2nd Place Solution - Learning Equality Challenge

Konrad Habel
habel.konrad@gmail.com

March 28, 2023

Contents

1 Links

The code for training is public available in the following GitHub repository:
https://github.com/KonradHabel/learning_equality

The inference code is available over Kaggle incl. all used models:

Leaderboard Prize:
https://www.kaggle.com/code/khabel/2nd-place-learning-equality-leaderboard-prize

Efficiency Prize:
https://www.kaggle.com/code/khabel/2nd-place-learning-equality-efficiency-prize

The translated training data are available over Kaggle:
https://www.kaggle.com/datasets/khabel/learning-equality-language-switch

For offline evaluation checkpoints trained on fold 0 are provided via Google Drive:
https://drive.google.com/drive/folders/102N-wVRLhzf9d5IL3r1HFdYCsCZqcNQF?usp=share_link

2 Introduction

This technical report describes the implementation for the 2nd Place Solution for the Learning Equality Challenge on Kaggle.

2.1 Personal Background

My name is Konrad Habel and I am a Ph.D. student at the University of the Bundeswehr Munich in Germany. I have a german diploma in Automotive Engineering, a Bachelor of Science (B.Sc.) in Business & Information Systems Engineering and a Master of Science (M.Sc.) in Computer Science.

3 Implementation Details

Solution is a single stage approach based on cosine similarity for retrieval using the InfoNCE loss [2] as training objective. For embedding creation the same text encoder in form of a transformer is used for both topic and content. The solution is a simple retrieval model, without any further post-processing or second stage re-ranking.

Input Data: No use of special token for separation instead, the # is used as separator.

Topic: Title # Topic-Tree # Description

The topic tree is reverse ordered and the same separator # is used:
Title # Parent # Grandparent # ... # Description

Content: Title # Description # Text (cut to 32 based on white space splitting)

If for example the Description is empty the model will see as input: Title # # Text

For both **Topic** and **Content** a maximum sequence length of 96 tokens is used.

