CSCI 373
Spring 2023
Assignment 2

Question 1 [100pt]
Language identification is the problem of determining the language a text is written in. N-grams can solve this problem effectively.

We will focus on identifying whether a text is written in English or Spanish. You have to create a language model for English and another language model for Spanish, and use these models to predict the language of new text.

Download file `language_detector.tgz` from the course website. It contains some code in Python and training and test documents downloaded from `http://www.gutenberg.org/wiki/Adventure_%28Bookshelf%29` (English) and `http://www.gutenberg.org/browse/languages/es` (Spanish).

`language_detector/language_detector.py` will create models for English and Spanish, and use them to predict the language of the test documents. To run it, use the following command:

```
$ python language_detector/language_detector.py \
    data/train/en/all_en.txt \
    data/train/es/all_es.txt \
    data/test/
Prediction for English documents in test:
news2.txt None
pg16.txt None
pg345.txt None
news3.txt None
pg1497.txt None
pg3526.txt None
news1.txt None
pg103.txt None

Prediction for Spanish documents in test:
pg14311.txt None
pg25956.txt None
news2.txt None
pg21906.txt None
news3.txt None
pg15725.txt None
news1.txt None
pg31465.txt None
```

The code does not do anything useful until you implement methods `create_model(path)` and `predict(file,model_en,model_es)`. Your job is to implement a **character** bigram model.

A few notes:

- Remember to add special characters (e.g., "$") before and after each token prior to calculating the trigrams. Add one special character when using bigrams (*the*: *the*), and two if you decide to work with trigrams (*the*: *$$the$$*)
- Use add-one smoothing to account for unseen n-grams during training. In general,

$$P(y|x) = \frac{\text{number of times xy occurs} + 1}{\text{number of times x occurs} + 26}$$

This formula is equivalent to adding one count of each possible bigram in your corpus (assuming there are 26 characters, which is true if you only account for lower case letters.
- Remember to use log probabilities instead of raw probabilities. Otherwise you will get zeroes and the predictions will be useless.
- A decent implementation runs in less than 10 seconds. Yours must run in less than 1 minute.

Submit all your source code and instructions to replicate your experiments on canvas. Write a short report (up to 4 pages, you can write less and get full credit) describing the challenges you found (bug fixes, results obtained, and any other findings). Specifically, discuss the following:

- Do you think it makes sense to create a language model at the character level instead of at the word level for this task? Why?
- Take a look at the test documents for English and Spanish. Are documents written only in one language?
- What is the minimum number of tokens you need to process to always make the right prediction when testing? You can try with 100 tokens, 200 tokens, etc. You do not need an exact number.
- If you create several models for English and Spanish (using different training data, using different preprocessing, etc.), how can you compare them?
- Can you train with less training data and still get the right predictions? How does training size affect predictions during testing? A graph showing how probabilities change is the best way to answer this questions (and a short interpretation of the graph).
- Get some document written in French (e.g., `http://www.gutenberg.org/files/36460/36460-0.txt`) and use your models to predict the language. How can you justify the prediction?

You must answer the above questions in your report.

Extra credit: You can get extra credit experimenting with:

- Trigrams instead of bigrams
- Other smoothing techniques (e.g., linear interpolation)
- Preprocessing the documents (both training and testing documents). For example, run the Porter stemmer (it is language dependent!), remove numbers, remove punctuation, etc. Discuss how preprocessing affects performance.