# LING/CSCI 5832 Fall 2016 Homework 4 - Named Entity Recognition (NER)

Due **Tue Oct 25** before class (**2pm**) on D2L.

In this homework, you will implement a named entity recognition (NER) system and evaluate it. The idea is to use a classifier (preferably a support vector machine (SVM)) to greedily, sweeping left-to-right, classify each word into classes that signal whether that word is part of a named entity or not, and if so, what kind of entity. You should be familiar with the textbook chapter 22.1; draft chapter here.

The focus in this task is on **feature engineering**. That is, you should develop features that are useful for a classifier to make a decision about what named entity, if any, the current word is.

You can, and are encouraged to use the scikit-learn SVM classifier class (discussed in lecture) for training the SVM. You should also use the `DictVectorizer` class to extract features for each example into a dictionary of feature/value pairs, where the features are strings and the values can be just the integer `1`. These are then automatically converted into sparse feature vectors of zeroes and ones.

See here for a minimal example about training a classifier with scikit-learn's SVM class and the DictVectorizer.

## The data

We will use real data for this task. The data comes from the CoNLL 2003 shared task, which is a NER-annotated resource, split into three parts `eng.train` (for training), and `eng.testa` (for testing), and `eng.testb` (for additional testing).

The following illustrates the first two sentences in `eng.train`, the training data.

```
-DOCSTART- -X- O O

EU NNP I-NP I-ORG
rejects VBZ I-VP O
German JJ I-NP I-MISC
call NN I-NP O
to TO I-VP O
```

```
boycott VB I-VP O
British JJ I-NP I-MISC
lamb NN I-NP O
. . O O
```

The first "sentence" is a dummy sentence just signaling the start of the document. After that, sentences are given in a space-separated four-field format as follows:

- word
- POS tag
- syntactic chunk tag (you'll most likely want to ignore this)
- the named entity tag (what you'll learn to predict)

The named entity tags have the format `I-TYPE`, which means that the word is inside an entity of type `TYPE`. Only if two entities of the same type immediately follow each other will the first word of the second entity have a tag `B-TYPE` to show that it starts a new entity. A word with tag `O` is not part of a named entity. These are then the classes you want to assign to each word in the test corpus. If you look at `eng.train`, you'll find that list of possible classes is fairly short:

```
B-LOC
B-MISC
B-ORG
I-LOC
I-MISC
I-ORG
I-PER
O
```

You can treat all of these as just arbitrary labels that you learn to assign to tokens.

This is essentially all you need to know about the data format. If you want more information about the CoNLL data format, see the official shared task web site.

The CoNLL data you will need is here.

## Reading the data

To simplify your task, the following Python function can read in a file of the CoNLL format and create a list of sentences, each sentence being a list of annotated words with four fields:

```
def readconll(file):
    lines = [line.strip() for line in open(file)]
    while lines[-1] == '':  # Remove trailing empty lines
        lines.pop()
    s = [x.split('_') for x in '_'.join(lines).split('__')]  # Quick split corpus i
    return [[y.split() for y in x] for x in s]
```

The result now looks like the following:

```
>>> sentences = readconll('eng.train')
>>> sentences[0:3]
[[['-DOCSTART-', '-X-', 'O', 'O']],
 [['EU', 'NNP', 'I-NP', 'I-ORG'],
  ['rejects', 'VBZ', 'I-VP', 'O'],
  ['German', 'JJ', 'I-NP', 'I-MISC'],
  ['call', 'NN', 'I-NP', 'O'],
  ['to', 'TO', 'I-VP', 'O'],
  ['boycott', 'VB', 'I-VP', 'O'],
  ['British', 'JJ', 'I-NP', 'I-MISC'],
  ['lamb', 'NN', 'I-NP', 'O'],
  ['.', '.', 'O', 'O']],
 [['Peter', 'NNP', 'I-NP', 'I-PER'], ['Blackburn', 'NNP', 'I-NP', 'I-PER']]]
```

## What to do

Your script should

- train the classifier on `eng.train` using features that you develop.
- read in `eng.testa` and classify each word in it into one of the above eight classes.
- output a file `eng.guessa` that is identical to `eng.testa`, except it adds a new field last which is the result of your classifier.

