Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Alan Akbik

Berlin, 26.04.2024
Lehrstuhl Maschinelles Lernen

Introduction to Natural Langauge Processing

Exercise 2

Submission: Hand your homework until 06.05.2024 (Monday), 23:59 via Github Classroom. Each exercise is worth a total of **10 points** and you need **80%** (of total points over all exercises) to be admitted to the final exam. Sometimes, there are optional tasks that will give you extra points. Exercises in Github Classroom should be completed and submitted individually. However, we encourage you to work on the problems in teams. Please note the information available in Moodle: `https://hu.berlin/nlp24-moodle`.

Github Classroom: `https://classroom.github.com/a/Re_pkagu`

Task 1 (FastText Classifier) 2 + 3 + 3 = 8 points

In this task, you are required to train your own FastText classifier. The FastText classifier uses n-grams to encode sentences into a continous space. Compare this idea idea to what you have learned in exercise 1 (bag-of-words classifier). In the repository, you find three important files for that task: `train.py` (which will perform the training process), `models.py` (which contains the model class), and `data.py` (which contains all data-related methods for loading and vectorizing the data). You need to complete the following tasks:

(a) Complete `WordEmbeddingClassifier` class in `models.py`. The constructor takes some default arguments (`vocab_size`, `num_labels`, `hidden_size`, `pretrained_embeddings`). You need to implement the option to either create an `torch.nn.Embedding` that maps from `vocab_size` to `hidden_size` or loads some pre-trained embeddings (such as GloVe or FastText) into the embedding module. We provide you with two pre-trained embeddings (one is in the `pretrained_embeddings` directory, the other can be downloaded here (>100MB).)

In the forward method, you need to pass the vectorized sentence through the embedding layer, obtain the mean-pooled representation, pass that representation through the linear layer, obtain the log-probabilities with `torch.nn.functional.log_softmax`, and compute the loss using the pre-defined loss function (e.g. `self.loss_function(log_probs, target)`)

(b) Implement `make_ngram_dictionary()` and `make_ngram_vectors()` in `data.py`. An n-gram dictionary contains all the n-grams (on word level) from a sentence. Example: *"A example sentence."* using bi-grams (n = 2) would result in `{0: "A", 1: "example", 2: "sentence", 3: ".", 4: "A example", 5: "example sentence", 6: "sentence ."}`. The `make_ngram_vectors()` should return a one-hot encoded vector for the ngrams present in the sentence. Make sure you can choose different n and pass a `unk_threshold` to filter n-grams that are rarely seen.

(c) Do some runs using `train.py`. You can select different hyperparameters such `learning_rates`, `max_ngrams`, `hidden_sizes`, `num_epochs`, `unk_thresholds` or `pretrained_word_embeddings_files`. You can set these parameters in the top-level execution part of the script (after `if __name__ == "__main__":`). This script will save results of your runs into the `results` directory and plot loss curves and training accuracy for you. Try to find a configuration that gives scores between 80-85 accuracy on the test set. Check out loss curves and training duration in our logs to understand the effect of different hyperparameters. Include a final text-file in tabular format that lists the scores you achieved using which hyperparameters:

Table 1: Example Output Table

Learning Rate	N-Grams	Unk Threshold	Num Epochs	Hidden Size	Pretrained Embedding	Accuracy on Test Set
0.1	1	0	10	100	None	85.2%
0.25	2	2	5	-	GloVe	87.6%

Task 2 (Odd-One-Out) **2 points**

In this task, you will use pre-trained word embeddings to perform the odd-one-out task (file `odd_one_out.py`). In this task, you are given a list of words of which one does not belong in that list. Use the two pre-trained word embeddings to encode the words and try to find out which word does not belong in the list. Hint: The list always contains three words of which one does not belong to the list. Compare each word embedding with each other using different distance functions. At last, print out the results which shows the original list and which of the words does not belong in there.