Ryan McCrory
Professor Flanegan
Computer Security
January 20, 2020

Assignment 1

Part 1:

1.1 To begin this project, I first wrote a program to iterate through the training set and add all words as keys to a unigram dictionary with the value being the number of occurrences in the file. I also adjusted all words that occur less than three times to the token '<UNK>' and added '<STOP>' symbols to the end of each sentence. This produced 26,602 unique tokens, as expected. Then I created and filled bigram and trigram dictionaries.

Then I had a difficult time calculating perplexities. It took me several days to get the function right, which was because I struggled to understand exactly what the equation was doing. The main thing I was doing wrong was calculating the probability of each key in the dictionary, rather than each word in the file. The perplexities for each model are calculated in nearly the same way, using the following equation:

$$\text{perplexity} = 2^{(\frac{-1}{M}) \sum_{i=1}^{m} log_2 p(x_i)}$$

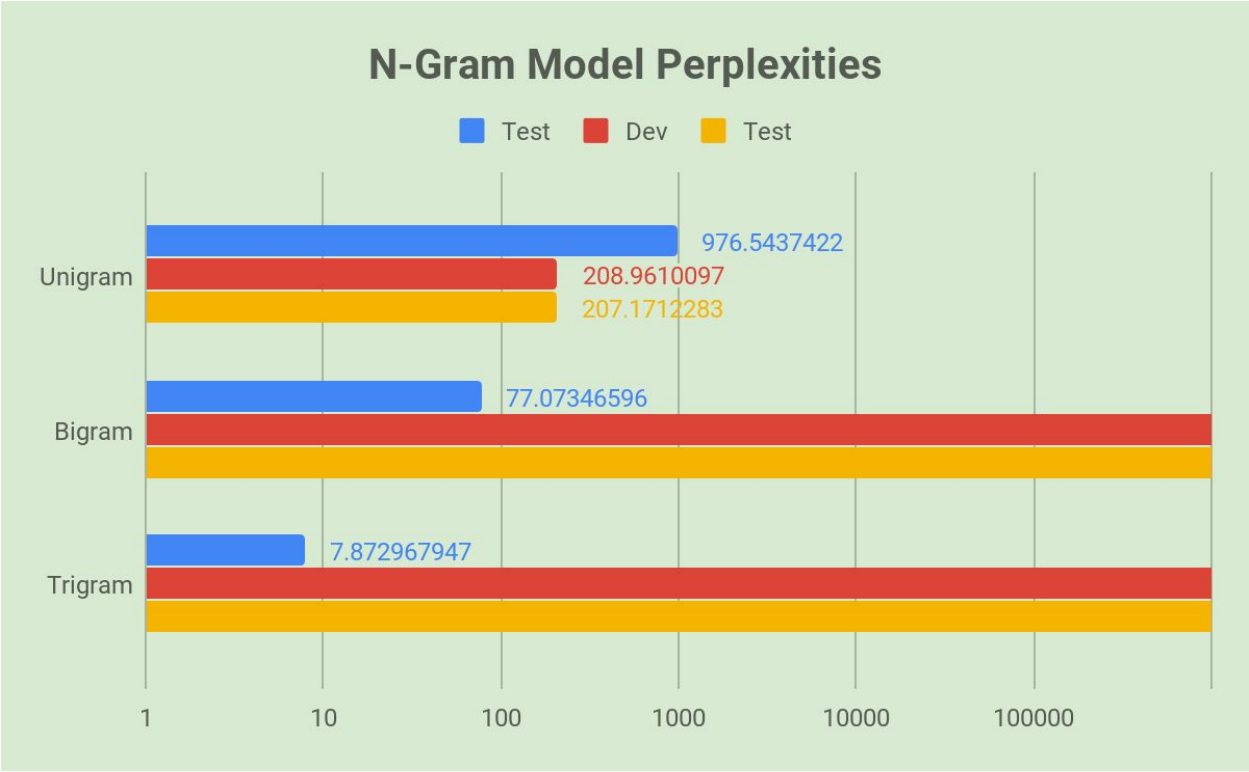
Here, M is the total number of tokens in the file, m is the number of sentences in the file, and p(x_i) is the probability of $sentence_i$. The probability of the sentence is the sum of the probability of its N-grams. The formulas to find the probabilities for a unigram, bigram, and trigram are shown below.

