Bigram Language Modeling

Goals The main goal of this module is for you to implement and play around with a bigram language model, to get experience with these types of techniques and understand what this looks like.

Code Setup

The code you have contains the following files:

```
wiki.train.tokens
wiki.valid.tokens
BigramLanguageModel.java
Main.java
```

You can either run the project on the command line or using `repl.it`. In either case, you want to run `Main.java`.

Your Task

Note that this assignment differs slightly from what's presented in the videos; you will not be implementing `getBestWord` or `getBestSentence`.

Question 1 Look in the `main` method You will see that `queryLm` is called on several different strings. Run the main and it will print the results of these queries before it crashes. This will print out a list of the top words that can follow these contexts along with their probabilities.

What do you notice about the distributions that the LM returns for "I like to" and "I want to"?

Question 2 Implement `sampleWord`. This function takes two arguments: first, the language model itself, and second, a context word to start with.

A sampling algorithm is described in the videos. A `Random r` is declared in Main. You can use `r.nextDouble()` to get a random number between 0 and 1. You should iterate through the vocabulary of possible next words, and as you go, keep a running sum of the probabilities of the words you've seen so far. As soon as your sum of probabilities exceeds the random double, return the result.

Once you've implemented this, run `main` and look at the samples drawn from $P(\text{word} \mid \text{I})$. Because there is randomness, the words may not always look reasonable given the context. Do you think they look reasonable most of the time, some of the time, or rarely?

Question 3 Implemented `sampleSentence`, which calls `sampleWord` to sample whole sentences.

You should initialize new ArrayList of Strings to store the sentence, then repeatedly sample the next word given the previous word given the method you write previously. You can either stop after a fixed number of steps or when you encounter the end-of-sentence token `BigramLanguageModel.END_SYMBOL`.

Once you've implemented this, run `main` and look at the samples being drawn following `I` and `It`. Do you think the sentences being built look reasonable most of the time, some of the time, or rarely?