Table of contents

1. Team Introduction

1.1. Competition Summary

- Competition Name: Coleridge Initiative - Show US the Data
- Team Name: ZALO FTW
- Private Leaderboard Score: 0.583
- Private Leaderboard Place: 0.576
- Ranking: 1st

1.2. About Us

Our team is named ZALO FTW and consists of two members who are currently working at VNG Corporation, Ho Chi Minh City, Vietnam:

- **Nguyen Tuan Khoi** (Khoi Nguyen): Senior Data Scientist at Zalo, VNG. Kaggle Account: https://www.kaggle.com/suicaokhoailang
  I have a bachelor degree in computer science from Hanoi University of Science and Technology. I have four years of experience working in the field of machine learning, especially with text data and language models. My current job revolves around solving NLP problems in Zalo, one of the most popular Vienamese OTT applications.

- **Nguyen Quan Anh Minh** (Minh Nguyen): AI Engineer at Zalo AI, VNG
  Kaggle Account: https://www.kaggle.com/dathudeptrai
  I graduated from Ho Chi Minh University of Technology with a major in computer science. I've been working in the field of machine learning for approximately three years, especially with CV, NLP and Speech domains. My present position at Zalo involves integrating AI products such as Text to Speech (TTS) and Voice Biometric. . I am also the creator of TensorSpeech/TensorFlowTTS, the largest Text To Speech framework written by TensorFlow.

2. Solution Summary

2.1. Overview

Our winning solution was a metric learning model with a transformer backbone. Precisely it was an ensemble of two models based on two architectures: BioMed-RoBERTa (Gururangan et al.) and SciBERT-base (Beltagy et al.).

The reason why our solution stood out is because it was able to overcome the problem of label scarcity within the training data by focusing on the context surrounding the labels instead of the labels themselves. Although the number of unique labels was very limited, the contexts in which they were introduced were indeed rich, which allowed the model to generalize. In our observation there are many different ways a dataset can be introduced, and thus, for each input, which we call the query, the model will look into a randomly selected set of support samples to see if there exists a pattern in the input similar to those in it then extract the information accordingly. Here, a support sample is a chunk of text that contains annotated dataset title(s).

We also employed a data cleaning method, which got rid of a large amount of false-negative samples within the training set.

We used TensorFlow to implement our solution. It took around 2 hours to train a single model on a Nvidia V100 GPU with 16GB VRAM.

2.2. Data Preparation

2.2.1 Data cleaning

It was clear from the beginning that the training data was not fully labeled. We thought of two possible solutions to this:

1. Manually label the rest of the data to gain more positive samples, which is extremely time consuming and hard to reproduce.
2. Focus on the originally labeled samples, then find a high-recall method to clear out any samples that are possibly unlabeled. We followed this option instead.

A quick glance at the training data showed that many datasets contained keywords such as “Survey”, “Study”, “Dataset” or their variations. Also, the first time a dataset is introduced, it usually follows the: “FULL NAME (ACRONYM)” formula e.g. “National Education Longitudinal Study (NELS)” . With that in mind, we decided to search for that specific pattern within the data and masked samples that contained them to be possible false negatives and unfit for training.

2.2.1.1. Detecting possible false-negatives

We extract candidate datasets in two ways, as shown below, and use them in the training/validation process:

1. We used AbbreviationDetector from scispacy to detect all capitalized strings in the form of "FULL-NAME (ACRONYM)" then only selected ones that contained specific keywords (Dataset, Database, Study, Survey .etc).
2. (Optional) We detect the keywords (Dataset, Database, Study, Survey, ...) position in the input string then look forward/backward from that keyword until it meets two consecutive lowercase words. We used the label extracted from this custom algorithm in our latest submission but we found that it didn't affect the result much (let say around 0.x %).

For all found candidates, only ones that have a Jaccard similarity of 0.5 or greater with any of the original train labels will be used for training, the rest will be passed to validation.