Unigram	$p(x_i) = \frac{number\ of\ x_i\ occurrences}{total\ tokens\ in\ the\ file}$
Bigram	$p(x_i)\|(x_{i-1}) = \frac{number\ of\ (x_{i-1}, x_i)\ occurrences}{number\ of\ (x_{i-1})\ occurrences}$
Trigram	$p(x_i)\|(x_{i-2},\ x_{i-1}) = \frac{number\ of\ (x_{i-2}, x_{i-1}, x_i)\ occurrences}{number\ of\ (x_{i-2}, x_{i-1})\ occurrences}$

The resulting perplexities from each model on each file are illustrated below. I calculated each using a separate perplexity function for unigrams, bigrams, and trigrams with the differences shown in the chart above. Note: Bigrams and Trigrams in the dev and test sets that are not found in the dictionaries have a probability of 0. To avoid an error when doing $log_2(0) = -\infty$, I set it equal to -1,000. This has essentially the same effect.

Perplexities

	Train	**Dev**	**Test**
Unigram	976.5437422200696	208.96100969173375	207.17122827010385
Bigram	77.07346595628817	2.8509753150915065e+82	1.19793585132983e+82
Trigram	7.872967947053928	1.167504470231214e+200	3.625677854497164e+199



As seen above, the bigram and trigram models have a nearly infinite perplexity for the development and training sets. This is the result of the bigrams and trigrams with a probability of 0. In the perplexity equation, we take the $log_2(p(x))$. When p(x) = 0, we get $log_2(0) = -\infty$, which ultimately pushes perplexity toward positive infinity. However, we can see that for the training set, the trigram perplexity is much better than the bigram, which is much better than the unigram. This is what we would expect given that the trigram model has the most context of the three.

Part 2:

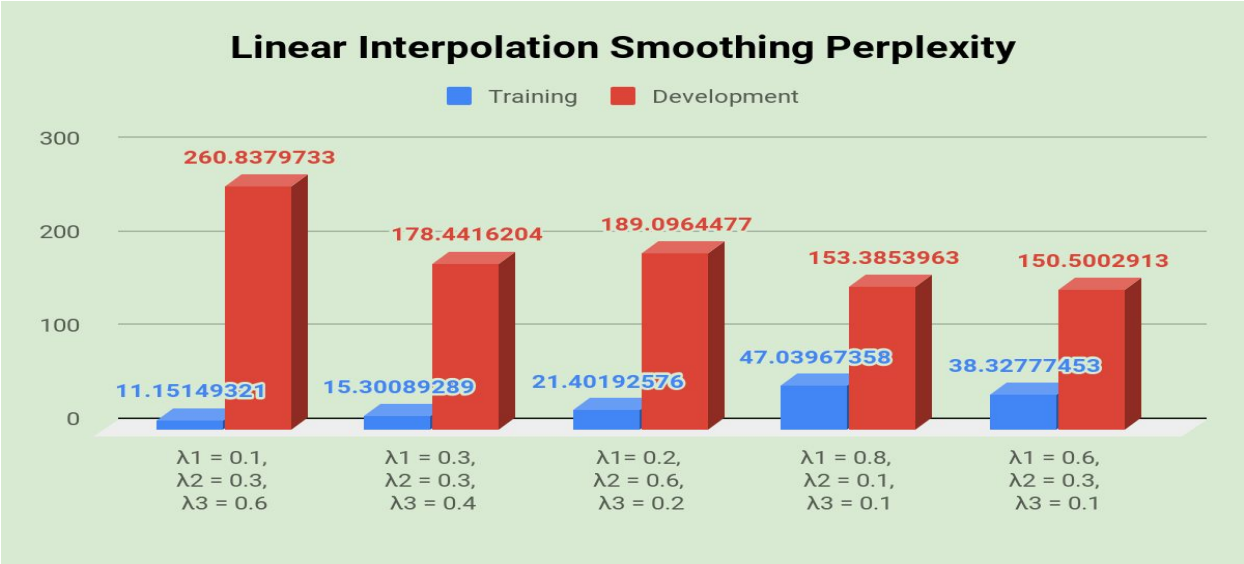
2.1 To improve my language model, I implemented linear interpolation smoothing. This adjusts so that there are no probabilities of a word that equal 0. This is done by..

