UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

NLP: Lab 1

Entropy and Language Models

Alejandro Hernandez, Javier Fernandez | Natural Language Processing, MAI | October 30th 2014

Table of Contents

Introduction

The present report explains how the tasks given for the first laboratory project were accomplished. In each sections each task is described by presenting the *Python* methods that compute each solution respectively and then the obtained results are discussed.

Running the project

To run the code for this project execute “python entropy.py” command or run “entropy.py” from a python interpreting environment. The script will start calculating and printing the results of each requested task in the order they were presented in the lab description file.

Task 1: Compute Unigram Entropy

The unigram entropy of the English corpus is: **11.23972528**

The function written for this task was:

```
compute_unigram_entropy(ngram_prob)
```

This function takes the unigram probability previously estimated and calculates the entropy by the summation given by the formula.

Task 2: Compute Bigram Entropy

The bigram entropy of the English corpus is: **5.9275724532**

The function written for this task was:

```
compute_bigram_entropy(unigram_probs, unigram_freqs, bigram_freqs)
```

The parameters of this function are the previously computed probabilities for the unigram and the frequency or counts of both the unigram and bigram.

This function estimates bigram probabilities using the frequentist approach, and performs the logarithm and summation of the entropy formula.

Task 3: Compute Trigram Entropy

The trigram entropy of the English corpus is: **2.00193747516**

The function written for this task was:

```
compute_trigram_entropy(unigram_probs, unigram_freqs, bigram_freqs, trigram_freqs)
```

This function is similar to the bigram computation described in the task 2, but adding the trigram model estimation and expanding the entropy formula to include the trigrams.

Task 4.1: Compute Entropies for ES Corpus

The entropies of the es corpus are:

unigram: **10.8472882621** bigram:**5.62341537324** trigram:**1.920375897**

The processing of the corpus into the n-gram models was wrapped in the function:

```
process_corpus(file)
```

This function receives the file name as a parameter; then loads it and perform all n-gram model computations. This allows for better reusability and maintenance of the code.

In the end of *entropy.py* this function is called with both, "en" corpus and "es" corpus.

Task 4.2: Comparison of ES and EN Corpus

The following table presents the results obtained for the entropy of the unigram, bigram and trigram calculated on the English (EN) and Spanish (ES) corpus.

N-Gram	EN Corpus	ES Corpus
<x> Unigram	11.23972528	10.8472882621
<x,y> Bigram	5.9275724532	5.62341537324
<x,y,z> Trigram	2.00193747516	1.920375897

From the results of the table above it can be observed how, for both corpora, entropy values decrease while the size of the unigram increases, particularly by taking into account just unigram, bigram and trigram. This is consistent with the idea that single words are providing more information (and then more complexity on their own) than a two-word sequence (bigram) and three-word sequence successively. This is also supported by the idea that the probabilities of finding equal trigrams are lower than bigrams, and lower than unigrams respectively. Being said in different words, higher n-grams (taking this unigram, bigram, trigram example) provide a language that is more specific and less variable, thus being less entropic (more certain).

Another key result observed is that the obtained entropies for the Spanish corpus are slightly lower than the entropies of English Corpus. This is contrary to the expected results, where Spanish language was expected to have higher entropy values given that is considered a language with higher complexity than English. However, this result is assumed to be derived from the sizes of the analyzed corpora.

Looking closely to the sizes of both corpus it can be observed that English corpus have 926761 words while Spanish corpus just 482373, few words over the half of the previous one. To back this assumptions we calculated the three n-grams for the half of the English corpus (463380 words). The table below shows the obtained results:

N-Gram	Half of EN Corpus	ES Corpus
<x> Unigram	11.1500606656	10.8472882621
<x,y> Bigram	5.53532605343	5.62341537324
<x,y,z> Trigram	1.67385819077	1.920375897

It can be observed how the entropy of the English corpora decreased consistently for three n-gram cases when reducing the size of the corpus. With a similar number of words Spanish corpus shows a slightly higher entropy for unigram and bigram models, and a bigger difference from the trigram model.

Task 5: Perplexity of the Trigram Model

The perplexity of the trigram model is: **2.6715688207**

This number was calculated by the very simple function:

```
perplexity(entropy)
```

This function only implements the formula of the perplexity, given a precomputed entropy.

However this function is wrapped by:

```
compute_trigram_perplexity(words, size, smooth=0, tagged_words={})
```

This function performs all the n-gram model computations, plus the entropy calculation in the end. Also, allows for smoothing, at different levels, and given a dictionary of tagged words.

Task 6: Perplexity of the Smoothed Model

Perplexities	Full corpus	½ corpus	¼ corpus
<x,y,z>	2.6715688207	2.23958100674	1.96258705019
<x',y,z>	11.2562878271	8.2622337167	6.23385294839
<x',y',z>	115.958036214	81.9338395609	54.9135868089

This perplexities contradict the common sense by going up when smoothed, it was expected that they go down the more smoothing is performed; however this might be the result of a corpus too small or shallow to be analyzed with the trigram model, and in fact the trigram entropy of the corpus without smoothing is quite low: 1.41768718205 , and this means that the corpus has many unique trigrams, and thus, there is very little information retained by this model. So given this, for this example adding POS-tags is increasing the information of the whole model by making the next combination to occur with highest probability in the pos-tagged n-gram. This is: given that for <x,y,z> there are many unique trigrams, pos-tagging a part of the trigram <x',y',z> is incrementing the amount of occurrences of the trigram itself (and also the occurrences of <x',y'>) thus incrementing the entropy (and the perplexity).

It can also be observed how perplexities for smaller sizes of the corpus are consistently lower. This is consistent with the idea that having lower amount of words and combinations leads to a less uncertain and then less entropic corpus; deriving in a lower perplexity.

Moreover, if the perplexity of the fully pos-tagged trigram throw the following results:

Perplexities	Full corpus	½ corpus	¼ corpus
<x',y',z'>	8.48268886721	8.02097548522	7.51422754792

For this fully pos-tagged trigram the perplexity is considerably lower than for the partially tagged trigrams and consistent in the different sizes of the corpus. Also is higher than the perplexity of the non pos-tagged trigram <x,y,z>. This shows the fully pos-tagged trigram is incrementing the

occurrences of the different trigrams, thus having less unique trigrams and incrementing entropy in comparison to <x,y,z>. At the same time there are considerably less different combinations in a fully pos-tagged corpus, having more frequent occurrences of the same combinations, so entropy does not grow as much as for the previous cases of partially pos-tagged trigrams.