Language Model

Ramabhadra V
ramavyasa@gmail.com

1 Pre-processing and data splits

I have split all the categories in Brown corpus into 70:30 split for training, testing. For Gutenburg corpus, I have used '.' as end of sentence and split the total number of sentences into 70:30 for training and testing. Since, I have used only stupid-backoff there was no need for any validation set as there were no hyperparameters to tune.
I have converted all the capital letters to small letters and removed non-alphanumeric characters. I have used NLTK lemmatizer to lemmatize the words.

2 Training

I have used two method to train.
1. Words are combined with ' ' in between. I have maintained seperate dictionaries for different n-grams.
ex: The sentence "This is a sentence" splitting into bi-grams becomes ["This is":1,"is a":1,"a sentence":1] 2. dictionary inside dictionaries are formed for different n-grams. ex: The sentence "This is a sentence" for bi-grams becomes {'This':{'is':1},'is':{'a':1},'a':{'sentence':1}}

model from 1 is used for calculating perplexities.
model from 2 is used to generate sentences.

3 Testing

Stupid backoff is used to calculate perplexities.
Steps:
1. Read a sentence.
2. start with n-grams. if we have seen the n-gram,

$$P(w_n|w_1, w_2, ..., w_{n-1}) = \frac{C(w_1, w_2, ..., w_n)}{C(w_1, w_2, ..., w_{n-1})}$$

3. If the above n-gram has not been seen then backoff,

$$P(w_n|w_2, w_3, ..., w_{n-1}) = \frac{C(w_2, w_3, ..., w_n)}{C(w_2, w_3, ..., w_{n-1})}$$

4. Now, perplexity is calculated as follows,

$$PP(W) = (\prod_{i=1}^{N} \frac{1}{p(w_i|w1...w_{i-1})})^{\frac{1}{N}}$$

,where N = length of the sentence. I have now proceeded in 2 ways.
1. Calculate perplexities of each sentence and take average.
2. Calculate perplexity for whole test set. Now, N = count of all words. Perplexities are summarized in table 1 and table 2

4 Sentence generation

Again, stupid backoff is used. At each step, a random word is picked from top 10 possible words.
steps:
1. start with producing a unigram.(pick a unigram randomly from top 10 most occuring unigrams.)
2. If possible, pick randomly from top 10 words that are possible in front of the previous unigram. Else start again
3. if there are any possible trigrams possible then pick from that set. Else backoff, choose from all possible words that can come infront of the previous word. Else, start again. So on.

5 observations

1. Perplexity is higher if lemmatizer is not used.
2. Sentences wont make any sense if Lemmatizer is used.
3. out of 10 words, around 6-7 makes sense. rest appear random.
4. perplexity decreases as n is increased.
5. S1 setting produces the lowest perplexity. Hence that is used to generate sentences.

Setting	1-gram	2-gram	3-gram	4-gram	5-gram
S1	1062	352	213	153	114
S2	1430	1112	741	564	459
S3	1594	513	300	211	156
S4	1576	1159	756	572	464

Table 1: Average Perplexity for sentences.

Setting	1-gram	2-gram	3-gram	4-gram	5-gram
S1	4.9	3.8	3.4	3.1	2.9
S2	6.4	7.2	6.7	6.4	6.2
S3	1.7	1.57	1.51	1.47	1.43
S4	3.7	3.8	3.6	3.5	3.4

Table 2: Perplexity for full test set.