A few example of the candidates that we've founded in the training data:

- Saccharomyces Genome Database
- National Institute for Educational Studies
- Michigan Study of Women's Health
- Institute of Legal Study
- Integrated Postsecondary Education Data Systems
- Program for Monitoring of the Greenland Ice Sheet
- ...etc

2.2.1.2. Labels normalization

In our observations, the labels could be divided into three categories:

1. FULL NAME
2. FULL NAME (ACRONYM)
3. ACRONYM

We tried to add more labels which we considered missing using the following rules:

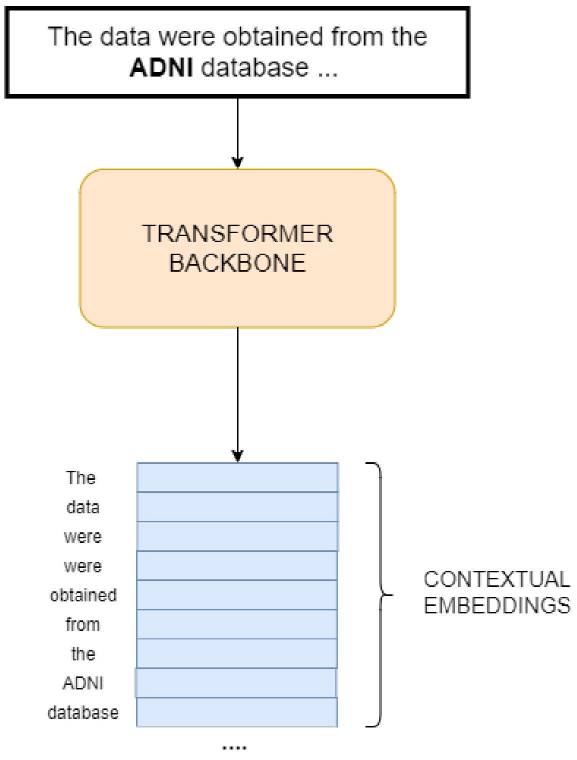
- If we found a match in the form of FULL NAME (ACRONYM), we would also add FULL NAME and ACRONYM to the list of labels.
- If we found a match in the form of FULL NAME only, we would try to find its corresponding ACRONYM and add it to the list of labels.

All of the matchings were done on the cleaned version of the training data and the labels. The cleaning function was provided by the organizers. A label is also invalid if its pre-cleaning form was in lowercase.

Finally, we would remove from training any sample that contained any label that belonged to both the training and validation set.

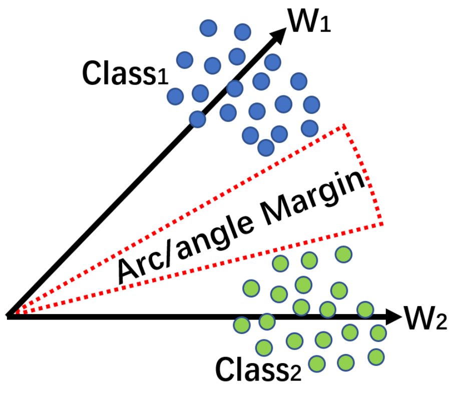
2.3. Modeling

2.3.1. The transformer backbone



A core component of our solution is the popular **transformer** (Vaswani et al.) model, more precisely the BERT (Devlin et al.) architecture, which was pre-trained to understand deep bidirectional representation of texts.
Intuitively, the model will be able to map each token in the input text to a vector (an embedding). This is different from methods like word2vec or GloVe, as in this case the representations of the same word will be altered based on the context surrounding it, that means each embedding contains not only the information about that specific word but the context itself.
This attribute proved to be extremely important for our solution and allowed it to generalize despite the lack of unique training labels.

