THEMES IN SCIENCE AND TECHNOLOGY EDUCATION
Volume 3, Number 1-2, Pages 81-92

A tagger for teaching purposes

Paul Bennett
Paul.Bennett@manchester.ac.uk

School of Languages, Linguistics and Cultures, University of Manchester, UK

Abstract

This paper describes a software system intended to help students learn about the process of tagging, the assignment of part-of-speech labels, which is basic to computational linguistics. It does this by (a) making it transparent why words have been analysed in a particular way, and (b) making it straightforward to add items to the system's lexicon and improve its grammatical coverage. An unforeseen and welcome side-effect is that users are led to reflect on various properties of English grammar.

Introduction

Tagging involves the assignment of part-of-speech labels or tags to tokens in a text. It is a basic computational linguistic technique, and one that it is important for students to understand, preferably in practical terms rather than just by reading about it. For a textbook presentation, see Jurafsky and Martin (2009, ch. 5), and for an overview, see Voutilainen (2003). Tagging has various applications, including disambiguation for text-to-speech systems, parsing (Dalrymple, 2006) and morphological analysis (Minnen, Carroll, & Pearce, 2001), and the provision of tagged corpora to enable more useful corpus investigations. The tags are usually more specific than just 'noun' or 'verb', and will normally include categories such as 'singular noun' or 'past tense verb'. The set of labels referred to in this paper is the C5 tagset (Garside, Leech, & McEnery, 1997, pp. 256–7), which is widely used for English. See the Appendix for an explanation of the tags mentioned in this paper.

Various taggers can be downloaded or run over the web, so students can gain a good insight into the output that taggers produce and into which constructions create

problems. However, it is often unclear just why a word was analysed the way it has been. For instance, the Claws4 tagger at the University of Lancaster (Leech, 1997) can be run online in such a way that it will inform the user how certain it is that a word will get one tag rather than another, but how such decisions are arrived at remains unclear. In general, stochastic taggers will not make their working clear to the non-expert user. Furthermore, even taggers that can be downloaded, such as the TreeTagger (Schmid, 1997), only allow limited extension by the user, such as adding new lexical entries. Trainable taggers (Brill, 1995) are often too complex for novice users (especially linguistics students with little computing background) to cope with. Of course it is to be expected that systems intended for real tagging tasks should be complex and not invite extension by neophytes.

There are also taggers intended to aid in, or be part of, the learning of programming for computational linguistics, but these have a slightly different aim. For instance, Mason (2000, ch. 10) provides Java code for a statistical tagger that includes a lexicon and a suffix analyser, but the emphasis is on improving the tagger by exploiting the reader's knowledge of the programming language rather than the more linguistic focus of our system. The taggers that form part of the Natural Language Toolkit (Bird, Klein, & Loper, 2009) are instructive in many ways, but again the emphasis is on programming (this time, in Python) and the use of such ideas as backoff tagging; linguistics students find these very hard to cope with.

The motivation behind this work, then, was to write a tagger which clarified why a word received a particular tag and which could be extended by the user in a fairly straightforward way; I am not aware of any other work along precisely these lines. The tagger is known as EPT (for Expandable Pedagogic Tagger). It is written in Perl, and has no further software requirements. EPT was written with the task of tagging English in mind, but the actual tagging process has little that is English-specific, and extending it to other languages would only (!) require new lexica and rules. The system is aimed at those who have no knowledge of programming and are just starting to learn about computational linguistics. It should be made clear that EPT is not intended to teach anything about the workings of stochastic taggers.

