CSEP 517 : Homework no. 1

Aerin Kim
aerinkim@cs.washington.edu

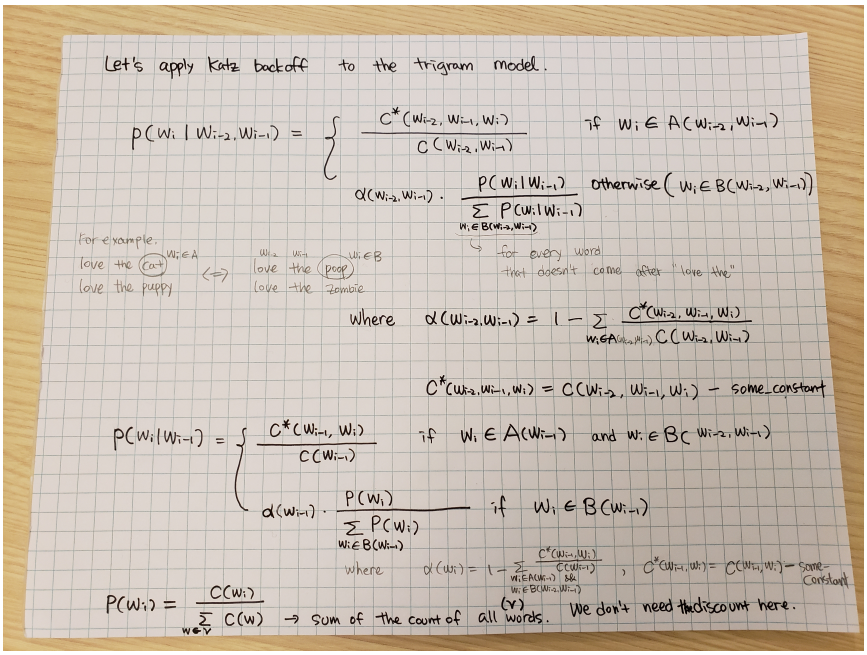
October 12 2018

1 Smoothing

1.1 Is the above model a valid probability distribution? Justify your answer. If it is not a valid probability distribution, suggest how to make it one by modifying p_1, p_2 and p_3, using p_{ML} (the maximum likelihood estimate) and/or the count function c, and briefly explain why your modification works

No. Above model's probability doesn't sum to one. If we replace an N-gram which has zero probability with a lower-order N-gram, we would be adding probability mass from another probability distribution and the total probability assigned to all possible strings by the language model would be greater than 1.

In order for a backoff model to give a correct probability distribution, we need to discount the higher-order N-grams to save some probability mass for the lower order N-grams. This can be done by using Katz backoff. In Katz backoff, we define α to set the maximum of the sum of probability when w belongs to $\mathbb{B}$. Thus even if the sum of the probability is 1, it gets discounted to α.



2 Language Model Classifier

2.1 Sketch out how you can make use of language models for text categorization. Use equations whenever possible.

Let's say we have words $x \in \mathbb{X}$, a sequence of words $s \in \mathbb{S}$ of a document where $\mathbb{S}$ is the document space, a fixed set of classes $\mathbb{Y} = \{y_1, y_2, \ldots, y_j\}$, and a training set $\mathbb{D}$ of labeled documents (s, y) where $(s, y) \in \mathbb{S} \times \mathbb{Y}$. For example, let's say we have millions of lines of dialogue text between customers and customer representatives in customer centers and managements want to control the quality of how their representatives treat their customers online.

