FNLP: Lab Session 6

Pointwise Mutual Information - Finding Collocations

Aim

The aims of this lab session are to 1) familiarize the students with pointwise mutual information (PMI) 2) show how to apply PMI for the task of finding word collocations 3) identify shortcomings of this approach . By the end of this lab session, you should be able to:

- Compute the PMI.
- Apply PMI to find word collocations in a corpus.

1 Running NLTK and Python Help

1.1 Running NLTK

NLTK is a Python module, and therefore must be run from within Python. To get started on DICE, type the following in a terminal window:

```
...: python
>>> import nltk
```

1.2 Python Help

Python contains an inbuilt help module that runs in an interactive mode. To run the interactive help, type:

```
>>> help()
```

To exit, press CTRL-d.

If you know the name of the module that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>)
```

To exit, type "q" (for "quit").

If you know the name of the module **and** the method that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>.<method_name>)
```

To exit, type "q" (for "quit").

2 Introduction

Before continuing with this lab sheet, please download a copy of the lab template (**lab5.py**) for this lab from the FNLP course website. This template contains code that you should use as a starting point when attempting the exercises for this lab.

In this lab we consider the task of identifying word collocations in a corpus to demonstrate the use of Pointwise Mutual Information (PMI).

$PMI(x_i, y_j) = \log \frac{P(X=x_i, Y=y_j)}{P(X=x_i)P(Y=y_j)}$

The data set consists of bigrams extracted from the Moby Dick novel.

```
>>> sentences=gutenberg.sents('melville-moby_dick.txt')
```

3 Estimating the Joint and Marginal Probability Distributions

In order to compute the PMI we need the joint probability $P(X = x, Y = y)$ and the marginal probabilities $P(X = x)$ and $P(Y = y)$. In our case $P(X = x)$ and $P(Y = y)$ will be the unigram probabilities of the two words that are considered to form a collocation, and $P(X = x, Y = y)$ will be the bigram probability.

Exercise 1:

In this exercise we will compute the joint and marginal probability distributions for the word bigrams. You will have to fill in two functions to achieve this:

- The function `BuildData` receives as parameters a list of sentences and the name of a Filter function. Two Filter functions are already defined, one eliminates just non-alphanumeric tokens, the other eliminates stop-words as well.
- The helper function `ex1` should return a list of bigrams and a list of unigrams extracted from the sentences.

Specifically:

a. Build the two data structures in the `BuildData` function. Lowercase the words and eliminate unigrams and bigrams that do not pass the filter.

   Remember, `help` is your friend:

   ```
   >>> help(nltk.bigrams)
   ```

b. The function `ex1` receives as parameters a list of bigrams and a list of unigrams and returns the corresponding probability distributions. Construct a `FreqDist` for each of the two lists. Transform each `FreqDist` into a probability distribution using the `FasterMLEProbDist` estimator.

   Again, `help` is your friend:

   ```
   >>> help(nltk.probability.FasterMLEProbDist)
   ```

Now uncomment the test code. Compare the top 30 most frequent bigrams arising from the two different filters.

Most of the former are made up of **closed-class** words. If we eliminate stopwords some interesting bigrams over content words show up.

4 Computing the Pointwise Mutual Information

In the previous section we estimated the joint and marginal probability distributions from the data. In this section we use these distributions to compute the PMI for a given sample. In order to avoid multiplication of small floating point numbers (probabilities), we can rewrite the formula for PMI as:

$PMI(x_i, y_j) = \log P(x_i, y_j) - (\log P(x_i) + \log P(y_j))$

Exercise 2:

The template of the function that you have to implement takes two parameters: the bigram and unigram probability distributions.

a. Create a list of pairs of samples in the distribution and its PMI, using the `logprob()` function of the two `FasterMLEProbDist` instances.

b. Make a dictionary from that list

c. Sort the list of pairs.

Uncomment the test code. Look at the PMI of some bigrams. We can see that the PMI for "sperm whale" is 5 binary orders of magnitude greater than the PMI of "of the". Can you think of some reasons why the PMI for "old man" is not as low as we would expect?

What can you observe by looking at the top 10 bigrams according to the PMI? How do low counts affect the PMI?

Exercise 3.

In the previous exercise we found that the PMI is very sensitive to data sparsity. Bigrams composed of low frequency words are ranked higher than bigrams with high frequency words according to PMI. One way to fix this issue is by putting a threshold for the frequency of words.

Edit the `ex3` function to do this:

a. Filter the full list of bigrams and their corresponding PMI to include only bigrams with frequency greater than 30.

What does a negative score say about a bigram?

Optionally you can eliminate stop-words from the corpus by applying the second Filter function, then recompute the PMI and investigate the top and last bigrams.