The Output of EPT

EPT shows how a word was analysed by using different kinds of 'links' between word and tag in its output:

(1) My/DPS neighbour/*NN1 Bob/!NP0 annoyed//VVD me/PNP ./PUN

The '/' shows that *me*, for instance, is in the system's lexicon and is assigned a tag by consulting that entry. Words with other links are not in the lexicon and are given a tag in other ways. With *Bob*, the '/!' shows that the proper noun tag was assigned because of the initial capital letter, while in the case of *annoyed* the '//' shows that the system has made use of an affix (here *-ed*) to assign a tag. The '/*' on *neighbour* means that this word has been treated as an unknown word and assigned a tag on this basis. This exploitation of different kinds of link is taken from an earlier online version of the Memory-Based Shallow Parser (Daelemans & Bosch, 2005), where '//' was used for words not in the system's lexicon.

EPT can be run in verbose mode, which means that during the process of tagging it will issue messages about how some words have been tagged, such as:

(2) a. Using suffix -ed for annoyed

b. Using unknown word tag for neighbour

As will be seen, verbose mode will also provide other information about the workings of the tagger.

How EPT Works

Like most taggers, EPT involves the following stages: tokenisation; lexical look-up; treatment of unknown words; disambiguation. We now explain these in more detail.

Tokenisation

EPT begins by tokenising the input. This essentially involves just separating out punctuation and performing modifications such as *isn't* to *is* + *n't* and *John's* to *John* + *'s* (Leech, 1997). Forms such as *o'clock* and *O'Reilly* are given special treatment so that they will be treated as single tokens. The tokenisation part of EPT is adapted from a program downloaded from Melamed (1996).

It is best to enter single sentences as input to EPT. This partly for the practical reason of not being overwhelmed by the amount of output in verbose mode, and also because it does not identify sentence boundaries. It has some special statements for the first word of a sentence but takes this as simply being the first word of the input.

Lexical Look-up

If a token consists entirely of digits, it is assigned the tag CRD. All other tokens are looked up in the lexicon, which contains entries such as:

(3) a. and CJC

b. best AJS

c. friend NN1

d. might VM0-NN1

e. . PUN

Punctuation marks are listed in the lexicon (cf. (3e)). We shall see later what happens when there is more than one tag given for a word, as in (3d). Adding entries to a lexicon structured like (3) is fairly straightforward. EPT checks that user-supplied tags are in the C5 tagset, in order to avoid typos and other errors.

One difference from CLAWS is that EPT does not posit multiwords, whereby a form such as *in spite of* is tagged as a single preposition (Leech, 1997, p. 21). In addition, 'subordinating conjunctions' are not regarded as a separate category from prepositions, so that *before* is tagged `PRP` in both these examples:

(4) a. John left before the end

b. John left before the play ended

However, the collapse of this distinction is not hardwired into EPT, and the user could re-introduce it if so desired.

If the first word of the input is not in the lexicon, the system treats it as if it were all lower case. So *My* in (1) is treated as if it were *my*. This will create problems if the first word could but need not be a proper noun (e.g. *Brown*, *Wall*). EPT does not work well with input including text that is in all capitals or all initial capitals, and users are advised not to use text formatted in this way.

Unknown Words

Given the size of EPT's lexicon, the processing of unknown words is more crucial than in taggers with realistically-sized lexica. For any token not in the lexicon, various possibilities exist. If the word is hyphenated, the part after the hyphen is tagged; thus *mile-high* is tagged as an adjective because *high* is listed as such in the lexicon. If the word has an initial capital, the tag `NP0` for proper noun is assigned. Also morphological information is used; a suffix lexicon contains entries such as:

(5) a. ous AJ0

b. ess NN1

c. ish AJ0

d. s NN2-VVZ

Even if not in the lexicon, *tigress* and *kindness* will both be assigned the tag NN1, since they match the entry (5b). Note that -*ess* in *kindness* is not a suffix from a linguistic point of view, but that does not matter to the tagger. A word will only match against a suffix if it has at least three additional letters; this is to avoid words like *dish* being mis-analysed as adjectives by (5c). As (5d) shows, suffixes can be ambiguous as to which tag they indicate. Suffixes are ordered with longer ones being tried for a match before shorter ones: otherwise, -*ess* would never be found if -*s* were tried before it.

