Ian MacDonald

CIS 530

Homework 3 Writeup

Basic n-gram Models

Generating Shakespeare

In this section, n-gram models were trained on various works of Shakespeare, and then told to generate their own 250 character length texts. For each value of n from 2 to 7, a different model was trained, with each one's generated text below.

N	Generated Text
2	Fir? le feal'd ne diest! In shourt now come That hous I bre flor quarrachad? CLEARSICK: If wity lateakee he uster liken my the he traseace he gives possio's dou beed knothe honce. CAESEY: 'Moon; I come, To to murta: I puty cauce; gaire yout hat foo
3	First fee.' Hight; Whath my hat of your dution To expent, I would by and reven, The ward, your. CASSALANDARUS: Thirdly goes it wipe, my love to must rest. Give This pray her reignoral did, door: Whose, my could to for most: Their vison his givil nig
4	First that more complater the me not be in the left as shall do some the Earl of thank you has thrice what need or a justly compler him, Pities the more her like thank you. SUFFOLK: Fond Servation you loves bird? Might in you in other love to flash

	Shakespeare Generated Text: c=5, k=0 First Clown: Macbeth That's pardon me bargain; draw not for reverend like duty thy chester Anne.
5	First Clown: Macbeth That's pardon me bargain; draw not for reverend like duty thy chester Anne. KATHARINE: Indeed I shall the matter The arch of this father in prince amongst them, are well. A murderers he not me, do; now quest you well as 'twill b
6	First wag thee henceforth our strength such as fearful-hanging. Further, not speak? When he complete with snake, In number'd o'er. Fie, fie! teach her with his fair dead of this foot, and their malt with: there's my staff? look up the rather, For thy
7	First Citizen: Your wonder'd to retort you now, we will destruction that is here to use as stone a-rolling, and many men depart,--that, in that profit. ELY: The thoughts be you recount thee out so prettily and time hath one consequence! Courtezan:

As the values of n increase, the generated text begins to make more and more sense. This occurs because the n-gram models are based on characters rather than words, which means that low values of n are creating words based off of only 2 or 3 characters, which is rarely enough information to make actual words, other than small simple ones (he, if, my, etc.). But, once the n values reach 4,5,6, and 7, words begin to form, and the text starts to look more like an actual story at first glance. However, upon closer look, the grammar is basically nonexistent and the actual words together make a nonsensical story.

The beginning word in each generated text always starts with an “f”, and after a certain amount of characters the beginning word will be “first” every time. This occurs because the model is training off of one large text where the first word is actually the word “first”. Because there’s only one input text, the only beginning letter the model has seen is the letter “f”, and at a large enough n, the only beginning word the model has seen is “first”, which means when it comes time to generate it’s own text, since “first” is the only word the model has seen at the beginning of the text (a unique case since we have the start of the text padded), it will generate that word every time as it is the only word it knows that fits that situation.

Perplexity, Smoothing, and Interpolation

This section adds smoothing and interpolation to the previously created n-gram models, then measures their performance by calculating the perplexity against other works of Shakespeare (specifically, some of his sonnets), as well as a New York Times article. Each model was trained on the same Shakespeare training set as before, and various different values for k, c, and lambda were tested in order to determine the effects varying each of them has on the test set perplexity values.

Varying k value, c value set at 4, lambdas set equal:

Test Set	k	c	Uninterpolated	Interpolated
Shakespeare Sonnets	.1	4	5.484399200017636	6.846294956071371
Shakespeare Sonnets	.5	4	5.973412238120488	7.137402971619595
Shakespeare Sonnets	1	4	6.485626437020774	7.378775753991396
Shakespeare Sonnets	2	4	7.31262845142698	7.728982160532891
New York Times Article	.1	4	9.654650150675293	9.726097308545807
New York Times Article	.5	4	10.301876277521302	10.096823463506924
New York Times Article	1	4	11.125617645800839	10.409097078272238
New York Times Article	2	4	12.437810989333128	10.844757543316039

The perplexity values for Shakespeare's Sonnets are much lower than those for the New York Times Article, which makes sense since the models were all trained on a set of texts written by Shakespeare. The lower k values seem to perform best here, with the k value of .1 performing best for both uninterpolated and interpolated models on both test sets. Additionally, interpolated models seem to work better for the New York Times Article, but uninterpolated work better for Shakespeare's sonnets.

