# Assignment 4

Introduction to Natural Language Processing
Fall 2016
Total points: 110
Issued: 11/09/2016
Part 4.2: Due: 11/18/2016
Part 4.1 and 4.3: Due 11/29/2016

All the code has to be your own (exceptions to this rule are specifically noted below). The code must run on the CAEN environment 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.

Start early!

## [60 points] 4.1. Natural Language Understanding for Dialog Systems

Write a Python program *NLU.py* that uses a supervised method to extract class id-s (e.g., EECS 280) and names (e.g., data structures and algorithms) from an annotated dataset of student statements. For each token in the input text, your classifier will have to assign an I(nside), O(utside), or B(eginning) label, to indicate if the token belongs to a class id or a class name.

To illustrate the IOB notation, consider the following examples:

*The best class I took here was computer and network security.*
*<class*
*name=computer and network security*
*link=Computer and Network Security*
*taken=true*
*sentiment=positive>*

This corresponds to the following IOB annotation:
*The/O best/O class/O I/O took/O here/O was/O computer/B and/I network/I security/I ./O*

*I was in EECS 579 last semester.*
*<class*
*id=579*

*department=EECS*
*taken=true*
*semester=last semester>*

The IOB annotation:
*I/O was/O in/O EECS/O 579/B last/O semester/O ./O*

*My favorite class was probably EECS 498, computer and network security.*
*<class*
*id=498*
*department=EECS*
*sentiment=positive*
*name=computer and network security*
*link=Computer and Network Security*
*taken=true>*

The IOB annotation:
*My/O favorite/O class/O was/O probably/O EECS/O 498/B ,/O computer/B and/I network/I security/I ./O*

In order to generate an IOB notation for an example, after the text is tokenized, the values of the "id" and "name" fields found under the <class> tags are mapped onto the text. For each of the sequences found in the text, the first token in the sequence is labeled as B and the remaining tokens in the sequence are labeled as I. All the tokens outside the sequences are labeled as O.

Your classifier will have to label each token as I, O, or B. You will train your classifier on the tokens extracted from the training data, and test the classifier on the tokens from the test data. For each token, you will have to extract the following five features:

- the value of the token
- is token all uppercase?
- does token start with capital?
- length of token
- does the token consist only of numbers?

Additionally, implement at least three other features of your choice. If you want, you can use the eecs_dict that maps class numbers to class names, provided in dicts.py. The use of this dictionary is optional.

Notes:

- For the purpose of this assignment, you should ignore the <instructor> tags
- You should also ignore the texts that do not have a <class> tag associated with them

Programming guidelines:
Your program should perform the following steps:

- Transform the training and the test examples into the IOB notation, as described before.
- For each token, generate a feature vector consisting of the five features described before, plus at least three features of your choice.
- Train your system on the tokens from the training data and apply it on the tokens from the test data. Use a classifier of your choice from sk-learn.
- Compare the IOB tags predicted by your system on the test data against the correct (gold-standard) IOB tags and calculate the accuracy of your system.

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

The program should produce at the standard output the accuracy of the system, as a percentage. It should also generate a file called NLU.test.out, which includes the textual examples in the test file along with the IOB tags predicted for each token.

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

- The accuracy of your system on the test data
- A brief description of the three (or more) additional features that you implemented

## [20 points] 4.2. 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. In a dialog session, you can play the role of either the advisor or the student. The file MockAdvisingSession.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. Also, make sure you use the participants' uniqnames when creating a session id (when you play the role of the advisor). 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 11/18. We will compile all the dialog sessions into a corpus that will form the basis for part 4.3.

## [30 points] 4.3. 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. Note that the two data files will only become available on 11/21.

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 simply have to adapt the Naive Bayes classifier implementation from the third assignment: 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 the third assignment, 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 your implementation of Assignment 3):

- 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).

Considerations for the Naive Bayes implementation:

- All the words found in the previous turn (statement) will represent the features to be considered
- Address zero counts using add-one smoothing
- Work in log space, to avoid underflow due to the repeated multiplication of small numbers

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.

Write-up guidelines:

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

- The accuracy of your system on the test data

General submission instructions:

- Include all the files for this assignment in a folder called *[your-uniqname].Assignment4/* **Do not** include the data files.
  For instance, lahiri.Assignment4/ will contain NLU.py, NLU.answers, DialogActs.py, DialogActs.answers.
- 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 files.
- Make sure all your programs run correctly on the CAEN machines.