There is also a prefix lexicon, consulted after the suffix lexicon, which states that, for instance, words beginning with *re-* are verbs. Specifically, they are either base or infinitive forms of verbs, since forms like *reanalysed* will already have been shown as either past tense or past participle by the suffix lexicon. However, the prefix lexicon is not very helpful, as a prefix is not a reliable guide to part of speech in English, and certainly not to inflectional class.

If a word is not in EPT's lexicon and the affix lexica cannot help, then it is treated as an unknown word and the tag NN1-VVB-VVI-AJ0-AV0 is assigned. A unknown sentence-initial word gets the tag NP0-NN1-VVB-VVI-AJ0-AV0. It is assumed that function words will be in the lexicon, so only lexical tags are used for unknown words. Only tags that imply the absence of any inflectional suffixes which indicate category are included: an unknown word ending in -*s*, for instance, will be tagged using (5d). But *house*, for example, gives no morphological indication of its category, and so is regarded as an unknown word.

The output of lexical look-up and processing of unknown words, then, will be for each token information about (a) the tag it will be assigned or a list of possible tags, and (b) the source of the tag(s), so that the correct link can be printed. The remaining task is to decide, in the case of multiple potential tags, which one is correct.

Amendment Rules

After lexical look-up, a set of rules is applied that can select one of the list of possible tags associated with a word as being most likely to be correct and discard all the other tags. These amendment rules consist of a sequence of items, one of which is the tag to be selected and the rest of which state the context in which this amendment applies. A simple example:

(6) AT0 NN2+

The '+' shows the tag on the item to be amended, and the other item in the rule shows the context, i.e. the preceding token is tagged AT0. So *answers* could be either a plural noun or a 3rd-singular present tense verb, and rule (6) states that in a sequence such as *the answers* it must be a plural noun.

Here is another amendment rule:

(7) VH. VVN+

This ensures that, after a form of HAVE, ambiguous forms such as *bought* and regular *-ed* verb forms are tagged as past participles rather than past tenses. In VH . , the final dot stands for 'any character', so the context is 'any item whose tag begins with VH', i.e. all and only forms of HAVE.

Most patterns in amendment rules refer to tags, but they can refer to tokens as well, though such a token cannot have the '+' appended to it. An example of a rule that picks out a specific token is for cases such as *simpler than*:

(8) AJC+ than

'Make AJC the sole tag before *than*': this cannot refer to the class of prepositions, because potential comparative adjectives with the suffix *-er* can occur as nouns before other prepositions (e.g. *a writer for hire*). An item in an amendment rule can also pick out both a token and a tag:

(9) as=AV0+ AJ0 as

'When it precedes an unmarked adjective and *as*, *as* is tagged as an adverb'; this applies to a sequence like *as tall as* (where the second *as* is analysed as a preposition). The token and the tag are linked in the rule by an equals sign. Alternative tokens can be listed too:

(10) she|he|it=PNP VVZ+

'Make VVZ the sole tag after the pronouns *she*, *he* or *it*.' The rule needs to be so specific because if it refers to the pronoun tag PNP only, it will give the wrong results in a case such as *We set them exercises*. Experience shows that this kind of rule is not needed very often.

The last kind of pattern that can be captured in amendment rules is that of a part of a word, specifically a suffix:

(11) AT0 -ing=AJ0+ N.

'A word ending in *-ing* must be an adjective when preceded by a determiner and followed by a noun' (as in *an entertaining film)*. The notation for a suffix involves

preceding it with a hyphen. In this case, and with the list of alternative tokens followed by a tag, EPT converts the statement in the amendment rule to a rather more complex form, and it is the latter that is printed out in verbose mode when such rules apply (see (13)).

