ASSIGNMENT 1

End-of-Sentence Detection and Text Segment Classification

Due date: Tuesday, February 26, 2013 in class

Part 1: End-of-Sentence Detection

The goal of the first part of the assignment is to create an algorithm for determining whether a given period (".") in a text indicates an end of sentence or just an abbreviation marker. The following examples illustrate some of the difficulties encountered in this distinction:

```
NOT-END-OF-SENTENCE  119   l viewpoint , as David  C. Robinson has recently shown , t
NOT-END-OF-SENTENCE  128   evealed , '' by Arthur  C. Clarke , Gentry Lee ( Bantam )
NOT-END-OF-SENTENCE  136   E   . The group led by  C. Delores Tucker , head of the NA
NOT-END-OF-SENTENCE  147   ence and Electronics ;  C. Scott Kulicke , on behalf of Se
NOT-END-OF-SENTENCE  184   g Committee chaired by  C. Rubbia . The report covers stat
END-OF-SENTENCE      192    occurs at 440 degrees  C. A hydrogenation test was carrie
END-OF-SENTENCE      210   ystem at 25 and 50 deg  C. Isotherms consist of five branc
END-OF-SENTENCE      239    C while not at 40 deg  C. Minima on the S/sub Pu / vs. C/
NOT-END-OF-SENTENCE  247   ellulases . Culture of  C. thermocellum will be optimized
NOT-END-OF-SENTENCE  255   the cellulase genes of  C. cellulolyticum and those from t
END-OF-SENTENCE      258   anging from 200 to 300  C. A system developed by the autho
END-OF-SENTENCE      262   ourse on programming in C. Finally, those who are interest
END-OF-SENTENCE      300    a house on 2213 Perry Dr. Then the Thomases were seen in
NOT-END-OF-SENTENCE  330     . <P>  Early in 1980 Dr. Thomas B. Reed of SERI and Pro
```

To help you develop a classifier for this distinction, an example set of 45,000 periods and their surrounding context has been provided, each one labelled as `EOS` (End-of-sentence) or `NEOS` (Not-end-of-sentence). The examples were extracted primarily from the Brown Corpus and are located in the file `$CS466/hw1/sent.data.train`, where `$CS466` is equal to `/users/rtfm2/cs466` on peregrine and `/home/1/yarowsky/cs466` on hops.

For easy manipulation, the training examples have been divided into tab-delimited columns containing the following information:

- Column 1: `EOS` or `NEOS`, indicating whether the period in that line marks an end of sentence marker or not.
- Column 2: The ID number of the sentence.
- Columns 3-9: The ±3-word surrounding context of the period.
- Column 10: The number of words to the left of the period before the next *reliable* sentence delimiter (e.g. ?, ! or a paragraph marker `<P>`).
- Column 11: The number of words to the right of the period before the next *reliable* sentence delimiter (e.g. ?, ! or a paragraph marker `<P>`).
- Column 12: The number of spaces following the period in the original text.

An example of the first 8 columns for the data above is:

```
TAG   ID#   -3            -2        -1   0      +1             +2
=========================================================================
NEOS  119   as          David      C    .    Robinson        has
NEOS  128   by          Arthur     C    .    Clarke          ,
NEOS  136   led         by         C    .    Delores         Tucker
NEOS  147   electronics ;          C    .    Scott           Kulicke
NEOS  184   chaired     by         C    .    Rubbia          .
EOS   192   440         degrees    C    .    A               hydrogenation
EOS   210   50          deg        C    .    Isotherms       consist
EOS   239   40          deg        C    .    Minima          on
NEOS  247   Culture     of         C    .    thermocellum    will
NEOS  255   genes       of         C    .    cellulolyticum  and
EOS   258   to          300        C    .    A               system
EOS   262   programming in         C    .    Finally         ,
EOS   300   2213        Perry      Dr   .    Then            the
NEOS  330   in          1980       Dr   .    Thomas          B
```

The classifier you develop should be able to take data of this format and predict whether the correct label is EOS or NEOS.

You may create your classifier either using hand-crafted rules or empirically derive a decision procedure from the data using a machine learning algorithm such as a decision list or neural net. The choice is up to you, although you are strongly encouraged to pursue an empirical approach.

To test the effectiveness of your program, your code will be applied to another file of 9,000 different examples in identical format (`sent.data.test`). For maximum fairness of the test, you will not be able to see this test data in advance.

To assist you, in the directory `$CS466/hw1/classes` there are some files containing wordlists of abbreviations, titles, unlikely-proper-nouns, and other word classes that may be of use in this classification. (See page 1 for the different directory paths for `$CS466` on the masters and ugrad machines).

