601.466/666 (Spring 2024) (Spring 2025)

Assignment 1: Machine Learning for Text Analysis

Due Dates: Part A on Friday, February 7 at 4:00 PM (Baltimore time)
and Part B on Friday, February 14 at 4:00 PM (Baltimore time)

Due Dates: Part A on Thursday, February 8 at 2:45 PM (Baltimore Time)
and Part B on Thursday, February 15 at 2:45 PM (Baltimore Time)

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:

```
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, 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           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`. 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`

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:

```
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       +-------------------------00o--(_)--o00-------------------------+
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                                    (_) (_)
```

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:

Part A: Quality, completeness, and creativity of algorithm/features	30%
Part A: Performance on training data	5%
Part A: Performance on independent test data	20%
Part B: Quality, completeness, and creativity of algorithm/features	20%
Part B: Performance on training data	5%
Part B: Performance on independent test data	15%
Part 3: Writeup	5%

Submission

We are using Gradescope for submitting assignments. Our course code is 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.