In verbose mode, EPT gives information about the application of amendment rules, e.g. (cf. (6)):

(12) Amending tag of 2: answers to NN2: AT0 NN2+

The '2' here refers to the second token in a sentence such as *The answers are obvious*. For *He answers the question*, where rule (10) has applied, the user sees:

(13) Amending tag of 2: answers to VVZ: ^(she|he|it)$=PNP VVZ+

The idea of amendment rules is based on the pattern-action rules described in Voutilainen (1999), except that amendment rules are simpler to write but less powerful. Voutilainen's rules can refer to unbounded context, and they can remove potential tags rather than just selecting the (hopefully) correct tag. Some of the amendment rules have similar effects to those found in Brill (1995), except that the rules there are learned by the system rather than written by a human.

There is a problem with amendment rules in that the tag referred to in the context part of a rule may not be the one finally assigned by EPT. For instance, consider *John's science course*, where *'s* is an ambiguous token. The following amendment rule applies:

(14) POS+ AJ. NN.

'Select POS before an adjective and noun'. The problem is that science is tagged NN1, but at the stage when rule (14) applies, it still has AJ0 as one of its potential tags, from the default tag for unknown words. As it happens, in this case we get the right result, but that is a matter of luck.

Default Tags

After the amendment rules have applied, some tokens will still have more than one potential tag. In these cases, EPT chooses the leftmost of the possible tags as a default. For this reason, the order of tags for an ambiguous token or affix matters, in that the tag listed first should be the default. In some cases, this will be the commonest tag, as for *might*, or for *own*, which is more commonly an adjective than a verb (though EPT itself makes no use of frequency information). In other cases, it is easier to state amendment rules for one or other of the potential tags, leaving the

default to be the tag chosen when no amendment rule applies to select any of the other tags. For example, rule (7) selects a past participle tag after `HAVE`. The default for the ambiguous forms is past tense, as it is easier to state when a past participle occurs than when a past tense form does.

Recall that an unknown word is assigned the tag `NN1-VVB-VVI-AJ0-AV0`. Unless one of the other tags is selected by an amendment rule, `NN1` will be chosen by the default rule. So any token for which EPT has no other information at all will be analysed as a singular noun. This seems reasonable given that there are more nouns than members of other word classes in English.

In verbose mode, EPT outputs a message that the default leftmost tag has been chosen, e.g.:

(15) Choosing leftmost tag for: annoyed

This would be the case for an example such as *The man annoyed us*.

Extending EPT

The user can extend the coverage of EPT by editing the files it employs (all are simple text files, so no complex formatting is needed). The files that can be edited are: (a) the lexicon file, (b) the files containing affixes (one for suffixes, one for prefixes), and (c) the file of amendment rules. Students familiar with Perl can of course amend the basic EPT program if they wish, though that is not the intention behind the system.

Banko and Moore (2004) have emphasised the crucial importance of an accurate lexicon in improving tagging accuracy. The lexicon file supplied with EPT has around 250 entries. It contains many function words but only a small selection of content words: most sentences will require additions to the lexicon, though quite often the system will give the correct result for an unknown word. Entries along the lines of those in (3) can be added quite simply, one entry per line. For the benefit of the user, it is best to keep the lexicon file in alphabetical order, but the order is of no relevance to EPT. When a word is added to the lexicon, the user should allow for all its potential tags, not just the one that is appropriate for the sentence being analysed. The affix files can also be edited in the same way (see the entries in (5)). Again, it is best to keep the entries in alphabetical order. EPT initially has information on a dozen suffixes and two prefixes.

Amendment rules also have one rule per line (with comments allowed following '%'). Some frequently-invoked rules are included already, but many new ones can be written. These rules are the hardest part of EPT for the user to grasp and writing new

rules requires a fair bit of practice. The system's User Manual gives some hints on writing amendment rules.

Verbose mode is very helpful in enabling the user to see why the tagger needs extending or why an extension is not behaving in the way intended.