There are three example programs in `$CS466/hw1/examp*.ps`. These are basic templates showing how one might begin a classifier in Perl. `examp1.prl` illustrates one way in which the wordlists in the `classes` directory may be used to test class membership.

You may program in any language of your choice, although your code must be able to run on data of the format given in the `sent.data.train`. Your program should return *for each line in the file* the classification assigned by your program, followed by the full line of data (including the "correct" classification and the example number). The `&ret` (return) function in `examp1.prl` gives an example of this.

The example version of `&ret` (return) keeps track of the total number of correct and incorrect answers given by the program when run over a sequence of labelled data lines. When run on the full training data, this should will give you quantitative feedback on the current effectiveness of your program. However, the quality of feedback can be improved by also noting which which `$rule` was responsible for each classification, and keeping a running total of both the number of correct and incorrect classifications resulting from the application of each rule. At the end of a data file, your program should (must) dump a summary of both the *utilization* of each rule (the number of times each one is used and the percentage of the total), as well as the *effectiveness* of each rule (the number and percentage of the times that a rule's classification was correct and incorrect when the rule is used). Finally, the program should print the overall total of correct and incorrect classifications, on a line by itself in the exact following format:

`### HW1A` *your-username* `- OVERALL CORRECT: 1706 = 85.3%   INCORRECT: 294 = 14.7%.`

When developing your program, you should periodically run it on the full training data set for quantitative feedback on its current performance. We will also evaluate your program by running it on 9,000 examples of held out test data, in the same format as the training data but previously unseen.

Part 2: Text Segment Classification

Much of the text encountered in real-world NLP systems is intermixed with non-textual components such as tables, figures, formulae, and email/netnews headers. Text itself may be standard paragraph style prose or specialized textual segments such as headlines or section headers, addresses, quoted text or email signature blocks. It is useful to distinguish these different segments, both for processing in IR or message routing systems and for obtaining clean prose as training data for language models.

The goal of this part of the assignment is to label each line in a text file with the segment type of the text block the line appears in. For example:

```
NNHEAD  From: desmedt@ruls40.Berkeley.EDU (Koenraad De Smedt)
NNHEAD  Newsgroups: comp.ai.nat-lang
NNHEAD  Subject: CFP: 5th European Workshop on Natural Language Generation
NNHEAD  Date: 24 Oct 1994 09:36:40 GMT
NNHEAD  Organization: Leiden University
NNHEAD  Message-ID: <38fv78$9du@highway.LeidenUniv.nl>
NNHEAD  Keywords: Natural Language Generation Workshop
#BLANK#
HEADL                           CALL FOR PAPERS
#BLANK#
HEADL         5th European Workshop on Natural Language Generation
#BLANK#
HEADL                            20-23 May 1995
HEADL                       Leiden, The Netherlands
#BLANK#
PTEXT This workshop aims to bring together researchers interested in Natural
PTEXT Language Generation from such different perspectives as linguistics,
PTEXT artificial intelligence, psychology, and engineering.  The meeting
PTEXT continues the tradition of a series of workshops held biannually in
PTEXT Europe (Royaumont, 1987; Edinburgh, 1989; Judenstein, 1991 and Pisa,
PTEXT 1993) but open to researchers from all over the world.
#BLANK#
HEADL                         Programme
#BLANK#
PTEXT Papers, posters and demonstrations are invited on original and
PTEXT substantial work related to the automatic generation of natural
PTEXT language, including computer linguistics research, artificial
PTEXT intelligence methods, computer models of human language processing,
#BLANK#
PTEXT   All contributions should be sent BEFORE 1 JANUARY 1995 to the
PTEXT   Programme Chairman at the following address:
#BLANK#
ADDRESS               Philippe Blache
ADDRESS               2LC - CNRS
ADDRESS               1361 route des Lucioles
ADDRESS               F-06560  Sophia Antipolis
ADDRESS               tel : +33 92.96.73.98
ADDRESS               fax : +33 93.65.29.27
ADDRESS               e-mail : pb@llaor.unice.fr
#BLANK#
#BLANK#
QUOTED  > SJC (San Jose, CA) has an open observation deck on its older terminal
QUOTED  > A.  I have not used this terminal in quite some time, so I don't know
QUOTED  > if sightseers still have access to it.  The problem with this deck was
QUOTED  > that the @@#$^#$%^& restaurant would block your view just as the jets
QUOTED  > were getting off the ground.  Nevertheless, you could still watch the
QUOTED  > planes taxi and then accelerate from the start of the runway.
QUOTED  >
QUOTED  > Why is it that the newer terminals no longer have these outdoor viewing
QUOTED  > areas?  Security, I suppose.  A sad sign of our times.
QUOTED  >
#BLANK#
#BLANK#
SIG                                 ``'''
SIG                                 (0 0)
SIG        +-------------------------OOo--(_)--oOO-------------------------+
SIG        |                              |                                |
SIG        | Rajeev Agarwal               |  "Hanging in there..."         |
SIG        |                              |                                |
SIG        | Dept. of Computer Science    | (601) 325-8073 or 2756 (Off.) |
SIG        | Mississippi State University  | (601) 325-7506 (Lab)          |
SIG        +------------------------------+--------------------------------+
SIG                                (_) (_)
```

