Statistical Natural Language Processing Coursework 1

Fayimora Femi-Balogun, 1049655

November 2014

Evaluation Baseline and Tests

The **Constant LM** is a language model whose probability is simply $\frac{1}{vocabularySize}$ for any word, hence the name constant. The **NGram LM** however differs. For unigrams, it calculates $p(\text{word}_i)$, for bi-grams, it's $p(\text{word}_i|\text{words}_{i-1})$ and for trigrams $p(\text{word}_i|\text{words}_{i-2,i-1})$.

In order to check that the baselines work, the sum of conditional probabilities of each word in the vocabulary given a set history of word(s) should sum to 1. This was tested on all models. This in turn proves that other methods it depends on are logically correct. **Perplexity** was used to evaluate each model.

Perplexity

The perplexity of a model is given as:

$$PP(Y) = \sqrt[N]{\prod_i^N \frac{1}{p(w_1, w_2, \ldots, w_N)}}$$

Unfortunately, the probability of each of the tokens were very low and complexity becomes Infinity duu to the fact the we will be multiplying thousands on very small number. The result of that is very close to 0. To solve this problem we use log probabilities instead. The new formula for perplexity becomes

$$PP*(Y) = e^{(\sum_i^N log(p(w_1,w_2,\ldots,w_N)))*-1/N}$$

where N is the number of ngrams in the document. $-1/N$ is a normalising factor. The log of a probability is a negative value so the minus helps normalise this. This gives us numbers that we can work with. Given a set of models, the model with the lowest perplexity is the "best" model. Without the minus in the normalising factor, we would search for the model with the highest perplexity.

The perplexity for the baseline language models are as follows.

Constant LM	0
Unigram LM	0
Bigram LM	0
Trigram LM	0

The perplexity of each of the models is 0 because some of the probabilities are 0. In the next section, we look at how we can improve this.

Improving Perplexity

Laplace Smoothing

As previously mentioned, some of our probabilities were 0 which made our resulting perplexity 0. In an effort to improve this, we introduce smoothing into our models. The idea is that we bump the count of each token by 1 and divide by the total number of tokens in our vocabulary. This is a monotonic operation and as a result, we do not loose any information. The results after smoothing are shown below.

Constant LM	394803.0002330778
Unigram LM	4789.669765542553
Bigram LM	9898.125430887805
Trigram LM	40670.40173817986

Improved Parsing

After a scan of some of the documents on our corpus, we noticed that the data is filled up with unicode characters and also tokens that aren't necessarily words. Also, the default sentence splitter provided did not account for tokens that are not words. All it did was split a sentence into tokens. The dataset was parsed and every token that does not comprise of just alphabets were removed. The result of this is shown below.

Constant LM	394802.99993733846
Unigram LM	1402.5505793473794
Bigram LM	761.13952916707340
Trigram LM	1458.3059158880415

From the table above we can see that there is a vital improvement in the models. The unigram model's perplexity has improved by over 3000PP, bigram model improved by over 9000PP and the trigram model improved by about 3000PP. Noticeably, the improvement to the constant is very little and almost negligible. This is because the constant model, as the name infers, has constant probability. This means that no matter what we do to the dataset, the perplexity will not improve as other models would.

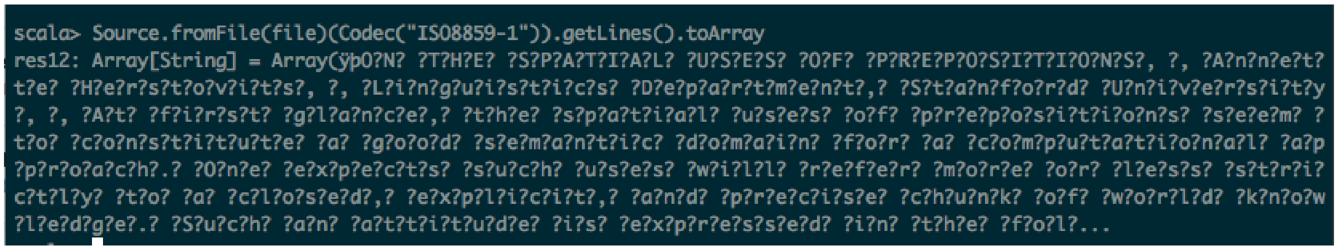
Further work

During experiments, it was observed that the perplexity of the models(except constant) improves as the training corpus gets larger. On the other hand, the perplexity of the constant model actually improves as the training corpus gets smaller. This is mainly because the constant probability increases as the size of the vocabulary decreases and smaller corpus means smaller probabilities hence the increase in perplexity. This goes to show how bad the constant model is and it should definitely not be used in practice. The table below shows the perplexity for the models when trained with less data(P79–P99).

