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. You will extract features from the input and experiment with different ML models. We will evaluate your algorithms on a test set that you will not have seen.

Note that Part A is due in two weeks and 1b is due in three weeks. You should start early on Part A, especially if you do not have any experience with Scikit-Learn.

Part A: 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:

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 ±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 Külicke
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 . celluloyticum 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. 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 provided starter code and some data files containing wordlists of abbreviations, titles, unlikely proper nouns, and other terms. You may find these useful when building your classifier.

The starter code extracts features from each line and trains a decision tree classifier. Actually, the training data already provides three features (columns 10, 11, and 12), so we simply add them to the feature vector. We have also started you off with three other features: an indicator of whether the word to the left of the period is in the provided abbreviation list, the length of the word to the left of the period, and an indicator of whether the word to the right of the period is capitalized. Note that these features must be numeric. Try uncommenting some of these features to see how they affect your model’s performance, and think about what other features can distinguish a period from an abbreviation marker.

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.

The starter code 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.

Since you cannot see the test data, we recommend you hold out a portion of your training data as development data, which your model can use to simulate test data. We provided a script splitdata.py which splits your data, and you can run it as follow:

```
python splitdata.py sent
```

This will create two files, sent.train1 and sent.dev1. You are encouraged to try different train/dev splits. Then you can train and test your model with the following command:

```
python hw1a.py --train sent.train1 --test sent.dev1 --output sent.out
```

Some other command line arguments may be helpful in debugging your code:

- **--errors errorfile** will create a file containing the correct label, your model’s classification, and the input for every line in the test data that your model got wrong.
- **--report** shows a precision/recall table for each label.

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](https://scikit-learn.org/stable/tutorial/basic/tutorial.html)

### Part B: 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:

```plaintext
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
```
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; Judenstein, 1991 and Pisa, 1993) but open to researchers from all over the world.

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, and other contributions. 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

We have provided some additional features, including the number of spaces and number of capital words. You may also consider structural features: in contrast to end-of-sentence classification in Part A, here the input examples are not independent of each other. For example, a PTEXT is likely to be followed by another PTEXT. You may also consider using an algorithm to automatically learn features.

Similar to Part A, you can split your training data using the `splitdata.py` script, and 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 `--format` switch indicates whether your program should classify by line or by segment.

**Part 3: Writeups**

Create two short writeups documenting your algorithms for Parts A and B. 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:

<table>
<thead>
<tr>
<th></th>
<th>Part A: Quality, completeness, and creativity of algorithm/features</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Part A: Performance on training data</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>Part A: Performance on independent test data</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Part B: Quality, completeness, and creativity of algorithm/features</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>Part B: Performance on training data</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>Part B: Performance on independent test data</td>
<td>15%</td>
</tr>
</tbody>
</table>

|                  | Part 3: Writeup                                                   | 5%  |

**Submission**

We are using Gradescope for submitting assignments. Our course code is WY8X6G. There will be two submission pages for this assignment: One for Part A and one for Part B. Note that Part A and Part B both have writeups. For Part A, please submit only `hw1a.py writeup_a.txt`. For Part B, please submit only `hw1b.py writeup_b.txt`.

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