For example, if `eng.testa` contains material like this:

```
West NNP I-NP I-MISC
Indian NNP I-NP I-MISC
all-rounder NN I-NP O
Phil NNP I-NP I-PER
Simmons NNP I-NP I-PER
```

Your `eng.guessa` output could look like this:

```
West NNP I-NP I-MISC O
Indian NNP I-NP I-MISC I-MISC
all-rounder NN I-NP O O
Phil NNP I-NP I-PER I-PER
Simmons NNP I-NP I-PER I-PER
```

Note the added fifth field, which is the output of your classifier.

Note also that each line in `eng.guessa` must have five fields; this includes the dummy first line, which should look like:

```
-DOCSTART- -X- O O O
```

## Evaluation

To evaluate your NER system, there is a script `conlleval.perl` . This script reads the output in the five-field format above, and produces an evaluation of overall accuracy, precision, recall, and F-score, and also P/R/F broken down by type of NER-tag. This evaluation looks something like the following with the command `perl conlleval.perl < eng.guessa`

```
processed 51578 tokens with 5942 phrases; found: 6106 phrases; correct: 5056.
accuracy:  97.66%; precision:  82.80%; recall:  85.09%; FB1:  83.93
              LOC: precision:  88.18%; recall:  88.95%; FB1:  88.56  1853
             MISC: precision:  83.43%; recall:  81.89%; FB1:  82.65  905
              ORG: precision:  71.72%; recall:  75.84%; FB1:  73.72  1418
              PER: precision:  85.49%; recall:  89.58%; FB1:  87.49  1930
```

The figure (which here is 83.93) representing the overall F-score is what you should be most interested in.

## What should my results look like?

You should beat the baseline classification accuracy that was provided by the CoNLL 2003 organizers, which is as follows:

```
    eng.testa: precision:  78.33%; recall:  65.23%; FB1:  71.18
```

That is, your F-score, as given by `conlleval.perl` for your output `eng.guessa` should be above `71.18`. This is very easy to achieve using obvious features such as those outlined in the textbook, section 22.1. If your F-score is not above this you should suspect serious problems with your code.

## Bonus points for excellent classification

You will get your homework score multiplied by a factor 1.3x-2.0x **if you beat the CoNLL-2003 top 3 participating systems** in F-score. You can see the results here. If you beat the best system (F>88.76), your homework score is multiplied by 2.0x. If you land second (F>88.31), your multiplier is 1.5x, and 1.3x for third place (F>86.07). Note that these results are on the alternate test set `eng.testb`. So, if you plan on receiving bonus points you should test on both `eng.testa` and `eng.testb`. Otherwise you can just evaluate on `eng.testa`. For hints on what kinds of features might be useful for achieving these kinds of scores, you can look at the paper by the 2nd highest scoring team.

## What to hand in

You should hand in an archive `firstname.lastname.5832.hw4.tar.gz` or `firstname.lastname.5832.hw4.zip` consisting at least of a Python file `firstname.lastname.5832.hw4.py` that runs as decribed if the `eng.train` and `eng.testa` files are located in the same directory and produces an output `eng.guessa`. We will run your code first, and then `conlleval.perl` as follows:

```
perl conlleval.perl < eng.guessa
```

and expect that `eng.guessa` is valid in formatting and that the evaluation script runs without trouble.

Put a `README` in the archive if you did anything outside a straightforward implementation of the above, or if you're claiming bonus points (don't claim bonus points without actually testing your code against `eng.testb`).

If you want **bonus points** you must also output a file `eng.guessb` from `eng.testb` since that is what we compare with for the bonus points.

## Notes

Note that while the training data contains the correct classes for each word, you **cannot** use future labels as features for your classifier. The reason for this is obvious: when you're classifying left-to-right, you don't have access to the information concerning the

following label, so why train on it. Previous labels can of course freely be used as features. You may also freely use the previous and future POS-tags as features in your classifier.