Constant LM	284560.00001434644
Unigram LM	29796.91781597074
Bigram LM	9500.71333308270
Trigram LM	14836.34153286496

Most Probable Sentences

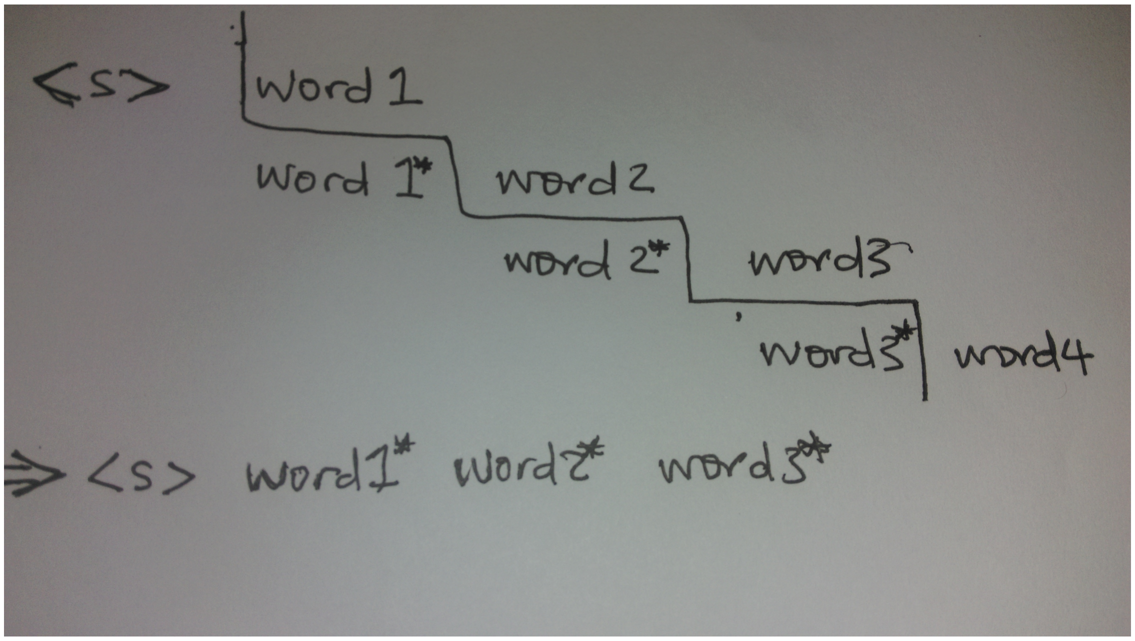
Unfortunately, the code suddenly started reading the input in a weird format, see image below. As I was unable to fix this problem, I could not generate the sentences, however, the code is in testLM(). Nonetheless, the intuition for generating the n most probable sentence is shown below.



