 without any issues. Our pipeline is similar to what's publicly available, mostly from Chris Deotte (https://www.kaggle.com/chasembowers/sequence-postprocessing-v2-67-lb).

*Authors are listed in alphabetical order

3.2 Sliding window (SW) Deberta-xl

Training deberta is a little trickier than longformers since we can't directly input a sequence longer than 512. Instead, we use a sliding window method when the sequence length is longer than 512. First, the first 512 positions are inputted to the deberta encoder. When we input the next segment into the deberta, we actually only increment the end position by 384, input position [512-64:512-64+512], and only concat the hidden states of the middle segment [64:448] to the hidden states of the first 512 positions, which avoids edge effects. We do this until we reach the end of the sequence and if the last segment is equal to smaller than 64, we simply take the last positions of the previous segment. After we run and concat all the segments, we run the sequence through a 2-layer GRU layer. Since the GRU processes the concatted segments sequentially, it essentially reconnects all the segments.

3.3 Augmentation

We used 2 forms of augmentations: 1. Masked aug, where we mask 152. Cutmix, similar to how cutmix works for images, we cut a portion of one sequence and paste it (and its labels) into another sequence in the same batch. Implementation is quite simple:

```
if np.random.uniform()<0.5:
    cut=0.25
    perm=torch.randperm(ids.shape[0]).cuda()
    rand_len=int(ids.shape[1]*cut)
    start=np.random.randint(ids.shape[1]-int(ids.shape[1]*cut))
    ids[:,start:start+rand_len]=ids[perm,start:start+rand_len]
    mask[:,start:start+rand_len]=mask[perm,start:start+rand_len]
    labels[:,start:start+rand_len]=labels[perm,start:start+rand_len]
```

3.4 Hyperparameters

Since we use augmentations, we train for 7 epochs at learning rates of [2.5e-5, 2.5e-5, 2.5e-5, 2.5e-6, 2.5e-6, 2.5e-6, 2.5e-7] for longformer and deberta-l; for deberta-xl, we use the same schedule with 1/5 of the learnig rate. For longformer, we use a batch size/max_length of 4/2048, for deberta-large 3/2048, and for debrta xl 2/1536 (mostly due to memory constraints).

3.5 Hyperparameters

During training, instead of padding every sequence to a fixed max length, we pad sequences to the max length in its batch, which speeds up training considerably. During inference, we also sort all texts by their lengths before batching, thereby minimizing the amount of padding in each batch, which speeds up inference quite a bit.

4 Stacking framework

Our stacking framework is the same as the one posted by chase bowers; we just made a lot of improvements to it. In short (for those who aren't aware of the notebook), for each discourse type, using out of fold predictions, we generate features for candidate spans based on a begin token probability threshold and a max span length (99.5 percentile for each discourse type), and train gradient boosting models that operate on these features and classify spans. Therefore, we have 7 binary classification models, one for each discourse type. During inference, we sort all candidate spans by predicted probability and take those as predictions until a low threshold, while removing spans that intersect more than 0.15/0.2 with existing predicted spans.

Below we will discuss the improvements.

4.1 CV setup

We set up cross-validation which ended up being mostly consistent with public lb. As a result, most of our improvements locally observed translated to public lb improvements as well. In some cases, however, improvements in noisy classes (e.g. Rebuttal), we saw some inconsistencies, but that was to be expected.

4.2 Features

First we fixed a minor error with calculations of probability that a word corresponds to either a 'B'-egin or 'I'-nside token for a class in the original stacking notebook, where B and I tokens happening for the same class are considered independent events. From:

```
prob_or = lambda word_preds: (1-(1-word_preds[:,disc_begin]) * \
(1-word_preds[:,disc_inside]))
```

to

```
prob_or = lambda word_preds: word_preds[:,disc_begin] + \
 word_preds[:,disc_inside]
```

We added more probabilities at the edges of the span as well as the probability of another B token for any class following the span. Additionally, we added the location of the max/min probabilities for the class in the span. Further, we added something we call instability, which is the average squared difference in prob_or from position to position:

```
s = prob_or(text_preds[pred_start:pred_end])
instability = 0
if len(s) > 1:
    instability = (np.diff(s)**2).mean()
```

Aside from span specific features, we added global features of the average prob_or of every discourse type and the positions with the max B token probs of every disourse type.

Last but not least, not all features are used for every discourse type, and instead we tune our feature selection based on CV. We have used around 25 features on average.

4.3 Increasing the amount of candidate spans

For some discourse types, we reduced the min begin token probability threshold so we have more candidate spans.

4.4 Gradient boosting models

In our best ensemble, we have both an lgbm and an xgb model, each with 5 folds trained on oofs. The predictions from both lgbm and xgb are weighted equally.lgbm is trained on dart mode. Since it is not possible to do early stopping on dart mode, first xgb is trained and its optimal number of trees x 1.4 is used for number of trees. We accelerated our gradient boosting models with RAPIDS ForestInference and got room for an extra deberta fold.

4.5 Decoding

All candidate spans are sorted by predicted probability and taken as predictions until a low threshold. The original notebook did not allow any intersection with existing predicted spans during decoding, but we were able to improve our score by allowing a certain level of intersection (normalized by span length) instead, such as 0.2.

5 Model Execution Time

Each fold of deberta-xl takes around 7 hours to train on an RTX A6000, and each fold of longformer takers around 5 hours to train. We use close to all of the submission time allower (9 hours) with 5 folds of xgboost and light gbm and 6 folds of deberta-xl/longformer.

References

[1] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[2] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654, 2020.