CIS530 HW3 Submission Template

1.5 Writing Shakespeare (6 points)

Shakespeare Texts

Below are the Shakespeare-like texts I generated using different n-gram models:

n = 2

Firsureactim! shat ought con,
Fou sto
A may law he ar fandlet hous be ock
Menour hery to low,
SPER:
Is wif your it pertul kingenzour wit a
Whenambity hemajectice, yours hand in's manne in fronearld yo

n = 3

Firs, what breasur-graces; not him?

LORIZEL:
O, if the slike and so passist:

BERO:
Nay,
Our be an is your his Ermet upon thout the rule breath willain's pray, theed a gread, I'll with dove, losincli

n = 4

First, we her please,
He child?
Your of my fathe, of him but as would half Sicils,
And ther:
Take you.
How! is
ther's possibility.
Would slended to't; for ther's parts,
Live the aboard; his coy abunda

n = 7

First Citizen:
This is the sad and we been impostor that sets up
his head, and Warwick knew it all of good?

OCTAVIUS CAESAR:
I must not his
name out such a lose and sudden command,
If crooked sadly t

…..(more if you like)

Discussion

Observations of the text prediction include increased complexity with growing n-gram size, fixed language patterns emerging, and syllable counts similar to the source being observed. We can clearly see that with an increased number of n, we can become more accurate in our predictions. The sentences aren't really comprehensible at n=2. At n=7, coherent conversations are emerging. You can clearly start to see sentences emerging from the ngram prediction.

2.2 Perplexity (6 points)

Procedure

1. I updated an N-gram model with Shakespeare's plays.
2. I calculated the perplexity for a different text: nytimes_article.txt.
3. I calculated the perplexity for a similar text: shakespeare_sonnets.txt
4. Below are my results and analysis.

Results

Text Type	**n**	**k**	**Perplexity**
nytimes_article.txt	2	1	11.5209
nytimes_article.txt	3	1	10.2216
nytimes_article.txt	4	1	11.2783
shakespeare_sonnets.txt	2	1	7.9101
shakespeare_sonnets.txt	3	1	6.1222

Text Type	n	k	Perplexity
shakespeare_sonnets.txt	4	1	6.485

Discussion

These results are as expected - we can observe the perplexity dropping as the ngram model length increases. This is due to the decreased range of possibilities after greater ngram strings occurring, and the model more often selecting the correct next letter. The model is making predictions and it is finding that the prediction is accurate within similar writing. Given a length 4 ngram, it has a better chance at predicting the next letter of shakespeare_sonnets.txt writing versus by using random char selection. It's easy to grasp that the available random letters after the ngram are most similar in writing to that of Shakespeare.

2.3 Smoothing and Interpolation (6 points)

Procedure

1. I updated an N-gram model with shakespeare_inputs.txt.
2. I calculated perplexities for nytimes_article.txt with different values of k.
3. I calculated perplexities for shakespeare_sonnets.txt with different combinations of lambda.
4. Below are my results and analysis.

**** When running your experiments, only change one variable at a time****
**** Also, make sure you train on the same text corpus for all experiments ****

Results for Different k Values

Text Type	n	k	Uninterpolated Perplexity	Interpolated Perplexity
nytimes_article.txt	2	0.1	12.0252	13.8714
nytimes_article.txt	2	0.5	11.6343	13.7197
nytimes_article.txt	2	1.0	11.3668	13.4647
nytimes_article.txt	2	2.0	11.3071	13.4397
nytimes_article.txt	2	3.0	11.3059	13.4391
nytimes_article.txt	2	3.2	11.3085	13.4413
nytimes_article.txt	2	3.4	11.3118	13.4432

Results for Different Lambda Combinations

Text Type	λ	Perplexity (Interpolated)
nytimes_article.txt	[0.25, 0.25, 0.25, 0.25]	11.583
nytimes_article.txt	[0.4, 0.3, 0.2, 0.1]	13.383
nytimes_article.txt	[0.1, 0.2, 0.3, 0.4]	10.493

Discussion

Based on the data gathered, and assuming the calculations were made properly, we can make the argument that smoothing only marginally increases the precision accuracy of a model. The larger the smoothing factor, the more accurate the model behaved. The smoothing factor actually flattens the statistical model, adding k redistributes some probability mass to unseen data. With limited data, we must smooth the data to account for events that haven't been seen, and avoid cases where the model perplexity can catastrophically fail and report infinity.

3.1 Own Model's Performance (5 points)

- **Validation Accuracy**: .6955

3.2 Model Description and Analysis (15 points)

Final Model Parameters

My best model uses the following parameters:

- **n**: 3
- **k**: .6
- **lambda**: [¼ ,¼ ,¼ ,¼]

This combination achieved the best validation accuracy as reported in 3.1.

Experiment Process and Final Decision

My model used an n-gram model with interpolation design. The model found the highest level of accuracy at ranges of n = 3, k = .6, with lambda being defined as the set of ⅓ at validation of .6955. We tested both NgramModel and NgramModelWithInterpolation to discover the best performance. The testing process was conducted using excel, and gathering each output from the chronological data. After gathering the data, we looked for patterns in performance to determine which n,k values.

Error Analysis

Below are examples of cities my model misclassified:

City Name	True Label	Predicted Label
l'houmeau	fr	af
grandecourt	fr	cn
xueguangzhang	cn	pk
uperro	in	pk
esnouveaux	fr	pk

Error Discussion

My model seemed to misclassify French cities at the highest rate. Perhaps it was the training data that didn't include a large volume of cities that had French names. Prior to the model being built, I would have speculated that Chinese cities might have been the easiest to identify. They are the only language when translated into English that uses xi in succession on a frequent basis, they also use zi quite frequently. In practice, my model was in fact most often surprised by Chinese cities. The highest perplexities were reported in mostly Chinese cities. I would have to look very closely at the training data to better report the error margin.