Homework #4: CMPT-825

Due in class on Oct 3, 2003

Anoop Sarkar – anoop@cs.sfu.ca

(1) (100pts) **Part of Speech Tagging**

All the files for this assignment are in `/cs/825/data/postags`

Part of speech tagging is the process of assigning to a sequence of words, a sequence of part of speech tags. The set of *part of speech tags* can be viewed as a set of class labels for words. This set is finite and fixed based on some linguistic analysis of the language being tagged. Each word in the language is assigned to at least one of these class labels. Our problem is to resolve ambiguities when one word can fall into more than one class. For example, in the sentence *They can fish*, both *can* and *fish* can receive more than one part of speech tag: *can* could be a noun or a verb, while *fish* could be a noun or a verb as well. This can lead to several possible tag sequences: `Pronoun Noun Noun`; or `Pronoun Noun Verb`; or `Pronoun Verb Verb`; or `Pronoun Verb Noun`. Part of speech tagging is the task of finding the best assignment of part of speech tags to each word in the sentence. A tagger should be able to mimic our preference for the last two tag sequences over the first two tag sequences.[1]

In this assignment we will be using one of the most widely used part of speech tagsets for English. It is part of the Penn Treebank annotation of about a million words of Wall Street Journal newspaper text. A guide to the Penn Treebank part of speech tags is available in the document `tagguide.pdf`. More information about the Penn Treebank is available from the following web page:

`http://www.cis.upenn.edu/~treebank/home.html`

The training data consists of each sentence (word sequence) associated with a corresponding tag sequence. This tag sequence has been verified by human linguistic experts to be the right part of speech for each word.[2] The testing data is the same format as the training data, but the true tag is not provided as input to the tagger, rather it is used to compute the accuracy after the tagger has proposed a part of speech tag for each word. The training and testing data is as follows:

- Training data: `sections01-09.postags.txt`
- Testing data: `section00.postags.txt`

I've provided a program `cleanPennTreebankPostags.pl` which takes the data files shown above and prints out the data in a form that is more convenient to process. The output looks as follows:

```
word11 tag11
word12 tag12

word21 tag21
word22 tag22
```

[1] Each tag sequence can be paraphrased to express the ambiguity in the sentence. For example, `Pronoun Verb Verb` refers to *their ability to fish*, while `Pronoun Verb Noun` to *their activity of canning fish*.

[2] However, there can be *noise* in the data since the experts can make mistakes.

In this format, each word and its associated part of speech tag appears one per line, e.g. `word11` is the first word of the first sentence and `tag11` is the part of speech tag associated with that word. A new sentence is marked with an empty line, e.g. `word21` is the first word of the second sentence.

Your task is to write a part of speech tagger. The tagger must take the sentences from the testing data as input and produce a proposal for the correct part of speech tag for each word in the sentence.

The testing data already contains the correct tag. Add the output of the part of speech tagger as an additional field to the testing data. Call this file `test.output` in the following format:

```
word11 tag11 proposedtag11
word12 tag12 proposedtag12

word21 tag21 proposedtag21
word22 tag22 proposedtag22
```

I've provided a program for evaluating your output: `scorePostags.pl` which takes your file `test.output` in the format shown above and reports on the accuracy of your part of speech tagger. Here is how to run the scoring program:[3]

```
perl scorePostags.pl < test.output
```

You can implement the part of speech tagger using **one** of the following methods. Whichever method you choose, you should submit your accuracy on the testing data as reported by `scorePostags.pl`.

a. **Rule-based approach**: Using the training data, come up with a set of rules (or regular expressions) that correspond to tagging decisions to be made.

