Generating Shakespeare

I trained some unsmoothed, uninterpolated n-gram models on some Shakespeare corpus. The results are below:

C = 2 (trigram)

Output:

Fire st, re. Hence lestake; goge,
To loven fillow didir.

PRIAGE:
Thisherthe spilve,
TH:
HAISBY Bito majescles notents the haveretillse,
ULINSIO tows arm, boyall to ge, shave,
A for hing, mand the oneresalks nam noinhor he words the ther ling.

LAULO

C = 3 (4-gram)

Output:

First thy disage
And Perpres your counds,
They fair grack, make that wings.
There and to don's sir?

PARINCESTER:
On Nept,
Thes we withould not hast! 'The but so dow, in of his occast to case you, and Timon.
As but trum of a prothis mars.

KING HENRY

C = 4 (5-gram)

Output:

First in this
more: never you may blood,
As that ever, is at life, thee versecurite.

Clown from wrong season or serves absence, and, so grove York, people: but to, for had as her vision on the moon-calf
flap is tragic, but a wonder, nor way
not no g

C = 7 (8-gram)

Output:

First Citizen:
That Englishman;
Shall this if she becomes the which
I find him to his God, the like those foundations leave, madam, such a trumpery; nothing.
I'll get 'em
But on the sun,
And make my hearts!

LUCIUS:
Dead.

DUCHESS OF AUVERGNE:
If it

Above, it is shown that as c increases (the context and determiner of the n-gram), the passages generally becomes more comprehensible. This is due to the fact that the model generates text character by character instead of word by word, so for a lower c the words are not that likely to make any coherent sense. The text that is generated when c = 7 is by far the most similar to Shakespeare, and we can even see some phrases included within the text like "First Citizen: ," "make my heart," and "that Englishman." It is safe to assume that at high values of c, the model may copy Shakespeare directly. Also, all of these examples begin with the letter "F". It makes sense because the first word in our training file is "First," making it the only word with start tokens ("~") prior to it. In fact, the first two words in our training data are "First Citizen:" which appears in our example when c = 7.

Perplexity and Add-k Smoothing

Next, I try improving the initial N-gram model by adding perplexity and add-k smoothing to account for zero probabilities. I compared the perplexity of Shakespeare's sonnets and a New York Times article to the original training dataset of Shakespeare's plays used previously. The results are below:

K = 1, C = 2

Shakespeare input (training data) perplexity: 50.181172436232586

Shakespeare sonnets perplexity: 35.89266116626415

New York Times perplexity: 53.11944813300934

K = 1, C = 3

Shakespeare input (training data) perplexity: 50.89927522783591

Shakespeare sonnets perplexity: 36.679544023391756

New York Times perplexity: 36.939002235701

K = 2, C = 2

Shakespeare input (training data) perplexity: 46.52141833802016

Shakespeare sonnets perplexity: 33.930221135511886

New York Times perplexity: 50.32534621936141

K = 2, C = 3

Shakespeare input (training data) perplexity: 46.63342222978484

Shakespeare sonnets perplexity: 35.6234895556057

New York Times perplexity: 35.516392100620344

K = 4, C = 2

Shakespeare input (training data) perplexity: 43.103286178190906

Shakespeare sonnets perplexity: 32.21265290426591

New York Times perplexity: 47.811787436134445

K = 4, C = 3

Shakespeare input (training data) perplexity: 43.845440979711796

Shakespeare sonnets perplexity: 35.6370466467556

New York Times perplexity: 35.11107605743637

As we can see above, the perplexity for Shakespeare Sonnets is much lower than the perplexity for the New York Times article, indicating that the Shakespeare Sonnets are similar to Shakespeare's plays, which is to be expected since they are written by the same author in the same time period. Furthermore, as k increases, the perplexity decreases for all text files. As c increases, the perplexity between the Shakespeare sonnets and the New York Times article become much closer to each other while the perplexity for the training data doesn't change much at all.

Interpolation

The default value for our lambda was initially set to 1/c+1, meaning that if c = 2, each lambda in the input would have a weight of ⅓, or 0.33. To test different values of the lambda values and how they affect the model, consider the text 'abab' and c = 1 (bigram model). If we consider the text without interpolation applied to the model, the probability for the bigram ('a', 'b') is 1. The result of the same bigram, ('a', 'b'), is 0.75. This is due to the adjusted probability by the lambda values. In this case, all lambda values are 0.5. If we reduce those lambda values to 0.25, we get a lower probability of 0.375. If we use two different lambda values for this bigram model, such as 0.2 and 0.3, we get a probability of 0.4 for the bigram ('a', 'b'). This shows that the lambda values can have an effect on the probability values of n-grams in our language models using interpolation.

Language Identification using n-grams

To configure my models into language identifiers, I trained each model for each given training text file using the uninterpolated version of the n-gram language model. I went through each string in each validation text file and calculated the probability for each model for that particular string (city, in this case). From all these probabilities collected from each model that was run, the greatest probability and its associated language are the predictions for that city. I then compared each of these predictions to the actual language(country) to get the accuracy. I did this for all strings in each validation file and found the accuracies of each model below:

af Afghanistan: 0.35 (35% accurate)
cn China: 0.77 (77% accurate)
de Germany: 0.45 (45% accurate)
fi Finland: 0.55 (55% accurate)
fr France: 0.62 (62% accurate)
in India: 0.3 (30% accurate)
ir Iran: 0.34 (34% accurate)
pk Pakistan: 0.54 (54% accurate)
za South Africa: 0.7 (70% accurate)

Overall, the most accurate model was cn (China) and following za (South Africa). This makes sense because the Chinese language is very specific and has combinations of characters that no other language model in this group has, making it relatively easy to identify.