# **CSEP 517, Fall 2015, Assignment 1**

<https://github.com/jogonzal-msft/UWNLPAssignment1>

## Problem1

### Problem1.Question1

The function **is not a valid probability distribution**. It lacks some sort of compensation/adjustment for the cases where P1 takes some of the values and P2 or P3 take the rest **(the sum of all probabilities gives us 2 in these cases**). Here is an example for clarity.

Corpora: “The dog is pretty. The dog is cool”

This table describes the P function that was called (P1, P2 or P3) for a given set of trigrams that **should sum one** (all trigrams possible given “the dog”). More formally:

P3 0.25 the dog {{STOPTOKEN}}

P3 0.25 the dog the

P3 0.25 the dog dog

P1 1 the dog is

P3 0.125 the dog pretty

P3 0.125 the dog cool

**The sum of all values in this table is 2, strongly hinting that there is some sort of missing compensation/discount in P2/P3.**

Note this is only one case where this function is not a valid probability distribution. Simply adding weighted values to P1, P2, and P3 will not suffice – the “weight” of their values has to be based on the number of bigrams that exist on each of the buckets (trigram combinations) – see below.

### Problem1.Question2

As mentioned in the previous answer, we need some sort of adjustment to prevent this “double counting” of probabilities. It is actually a very similar adjustment to the one made in Katz backoff.

We define a value β that will be subtracted from all counts, thus creating a new Pml function “Pml’”.

P1 will then be:

The total probability mass we take off P1 is defined by:

And this weight will be distributed along P2 and then P3. First we assign some of this weight to P2:

The probability mass you take off P2 can be assigned then to P3:

This leaves us with a well-defined function P. For this model, a compensation factor (β) of 0.75 was used.

This modification works because it extracts probability mass from P1 and gives it to P2. Similarly, it extracts probability mass from P2 and gives it to P3. Whenever the trigram does not exist in the training corpus, will be equal to 1 and P2 will take all the probability (when the trigram is in the set ). The same goes for P3 when the trigram or bigram are not present (when the trigram is in the set )

## Problem2

### Problem2.Question1

See the class Problem1Model.cs and the method “P” in it. Option 1 in the console app. Note: This takes a long time to execute, but results are shown below.

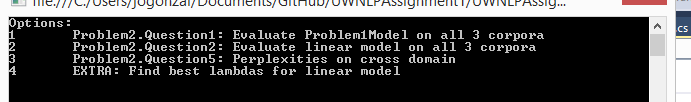
### Problem2.Question2

See the class LinearModel.cs and the method “P” in it. Option 2 in the console app.

### Problem2.Question3

Run application (console application) and choose option 1 or 2.

\*NOTE\*: Running the Problem1Model (option 1) takes a long time to run in my machine (all 3 corpora)



***Choosing UNKs for perplexity calculation***

After a few comparisons between the development part of the corpus and the training corpus, I decided to use mark 9% of words that appear only once in the training corpus as UNKNOWNs. This approach was used on both models on all corpora in order to be able to compare the Problem1Model and the linear model complexities across corpora.

|  |  |  |
| --- | --- | --- |
|  | Problem1Model | LinearModel |
| Brown | 123.1152 | 131.0975 |
| Gutenberg | 279.3173 | 234.2599 |
| Reuters | 57.1245 | 69.5992 |

### Problem2.Question4

See option 3 in the console app.

I chose the values **0.1, 0.5 and 0.4**. (Trigram, Bigram, Unigram) and found them using grid search. The way I found them was by testing several values and computing the perplexity for each of them (grid search). I computed a set of possible Lambda values and ran a program that determined which of those would have the lowest perplexity for all 3 corpora. Here is the set of possible values I considered.

private static readonly List<Tuple<double, double, double>> Possibilities = new List<Tuple<double, double, double>>

{

new Tuple<double, double, double>(0.3, 0.3, 0.4),

new Tuple<double, double, double>(0.4, 0.3, 0.3),

new Tuple<double, double, double>(0.3, 0.4, 0.3),

new Tuple<double, double, double>(0.2, 0.4, 0.4),

new Tuple<double, double, double>(0.4, 0.4, 0.2),

new Tuple<double, double, double>(0.2, 0.4, 0.4),

new Tuple<double, double, double>(0.1, 0.5, 0.4),

new Tuple<double, double, double>(0.1, 0.4, 0.5),

new Tuple<double, double, double>(0.5, 0.4, 0.1),

new Tuple<double, double, double>(0.6, 0.2, 0.2),

new Tuple<double, double, double>(0.2, 0.6, 0.2),

new Tuple<double, double, double>(0.2, 0.2, 0.6),

new Tuple<double, double, double>(0.8, 0.1, 0.1),

new Tuple<double, double, double>(0.1, 0.8, 0.1),

new Tuple<double, double, double>(0.1, 0.1, 0.8),

};

