Introduction

Processing text may seem easy for humans like ourselves but is non-trivial for computers. In this assignment, you will develop classifiers for two text analysis tasks: end-of-sentence detection and text segment classification. We’ll approach these tasks from a machine learning perspective.

One major paradigm in machine learning is supervised learning. In supervised learning, your model learns from example input-output pairs to predict an output from a given input. Within supervised learning, classification is the task of predicting a label corresponding to some input. For example, in gender classification, you are given a sentence and you want to predict the gender of the person who wrote it.

For the two tasks in this homework, we will give you some training data to train a model or develop an algorithm. This is not a ML class, so feel free to apply off-the-shelf machine learning packages. As discussed in class, we recommend you to split your data into a train set and development set for testing purposes. We will evaluate your algorithms on a test set that you will not have seen.

Part 1: End-of-Sentence Detection

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

<table>
<thead>
<tr>
<th>Example</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 viewpoint, as David C. Robinson has recently shown, t</td>
<td>EOS</td>
</tr>
<tr>
<td>for Arthur C. Clarke, Gentry Lee (Bantam)</td>
<td>NEOS</td>
</tr>
<tr>
<td>The group led by C. Delores Tucker, head of the NA</td>
<td>EOS</td>
</tr>
<tr>
<td>ence and Electronics; C. Scott Kulicke, on behalf of Se</td>
<td>EOS</td>
</tr>
<tr>
<td>Committee chaired by C. Rubbia. The report covers stat</td>
<td>EOS</td>
</tr>
<tr>
<td>occurs at 440 degrees C. A hydrogenation test was carrie</td>
<td>EOS</td>
</tr>
<tr>
<td>str system at 25 and 50 deg C. Isotherms consist of five branc</td>
<td>EOS</td>
</tr>
<tr>
<td>C while not at 40 deg C. Minima on the S/sub Pu / vs. C/</td>
<td>EOS</td>
</tr>
<tr>
<td>ellulas. Culture of C. thermocellum will be optimized</td>
<td>EOS</td>
</tr>
<tr>
<td>the cellulase genes of C. cellulolyticum and those from t</td>
<td>EOS</td>
</tr>
<tr>
<td>ang from 200 to 300 C. A system developed by the autho</td>
<td>EOS</td>
</tr>
<tr>
<td>course on programming in C. Finally, those who are interest</td>
<td>EOS</td>
</tr>
<tr>
<td>a house on 2213 Perry Dr. Then the Thomases were seen in</td>
<td>EOS</td>
</tr>
<tr>
<td>. &lt;P&gt; Early in 1980 Dr. Thomas B. Reed of SERI and Pro</td>
<td>EOS</td>
</tr>
</tbody>
</table>

To help you develop a classifier for this distinction, we provide an example set of 45,000 periods and their surrounding context. Each example is 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 hw1/sent.train.
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 \( \pm 3 \)-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>NEOS</td>
<td>119</td>
<td>as</td>
<td>David</td>
<td>C</td>
<td>.</td>
<td>Robinson</td>
<td>has</td>
</tr>
<tr>
<td>NEOS</td>
<td>128</td>
<td>by</td>
<td>Arthur</td>
<td>C</td>
<td>.</td>
<td>Clarke</td>
<td></td>
</tr>
<tr>
<td>NEOS</td>
<td>136</td>
<td>led</td>
<td>by</td>
<td>C</td>
<td>.</td>
<td>Delores</td>
<td>Tucker</td>
</tr>
<tr>
<td>NEOS</td>
<td>147</td>
<td>electronics;</td>
<td>C</td>
<td>.</td>
<td>Scott</td>
<td>Kulicke</td>
<td></td>
</tr>
<tr>
<td>NEOS</td>
<td>184</td>
<td>chaired</td>
<td>by</td>
<td>C</td>
<td>.</td>
<td>Rubbia</td>
<td></td>
</tr>
<tr>
<td>EOS</td>
<td>192</td>
<td>440</td>
<td>degrees</td>
<td>C</td>
<td>.</td>
<td>A</td>
<td>hydrogenation</td>
</tr>
<tr>
<td>EOS</td>
<td>210</td>
<td>50</td>
<td>deg</td>
<td>C</td>
<td>.</td>
<td>Isotherms</td>
<td>consist</td>
</tr>
<tr>
<td>EOS</td>
<td>239</td>
<td>40</td>
<td>deg</td>
<td>C</td>
<td>.</td>
<td>Minima</td>
<td>on</td>
</tr>
<tr>
<td>NEOS</td>
<td>247</td>
<td>Culture</td>
<td>of</td>
<td>C</td>
<td>.</td>
<td>thermocellum</td>
<td>will</td>
</tr>
<tr>
<td>NEOS</td>
<td>255</td>
<td>genes</td>
<td>of</td>
<td>C</td>
<td>.</td>
<td>cellulolyticum</td>
<td>and</td>
</tr>
<tr>
<td>EOS</td>
<td>258</td>
<td>to</td>
<td>300</td>
<td>C</td>
<td>.</td>
<td>A</td>
<td>system</td>
</tr>
<tr>
<td>EOS</td>
<td>262</td>
<td>programming</td>
<td>in</td>
<td>C</td>
<td>.</td>
<td>Finally</td>
<td></td>
</tr>
<tr>
<td>EOS</td>
<td>300</td>
<td>2213</td>
<td>Perry</td>
<td>Dr</td>
<td>.</td>
<td>Then</td>
<td>the</td>
</tr>
<tr>
<td>NEOS</td>
<td>330</td>
<td>in</td>
<td>1980</td>
<td>Dr</td>
<td>.</td>
<td>Thomas</td>
<td>B</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 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.