Since the k value of .1 worked well above, it was fixed at .1 for the following experiments.

Varying c value, k value set at .1, lambdas set equal:

Test Set	k	c	Uninterpolated	Interpolated
Shakespeare Sonnets	.1	2	7.996946415762461	10.288650009339838
Shakespeare Sonnets	.1	3	5.883794503961181	7.961566982637632
Shakespeare Sonnets	.1	4	5.484399200017636	6.846294956071371
Shakespeare Sonnets	.1	5	6.304539471924622	6.358891990883289
Shakespeare Sonnets	.1	6	8.595336514841614	6.250214957109947
Shakespeare Sonnets	.1	7	9.232587005736804	6.379605834276571
New York Times Article	.1	2	11.868267891525216	13.662849747389656
New York Times Article	.1	3	10.112745688329479	11.167435244147844
New York Times Article	.1	4	9.654650150675293	9.726097308545807
New York Times Article	.1	5	11.968908404718281	9.22360833778678
New York Times Article	.1	6	16.691756828310826	9.232587005736804
New York Times Article	.1	7	24.413003413286653	9.517284238626146

For c values, the ideal c seems to be around 3,4,5, or 6, depending on whether or not you are looking at the Shakespeare test set or New York Times test set, and whether or not you are looking at the interpolated or uninterpolated model. Regardless, they all exhibit similar behavior, in the sense that as the c gets larger the perplexity goes down, until it hits a point where the perplexity starts getting larger as c gets larger. For interpolated models, the increase in perplexity after hitting that point is less drastic than that of the uninterpolated models. This makes sense, because interpolation smoothes the probabilities out over all the characters, so adding more characters to the context won't hurt it as much as it would when adding them to the uninterpolated model.

Since the c value of 4 worked well above, it was fixed at 4 for the last experiment.

Varying lambdas, k set at .1, c set at 4:

Test Set	k	c	lambdas	Interpolated
Shakespeare Sonnets	.1	4	[0.25, 0.25, 0.25, 0.25]	6.152562500624543
Shakespeare Sonnets	.1	4	[0.1, 0.2, 0.3, 0.4]	7.000380738836457
Shakespeare Sonnets	.1	4	[0.4, 0.3, 0.2, 0.1]	5.620046957719924
New York Times Article	.1	4	[0.25, 0.25, 0.25, 0.25]	8.902817057442693
New York Times Article	.1	4	[0.1, 0.2, 0.3, 0.4]	9.819512332516313
New York Times Article	.1	4	[0.4, 0.3, 0.2, 0.1]	8.445109658138115

There was similar behavior for both test sets, as the best lambda combination was [0.4, 0.3, 0.2, 0.1] for both, followed by [0.25, 0.25, 0.25, 0.25] and [0.1, 0.2, 0.3, 0.4] in that order. This indicated the best performance occurred when placing a heavier weight on higher order n-gram probabilities, and the worst performance occurred when placing a heavier weight on lower order n-gram probabilities.

Text Classification Using n-grams

For this section, I implemented both uninterpolated models as well as interpolated models, trying out various different values for all of the possible hyperparameters (c, k, lambdas). The hyperparameter values tried were similar to the hyperparameters tried in the previous sections, as those seemed to give a wide, varying selection of models to choose from. Below are all of the different model/hyperparameter combinations I tried, and their performance on the development set .

