Assignment 1: N-gram Language Modeling

Henry Nguyen
SID: hnguye87

Abstract

The problem in this assignment was to build a unigram, bigram, and trigram language model in Python and implement linear interpolation smoothing between the three classes to improve the language model. I worked on this assignment alone. I found that the unigram model was much worse in perplexity than bigram and trigram, but it had lower perplexity when dealing with unseen words not in the dictionary. Linear interpolation smoothing helped fixed this issue.

Problem

The first part in this assignment was to create a unigram, bigram, and trigram language models. Out-of-vocabulary (OOV) words that appeared less than three times in the training data were to be converted to <UNK>, (unknown), tokens. I began by reading in the training data into a list token-by-token. Using the list, A unigram dictionary was then created to keep count of the number of times each token appears with the token as the keyword and replaced the tokens that appeared less than 3 times with <UNK>. The process is repeated two more times to create a bigram and trigram dictionary with <UNK> token put into them. I then read in the dev and test data line-by-line then entered <UNK> into both. Functions were then made to calculate probabilities of a sentence for all three language models and separate functions for the perplexity of all three models given a data set.

The second part of the assignment was to implement linear interpolation smoothing between all three models to improve the language model. I made the smoothing into one function that utilizes all three dictionaries in order to calculate the probability of each step in the function. To calculate the complexity of the smoothing, I had to count the number of tokens in each file first. After that I created another function to calculate the complexity of the smoothing using very similar code.

PART 1 Programming: n-gram language modeling (50%)

Perplexity Table for n-gram Models

	Unigram Model	**Bigram Model**	**Trigram Model**
Training Set	983.4429002966516	76.53277160033987	6.444134116486955
Development Set	900.1971583364631	∞	∞
Test Set	904.3618500568698	∞	∞

Based on the perplexity results of the training set, the Unigram model is by far the worst model with a perplexity of over 1000. The Bigram Model is a big improvement from the Unigram model. The Trigram Model is also a noticeably big improvement from Bigram model. When working with the Development Set and Test Set however, there is an issue with the Bigram model and Trigram model where the models assign $p(\bar{x}) = 0$ for words/combinations not seen before. Since $log_2 0 = -\infty$, the perplexity equation will come out to ∞ perplexity as there are infinite choices for the next possible word the model has to make for an unknown word. This is a shortcoming of the Bigram and Trigram model that makes them situationally worse than the Unigram. The Unigram model however had a very high, bad perplexity on all sets of data making it still a very bad model.

PART 2 Programming: Smoothing (50%)

#1 Perplexity Table for ($\lambda_1, \lambda_2, \lambda_3$)

	(0.1, 0.3, 0.6)	**(0.1, 0.2, 0.7)**	**(0.2, 0.3, 0.5)**	**(0.05, 0.3, 0.65)**	**(0.3, 0.3, 0.4)**	**(0.5, 0.3, 0.2)**
Training	11.14284303932996	10.138254218618329	12.877860668945356	10.446097638037637	15.29620016932151	25.06690228847442
Dev.	351.40599588398726	395.24285244139816	305.7579691487017	407.6018578193292	286.4849099367519	280.70814761592436

Weighing the unigram and bigram model with the trigram model has increased the perplexity if you were to work with a data set without any unknowns (test set). The opposite result happens if you work with a data containing unknowns (dev set). While the perplexity is still high, weighing the unigram model with the bigram and trigram model has allowed the perplexity to not be infinite. The higher the weight of the unigram the lower the perplexity. There seems to be a tipping point though where the perplexity loss starts to be negligible if the unigram weight is too high. Based off the results, $(\lambda_1 = 0.3, \lambda_2 = 0.3, \lambda_3 = 0.4)$ are the weights I am choosing since higher weight on the unigram model seems to significantly improve the complexity on the dev data contain unknown words. Unknown words are much likely to appear in this model since our dictionary is not very large, so the results from the dev data seems to be much more significant.

#2 Using $(\lambda_1 = 0.3, \lambda_2 = 0.3, \lambda_3 = 0.4)$, the complexity from the test data was

286.0284808631883

The results are very close to the results from the dev data set which also contained many unknown words.

#3 Using half of the training data will increase the perplexity on unseen data. With less words learned during training, all n-gram language models will encounter more OOV, unseen words which will cause their perplexity to go up. Linear interpolation smoothing uses all three of these models so it will also undoubtedly rise. When these models encounter an unseen word, there are infinite choices for the next possible word which raises the perplexity.

#4 Perplexity Table for n-gram Models with tokens less than 5 converted to <UNK>

	Unigram Model	**Bigram Model**	**Trigram Model**	**Smoothing**
Training Set	822.1036537392064	73.92034062090495	7.112411206164493	16.375453295016516
Development Set	774.0351373069249	∞	∞	239.3143448403995

The evidence suggests that the complexity decreases for the unigram model if the tokens that appear less than 5 times are converted to <UNK>.Compared to before when only a few words that appeared less than three times would be converted to <UNK>, the dictionary now contains many more counts of <UNK>. The chances of encountering an <UNK> is also higher. These two facts will make the probability of token, <UNK> very higher in a unigram model. It did seem to improve the bigram model a little as the count of (‘<UNK>’, ‘<UNK>’) increased. It negatively affected the trigram model as the perplexity slightly went up. For smoothing, the perplexity increased a little for the training set as the trigram model has a larger effect on an overfit model. For the development set however, the unigram plays a larger role and the improvement in complexity is large.

Conclusion

This assignment shows the shortcomings of the bigram and trigram model when dealing with unseen words, causing the complexity to be *infinite*. No model is strictly better than the other in this sense. Linear interpolation smoothing fixes this issue of the bigram and trigram and takes advantage of all three models. Weighing all three models, smoothing allows for unseen words to have a probability higher than 0. Giving some weight to the unigram model (in my case $\lambda_1 = 0.3$) allows for the model to give a higher probability to unseen words improving its complexity. The somewhat high weight for the unigram model was because the training data provided for this assignment was not very much. In reality the training would be much more extensive and ideally there would be much fewer <UNKS>. In that case a higher weight for the trigram would be needed. While the data shows the smoothing model would perform worse than the trigram in the overfit model due to the weights, it is unrealistic to assume that no unseen words will show up in reality.