CS 410/510: Natural Language Processing
Spring 2024

1 Problem 1

[100 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. However, 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. Remember 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 with previous assignments, you'll receive training, development, and test datasets following our established format, but this time with two additional columns compared to the last assignment. The training data will have four elements per line (tab separated): the current token, part-of-speech, BIO chunk tag, and name tag. The development data will come in two forms: the input for tagging will have a token, part-of-speech, and chunk tag in each line; the input for scoring will have a token and name tag on each line. Finally, our provided test data will have three elements per line: the token, part-of-speech, and chunk tag. The response/output you return should have two (tab-separated) elements 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 in our previous assignment, 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 predictions for NER, we will use Maximum Entropy Markov Models (MEMM) so you can focus on designing appropriate features for the problem. To facilitate consistency for the modeling process, we will use the OpenNLP MaxEnt[1] (ver. 3.0.0) package in Java for MEMM. 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 system available in NLTK.)

With this package, you could proceed as follows:

1. Compilation:

   – Install JDK Development Kit (e.g., version 22.0.1): `https://www.oracle.com/java/technologies/downloads/`.

   – Download the necessary jar files: `maxent-3.0.0.jar`[2] and `trove.jar`[3]. Download our java files for the MEMM model: `MEtrain.java`[4] and `MEtag.java`[5]. Put them in a directory called "`maxent`". Open your terminal and go to `maxent`.

   – Compile `MEtrain.java` and `MEtag.java`. In your terminal, run:

     ```
     javac -cp maxent-3.0.0.jar MEtrain.java
     javac -cp maxent-3.0.0.jar MEtag.java
     ```

     This will produce the classes file `MEtrain.class` and `MEtag.class` in your current directory `maxent`.

2. Feature Generation: prepare a program `FeatureBuilder` that will read in one of the .pos-chunk-name or .pos-chunk data files we provide and generate a "feature file". Each line in this feature file corresponds to a feature vector for each token in the data file. Each line will consist of the token and zero or more additional features, separated by tabs, i.e.,

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

   If a .pos-chunk-name file is the input, this line in the feature file 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-speeches, 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

[1] `http://maxent.sourceforge.net/howto.html`

[2] `https://ix.cs.uoregon.edu/~thien/nlpclass/maxent-3.0.0.jar`

[3] `https://ix.cs.uoregon.edu/~thien/nlpclass/trove.jar`

[4] `https://ix.cs.uoregon.edu/~thien/nlpclass/MEtrain.java`

[5] `https://ix.cs.uoregon.edu/~thien/nlpclass/MEtag.java`

use the symbol “@@” to refer to the previous tag. Your main task in this assignment is to design good features for NER and implement them to produce feature files.

3. Given your feature file generated for the training data “train.pos-chunk-name” (assuming it’s named `training_feature_file` in the current directory `maxent`), train a MEMM model:

   ```
   java -cp maxent-3.0.0.jar:trove.jar:.  MEtrain training_feature_file NER-MEMM.model
   ```

   Depending on your feature file, this might take some time. After it finishes, it will produce a file `NER-MEMM.model` to store the resulting trained model in your current directory `maxent`.

4. Given your feature file generated for a dev/test data file “dev.pos-chunk” or “test.pos-chunk” (named `tag_feature_file`), use your trained model to tag the data:

   ```
   java -cp maxent-3.0.0.jar:trove.jar:.  MEtag tag_feature_file NER-MEMM.model output.txt
   ```

   This will produce the `output.txt` file to store the predicted tags for the data file `tag_feature_file`, following the same format as dev.name. You can use this output file for performance scoring later.

5. Use the provided scorer[6] (Python3 coded) to score the result.

   ```
   python score.name.py keyFileName responseFileName
   ```

   This will return tag accuracy, precision, recall, and F1 scores for your model performance.

   If you are not satisfied with the results, modify `FeatureBuilder` to add or change features and repeat the process.

6. When you are satisfied, apply the model to the test corpus and submit the output result (obtained from running `MEtag`).

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

The important question is what features to compute on the word sequence for NER. 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 feature set to tag 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 should describe the feature sets you have tried and the performance for them on the development corpus. All the instructions to running the code or so should be put in the end of this file as well.

[6]`https://ix.cs.uoregon.edu/~thien/nlpclass/score.name.py`
[7]`https://ix.cs.uoregon.edu/~thien/nlpclass/dist.all.last.txt`
[8]`https://ix.cs.uoregon.edu/~thien/nlpclass/LargestCity.txt`

- your code (preferably put in a separate directory).
- the output file of your model 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 75 points out for meeting the minimal requirement and up to 25 additional points 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). The maximal point is thus 100. One example of the more elaborate features is to use the Brown word clusters[9] [10]. In particular, we will run the scorer to compare your submitted file with the golden annotation file we have for test data, obtaining the performance 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 performance on test set would be zero. In order to ensure the correct format for your output, please use the provided scorer to see if it would fail with your output file format (e.g., using development data).

We expect that most students will write code in Python. If you wish to do this assignment in another language, please make sure you include a 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.

[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.