Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.863J/9.611J Natural Language Processing, Spring, 2003

Laboratory 2: Statistical and Rule-based Tagging

Handed Out: January 24

Due: March 05

Introduction: Goals of the Laboratory

By now you have heard a lot about different part of speech taggers. In this laboratory, you will get a chance to evaluate both a statistical, Hidden Markov Model (HMM) tagger and the rule-based Brill tagger, running them on large chunks of text, and evaluating their output. Your job will also be to figure out where they go wrong, if at all, and why. The Brill tagger is described in chapter 8 of the Jurafsky and Martin text (and in class); the HMM tagger is one from the Edinburgh language technology group. Instructions for running each of these taggers are available in running_taggers.html. A pdf version of the paper describing Brill's tagger is available at brill95transformationbased.p. (Apologies in advance for the lousy conversion - not on my watch.) Technical papers describing the HMM parser are available at cl-unknown.ps. Both of these taggers were trained on data from the Wall Street Journal corpus, as hand-tagged by the University of Pennsylvania – and roughly the same training text. (This is to get the bigram probability estimates. You change the training statistics, as the running-tagger document describes, if you wish – but this is a more ambitious, and computationally intensive, goal.)

Note further that both taggers use the **Penn treebank tags**. (See your text or the class slides for a listing of these and their cryptic names, e.g., "JJ" = adjective.)

Your basic job will be to run both taggers on the 13 text collections below. We have partitioned these because some of the files come from different texts than what the taggers were trained on; all the files are collected together into one large file `all.raw` and `all.pos`. You will make use of this partitioning, as you'll see below. As you will note from the instructions page, you should capture the tagged output by redirecting it to a file in your own directory. As usual, you should write up your results as a report in html format, and submit the URL to the TA. In particular, please include the information labeled "INCLUDE" below (in addition to other discussion that we shall point out).

Section 1. Warmup

Run each of the taggers on the file per the instructions on the file `magi.txt`, available at the web page. (Do **not** use the file magi-raw.txt – this file does not break up, or tokenize punctuation properly, and is meant for you just to read.) This is a complete version of O'Henry's famous short story "Gift of the Magi." *Please* direct any output to your own directory.

(a) The word ***let's*** occurs three times in the magi text. What does the Brill tagger give as its final tags for each of these three occurrences? Are they correct? If they are incorrect, point out which tags are incorrect, and explain why. Please include in your discussion a list of what *initial* tag is assigned to each occurrence, and which (if any) contextual rules are used by the Brill tagger to change this initial setting to its intermediate and then final values. You will have to use the program option `-I` so as to write the intermediate results to a file (see the description of the running the taggers). (The three occurrences: "Take yer hat off and let's…"; "Let's be happy. You don't know….;" "let's put our Christmas presents away…")

(b) Repeat using the HMM tagger (except this tagger does not produce any intermediate output, of course) – we just want you to see what it does on ***let's.***

INCLUDE in your report: answers to part (a) and (b) above.

Section 2. Comparing tagger performance, part 1

Now we will repeat this experiment on a much longer run of text. Run each of the taggers on the following texts from the Penn Treebank and compare their output to the "gold standard" tagged texts. In each of the lines below, the link to "Text *n*" (e.g., "Text 1") is to a version of the text formatted with one sentence per line - this is easier to read, but you **should not use it** for the actual tagging experiments. The second column has the **untagged** versions of the same texts, tokenized and formatted for use by the taggers. **These are the files you should use as input to the taggers. These files are all labeled as *.raw.** The third column, with "Tagged" link is to the human-tagged file from the Treebank, which is the "gold standard" – truth – to be used to evaluate the tagging. **These corresponding tagged files are all labeled *.pos.** Their tags are gospel. (You will notice that they are actually bracketed, or parsed as well, but we will ignore this information for now.) The texts are divided into chunks you see for ease of handling -- all the texts are bundled together in one file as well, but that might prove to be too large to look at (it is, however, useful for evaluating the taggers). These files are made available on the web page.