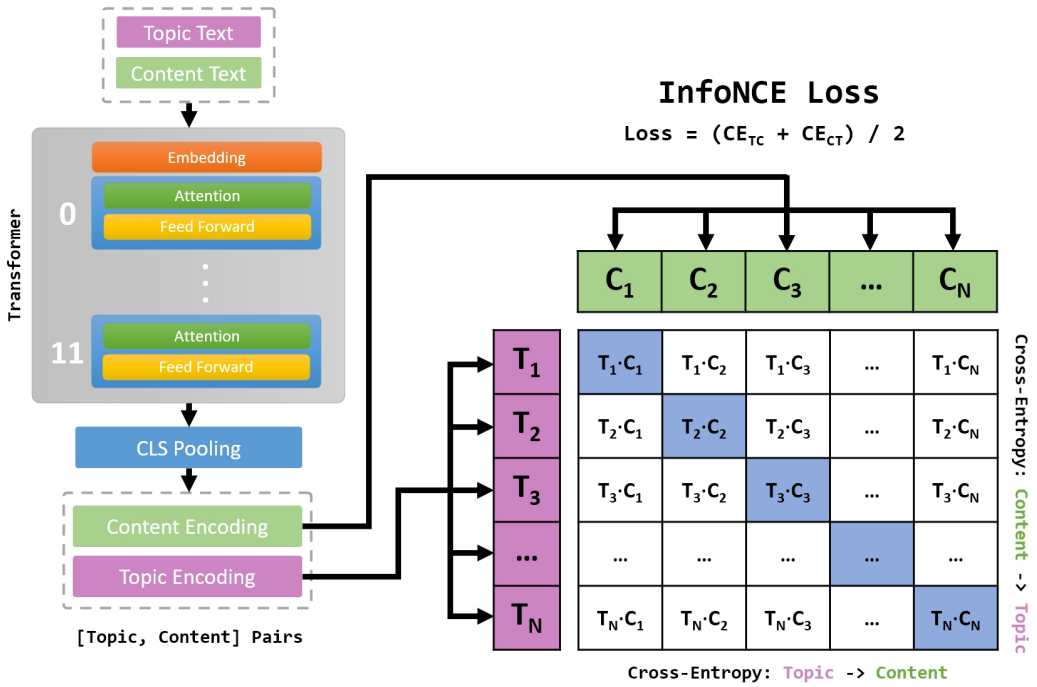

Figure 1: Model Pipeline and Training Objective

3.1 Training Objective

When using InfoNCE as training objective, we want a matches in form of highest similarities only on the diagonal of the similarity matrix, when calculating Cross-Entropy in both directions $Topic \rightarrow Content$ and $Content \rightarrow Topic$. Because we have here a $n \times m$ matching problem we have to take care, not to sample pairs for the same topic with different content and also not to sample related content into the same batch during batch creation.

For example if we would have two topics in the same batch that might share the same content we end up with high similarities besides the diagonal of the matrix whereas the label only says the high similarity on the diagonal are a correct match. On possibility to circumvent this problem is using label smoothing for the cross-entropy loss. But, having topics with related content in the same batch is simply noise for the model and should be avoided.

The easiest way in Pytorch when using DP and not DDP is to write your own custom shuffle function and set shuffling in the data-loader to false, what means nothing else than using a sequential sampler.

My own custom shuffle function in the dataset simply calculates before each epoch the composition of the batches and avoids sampling topics with related content in the same batch. Additionally after each epoch predictions for the whole training data are calculated to detect for each topic content that we miss and content that would be incorrectly assigned to that topic.

- **Missing Content** is stored in a list and the specific pair (topic, content) gets oversampled during shuffling.

- **Wrong Content** retrieved for a topic is more difficult to solve. Let's say we have topic **t1** with the wrong content **c1** for that topic with a currently high similarity to that content. Now what we need is another pair **(t2, c1)** of our ground truth in the same batch, to push away **c1** from **t1** cause when using the InfoNCE loss all other **N-1** contents in the batch are negatives for that sample. This sampling strategy is highly effective cause we increase the margin to all negative picks during the next epoch. Or at least we try, because if adding **t2** would lead to a conflict based on related content already in the batch, it is rejected for adding it in the current batch. This is done for a specific topic up to a maximum number of 128 hard negatives for that sample.

With that shuffle/sampling strategy we end up with batches without conflicts in related content or ambiguities, so we have only a correct match on the diagonal line and lots of near but incorrect content (hard samples) in the batch.

All models are trained for 40 epochs using AdamW [1] as optimizer and 0.1 as value for label-smoothing when calculating the cross-entropy in the InfoNCE loss.

3.2 Language Switching

I translated the most common languages **en, es, pt, fr** into each other for additional training data. But just adding this data to our training is not advisable when using InfoNCE, cause if we translate for example an topic and content item from English into French and a quite similar topic and content already exists in the original French content, we will end up again with noise during our loss calculation. So I recommend a simple switching strategy after each epoch, shown in figure 2.

Figure 2: Language switching after every second epoch.

When using a simple switching strategy, we can use the correlation.csv without having any trouble of ambiguities of translated to original content during loss calculation. Of course translation is never perfect and this alone creates some noise and further more we have a change in the distribution of the training data. That's the reason a switch is only used every second epoch and only between the languages **en, es, pt, fr** with good translation quality.

Using language switching leads to a score boost of up to 0.01–0.02 what is not that much as expected maybe due to the noise this introduces during training. It is also advisable to stop language switching for the last epochs and train at the end only on the original distribution.

For translation I tested different approaches. Best offline translation with on huggingface available models are in my opinion the MarianMT[1] models. Multilingual models were even worse. But in the end, I did a simple export in an .xlsx file where first column is a numerical ID and second column is what I want to translate and uploaded this to google translate → document translation. From my perspective google translate is much faster and generates more reliable results than MarianMT models and the file translate option is free to everyone over any web browser.

