Diep (Emma) Vu

Project 1 Write Up

Abstract

In this write up I will present my experimentation for N-gram models with unsmoothed and smoothing techniques and with interpolation on the evaluation test of Shakespeare Sonnets and NY Times articles. I will also examine the text classification task on country codes and city names.

I. Basic N-gram Models

I trained some unsmoothed, uninterpolated N-gram models on Shakespeare corpus and here is the results shown in this table, where *c* is the number of characters:

c	Output
2	Firitheepost to: mer ne thatenchipearsent HENTIANDOLING He's payssubtle tand me, ty's bove therin; NO: Is sler purea. Her it; wark, liall comion of thople dong! tactizan ane shourin hat! haputy bre Thown buty, EDWALODON: Goord: GUES: Diank ant'strab
3	Fires parth the hat let forth Is arted, I know. Pyrance. What duloud Humannoth to-morn batent to compleaves it withose most repeak to what the of hee writ, hast of it know't; anot must Sergot And her that do? FERDINERIGO: God Lucius good in prest to

4	First Senatorious they captain: The name. What, I will do not did this funeral line of the me note. OPHELIA: DROMIO OF YORK: Away, I would take heaven But wilt: if the great womans hark, What I may as it before to Some one. DUKE OF YORK: Please Th
7	First Citizen: Peace, person in your own life hath eat of her rest. KATHARINA: Not I, believe not see, There is a life! MERCUTIO: A charity, all go along imprisonment. How now, my as fair-play; His sceptre, Unless, by filling the mighty prelate Int
10	First Citizen: For your fair show shall we be sunder'd? shall we about it? ORLEANS: You are too shallow and too quick. ROSALIND: Out, fool! THERSITES: He must think, if we give you battle presently go learn me the proclamation; but sure, though th

Table 1: Generated Shakespeare text from basic N-gram model

We can see that as *c* increases, the output becomes more coherent. When *c = 2,* the output hardly makes any sense versus when *c = 10,* we can understand the output better. This is because the model is based on generating text character-by-character, rather than word-by-word, so for small *c* the words are unlikely to make any sense. Furthermore, the text generated by *c = 10* is the most similar to Shakespeare, and we can see elements of Shakespeare present in the text, such as “Shallow”, “proclamation”. It is likely that at this high value for *c*, the model is just copying Shakespeare’s words from the training dataset.

It’s worth noting that all of the passages generated always start with “F.” Perhaps this is due to the fact that the first word in the training file is “First,” so it is the only word padded with tildes “~”. Thus, since the first context given is always the pad, the first word generated is always “First” when the value of *c* is large enough.

II. Perplexity, Smoothing and Interpolation

One way to improve the basic N-gram model is to add perplexity, smoothing and interpolation. I compared the perplexity of a New York Times articles (NY Times), a selection of Shakespeare’s sonnets (Sonnets) to the Shakespeare training dataset and here is the result To calculate these perplexities, I concatenated all paragraphs first. Then I fix *c = 2* and vary *k* from 1 to 4. For the interpolated models, the weights of λ are equal (so $\lambda_i = 1/(c+1)$, where $i \in [0, c]$). Here are the perplexity results:

When *c = 2*	**k**	**NY Times**	**Sonnets**
Uninterpolated	1	11.018771417968004	7.910100720888779
	2	10.980952044452271	7.932029974368767
	3	10.991411266905272	7.966470772488156
	4	11.017864688791038	8.004420975054114
	7	11.128112204406703	8.122121129774436
Interpolated	1	13.026137835867438	10.328866826678306
	2	13.031292211555522	10.370481247383
	3	13.049554289526354	10.409329628452168
	4	13.072232911836327	10.445961829746775
	7	13.148047484616907	10.546335205417773

Table 2: Perplexities of different text types to Shakespeare’s Plays on Uninterpolated and Interpolated N-gram models when *c = 2, k vary*

As we can see above, the perplexity for Shakespeare Sonnets is much lower than the perplexity for NY Times articles, indicating that the Shakespeare Sonnets are similar to Shakespeare's plays, which is expected since the Sonnets and Plays are by the same author and time period. Furthermore, as *k* increases, the perplexity decreases for both text types. The perplexity for the interpolated models is also higher than the perplexity for the uninterpolated models for both text types.

