LINGUIST492A: Homework #3

n-gram models

In this assignment, you will train character bigram models. A bigram character model is like a word bigram model, except that characters, rather than words, are the events whose probability you are modeling.

In this homework, you will perform a language identification task. Your goal is to train a system that can distinguish Greenlandic from Ilocano text. Greenlandic is an indigenous languages of North America. Greenlandic (Kalaallisut) is the only indigenous language of North America that is an official language of government (in Greenland).

Ilocano is an Austronesian langauge like Tagalog, Hawaiian, Malay, and others. It is spoken by more than 8 million people in the Northern Luzon region of the Phillippines.

Learn more about these languages on the Kalaallisut wiki page and on the Ilocano wiki page.

The primary goals of this homework are to implement realistic statistical inference, including various smoothing methods, and evaluate the performance of different smoothing techniques by testing your model's generalization to unseen text.

Part 1: Build a bigram model

Download the Greenlandic and Ilocano texts from the Moodle (greenlandic.txt and ilocano.txt, respectively). Each text is a single string representing a text in each language. Specifically, each text is the Universal Declaration of Human Rights in the respective langauge.

In Part 1, we would like you to write one function:

- Write a function that takes a list (**not** a string!) as input and creates from this a bigram language model based on this data. In this assignment, the list argument to this function should be a list of all the characters in each text. The bigram model should represent the conditional probably of every character c_i in the corpus given every other character c_j. That is, your function should return some representation of $P(c_i|c_j)$ for all possible values of c_j and c_i. You may choose to put a start and end symbol at the ends of the corpus, but you do not need to.

For this assignment, do not strip spaces from your text. Your bigram character model should account for the probability of a space character occuring in a sequence as if it was any other character.

Hint: You may also want to allow the user to explicitly pass a vocabulary argument, which is a list (or set) of all the characters whose probability will be estimated from the data. This is an important part of error handling for this assignment, because the characters used in Greenlandic are different from those used in Ilocano! For this assignment, it might be good to set the 'vocabulary' argument, by default, to the list of all characters in the standard Latin alphabet.

Your function should also implement Add-*k* smoothing, such that the user can pass the smoothing parameter *k* to your bigram model function to determine how much smoothing they would like.

Advice: We will be testing your function on standardized data, to make sure it yields the correct values! Therefore you want to be sure it is giving you sensible probabilities before turning it in. But it can be difficult to know if your function is doing something sensible with all this data. We recommend you test your function on a very small, toy corpus, small enough to calculate the correct probability values by hand, so you can be sure it's giving you the right values!

More advice: Program this assignment in chunks. First, write the function for part 1, and check that it is doing something senisble. Then, write the function for part 2, and do the same. When you have both of those in place, then move on to part 3.

Part 2: Define functions to calculate perplexity and cross-entropy

In part 2, we would like you to write one function:

- A function called `perplexity` that will calculate the perplexity of a test text, given a bigram model. Your function should take i) a set of test data, ii) a bigram model (from Part 1), and return the perplexity of the test data with the bigram model.

A note on calculating with probabilities / perplexity

You should never multiply probabilities in your code. Probabilities are extremely small, and when you multiply them together, they get even smaller. By round-off error the total probability will come out as 0 for any sizeable input. This is why, in practice, you should work with the logarithm of probabilities in your code. In your implementation, you should **not** try to directly implement equation 3.16 from Jurafsky and Martin. Instead, you should take the logarithm of the probabilities, use this sum to find the perplexity. In other words, use the equation in 3.52 (and the class notes):

$$Perplexity = 2^{-\frac{1}{n}\sum_i logm(x_i)}$$

Where $m(x_i)$ is the probability of the ith character given your bigram model.

To find the log of a number x in python, import math, then type "math.log(x, base)" so the log base 2 of 32 would be "math.log(32,2)" . To implement perplexity for a given text, you should find the log probability (base 2) of each character by summing the logs of each of the n-gram probabilities of each character in the text. This could be done by keeping a running sum of the log probabilities of all the words (lets assume this sum is called "log_sum").

Part 3: Putting it all together

In part 3, we would like you to put it all together in `bigram_stub.py` . Using the functions you wrote in Part 1 and Part 2, round out the stub script so that your script does the following:

- Loads in both the Ilocano and Kalaallisut training data.
- Fits a Kalaallisut character bigram model and a Chicksaw one. Both models should be fit using the same smoothing parameter, which the user should be able to pass in as a parameter on the command line.
- Calculate the perplexity of a test sentence on both the Ilocano and Kalaallisut models. The user should be able to pass a test sentence as a string from the command line.
- Print out a decision about which language the test sentence is likely to be, by choosing the language with a better perplexity score.
- Print out whatever additional information you find useful for answering the questions in Part 4.

Deliverable for Part 3: Turn in a modified version of the `bigram_stub.py` script that includes the functions from parts 1 and 2, and does what is asked in part 3. Specifically, the user should be able to run the script and specify a smoothing parameter and a test sentence, and the script should print whether the test sentence is Ilocano or Greenlandic.

Part 4: Questions

Finally, use your script to run some experiments to address the following questions:

- Choose some random sentences of Kalaallisut text from this Kalaallisut wikipedia page, and Ilocano text from this Ilokano wikipedia page. Run your script on the test sentences you chose. How accurate was your script at identifying the target language? Report your results. What are the features of the target language that aided classification, do you think?
- Try and identify some single words in the Ilocano and Kalaallisut webpages that you think

your script might misclassify. To try and come up with these difficult words, inspect the bigram models you fit - try to find high and low probability character bigrams sequences in each language, and select words in both Kalaallisut and Ilocano that you think will be misidentified. Explain why you chose the words that you did: What character bigrams did you target, and why? Did you successfully trick your bigram models?

- Explore the performance of your model across different values of the smoothing parameter. What happens if you do not do any smoothing, that is, you just use Maximum Likelihood Estimation of the probabilities? Does the model's ability to classify the test sentences you chose vary with different smoothing parameters?
- Inspect the probability values given to character bigrams that don't occur in a text, e.g., the unobserved character sequences. How does that probability value change as you change the value of the smoothing parameter? Explain why.
- Lower perplexity scores indicate a better model fit to a set of test data. Explain why in your own words.