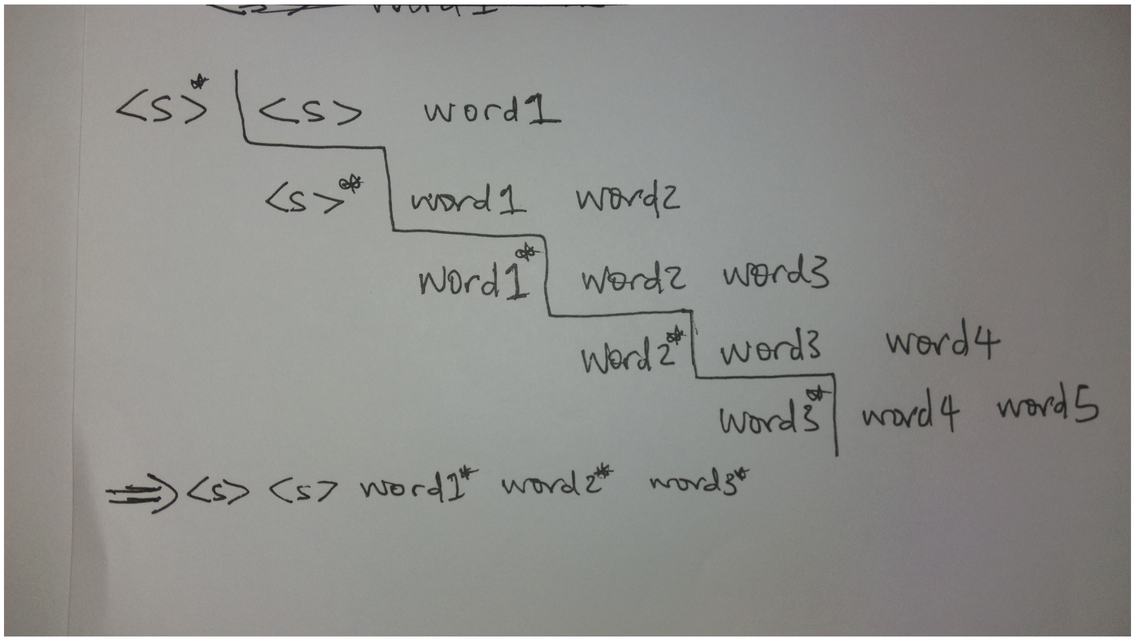
```
scala> Source.fromFile(file)(Codec("ISO8859-1")).getLines().toArray
res12: Array[String] = Array(ÿþO?N? ?T?H?E? ?S?P?A?T?I?A?L? ?U?S?E?S? ?O?F? ?P?R?E?P?O?S?I?T?I?O?N?S?, ?, ?A?n?n?e?t?
t?e? ?H?e?r?s?t?o?v?i?t?s?, ?, ?L?i?n?g?u?i?s?t?i?c?s? ?D?e?p?a?r?t?m?e?n?t?,? ?S?t?a?n?f?o?r?d? ?U?n?i?v?e?r?s?i?t?y
?, ?, ?A?t? ?f?i?r?s?t? ?g?l?a?n?c?e?,? ?t?h?e? ?s?p?a?t?i?a?l? ?u?s?e?s? ?o?f? ?p?r?e?p?o?s?i?t?i?o?n?s? ?s?e?e?m? ?
t?o? ?c?o?n?s?t?i?t?u?t?e? ?a? ?g?o?o?d? ?s?e?m?a?n?t?i?c? ?d?o?m?a?i?n? ?f?o?r? ?a? ?c?o?m?p?u?t?a?t?i?o?n?a?l? ?a?p
?p?r?o?a?c?h?.? ?O?n?e? ?e?x?p?e?c?t?s? ?s?u?c?h? ?u?s?e?s? ?w?i?l?l? ?r?e?f?e?r? ?m?o?r?e? ?o?r? ?l?e?s?s? ?s?t?r?i?
c?t?l?y? ?t?o? ?a? ?c?l?o?s?e?d?,? ?e?x?p?l?i?c?i?t?,? ?a?n?d? ?p?r?e?c?i?s?e? ?c?h?u?n?k? ?o?f? ?w?o?r?l?d? ?k?n?o?w
?l?e?d?g?e?.? ?S?u?c?h? ?a?n? ?a?t?t?i?t?u?d?e? ?i?s? ?e?x?p?r?e?s?s?e?d? ?i?n? ?t?h?e? ?f?o?l?...
```

For unigrams, this is very simple. We simply take a map of unigramsand their respective probabilities and sort. The top n word stringed together would represent the n most probable sentence for the unigram model.

For the bigram model, it get's alittle bit more complicated. First we find all the bigrams with format "$< s > word1$" where *word1* is any word. We then select from this list the token with the highest probability and save it. Next, we look for strings in the map with format "$word1word2$". Note that *word1* is the last token in the previous ngram. We keep doing this until we form a sentence of n. This sentence is out n most probable sentence using bigrams. The image below shows the intuition of bigrams and trigrams respectively.



<s>
word 1
word 1*
word2
word 2*
word3
word3*
word4
⇒ <s> word1* word2* word3*



<s>*
<s> word1
<s>*
word1 word2
word1*
word2 word3
word2*
word3 word4
word3*
word4 word5
⇒ <s> <s> word1* word2* word3*