   For example, one possible rule-based system you could implement would start with a default assignment of a postag to every word in the testing data. Then each subsequent pass through the data could make a change from one postag to another based on the surrounding context. The pseudo-code for applying rules to the test data would look like this:

```
FOR each word:
  IF word was seen in training data
  THEN assign most frequent postag seen for this word
  ELSE assign the postag 'NNP' (proper noun)

FOR each word:
  Apply each rule and then break:

    IF current word has postag IN (preposition)
      AND the word two positions to the right is 'as'
    THEN change postag to RB (adverb)

    IF current word has postag VBP (verb, non-3rd-person singular present)
      AND the word 'n't' appears either one or two positions to the left
    THEN change postag to VB (verb, base form)

    ... and so on ...
```

   Note that you don't need much linguistic intuition: the rules can be written by looking at the *errors* made by the rule-based system *on the training data* and trying to fix those errors. One way of sorting these rules is by the number of errors made on the training data.

[3] You can change the delimiter between fields using the `-d` option. The default is a space character.

b. **Classifier approach**: The training data can be used to create a classifier that attempts to find the best class for each word. The set of classes is equal to the set of part of speech tags. Let's assume that the set of part of speech tags is C and $c \in C$. Also assume that each word can be represented as set of attributes $\mathbf{x} = x_1, \ldots, x_n$ (such as the word itself, the previous word, the most frequent tag in training for the previous word, the last three characters of the word, ...). One way to define a classifier is by using Bayes rule:

$$P(c \mid \mathbf{x}) = P(c) \times P(\mathbf{x} \mid c)$$

A *Naive Bayes* classifier simply assumes that each of the attributes $x_1, \ldots, x_n$ in $\mathbf{x}$ are independent of each other:

$$P(\mathbf{x} \mid c) = P(x_1, \ldots, x_n \mid c) = \prod_{x_i \in \mathbf{x}} P(x_i \mid c)$$

In practice, it is easier to compute the log probability $log(P(c \mid \mathbf{x}))$.

$$log(P(c \mid \mathbf{x})) = log(P(c)) + log(P(\mathbf{x} \mid c)) = log(P(c)) + \sum_{x_i \in \mathbf{x}} log(P(x_i \mid c))$$

The part of speech tag c is assigned to each word w based on which class c gets the highest probability $P(c \mid \mathbf{x}(w))$.[4] Note that in the classifier approach, the best sequence of tags is not computed. Instead, for each word, the classifier will pick the best tag based on the attributes.
In the directory `/cs/825/software/nb.pl` I have provided an implementation of a Naive Bayes classifier. The program is called `nb` and you can look at the options it takes using `perl nb.pl -h`. The training data for this classifier is in the following form[5]:

```
x1 x2 ... xn c
```

where `x1 x2 ... xn` are the attributes and `c` is the class label. The command

```
perl nb.pl -f nb.train -l nb.input -m nb.output
```

will train the classifier on the training data `nb.train` and then label the contents of `nb.input` which has the same format as the training data except the class labels (the last column) are missing.

c. **Noisy channel approach**: This approach is similar to your previous homework on the noisy channel model. In this case, the best sequence of tags is computed according to the product of a tag-based language model and the mapping between words (or their distinctive attributes) and the tags.
The best (or most likely) tag sequence $t_0^*, \ldots, t_n^*$ is:

$$t_0^*, \ldots, t_n^* = \underset{t_0, \ldots, t_n}{\arg\max} \; P(t_0, \ldots, t_n \mid w_0, \ldots, w_n)$$

The noisy channel model implies that we can write this as follows:

$$P(t_0, \ldots, t_n \mid w_0, \ldots, w_n) = \prod_{i=0}^{n} P(w_i \mid t_i) \times P(t_i \mid t_{i-2}, t_{i-1})$$

Notice that this can be re-written as:

$$P(t_0, \ldots, t_n \mid w_0, \ldots, w_n) = P(t_0, \ldots, t_n) \times \prod_{i=0}^{n} P(w_i \mid t_i)$$

You can use the SRI Language Modelling toolkit in order to compute $t_0^*, \ldots, t_n^*$.

[4] Or, pick class c for the lowest log probability $log(P(c \mid \mathbf{x}))$.
[5] If you are familiar with decision-trees you could use C5.0 instead of Naive Bayes.