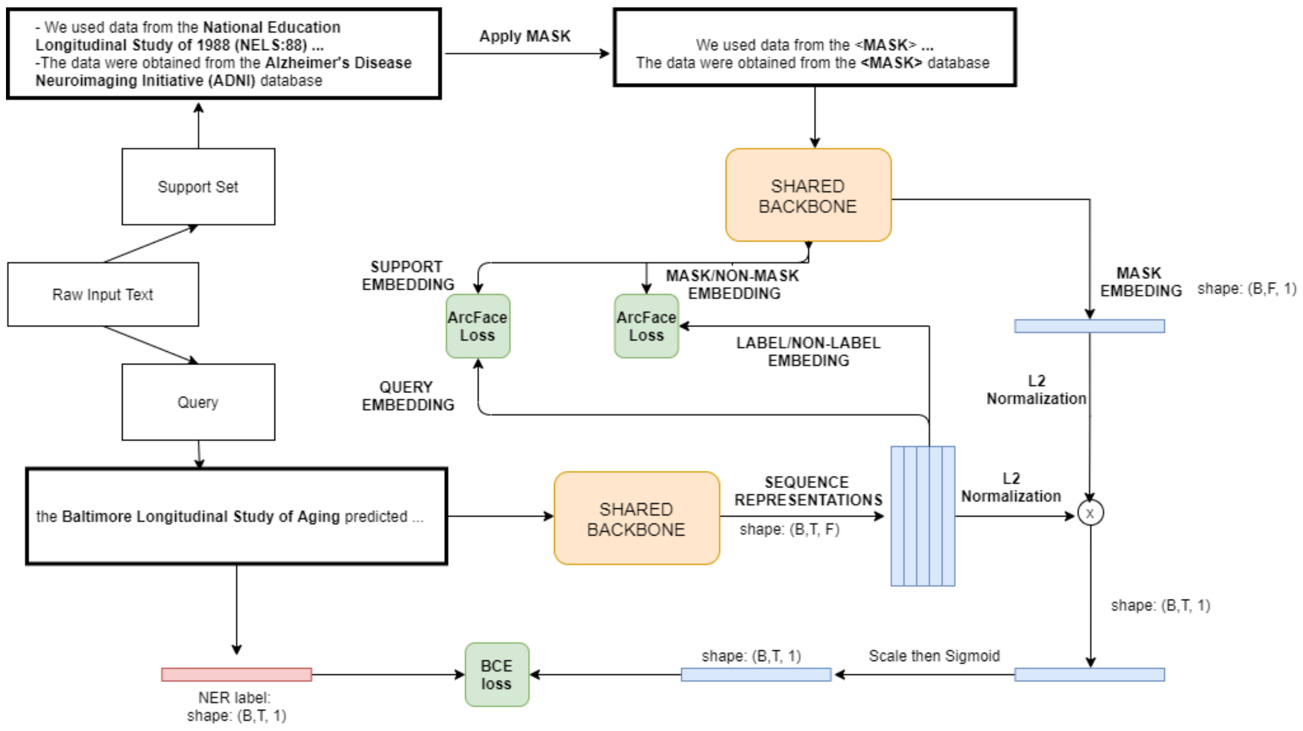
2.3.2. ArcFace loss function



ArcFace, or Adaptive Angular Margin loss (Deng et al.), is a loss function used in recognition tasks such as Face Recognition and Voice Identification. In these tasks, the softmax loss is traditionally used. However, the softmax loss function does not explicitly optimise the feature embedding to enforce higher similarity for intra-class samples (same class samples) and diversity for inter-class samples (different class samples), which result in a performance gap for recognition tasks under large intra-class appearance variations. Especially, in the case that the training set is small, the model tends to overfit and domain generalization cannot be achieved.

The ArcFace loss, on other hand, simultaneously enhances the intra-class compactness and inter-class discrepancy. As the above figure shows, the ArcFace loss can enforce a large evident gap between each class.