- Text 1 (ce04.txt) Untagged (ce04.raw) Tagged (ce04.pos)
- Text 2 (cf05.txt) Untagged (cf05.raw) Tagged (cf05.pos)
- Text 3 (cj01.txt) Untagged (cj01.raw) Tagged (cj01.pos)
- Text 4 (ck09.txt) Untagged (ck09.raw) Tagged (ck09.pos)
- Text 5 (cl04.txt) Untagged (cl04.raw) Tagged (cl04.pos)
- Text 6 (cm05.txt) Untagged (cm05.raw) Tagged (cm05.pos)
- Text 7 (cn03.txt) Untagged (cn03.raw) Tagged (cn03.pos)
- Text 8 (cp01.txt) Untagged (cp01.raw) Tagged (cp01.pos)

- Text 9 (cr03.txt) Untagged (cr03.raw) Tagged
- Text 10 (sw2019.txt) Untagged (sw2019.raw) Tagged (sw2019.pos)
- Text 11 (wsj_0579.txt) Untagged (wsj_0579.raw) Tagged (wsj_0579.pos)
- Text 12 (wsj_1220.txt) Untagged (wsj_1220.raw) Tagged (wsj_1220.pos)
- Text 13 (wsj_1975.txt) Untagged (wsj_1975.raw) Tagged (wsj_1975.pos)

All the texts in one file: Untagged (all.raw) Tagged (all.pos). Choose **five tagging errors** made by each tagger (*i.e.*, 10 errors in total) and discuss the possible reasons for these errors.

In addition, as you may learn from looking at them, these texts are drawn from different sources. In particular, only the last three (Texts 11, 12, 13) are from the Wall Street Journal – the same material the taggers were trained on. Text 10 is particularly interesting, because it is from the so-called "Switchboard" data of actual phone conversations (more or less). Texts 1-9 are from written material other than the Wall Street Journal. Thus, we would expect performance to vary. Please study the differences in tagging errors (if any) from genre to genre (naturally, we might expect the last three texts to give the taggers the least problems).

INCLUDE in your report: Log files exhibiting the tagging errors you are discussing, and your discussion of the errors, including any shifts as the text genre itself changes.

Section 3. Comparing tagger performance, part 2

Quantitatively compare the performance of the two taggers. To do this, you will use the file compare-taggers.pl to compute the confusion matrices comparing each tagger's output to the gold standard and to compute Kappa for each tagger.
A description about how to use the program is given in evaluating-taggers.html.

- Compute Kappa for each tagger. Which one performs better on these data? Do the comparison text by text (1 through 13, and then all, for 14 comparisons, and construct a histogram).
- What is causing the errors? Use the *confusion matrices* to identify any systematic errors. (Yes, I know, you may have to print out the confusion matrix in a more readable fashion, so as to fit it onto one page.) Describe **three** of these systematic errors and show an example of each. If genre changes are relevant, discuss them also.
- How might you *repair* these systematic errors? (Don't go overboard here – one paragraph is enough, but not glittering generalities please, like "more data."

INCLUDE in your report: Kappa values for each tagger, and answers to the above questions as to performance, systematic errors, and possible fix-ups.

Section 4. Taggers and Kimmo

It has probably not escaped your attention (since I've said it three times in class) that the tags that these engines use are impoverished – the so-called Brown corpus tags. We certainly don't have the richness provided by the PC-KIMMO machinery, with fine-grained features and word decomposition – like the parsing of Spanish verbs into their tense and endings. So why use Kimmo at all? In no more than one page, please come up with the (rough!) specifications for a design that could ***integrate*** these two tasks (1) PC-KIMMO word parsing; and (2) tagging; describing how each might assist (or not) the other. Please illustrate on three or four examples (no need to implement unless you feel *very* ambitious).

INCLUDE in your report: your answer to the above 'resolution' between PC-KIMMO and tagging.