CSE 143 HW2

Maximum Likelihood Estimate

Arnav Nepal, Harsh Jha

May 22, 2024

1 N-gram models

All of our N-gram models followed a similar underlying framework. We first took a tokenized list of each sentence, created N-gram out of them to use as dictionary keys, and tallied up the occurrences of keys for the calculation of probability later. The actual calculation for the Maximum Likelihood probability estimates was done utilizing the mathematical formulas outlined in class: $\frac{c(w_i)}{N}$ for Unigram, $\frac{c(w_{i-1},w_i)}{c(w_{i-1})}$ for Bigram, and $\frac{c(w_{i-2},w_{i-1},w_i)}{c(w_{i-2},w_{i-1})}$ for Trigram. The perplexity calculations also followed the formulas outlined in class, the result of which can be seen below:

Model	Train set	Dev set	Test set
Unigram	530.9	472.3	481.6
Bigram	55.1	19.3	19.5
Trigram	5.9	2.36	2.37

Based on the dataset above, we can safely conclude that a higher N generally produces a better outcome (lower perplexity indicates a model that is more accurate). However, it should be noted that across the board, we got a *lower* perplexity score across the board for both the dev and test sets compared to the train set.

2 Additive Smoothing

For this part of the program, we duplicated our code from our unsmoothed implementation and extended it with the additive smoothing features. The implementation of additive smoothing can be found in `utils_additive.py` in our codebase.

2.1 Additive smoothing: $\alpha = 1$

We gathered the following perplexity scores using our implementation of additive smoothing for $\alpha = 1$:

Model	Train set	Dev set	Test set
Unigram	530.6	472.6	481.8
Bigram	10557	753.4	767.7
Trigram	75208	54.1	55.3

2.2 Additive smoothing: $\alpha = 2$ & $\alpha = .5$

We gathered the following perplexity scores using our implementation of additive smoothing for $\alpha = 2$

Model	Train set	Dev set	Test set
Unigram	531	473.4	482.6
Bigram	17163	1144	1167
Trigram	107378	65.7	67.2

The following table shows the results of using $\alpha = .5$

Model	Train set	Dev set	Test set
Unigram	530.6	472.3	481.6
Bigram	6254.6	488.9	497.5
Trigram	48476	43.7	44.6

The extremely high perplexity scores, especially for the Trigram and Bigram models using $\alpha = 2, .5$ indicate that these models likely require significantly lower alpha values to get a better model.

2.3 Best α values

Through experimentally adjusting α values for each of out N-gram models, we've determined that the best values for additive smoothing and their associated perplexity scores (D) are as follows:

- Unigram: $\alpha = .1$, $D_{train} = 530.86, D_{dev} = 472.28, D_{test} = 481.6$
- Bigram: $\alpha = .000001$, $D_{train} = 55.47, D_{dev} = 19.4, D_{test} = 19.57$
- Trigram: $\alpha = .0000001$, $D_{train} = 6.06, D_{dev} = 2.37, D_{test} = 2.38$

From our results, we see that additive smoothing seems to get better the closer to zero that a number is (we chose these values as we found diminishing returns in improvement for smaller and smaller values for each model). However, we also observed that additive smoothing does not seem to actually improve perplexity scores, as the resultant scores from N-gram models with smoothing did not actually see a significant improvement from the base model.

3 Linear Interpolation

To implement linear interpolation in our model, we followed the mathematical formula and multiplied each N-gram model's probability by the associated hyperparameter. Linear Interpolation is implemented directly in `main.py`, and can be found by searching for the if-statements with 'interpolation' as the boolean condition.

3.1 Case by Case Perplexity

Hyper-parameters ($\lambda_1, \lambda_2, \lambda_3$)	Train set	Dev set	Test set
(.1, .3, .6)	8.2	2.58	2.59
(.2, .2, .6)	8.4	2.64	2.65
(.3, .3, .4)	10.94	2.82	2.83
(.5, .3, .2)	17.2	3.19	3.2
(.1, .1, .8)	7.0	2.52	2.53
(.3, .4, .3)	12.69	2.89	2.9

3.2 Hypothetical: Using half the training data

Using half the training data should not significantly affect perplexity scores, especially when using linear interpolation for smoothing. This might be because the number of tokens derived from our training set is already sufficiently large, so a reduction by half may not affect it. This is in fact what we found was the case when we tested this hypothesis, as running linear interpolation with only half the training data only increased perplexity scores by about +0.1 across all data sets. We theorize this slight increase is probably explained by the reduction in data leading to more uncertain predictions (and thus higher perplexity). If more data was removed, the increase in perplexity might be more apparent.

3.3 Hypothetical: Increasing <UNK> tokens

We think increasing the number of <UNK> tokens by increasing the count limit for <UNK> tokens will decrease perplexity scores in the context of this assignment. This might be because rarer words make the model more complicated and might contribute to over fitting. By reducing the vocab size, we make the model more accurate by focusing it on the more frequent words in a language. This effect likely has diminishing returns, as increasing the limit too high could cause the model to start losing accuracy in excess of the benefit gained from focusing on more frequent words. This hypothesis matched our findings, as we found that increasing the <UNK> limit to 5 led to an across the board reduction of about 5% for perplexity scores.