

COM6115: Lab Class 3

Naive Part-of-Speech Tagging

This week's lab involves counting over multiple labels/terms, which is a common task in statistical text processing. A key aim, however, is for you to practise the use of Python dictionaries; in particular, **two-level** dictionary structures.

In this lab, we will be looking at some [part-of-speech \(POS\)](#) tagged data, extracting some distributional information from it, and then using counts of word/POS tag co-occurrence as a basis for constructing a simple POS tagger which could be applied to new data.

Exercises

1 Get the data

1. Download the files `POSTAG_DATA.zip` and `postagger_STARTER_CODE.py` from the module homepage.

Unzipping `POSTAG_DATA.zip` gives two files of POS tagged text, for use in training and testing your simple POS tagger.

2. Open one of the data files to see the data format (known as *Brill format*), which is a convenient format for processing in which the text is given with one sentence per line, and is already tokenised, with a single space between each token. Each token is given together with its POS tag, in the form `TOKEN/POS`. You will see that some items of punctuation are treated as separate tokens in this tokenisation scheme.

This data is drawn from the *Penn Treebank* (PTB), and consists of *Wall Street Journal* newspaper text. The POS tags provided are from the PTB tagset.

Please note: this data is available only under strictly licensed conditions. You should not retain it after your involvement in this course, or pass it on to anyone else.

2 Counting POS tags in the training data

1. Extend the starter code provided to give a script that will process the training data file, splitting each sentence into its tokens, and counting the co-occurrence of tokens and POS tags. Take the tokens as given in the data file (i.e. do *not* attempt to normalise them, e.g. for capitalisation).

Your data structure should be a two-level dictionary, which maps from tokens to POS tags to counts (i.e. $\{\text{term} \mapsto \{\text{postag} \mapsto \text{count}\}\}$), where count is the number of times that the token has appeared with that POS tag. We shall call this dictionary the term-postag-count dictionary.

2. By processing over the term-postag-count dictionary, determine answers to the following questions, which your code should print to the screen.
 - (a) Determine the full set of POS tags used in the data, and their overall relative frequencies. Print them out in descending order of relative frequency. What is the single most common tag? What accuracy score would be achieved by an extremely simplistic tagging method that just assigned this one tag to every word?
 - (b) What proportion of the items ('types') in your lexicon are **ambiguous**, i.e. have more than one POS tag? What proportion of the overall token occurrences in the training data correspond to lexicon items that are **unambiguous**?

3 Building a POS tagger

We are going to consider a naive approach to POS tagging which simply assigns to each token its own **most common** POS tag, as found in the training data. This simple approach works surprisingly well for English.

1. Begin by determining how well this approach would work over the training data itself, which you can do by directly processing over the term-postag-count dictionary.

In particular, compute an **accuracy score**, i.e. indicating what proportion (or percentage) of tokens would be correctly tagged by this method.
2. Next, apply the naive tagging approach to the test data, using the counts derived from the training data.

You will face the complication that the test data will contain so-called **unknown words**, i.e. words that were not seen in the training data. Begin by assuming that all unknown words are tagged incorrectly, by assigning them a non-PTB-tag symbol, such as UNK, as their POS tag.

Determine the tagging accuracy this method achieves over the test data.

Note: the test data is supplied in the same format as the training data, i.e. with gold standard POS tag attached to each token. So, you need to separate the token and tag in

each TOKEN/POS pair, apply the naive tagger to the token and then compare with the gold standard POS tag, computing accuracy as you go.

3. A key factor in how well real world tagging systems perform is their handling of unknown words. Explore alternative ways of handling the unknown words in the test data here, in an effort to achieve the best accuracy you can for your naive tagger.

An obvious first move is to assign each unknown word the most common tag overall (which you should have determined above).

You might also consider if there are any subclasses of terms that can easily be recognised for the assignment of a different default tag; e.g. think about how you might identify numbers, names, etc.

If you have time, you could explore using word-ending suffixes as a cue to predict the POS of unknown words. For example, words ending in -ing in English are commonly verbs (with POS VBG), whilst words ending in -ly are commonly adverbials (RB).

4. If you speak any languages other than English, think about whether you believe this approach would work as well for them. Are there any modifications you would consider making?
5. If you want to compare the performance of this naive tagger against other methods, the [NLTK library](#) comes with several POS taggers. You'll need to convert the text back into a plain format to pass into NLTK.