Peter Schaldenbrand

Homework 2

Dr. Hwa

When an N-gram language model is trained on a large training data set such as the Project Gutenberg, I had some interesting issues. The first was quite obvious that Java gave me errors when it ran out of memory. The smooth models contain multiple N-grams, and each N-gram gets very large with huge amounts of training data. So, when making the N-grams, I occasionally got an error that said there wasn’t enough memory to complete the program. This was solved by using less training data. Another issue was that more data doesn’t always mean better results. When testing the smoothed quadgram, I had tried it with five of the training data files and got a score of 257 out of 1040. When I increased the amount of training data by tripling it, I got a score of 242 out of 1040. Training data is useless if it doesn’t contain information that pertains to the problem. Adding more files to the N-gram worked by misleading the model to choosing the wrong words.

The development data helped most with the smoothed N-gram models. Using the development data, I was able to adjust the lambda values that gave each N-gram in the smoothed model a certain weight. The bigram model in particular didn’t work very well, but the unigram and trigram did decently. So on the smoothed trigram model, using the development data, the lambda coefficients adjusted to give less weight to the bigram model’s perplexity readings and more weight to the other models’.

Dr. Hwa had said in class that we should shoot for around 30% on correct completed sentences, but the best I had gotten was 24.71%. The results of my N-grams are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Score out of 1040 | Overall Avg. % | Dev % | Test % | Time (sec.) |
| Unigram | 242 | 23.27 | 23.65 | 22.88 | 15 |
| Bigram | 191 | 18.37 | 18.27 | 18.46 | 18 |
| Smooth Bigram | 241 | 23.17 | 23.17 | 22.69 | 27 |
| Trigram | 234 | 22.50 | 20.96 | 24.04 | 31 |
| Smooth Trigram | 241 | 23.17 | 23.65 | 22.69 | 96 |
| Reverse Bigram | 202 | 19.42 | 19.81 | 19.04 | 18 |
| Smooth Reverse Bigram | 192 | 18.46 | 17.50 | 19.42 | 23 |
| Reverse Trigram | 233 | 22.40 | 21.73 | 23.08 | 32 |
| Smooth Reverse Trigram | 212 | 20.38 | 21.35 | 19.42 | 122 |
| Quadgram | 217 | 20.87 | 19.04 | 22.69 | 54 |
| Smooth Quadgram | 257 | 24.71 | 26.34 | 23.08 | 129 |

I was disappointed that in my many, many attempts at this challenge I was unable to get an N-gram to perform around 30%. The N-gram model alone does not perform well because it only uses patterns to figure out data. Understanding language is much more than just recognizing a pattern in a sentence, especially with how large languages are. To complete a sentence, there must be knowledge of the context of the missing word. Part of that is just recognizing what part of speech it is, but another part is understanding what is happening in the sentence before and after. Because N-grams do pattern recognition, they rely on tons of training data in order to become accurate, and many computers cannot handle this. If you were to somehow add definitions to words in an N-gram model, this would help. Having an N-gram model where each word accounts for its own synonyms, would work very well especially if there was limited training data. Knowing synonyms could make two sentences that mean the same thing, be represented by the same data in a model.

As part of the bonus, I had tried six extra N-gram models. I was thinking that reading a sentence in reverse might actually provide some usefulness when trying to find patterns. So I implemented a reverse bigram and trigram model. Interestingly enough, the reverse bigram model performed better than the regular bigram model, and the reverse trigram model was as good as the regular trigram model. I then tried to mix forwards and backwards N-grams in a smoothing model. The smooth reverse bigram uses a normal bigram and a reverse bigram. Using these two models together, they ended up performing very poorly. I have a hunch that it is because the forward and reverse patterns work oppositely. When one direction gives a low perplexity the other gives high perplexity and so in the end there is no good ‘winner’ to the sentence completion. The smooth reverse trigram model uses a normal trigram, reverse trigram, and reverse bigram in its implementation. This model, too, did not perform well, and I have a feeling that it was for the same reasons as the reverse bigram.

The other direction I went with the bonus, was with a larger N-gram. I had implemented a 4-gram in the same way that I had done the trigram and bigram. This model performed by itself pretty well but not amazingly. I would account for this lackluster performance by saying there was not enough training data used. Because the 4-gram is so large in data, I wasn’t able to implement it using all the training data without running out of memory. With a large N-gram, it is very difficult to get exactly three words in a row to match three words in a row that you found in your training data, and that is why you must have so much training data. I implemented a smoothed 4-gram using a 4-gram, trigram, and unigram. I chose to use the unigram instead of the bigram because of how well the unigram performed. This configuration of N-grams worked very well and got the highest score on the challenge. This triumph was due to having a wide range of patterns to choose from.