Exercise 2

WS 2021 - 188.977 Grundlagen des Information Retrieval

Organization

- Deadline: 13.01.2022, 23:55 upload in TUWEL
- Same groups as in exercise 1
- Exercise will be evaluated based on your submitted report and code
- In case of questions, use the TUWEL forum so that others also benefit from the question/answer. For specific questions, contact me markus.zlabinger@tuwien.ac.at directly.

Environment

In this exercise, you are allowed to use the following external libraries.

- gensim: Used to infer word embeddings
- scikit-learn: To compute cosine similarity between texts
- scipy: To compute the Pearson correlation

Part1: Warmup (20 points)

1. Download the following language model for inferring word embeddings:
https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M-subword.vec.zip

2. Load the language model using the gensim library.

3. Infer word embeddings for the following word pairs and compute the cosine similarity between the word vectors. Add the cosine similarities to your report.

```
pair1: ("cat", "dog")

pair2: ("cat", "Vienna")

pair3: ("Vienna", "Austria")

pair4: ("Austria", "dog")
```

4. Compute the top-3 most similar words for the following words and describe them in the report.

```
word1: "Vienna"

word2: "Austria"

word3: "cat"
```

Part2: Short-Text Similarity (45 points)

The goal of the second part of this exercise is to create text embeddings to compare the similarity between short-texts (such as questions, sentences, phrases, etc.). Use the file “dataset.tsv” for this task. The dataset is a tab-separated list containing lines of

```
ground_truth, text1, text2
```

The ground_truth label ranges from 0 (dissimilar) to 5 (most similar).

- **Preprocessing:** Implement text tokenization, including lowercasing and stopword removal. Do not apply stemming or lemmatization. Feel free to use any library for this or your preprocessing function from exercise 1.
- **Short text embedding:** Implement three methods to infer a vector representation from texts.
    - Vector space model using TF-IDF weighting (use TfidfVectorizer() from the sklearn library with default parameters).
    - Compute a short text vector representation from word embeddings in two steps: (1) For each word appearing in a text, compute a word embedding. (2) The word embeddings are aggregated via mean averaging to infer a vector representation for the text. As language model to infer word embeddings, use the same as in the Warmup part of this exercise.
    - Compute a short text vector representation from word embeddings in two steps: (1) For each word appearing in a text, compute a word embedding. (2) The word embeddings are aggregated using a weighted averaging based on each word’s IDF (Inverse Document Frequency) value. As language model to infer word embeddings, use the same as in the Warmup part of this exercise. You can extract the IDF values from the sklearn TfidfVectorizer() method)
- **Evaluation:** For each implemented method, infer the text vector representation for the text pairs in “dataset.tsv” and then compute the similarity between the text pairs using the cosine similarity. Measure the Pearson correlation between the ground_truth labels and the similarities computed by each method. Use the function pearsonr from the scipy library.

  ```
  from scipy.stats.stats import pearsonr
  ...
  pearsonr(gt_scores, predicted_scores)
  ```

  Compute results with two preprocessing functions:
    - lowercasing, tokenization, and stopword removal
    - lowercasing and tokenization.

  Add the evaluation results to your report.

Part3: Training new language models (35 points)

Pre-trained language models are not always effective. For example, try to infer the German word “apfel” using the language model from the warmup part of this exercise. It will not work because the language model was trained on English data and the word “apfel” did not appear during the training procedure. Your aim in this task is to train a language model from scratch for German texts.

1. Corpus Acquisition: Acquire German text data for the training procedure. You have the freedom to select any data source you like: For example, pre-packed datasets of Twitter, Wikipedia or newspaper articles. Note that you need to find the right balance for data (too much data: slow

training; not sufficient data: ineffective language model). I recommend to use somewhere between 300 MB to 1.5 GB of text data for training.

2. Training: Use the gensim library to train a Word2Vec language model. You can use preprocessing from gensim (but disable stemming+lemmatization).

```
import gensim

from gensim.parsing.preprocessing import *

FILTERS = [strip_tags,
strip_punctuation,strip_multiple_whitespaces,strip_numeric,strip_short]

texts = [“the first text”, “the second text”,...]

texts_tokenized = [preprocess_string(text.lower(),FILTERS) for text in texts]

german_model = gensim.models.Word2Vec(sentences=texts_tokenized, vector_size=100, window=5,
min_count=1, epochs=10, workers=TODO)
```

3. **Inference:** Compute for 3 German words of your choice the top-3 most similar words and describe them in the report.

Notes

- The cosine similarity can be computed using sklearn

  ```
  from sklearn.metrics.pairwise import cosine_similarity
  ```

- Gensim can be used to open the language model and infer word embeddings

  ```
  model =
  gensim.models.KeyedVectors.load_word2vec_format(“path.to.model.vec”)

  model.get_vector("word", norm=True) # normalizing usually improves
  performance
  ```

- Some words in the texts of “dataset.tsv” will not be available in the language model. Filter these words before aggregation.
- If you don’t manage to find appropriate German text datasets, you might use one of the following sources:
  - German Tweets: https://zenodo.org/record/3633935#.YbpSgGiZOHs
  - Austrian Newspapers: https://github.com/UB-Mannheim/AustrianNewspapers