Assignment 4

Introduction to Natural Language Processing
Fall 2017
Total points: 100
Issued: 11/06/2017 Due: 11/13/2017 (part 1), 11/20/2017 (parts 2-3)

All the code has to be your own (exceptions to this rule are specifically noted below). The code must run on the CAEN environment using Python 2.7 without additional installation or additional files (except for the data files specified in the assignment).

You can discuss the assignment with others, but the code is to be written individually. You are to abide by the University of Michigan/Engineering honor code; violations will be reported to the Honor Council.

4.1. [20 points] Dialog Data Collection

Using an interface created by Prof. Walter Lasecki and his CRO-MA lab, you will have to participate in five mock dialog sessions for academic advising. A dialog session requires two participants; you can pair up with people of your own choosing (you can work with the same person for all five dialogs, or with different persons; you can work with someone from the class or from outside the class). In a dialog session, you can play the role of either the advisor or the student. The file Assignment4.DialogDataGuidelines.pdf in the Files/ section on Canvas has detailed information on how to conduct the dialogs.

Please make sure you use your uniqname when creating an account. This is the only way in which we will be able to track down your contribution for this part of the assignment.

Note that this part of the assignment is due on 11/13. We will compile all the dialog sessions into a corpus that will form the basis for part 4.3.

4.2. [60 points] Naive Bayes Word Sense Disambiguation

Write a Python program *WSD.py* that implements the Naive Bayes algorithm for word sense disambiguation, as discussed in class. Specifically, your program will have to assign a given target word with its correct sense in a number of test examples. Please implement the Naive Bayes algorithm yourself, do not use scikit-learn (or other machine learning library).

You will train and test your program on a dataset consisting of textual examples for six nouns *plant*, *bass, crane, motion, palm, tank.* Each of these nouns has a corresponding file, provided in the WSD.zip archive under the Files/ section on Canvas, consisting of examples drawn from the British National Corpus, annotated with the correct meaning; each noun has two possible meanings. Consider for example the following instance from the file *plant.wsd*:

<instance id="plant.1000002" docsrc = "BNC/A0G">
<answer instance="plant.1000002" senseid="plant%living"/>
<context>
September 1991 1.30 You can win a great new patio Pippa Wood How to cope with a slope Bulbs <head>plant</head> now for spring blooms
</context>
</instance>

The target word is identified by the SGML tag <*head*>, and the sense corresponding to this particular instance is that of *plant%living*.

Programming guidelines:

Your program should perform the following steps:

- Take one argument consisting of the name of one file, which includes the annotated instances.
- Determine from the entire file the total number of instances and the possible sense labels.
- Create five folds, for a five-fold cross-validation evaluation of your Naive Bayes WSD implementation. Specifically, divide the total number of instances into five, round up to determine the number of instances in folds 1 through 4, and include the remaining instances in fold 5. E.g., if you have 122 total instances, you will have five folds with sizes 25, 25, 25, 25, and 22 respectively.
- Implement and run the Naive Bayes WSD algorithm using a five-fold cross-validation scheme. In each run, you will:
  (1) use one of the folds as your test data, and the remaining folds together as your training data (e.g., in the first run, use fold 1 as test, and folds 2 through 5 as training; etc.);
  (2) collect the counts you need from the training data, and use the Naive Bayes algorithm to predict the senseid-s for the instances in the test data;
  (3) evaluate the performance of your system by comparing the predictions made by your Naive Bayes word sense disambiguation system on the test data against the ground truth annotations (available as senseid-s in the test data).

Considerations for the Naive Bayes implementation:

- All the words found in the context of the target word will represent the features
- Address zero counts using add-one smoothing
- Work in log space, to avoid underflow due to the repeated multiplication of small numbers

The *WSD.py* program should be run using a command like this:
% *python WSD.py* <word>.wsd

where <*word*> will be replaced with each of the six nouns in the data.