[1] MarianMT https://huggingface.co/docs/transformers/model_doc/marian

3.3 Knowledge Distillation (Efficiency Prize)

For Knowledge Distillation the already trained models on that task are used as teacher and half of the transformer layers are dropped. For that run the MSE-Loss is used as described in figure 3. Weights of the student are initialised with the weights of the pre-trained teacher model. Distillation leads only to a slightly drop in performance when using 6 Layers, so the sweet spot for me was not dropping more layers, but of course this is always a trade of speed vs. accuracy.

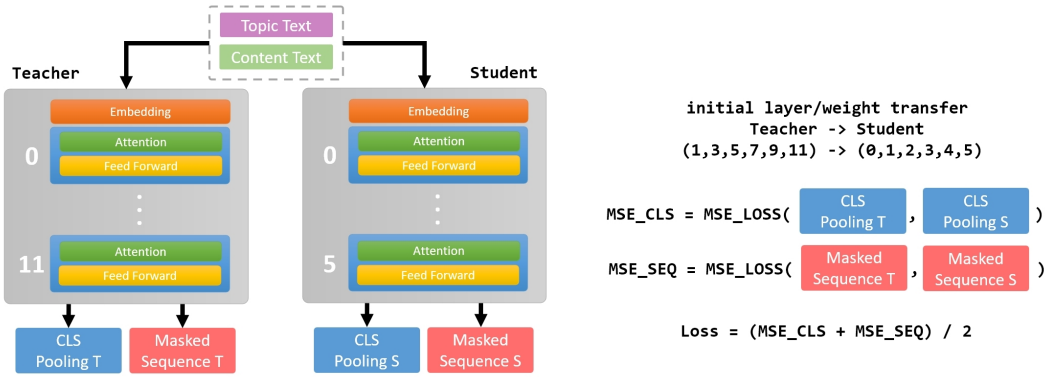

Figure 3: Knowledge Distillation using MSE-Loss as training objective to imtate the teacher.

3.4 Quantization + JIT-Trace (Efficiency Prize)

I used Torch Post Training Dynamic Quantization what is probably not optimal. Unfortunately, there is no pre-installed huggingface optimum in the CPU kernel so using Intel Neural Compressor or OpenVINO needs offline installation, and if every second counts it makes not much sense to waste time with installation of additional packages. But as shown in table 1 at least in case of scores quantization leads to a lower score drop than distillation.

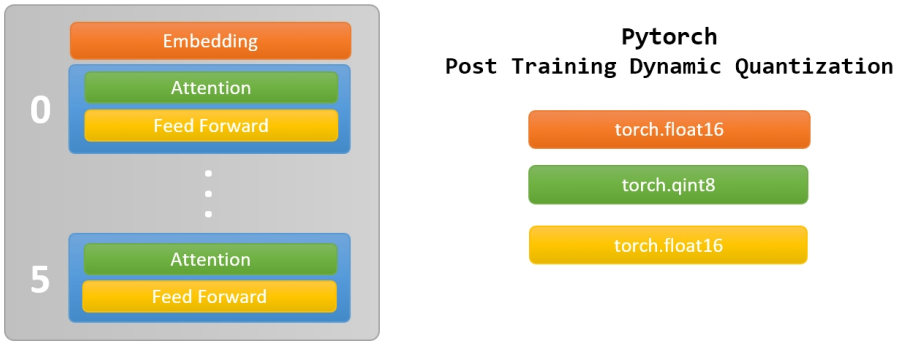

Figure 4: Quantization using Pytorch Post Training Dynamic Quantization.

Unfortunately FX Graph Mode Quantization of Pytorch did not worked out for me with the huggingface models, otherwise I would have tested quantization aware training for better results.

I ended up with just using **Eager Mode Quantization** → **Post Training Dynamic Quantization** and compiling it into a jit-traced model. This leads to a performance drop of around 0.01 on my cv-scores but increases the throughput and lowers the execution time on CPU. If using **qint8** also on the Feed Forward part of the transformer on the intermediate up sample or output layer, the score drop is even higher so I ended up in only using **qint8** on the attention layer. For applying qint8 also on the other parts maybe quantization aware training could be helpful.

