CS 124/LING 180/LING 280
From Languages to Information
Week 2: Group Exercises on Language Modeling
Winter 2018

Dan Jurafsky

Tuesday, January 23, 2018

1 Part 1: Group Exercise

We are interested in building a language model over a language with three words: A, B, C. Our training corpus is

`AAABACBABBBCCACBCC`

1. First train a unigram language model using maximum likelihood estimation. What are the probabilities?
   Reminder: We don't need start or end tokens for training a unigram model, since the context of each word doesn't matter. So, we won't add any special tokens to our corpus for this part of the problem.

   Answer:

$$P(A) = \frac{6}{18}$$
$$P(B) = \frac{6}{18}$$
$$P(C) = \frac{6}{18}$$

2. Next train a bigram language model using maximum likelihood estimation. For this problem, we'll add an end token, $< end >$, at the end of the string, so that we can model the probability of the sentence ending after a particular word. If you chose to add a start token as well, that's fine too, but these solutions assume no start token. Fill in the probabilities below. Leave your answers in the form of a fraction.

   Answer:

   Our corpus now becomes: `AAABACBABBBCCACBCC<end>`

$$P(A|A) = \frac{count(AA)}{count(A)} = \frac{2}{6}$$
$$P(A|B) = \frac{count(BA)}{count(B)} = \frac{2}{6}$$
$$P(A|C) = \frac{count(CA)}{count(C)} = \frac{1}{6}$$
$$P(A| < end >) = \frac{count(< end > A)}{count(< end >)} = \frac{0}{1}$$
$$P(B|A) = \frac{count(AB)}{count(A)} = \frac{2}{6}$$
$$P(B|B) = \frac{count(BB)}{count(B)} = \frac{2}{6}$$
$$P(B|C) = \frac{count(CB)}{count(C)} = \frac{2}{6}$$
$$P(B| < end >) = \frac{count(< end > B)}{count(< end >)} = \frac{0}{1}$$
$$P(C|A) = \frac{count(AC)}{count(A)} = \frac{2}{6}$$
$$P(C|B) = \frac{count(BC)}{count(B)} = \frac{2}{6}$$
$$P(C|C) = \frac{count(CC)}{count(C)} = \frac{2}{6}$$
$$P(C| < end >) = \frac{count(< end > C)}{count(< end >)} = \frac{0}{1}$$
$$P(< end > |A) = \frac{count(A < end >)}{count(A)} = \frac{0}{6}$$
$$P(< end > |B) = \frac{count(B < end >)}{count(B)} = \frac{0}{6}$$
$$P(< end > |C) = \frac{count(C < end >)}{count(C)} = \frac{1}{6}$$
$$P(< end > | < end >) = \frac{count(< end >< end >)}{count(< end >)} = \frac{0}{1}$$

3. Now evaluate your language models on the test corpus:

   ```
   ABACABB
   ```

   What is the perplexity of the unigram language model evaluated on this corpus? Since we didn't add any special start/end tokens when we were training our unigram language model, we won't add any when we evaluate the perplexity of the unigram language model, either, so that we're consistent.

Answer:

$$\sqrt[-7]{(\frac{1}{3})^7} = (\frac{1}{3})^{-1} = 3 \quad (1)$$

What is the perplexity of the bigram language model evaluated on this corpus? Since we added an end token when we were training our bigram model, we'll add an end token to this corpus again before we evaluate perplexity.

Answer: In this case, the corpus becomes: `ABACABB<end>`

The perplexity is therefore:

$$\sqrt[-7]{P(B|A)P(A|B)P(C|A)P(A|C)P(B|A)P(B|B)P(< end > |B)} \quad (2)$$

Note that in our unsmoothed bigram model, $P(< end > |B)$ is 0, since $< end >$ never occurs after B in our training set. Therefore, the probability of this corpus is 0, so its perplexity is infinite.

4. Now repeat everything above for add-1 smoothing.

   Answer: Add-one smoothing is simple for the unigram model. We simply add 1 to each numerator, and 3 (the vocabulary size) to each denominator. For this particular corpus, the actual probabilities are the same as without smoothing; all unigram probabilities are equal to one-third. Note that this is not true for most corpuses, as smoothing usually does in fact change the probabilities in your language model.

$$P(A) = \frac{7}{21}$$
$$P(B) = \frac{7}{21}$$
$$P(C) = \frac{7}{21}$$

   Add-one smoothing is slightly more complicated for the bigram model. Since we used an end-token, $< end >$, our vocabulary size is 4 in this case, so we add 1 to each numerator and 4 to each denominator.

$$P(A|A) = \frac{count(AA)+1}{count(A)+4} = \frac{3}{10}$$
$$P(A|B) = \frac{count(BA)+1}{count(B)+4} = \frac{3}{10}$$
$$P(A|C) = \frac{count(CA)+1}{count(C)+4} = \frac{3}{10}$$
$$P(A|<end>) = \frac{count(<end>A)+1}{count(<end>)+4} = \frac{1}{5}$$
$$P(B|A) = \frac{count(AB)+1}{count(A)+4} = \frac{3}{10}$$
$$P(B|B) = \frac{count(BB)+1}{count(B)+4} = \frac{3}{10}$$
$$P(B|C) = \frac{count(CB)+1}{count(C)+4} = \frac{3}{10}$$
$$P(B|<end>) = \frac{count(<end>B)+1}{count(<end>)+4} = \frac{1}{5}$$
$$P(C|A) = \frac{count(AC)+1}{count(A)+4} = \frac{3}{10}$$
$$P(C|B) = \frac{count(BC)+1}{count(B)+4} = \frac{3}{10}$$
$$P(C|C) = \frac{count(CC)+1}{count(C)+4} = \frac{3}{10}$$
$$P(C|<end>) = \frac{count(<end>C)+1}{count(<end>)+4} = \frac{1}{5}$$
$$P(<end>|A) = \frac{count(A<end>)+1}{count(A)+4} = \frac{1}{10}$$
$$P(<end>|B) = \frac{count(B<end>)+1}{count(B)+4} = \frac{1}{10}$$
$$P(<end>|C) = \frac{count(C<end>)+1}{count(C)+4} = \frac{2}{10}$$
$$P(<end>|<end>) = \frac{count(<end><end>)+1}{count(<end>)+4} = \frac{1}{5}$$
$$5678uj65442322409$$

