N-Gram Model Analysis

Lucy Manalang, Alec Chen

October 6, 2024

1 Overview:

In this project, we created 4 n-gram models and calculated their perplexities For each Language Model, we kept these questions in mind while writing our methods:

1. **How do different implementations of smoothing (or lack of smoothing) affect *training set* perplexity?**
2. **How do different implementations of smoothing (or lack of smoothing) affect *validation set* perplexity?**
3. **How does your choice of n affect the perplexity of your model? Does adding smoothing affect your answer?**

2 Introduction:

How can we predict the next word someone says computationally? Our human minds are fascinating since we subconsciously process textual clues leading up to a certain word in a sentence. In classical natural language processing, we decide that the certain context of words that lead up a specific word determines what that word is. That's what n-gram models do. In this paper, we investigate the effects of different algorithms in n-gram models on the n-gram model performance.

3 Maximum Likelihood Estimation:

In maximum likelihood estimation, we estimate the probability of the next word in a sentence based on the n words that come before it. Specifically, we take the counts of the number of times the word appears after a given context and divide it by the number of times any word appears after a given context. To evaluate these models, we used a metric called perplexity, which is the average negative log-probability that specific n-gram. The smaller the perplexity the better.

1. When running our MLE model using *brown.train.txt* as both our training and validation with an n of 3, we get a perplexity score of 5.712. Interestingly, running the same model using *unk* tokens, the perplexity goes up to 6.225. This is likely because when validating the training data against itself, *unk* tokens dilute the probabilities of previously high-probability n-grams. This is

only an issue when perplexity is calculated by validating the training data against itself.

2. When running our MLE model using *brown.train.txt* as our training data and *brown.valid.txt* as our validation data, we expectantly get a perplexity of infinity regardless of our n. This is due to a lack of smoothing of the data, which leads to (sometimes many) 0-probability n-grams. The use of *unk* tokens does not help with this, as they target 0-probability words, not n-grams.

3. Lower n values do not affect perplexity with MLE, as it is infinity every time regardless of the n value. Using *unk* tokens does not change the outcome, as they are already typically infinite.

4 Laplace Smoothing:

One of the problems we have in n-gram models is when the count of unseen n-grams is 0. This breaks our probability calculations sending our perplexity to infinity. To resolve this problem, we used two methods: unknown tokens and Laplace smoothing. We ran all of our models with unknown tokens which, replace unseen tokens with unknown tokens (of which we generate from low probability tokens in our training set).

The second method we used is called Laplace smoothing. We add 1 the count of all counts of every word in our vocabulary which removes the count of 0 for unseen tokens in our model and assigns a small probability to them. This method works but isn't as good as some of the other algorithms we have.

1. When running the Laplace model using *brown.train.txt* as both our training and validation with an n of 3, we get a perplexity score of 113,900. While this is not ideal, it is expected that Laplace smoothing would not have the best perplexity. This is improved slightly by *unk* tokens that get a perplexity of 107,300

2. When running our Laplace model using *brown.train.txt* as our training data and *brown.valid.txt* as our validation data, we end up with a perplexity of 181,600 without *unk* tokens and 159,900 with *unk* tokens. Similar to the previous part, this is not ideal, but expected.

3. With an n of 5, our Laplace model performs worse with a perplexity of 272,200 without *unk* tokens and 260,800 with *unk* tokens.

5 Interpolation Smoothing:

Another way to address our data sparsity problem, having the count of an unseen n-gram being 0 or close to 0, is to combine the probability distributions of all of our n-gram models (adjusting the weights of each probability such that the total probability sums to one). This means that for a trigram model, we would multiply the probability of our trigram model, bigram model, and unigram model by certain coefficients, and then add those models together. **All perplexity scores have an *n* of 3 and needed *unk* tokens.**

1. The implementation of interpolation smoothing when validating the training set against itself gave us a perplexity of 2,126 with coefficients of 0.2, 0.2, and 0.6.

2. When using interpolation for the validation set, perplexity surprisingly decreased to 2,031 with coefficients of 0.2, 0.2, and 0.6. We believe this is a very good perplexity for interpolation.

3. Increasing n in our interpolation model increases the perplexity to 4,449 with an n of 5 and coefficients of 0.05, 0.1, 0.2, 0.3, 0.35.