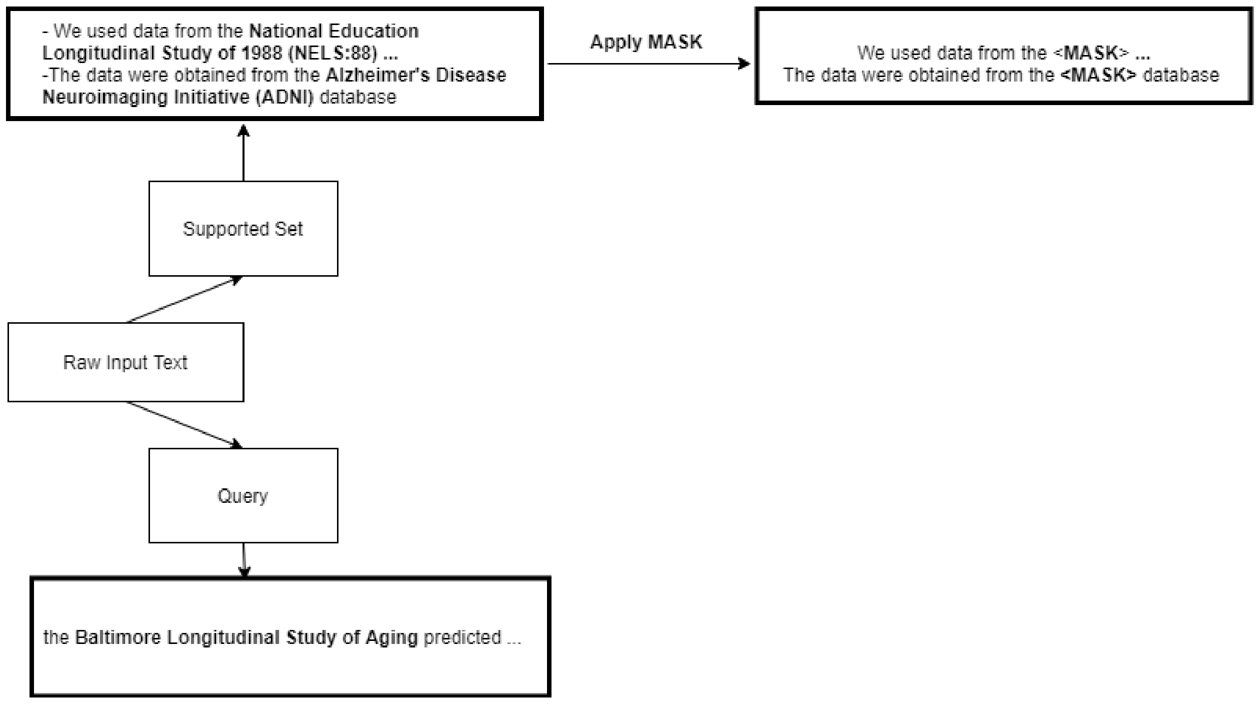
2.3.3. Model architecture



2.3.3.1. Terms

- **Sample**: The training data is split into overlapping chunks of 200 words (50 words are overlapped with nearby chunks), each chunk is a training sample.
- **Group**: Each unique label is a group, a training sample will belong to that group if it only contains that label. Any sample that contains more than one label will belong to a special group.
- **Masking**: The process of replacing certain input tokens with a special masking token, with BERT and other similar models this token is “[MASK]”.
- **MASK embedding:** The representation of the [MASK] token after feeding the input to a transformer backbone. If there were multiple masks, this would be the averaged value.
- **non-MASK embedding**: The averaged representation of all the non-[MASK] tokens after feeding the input to a transformer backbone.
- **LABEL embedding**: The averaged representation of all the tokens that belong to a dataset mentioned after feeding the input to a transformer.
- **non-LABEL embedding**: The averaged representation of all the tokens that do not belong to any dataset mentioned after feeding the input to a transformer.
- **QUERY/SUPPORT embedding**: The represent of the [CLS] token after feeding the query/support sample to a transformer. [CLS] is another special token that is typically used to represent the information of the whole input sequence.

2.3.3.2. The query and support set


- We used data from the National Education Longitudinal Study of 1988 (NELS:88) ...
-The data were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database
Apply MASK
We used data from the <MASK> ...
The data were obtained from the <MASK> database
Supported Set
Raw Input Text
Query
the Baltimore Longitudinal Study of Aging predicted ...

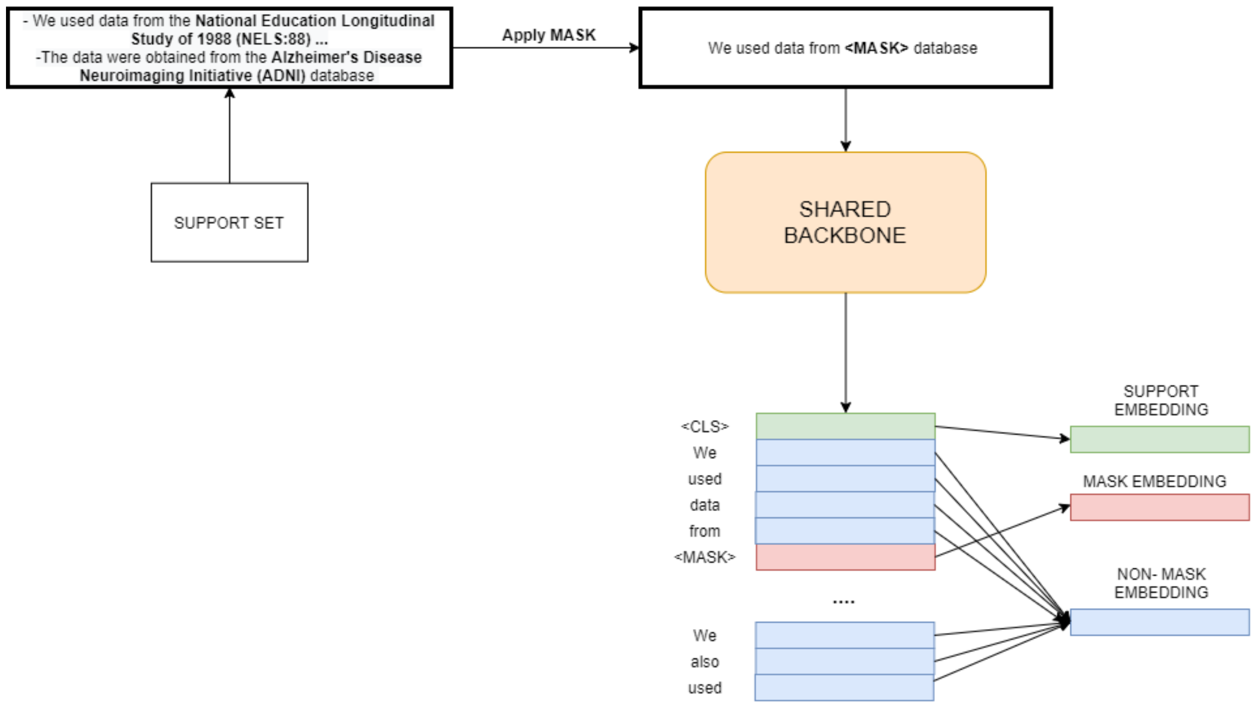
Each training step involves a query and a supported set which are randomly selected from the available training samples:

- There will always be 1 query sample and K support samples, in our training we used K = 3.
- The support samples are always positive, which means they belong to at least one group. On the other hand the query sample is set to be positive 50% of the time.
- Each of these K + 1 samples' groups must be non-overlapping.

The intuition behind this is as follow:

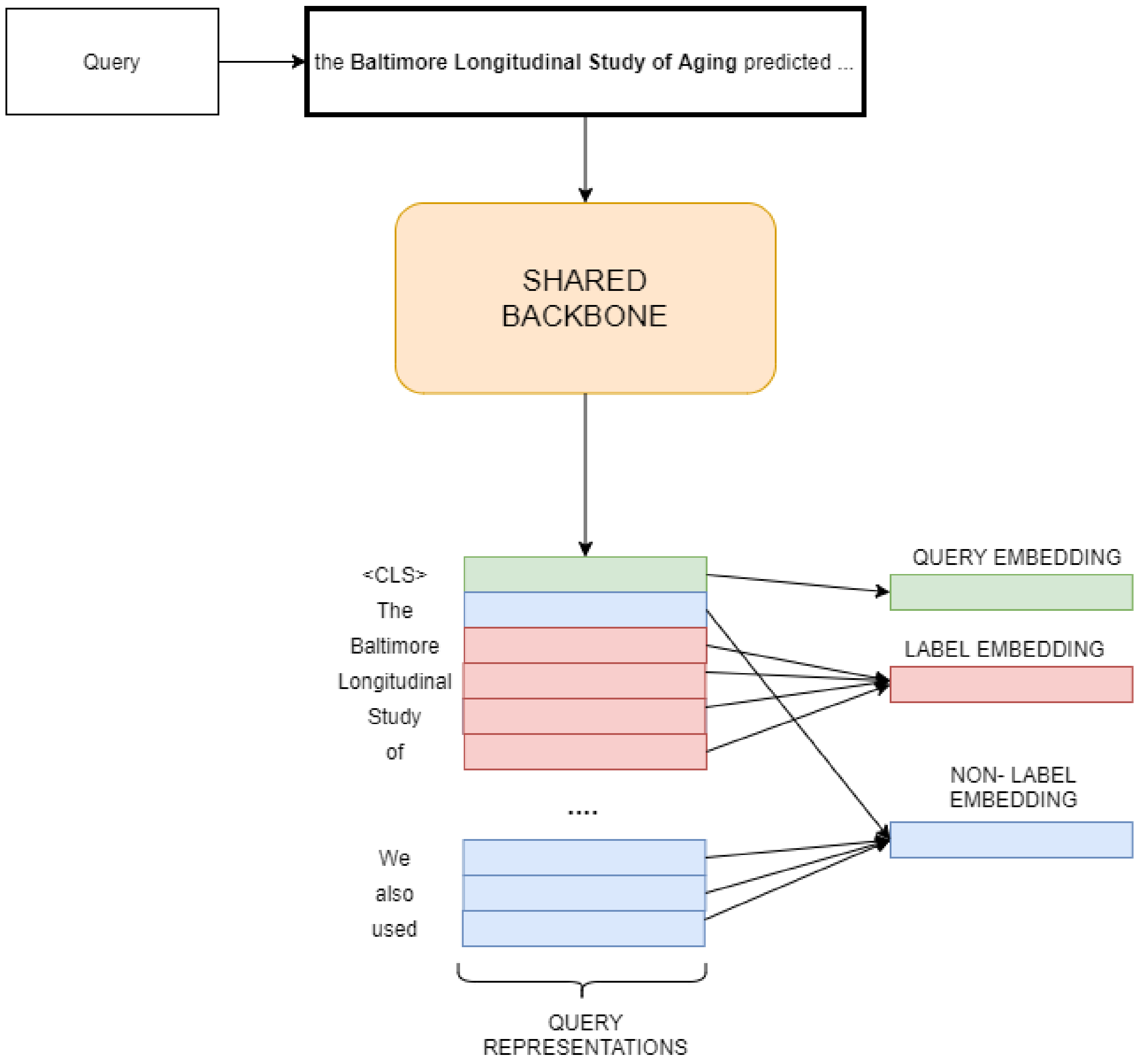
- The query is our interest, it's a chunk of text that potentially contains the label we want to extract.
- The support set is a set of samples that were known to contain the label.
- With a query, we would want to look at the support set to determine if the context in the query is similar to the known contexts in it.
- In real world use cases, the query will come from new, unseen data, while the support set is randomly sampled from annotated training data. The support set's role is to help guide us in the extracting process, hence the name.

