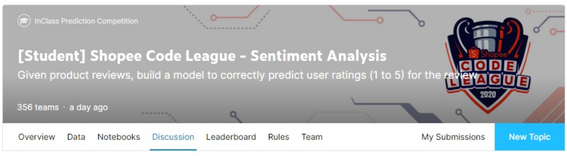

garyong
1st place

1st place solution

posted in student-shopee-code-league-sentiment-analysis a day ago

9

Initial Model

I started out trying many ideas. But they are mostly referenced from the recent competition.
https://www.kaggle.com/c/jigsaw-multilingual-toxic-comment-classification/overview
Here are some of the models i tried.

Model	Score Early stopping CV	last epoch CV	leaderboard
TFIDF logistic regression		0.45	0.45	0.408
Roberta unpaired		0.4749	0.4728	
Roberta cased auxuary loss		0.4762	0.4728	
Roberta cased label smoothing		0.4762	0.4762	
Roberta cased backtranslation augmentation				
Roberta max+mean pooling				
xlm-roberta cased		0.4836	0.481	0.44618
xlm-roberta cased max+mean pooling				
xlm-roberta cased label smoothing				
xlm-roberta pseudo labelling				0.44204

After the initial results it decided to work on XLM-roberta base. I think XLM worked better because it was trained on many languages including Indonesian and there were alot of indonesian words inside. The model is pretty simple and here is how it looks like in pytorch.

```
class CustomRoberta(nn.Module):
    def __init__(self):
        super(CustomRoberta, self).__init__()
        self.num_labels = 5
        self.roberta = transformers.XLMRobertaModel.from_pretrained("xlm-roberta-base", output_hidden_states=False)
        self.dropout = nn.Dropout(p=0.2)
        self.ln = nn.LayerNorm(768)
        self.classifier = nn.Linear(768, self.num_labels)

    def forward(self,
                ids=None,
                attention_mask=None,
                position_ids=None,
                head_mask=None,
                inputs_embeds=None):
        o1, o2 = self.roberta(ids.squeeze(1),
                              attention_mask=attention_mask.squeeze(1),
                              position_ids=position_ids,
                              head_mask=head_mask,
                              inputs_embeds=inputs_embeds)
        x1 = torch.mean(o1, 1)

        x = x1

        x = self.ln(x)
        x = self.dropout(x)

        logits = self.classifier(x)

        return logits
```

THE MAGIC

The difference between CV and and LB gave me a hint on how to post process the scores to overfit the leaderboard. Unlike @huikang i was giving subtle hints to others on how to increase their score. But the trick is this:

test is the output of your model after softmax.

```
factor =
torch.tensor([0.11388,0.02350,0.06051,0.39692,0.40519])/test.mean(dim=0)
out = (test*factor).argmax(dim=1)
```

What does this do? It maps the distribution of the outputs to the distribution of public test set. Note that the distribution of the public test set is a 'hard' distribution while output model distribution is a 'soft' probability distribution. Here hard means each sample only takes one rating while soft means each sample takes a probability of each rating. I have also tried mapping from hard to hard but it did not work. Another thing that I tried was scaling the outputs by submitting the outputs of your model for each rating. For example 1s and the rest to some nonsense value to see how well your model scores for each category and then scaling it accordingly. This also does not work. You can try this magic and see an immediate boost!

Training code is here. There are some modifications to the final training code which I used colab to run. But mainly it is the additional data provided by Tony. Thanks for sharing!

https://www.kaggle.com/garyongguanjie/train-sentiment-cased-xlm

Options

Comments (6)

Sort by
Hotness

Tong Hui Kang • (4th in this Competition) • a day ago • Options • Reply 1

My function was

```
target_distirbution = [0.11388, 0.02350, 0.06051, 0.39692, 0.40519]

def scale_by_multiply(ratio):
    return np.argmax(np.multiply(test_preds_original, ratio), axis=1)

modification_function = scale_by_multiply  # change this if you want

def calc_class_share(test_class):
    c = collections.Counter(test_class)
    return [c[clf]/len(test_preds) for clf in range(5)]

def loss(ratio):
    test_class = modification_function(ratio)
    class_share = calc_class_share(test_class)
    loss = [abs(x-y) for x,y in zip(class_share, target_distirbution)]
    return sum(loss)

from scipy.optimize import minimize
res = minimize(loss, [1,1,1,1,1], method="Nelder-Mead")
res.fun, res.x  # loss
```

`test_preds_original` is the probabilities for each data.

It optimises the ratio to multiply for each probability class until the class distribution fits.

How much did your improve by mine was 0.67 to 0.72

Tong Hui Kang • (4th in this Competition) • a day ago • Options • Reply 0

The accuracy of model predictions have a high variance, my best results happen when the 5s is overwhelming preferred over 4s.

My score improved from 0.45 to 0.68.

slm37102 • (20th in this Competition) • 9 hours ago • Options • Reply 0

@garyongguanjie Thanks for sharing too!! I just started learning these, still have many things to learn from you all.

Samuel • (98th in this Competition) • a day ago • Options • Reply 0

@garyongguanjie Hello, thanks for sharing really nice work! I'm curious about the performance gap between 0.48(cv on roberta-xlm) and 0.67 you mention in the comment. How did you improve such a big performance gap.

garyong Topic Author • (1st in this Competition) • 12 hours ago • Options • Reply 1

Roberta_XLM with Magic:0.53103

Roberta_XLM with Tony's data: 0.67258

Roberta_XLM with Tony's data and magic and Label Smoothing: 0.72422

I still think Tony's data is leaky somehow some information was leaked! Might do some investigation if i have time.

Label smoothing improved CV slightly but i did not make separate submission for it.
Code for label smoothing:

```
class LabelSmoothingLoss(nn.Module):
    def __init__(self, classes, smoothing=0.0, dim=-1):
        super(LabelSmoothingLoss, self).__init__()
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.cls = classes
        self.dim = dim

    def forward(self, pred, target):
        pred = pred.log_softmax(dim=self.dim)
        with torch.no_grad():
            # true_dist = pred.data.clone()
            true_dist = torch.zeros_like(pred)
            true_dist.fill_(self.smoothing / (self.cls - 1))
            true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        return torch.mean(torch.sum(-true_dist * pred, dim=self.dim))
```