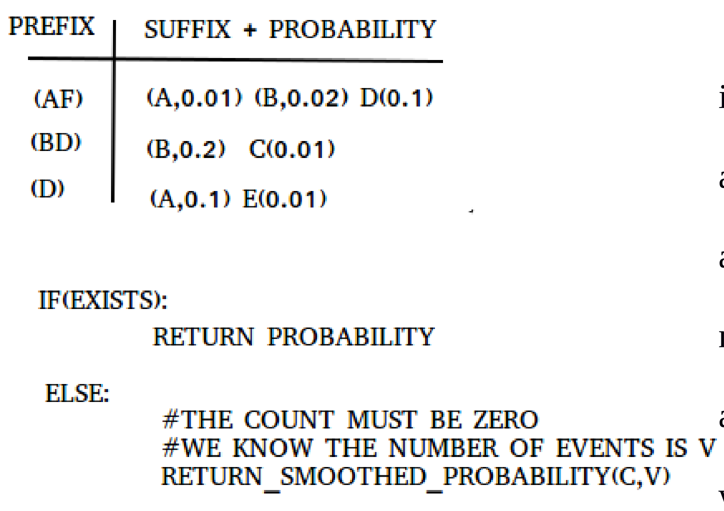
CS1671 Assignment II

Ashish Juneja

1*) What problems can occur (or have occurred in your experiments, if there is any) when the N gram language model you implemented in Part I is trained on a large training data such as the Project Gutenberg? Given that you have access to the development data, how did it help you to adapt and/or train your models?*

The biggest problem I faced in adapting my basic n-gram model to handle Project Gutenberg data was effective storage. During my toy example, I only focused on developing a working model and ignored the fact that I was storing all possible bigrams and trigrams (so I was using |V| + |V^2| + |V^3| storage. When I got to part III, my ngram.py froze my computer which is when I realized I needed to find a better way. My solution was to factor out the missing grams and compute their probabilities on the fly instead of storing them. I had a probability dictionary (shown below) with P(A|B) with the event B as the key and the event A as the value (Given A happened, B stored a list of tuples with the next possible event and the corresponding probability).
Since the assignment just said to use a smoothing technique and did not specify which one, I experimented with two types of smoothing (add one smoothing and Jelinek-Mercer smoothing).



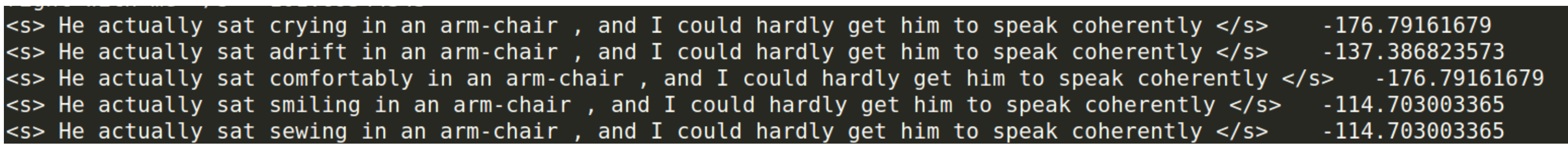
PREFIX	SUFFIX + PROBABILITY
(AF)	(A,0.01) (B,0.02) D(0.1)
(BD)	(B,0.2) C(0.01)
(D)	(A,0.1) E(0.01)

```
IF(EXISTS):
        RETURN PROBABILITY
ELSE:
        #THE COUNT MUST BE ZERO
        #WE KNOW THE NUMBER OF EVENTS IS V
        RETURN_SMOOTHED_PROBABILITY(C,V)
```

The development data helped me set the parameters in my JM Smoothing. I used a greedy descent algorithm where I would compute the perplexity for all five options and then run my test cases for each model (unigram,bigram, and trigram) to the actual answer. If the answer from the trigram was correct, I would increment its lambda parameter by a given step size constant and subtract ½ of that step size away from bigram and unigram. If the bigram model was correct and the trigram was wrong I would decrement the trigram and unigram and increase the bigram weight, and do the same for unigram if it was the only correct option. If all three were wrong, I would do nothing. Once any of the lambdas got to as low as 0.05 I would break out the descent and keep those values. I later experimented with a weighted version of JM smoothing in which I ignored the dev file and just factored the relative weights of each gram into its lambda probability (a method described by an Daniel Jurafsky in an online lecture), but found that had decreased by accuracy by around 3% so I got rid of it.

2) *How did your models perform? Were they as you expected? Why wasn't the N gram language model alone good enough for the sentence completion task? What additional tools or techniques do you think are necessary? Can the language model itself be changed to account for more ambiguities?*

The highest accuracy I received on my ngram model was 27.69%, when I trained on twentyfive of the book corpuses. I got a 23% when I trained on one corpus, and a 24% when I trained on four corpuses. The n-gram model in general is not very good at extrapolating. In this challenge we are given one sentence with five different possible words. Since all the sentences differ by one word, if the ngram model had not seen any of the five word choices, it would default to <unk> and therefore the probabilities of that word would all be the same as many of the other choices:



```
<s> He actually sat crying in an arm-chair , and I could hardly get him to speak coherently </s>    -176.79161679
<s> He actually sat adrift in an arm-chair , and I could hardly get him to speak coherently </s>    -137.386823573
<s> He actually sat comfortably in an arm-chair , and I could hardly get him to speak coherently </s>   -176.79161679
<s> He actually sat smiling in an arm-chair , and I could hardly get him to speak coherently </s>   -114.703003365
<s> He actually sat sewing in an arm-chair , and I could hardly get him to speak coherently </s>    -114.703003365
```

There are two ways of midigating this problem. One is to simply train on more grams so that the one word that is supposed to be different in the sentence is actually different and not just an <unk> so the perplexities will differ. This is what I did when I increased the number of training corpuses to fifteen, I saw a jump in improvement. If I trained on 50-60 corpuses I am confident that my accuracy would be improve even more.

Another solution, would be to implement a stronger smoothing technique, particularly one known as 'Kneser-Ney', which is generally as one of the most sophisticated smoothing techniques. Although we didn't mention it in class, it works by assigning probability based on the number of words a token follows rather then just the absolute count of the token. For instance assign a high probability to the word Fransisco only if it follows San, otherwise even if its absolute count is high in the corpus, its probability by itself will be low. If I had researched this smoothing method a bit earlier I would have chosen to implement that.

The N-gram model performs low on this task specifically because N-gram models are terrible at predicting long sentences. Most words in everday conversation are short and therefore N-grams work well enough (Ex: “I am hungry.” “Lets get food.” “Where should we go”, but when you try and apply them to sentences like the one above “He actually sat ____ in an arm-char, …. etc. “ they fall apart quickly due to the existence of subclauses, and conjunctions of ideas. A more sophisticated model, that is less linear like ngrams and perhaps look at the parse structure of a sentence, assign verbs to nouns, and break the part of speech into a more formal representation then just the less few words would probably work much better especially for large sentences such as these. The language model itself can be adjusted to drastically increase performance.

Just one example of this is looking at tense. For example consider the sentence completion challenge : He was eating _______” a) pizza, b) shoes c) window d) running e) fire

Since the ngram just looks at words, if it saw the word eat but not eating, it would mark eating of as an <unk>. However a smarter model could tag eating as a version of eating and if the model had seen 'eat pizza' in the training corpus, it could extrapolate that a) is the correct answer, unlike the current ngram model which would mark 'eating' as <unk> and all that information on context would have gone to waste. That's just one instance, so in conclusion, there are a lot of things in the language model that can be changed to account for ambiguity.