This file documents what I have done during the internship project at NCI, as well as some ideas that I haven’t got to try, which may be helpful for those who will take over (if any).

## What I have tried

### Bert & its variants

I tried Distilbert, Bert-base and Roberta. All these models are similar in structure, and are reported to have similar performance (according on their respective papers). And it turned out that their performance on our text classification task is similar.

I started with a pretty complex header (**Structure 1**), which consists of

1) A sublayer that Sums up the last-layer outputs using attention (see AttentionPooling)
2) A sublayer that extracts some residual features to complement the output of the first sublayer, whose structure is quite similar to a transformer (see TransformerPooling).
3) A fully connected layer that maps the sum of outputs of the first 2 sublayers to logits

Because I thought classifiers of different labels would generally attend to different parts of the output, the first 2 sublayers are actually duplicated for each of the labels. I wish such duplication (and therefore isolation) could prevent the predictors of minority classes from being over-influenced by the training data of majority classes.

The resultant weighted (by frequency) average of f1scores of all the labels on the testing set (10% of the labelled data) were always around 0.58, regardless of which model was used.

The first sublayer was later removed (**Structure 2**) without affecting the performance (by f1score), although it seemed that it would take more steps to train the model to achieve the same performance after the removal.

I later saw that the output hidden state corresponding to the first token in Bert-based models can be directly used as a feature vector for classification, so I tried directly using that with a simple logistic regression header for classification, which yielded terrible results.

My final structure of header for Bert-based models adds the hidden state of the first token to the output of the TransformerPooling layer, the sum then goes through a fully connected layer before being mapped to logits of the labels.

The final performance of all the aforementioned headers is actually quite similar, and the final structure is chosen only because its structure is relatively simpler (compared to **Structure 1**). Also, when a weight regularization is imposed, the TransformerPooling layer would in theory perish (which is not the case for

**Structure 2**), if the hidden layer of the first token were a perfect feature vector for the classification problem.

### Training strategy

Since the structure of headers are reasonably complex, I fixed the weights of the model for the first several epochs of the training, and then reduced the learning rate and fine-tuned both the model and the header together. The f1score on training set can actually go all the way up to more than 0.9, while on testing set it plateaus out at somewhere around 0.58.

### Evaluation

I took a look at the f1scores for different labels and realised the predictor was only doing well on the label **Clinical & Health**, which was the absolute majority of the dataset (2/3 of the dataset has that label). I attributed the poor performance on other labels to the lack of training data, so I decided to try some newer models which are trained to have more common knowledge to offset that.

## Flan-t5

Google's Flan-t5 is one of the newest models publicly available and it was finetuned in a pretty interesting way (Instruction Finetuning), which seems to provide it with more common knowledge. Instruction finetuning also enables the model to conduct specified tasks in the input, so we don't need to get the model used to our classification task from scratch. Other preferable properties of Flan-t5 include that

(1) it has no hard limit on input length due to its usage of **Relative Position Encoding**,

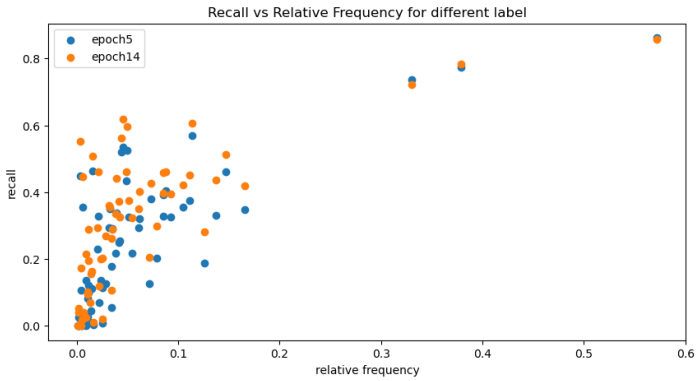
(2) it is an **Encoder-Decoder** model, which makes it easy for us, by tweaking the input to the decoder, to enforce the shape of the output hidden states, and subsequently to feed them to the header.

Due to the time constraints, I only adopted the simplest construct, i.e. a single starting token. The output hidden state directly goes into a linear layer and gets converted into the logits.

## Training Strategy & Evaluation

Since the header is really simple, I directly trained the model together with the header. After 4 epochs the average (weighted by frequency) f1score will be around 0.63. I found that the model tend to have decent precision on predicting the minority labels, while the recall is pretty low, and there seemed to be a **positive correlation** between the **frequency of a label** and the **recall of the that label**. I interpreted that as a sign of **undertraining** for the minority predictors. This undertraining is understandable, due to the usage of Conditional Loss, because the output of the minority classes got masked out most of the time and got updated much less frequently compared to that of the majority classes.

I then removed some datapoints of the majority classes (for more efficient training) and continue training the model for 10 epochs, and the recall of many of the minority classes saw significant increase, as shown below



Most of the increase in recall came at no cost of precision, some of the labels even saw non-trivial increase in precision as well. This result verifies my aforementioned conjecture on undertraining.

However, the precision of the majority classes decreased quite a bit (e.g. precision of **Clinical and Health** went down from 0.835 to 0.794), which ended up neutralising all the improvement (in terms of f1score) in those minority labels, as the distribution of labels was extremely unbalanced.

That said, I still think the further training was meaningful as the minority classes shall not be ignored and sacrificed for our problem of literature analysis.

## Potential Next Steps

### More Complex Header and/or output states

Since Flan-t5 has successfully out-performed Bert-based models, it would be better to stick with it and try further exploiting it.

It is worth trying asking the model to generate longer outputs and/or using hidden states from more than only the very last layer. A more complex header may be adopted as appropriate in these cases.

### Larger Model and Longer Input

All my results are based on the smallest version of Flan-t5, which is only about 300 MB on disk, and is even smaller than most of the Bert-based models. It is generally expected that larger models would result in better performance. I attempted to get the xl version running but failed because I didn't know the existence of the dgx queue. When I realised that, we were almost running out of Service Units and I decided to leave them to other interns (~~also because I was being lazy~~).

The training script uses the 🤗 Accelerate framework, which can automatically deploy the resources appropriately to accelerate training, as long as it is given sufficient resources.

Apart from larger model, allowing longer input token sequence would also worth trying.

### More Minority Data

You should have been convinced that more data generally results in better performance by far, so getting more training data should also be an option to consider.

My suggestion is to collect more data of the minority classes for training, and these can also not be related to Covid. If collecting data is not possible, you could also try applying some data augmentation on the minority classes to fabricate more data yourself.

### More Diverse Training Tasks

The improvement in performance of Flan-t5 is generally attributed to (at least partially) the diversity of its finetuning tasks. Currently the model is using a specific header for classification, but you may also want to try going back to the original text-to-text frame work and train the model with more tasks than simply predicting the labels