Next, fix *k = 2* and vary *c*. For the interpolated models, the λ's are still equal. The results are below:

When *k = 2*	**c**	**NY Times**	**Sonnets**
Uninterpolated	1	14.879866078127524	12.20273897297545
	2	10.980952044452271	7.932029974368767
	3	10.135816046531493	6.388148189027978
	4	12.163357190547316	7.31262845142696
	7	**38.852973975340625**	**26.549946745914312**
Interpolated	1	17.181687490029898	14.273089483911228
	2	13.031292211555522	10.370481247383
	3	11.134214572426878	8.364833744177451
	4	10.596109224342902	7.728982160532993
	7	**12.260098046699648**	**8.758155870973066**

Table 2: Perplexities of different text types to Shakespeare's Plays on Uninterpolated and Interpolated N-gram models when *k = 2, c vary*

With a fixed *k*, we can see that once again, the perplexity for Shakespeare Sonnets are much lower than NY Times articles, thus vary more significantly. This supports the fact that the Shakespeare Sonnets are most similar to the Shakespeare Plays from the training inputs because they both come from the same author and around the same time period. The interpolated perplexities are also higher than the uninterpolated perplexities on average, with the exception of when *c = 7*. Interestingly, the uninterpolated perplexity spikes when *c = 7*.

Next, I continue the experiment by varying λ when *c = 3, k = 2*. I tried an even weighting of λ, a heavier λ on smaller N-gram models, and heavier weighting on larger N-gram models. It is suspected that the perplexity of the interpolated model with larger λ for larger N-gram

models to be smaller, since more of the context is considered. Here are the results which supports the hypothesis:

When c = 3, *k* = *2*	**λ**	**Perplexity**
NY Times	[0.25, 0.25, 0.25, 0.25]	11.134214572426878
	[0.4, 0.3, 0.2, 0.1]	11.134214572426878
	[0.1, 0.2, 0.3, 0.4]	10.041644340024261
	[0.01, 0.01, 0.01, 0.97]	9.978199614279989
	[0.01, 0.01, 0.97, 0.01]	10.847679707364945
	[0.01, 0.97, 0.01, 0.01]	14.314343311516835
	[0.97, 0.01, 0.01, 0.01]	24.33160439369806
Sonnets	[0.25, 0.25, 0.25, 0.25]	8.364833744177451
	[0.4, 0.3, 0.2, 0.1]	8.364833744177451
	[0.1, 0.2, 0.3, 0.4]	7.260746113420862
	[0.01, 0.01, 0.01, 0.97]	6.388681236227017
	[0.01, 0.01, 0.97, 0.01]	7.836817387026069
	[0.01, 0.97, 0.01, 0.01]	11.510190202585733
	[0.97, 0.01, 0.01, 0.01]	21.10094394472628

Table 3: Perplexities of different text types to Shakespeare's Plays when *c = 3, k = 2* and λ vary

It's also worth noting that across all three tables, we always have that the perplexities of the texts for the Sonnets are always less than the NY Times articles. We see that varying *c, k, and λ's* change the resulting perplexity, but are proportionally increasing.

III. Language identification using n-grams

For the text classification task, I tried both the uninterpolated N-grams model and the interpolated N-grams model, both with varying values for *c* and *k* (and distribution of *λ* weights for the interpolated model) which can be shown in the tables below. In general, the way to configure the model for discovering the best-performing interpolated model was first finding the accuracy of the uninterpolated models on the validation set and using this information to tweak the variables.

