Data Science Technical Exercise

Submitted by: Satya Sai Srinath

Problem statement (high-level): Given some data about a job (in form of job description), get the top K ONET classes

Approach (high-level):
At the core, the ideas of sentence-embeddings, contrastive-loss and FAISS nearest neighbors are used. The motivation to formulate the problem in this manner can be inspired from my previous experiences and also from CLIP training (train embeddings jointly for close/relevant data, for a new data point, the embedding is useful to classify)

Approach (low-level details):
As suggested, I used sentence-transformers/all-MiniLM-L6-v2 model for this assignment!

First, I constructed a pair of input sentences i.e BODY and it's corresponding ONET_NAME as part of preprocessing for sentence-transformers. To finetune, I used "Multiple Negatives Ranking Loss" as we don't have label scores for this pair of sentences. And we don't currently have negative samples.

Once the model is trained, I create embeddings for unique ONET_NAME and build a FAISS index. This will be used to retrieve the top-10 ONETs for every test BODY.

It takes around 40 sec to train for 1 epoch for a batch size 256. Also, a notebook is attached which has some exploratory data analysis!

Testing: When a new job description is given i.e BODY from test_data, I generated embeddings for this test data point (using the fine tuned sentence transformer) and fetched the 10 nearest neighbors (using FAISS index).

The intuition is, as the model is trained jointly on BODY and ONET_NAME, it tries to generate embeddings that are close for a positive pair. This contrastiveness can be used to generate the embeddings for BODY during test time and look for nearest ONET_NAME embeddings to get the top categories. This is reflected in evaluation as well!

Evaluation: I evaluated using standard accuracy where if the actual ONET_NAME is present in the top-10 predicted ONET_NAME for a test-input, it's considered as correct. Here I report the training and test accuracies for different settings.
(No fine-tuning) Original sentence transformers model: 53.86%, 52.94%
Fine-tuned for 10 epochs: 73.68%, 70.5%
Fine-tuned for 50 epochs: 93.82%, 82.30%

The Github repo has the following files: Implementation code(train.py), some exploratory analysis (analysis.ipnyb) and model checkpoints! The data is not placed as it's a public repository!
Summary: By fine tuning (for just 10 epochs) the sentence transformer using a loss function specific for this example, we are able to observe a boost of ~20% in performance in terms of accuracy!

Next steps:

1. One easy way is to train it for longer to better align the BODY and ONET_NAME thus generating better embeddings and thus better accuracy. We can also plan to optimize this further by including better models, better hyperparameters.
    a. Experiment with different loss functions: One extension that can be interesting to experiment is to bring in negative samples and formulate it as a Triplet Loss. Or, having pseudo label scores for a pair of BODY, ONET_NAME and using this label score while fine tuning the model.

2. Evaluating and extracting the losses while fine-tuning these sentence-transformers is not trivial! (eg: https://github.com/UKPLab/sentence-transformers/issues/719 , https://stackoverflow.com/questions/75655918/sentence-transformer-training-and-validation-loss).
    a. So, I didn't spend much time evaluating the model during training but that's equally important as understanding the test metrics. As a proxy metric, I included the training accuracy for different intermediate epochs (I agree that increasing training accuracy across epochs doesn't exactly translate to reducing training loss, but in most cases, we can make a good assumption)

3. Accuracy can be defined more broadly (top-1, top-10) etc; giving more understanding on metrics implemented.

4. **Another approach** -
    a. We can formulate it as a text classification problem using transformers i.e train a classifier on BODY as input and ONET_NAME as output and for the test BODY, use the logits to get top-10 ONET_NAME. There's a whole space of models, hyperparameters to play with if we chose this approach.
    b. We can also formulate it as an entailment problem (the BODY will be the premise and ONET_NAME as hypothesis)
    c. Another approach can be is to use LLMs in zero/few shot setting with Chain of Thought! Basically using the zero/few shot capabilities of LLMs to classify the job BODY. But this brings in challenge on how to classify the BODY to one of ~700 ONET (LLMs might be good at choosing 1 in 4 options but not 1 in 700, maybe finetuning helps, but it won't be super intuitive)