CSEP 517: Homework 1 Language Models

Question 1 - Smoothing

When doing maximum likelihood estimate $\sum_i P(w_i \mid w_{i-2}w_{i-1}) = 1$. Therefore if we back off to a bigram model for the case where the trigram probability is 0 (as is the case with p2 and p3) then we would be adding extra probability mass. This would result in the total trigram probability to be more than 1! To fix this we must use some discounting on the trigram probability and redistribute it to the bigram and unigram models. The probability for p1 then needs to be discounted. The amount of discount needs to be determined based on how much we think we will be missing. If we think our language model only covers 50% of available trigrams we can use a discount of 50%. If the current probability of a trigram is $P_1(w_{i-2}, w_{i-1}, w_i) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-2}, w_{i-1})}$ The new probability for p1 will then be $P_1{}^*(w_{i-2}, w_{i-1}, w_i) = \frac{c(w_{i-2}, w_{i-1}, w_i) - \lambda_3}{c(w_{i-2}, w_{i-1})}$ where λ_3 is the discount amount.

The next step will be to redistribute this discount to the other probabilities p2 and p3. To calculate the discount amount we just subtract our new discounted probability from 1. Let set $\alpha(w_{i-2}, w_{i-1}) = 1 - \sum_{w \in A(w_{i-2}, w_{i-1})} \frac{c(w_{i-2}, w_{i-1}, w_i) - \lambda_3}{c(w_{i-2}, w_{i-1})}$. This gives us the total amount of probability we took away from the trigram probability. We can now repeat this step for the bigram probabilities, since in the case of p3, we don't have a trigram or a bigram probability. $P_2{}^*(w_{i-1}, w_i) = \frac{c(w_{i-1}, w_i) - \lambda_2}{c(w_{i-1})}$ and the discount being $\alpha(w_{i-1}) = 1 - \sum_{w \in A(w_{i-1})} \frac{c(w_{i-1}, w_i) - \lambda_3}{c(w_{i-1})}$. We would just carry these probabilities down to the lower level. Every time we discount and we distribute. The resulting equations then become:

1. P1 gets discounted resulting in $P_1 = P_1^*\left(w_i \mid w_{i-2},\ w_{i-1}\right)$ where

$$P_1^*\left(w_{i-2}, w_{i-1}, w_i\right) = \frac{c\left(w_{i-2}, w_{i-1}, w_i\right) - \lambda_3}{c\left(w_{i-2}, w_{i-1}\right)}$$

2. P2 gets the discounted amount from P1 and gets discounted itself resulting in

$$P_2 = \alpha\left(w_{i-2}, w_{i-1}\right) \frac{P_2^*\left(w_i \mid w_{i-1}\right)}{\sum_{w \in B\left(w_{i-2}, w_{i-1}\right)} P_{ML}\left(w \mid w_{i-1}\right)}$$ where

$$P_2^*\left(w_i \mid w_{i-1}\right) = \frac{c\left(w_{i-1}, w_i\right) - \lambda_2}{c\left(w_{i-1}\right)}$$

3. P3 gets the discount remaining from P2 resulting in

$$P_3 = \alpha\left(w_{i-1}\right) \frac{P_{ML}\left(w_i\right)}{\sum_{w \in B\left(w_{i-1}\right)} P_{ML}(w)}$$

Question 2 - Language Model Classifier

For text categorization let's assume we are doing a spam filter. There are 2 categories possible: 1. Spam, 2. Not Spam. We would need a training data set for each of the categories. A training set of spam examples, and a training set of not spam examples. We would then compute the ngram counts on each of the data sets. This will result in a spam counts look up table, and a non spam counts lookup table.We would also apply some smoothing like linear interpolation to account for missing values. When running our evaluation we would tokenize the incoming text and calculate the probabilities of the text based on each of the 2 sets. The set that provides a higher likelihood (probability) of the incoming text would be the winning categorization. This can be expanded to other categorization problems such as news topic or authorship attribution. The only difference would be the number of training sets we would need. The more categorize the closer the resulting probabilities will be to each other. To make sense of this we can also provide a confidence level. This level would be based on how close the resulting probabilities were to each other. If for example, two categorizations provide probability .95 and .94, then the category corresponding to .95 would win but with low confidence. While if the difference is large, .9 to .5 for example, then we would have high confidence.

