Peter Nalyvayko
np2240@columbia.edu
COMS W4115 Programming Languages and Translators
Stephen A. Edwards

Part-Of-Speech (POS) Tagger Language

POSTaL

Here is a proposal for a domain specific language offering lexical and syntactical constructs to allow defining and implementing various part-of-speech tagging algorithms.
The POS tagging is a process to assign a lexical class to the words in a text. The task of part-speech tagging closely resembles the process of tokenizing a general purpose programming language. However, the tagging of a natural language is much more difficult due to its intrinsic ambiguity as well as practically infinite number of permutations. The tagging is extremely important to the NLP, and there exist already a number of taggers, such as stochastic taggers, rule-based taggers and transformation based tagging. Majority of taggers, be that stochastic, rule based or transformation based, are, in fact, supervised machine learning algorithms and require a training corpus, i.e. a pre-tagged corpora by either humans, or by means of bootstrapping. Existing corpuses are Penn Treebank and Brown corpus.
In addition to the pre-tagged corpus, the taggers rely on various linguistic databases to improve the precision. Among such databases are FrameNet and WordNet. The WordNet project has been developed and maintained by the Princeton University and is a large lexical database populated by the professional linguists. The information is organized into a network of interconnected nodes, called synsets, eg. synomous sets, linked by a mesh of lexical and semantic relationships.

The part-of-speech tagging goes like that: given an untagged input, the tagger produces a tagged output which contains the original data labeled with one or more POS tags.

The proposed language shall support types and operations commonly used in the field of part-of-speech tagging. The features may include but not limited to conditional probability computations (e.g. P (tag | word)), end-of-sentence detection, annotations, list operations (sorting, selecting, adding or deleting of the elements), as well as basic math operations such as addition, division, logarithmic and trigonometric functions. There are also input/output operations to load data from a disk or store it back (e.g. loadfile and savefile).

The language is type-insensitive. There just two types: an atom and a list. Atom refers to a variable of any type and the list represents a linked collection. There are built-in functions such as “next”, “prev”, “sort”, etc. to manipulate the lists and their elements.

The example (the syntactical structures are not final and may change during the development).

```
#tagged_corpus browncorpus
“c:\trainingdata\BrownCorpus.xml”

/*
White space will be stripped off and the string will be
tokenized to a linked list of tokens.
*/
var sentence = list (“a red fox jumped over the fence”);
/* STEP 1. Assign most likely tag using the tagged corpus.
“Corpus” is a reserved word initialized by the
#tagged_corpus pragma; subject to change.
*/

/* The following fragment iterates the elements of the list
and for each element assigns it a best POS tag using the
given corpus data. Corpus (training data) is also a list as
far as the language is concerned.
*/
var p;
var temp = sentence;
while (valid(temp)) {
     var p = next(temp);
     attribute(p, “tag”) = first(sort(
          select(browncorpus, lambda(x) { return
     attribute(x, “value”) = attribute(p, “value”); }),
           lambda (x, y) { return attribute(x, “count”) >
     attribute(y, “count”); }));
}
/* Prints all elements of the sentences and their
associated attributes
*/
print (sentence, lambda (x) { print(x); });
```

“Lambda” notation declares an anonymous function which is used as a predicate, an action or a comparator to perform tasks on the elements of the list. Lambdas are useful when sorting the elements of the list of selecting a subset of the collection.