Part of Speech Tagging

The goal of this exercise is to allow you to implement your own, simple version of the Brill tagger. Download the additional material from the web page. This archive contains a training and a test set, which consist of child-directed sentences from the CHILDES corpus which have been transcribed and PoS-tagged. Additionally, we provide a simple Python script in order to help you with an efficient implementation of the training algorithm.

Python script

Training the Brill tagger is done by building all possible candidate rules, picking the best one, and repeating. In order to assess which rule is the best, both the positive and the negative effects of that rule must be considered. If we try each rule on the text and count the changes it made, we end up with a runtime proportional to the text length times the number of rules. Since rules always take the form the current tag is $t_i$ and the next / second next / previous / etc. tag is $t_j$, a more efficient implementation is to step linearly through the text, count the relevant tag combinations, and use these to assess the effects of each rule. The script builds a set of dictionaries which represent these counts. If you are using language other than Python, you should be able to translate the script easily.

The algorithm: Training

Initialization

- Assign every word its most likely tag according to frequency on the training data.
- Assign to unknown words the noun tag $n$.

Candidate rules

For every incorrectly tagged word, generate the possible transformation rules repairing this tag and add them to the set of candidate rules.

Rules take the form $t_i \rightarrow t_{new} \text{ if condition}$. There are 9 different triggering environments resulting in up to 15 transformation rules per incorrectly tagged word:

1) The current tag is $t_i$ and the previous tag is $t_j$
2) The current tag is $t_i$ and the following tag is $t_j$
3) The current tag is $t_i$ and one of the two previous tags is $t_j$
4) The current tag is $t_i$ and one of the two following tags is $t_j$
5) The current tag is $t_i$ and one of the three previous tags is $t_j$
6) The current tag is $t_i$ and one of the three following tags is $t_j$

7) The current tag is $t_i$, the previous tag is $t_j$, and the following tag is $t_k$

8) The current tag is $t_i$, the previous tag is $t_j$, and the tag in position $+2$ is $t_k$

9) The current tag is $t_i$, the tag in position $-2$ is $t_j$, and the following tag is $t_k$

**Rule learning**

Implement the learning algorithm to find the set of transformation rules to be used. $E(C_k)$ is the number of incorrectly tagged words in corpus $C_k$. $v(C_k)$ is the corpus resulting from applying the transformation $v$ to $C_k$. $\epsilon$ is an input error threshold. The algorithm proceeds as follows:

1: $r :=$ empty list of transformation rules
2: $C_0 :=$ initialized corpus
3: for $k := 0$ step 1 do
4: $v :=$ the transformation minimizing $E(v(C_k))$
5: if $E(C_k) - E(v(C_k)) < \epsilon$ then
6: break /* Skip this rule if improvement is below threshold */
7: end if
8: $C_{k+1} = v(C_k)$ /* Apply rule to corpus */
9: $r_{k+1} = v$ /* Add rule to rule set */
10: end for
11: return $r$ /* Return list of transformation rules */

**The algorithm: Tagging**

After picking the transformation rules, your Brill tagger is ready to be used. Application works as follows:

- Initialize the input text as before;
- Apply the transformations in the order given by the training algorithm to the entire text.

**Task**

Implement the algorithm described above. Test your implementation on the test portion of the dataset and report the resulting accuracy as well as the rules chosen. Try different values for $\epsilon$, the error threshold used during rule learning. How does the choice of $\epsilon$ affect accuracy?

Your submission should include a brief report on your solution, as well as the source code you wrote to solve the exercise. If possible, avoid using nonstandard libraries not included in the distribution of the programming language you used. Make sure to include brief instruction on how to run your code.

Send your solutions to arif.khan@lsv.uni-saarland.de (if you are attending the Wednesday tutorial) or andrea.fischer@lsv.uni-saarland.de (if you are attending the Tuesday tutorial) by Friday, 28 June. Important: Please use PDF as a document format. If you need to compress files, use ZIP or GZIP.