COL764 Assignment-1

Submitted by Raj Chanda (2021ME10983)

Task 1

In Task 1, it was asked to implement Simple tokenizer, Byte-Pair Encoding and WordPiece Tokenizer from scratch.

Simple Tokenizer

- Simple Tokenizer was simply constructed using regular expressions functionality in python.
- Since the delimiters were already given, I used them to split a text, and return the tokens for the text.
- I removed the non-ascii tokens from the dataset.

Byte-Pair Encoding

- For the implementation of this, I have used the “defaultdict” and “set” data structure of python.
- After removing all non-ascii characters from the texts, I have encoded every token in the initial corpus into a byte (encode(‘utf-8’)). This was done for better representation.
- Further, for attaching and end of word token, I have used “|” as the end of word token, since this was only occurring in the training doc file for 14 times, I assumed it would be safe to use it instead of any other special token.
- For merging tokens, every time two tokens merged, I store them in a merge file for easier encoding in later tasks. A new token number is assigned to the token, which is saved, and added to the vocabulary.

WordPiece Tokenizer

- Similar to BPE, non-ascii values were removed, but no utf-8 encoding was done in this.
- For computing pair probabilities for likeliness value of WordPiece, I used a marker, as to all characters of a word should be merged up to the same merges, and then only new merges will be done. In this way, the searching for the best pair became efficient than the naïve method.

Statistics for Task 1:

1. Simple tokenizer is taking 20 seconds for training generating 463k tokens
2. BPE is taking almost 5 mins for generating a vocab size of 1000 tokens
3. Word Piece is taking almost 5 mins for generating 1000 tokens in vocab

Task 2

For construction of index and postings list, I instantiated the tokenizer classes developed in task 1 for training the tokenizers.

I go through every file in the training corpus, and tokenize it using the tokenizer, and add them into postings list.

Token ids are also allotted to every token, and every document is mapped to an integer document id. These mappings are also stored for usage in task 3.

In case of BPE, the merges are stored in the same directory, since they will be used in encoding text in task 3.

Statistics:

Simple Tokenizer: 50 seconds

BPE: 1100 seconds for vocab size of 1600

WordPiece: 1100 seconds for vocab size of 1400

Task 3

For tf idf search

- First the vocab and index files are read into memory
- The document mapping is read into memory
- The tokenizer data are also read – vocab and merges for BPE, and vocab for WordPiece
- The queries are retrieved and stored in memory after tokenizing them. During this tokenization, the weight vectors are also computed and stored.
- The document weights and their magnitudes are calculated. This is done beforehand since this will be used every time similarity is calculated.
- Finally, the rankings are retrieved for every query in the query file.

Statistics:

For BPE

```
metrics for query 1 -> precision : 0.12, recall : 0.017167381974248927, F100 : 0.030037546933667076
metrics for query 2 -> precision : 0.07, recall : 0.020895522388059702, F100 : 0.03218390804597702
metrics for query 3 -> precision : 0.26, recall : 0.03987730061349693, F100 : 0.06914893617021277
metrics for query 4 -> precision : 0.11, recall : 0.019400352733686066, F100 : 0.03298350824587706
metrics for query 5 -> precision : 0.07, recall : 0.010835913312693499, F100 : 0.018766756032171584
metrics for query 6 -> precision : 0.49, recall : 0.04929577464788732, F100 : 0.08957952468007312
metrics for query 7 -> precision : 0.58, recall : 0.11068702290076336, F100 : 0.18589743589743588
metrics for query 8 -> precision : 0.14, recall : 0.021604938271604937, F100 : 0.03743315508021391
metrics for query 9 -> precision : 0.2, recall : 0.09569377990430622, F100 : 0.12944983818770228
metrics for query 10 -> precision : 0.58, recall : 0.11670020120724346, F100 : 0.19430485762144056
metrics for query 11 -> precision : 0.07, recall : 0.01583710407239819, F100 : 0.025830258302583026
metrics for query 12 -> precision : 0.32, recall : 0.04938271604938271, F100 : 0.0855614973262032
metrics for query 13 -> precision : 0.24, recall : 0.02608695652173913, F100 : 0.04705882352941176
metrics for query 14 -> precision : 0.29, recall : 0.10622710622710622, F100 : 0.1554959785522788
metrics for query 15 -> precision : 0.03, recall : 0.006726457399103139, F100 : 0.01098901098901099
metrics for query 16 -> precision : 0.25, recall : 0.06097560975609756, F100 : 0.09803921568627451
metrics for query 17 -> precision : 0.68, recall : 0.09483960948396095, F100 : 0.16646266829865358
metrics for query 18 -> precision : 0.47, recall : 0.07057057057057058, F100 : 0.12271540469973892
metrics for query 19 -> precision : 0.2, recall : 0.17094017094017094, F100 : 0.18433179723502305
metrics for query 20 -> precision : 0.6, recall : 0.07926023778071334, F100 : 0.14002333722287047
metrics for query 21 -> precision : 0.49, recall : 0.0745814307458143, F100 : 0.1294583883751651
metrics for query 22 -> precision : 0.39, recall : 0.06554621848739496, F100 : 0.11223021582733814
metrics for query 23 -> precision : 0.63, recall : 0.15949367088607594, F100 : 0.2545454545454545
metrics for query 24 -> precision : 0.47, recall : 0.10444444444444445, F100 : 0.17090909090909093
metrics for query 25 -> precision : 0.26, recall : 0.04521739130434783, F100 : 0.07703703703703704
```

For WordPiece

```
metrics for query 1 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 2 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 3 -> precision : 0.02, recall : 0.003067484662576687, F100 : 0.005319148936170213
metrics for query 4 -> precision : 0.01, recall : 0.001763668430335097, F100 : 0.002995007496251873
metrics for query 5 -> precision : 0.01, recall : 0.0015479876160990713, F100 : 0.002680965147453083
metrics for query 6 -> precision : 0.01, recall : 0.001006036217303823, F100 : 0.0018281535648994518
metrics for query 7 -> precision : 0.01, recall : 0.0019083969465648854, F100 : 0.003205128205128205
metrics for query 8 -> precision : 0.01, recall : 0.0015432098765432098, F100 : 0.00267379679144385
metrics for query 9 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 10 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 11 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 12 -> precision : 0.03, recall : 0.004629629629629629, F100 : 0.008021390374331552
metrics for query 13 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 14 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 15 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 16 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 17 -> precision : 0.29, recall : 0.040446304044630406, F100 : 0.07099143206854346
metrics for query 18 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 19 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 20 -> precision : 0.06, recall : 0.007926023778071334, F100 : 0.014002333722287048
metrics for query 21 -> precision : 0.01, recall : 0.0015220700152207, F100 : 0.002642007926023778
metrics for query 22 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 23 -> precision : 0.01, recall : 0.002531645569620253, F100 : 0.00404040404040404
metrics for query 24 -> precision : 0.0, recall : 0.0, F100 : 0
metrics for query 25 -> precision : 0.01, recall : 0.0017391304347826088, F100 : 0.002962962962962632
```