Uninterpolated

c	k	Development Accuracy
2	.1	0.6311111111111111
2	.5	0.6566666666666666
2	1	0.6577777777777778
2	2	0.6577777777777778
3	.1	0.6233333333333333
3	.5	0.6466666666666666
3	1	0.6566666666666666
3	2	0.6466666666666666
4	.1	0.5922222222222222
4	.5	0.6111111111111112

4	1	0.5966666666666667
4	2	0.5666666666666667
5	.1	0.5466666666666666
5	.5	0.5533333333333333
5	1	0.53
5	2	0.5055555555555555
6	.1	0.5033333333333333
6	.5	0.5011111111111111
6	1	0.48333333333333334
6	2	0.45666666666666667
7	.1	0.4855555555555556
7	.5	0.4711111111111111
7	1	0.4633333333333333
7	2	0.44333333333333336

Interpolated

c	k	lambdas	Development Accuracy
2	.1	[0.9, 0.1]	0.6511111111111111
2	.1	[0.75, 0.25]	0.6544444444444445
2	.1	[0.6, 0.4]	0.66
2	.5	[0.9, 0.1]	0.6611111111111111
2	.5	[0.75, 0.25]	0.6577777777777778
2	.5	[0.6, 0.4]	0.6644444444444444
3	.1	[0.8, 0.15, 0.05]	0.6455555555555555
3	.1	[0.7, 0.2, 0.1]	0.6544444444444445
3	.1	[0.5, 0.3, 0.2]	0.6722222222222223
3	.5	[0.8, 0.15, 0.05]	0.6833333333333333
3	.5	[0.7, 0.2, 0.1]	0.6911111111111111
3	.5	[0.5, 0.3, 0.2]	0.7033333333333334

The uninterpolated models were tested first, and performed alright. However, there was a clear decrease in accuracy as the c values increased. The k values did not seem to change the accuracies too much, but it still seemed that .1, .5, and 1 would consistently perform better than 2. After this, the interpolated models were created and tested. I chose values of .1 and .5 for k, since the previous experiments as well as the uninterpolated model in this section indicated that those were good k values to try. I chose c values of 2 and 3 since the uninterpolated model performed far better with these values as compared to higher c values. Finally, I chose various lambda combinations, but all of them gave more weight to the higher order n-gram probabilities, since the previous experiments strongly indicated that that leads to better models. On average, the interpolated models vastly outperformed the uninterpolated models, and therefore unsurprisingly the best model was an uninterpolated model with c=3, k=.5, and lambda=[.5, .3, .2], with an accuracy of just over 70%. This model was the one chosen to try on the leaderboard testing set. The model generalized very well, as the testing accuracy was extremely close to the validation accuracy, coming in at just under 70%.

Error Analysis

Country	Development Accuracy
Afghanistan	.63
China	.95
Germany	.69
Finland	.81
France	.82
India	.51
Iran	.59
Pakistan	.67
South Africa	.66

A closer look into the error by country rates shows some interesting trends for the final model. For some reason, the model does extraordinarily well on Chinese cities, with an accuracy of 95%. It also does pretty well on Finland and France, scoring above 80% for both. However, the model performs quite poorly on India and Iran, with sub-60% accuracy for both. A possible reasoning for the accuracy disparities is that some countries just might have city names very similar to those of another country on the list. For example, the model performs pretty well on Finland, but I wonder if it would perform that well if a country such as Norway or Sweden was added in.

Shown below are a few examples of cities the final model predicted incorrectly:

City Name	Actual Country	Predicted Country
qareh burun	Iran	Afghanistan
sepahdar	Iran	Afghanistan
satin	Pakistan	India
shinalai	Pakistan	India
vilafrime	Afghanistan	France
alluana	Pakistan	Finland
mamfenweni	South Africa	China

A closer look at the individual cities missed shows some understandable misses – Iran and Afghanistan are geographically next to each other, and likely have pretty similar language patterns, which could explain why the model predicted Afghanistan for a few cities in Iran. A similar logic applies to the missed cities in Pakistan that were predicted to be in India. However, there's some misses here that are unexpected – Pakistan and Finland are nowhere near each other and have populations that speak largely different languages, but the model predicted a city in Pakistan to be in Finland. Even looking at the city name though, it is difficult to figure out why exactly the model is getting it wrong and how it should be tweaked to perform better, and that applied to a lot of examples. Additionally, there may be some mislabeled or misspelled city names in the test set, as the model predicted the city of "vilafrime" to be in France rather than Afghanistan. However, a quick google search of "vilafrime" actually doesn't return a city in either Afghanistan or France, it returns a city in Spain, so this one can likely be chalked up to a data error, and there may be more like that upon closer inspection.