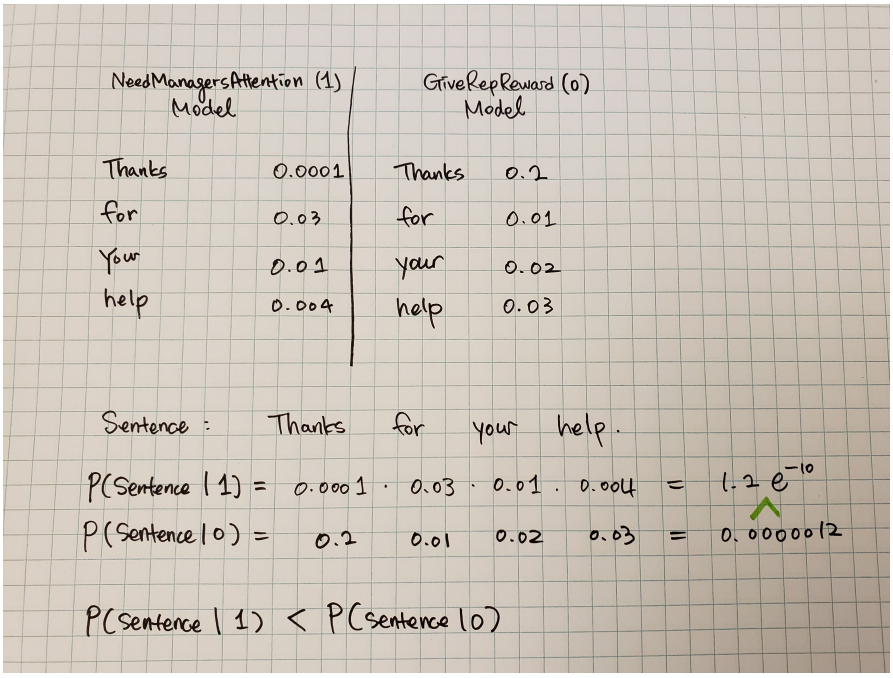
$$\mathbb{X} = \{are, youkidding, me, thanks, for, your, help, \ldots\}$$

$$\mathbb{Y} = \{NeedManagersAttention, GiveRepReward, Neutral\}$$

$$(s, y) = (\text{“}Are\ you\ kidding\ me\text{”}, NeedManagersAttention)$$

$$(s, y) = (\text{“}Thanks\ for\ your\ help\text{”}, GiveRepReward)$$

To use language model as a classifier, first, we assume each class is the unigram language model conditioned on the class. Therefore We assign each word the probability $P(\text{word}) = P(x) = P(x|y)$ and assign each sentence the probability $P(\text{sentence}) = \prod_{x \in s} P(x) = \prod_{x \in s} P(x|y)$. Once we get the $P(\text{sentence})$, we'll just pick whichever class (model) that has the higher probability as the more likely class.



As you could see, this model is the same as Naive Bayes.

Also, we need to use every word appeared in the dialogue text so that $P(x_i) > 0$ and make sure $\sum_{x \in X} P(x|y) = 1$ for each class using back-off. This way, the model meets the requirements of the Language Model.

3 [Programming] Language Models

3.1 Report the perplexity scores of the unigram, bigram, trigram language models for your training, dev and test sets, and elaborate all your design choices (e.g., if you introduced START symbols in addition to STOP symbols, how you handled out-of-vocabulary words). Briefly discuss the experimental results.

One of the important decisions that I had to make in this problem was what the MLE distribution should be when the n-gram has never been seen in the training set. More specifically, what should I do with $P_{mle}(w_i|w_{i-2}, w_{i-1})$ when $c(w_{i-2}, w_{i-1}) = 0$? Instead of implementing the back-off, I used the add-K smoothing which is equivalent to using a uniform distribution $1/|V|$ where V is a set of entire vocabulary (including special tokens such as OOV words and sentence boundaries). In our development dataset, total 7.5 % of trigram has not been seen in the training set. Another design decision was I only used STOP symbols and didn't use the START symbol.

First, I experiment with different OOV thresholds on the dev set. All corpus was lower-capped. The way I converted the words to UNK in test set was if the token appear less than the threshold in the training set, I replace that token to the UNK.

Perplexity scores on dev set (lowercased)			
oov thres	unigram	bigram	trigram
1	778.83	4006.63	23559.44
5	515.97	1566.68	12958.39
10	381.85	818.46	7828.53
20	248.49	349.75	3288.90
30	184.23	202.07	1715.60

Interestingly, the perplexity of trigram was higher than that of bigram and unigram. I suspect this might be because we don't have enough training data. So I looked up top 30 most frequent bigrams and trigrams in the training set and most of them were related to the ¡STOP¿ tokens and end of the sentences therefore having a weak prediction power in the actual body of the sentences. Below is the table of the top 30 most frequent trigrams/bigrams in the training data.