Unigram perplexity ends up being the same as before (since the smoothed unigram probabilities are the same as the unsmoothed unigram probabilities for this specific corpus):

$$\sqrt[-7]{(\frac{1}{3})^7} = (\frac{1}{3})^{-1} = 3 \tag{3}$$

Bigram perplexity is a little bit different with smoothing. We use the same formula as before, but this time our probability is no longer 0, so we can compute a finite perplexity:

$$
\begin{aligned}
&\sqrt[-7]{P(B|A)P(A|B)P(C|A)P(A|C)P(B|A)P(B|B)P(<end>|B)} \\
&= \sqrt[-7]{(\frac{3}{10})^5 \frac{2}{10} \frac{1}{10}} \\
&= \sqrt[-7]{\frac{486}{10^7}} \\
&= \frac{10}{486^{\frac{1}{7}}} \\
&\approx 4.132
\end{aligned}
$$

2 Problem 2: Challenge Problems

1. What is the difference between using an UNK token (for unknown words) and smoothing? What situations would you use one versus the other?

   Answer: UNK tokens are used when unknown words are observed in the dataset. This can happen in both train and test datasets - for some language model applications, it is desirable to have a fixed vocabulary, and use the UNK token to treat any other words both when training and testing. If that is the case, at training and test time, any token not belonging to the fixed vocabulary would be converted into an UNK token before counting even starts.

   Smoothing, on the other hand, is a way to redistribute the probabilities of observed occurrences from the training dataset. For almost all smoothing methods, the events (unigrams, bigrams ... etc) with higher probability would be discounted and rare events would be boosted. This is a good remedy for small datasets, where probability distribution between unigrams and bigrams may be skewed due to the small size. However, in some cases this also cause the higher probability events to be discounted too much. Smoothing could potentially be used to deal with unknown words - essentially treating unknown words as tokens that appear 0 times in the training corpus, and their probabilities are estimated from the smoothing method.

   They are really disjoint methods that should be used concurrently in a complete language model. The language models that you build only deal with UNK tokens at test time, where you should think about how to check for UNK tokens, and what probability to assign to the UNK tokens. At train time however, make sure you already perform the desired smoothing to the probabilities, so that you don't have to be worrying about that at score time. Even though the score function follows directly under the train function in the language model code, you should really be thinking about them as separated processes, where you apply smoothing in train and check for UNK in test.

2. Suppose you build an interpolated trigram language model, with three weights lambda1 for unigrams, lambda2 for bigrams, and lambda3 for trigrams. Normally we set these lambdas on a held-out set. Suppose

instead we set them on the training data. This will cause the lambdas to take on very unusual values. What will these lambdas look like? Why?

$$(\lambda_1, \lambda_2, \lambda_3) = (0, 0, 1) \tag{4}$$

Answer: An n-gram model with a larger value of n is always a better fit for the training data, so the lambda value corresponding to the trigram model will be set to 1, and the bigram and unigram models will have lambda values equal to 0 (in other words, the bigram and unigram models will be completely ignored by the interpolated model).

This isn't usually desirable behavior, because even though our trigram model might fight our training data very well, it might overfit, and might not generalize well to an unseen test set. This is why we use a held-out set, which wasn't seen in training, for setting the lambda values.

3. Show that if we estimate two bigram language models using unsmoothed relative frequencies (MLE), one from a text corpus and the second from the same corpus in reverse order, the models will assign the same probability to new sentences (when applied in forward and backward order respectively). Hint: First write out the entire equation for sentence probabilities in terms of counts.

   Answer: Let the sentence be $(w_1, w_2, w_3, ..., w_n)$. Begin and end are the start and end tokens. T is the total number of sentences.

   For forward order, the probability of the sentence is:

$$Pr(Begin, w_1) \prod_{i=1}^{n-1} Pr(w_{i+1}|Prevwordisw_1) Pr(End|w_n)$$
$$= \frac{count(Begin, w_1)}{T} \prod_{i=1}^{n-1} \frac{count(w_i, w_{i+1})}{count(w_i)} \frac{count(w_n, End)}{count(w_n)}$$

   For backward order, the probability of the sentence is:

$$Pr(w_n, End) \prod_{i=1}^{n-1} Pr(w_i|Nextwordisw + 1) Pr(Begin|w_i)$$
$$= \frac{count(w_n, End)}{T} \prod_{i=1}^{n-1} \frac{count(w_i, w_{i+1})}{count(w_i + 1)} \frac{count(Begin, w_i)}{count(w_i)}$$

   Both are equal because

$$count(w_1) \prod_{i=1}^{n-1} count(w_{i+1}) = count(w_n) \prod_{i=1}^{n-1} count(w_i) \tag{5}$$