3.5 Dynamic Threshold

Using a dynamic threshold instead of a static threshold leads to better performance up to +0.02 on the F2 Score during retrieval. This calculation using a $margin = 0.16$ is also used during training to find the hard negatives for the next epoch and during inference. The basic idea is described in figure 5.

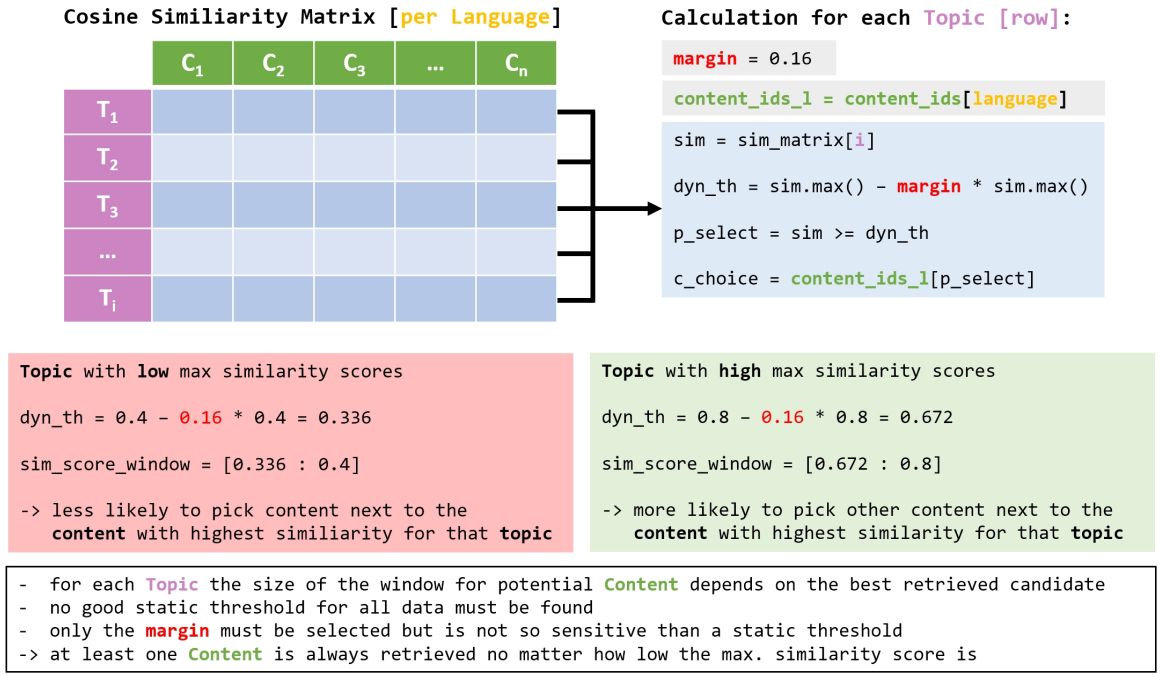

Figure 5: Dynamic Threshold calculation

The biggest advantage is, that this is much more stable than finding an optimal static threshold. Every model I trained before leads to a different optimal static threshold. With using this dynamic calculation the results was always the same and optimal values for margin are between $[0.14 : 0.18]$ on different folds. Using a higher margin leads to better recall by losing precision and because of using the F2 Score, perhaps on the test-set a higher margin leads to better results. During training I am using **0.16** as margin, for my best submission I used **margin=0.18** for inference.

4 Train and Evaluate

The code for training is available on GitHub: https://github.com/KonradHabel/learning_equality
Everything needed for training is described in the GitHub README.md and the configuration dataclass at the beginning of the scripts.

