Devin Wu <- replace with your name

CS 585 Fall 2024 Programming Assignment #01

Due: **Sunday, September 22, 2024 at 11:59 PM CST**
Points: **100**

Instructions:

1. Place **all your deliverables (as described below) into a single ZIP** file named:

`LastName_FirstName_CS585_Programming01.zip`

2. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

Objectives:

1. (100 points) Implement the Byte Pair Encoding tokenization algorithm.

Task:

Your task is to implement the Byte Pair Encoding (BPE) tokenization algorithm (both Learner and Segmenter parts).

Deliverables:

Your submission should include:

- Python code file(s). Your main .py file should be named:

`CS585_P01_AXXXXXXXX.py`

where `AXXXXXXXX` is your IIT A number (this is REQUIRED!). If your solution uses multiple Python files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well (for example: `CS585_P01_AXXXXXXXX_EXTRAS.py`).

- This document with your observations and conclusions. You should save it as:

`LastName_FirstName_CS585_P01.pdf`

Implementation:

Your task is to implement (**from scratch – you can't use existing Python package tokenizers | you can ONLY use re and pandas [or similar-subject to approval] packages**) the BPE tokenizer (as explained in class) and apply it to tokenize text.

Your program should:

- Accept three (3) command line arguments, i.e. so your code could be executed with

```
python CS585_P02_AXXXXXXXX.py K TRAIN_FILE TEST_FILE
```

where:

- `CS585_P01_AXXXXXXXX.py` is your python code file name,
- `K` is the number of byte pair merges (a positive integer),
- `TRAIN_FILE` is an input file name corresponding to a training text file that your BPE learner will use to train / build its vocabulary, while `TEST_FILE` is the test text file with some sentenced to be tokenized by your TRAINED BPE (its Segmenter component).

  Assumptions:

  - both input files are ALWAYS in the same folder/directory as your main python file (i.e. no need to provide/analyze path, just the file name),
  - both input files MAY contain characters outside of the INITIAL vocabulary V (as described below).

Example:

```
python CS585_P01_A11111111.py 20 TRAIN.TXT TEST.TXT
```

Parameter checks:

- If the number of arguments provided is NOT three (none, four or more) display an error message and exit.
- If one of the files does NOT exist in the same folder / directory, display error message and exit,
- If the `K` argument is out of the specified range, assume that the value for `K` is `5`.

- Load and process training data file:
  - treat every input file as a single sentence, even if it is made of many (no sentence segmentation needed),
  - remove all punctuation if any,
  - you may need to clean up the input file first to remove all non-printable characters other than space (to be used for space-based tokenization step) and other characters that are NOT in the INITIAL vocabulary V.

- Train your BPE learner on your data set (`TRAIN_FILE`):
  - assume that INITIAL vocabulary V is the set of
    - all letters (uppercase, lowercase) in English alphabet,
  - add the stop token character to your vocabulary,
  - train your BPE Learner by performing `K` merges,
  - this will be your **FINAL** vocabulary V to be used by the BPE Segmenter to tokenize test file text.

- Load and process test data file (TEST_FILE):
  - treat every input file as a single sentence, even if it is made of many (no segmentation needed),
  - remove all punctuation if any,
  - you may need to clean up the input file first to remove all non-printable characters other than space (to be used for space-based tokenization step) and other characters that are NOT in the INITIAL vocabulary V.

- Tokenize test data using your trained BPE Segmenter.

- Save results to disk (in the same folder / dictionary as your main .py file):
  - **FINAL** vocabulary V should be saved (one token per line in order of merges; no additional characters) to a text file:

    CS585_P01_AXXXXXXXX_VOCAB.txt

  - Complete result of TEST_FILE content tokenization should be saved to a text file:

    CS585_P01_AXXXXXXXX_RESULT.txt

    where: AXXXXXXXX is your IIT A number (this is REQUIRED!).