Trigram	Count
<UNK>. <STOP>	1768
it . <STOP>	538
him . <STOP>	408
of the <UNK>	399
said . <STOP>	349
the <UNK>of	280
one of the	273
<UNK>of the	260
them . <STOP>	260
the United States	252
<UNK>and <UNK>	250
as well as	194
in . <STOP>	180
time . <STOP>	169
in the <UNK>	162
<UNK><UNK> <UNK>	161
Af . <STOP>	161
<UNK>in the	147
out of the	142
me . <STOP>	142
her . <STOP>	139
the <UNK>.	138
<UNK><UNK>and	134
and <UNK>.	133
the <UNK>and	127

In the second experiment, I didn't convert the corpus into lowercase. Other conditions were the same as the first experiment.

Perplexity scores on dev set (case sensitive)			
oov thres	unigram	bigram	trigram
1	871.80	4468.40	24737.25
5	545.51	1590.44	12743.22
10	392.19	794.28	7414.69
20	248.45	328.68	2976.69
30	180.02	184.11	1486.43

This result makes sense because the more characters used in the corpus, it's translated into more uncertainty. Therefore, the entropy increases as the uncertainty of which character will be sent increases.

Finally, I decided to lowercase everything and used the oov threshold of 1. Even if higher oov thresholds gave me the better result, I stick to 1. Because if we convert more words into oov, it's cheating for the performance number. Below is the result on the training/dev/testing sets.

Perplexity scores			
dataset	unigram	bigram	trigram
training	993.81	50.70	4.02
dev	871.80	4468.40	24737.25
test	880.97	4479.84	24699.08

This result seems reasonable in that the result on the training set is much higher than those of dev/test sets and dev/test results were pretty similar. Also, if you look at the result on training set, the perplexity of trigram is much lower than that of bigram and unigram. This was the expected behavior for this question however we didn't have the enough training data.
One last thing to note is that this performance can also be improved a lot by how we parse the tokens. I simply used string.split() in python however there are several industry-grade parsers that will do better jobs in parsing and tokenizing.

4 [Programming] Smoothing

4.1 (a) Report perplexity scores on training and dev sets for various values of K (no interpolation). Try to cover a couple of orders of magnitude (e.g. K = 10, K = 1, 0.1, ...). (b) Similarly, report perplexity scores on training and dev sets for various values of lambdas (no K smoothing). Report no more than 5 different values for your lambdas. (c) Putting it all together, report perplexity on the test set, using different smoothing techniques and the corresponding hyper-parameters that you chose from the dev set. Specify those hyper-parameters.

(a) Interestingly, I got the same perplexity numbers regardless of the Ks that I used.

Perplexity scores on training set			
K	unigram	bigram	trigram
any	872.53	14245.13	8481.46

Perplexity scores on dev set			
K	unigram	bigram	trigram
any	778.83	14286.78	8535.93

Maybe my code has a bug, however one possible interpretation could be there were many bigrams and trigrams which were zero-count. If that's the case, regardless of K, the probability of trigrams follows uniform distribution which is $1/V$ (where V is the total number of unique unigrams).

(b)

Perplexity scores on training set	
lambdas	Perplexity
(1/3, 1/3, 1/3)	6.20
(0.1, 0.1, 0.8)	4.05
(0.1, 0.8, 0.1)	10.41
(0.8, 0.1, 0.1)	14.59
(0.2, 0.3, 0.5)	4.92

Perplexity scores on dev set	
lambdas	Perplexity
(1/3, 1/3, 1/3)	107.65
(0.1, 0.1, 0.8)	161.16
(0.1, 0.8, 0.1)	150.19
(0.8, 0.1, 0.1)	136.64
(0.2, 0.3, 0.5)	115.31

I chose to use (1/3, 1/3, 1/3) for lambdas.

(c) Perplexity on the test set was 109.03 with the lambda of (1/3, 1/3, 1/3) and K = 0.0000001.

4.2 If you use only half of the training data, would it in general increase or decrease the perplexity on the previously unseen data? Discuss the reason why.

Generally it will decrease the perplexity or increase the performance. Perplexity is an exponentiation of the cross entropy. More data decreases the generalization error because the model becomes more general by virtue of being trained on more examples.

4.3 If you convert all words that appeared less then 5 times as UNK (a special symbol for out-of-vocabulary words), would it in general increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of words that appeared just once as UNK? Discuss the reason why.

In general, it will decrease the perplexity. For example, let's look at P('She went to Kimshin') = P(She) * P(went) * P(to) * P(Kimshin). If we relax the OOV threshold, we could substitute P(Kimshin) with P(UNK) which has much higher probability than P(Kimshin).