Problem

When we build a bigram model from a training corpus, we want to compute $P(w_{i+1}|w_i)$ for which we compute the probability of word i + 1 given the previous word i.

Take a toy corpus “<s> I am a student </s> <s> I like dance </s>” for an example, P(am | I) = ½ because “am” occurs one of two times after “I”. We build a bigram model going through the training corpus calculating each $P(w_{i+1}|w_i)$ and test the model on the following corpus “<s> a student like dance <s>”. The probability the model assigns to this corpus P = P(a | <s>) * P(student | a) * P(like | student) * P(dance | like) = 0 which suggests this testing corpus is very rare.

However, the probability the model assigns to the testing corpus should not be zero as all the words come from the toy corpus. Bigram models only care about the relationship between two adjacent words. The reason why P(like | student) goes to zero is because the model does not have any information on “like” given “student” and thus assigns zero. The zero probability that bigram models assign has been a problem when used for testing.

Solutions

Add-one smoothing

How do we deal with the problem? A simple solution is to apply add-one smoothing to a bigram model so $P(w_{i+1}|w_i)$ will never be zero. Here is how it goes:

$P(w_{i+1}|w_i) = c(w_i, w_{i+1}) + 1 / c(w_i) + V$

$c(w_i, w_{i+1})$ represents the number of times word i + 1 occurs after word i, and $c(w_i)$ represents the number of times word i occurs, and V represents the number of unique words in the training corpus.

Back to our toy corpus examples, p(am | I) = (1 + 1) / (2 + 8) = ⅕ which shrinks ½ - ⅕ = ⅘, and p(like | student) = (0 + 1) / (2 + 8) = 0.1 which boosts 1/10 from 0. It looks like we have already solved the problem with add-one smoothing, but is there any flaws for the smoothing method when applied on bigram models?

One flaw is when a bigram model has too many zero entries, add-one smoothing will make the probability assign to a non-zero entry less convincing than it was. The following table shows why.

	<s>	I	am	a	student	like	dance	</s>
<s>		1						
I			0.5			0.5		
am				1				

a					1			
student								1
like							1	
dance								1
</s>								

Bigram model without smoothing

Those empty entries should be filled in with zeros but we leave them empty just for simplicity.

We can observe from the table P(a | am) = 1 which tells us our bigram model predicts word “a” always follows word “am”. Here is how row 4 changes after add-one smoothing is applied.

am	1/9	1/9	1/9	2/9	1/9	1/9	1/9	1/9

Row 4 applied with add-one smoothing

P(a | am) = 2/9 and it tells us our smoothed bigram model predicts only 2 out 9 times “a” follows word “am”, and it is not what we interpret from the training corpus. The question comes up: what smoothing methods would be better to deal with bigram models with lots of zero entries? Good-Turing is one of the answers.

Good-Turing Smoothing

Good-Turing smoothing is one of the advanced smoothing methods which use the count of things we have seen once to help estimate the count of things we have

never seen before. Good-Turing smoothing will replace those zero entries of bigram models with something else we will cover shortly.

Before we start covering how Good-Turing smoothing works, we want to introduce a notation and terminology associated with, Nc (Frequency of frequency c). Nc denotes how many count a frequency c is. The following table shows the count of each unique word in our training corpus.

I --- 2	am --- 1	a --- 1	student --- 1
like -- 1	dance --- 1	<s> --- 2	</s> --- 2

N1 = 5 and N2 = 3.

We covered an example in class “a fisherman has caught a total of 15 fish of which have 7 carps, 2 perch, 3 white fish, 1 salmon, 1 trout, and 1 eel”. We were asked “How likely the next fish will be a new species?”

Good-Turing estimates how likely a new thing occur by using the likelihood of things that occur once. From the example, we calculate N1 = 3 so the probability of catching a new species is 3/15 = 1/5. If we were to be asked “How likely the next species is trout”, we would have answered in tuition “trout occurs once every 15 fish caught, so the answer is 1/15”. It turns out the answer should be less than 1/15. 1/15 is calculated under the assumption that there are only six species that exist. Since we have the probability of the next fish in a new species calculated with Good-Turing, 3/15, and thus the probability of the next fish in trout will be less than 1/15. The question comes up “how do you calculate the probability?”

As shown above, Good-Turing has a formula calculating things with zero count P = N1 / N. Good-Turing has another formula calculating things with more than zero count **P = b / N** where **$b = (c + 1) N_{c+1} / N_c$** . We have seen “trout” once so c = 1, b = 2 * N2/ N1 = 2/3. P = (⅔) / 15 = 2 / 45 which drops 3/45 - 2/45 = 1/45.

How do we calculate N_{c+1} if c is the biggest number in frequency count of training corpus? What if N_{c+1} does not exist when c gets big? We will use Simple Good-Turing replacing N_c with a best-fit power low, so we will only calculate things with low number of frequency count. For the purpose of demo, we will only apply Good-Turing smoothing to the entries that have frequency count 0 and 1.

Demo

In this demo, we will make use of the bigram model we built in homework 2 part 3 and compare the perplexity with and without Good-Turing smoothing. Code and output files can be found in the project folder.

Observations

After we apply Good-Turing smoothing to the entries that have frequency count 0 and 1, the output file generated “Good_Turing_smoothed_eval” is exactly the same as the output file our bigram model generate without any smoothing. The reason is that the testing corpus is very small and $C(w_i, w_{i+1})$ for each word in the corpus is greater than 1 and thus our Good-Turing smoothing does not do much help on smoothing the

distribution. For future testing, we may want to choose larger size testing corpus which has adjecent words i and i+1 such that $C(w_i, w_{i+1}) = 0$ or 1.

Another observation is Good-Turing smoothing does way better than Add-one smoothing (add-delta smoothing in our demo). The proof is the perplexities in “smoothed_eval” are bigger than the ones in “Good_Turing_smoothed_eval”. The reason is just as I briefly mentioned at the end of “Add-one smoothing” section which is our bigram model has so many zero entries and to add anything to every entry of the table and normalize the table will dramatically drop the predominance of non-zero entries.