6 Backoff Smoothing:

In n-gram models, we oftentimes use higher order n-grams in order to achieve more contextual accuracy. However, the larger the n-gram, the higher the data sparsity since the n-gram is capturing longer sequences of words. To compensate for this data sparsity, we can use a strategy called simple backoff. When we cannot find a unseen higher order trigram from our train set, we can backoff to a lower order n-gram model that has has more diversity in data to assign a non-zero probability to that unseen higher order trigram. **We used a discount of 0.5, *unk* tokens, and an n of 3 and discount factor of 0.5 for questions 1 and 2.**

1. When running the Backoff model using *brown.train.txt* as both our training and validation with an we got a perplexity score of 6.225. A perplexity this low is expected when we validate on the train set.

2. Similarly to Interpolation, our Backoff score decreased to a perplexity of 34.88 when we used the *brown.valid.txt* data. It is clear that the unseen tokens cause great mayhem in our model but it seems that the simple-backoff procedure softens the impact these unseen tokens have on our model perplexity. The inclusion of unknown tokens also softens the impact of unseen tokens in validation

3. Increasing n in our backoff model decreases our perplexity. This makes sense because we get to achieve greater contextual accuracy while also resolving our data sparsity problem by backing off to lower order ngram models. Increasing our n to 5 with a discount factor of 0.5 decreased the perplexity to 1.597.

4. Increasing our discount factor while having an n = 5 on the validation set decreases our perplexity. With discount factor of 0.5, we have perplexity of 2.273 and with a discount factor of 0.7 we have a perplexity of 1.74. This is odd given that we are using the less probability mass in the lower order n-gram distribution to calculate our perplexity.

7 Testing:

We unit tested our ngram models by determining whether or not the probability distribution of our ngram models summed to 1. We found that our MLE, Laplace Smoothing, and models indeed summed to 1. However, our interpolation algorithm did not sum to one. This may indicate our interpolation algorithm may be incorrect.

8 Qualitative Analysis Results:

For our qualitative analysis, we were asked to create 5 preambles to help generate sentences. We tried to create sentences with commonly used words that **1.** would test the full capabilities of the model (preambles or words that may surprise it) and **2.** could generate unique sentences. The preambles we chose were:

1. Once upon a time
2. <eos>
3. When I grow up I want to be
4. Macalester College is a great
5. Due to differences in

Many of these preambles would be easy for a typical human to finish, however, we wanted to try test how the computer would handle these cases. For example, we wanted to see how the model would handle the beginning of a sentence being an <eos> token.

We did not expect the models to perform particularly well, and were unfortunately proven right. Many of the completed sentences seemed random, which may be explained by the size of the vocabulary diluting the probabilities of the most likely words. While many of the more likely words have higher probabilities, due to smoothing, the least-likely words take precedence and are chosen more often. This was an issue while creating our qualitative analysis, but after debugging and realizing the most probable word and least probable word were the same magnitude (10^{-5}), we realized we could not change the outcome in the time we had.

However, we did notice the models completing sentences using words that have relation to each other. This was especially prevalent with our interpolation model. For example, the words trolls and tricksters are placed together. Looking through the completed sentences, it is possible to find other words with a relation to each other. Of course, this could also be a coincidence.

Each model has similar behavior to the other models, however, one common theme throughout this project has been how MLE does not work. The lack of smoothing led to many 0s or infinities, which made it nearly impossible to randomly choose a word without infinity count or 0 count errors.

9 Conclusion:

In conclusion, we discussed and used a variety of different n-gram models and techniques to predict the next word in our sentence. We used perplexity (average negative log-probability) to evaluate each of these models and found that mixed effects on perplexity when we increased or decreased n. We attribute the increase in perplexity to the increase in data sparsity for higher order n-grams, which means the model is more "surpised" when n is increased beyond a certain point. We attribute the decrease in perplexity from greater n to the greater contextual accuracy larger n-gram models have. We also find that of all the models we tested, the simple backoff model turns out to have the least perplexity of all the models. This is odd given our intuition that interpolation would have more accurate modelling of the probability of our data set given the averaging of over all n-gram distributions instead one in the simple backoff process. Further, we did not sum to one on our unit

test for interpolation. Despite certain discrepancies in the trends of our models, we were able to accurately assess and evaluate the differences and similarities between our models.