4.1 CV-Split

1. Download the data for the competition[2] and place it in: **./data/kaggle**
2. Download the data for language switching[3] and place it in: **./data/switch**

You should have the following data structure:

```
learning_equality
├── data/
    ├── kaggle/
    │   ├── content.csv
    │   ├── correlations.csv
    │   ├── sample_submission.csv
    │   └── topics.csv
    └── switch/
        ├── content_1.csv
        ├── content_2.csv
        ├── content_3.csv
        ├── topics_1.csv
        ├── topics_2.csv
        └── topics_3.csv
```

Figure 6: Data structure

Steps for data preparation:

1. create CV-Splits for offline evaluation: **cv_split.py**
2. data preparation and add folds to translated data: **data_preparation.py**

CV Splits can be modified in **cv_split.py**. A 10 fold CV split was used. In this competition we have to split the topics into folds and all topics can have multiple attached content to it. Throughout this is a $n \times m$ relation of $Topic \times Content$, it is difficult to create perfect splits that are aligned with the leaderboard.

During CV-Split creation I try to minimize the overlap of having the same content in different folds, by creating 10 buckets and add topics to the bucket were all attached content of that topic creates the least overlap having the same attached content in different buckets. At least for fold 0 of my 10 folds the alignment to the public leaderboard is quiet well.

For the final results in the competition, models trained on all data without any hold-out validation set were used and are available over the inference notebooks on Kaggle.

4.2 Training

Follow the instructions in the GitHub README.md[4]. All settings are done by the configuration dataclass at the beginning of the script **train.py**.

[2]competition: https://www.kaggle.com/competitions/learning-equality-curriculum-recommendations
[3]translated data: https://www.kaggle.com/datasets/khabel/learning-equality-language-switch
[4]readme: https://github.com/KonradHabel/learning_equality/blob/master/README.md

4.3 Evaluation

Also follow the instructions in the GitHub README.md for offline evaluation. I provide checkpoints trained on fold 0 of 10 over Google Drive. These checkpoints[5] can be used for offline evaluation and threshold/margin testing.

The checkpoints trained on all training data without evaluation hold-out are provided via Kaggle in the inference notebooks for Lederboard Prize[6] and Efficiency Prize[7].

The evaluation scripts **eval_gpu.py** and **eval_cpu.py** also save the extracted features for topic and content and also a traced version of the model. The extracted features can be used to evaluate an ensemble with the script **eval_ensemble.py**.

There are two versions of the eval scripts **eval_gpu.py** and **eval_cpu.py**. The distilled version with only 6 transformer layers can also be evaluated with the GPU version, what ist much faster and advisable. The CPU version is only for the Efficiency Prize cause it does the Quantization and saves a JIT traced version used in the CPU inference notebook on Kaggle. Evaluation and extraction of all known content takes around 40 minutes on an 8 core CPU when using the CPU version for evaluation.

The evaluation script and inference kernel on Kaggle uses a different dataloader as during training. The run-time optimized dataloader sorts after tokeniziation content and topics according to the sequence length in descending order. Be aware that different models and tokenizers leads to different order, so a reorder of the extracted features is necessary if combining more than one model into an ensemble. Reordering is done automatic if using the evaluation scripts or the inference kernels on Kaggle.

4.4 Hardware

Checkpoints used in the competition (using all training data) were trained on a single IBM AC922 (ppc64le) machine with 4 x Nvidia V100 GPUs. The checkpoints provided via Google Drive (using fold 0 for eval) are trained instead on a single RTX 3090 using gradient check-pointing and a lower batch size of only 512 instead of 768.

Results are quite similar, there is no need for high end hardware to train the models. The run-time for training of 40 epochs for example for the **sentence-transformers/LaBSE** takes on a single RTX 3090 (power target set to 300 watt) around 13 h, whereas training on 4 x Nvidia V100 GPUs without gradient check-pointing took only 4 h.

When training on consumer-grade hardware follow the instructions in the README.md and activate gradient_checkpointing. It is not advisible to lower the batch_size instead of using gradient check-pointing, because contrastive training with InfoNCE loss profits from a higher batch size for having enough hard negatives in the batch.