$$\Theta' x_i|x_{i-2},\ x_{i-1} = \lambda 1 \Theta x_i + \lambda 2 \Theta x_i|x_{i-1} + \lambda 3 \Theta x_i|x_{i-2},\ x_{i-1}$$

Here, Θ' represents the smoothed parameters, and the hyperparameters λ1, λ2, λ3 are weights on the unigram, bigram, and trigram language models, respectively, where λ1 + λ2 + λ3 = 1. I implemented this into my program by creating a smooth() function that takes in several arguments. These are λ1, λ2, and λ3, the unigram, bigram and trigram dictionaries, the file name, and the total number of words in the file. This function iterates through the specified file, line by line, word by word, and calculates the unigram, bigram, and trigram probabilities of each word in the line. This is done the same way as in part one. When the probabilities are calculated, they are multiplied by the corresponding hyperparameter, λ_i. The sum of these products is taken, and the perplexity formula is the same from there. Below are the smoothed perplexity scores on the train and dev sets for 5 different sets of λ's, including λ1 = 0.1, λ2 = 0.3, λ3 = 0.6.

Perplexity

	λ1 = 0.1, λ2 = 0.3, λ3 = 0.6	**λ1 = 0.3, λ2 = 0.3, λ3 = 0.4**	**λ1= 0.2, λ2 = 0.6, λ3 = 0.2**	**λ1 = 0.8, λ2 = 0.1, λ3 = 0.1**	**λ1 = 0.6, λ2 = 0.3, λ3 = 0.1**
Training	11.151493208689 14	15.300892892 48603	21.401925763 801437	47.039673584 90623	38.327774534 63394
Dev	260.83797331 87371	178.44162037 38623	189.09644769 568447	153.38539633 709487	150.50029131 383823



2.2 Putting it all together, the perplexity on the test set using the best hyperparameters from the dev set (λ1 = 0.6, λ2 = 0.3, λ3 = 0.1) is 149.8419704516406. This is obviously far better than all of the original perplexities on the test set without smoothing, which were: unigram = 207.17, bigram = ∞, trigram = ∞. This proves that linear interpolation smoothing does in fact improve my language models.

2.3 Below is a chart showing the smoothed perplexities on the unseen data sets using the same hyperparameters as above (λ1 = 0.6, λ2 = 0.3, and λ3 = 0.1). The chart compares the results of using the full training data vs half the training data. I tested this by breaking the loops that fill the dictionaries after iterating up to the 30,765th line of the training set, which is half way through it.

	Full Training Data	**Half Training Data**
Dev	150.50029131383823	142.587174768113
Test	149.8419704516406	141.87378798350034

As shown above, using half the training data results in a slightly lower perplexity for both unseen data sets. This is because it results in a smaller vocabulary, making it more likely to the correct word as there are fewer words to choose from.

2.4 Below is a chart showing the smoothed perplexities on the unseen data sets using the hyperparameters λ1 = 0.6, λ2 = 0.3, and λ3 = 0.1. The chart compares the results of setting tokens that occur under 3 times to UNK, which is what I have done up until this point, and the results of setting tokens that occur under 5 times to UNK.

	Toks < 3 = UNK	**Toks < 5 = UNK**
Dev	150.50029131383823	128.36544316164728
Test	149.8419704516406	127.71806482636886

As shown above, setting tokens that occur under 5 times to UNK decreases and improves perplexity. This is because it shrinks the vocabulary, making it more likely to the correct word as there are fewer words to choose from.