CSE 556: Natural Language Processing
Assignment 1

Date: Jan 29, 2024
Due Date: Feb 5, 11:59:59 pm Max Marks: 100

General Instructions:

- Every assignment has to be attempted by 4 people. At least one part has to be done by one team member. All members need to have a working understanding of the entire code
- Institute policies will apply in cases of plagiarism
- Create separate .py files for each part attempted by each student. The file should follow the format: A1_<task/subtask_number>_<student roll no>.py
- Create a single .py/ipynb file for generating the final outputs that are required for submittables. Clearly indicate which cell/python method corresponds to which task/subtask. Outputs will be checked from this inference file only.
- Only one person has to submit the zip file containing all the above-mentioned files, along with the report as a PDF. It will be named A1_<Grp No>.zip The person having the alphabetically smallest name should submit.

Tasks: The assignment consists of 2 tasks:

1. Implement tokenization using BytePair encoding.
2. Create a Bigram Language Model (LM), and modify the standard implementation of Bigram LM to generate emotion-oriented sentences

Dataset: It can be downloaded from the link. The folder consists of the files corpus.txt and labels.txt. This is a subset of the twitter emotion dataset available on Hugging Face (link). The file corpus.txt consists of text samples, 1 sample per line. The file labels.txt consists of the corresponding emotion labels. Please use only this corpus for tasks 1 and 2.

Task 1 (30 marks)

Implement **FROM SCRATCH** a tokenizer based on the BytePair encoding algorithm (link). You are only allowed to use standard Python libraries and objects (lists, arrays, dictionaries, and collections library). Use of existing frameworks (such as nltk, HuggingFace, Spacy, TextBlob) is not allowed.

Specifically, you are required to create a Tokenizer class in python, which implements the following methods:

- **learn_vocablury()**, which takes as parameter the corpus number of merges and learns the split rules and frequencies; and
- **tokenize()**, which takes as input a sample and tokenizes it based on the learnt rules.

(12 marks)

Evaluation:
Submit .txt files for each of the following:

1. All possible tokens in the vocabulary (1 line for each token, for a specified number of merges as specified by the TA **(6 marks)**
2. All the merge rules learnt after learning the vocabulary for number of merges specified by the TA. A merge rule can be written as a pair of comma separated (a, b), one rule per line **(6 marks)**
3. Split tokens after tokenizing a set of test samples provided by the TA. Tokens obtained are to be written to a file, separated by commas, one sample per line **(6 marks)**

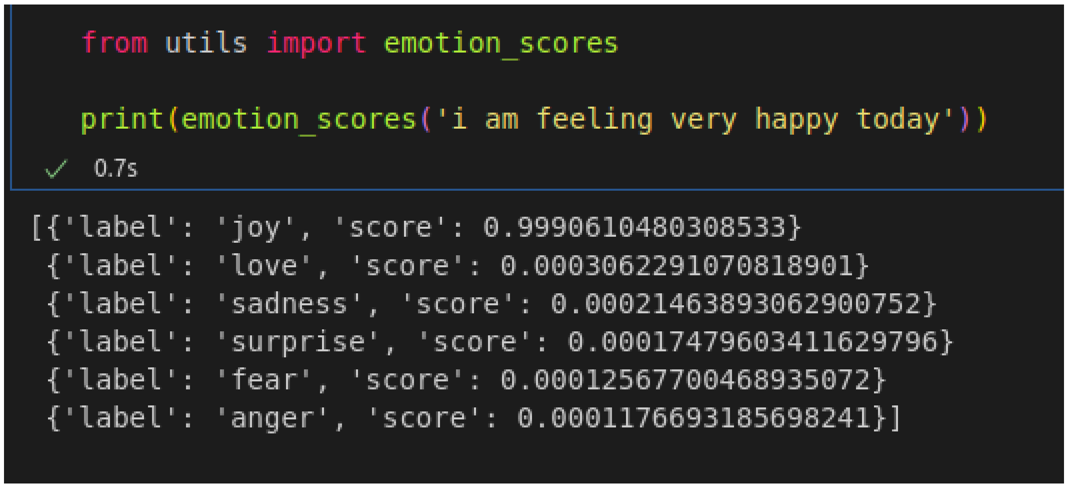
Refer to the following folder (link) for a demo about how to submit the .txt files (refer to files tokens.txt, merge_rules.txt and tokenized_samples.txt)

Task 2 (70 marks)

1. Implement a Bigram Language Model from scratch. You are only allowed to use standard Python methods and the NumPy library for implementation.
   Specifically, create a class called BigramLM, which has a methods for learning the bigram model from the dataset, and stores the learned LM and other supporting methods and objects **(10 marks)**
2. Implement the following two smoothing algorithms in the BigramLM class: Laplace and Kneser-Ney smoothing. Compare the probabilities obtained from both and give an argument for which is better **(5 + 5 marks)**
3. Download the following file: utils.py. Use the function emotion_scores() to get the emotion scores of a sample sentence. Using these emotion scores, modify the standard probability of the bigram model

$$P(w_i|w_{i-1}) = (count(w_i)/count(w_{i-1})) + \beta$$

   where β is the emotion component. Note that this β can be included in any part of the calculation, and at the unigram, bigram or sample level. Using this modification, you need to generate emotion-oriented samples (generate samples corresponding to a specific emotion) **(20 marks)**



```
from utils import emotion_scores

print(emotion_scores('i am feeling very happy today'))
```

✓ 0.7s

```
[{'label': 'joy', 'score': 0.9990610480308533}
 {'label': 'love', 'score': 0.0003062291070818901}
 {'label': 'sadness', 'score': 0.0002146389306290075２}
 {'label': 'surprise', 'score': 0.00017479603411629796}
 {'label': 'fear', 'score': 0.00012567700468935072}
 {'label': 'anger', 'score': 0.00011766931856982４1}]
```

4. **Extrinsic evaluation**:
    a. Generate 50 samples for each of the 6 emotions for which you can get scores. Store these outputs in .txt files for each emotion using the file name format gen_<emotion>.txt. These generated samples will be used for extrinsic evaluation of your LM. **(3 marks)**
    b. Carry out extrinsic evaluation with the original corpus as your training data, and the generated samples as your testing data (labels for the generated samples will be the emotion corresponding to which you generated each sample).
    Train a SVC model from Scikit-Learn library (link), and use the TF-IDF vectorizer (link) for vectorizing the text samples. Conduct Grid Search (link) to find out the best parameters and use the best model obtained to test out the performance of your emotion based modification **(12 marks)**

    c. Submit the generated samples at the time of evaluation to the TA, who will verify the accuracy using our own models. **(5 marks)**

Evaluation:

Include the following components in the report: **(2x5=10 marks)**

1. Top 5 bigrams before smoothing and after each of the 2 selected smoothing techniques along with their probabilities **BEFORE** applying emotion component.
2. Reasoning for method used for including emotion component
3. 2 generated samples for each emotion, for a total of 12 generated samples
4. Accuracy and macro F1 scores obtained from extrinsic evaluation
5. For each emotion, pick 1 of the generated sample and reason why it is generated according to its corresponding emotion.
6. A credit statement reflecting the contribution of each member of the group for the assignment.