[5] fold 0: https://drive.google.com/drive/folders/102N-wVRLhzf9d5IL3r1HFdYCsCZqcNQF?usp=share_link
[6] leaderboard: https://www.kaggle.com/code/khabel/2nd-place-learning-equality-leaderboard-prize/input
[7] efficiency: https://www.kaggle.com/code/khabel/2nd-place-learning-equality-efficiency-prize/input

5 Results

Because the contrastive training seems to be stable and showing absolutely no signs towards overfitting, all models used for the submissions are trained for 40 epochs without any evaluation hold-out. My experiments on fold 0-2 shows always, that the difference in the last epochs to the best epoch is on the third digit, so training on the whole training data without any validation hold-out should be fine.

5.1 Competition Results

1. For the Leaderboard Prize an ensemble of 5 models with a run-time of only 9 minutes on the P100 Instance is used.

2. For Efficiency Prize an ensemble of only two models is used, cause a second model leads to the biggest jump in scores by only increasing the run-time to 23 minutes on the CPU only Instance.

GPU - Ensemble:

```
Ensemble:
Num. Models:      5

Score:
Privat:           0.75479
Public:           0.70977

Inference:
Kernel:           P100 - GPU
Runtime:          9 min
Max. Seq. Length: 96

Trained:
Data:             all data (no folds)
Epochs:           40
Batch Size:       768 - pairs of [Topic, Content]
Seq. Length:      96
LR-Schedule:      polynomial decay with warmup (2 Epoch)
Max. LR:          0.0003
LR-End:           0.0001
GPU:              4xV100 (32GB)

Models:
- 'sentence-transformers/LaBSE'
- 'facebook/mcontriever-msmarco'
- 'sentence-transformers/stsb-xlm-r-multilingual'
- 'sentence-transformers/paraphrase-multilingual-mpnet-base-v2'
- 'sentence-transformers/xlm-r-100langs-bert-base-nli-mean-tokens'
```

CPU - Ensemble:

```
Ensemble:
Num. Models:      2

Score:
Privat:           0.73118
Public:           0.68959

Inference:
Kernel:           CPU
Runtime:          23 min
Max. Seq. Length: 96

Destillation:
Teacher:          pre-trained Model of GPU Submission
Layers keep:      6
Quantization:     post training dynamic
Data:             all data (no folds)
Epochs:           40
Batch Size:       1024 - pairs of [Topic, Content]
Seq. Length:      96
LR-Schedule:      polynomial decay with warmup (2 Epoch)
Max. LR:          0.0003
LR-End:           0.0001
GPU:              4xV100 (32GB)

Models:
- 'sentence-transformers/LaBSE'
- 'sentence-transformers/paraphrase-multilingual-mpnet-base-v2'
```

Figure 7: Results and Summary

5.2 Results on Fold 0

Ablation for fold 0 demonstrating the difference when using a model with 12 transformer layers vs. the distilled or even quantized version are provided in table 1.

Table 1: Results for sentence-transformers/LaBSE trained on fold 0 with margin=0.16.

Model	Fold 0	Public	Privat	Run-Time (test-set)
12 layer	0.6660	0.6637	0.7026	P100 4 min
6 layer (distilled)	0.6631	0.6523	0.6907	P100 3 min
6 layer (distilled + quantized)	0.6609	0.6526	0.6895	CPU 13 min

Results on fold 0 for **sentence-transformers/LaBSE** and **sentence-transformers/paraphrase-multilingual-mpnet-base-v2** are demonstrated in table 2. The models in this section are all trained with a single RTX 3090 for 40 epochs and results on the test set are verified via late submission.

Table 2: Results for models trained on fold 0 with margin=0.16.

