Assignment 3

Query Expansion with Word Embeddings

Date: 22/03/2024

Laura Givskov Rahbek

Description

This folder contains assignment 3 for Language Analytics. The objective of the assignment is to properly pre-process text data, use pre-trained word embeddings for query expansion, and to implement command line tools to generate results based on a given input. More specifically, the `gensim` pretrained wordembeddings model `glove-wiki-gigaword-50` and the method `model.most_similar()` will be used to identify and extract the ten most similar words to a given keyword, these will then be used to perform the expanded query on the *Spotify Million Song Dataset*, described further in the Data section below.

The `keywordcounter.py` script does the following:

- Takes two arguments; the name of an artist with the flag -a (*artist*) and a keyword with the flag -k (*keyword*).
- Loads the *Spotify Million Song Dataset* and checks if any songs exists by the artist given, if the artist is not in the dataset, 'Artist not found' is returned in the terminal.
- Loads the `glove-wiki-gigaword-50` word-embedding model and returns a list of the ten most similar words, to the given keyword, and the keyword itself.
- Cleans the texts by the given artists, by tokenising, making each token lower case and stripping punctuation.
- Counts the number of texts any of the keywords are present in, by the given artist, and appends the results to the `output.csv` file. Each row in the file contains the keyword, the ten similar words, the name of the artist, the total number of songs by the artist, the total number of songs by the artist containing any of the words and finally the percentage of songs by the artist containing any of the words.
- In the terminal the percentage of the artist's songs containing the words is returned.

Data

The data used in this assignment is the *Spotify Million Song Dataset*, a corpus containing 57,650 English-language songs with their titles, the artist, a link and the lyrics. The corpus can be found here.

Usage and Reproducing of Analysis

To perform the expanded query using `keywordcounter.py` do the following:

- Place the downloaded file `Spotify Million Song Datset_exported.csv` in the `in` folder.
- Run `bash setup.sh` from the command line, it creates a virtual environment and installs packages and dependencies in to it.
- Run `bash run.sh` in the terminal with the two arguments -a and -k, this opens the virtual environment and runs `keywordcounter.py` on the given input and returns an appende row to the `output.csv` file, then exits the environment again. (Note: if the artist' name conatins a space, put it in quoatation marks, e.g. "Taylor Swift").

For example to find out how many songs Taylor Swift has about words related to love, run the following in the terminal:

```
bash run.sh -a "Taylor Swift" -k love
```

Discussion

The results of the expanded keyword query are reflective of the method used; word embeddings and the data those are trained on, in this case a very large amount of wikipedia articles. This is reflected in one of the examples seen in the `output.csv` file; performing the expanded keyword query on Justin Bieber songs with the keyword 'baby', returns the following 10 words: 'babies', 'boy', 'girl', 'newborn', 'pregnant', 'mom', 'child', 'toddler', 'mother', 'cat'. Considering the meaning of 'baby' in the song, a term of endearment, the returned words do not reflect that same sentiment. It makes sense that a word embedding model trained on text meant to factually inform, captures other meanings than those used in songs or other creative language.

Further, the individual word embeddings are kept as different conjugations and versions of the same word. An example of this is in the `output.csv` file, where both 'loves' and 'loving' are returned as some of the closest words to 'love'. When performing the exapnded query, there are limits to how much additional information is given when these words are included in addition to 'love'. There are of course work arounds to this, for example extracting the root of the returned words and only keeping them if they aren't identical to the keyword. This could potentially be implemented to increase the scope of the query.

`codecarbon` was used to track the environmental impact when running this code, the results and an exploration of this can be found in the `Assignment-5` folder in the repository.