We will test the effectiveness of your classifier on another file with the identical format as the training data. For maximum fairness of the test, you will not be able to see this test data in advance.

To assist you, we have included other files containing wordlists of abbreviations, titles, unlikely proper nouns, and other terms. You may find these useful when building your classifier.

To simplify testing and grading, your code must be runnable using the following command:

```
python hw1a.py --train traindata --test testdata --output outputfile
```

where `traindata` is the path to the training data, `testdata` is the path to the test file, and `outputfile` is a file that will contain the line-by-line classification from your algorithm.

We have provided starter code that conforms to this specification and already handles loading the data. In addition, it will compute and print out accuracy on your test data. If your code needs to load any other
files, please use relative paths, not absolute paths. We will run your program using Python 3.7.2. If you use any external packages, let us know in your writeup (Part 3) so we can install them if needed.

We included two sample classifier implementations. The first one (EOSClassifier1) trains a model to always predict the most common label that it has seen in the training data. If you use sent.train as both the training and test data, using the following command,

   python hw1a.py --train sent.train --test sent.train

you’ll see that the performance on the training data is over 90%. This reveals that the data is heavily unbalanced. The second classifier (EOSClassifier2) doesn’t look at the training data at all! It simply examines the word immediately to the left of the period. If this word is an abbreviation in the provided abbreviations wordlist, then it predicts 'NEOS', otherwise 'EOS'. This classifier gets 95% accuracy on the training data. Nevertheless, there is still room to improve. However, beware of overfitting. Just because your classifier gets 99.9% accuracy on the training data does not mean it will perform similarly on the test data. As mentioned before, we recommend that you split your training data into a training set and development set and only use the development set for testing.

You’ll notice that the starter code uses scikit-learn. Feel free to use scikit-learn or your favorite ML package. One benefit of scikit-learn is that you can easily try out many different ML models, and we encourage you to do so. The official tutorial can be found at https://scikit-learn.org/stable/tutorial/basic/tutorial.html

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.

Here is a sample of the data:

```
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#
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 1995 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@llaor.unice.fr

SJC (San Jose, CA) has an open observation deck on its older terminal. 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 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.

In this part of the assignment, you will develop a classifier that labels each line and each segment with the segment type (left column), where a segment is the concatenation of adjacent lines of the same segment type. The provided code already combines adjacent lines into segments for you. A description of the different segment types and examples of them may be found in the directory hw1/segment. 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, though other segments may be harder to distinguish.

The standards for classification are located in the file: hw1/segment/standards, but don’t 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.

We have provided starter code that trains a decision tree classifier. We preprocess each input into a feature vector of four features:

- number of characters
- number of characters after trimming whitespace
• number of words
• 1 or 0 depending if a > is present

With only these four features, this classifier gets 85% accuracy on the training data. This method of manual feature extraction is one possible way to approach this problem. You may also consider using an algorithm to automatically learn features.

Similar to Part 1, we must be able to run your code using the following commands:

```
python hw1b.py --train traindata --test testdata --output outputfile --format line
python hw1b.py --train traindata --test testdata --output outputfile --format segment
```

where the --output switch indicates whether your program should classify by line or by segment.

**Part 3: Writeup**

Create a short writeup documenting your algorithms for Parts 1 and 2. You should describe the algorithms you developed and/or the models you used, report their performance on the training data, and comment on anything else you found interesting, e.g. what features seemed to work well/poorly, other methods you tried but didn’t work, why do you think something worked or not, etc.

**Evaluation:**

Submissions will be evaluated as follows:

| Part 1: Quality, completeness, and creativity of algorithm/features | 30% |
| Part 1: Performance on training data | 5% |
| Part 1: Performance on independent test data | 20% |

| Part 2: Quality, completeness, and creativity of algorithm/features | 20% |
| Part 2: Performance on training data | 5% |
| Part 2: Performance on independent test data | 15% |

| Part 3: Writeup | 5% |

**Submission**

We are using Gradescope for submitting assignments. Our course code is MPYGGD. Please submit a gzipped tar file of a folder containing the files for this assignment. To do this:

1. Create a folder named jhuid-hw1 (replacing jhuid with your JHU ID) containing hw1a.py, hw1b.py, writeup.txt, and all other files (including the input files we provided) that your program needs.

2. Compress this folder. If your folder is called abc123-hw1, you can compress it like this:

```
tar czf abc123-hw1.tgz abc123-hw1
```

If you have any questions, feel free to ask a TA or post on Piazza.