2.3.3.3. Embeddings extraction


- We used data from the National Education Longitudinal Study of 1988 (NELS:88) ...
-The data were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database
Apply MASK
We used data from <MASK> database
SUPPORT SET
SHARED BACKBONE
<CLS>
We
used
data
from
<MASK>
....
We
also
used
SUPPORT EMBEDDING
MASK EMBEDDING
NON- MASK EMBEDDING

With the support set:

1. We mask every mentioned label with a [MASK] token and feed these K samples to the backbone.
2. We now have K representations, one for each support sample, each has shape of (T, V) which T is the sequence length and V is the size of the embedding:
    a. For each sample, we obtain the MASK-embedding and non-MASK embedding as explained in 2.3.3.1.
    b. We then take the average of MASK and non-MASK embeddings from all the support samples to obtain the final MASK and non-MASK embeddings for the support set.



With the query:

1. We do not apply masking here, instead we will obtain the LABEL and non-LABEL embeddings as explained in 2.3.3.1. Note that if this query is negative then there won't be a LABEL embedding.
2. We also keep the full sequence embedding for use later.

We now have extracted these representations:

1. MASK embedding from the support set
2. non-MASK embedding from the support set
3. The SUPPORT embedding
4. LABEL embedding from the query
5. non-LABEL embedding from the query
6. The QUERY/SUPPORT embedding
7. Full sequence representations of the query.

2.3.3.3. Applying criterions

We apply ArcFace loss to two pairs of embeddings:

- **SUPPORT MASK** and **QUERY LABEL** (if available) are considered class 0.
- **SUPPORT non-MASK** and **QUERY non-LABEL** are considered class 1.

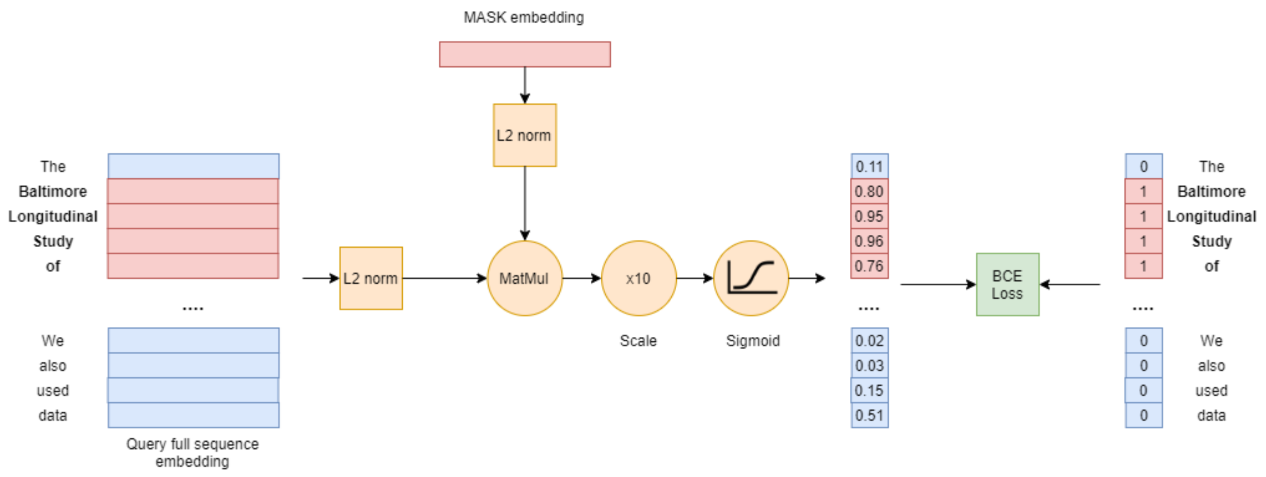
The intuition behind this is as follow:

- The MASK embedding should contain the information about the [MASK] token itself, as well as the surrounding context.
- Likewise, the LABEL embedding should contain the information about the label tokens themselves, as well as the surrounding context.
- Since [MASK] is a generic token that has no distinct meaning by itself, we assume that the MASK embedding would contain a rich amount of information about the context.
- By forcing the MASK and LABEL embedding to be close together, we were forcing the model to attend more to the surrounding context when looking at every label token.