### Problem2.Question5

#### Problem2.Question5.PointA

See option 4 in the console app.

|  |  |
| --- | --- |
| Test | Perplexity |
| Train Reuters and test on Brown | 278.3774 |
| Train on Brown and test on Gutenberg | 343.4650 |
| Train on Gutenberg and test on Reuters | 560.9243 |

#### Problem2.Question5.PointB

Noticeably, the language models perform worse when their perplexities are evaluated outside of their domain.

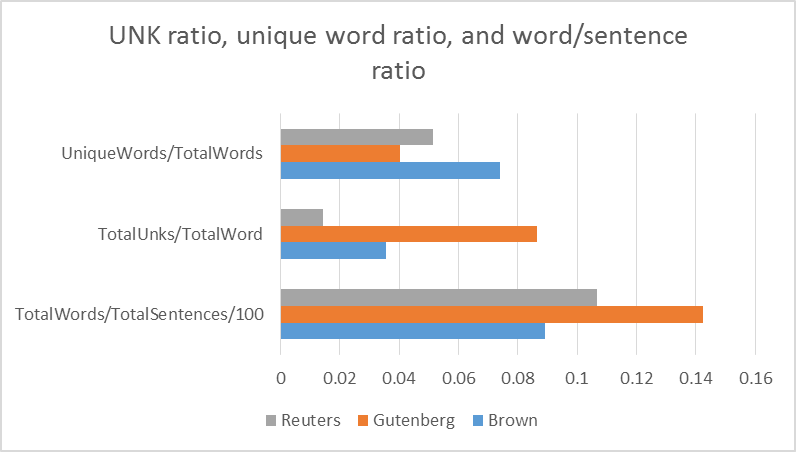
One very noticeable case is the perplexity when testing against Reuters, where the Reuters model did pretty well (~70), but the Gutenberg model did poorly (~560). This tells us that the language used in Reuters and Gutemberg isn’t so similar. Knowing where the text come from - Reuters is an international news agency, Gutemberg is a collection of free ebooks, novels and all sorts of publications. Another thing that shows us that we won’t do well on perplexity is the similarities between the corpora in terms of % of UNKs per word, Words/Sentence and total number of unique words (see chart below), and the top bigrams where they don’ Gutenberg has the most unknowns with the longest sentences, while Reuters and Brown have lower amounts.

Another thing to notice is that the Reuters model didn’t do badly when tested on Brown – the contents of Brown are similar in terms of corpora characteristics (see chart/table) below. In addition, the content in Brown is designed to be a “standard corpus for American English” which seems to be more compatible with the Reuters domain. We can see a similar pattern by taking a look at the top bigrams.

In conclusion, the similarities of the corpora used to train the model and the test corpus (UNK ratio, unique word ratio, vocabulary), including the language that is being used in them, will determine the calculated perplexity. The more similar the language + characteristics of the corpora, the lower the perplexity.

Comparison of some of the top bigrams for Brown, Gutenberg, Reuters.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Brown | | Gutenberg | | Reuters | |
| of | the | , | and | ' | s |
| {{STARTTOKEN}} | the | of | the | lt | ; |
| in | the | in | the | & | lt |
| , | and | and | the | {{STARTTOKEN}} | the |
| , | the | - | - | said | {{STOPTOKEN}} |
| to | the | {{STARTTOKEN}} | " | of | the |
| - | - | ? | {{STOPTOKEN}} | in | the |
| ; | {{STOPTOKEN}} | the | lord | {{STARTTOKEN}} | s |



## BONUS

See option 5 in the console app.

An approach for achieving this could be to add the target domain data with more weight (e.g. count each trigram/bigram/unigram found in the target domain X times instead of just once). That way we’ll be assigning more weight to it. Using X of 10 and 10% of the target domain (not in the test corpus), here are the new performances. This approach works better as you add more target domain data and you increase its weight.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | CrossDomainPerplexity | ImprovedCrossDomain | OriginalPerplexity |
| Train Reuters and test on Brown | 278.3774 | 136.87 | 131.0975 |
| Train on Brown and test on Gutenberg | 343.4650 | 380.45 | 234.2599 |
| Train on Gutenberg and test on Reuters | 560.9243 | 178.96 | 69.5992 |