COM6115: Lab Class 5B

Text Preprocessing

Text preprocessing is composed of a set of subtasks in order to get a text ready to be analysed by machines. For example:

- How can a program distinguish between “LOVe” and “love”?
- How can a program identify that “food” and “food.” are the same word?
- How can a program identify that “I’m” corresponds to “I am”?
- How can a program recognise that the punctuation in “This is a lovely night.” is different from the decimal point in “The book costs 3.45 pounds”?

The preprocessing steps which are appropriate in a particular scenario will depend on the task. For example, converting everything to lower case can be useful in many cases, but could be harmful for (English) named entity recognition.

This lab will give you a chance to apply some preprocessing techniques to a dataset of Tweets about US airline companies, and to familiarise yourself with some of the functionality of the Natural Language Toolkit (NLTK) library.

Exercises

Get the data

1. Download and extract the zipped data files from the module homepage.

`Tweets_short.csv` is a dataset containing tweets about US airline companies. This dataset is a comma-separated file. Each sample is composed of a **tweet id** (first column), followed by the **sentiment** and the **text** of the tweet. You will only be using the column with the text for now.

Tokenisation using Regular Expressions

2. Use the concepts you learned in the previous lab (Regular Expressions) to write some basic code for **tokenising** and **lowercasing** a piece of text. You will need to use the concept of **substitution** in order to manipulate the text appropriately.
3. Test your code with the following sentences before applying to the entire dataset:

```
He says I'm depressed most of the time.
For the first time I get to see @username actually being hateful! it was beautiful:)
"The San Francisco-based restaurant," they said, "doesn’t charge $10.5"
```

4. Compare the results of the above with your fellow students. Did you do anything different? In particular, how did you treat the following?:

- I’m
- @username
- :)
- doesn’t
- $10.5

Tokenisation with the help of NLTK

Fortunately, Python has some functions and libraries that can help with the task of tokenisation and lowercasing. NLTK toolkit, in particular, has many options for tokenisation.

6. The task is to implement the tokeniser based on regular expressions as described on pages 16-17, Chapter 2 of Dan Jurafsky and James Martin’s book on Speech and Language Processing (https://web.stanford.edu/~jurafsky/slp3/2.pdf).

This needs to be adapted slightly to work as described in the book. Specifically, capturing groups need to be converted to non-capturing groups (e.g. `([A-Z]\.)+ -> (?:[A-Z]\.)+)`, as shown below:

```
pattern = r'''(?x)          # set flag to allow verbose regexps
       (?:[A-Z]\.)+         # abbreviations, e.g. U.S.A.
     | \w+(?:-\w+)*         # words with optional internal hyphens
     | \$?\d+(?:\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
     | \.\.\.               # ellipsis
     | [][.,;"'?():_`-]     # these are separate tokens; includes ], [
   '''
```

7. Compare how the outputs differ from your own tokeniser.

Tokenisation with even more help from NLTK

NLTK has a built-in function that can perform tokenisation directly, without the need for explicitly defining regular expressions.

8. Have a look at how this is done in Chapter 3 of the Steven Bird, Ewan Klein, and Edward Loper book on Natural Language Processing with Python (https://www.nltk.org/book_1ed/ch03.html).

Preprocessing Tweets

Tweets (and other types of user-generated content) usually require extra preprocessing beyond tokenisation and lowercasing. For instance, it may be useful to normalise @mentions, hashtags and emoticons/emojis.

9. Your task is to replace all @mentions, hashtags and text-based emoticons in the US airline companies dataset with: `<MENTIONS>`, `<HASHTAGS>` and `<EMOTICONS>`. For example:

```
@SouthwestAir → <MENTIONS>
@united → <MENTIONS>
#WorstFlightEver → <HASHTAGS>
:) → <EMOTICONS>
```

OPTIONAL: For an additional challenge, you can also attempt to replace pictographic **emojis** with an `<EMOJIS>` tag. Note that to do this using a regular expression, you will need to match the Unicode codes which correspond to each emoji.

Counting words

One important statistic about textual data is the number of words. However, this is not always a direct count of the number of tokens in our document. We usually distinguish between the number of **types** (number of unique words) and the number of **tokens** (all words in a document). Number of types is usually the size of our vocabulary, i.e. how many unique words a given corpus includes. Imagine that we train a system on this set of text, we can then say that the system's vocabulary are the words that the system knows.

Type-token ratio (**TTR**) is a useful statistic to measure the lexical complexity of a corpus. A high TTR means a high lexical diversity (high number of unique words), while a low TTR means the opposite.

10. In this activity you will calculate the TTR for the US airline companies corpus, considering the following cases:

    (a) Entire data without any preprocessing (using the `.split(" ")` method to separate tokens).

    (b) Tokenised data (you can use ANY of the methods above)

    (c) Tokenisation + lowercase

    (d) Tokenisation + lowercase + tweet preprocessing

    Important: You should not consider punctuation and other special characters as words.

11. How does the TTR change in each configuration? Why does this happen, and what do you think could be the impact on downstream tasks (e.g. sentiment analysis)?

Extra activity

You can start developing methods that can calculate the frequency of words (which will later be useful for calculating likelihoods).

Calculate the frequency of each word in the US airlines dataset. Word frequency can be defined as:

$$freq(w_i) = \frac{count(w_i)}{\sum_0^n count(w_i)}$$

where n is the total number of unique words, w_i is the i-th word and count is a function that counts the number of times a word appears in a corpus (you should recognise this as tf(t, d) from information retrieval).

Check what is the frequency of the preprocessed tags `<HASHTAGS>`, `<MENTIONS>` and `<EMOTICONS>`.