We also apply ArcFace loss for these embeddings:

- **SUPPORT embeddings** are considered class 0
- **Positive QUERY embeddings** are considered class 0
- **Negative QUERY embeddings** are considered class 1

In our case, we want to use [CLS] token embedding to answer the question: "whether there exists a dataset title in the sentence or not?". If the sentence contains a dataset title, its feature should be closer to class 0, otherwise, it should be closer to class 1.



Now we go to the primary objective, which is **token classification.**

1. We compute the cosine similarity between the extracted MASK embedding and each token embedding from the query's full sequence embedding.
2. The result will then be multiplied by 10 and then normalized by sigmoid function, the rescale is to ensure the output range to be (almost) in the range of (0,1).
3. The ground truth is a binary vector with every token that belongs to a dataset name denoted as 1 and the rest as 0.

4. The BCE loss is then applied for the output and the groundtruth, and a training step is completed.

2.4. Training

2.4.1. DataLoader parameters

Below are some parameters we used for Data Sampling:

- batch_size: 4
- K (number of support samples / 1 query sample): 3
- training steps per epoch: 1000
- number of training epoch: 10
- maximize number of tokens per sample: 320

Note that we only used 1/8 of negative samples from the training dataset to train a model and did not set a random seed to make a better diversity for ensembling later.

2.4.2. Optimizers

We used AdamWeightDecay with Warm Up Polynomial Decay (see Hugging Face's transformers documentations):

- initial_learining_rate: 3e-5
- end_learning_rate: 1e-5
- num_warmup_steps: 60
- weight_decay_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- epsilon: 1e-8
- exclude_from_weight_decay: ["LayerNorm", "layer_norm", "bias"]

2.4.3. Augmentations

To make our model more robust, we used a few form of augmentations:

- **Random label swapping**: 50% of the time, the original dataset label will be replaced with one that came from an external dataset, which is a combination of labels from RCDatasets and the old competition dataset. Note that we only use labels that we considered hard, which are the ones that do not contain keywords such as Dataset, Database, Study, Survey and its variants.
- 50% of the time, we also do one of the followings:
    - **Random lowercasing**: We convert the dataset name to lowercase but keep the ACRONYM intact.
    - **Random word dropping**: We drop the last word from the dataset name if it was an ACRONYM.

2.5. Inference

2.5.1. Making predictions

Inference is a bit more simple since we can ignore the ArcFace route:

1. We extract every possible MASK embedding from the training set.
2. We extract every support embedding (positive samples) from the training set.
3. We then treat each test sample as a query:
    - Compute the query's sequence representations.
    - Compute the query embedding.
    - Randomize K samples to create a support set and then compute its MASK embedding.
    - Randomize K support embeddings.
    - Compute the cosine similarities between the query embedding and the K support embeddings then average them. This cosine value is used to classify if the query sample is positive or negative in 2.5.3.1.
    - Compute the cosine similarities between the MASK embedding and the query embedding, scale up and apply sigmoid to get the model's prediction.

The model's prediction is a vector of shape (T,) which T is the sequence length. We then apply a threshold **TH** to get a binary vector and then take any sub-sequence of True values whose length is at least 4.

The threshold **TH** we used for the inference is divided into two intervals:

- **TH >= 0.7**: We get all predictions in this interval.
- **TH >= 0.5**: We only accept predictions that contain keywords such as study, survey, database, ...etc.

2.5.2. Ensembling

In our experiments, we found that our single models have low **FP** but high **FN** in the evaluation set. Moreover, the **FP** predictions between models are quite overlapping. This is a reason that we decided to ensemble multiple models by unionizing their predictions.

Our latest submission was an ensemble of BioMed-RoBERTa and SciBERT-base.

2.5.3. Post-processing

2.5.3.1. Valid prediction rules

Our models can predict a lot of dataset titles with different forms but not all of them are good. We used a few rules to filter out bad predictions:

1. **Incomplete words**: For example, if the input string is **"**We used the data from the National Study of Youth**"** and our predicted label is **"**al Study of Youth**"**.

