ASSIGNMENT 1
End-of-Sentence Detection and Text Segment Classification

Due date: Thursday, February 22, 2018 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 | 1 viewpoint, as David C. Robinson has recently shown, t |
| NOT-END-OF-SENTENCE | 128 | true; 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 Kulick, 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 carry |
| END-OF-SENTENCE | 210 | system at 25 and 50 deg C. Isotherms consist of five bran |
| END-OF-SENTENCE | 239 | C while not at 40 deg C. Minima on the S/sub Fu/ vs. C/ |
| END-OF-SENTENCE | 247 | ellulases. Culture of C. thermocellum will be optimized |
| END-OF-SENTENCE | 255 | the cellulase genes of C. cellulolyticum and those from t |
| END-OF-SENTENCE | 258 | anger from 200 to 300 C. A system developed by the autho |
| END-OF-SENTENCE | 262 | course 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 | 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 the masters network (e.g. gradx) and /home/1/yarowsky/cs466 on the undergraduate network.

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:

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

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

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:

From: demedt@rules40.Berkeley.EDU (Koenraad De Smedt)
Newsgroups: comp.ai.mat-lang
Subject: CFP: 5th European Workshop on Natural Language Generation
Date: 24 Oct 1994 09:30:40 GMT
Organization: Leiden University
Message-ID: <3Sf7v7@stu08highway.Leiden|niv.nl>
Keywords: Natural Language Generation Workshop

CALL FOR PAPERS

5th European Workshop on Natural Language Generation
20-23 May 1995
Leiden, The Netherlands

This workshop aims to bring together researchers interested in Natural
Language Generation from such different perspectives as linguistics,
artificial intelligence, psychology, and engineering. The meeting
continues the tradition of a series of workshops held biannually in
Europe (Royaumont, 1987; Edinburgh, 1989; Jerusalem, 1991 and Pisa,
1993) but open to researchers from all over the world.

Programme

Papers, posters and demonstrations are invited on original and
substantial work related to the automatic generation of natural
language, including computer linguistics research, artificial
intelligence methods, computer models of human language processing,
All contributions should be sent BEFORE 1 JANUARY 1996 to the
Programme Chairman at the following address:

Philippe Blache
2LC - CNRS
1361 route des Lucioles
F-06560 Sophia Antipolis
tel : +33 92.96.73.98
fax : +33 93.65.29.27
e-mail : pb@laor.unice.fr

SJC (San Jose, CA) has an open observation deck on its older terminal
A. I have not used this terminal in quite some time, so I don’t know
if sightseers still have access to it. The problem with this deck was
that the O.J.S.’s restaurant would block your view just as the jets
were getting off the ground. Nevertheless, you could still watch the
planes taxi and then accelerate from the start of the runway.
Why is it that the newer terminals no longer have these outdoor viewing
areas? Security, I suppose. A sad sign of our times.

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.pl 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 effectivness, 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:

```bash
    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.