Model	Fold 0	Public	Privat
sentence-transformers/LaBSE	0.6660	0.6637	0.7026
sentence-transformers/paraphrase-multilingual-mpnet-base-v2	0.6615	0.6618	0.7092
ensemble (50%/50%)	0.6849	0.6808	0.7238

The results are below the scores from the Efficiency Prize ensemble despite when using no distillation and quantization, because when training only on fold 0, both models did not had seen 1/10 of the training data and also do not know that content, possibly also part of the test-set. But this demonstrates the effect of combining two different models into an ensemble.

Scores per language when using the ensemble of two models are visualized in figure 8.

```
------------------------------[Ensemble]------------------------------
en  Score: 0.68640 - Precision: 0.65308 - Recall: 0.742 - Selected:   5 - (2806x65939)
es  Score: 0.76713 - Precision: 0.71482 - Recall: 0.835 - Selected:   4 - (1177x30844)
pt  Score: 0.79943 - Precision: 0.74148 - Recall: 0.860 - Selected:   6 - (343x10435)
ar  Score: 0.54267 - Precision: 0.56559 - Recall: 0.662 - Selected:   4 - (318x7418)
fr  Score: 0.61698 - Precision: 0.64399 - Recall: 0.662 - Selected:   7 - (304x10682)
bg  Score: 0.70410 - Precision: 0.67926 - Recall: 0.756 - Selected:   7 - (242x6050)
bn  Score: 0.17561 - Precision: 0.11966 - Recall: 0.230 - Selected:   7 - (237x2513)
sw  Score: 0.71674 - Precision: 0.65091 - Recall: 0.789 - Selected:   5 - (209x1447)
gu  Score: 0.77115 - Precision: 0.68613 - Recall: 0.839 - Selected:   5 - (181x3677)
hi  Score: 0.71008 - Precision: 0.68468 - Recall: 0.774 - Selected:   7 - (138x4042)
it  Score: 0.87877 - Precision: 0.88017 - Recall: 0.904 - Selected:   4 - (73x1300)
zh  Score: 0.66702 - Precision: 0.59670 - Recall: 0.758 - Selected:   9 - (68x3849)
mr  Score: 0.72963 - Precision: 0.69001 - Recall: 0.798 - Selected:  11 - (24x999)
fil Score: 0.80584 - Precision: 0.69457 - Recall: 0.882 - Selected:   7 - (23x516)
as  Score: 0.45620 - Precision: 0.31313 - Recall: 0.662 - Selected:   7 - (13x641)
my  Score: 0.76245 - Precision: 0.82500 - Recall: 0.829 - Selected:   3 - (12x206)
km  Score: 0.95030 - Precision: 0.94697 - Recall: 0.958 - Selected:   4 - (11x505)
kn  Score: 0.63193 - Precision: 0.52910 - Recall: 0.744 - Selected:  10 - (9x501)
te  Score: 0.77938 - Precision: 0.54730 - Recall: 0.946 - Selected:  16 - (7x285)
or  Score: 0.72903 - Precision: 0.71429 - Recall: 0.733 - Selected:   8 - (5x326)
ta  Score: 0.68842 - Precision: 0.45114 - Recall: 1.000 - Selected:   9 - (5x216)
ur  Score: 0.35907 - Precision: 0.22956 - Recall: 0.448 - Selected:   9 - (5x245)
pnb Score: 0.89423 - Precision: 0.82500 - Recall: 0.938 - Selected:   9 - (4x184)
ru  Score: 0.70697 - Precision: 0.67778 - Recall: 0.767 - Selected:  11 - (3x188)
pl  Score: 0.66623 - Precision: 1.00000 - Recall: 0.632 - Selected:  28 - (3x319)
swa Score: 0.16744 - Precision: 0.11597 - Recall: 0.320 - Selected:  40 - (3x495)
tr  Score: 0.71442 - Precision: 0.87698 - Recall: 0.696 - Selected:  23 - (3x225)
--------------------------------------------------------------------------------
Eval Score: 0.68489 - Precision: 0.64888 - Recall: 0.748
--------------------------------------------------------------------------------
```

Figure 8: Results of ensemble per language on fold 0.

References

[1] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[2] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.