Uninterpolated		
c	**k**	**Accuracy**
0	0	0.49333333333333335
0	1	0.49333333333333335
0	2	0.49333333333333335
0	3	0.4922222222222222
0	8	0.4911111111111111
1	0	0.052222222222222225
1	1	0.6333333333333333
1	2	0.6366666666666667
1	3	0.6377777777777778
1	**8**	**0.6488888888888888**
2	0	0.008888888888888889
2	1	0.6311111111111111
2	2	0.6344444444444445
2	3	0.6311111111111111
2	8	0.6088888888888889
3	0	0.0011111111111111111
3	1	0.5844444444444444
3	2	0.5566666666666666
3	3	0.5188888888888888
3	8	0.40444444444444444
4	**0**	**0.0**
4	1	0.4177777777777778
4	2	0.3522222222222222
4	3	0.3188888888888889
4	4	0.29333333333333333

4	8	0.21666666666666667
7	**0**	**0.0**
7	1	0.14444444444444443
7	2	0.13444444444444445
7	7	0.11777777777777777
7	8	0.11555555555555555

Table 4: Accuracy results for Uninterpolated models

We see that the best accuracy for Uninterpolated models happens when *(c,k) = (1,1), (1,2), (1,3), (1,8), (2,1), (2,2), (2,3), (2,8)* with accuracy over **60%**. The worst accuracy happens when *(c,k) = (1, 0), (2, 0), (3,0), (4,0), (7,0), (7,1), (7,7), (7,8)* with accuracy **less than 20%.** When *c = 0,* and *k* varies, the model only nearly predicts half of the result right (**less than 50%**)

The results show that higher values of *k* along with a zero-value of *c* are harmful to the performance of the uninterpolated model. This might be because a zero-value for *c* lacks information regarding the different forms of city-names in different countries, and higher values of *k* are over-fitted to the training set. Additionally, without smoothing, the models severely underperform, with accuracy of **roughly 0%**. This is to be expected since without add-k smoothing, the models likely struggle under new characters or contexts.

So the best accuracy is **nearly 65%** when *c = 1* and *k = 8.*

For configuring the interpolated models, I chose the pair *(c,k)* that produced the best accuracy from uninterpolated models and added a few pairs when *c = 1, 2, or 3* reflecting the relative performance of these uninterpolated models. In all of the models I let $\lambda_0 = 0$ since the uninterpolated models with *k = 0* generally performed very poorly, so I decided to give it 0 for the first weight in the interpolated models. Interestingly, I saw the accuracy increasing until hitting the peak of **68.33%** when *c = 1, k = 3* and then decreasing.

Interpolated			
c	**k**	**λ**	**Accuracy**
1	1	[0, 0.4, 0.3, 0.3]	0.6388888888888888
1	2	[0, 0, 0.5, 0.5]	0.6322222222222222
1	3	[0, 0, 0.6, 0.4]	0.6311111111111111

1	8	[0, 0.3, 0.4, 0.3]	0.6333333333333333
2	1	[0, 0.4, 0.3, 0.3]	0.6633333333333333
2	2	[0, 0, 0.5, 0.5]	0.6633333333333333
2	3	[0, 0, 0.6, 0.4]	0.6633333333333333
2	8	[0, 0.3, 0.4, 0.3]	0.6422222222222222
3	**1**	[0.25, 0.25, 0.25, 0.25]	**0.6833333333333333**
3	2	[0.25, 0.25, 0.25, 0.25]	0.6711111111111111
3	3	[0.25, 0.25, 0.25, 0.25]	0.6644444444444444
3	8	[0.25, 0.25, 0.25, 0.25]	0.6311111111111111

Table 5: Accuracy results for Interpolated models

So the best accuracy for Interpolated models is **roughly 68%** when $c = 3$ and $k = 1$.

I compiled randomly some of the misclassified cities from the best performing models under the validation set. Here are the results for both Uninterpolated and Interpolated:

	City Name	**True Label**	**Predicted Label**
Uninterpolated ***(c = 1, k = 8)***	laferronay	af	fr
	souk tafetecht	af	fi
	dagsmark	fi	za
	ittefaqabad	pk	ir
	novezan	fr	za
Interpolated ***(c = 3, k = 1)***	sukka	in	fi
	puotinharju	fi	in
	bar bamakhel	pk	af
	iremela	ir	fr
	kranzegg	de	za

Table 6: Samples of misclassified cities for both Uninterpolated and Interpolated models.

There doesn’t seem to be a pattern to the flaws of the models. However, I found out that the most mislabeled country is **Iran** for both models. Classification task is indeed more complex than I first believed it to be but I’m satisfied with the overall performance of the models.