# Assignment 4
# Natural Language Processing
# Total points: 70
## Due Saturday, April 6 at 11:59 pm on Canvas

## Please Work Individually

All the code has to be your own. The code is to be written individually. Violations will be reported to the University.

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

Write a Python program *NLU.py* that uses a supervised method to extract classid-s (e.g., EECS 280) and names (e.g., data structures and algorithms) from an annotated dataset of student statements provided in the **NLU.train** and **NLU.test** files. 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:

❖ How complete your program is. Even if your program is not complete or you are getting compilation errors, you will get partial credit proportionally. Just mention **clearly and accurately** how far you got
❖ If your program is complete, the accuracy of your system on the test data
❖ If your program is complete, a brief description of the three (or more) additional features that you implemented

## [25 points] 4.2. 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.

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:

- ❖ How complete your program is. Even if your program is not complete or you are getting compilation errors, you will get partial credit proportionally. Just mention **clearly and accurately** how far you got
- ❖ If your program is complete, the accuracy of your system on the test data

## There will be two links for submission on Canvas:

1. Please save your full code as PDF **(plain text)** and submit it by itself.
2. Please Submit a zip file that includes all your files, ***NLY.py*** and ***NLU.answers***, ***DialogActs.py,*** and ***DialogActs.answers,*** but **do not** include the data files.
3. **Two screenshots** showing the two runs of the program for the two datasets (whether succeeded or failed), and the output or part of the output (as your screenshot allows). If a fewer number of screenshots shows the results for all datasets, it should be acceptable as well. **Please make sure the date is shown in the screenshots.**

## <u>General Grading Criteria:</u>

- Submitting all required files (even with a non-compiling code): 15/70.
- A **reasonable attempt** and a reasonable code that doesn't compile: 35-40/70.
- A **<u>reasonable program/code</u>,** which runs successfully, but doesn't give the correct output: (45-50/70) (ex: few mistakes in functions or metrics, accuracy rates not rational, etc.)
- A successful program with the correct output, but not fulfilling all requirements (ex: Write-up not complete, etc.):60/70.
- A program fulfilling all the requirements: 70/70.