Report for Homework 3

1. Part I

1.1 Generating Shakespeare using a non-smoothing and non-interpolative n-gram model. Here is a brief overview.

n	output
2	Finuarst losenceigh of cus'd mazy, suffeld-sent, my her's the anuch thy son ar, you wittake, I theity, a my how bethou may! LE: Go th Thond allooddeare and tone thimarge, as Sir. KINDARD: Yeakengue hapere, Frin youn ford riker, may way bes, wit, ho
3	First trike is the Lord ture, the cand, or In rious eater as and firs? CAIUS: Sweepill neal Thought they cool, way overe. THELENA: Sir our delanus, that took; but nor they so more his boy would forth. CASSIO: Which I conded warms truction; and mer
4	First had it not him. By that is worth such as of delive, in be you walk, To further. GLOUCESTER: Let me to the be another, sir, sweary rash thy where new for song, go will one they his gone. But, when let the done. Give meternal parks, or purposed
7	First Citizen: We'll buy him off, And she incites me a living. O thou dost snore upon John is taken in't. CLOTEN: I have work on him, That they will spare the sceptred sway;

	It was this, And happy too: The obsequies: come, Cut three months old.

Overall, the quality of the generated text increases as the number of n increases. Since the model generate text by guessing next most possible characters rather words then when there is a small n, we may find the sentence is grammatically incorrect.

With a high n, the sentence seems correct in grammar and carries some meaning. I think if n gets large enough, this heuristic is probably better than 1000 monkeys working at 1000 typewriters.

Discussion: Why the first word is always “first”?

Because the first character’s name in the training file is “first citizen” which is padded with “~”. Then when the model starts to generate its first character (i.e., predicting char given “~”*n context), “f” turns out to be the most likely one which then becomes “first”.

2. Part II Perplexity, Smoothing and Interpolation

Discussion: First what c, k value should we use for our n-gram model. After pinning down c and k, how do we choose the coefficients for interpolation.

To choose the right c and k, I first trained an un-interpolated model and an interpolated model with equal weights. Result is the following.

c	k	Text type	Uninterpolated	Interpolated (equal we
0	1	Shakespeare Sonnets	23.447	23.447
0	1	NYTimes Article	27.661	27.661
0	2	Shakespeare Sonnets	23.447	23.447
0	2	NYTimes Article	27.633	27.633
1	1	Shakespeare Sonnets	11.971	14.247
1	1	NYTimes Article	14.238	16.741
1	2	Shakespeare Sonnets	11.974	14.252
1	2	NYTimes Article	14.306	16.808
2	1	Shakespeare Sonnets	7.582	10.195
2	1	NYTimes Article	10.058	12.417
2	2	Shakespeare Sonnets	7.662	10.261
2	2	NYTimes Article	10.284	12.586
3	1	Shakespeare Sonnets	5.732	7.999
3	1	NYTimes Article	8.239	10.209
3	2	Shakespeare Sonnets	6.063	8.208
3	2	NYTimes Article	8.997	10.615
4	1	Shakespeare Sonnets	5.695	7.097
4	1	NYTimes Article	8.205	9.221
4	2	Shakespeare Sonnets	6.605	7.498
4	2	NYTimes Article	10.119	9.964
5	1	Shakespeare Sonnets	6.948	6.833
5	1	NYTimes Article	10.222	8.978
5	2	Shakespeare Sonnets	8.957	7.436
5	2	NYTimes Article	13.993	10.033

We can see the result of the interpolated model consistently outperforms the un-interpolated model. We get lowest perplexity scores for both models when (c, k) = (4,1).

Also we see for any model, the perplexity score of Shakespeare Sonnets is lower than NYTimes, which is expected as the training data is also from a corpus of Shakespeare's writing.

Then we will further optimize the coefficients based on (c, k) = (4,1).

We tried some exemplary lambda values as listed below. One strategy employed is assigned greater weight to models with higher c values, which seems work better in this case.

Text	Lambdas	Perplexity
Shakespeare Sonnets	[0.2, 0.2, 0.2, 0.2, 0.2]	7.097
NYTimes Article	[0.2, 0.2, 0.2, 0.2, 0.2]	9.221
Shakespeare Sonnets	[0.018, 0.073, 0.164, 0.291, 0.454]	5.916
NYTimes Article	[0.018, 0.073, 0.164, 0.291, 0.454]	8.397
Shakespeare Sonnets	[0.16, 0.16, 0.2, 0.24, 0.24]	6.817
NYTimes Article	[0.16, 0.16, 0.2, 0.24, 0.24]	9.243

Therefore, I conclude in my experiment, (c, k, lambdas) = (4, 1, [0.018, 0.073, 0.164, 0.291, 0.454]) seems to work most effectively.

3. Part III Text Classification

Given the task of given a city name, predicting the country it belongs to, we can use train |country| n-gram models in which each model predicts the likelihood of each country and we choose the most likely one as the predicted value.

First experiment with every possible (c, k) pair in which c ranges from 0 to 5 and k ranges from 0 to 3 to train two models. First model is an un-interpolated model and the second is an inter-polated model with equal weights. See attached below.

c	k	Uninterpolated	Interpolated
0	0	49.222	49.222
0	1	49.222	49.222
0	2	49.222	49.222
0	3	49.111	49.111
1	0	64.000	62.889
1	1	64.778	62.889
1	2	64.333	62.889
1	3	64.222	62.889
2	0	51.333	66.222
2	1	65.778	67.222
2	2	65.778	67.222
2	3	65.778	67.111
3	0	26.556	67.333
3	1	65.667	68.222
3	2	64.667	68.111
3	3	61.667	67.889
4	0	17.667	66.111
4	1	59.667	69.222
4	2	56.667	68.000
4	3	53.000	66.333
5	0	13.667	65.111
5	1	53.000	68.111
5	2	50.556	66.667
5	3	46.111	65.667

We can also see this pattern that equal-weighted interpolated model consistently outperforms the un-interpolated one. And we can see when (c, k) = (4, 1), we get the best outcome, which is in line with previous experiments.

Then we will use (c, k) = (4, 1) to further experiment with lambdas. See the result on the development set attached.

(c, k)	Lambdas	Perplexity
4, 1	[0.018, 0.073, 0.164, 0.291, 0.455]	67.889
4, 1	[0.2, 0.2, 0.2, 0.2, 0.2]	69.222
4, 1	[0.16, 0.16, 0.2, 0.24, 0.24]	69.444

We can see [0.16, 0.16, 0.2, 0.24, 0.24] is marginally better than the equal weighted result. But after I submitted to the leaderboard, the performance is the opposite. Then we cannot really conclude which one is definitely better. I submitted the equal weighted model to the Gradescope.