Statistical NLP — Homework Exercise 5

Ran Zhang, Daniil Larionov, Philipp Cimiano
Semantic Computing Group
Bielefeld University
Winter Semester 2023/2024

Note on Submission You are allowed to submit in groups of two people. When you want to submit in a group put the name of your colleague in the comment field. Only one person needs to submit it.

All solutions have to be uploaded together as a single zip file to LernraumPlus. Solve the exercises by completing the functions in file `exercise_sheet5.zip` Provide some information about how to execute your Python code. Non-code answers should be included in a PDF file. **For this exercise, you are allowed to use the entire Python standard library as well as `gensim`, `scipy`, and other necessary packages.**

Task 1 — Word Similarity [10 points]

In this task, you will evaluate how well pretrained word2vec embeddings perform in word similarity task on the WS-353 dataset. This is a classic dataset consisting of 353 word pairs and human annotations of their similarity. Complete the code in `task1-w2v-sim.py`.

All parts of the code that should be edited are marked with `# TODO: Exercise Name`. If you edit any other parts of the code, please mark your changes with `# MODIFIED: Reason`.

a) (1p) Download the pretrained 300-dimensional word2vec embeddings from Google [1] and load them with gensim. Hints:

- `https://radimrehurek.com/gensim/downloader.html` might be useful for downloading the word embeddings programmatically.
- Read `https://radimrehurek.com/gensim/models/keyedvectors.html` might be helpful.
- If you run into memory errors because the embeddings are too large, you may use the limit parameter of the load_word2vec_format method.

b) (2p + 2p) To explore the structure of the embedding space, it is necessary to introduce a notion of distance.[2] There exist multiple such "distance" measures, such as Euclidean distance and cosine similarity. Use gensim to compute the cosine similarities of the word pairs from WS-353 (2p). Then, compute the Euclidean distance one more time (2p). (Hint: you could use the embedding vectors from gensim and distance function from scipy).

c) (2p + 1p) Spearman's rank correlation coefficient is a typical choice for measuring the ranking of two variables. Compute the coefficient between the values assigned by humans and your computations, namely cosine similarity results and Euclidean distance results from (b) using scipy. Return both results (2p) and interpret the resulting coefficients in one or two sentences (1p). (Hint: you can compare the differences of the output and make some comments with respect to the difference in distance measure).

d) (2p) In this task we will investigate the inherent gender bias of the Google word2vec embeddings. You will receive a list of words $[word1, word2, word3]$. Use functions from gensim to identify 5 words that are most similar to the word-pair $[word1, word2]$ and least similar to word3.

[1] `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?resourcekey=0-wjGZdNAUop6WykTtMip30g`
[2] Here are some hints for the task with GloVe embedding. You need to implement similar tasks with word2Vec embedding using gensim. `https://www.cs.toronto.edu/~lczhang/360/lec/w05/w2v.html`

Task 2 — Text Classification with Keras [5 points]

This task is designed to get you familiar with text classification pipeline in Keras & Tensorflow. You will chose one of the provided text classification datasets and then create deep learning model. This task focuses on models based on word embeddings.

All parts of the code that should be edited are marked with `# TODO: Exercise Name`. If you edit any other parts of the code, please mark your changes with `# MODIFIED: Reason`.

Complete code in `task2-keras-text-classification.py`. Before that, you may install required dependecies by running bash script in `install_requirements.sh`.

a) (not graded) Choose one of the following datasets. Indicate your choice by uncommenting respectful line in code.
   a) Emotion[3] - Dataset for emotion classification
   b) DBpedia14[4] - Dataset for ontology classification
   c) The Multilingual Amazon Reviews Corpus[5] - dataset for classifying product reviews (English language subset)

b) (1p) Expore the dataset and choose appropriate text preprocessing steps. Implement them by completing code in function *preprocess_text(..)*. Note that this requires you to use Tensorflow's methods for text preprocessing. See code comments for details.

c) (1p) Choose word embeddings and implement loading them by completing code in *load_word_embeddings(..)*. Your embeddings can be either loaded with Gensim's download api, or downloaded separately. In the latter case please also complete script in file `install_requirements.sh` by including code to download and unpack embeddings file.

d) (1p) Design and implement model architecture by completing code in function *create_model(..)*. See code comments for details.

e) (1p) Implement model compilation and training.

f) (1p) Experiment and iterate over various hyperparameters(also optimizers, word embeddings, etc.) in order to achieve the best test result possible. Report configuration used to achieve at least 50% categorical accuracy. In case if you find it impossible to achieve 50% threshold - describe in free form what have you tried.

Task 3 — Text Classification with BERT [5 points]

In this task you will build a text classification model through fine-tunning pre-trained transformer model BERT. As in previous task, you will choose one of the provided text classification datasets. You will also get familiar with *transformers* library.

All parts of the code that should be edited are marked with `# TODO: Exercise Name`. If you edit any other parts of the code, please mark your changes with `# MODIFIED: Reason`.

Complete code in `task3-bert-text-classification.py`. Before that, you may install required dependencies by running bash script in `install_requirements.sh`.

a) (not graded) Choose one of the following datasets. Indicate your choice by uncommenting respectful line in code. Datasets are the same as in the previous task.

b) (1p) Choose pre-trained transformer model from Huggingface Hub[6], and implement code to load pre-trained tokenizer for that model.

[3] `https://huggingface.co/datasets/emotion`
[4] `https://huggingface.co/datasets/dbpedia_14`
[5] `https://huggingface.co/datasets/amazon_reviews_multi`
[6] `https://huggingface.co/models?pipeline_tag=fill-mask&sort=downloads`

c) (1p) Implement batch-tokenization of input data using loaded pre-trained tokenizer. Complete code inside *tokenize_fn(..)* function. Hint: consult with documentation[7] for pre-trained tokenizers.

d) (1p) Choose hyperparameters for the model training.

e) (1p) Compile and train the model. Use prepared training and validation dataset, as well as the optimizer. Use categorical accuracy as metric.

f) (1p) Experiment with various hyperparameter and pre-trained models in order to achieve the best possible categorical accuracy on the test set. Report hyperparameters and pre-trained model used to achieve at least 55% accuracy. If you find it impossible to achieve 55% accuracy - describe in free form what have you tried.

Note: you may want to remove subsampling of the training data, but this can drastically increase training time. If you have access to GPU resources - you can also comment out lines 2 and 3 in provided file, which will allow you to use GPU for accelerated training. You can use free GPU compute at Colaboratory[8]. In this case you will need change runtime type to ”GPU”, upload your code files and execute them from colaboratory code cells.

[7] https://huggingface.co/docs/transformers/main_classes/tokenizer#transformers.PreTrainedTokenizer
[8] https://colab.research.google.com