The program should produce at the standard output the accuracies of the system (as a percentage) for each of the five folds, as well as the average accuracy. It should also generate a file called

<word>.wsd.out, which includes for each fold the id of the words in the test file along with the senseid predicted by the system. Clearly delineate each fold with a line like this "Fold 1", "Fold 2", etc. For instance, the following are examples of lines drawn from a *plant.wsd.out* file

Fold 1
plant.1000000 plant%factory
plant.1000001 plant%living
…
Fold 2
plant.1000041 plant%living
plant.1000042 plant%living
...

Write-up guidelines:

Create a text file called WSD.answers, and include the following information:

- For each of the six datasets (*bass.wsd, crane.wsd, motion.wsd, palm.wsd, plant.wsd, tank.wsd*) include the following information:
    - A line consisting of the name of the dataset
    - The accuracies of your Naive Bayes system for each of the five folds, as well as the average accuracy.
- For the word *plant*, identify three errors in the automatically sense tagged data, and analyse them (i.e., for each error, write one brief sentence describing the error, the possible reason for the error, and how it could be fixed)

4.3. [20 points] Dialog Act Classification

Dialog act classification is an important component of a dialog system; it helps the system determine what it should do next (e.g., ask a question, ask for a clarification). Using the training and test data provided in the *DialogAct.train* and *DialogAct.test* files respectively, train and evaluate a dialog act classifier. You are provided with two sample files, which you can use for the purpose of development. The final train/test files will only become available on 11/14.

For each advisor turn in the data, you will use the previous turn (statement) to extract features, and use the dialog act of the advisor as the label. For this part of the assignment, you will adapt the Naive Bayes classifier implementation from 4.2: train the classifier on the dialog acts in the training data, and use the classifier to predict the dialog acts in the test data. Similar to 4.2, the features consist of the words in the turns.

Assume the following example:
Student: I was hoping to take 280 next semester.
Advisor: [push-general-info-inform] You have to enroll in 183 before you can enroll in 280.
Student: So maybe I should take 183 instead?
Advisor: [push-tailored-info-suggest] Yes, I recommend that you take 183.

This will result in two learning instances:
Features extracted from "*I was hoping to take 280 next semester."* Label: push-general-info-inform
Features extracted from "*So maybe I should take 183 instead?*" Label: push-tailored-info-suggest

Programming guidelines:
Write a Python program *DialogAct.py* that implements the Naive Bayes algorithm to predict a dialog act. Your program should perform the following steps (most of the code will be borrowed from WSD.py implementation):

- Collect all the counts you need from your training data.
- Use this Naive Bayes classifier to predict the dialog acts in the test data.
- Apply the Naive Bayes classification on the test data.
- Evaluate the performance of your system by comparing the predictions made by your Naive Bayes classifier on the test data against the ground truth annotations (available as dialog act labels in the test data).

The *DialogAct.py* program should be run using a command like this:
% *python DialogAct.py* DialogAct.train DialogAct.test

The program should produce at the standard output the accuracy of the system, as a percentage. It should also generate a file called *DialogAct.test.out*, which includes all the turns in the test data and the predicted dialog acts next to each advisor turn. For instance, below is a sample output (in this case, for both advisor turns, the system made the same prediction):
Student: I was hoping to take 280 next semester.
[push-general-info-inform] Advisor: [push-general-info-inform] You have to enroll in 183 before you can enroll in 280.
Student: So maybe I should take 183 instead?
[push-general-info-inform] Advisor: [push-tailored-info-suggest] Yes, I recommend that you take 183.

Write-up guidelines:
Create a text file called DialogAct.answers, and include the following information:

- The accuracy of your system on the test data
- Identify three errors in the automatically predicted dialog acts, and analyse them (i.e., for each error, write one brief sentence describing the error, the possible reason for the error, and how it could be fixed)

General submission instructions:

- Include all the files for this assignment in a folder called *[your-uniqname].Assignment4/* Do not include the data files. This folder will contain the files WSD.py, WSD.answers, the six files *.wsd.out corresponding to the six nouns; DialogAct.py, DialogAct.answers, DialogAct.test.out.
- Archive the folder using zip and submit on Canvas by the due date.
- Include your name and uniqname in each program and in the *.answers file
- Make sure all your programs run correctly on the CAEN machines using Python 2.7.