Homework 2 Report **Caleb Nelson**
CS272: Statistical NLP caleban@uci.edu
Prof. Sameer Singh Due Date: 2/7/19

1 Part 1: Implementing a Language Model

I decided to implement a modular n-gram model, where n was designed to be a tunable hyperparameter. I opted for this approach instead of hardcoding a trigram or any other n-gram model, because I wanted to test different lengths of n-grams with different techniques easily. In order to implement this, I changed the model from a simple dictionary with word keys and probability values to a list of dictionaries with tuple keys, with each dictionary responsible for storing a different length of gram, represented in the data by a tuple. When I processed a sentence, I would keep a running history of the previous words seen and update all of the apporpriate dictionaries with each new word. This implementation led to a natural supporting of "start of sentence", as when the algorithm reads the first word it will store it in the unigram dictionary, when it reads the second word it will store it in the unigram dictionary and store the first two words in the bigram dictionary, and so on until n words are read, after which only the most n recent words are stored in this fashion. While this did lead to an increase in memory usage and processing time, the processing time was not too terribly impacted by this additional storage.

For the normalization step, I started at the largest n-gram dictionary stored in the model and calculated the log of the number of occurrences of each n-gram divided by the number of occurences of each n-1 gram with the last word excluded. This division led to the probability that the sequence of n-1 words would be followed by the next word in the n-gram, and so it was analogous to the normalization step as defined in the unigram model of dividing the number of occurrences of each word by the total size of the corpus. Starting from the dictionary using the largest grams as keys, I was able to implement this normalization in every dictionary except the unigram dictionary, where I simply used the normalization used in the unigram model.

Due to the implementation of my model, I was also able to add back-off smoothing in order to decrease the perplexities of my model. In order to do so, when tasked with calculating the conditional probability of a word in a sentence, I would assemble the largest n-gram I could given the parameters of my model and the previous words of the sentence. I would then search the corresponding dictionary for the corresponding probability to the gram I had constructed, and if I was unable to find it, I would remove words from the history until a probability was found or I was left with only the original word. If I was still unable to find the original word in the unigram dictionary, I would return the probability of an unknown word as in the unigram model.

After implementing this model, I added one more optimization - I preprocessed the data to cast all words as lowercase so different capitalizations would not be considered different words. This led more accurate frequencies of many words in the dictionary and noticably improved my results, as the following perplexity tables show.

2 Part 2: Perplexity Tables

In each table, the row represents the corpus the model was trained on and the column represents the corpus the model was evaluated on.

2.1 Unigram baseline

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	1737.54	15038.1	2311.56	brown	1758.25	15344.3	2308.54
reuters	6534.23	1580.91	10578.8	reuters	6616.48	1576.85	10561.7
gutenberg	3778.55	37095.4	1060.54	gutenberg	3819.4	37900.9	1035.78

2.2 Bigram model without lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	569.898	7678.17	1098.54	brown	583.483	7822.17	1107.38
reuters	3162.27	192.695	6325.75	reuters	3234.22	189.429	6398.29
gutenberg	1849.08	25074.2	302.597	gutenberg	1869.32	25609.5	293.901

2.3 Bigram model with lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	492.291	4044.85	854.857	brown	501.187	4093.97	860.016
reuters	2444.5	171.694	4579.57	reuters	2444.5	171.694	4579.57
gutenberg	1508.28	15257.8	266.639	gutenberg	1508.28	15257.8	266.639

2.4 Trigram model without lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	461.035	7071.43	977.172	brown	473.41	7217.83	983.393
reuters	2897.06	109.341	6011.83	reuters	2957.5	107.809	6093.48
gutenberg	1642.57	24041.9	204.721	gutenberg	1661.72	24574.9	196.911

2.5 Trigram model with lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	394.7	3710.43	754.563	brown	403.077	3761.06	757.396
reuters	2222.83	94.4735	4330.27	reuters	2254.64	93.851	4382
gutenberg	1333.7	14590.9	178.721	gutenberg	1337.52	14755.6	172.236

2.6 Quadgram model without lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	446.541	7015.82	964.04	brown	458.846	7163.47	970.797
reuters	2874.87	91.8294	5992.14	reuters	2933.66	91.1811	6074.81
gutenberg	1615.14	23952.8	182.865	gutenberg	1634.76	24487.3	174.685

2.7 Quadgram model with lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	381.315	3677.8	743.146	brown	389.705	3728.13	746.637
reuters	2203.82	78.176	4314.1	reuters	2233.75	78.1322	4366.83
gutenberg	1309.18	14527.9	158.843	gutenberg	1313.67	14694	152.06

2.8 5-gram model without lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	445.445	7012.83	963.398	brown	457.657	7159.16	970.119
reuters	2873.46	88.4573	5991.43	reuters	2932.42	87.9802	6074.51
gutenberg	1613.26	23947.5	178.813	gutenberg	1632.42	24482.1	170.62

2.9 5-gram model with lowercase casting

Dev data	brown	reuters	gutenberg	Test Data	brown	reuters	gutenberg
brown	380.315	3676.05	742.54	brown	388.62	3725.53	746.029
reuters	2202.57	75.0169	4313.52	reuters	2232.57	75.0993	4366.54
gutenberg	1307.49	14524.6	155.11	gutenberg	1311.65	14690.7	148.285

Note: I tried n-gram models up to an n-value of 10, but their results were almost identical to the 5-gram models so I've opted not to include them in the results.

3 Qualitative Analysis on In-Domain Text

As can be seen from the results, the perplexity was negatively correlated with the size of the maximum n-gram in all cases, but preprocessing all of the words to be lowercase was even more effective. Even the bigram model with lowercase casting performed much better than the 5-gram model without it, took up less memory, and was able to be trained much faster. Unfortunately neither of these models nor any other I trained were able to construct a sensible sentence.

4 Qualitative Analysis on Out-of-Domain Text

In all cases, the model that generalized best to a non-native corpus was models trained on the brown corpus reading the gutenberg corpus. It was noticably better than second place, which were models trained on the gutenberg corpus reading the brown corpus, which itself was much better than any non-native result involving the reuters corpus. This most likely means that the brown and gutenberg corpus are more similar to each other than either one is to the reuters corpus. This would make a lot of sense, as the reuters corpus includes lots of technical terms and abbreviations, in addition to things like company names that do not match how the words are used in native english. In contrast, both the brown corpus and the gutenberg corpus are comprised of conversational english with a smaller density of proper nouns. Between the two, the brown corpus seemed to be much more similar to the reuters corpus than the gutenberg corpus. This is most likely because the gutenberg corpus is comprised of novels that contain a lot of antiquated english and archaic terms, none of which appear in the reuters corpus. This places the brown corpus firmly in the middle of the reuters and the gutenberg corpuses, more similar to either one than the two are to each other, which explains why models trained on it generalize the best.

5 Statement of Collaboration

In addition to my interactions with CampusWire, I collaberated with Daniel Ruiz and John English on parts of the homework. Together the three of us discussed what would become the list of dictionaries model that I implemented in my code. We also helped each other understand the provided code and debug code we had written.