Question 3 - Language Models

Design choices:

1. Not Using Start key word
    a. Although using start makes computing bigram and trigram probabilities more accurate, I ran into issues understanding what to do in the beginning case. What is P(Start,Start) for the denominator? Following the first equation was simple as I just got the unigram and bigram probabilities and multiplied.
    b. Using start would have given better numbers since p(first word) is different than p(first word | START). The later gives a better estimate. The word To for example rarely starts a sentence, which would be captured if we modeled with start.
2. Data Structure
    a. Instead of using a Hashmap with key - value, I used a map of key, map where the second map had key,value. In trigram example, the look up would be: 1) look at the hashmap for the bigram(wi-2,wi-1), if it exists get the value map. 2) check the value map for wi.
3. OOV words
    a. As suggested I implemented UNK replacement for low frequency words. The way I did this was by preprocessing all the data.
    b. First step was to get the vocabulary, using the training set we got a word count for each word (basically unigram counts). Then all words that were below the unk Threshold would be removed and added to the unk count.
    c. Step 2 was then taking the vocabulary and preprocessing all 3 files, replacing any word not in vocabulary with UNK.
    d. I used an UNK Threshold of 2.

Results:

1. Unigram model
    a. Training Perplexity: 137
    b. Dev Perplexity: 125
    c. Test Perplexity: 125
2. Bigram model
    a. Training Perplexity:22
    b. Dev Perplexity: Infinity
    c. Test Perplexity: Infinity
3. Trigram model
    a. Training Perplexity: 4
    b. Dev Perplexity: Infinity

c. Test Perplexity: Infinity

4. Discussion
    a. Without smoothing, once we step into bigram and trigram models, we run into a lot of 0 values. Since they are not handled we get 0 probabilities and therefore infinite perplexity
    b. I find it weird that for unigram my perplexity is higher on the train set. If our probabilities are underestimating the word probabilities in the dev and test set, we may end up with less “choices” for each word and thus a lower perplexity. Meaning due to many unknown words and underestimation of probabilities, the perplexity becomes high. If we were to do some extrinsic evaluation on this we would end up with better numbers for train set.

Question 4 - Smoothing

Part 1

1) Add-K Smoothing
    a) Training set

K value	0.01	0.1	1	10
Perplexity	47	181	519	800

    b) Dev set

K value	0.01	0.1	1	10
Perplexity	291	415	622	805

2) Linear Interpolation
    a) Training set

Lambda values $\lambda_1, \lambda_2, \lambda_3$	0.33,0.33,0.34	0.1,0.3,0.6	0.6,0.3,0.1	0.3,0.6,0.1
Perplexity	24	8	62	34

    b) Dev set

Lambda values	0.33,0.33,0.34	0.1,0.3,0.6	0.6,0.3,0.1	0.3,0.6,0.1
Perplexity	14	4	42	19

3) Test Set results
    a) Add-K given best K of .01
        i) Perplexity = 291
    b) Linear Interpolation given best lambdas of 0.1,0.3,0.6
        i) Perplexity = 4

Part 2

1) If we see half the training data we can assume our vocabulary size would decrease. A smaller vocabulary would result in there being less “choice” on every word in general. Since perplexity is the branching factor measurement of “how many choices for each instance is there”, perplexity would go down. This is given that we replace unknown words with UNK. If we were to not use UNK then the result of the probability for a sentence would more likely be 0 (since we have seen less words, so we are more likely to see new words). This would then increase the perplexity. The extrinsic evaluation of this model would not be good though no matter the use of unk. Using half the training data would reduce the ability to differentiate popular from unpopular unigrams, bigrams, and trigrams. Although the perplexity will go down, the model will be predicting garbage since there is a smaller vocabulary.
2) In short. Use UNK and half training data = lower perplexity. Don’t use UNK and use half training data = higher perplexity.

Part 3

1) The higher the UNK threshold, the lower the perplexity we get. In the extreme case of where we replace every word with UNK, our perplexity would become 1. Since there is only 1 choice for each word, unk. The extrinsic evaluation would prove this is really bad since not every word in english is UNK. Therefore a model that has UNK threshold at 5 would have a lower perplexity than one with threshold of 1, but as mentioned before, might produce a worse model once we apply some extrinsic evaluation.