My standard practice has been to suggest sentences that students should enter, so they will understand how various parts of the system operate. So *He'll put them on the shelf* illustrates tokenisation of *He'll* as *He* + *'ll*, and an amendment rules that ensures that (in this case) *put* after a modal is tagged as an infinitive. In *My friend has gone to the theatre*, *gone* is given a default tag as a noun and needs to be added to the lexicon. For *Did you get a paper?*, *get* is tagged as the base form of a verb, and a new amendment rule is needed that will ensure that the sequence of a form of *do* and a pronoun is followed by an infinitive. Once students have learned the basics, they can enter sentences from the web or of their own devising, and see what mistakes EPT makes and how they are best corrected. A good way to test EPT is to input sentences where an ambiguous word has different analyses and to see whether these are handled correctly (e.g. one with *hope* as a noun and one with it as a verb). If not, writing new amendment rules will, hopefully, solve the problem. On the whole, students feel far more engaged when working on a system that they can extend and modify themselves than when running software over the web, which is an essentially passive experience, however instructive.

Students who use EPT are final-year undergraduates, mostly taking degrees in Linguistics or English Language, who as part of a computational linguistics module have previously attended one general lecture on tagging and one lecture specifically on EPT, as well as having some experience of running the CLAWS4 tagger online. Over one hundred students have now used the system. In a single laboratory session, students are able to run the system, add lexical and suffix entries, and gain some initial experience of amendment rules. They can also choose to write a coursework assignment on EPT, which involves selecting their own input text(s), expanding the system's coverage so that the input is analysed as accurately as possible, and then discussing and motivating the additions they have made. The discussion of student experience in the next section is mainly based on the work of those who have completed this larger assignment.

Evaluation

The criteria against which EPT will be evaluated are those of accuracy, transparency and extensibility.

EPT has mainly been tested and evaluated by its developer using sample sentences drawn from the BBC news website (http://news.bbc.co.uk). 'Out of the box', with no additions to the lexica or new amendment rules, EPT tags sentences with approximately 80% accuracy. Given the size of the lexicon, this figure includes massive reliance on affixes and unknown word tags. Simply adding words to the lexicon can increase this figure to roughly 90%. One student used EPT on a text of 270 words; in its original form EPT tagged 239 tokens correctly (88.5%), and after the expansion of its lexica and addition of amendment rules, it scored 256 correct (94.8%). This figure is still below the accuracy achieved by 'proper' taggers, but the point is that EPT can still be instructive when it makes mistakes, as this shows students what some of the problems with tagging are. A much lower accuracy rate (such as 60%) would be unacceptable, but 80–90% still provides an insight into how some tagging problems can be solved and how others (e.g. VP coordination) are much harder to tackle. For instance, it is difficult to ensure that *send* in (16) is tagged `VVI` as the head of one conjunct dependent on *must*:

(16) Candidates must complete an application form and send it to the following address.

The following example is also tricky:

(17) The weapon was handed in to police.

Since *police* can be a verb as well as a noun, and *to* is ambiguous between preposition and infinitive-marker, ensuring the correct analysis in (17) is very difficult. Having students assess the accuracy of EPT and identify where the problems lie is very educational for them.

According to Halteren, Daelemans, & Zavrel (2001), the tag pairs most often confused by taggers for English are:

(18) a. past tense vs. past participle

b. adjective vs. singular noun

c. preposition vs. subordinating conjunction

d. base form of verb vs. singular noun

e. preposition vs. adverb particle

Of these, (18c) is not a problem for EPT as supplied, since this distinction is not made for words like *before* and *after* (see discussion of (4) above). The choice between preposition and particle in *John turned on the radio/sat on the chair* (cf. (18e)) is very difficult, and perhaps this distinction could also be collapsed. The case in (18a) can be

handled, in part, by amendment rules such as (7). Nouns, verbs and adjectives (as in (18b, d)) can also in most instances be disambiguated by using amendment rules. But there is still plenty of work for the user to do.