Your program on-screen output should look like this

```
Last Name, First Name, AXXXXXXXX solution:
Number of merges: K
Training file name: TRAIN_FILE
Test file name: TEST_FILE

Training time: yyy seconds
Tokenization time: zzz seconds

Tokenization result: <tokenization result here>
```

Where:
- K, TRAIN_FILE, and TEST_FILE are command line inputs listed above,
- yyy the actual time (in seconds) it took to generate FINAL vocabulary V (excluding pre-processing),
- zzz the actual time (in seconds) it took to tokenize TEST_FILE data/text (excluding pre-processing) FINAL vocabulary V,
- <tokenization result here> is actual BPE tokenized test file TEST_FILE data/text with space used as a separator.
  - If tokenization result is compromised of more than 20 tokens, stop displaying after the 20th token and add the "Tokenized text is longer than 20 tokens" underneath

Analysis, summary, and conclusions:

Run the algorithm

- for `K` = 50, 100, 150, 200
- and for every value of `K` above, change `TRAIN_FILE` size: 500, 1000, 1500, 2000 tokens (words). If sample files not provided, create your own.,
- OPTIONAL: feel free to run the tests on higher values of `K` and `TRAIN_FILE` size.

When measuring run time for this analysis, you may need to run the same process a number of times and calculate the average. Feel free to create a separate, modified version of your code for that purpose that you DON'T need to submit.

What are your observations and conclusions? How did the number of merges affect the tokenization results (morphemes, prefixes, suffixes, common words discovered or not)? Add comparison plots below.

Summary / observations / conclusions
For the training text, I chose random passages from the wiki page of the Garchomp Pokemon. <u>50 merges with ~500 words training file:</u> The training time results were: 0.043, 0.04618, 0.04536, 0.03931, 0.04301 for average of 0.043372s The tokenization results include the topic of the passages in question, Garchomp, which is good. It also includes a bunch of common words, prefixes, and suffixes like: it, and, the, ly, y, ing, ed, of, and to. <u>100 merges with ~1000 words training file:</u> The training time results were: 0.16314, 0.17389, 0.16676, 0.16114, 0.16299 for average of 0.165584s The tokenization results are now including larger common words like: with, was, from, and own. It is also contains parts of larger words like battl, Pokem. The time it took to train has increased 4 times. I think this is expected because we have double the amount of merges and double the amount of words. <u>150 merges with ~1500 words training file:</u> The training time results were: 0.31752, 0.323816, 0.33693, 0.32571, 0.35208 for average of 0.3312112s The tokenization results now include full large words like: battle, appear, Mega,

Cynthia. Parts of common words are also included like: atch, oth, ons, ght, ent. The time also increased roughly double which I think is expected because 1.5*1.5 = 2.25 which is roughly double.

200 merges with ~2000 words training file:

The training time results were: 0.54033, 0.53788, 0.54029, 0.56186, 0.53004 for average of 0.54208s

Even more full sized words are in the tokenization results like: again, against, Meteor, other, Dragon, Team. Even more parts of common words are also there like: ding, fter, ock. The time seems to be increasing as expected too. 2000 is 1.33x larger than 1500 and so as 200 to 150. 1.33*1.33 = 1.77. 0.54 is close enough to 0.33*1.77. The times do seem a bit faster than this way of calculating though. I expected 200 merges with 2000 words to take 4 times longer than 100 merges with 1000 words, but it is faster than I expected.

	500	1000	1500	2000
50	0.043372	0.10077	0.13113	0.16847
100	0.08007	0.165584	0.24246	0.31099
150	0.10491	0.22914	0.3312112	0.44205
200	0.13473	0.28496	0.40373	0.54208

GENERAL OBSERVATIONS:

As the number of merges go up while train sizes stay the same, the length of each token increases to capture long words.
As the train sizes go up while the number of merges stay the same, the tokenization results stay roughly the same. Very common words are captured but everything else stays short, roughly 2-3 in length. I would assume this is because at a very high level, most of English text have many similar word patterns. For example, “the” is the most common word in English so we expect the tokenization results to stay roughly the same and possibly even more stabilized as we add a larger training size.
The time to train increases linearly across the merge axis or the training size axis. Combining these two by going diagonally shows an exponential increase.