CIS 410/510: Natural Language Processing
Fall 2019

due 11:55pm Friday, Nov. 16, 2019

1 Problem 1

[30 points]

This is the second programming assignment for the NLP class.

We will build a name tagger (for the Named Entity Recognition task – NER) in this assignment.

Name tagging is much more domain-dependent than part-of-speech tagging. At the same time, it is easier to produce moderate-quality training corpora than for the other tasks. As a result, there are a wide range of NER corpora with similar but not identical tags sets. We will use a common corpus for NER in this assignment. This corpus identifies three types of names: person, organization, and location, and a fourth category, MISC (miscellaneous) for other types of names. Remeber that we can cast this as a sequence labeling problem using the BIO annotation schema (i.e., you would have 2x4 + 1 = 9 tags for this dataset).

As before, training, development, and test corpora will be provided and will follow the usual pattern with two more columns than the previous assignment. The training data will have four items per line (tab separated): the token, the part-of-speech, the BIO chunk tag, and the name tag. The development data will come in two forms: the input to tagging will have a token, part-of-speech, and chunk tag on each line; the input to scoring will have a token and name tag on each line. Finally, the test data we provide will have three items per line: the token, ppart-of-speech, and chunk tag. The response you return should have two (tab-separated) items per line: the token and name tag predicted by your system.

We are providing the part-of-speech and chunk information for your convenience (in case you want to use it). There is no guarantee that they will be helpful features. Note that these two columns are produced by automatic taggers and not checked by hand.

Like the data for Assignment 2, this data is made available to you as a resource through Canvas in the form of a zip file, NAME_CORPUS_FOR_STUDENTS.zip. This zip file contains

- train.pos-chunk-name: words, POS, chunk, and name tags for training corpus
- dev.pos-chunk: words, POS, and chunk tags for development corpus
- dev.name: words and name tags for development corpus (for scorer)

- test.pos-chunk: words and POS and chunk tags for test corpus

As the classifier to make these predictions, we recommend that you use a Maximum Entropy Markov Model, as discussed in class, and specifically the opennlp MaxEnt[1] (ver. 3.0.0) package in Java. The package is relatively well documented and makes the train/test cycle quite simple. (Note: there are more recent releases of OpenNLP through Apache, but they come with minimal documentation and so are only for the adventurous. For Python aficionados, there is a full MaxEnt systtem available in NLTK.)

With this package, you could proceed as follows:

1. Prepare a program `FeatureBuilder` that will read in one of the .pos-chunk-name or .pos-chunk files we provide and generate a feature vector for each token. Each line written by this program will consist of the token and zero or more additional features, separated by tabs:

   ```
   token feature=value feature=value ...
   ```

   For the .pos-chunk-name file, this should be followed by a tab and the tag to be assigned to the current token:

   ```
   token feature=value feature=value ...  tag
   ```

   There may not be any whitespace within a `feature=value`

   The features you generate may involve the previous, current, and next tokens, parts-of-speech and chunking tags, as well as combinations of these. They may also involve the tag assigned to the previous token, but with one complication. When processing a .pos-chunk file, `FeatureBuilder` doesn't know what tags will be assigned; it should use the symbol "@@" to refer to the previous tag.

   Apply this program to the .pos-chunk-name and .pos-chunk files, producing "feature-enhanced" files.

2. Compile and run `MEtrain.java`[2], giving it the feature-enhanced training file as input; it will produce a MaxEnt model. `MEtrain` and `MEtest` use the maxent and trove packages, so you must include the corresponding jar files, `maxent-3.0.0.jar`[3] and `trove.jar`[4], on the classpath when you compile and run.

3. Compile `MEtag.java`[5] and then use this program and the model you trained to tag the feature-enhanced development corpus.

4. Use the provided scorer[6] (Python3 coded) to score the result. If you are not satisfied with the result, modify `FeatureBuilder` to add or change features and repeat the process.

[1] `http://maxent.sourceforge.net/howto.html`
[2] `https://ix.cs.uoregon.edu/~thien/nlpclass/MEtrain.java`
[3] `https://ix.cs.uoregon.edu/~thien/nlpclass/maxent-3.0.0.jar`
[4] `https://ix.cs.uoregon.edu/~thien/nlpclass/trove.jar`
[5] `https://ix.cs.uoregon.edu/~thien/nlpclass/MEtag.java`
[6] `https://ix.cs.uoregon.edu/~thien/nlpclass/score.name.py`

5. When you are satisfied, apply the tagger to the test corpus and submit the result.

With this arrangement, the `FeatureBuilder` program can be written in any programming language.

The important question is what features to compute on the word sequence. You probably have some intuition regarding good indicators of person, organization, and place names; you can make a few of these into features. For example, a name which is followed by a comma and a state or country name is probably the name of a city. You may also use as features lists which can be found on the web, such as lists of common names[7] [8].

The minimal requirement for this assignment involves trying several feature sets, determining their scores on the dev corpus, then picking the best and tagging the test corpus. The assignment you submit through Canvas should include three attachments:

- a write-up (in the PDF format named "write-up.pdf"). This will describe the feature sets you tried, reporting the score on the development corpus. All the instructions to running the code or so should be put in the end of this file as well.
- your code (preferably put in a separate directory).
- the output of your code when run on the test data; this should have the file name "test.name" and the same format as the "dev.name" file.

We anticipate giving 30 points out for meeting the minimal requirement (i.e., full credit) and up to 10 additional points (bonus) for more elaborate submissions (i.e., trying more features/methods to improve the model that can lead to better performance on the development set based on your experiments) One example of the more elaborate features involve the Brown word clusters[9] [10]. The submissions with top performance on the test set will be rewarded 10 additional points/bonus (so the maximal point is 50). In particular, we will take your submitted output file for the test set, run the scorer that compares your submitted output file with the golden annotation file we have for test data, and obtain the score for your output file. Please make sure your output file follows the format of the "dev.name" file so our scorer can run correctly. Once you submitted your files, we won't be responsible for the wrong format of your output file or any other issues. If our scorer crashes with your output file, your score would be zero. In order to ensure the right format for your output, you are encouraged to use the provided scorer to see if it would fail with your output file format (e.g., in the development data).

We expect that most students will submit assignments in Python. If you wish to submit this assignment in another language, please make sure you include an clear instruction on how to install the requirements and run your code.

You may discuss methods with fellow students, but the code you submit must be your own.

[7] `https://ix.cs.uoregon.edu/~thien/nlpclass/dist.all.last.txt`

[8] `https://ix.cs.uoregon.edu/~thien/nlpclass/LargestCity.txt`

[9] `https://ix.cs.uoregon.edu/~thien/nlpclass/brown-c1000-freq1.txt`

[10] Again, we only provide these resources (i.e., lists of common names, Brown word clusters) for your convenience. There is no guarantee that they will be helpful for your system.