In terms of transparency, EPT definitely scores very highly. The user is given a lot of information about why a token has been tagged in a particular way. Without using verbose mode, the links shown in output such as (1) convey a great deal, while verbose mode provides more information about the use of affixes, unknown-word rules, the choice of default tags and the application of amendment rules. This helps in both understanding the tagger's working and in improving its performance or extending its coverage.

Extensibility is another feature. Adding items to the word and affix lexica is simple, and additions can be tested quickly and easily. Some students have developed their own ways of extending EPT's lexical coverage, by incorporating vocabulary listed in Kilgarriff (1998). Amendment rules are harder, but experience suggests that once they 'get the idea', writing new and accurate rules is achievable for most students. An unanticipated consequence is that students are led to reflect on English sentence structure and the best way to disambiguate certain tricky forms, by asking themselves questions such as 'How can we do our best to ensure the correct analysis of word X using the limited amount of information available to a tagger?'.

Conclusion

EPT succeeds well in its aim of providing a framework within which students, especially those from a linguistics background, can learn (and teach themselves) about tagging. They learn about English affixes and vocabulary, lexical ambiguity, and ambiguity resolution. They learn how to manage a small system for computational analysis of language, including ensuring that files are consistent and up-to-date. Users also learn a fair amount about the structure of English.

A similar system for other areas of computational linguistics, such as chunking or parsing, would be a major undertaking. However, extending the system by converting it to work for other languages, where the same underlying framework would still operate but different lexica and amendment rules would be needed, would be relatively straightforward. One possibility would be to parameterise the system so that, for instance, the set of tags for unknown words would be listed in a separate file from the basic tagging program and could be altered straightforwardly by users.

References

Banko, M., & Moore, R. (2004). Part-of-speech tagging in context. *In Proceedings of Coling 2004* (pp. 556-561). Geneva.

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. Sebastopol, CA: O'Reilly.

Brill, E. (1995). Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging. *Computational Linguistics*, 21, 543-565.

Daelemans, W., & van den Bosch, A. (2005). *Memory-based language processing*. Cambridge: Cambridge University Press.

Dalrymple, M. (2006). How much can part-of-speech tagging help parsing?. *Natural Language Engineering*, 12, 373-389.

Garside, R., Leech, G., & McEnery, A. (Eds.) (1997). *Corpus annotation*. London: Longman.

Halteren, H. van, Daelemans, W., & Zavrel, J. (2001). Improving accuracy in word class tagging through the combination of machine learning systems. *Computational Linguistics*, 27, 199-229.

Jurafsky, D., & Martin, J. (2009). *Speech and language processing*. Upper Saddle River: Prentice Hall (Second ed.).

Kilgarriff, A. (1998). *BNC database and word frequency lists*. Retrieved 18 February 2010 from http://www.kilgarriff.co.uk/bnc-readme.html

Leech, G. (1997). Grammatical tagging. In R. Garside, G. Leech, & A. McEnery (Eds.), *Corpus Annotation* (pp. 19-33). London: Longman.

Mason, O. (2000). *Programming for corpus linguistics*. Edinburgh: Edinburgh University Press.

Melamed, D. (1996). *NLP research software library*. Retrieved 8 February 2010 from http://cs.nyu.edu/~_melamed/genproc.html.

Minnen, G., Carroll, J., & Pearce, D. (2001). Applied morphological processing of English. *Natural Language Engineering*, 7, 207-223.

Schmid, H. (1997). Probabilistic part-of-speech tagging using decision trees. In D. Jones & H. Somers (Eds.), *New methods in language processing* (pp. 154-164). London: UCL Press.

Voutilainen, A. (1999). Hand-crafted rules. In H. van Halteren (Ed.), *Syntactic wordclass tagging* (pp. 217-246). Dordrecht: Kluwer.

Voutilainen, A. (2003). Part-of-speech tagging. In R. Mitkov (Ed.), *The Oxford handbook of computational linguistics* (pp. 219-232). Oxford: Oxford University Press.