Kashish Gupta & Shiva Suri

Computational Linguistics (CIS 530)

Homework 5 Writeup

What the F?

When generating text in Part 1 after training on "shakespeare_input.txt," the first letter in the paragraph generation was consistently 'F': this is because the very first word of the textfile on which we trained was "First," so when the model sees no history, there is a 100% chance that the generated character will be an 'F.' If we had trained with more files with different first letters, this would not be the case. When the order reaches 4, the first word generated seems to always be 'First.' This has to do with how we are padding the very beginning with '~'s--we pad as many tildas as is the order. Theoretically, we believe that when the order is 6, the first word generation of "First" is 100% guaranteed because "First " is 6 characters long (note the space at the end; variations of 'First' such as 'Firstmost' and 'Firstly' exist.)

We continued the assignment by performing an intrinsic evaluation to test the goodness of our language model. Namely, we wrote a function to calculate the *perplexity*, and noted that the perplexity of the Shakespearean sonnets was 6.7886 whereas New York Times' text perplexity was roughly 12.6912 when order was 4 and k-value was 0.5, which makes sense because we trained on a Shakespearean text, so character combinations will be more familiar (and less perplex!) We used add-k smoothing to get rid of zero values for the perplexity calculation. Below is the formula we used to determine perplexity:

$$\begin{aligned} Perplexity(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}} \end{aligned}$$

We ultimately want to reduce perplexities, especially for the Shakespearean sonnets to validate our language model. We experimented by varying our k-value for the smoothing factor and by testing different sets of orders for the language models.

K-value	**Order**	**Sonnet Perplexity**
0.25	1	12.324906987580107
0.25	2	8.004735046892488

0.25	3	6.098105536504878
0.25	4	6.219678058370482
0.5	1	12.287030974989076
0.5	2	7.999816098798104
0.5	3	6.256596066835674
0.5	4	6.788634256001164

Here, we can see that perplexity is minimized when the order is 3. Now, let's find the optimal k-value given an order of 3.

K	Perplexity	K	Perplexity
0.01	6.053206524750025	0.1	6.009626464337175
0.02	6.018780184029833	0.2	6.066437844318308
0.03	6.004812088318713	0.3	6.130162994044898
0.04	5.99866337741245	0.4	6.194000057437073
0.05	5.996544112004246	0.5	6.256596066835674
0.06	5.996828922614706	0.6	6.317650401694769
0.07	5.998680429192945	0.7	6.377151301658576
0.08	6.001614978093654	0.8	6.435175358880319
0.09	6.005330345328451	0.9	6.491823402234225

Our results suggest that a k-value of 0.05 is optimal for minimizing perplexity. Furthermore, the perplexity of the new york times article with k-value of 0.05 and order of 3 is 10.813343965969292 confirming our hypothesis as the perplexity went down.

Optimizing Lambda values:

We measured the accuracy of different lambda values testing across the leaderboard.

λ1	λ2	λ2	Accuracy

0.333	**0.333**	**0.333**	**0.6755**
0.200	0.200	0.600	0.6677
0.300	0.300	0.400	0.6744
0.4	0.4	0.2	0.6677

The first row with equivalent lambda values yielded in the best performance.

Why we think our model performed so well:

In order to compute P(D | c) we read the paper linked in the handout and used the sum of log probabilities of each character in the city name. Furthermore, we did not pad the beginning of the city names with '~' characters, rather we skipped the first 'order' number of characters and started from character at index 'order.' This allows us to avoid the trap where the language model was always predicting F for the first letter of the generated text because it had never seen anything else. It also makes the code easier to write, and we believe this is why our classification performed well.