2. **Short predictions**: We applied the cleaning function that was provided by the organizers and removed the first word of a prediction if it's a stop word (a, an, the).Then, we removed all predicted labels that have word count less than 3 and character count less than 10 .e.g:
    - The National Study.
    - National Data
    - The Survey of
    - ...etc
3. **Negative Query**: In the case our model classifies the query sample as negative, we only accept predictions that contain keywords such as study, survey, database, ...etc.

2.5.3.2. Thresholding prediction frequency

The intuition is that if a *real* dataset is mentioned in a paper, it's likely to be referenced by other papers too. The more frequent a prediction occured, the more likely for it to be valid dataset title. Based on this idea, we introduced a threshold **C,** which means if our model can detect a dataset at least **C** times then we consider that label to be a confidence label, otherwise, it's a low confidence label. For a confidence label, we just apply the "valid prediction rules" as shown above. For low confidence labels, we also apply the rules but only get the ones that contain keywords such as study, database, dataset, survey, ...etc.
In our latest submission, we used **C** equal to 2. We also noticed that the **C** in range (1, 4) didn't affect our result much (less than 1%), which demonstrated the stability of our model.

2.5.3.3. Dataset lookup table

As explained, a dataset could be referenced by multiple papers. Sometimes, however, it can only be detected in certain cases since the contexts may vary and are not always clear. For example, the dataset "National Study of Youth" is referenced from paper A, B and C but our model could only detect it in paper A, so we should also add it to the predictions of B and C. Based on this idea, we built a lookup table of all predicted datasets and searched for all of them for each paper.
Note that, in the searching process, we only accept the non-lowercase titles. Sometimes, there are multiple overlapped predictions of the same dataset in one paper, in this case, we only get the longest prediction. For example, if the dataset lookup table contains two predictions "National Study of Youth", "National Study of Youth data" and the input string is "The National Study of Youth data provided data for ..." then we only accept the longer prediction, that is " National Study of Youth data". In this case, "National Study of Youth" is considered a False Positive prediction.

2.5.3.4. Acronym detection

As mentioned in 2.2.1.2, the labels could be divided into three categories: FULL NAME, FULL NAME (ACRONYM) and ACRONYM. We recognized that if we detected a FULL NAME then we could also detect FULL NAME (ACRONYM) and ACRONYM. We tried to add all categories of every predicted dataset but the score dropped significantly. We could not find a definitive answer to this problem since we're not clear how the annotation process worked.

For example, in the paper A, the same dataset was independently referenced in three ways:

- National Study of Youth
- National Study of Youth (NSY)
- NSY

We think the possible ground truths can be:

- National Study of Youth | NSY
- National Study of Youth (NSY) | NSY
- National Study of Youth | National Study of Youth (NSY) | NSY

We tried all 3 cases and found that “National Study of Youth | NSY” got the best result in public LB, thus, in our latest submission, we only took the FULL NAME and searched its ACRONYM counterpart then added it to the prediction if it’s available.

2.6. Interesting Findings and Hindsights

2.6.1. Attempts to create an alternative leaderboard

During the competition, it became clear that the public leaderboard was highly unreliable since a large portion of public labels could be found in training. Indeed, a simple string matching method scored a very hard-to-beat baseline of 0.57.

To better assess our *real* performance, we used a simple post processing function to remove any prediction that has a Jaccard similarity of 0.5 or greater to any of the known train labels; we also encouraged others to do the same. The scores shared in our thread showed that our true performance was very good compared to other competitors, this motivated us to stick to our approach and it finally paid off.

3. References

AllenAI. “ScispaCy.” https://github.com/allenai/scispacy.

Ashish Vaswani, et al. *Attention Is All You Need*. https://arxiv.org/abs/1706.03762.

“Hugging Face's transformers documentations.”
https://huggingface.co/transformers/main_classes/optimizer_schedules.html.

Iz Beltagy, et al. *SciBERT: A Pretrained Language Model for Scientific Text*.
https://arxiv.org/abs/1903.10676.

Jacob Devlin, et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. https://arxiv.org/abs/1810.04805.

Jiankang Deng, et al. *ArcFace: Additive Angular Margin Loss for Deep Face Recognition*.
https://arxiv.org/abs/1801.07698.

Suchin Gururangan, et al. *Don't Stop Pretraining: Adapt Language Models to Domains and Tasks*. https://arxiv.org/abs/2004.10964.

.