A description of the different segment types and examples of them may be found in the directory `$CS466/hw1/segment`.

You are provided with a program `make_block.prl` that adds the designator `#BLANK#` between text segments. A recommended strategy is to read in all lines of a text segment between `#BLANK#`'s into an array and classify the segment as a single unit. Many segments may be classified by the presence of relatively simple patterns within the segment, such as `Message-ID:` or `From:` in a netnews header `NNHEAD`.

Because of the difficulty in formalizing precise standards for segment classification, the standards for correctness will be quite liberal. The major purpose of this exercise will be to get practice with Perl pattern recognition and observe the complexity of real-world text streams. The standards for classification are located in the file:
`$CS466/hw1/segment/standards`, but one need not worry about conforming too closely. Priority will be placed on the creativity and completeness of the segment classifier, not on conforming to any arbitrary definitions of what constitutes a signature or table, for example.

Overall performance, as well as rule-by-rule effectiveness, should be reported in the same manner as described at the end of section 1.

Evaluation:

Submissions will be evaluated as follows:

- 50% - PART 1: Quality, completeness and creativity of algorithm
- 5% - PART 1: Performance on training data
- 20% - PART 1: Performance on independent test data
- 20% - PART 2: Quality, completeness and creativity of algorithm
- 5% - PART 2: Performance on independent test data

What to Turn in:

Please hand in your assignment both in hard copy and electronically. For hard copy submissions, please provide printouts of your code, commented sufficiently so that we can understand what you did solely from reading the comments. It is not necessary to include printouts of any large secondary data files, such as new class members.

To simplify the electronic submission of code and data files, you are required to submit a compressed tar file of your homeworks via the class's electronic submission program.

Here are the steps:

1. Create two subdirectories called `hw1a` and `hw1b`
2. Place all your perl code and modified data files in the appropriate subdirectory for Part 1 (hw1a) or Part 2 (hw1b). Your programs should be written so that if they are copied to another `hw1a` directory without read access to your account, they will still run. Thus any directory path hardwired into your program should either reference the common course directory (e.g. `$COMMON_PATH = "/users/rtfm2/cs466/hw1/classes"`) or the current directory (e.g. `$MY_PATH = "."`, which would reference any supporting files you've copied to hw1a or hw1b), but never `$MY_PATH = "/users/rtfm2/jdoe/hw1a"` (where the files may change or be inaccessible to the graders).
3. Back up one level (cd ..) so that you are not inside the hw1 directory, but instead its parent directory.
4. Type:

   ```
   tar -cvf - hw1a | gzip > jdoe.hw1a.tar.gz
   tar -cvf - hw1b | gzip > jdoe.hw1b.tar.gz
   ```

   .. replacing "jdoe" with your username.

Now, load your favorite web browser. Go to:
`http://www.ugrad.cs.jhu.edu/cgi-bin/cgiwrap/cs466/submit.pl`

Your password details will be emailed to you. Use this password for HW1 and all future submissions.

Go to the first form (submit a program). You need to enter your login, your password, choose the assignment you are submitting, pick the file that you are submitting (it will pop up a file browser and you have to find the myusername.hw1a.tar.gz file that you just created), and then type the name of the file again in the last box. Then click Continue. It will now suck the file off your computer and drop it into your directory on our account.

Finally, you will probably want to check the file(s) you have submitted. To do this, go to the second form (view files I submitted) and enter your login, password and the assignment you want to check on. It will show you a listing of all the files that you have submitted for that assignment. Additionally, if for each file that is .tar or .tar.gz it will show you all the files that are inside that compressed file, so you will know that you compressed it correctly.

If you submit the same file more than once, it will over-write the old file, unless you give it a new name. So if you want to submit a correction or update and it is after the deadline, submit just the file(s) that have changed and not the whole tar.gz file again so we will know what you submitted and when.

Please contact one of the TAs if you have any problems with or questions about the submission program.