Homework Number 5

Due Date of the 16th Class

1. You will write a Noun Group tagger, using similar data that you used for Homework 4. However, for this program we will focus more on feature selection than on an algorithm.
    1. Download the WSJ_CHUNKFILES.zip from NYUClasses (Resources). This includes the following data files:
        1. WSJ_02-21.pos-chunk -- the training file
        2. WSJ_24.pos -- the development file that you will test your system on
        3. WSJ_24.pos-chunk -- the answer key to test your system output against
        4. WSJ_23.pos -- the test file, to run your final system on, producting system output
    2. Download MAX_ENT_files.zip, also from NYUClasses resources. This includes the following program files:
        1. maxent-3.0.0.jar, MEtag.java. MEtrain.java and trove.jar -- Java files for running the maxent training and classification programs
        2. score.chunk.py -- A python scoring script
    3. Create a program that takes a file like WSJ_02-21.pos-chunk as input and produces a file consisting of feature value pairs for use with the maxent trainer and classifier. As this step represents the bulk of the assignment, there will be more details below, including the format information, etc. This program should create two output files. From the training corpus (WSJ_02-21.pos-chunk), create a training feature file (training.feature). From the development corpus (WSJ_24.pos), create a test feature file (test.feature). See details below.
    4. Compile and run `MEtrain.java`, 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` and `trove.jar`, on the classpath when you compile and run. Assuming all java files are in the same directory, the following command-line commands will compile and run these programs -- these commands are slightly different for posix systems (Linux or Apple), than for Microsoft Windows.
        1. For Linux, Apple and other Posix systems, do:
            1. javac -cp maxent-3.0.0.jar:trove.jar *.java ### compiling

2. java -cp .:maxent-3.0.0.jar:trove.jar MEtrain training.feature model.chunk ### creating the model of the training data
3. java -cp .:maxent-3.0.0.jar:trove.jar MEtag test.feature model.chunk response.chunk ### creating the system output

2. For Windows Only -- Use semicolons instead of colons in each of the above commands, i.e., the command for Windows would be:
    1. javac -cp maxent-3.0.0.jar;trove.jar *.java ### compiling
    2. java -cp .:maxent-3.0.0.jar;trove.jar MEtrain training.chunk model.chunk ### creating the model of the training data
    3. java -cp .:maxent-3.0.0.jar;trove.jar MEtag test.chunk model.chunk response.chunk ### creating the system output

5. Score your results with the python script as follows:

    - python score.chunk.py WSJ_24.pos-chunk response.chunk ### WSJ_24.pos-chunk is the answer key and response.chunk is your output file

6. When you are done creating your system, run step 1.4.3 with the test corpus (WSJ_23.pos) as input to create your final response file (WSJ_23.chunk). **Your submission to gradescope must contain the file WSJ_23.chunk or it will not be able to grade your work.**
7. This pipeline is set up so you can write the code for producing the feature files in any programming language you wish. You have the alternative of using any Maxent package you would like, provided that the scoring script works on your output.

2. As mentioned in section 1.3, you are primarily responsible for a program that creates sets of features for the Maximum Entropy system.
    1. Format Information:
        1. There should be 1 corresponding line of features for each line in the input file (training or test)
            1. **If the input and feature files have different numbers of lines, you have a bug**
        2. Blank lines in the input file should correspond to blank lines in your feature file
        3. Each line corresponding to text should contain **tab separated values** as follows:
            1. the first field should be the token (word, puncutation, etc.)

            2. this should be followed by as many features as you want (but no feature should contain white space). Typically, features are recommended to have the form attribute=value, e.g., POS=NN

                - This makes the features easy for humans to understand, but is not actually required by the program, e.g., the code does not look for the = sign.

            1. for the training file only, the last field should be the BIO tag (B-NP, I-NP or O)
            2. for the test file, there should be no final BIO field (as there is none in the .pos file that you would be training from)
            3. A sample training file line (where \t represents tab): 'fish\tPOS=NN\tprevious_POS=DT\tprevious_word=the\tI-NP ## actual lines will probably be longer
        2. There is a special symbol '@@' that you can use to refer to the previous BIO tag, e.g., Previous_BIO=@@

            - This allows you to simulate an MEMM because you can refer to the previous BIO tag

    1. Suggested features:
        1. Features of the word itself: POS, the word itself, stemmed version of the word
        2. Similar features of previous and/or following words (suggestion: use the features of previous word, 2 words back, following word, 2 words forward)
        3. Beginning/Ending Sentence (at the beginning of the sentence, omit features of 1 and 2 words back; at end of sentence, omit features of 1 and 2 words forward)
        4. capitalization, features of the sentence, your own special dictionary, etc.
3. When you have completed the assignment, submit the following in a zip file through GradeScope (link to be added):
    1. Your code
    2. A short write-up describing the features you tried and your score on the development corpus.
    3. Your output file from the test corpus, i.e., WSJ_23.chunk. **Your submission to gradescope must contain the file WSJ_23.chunk or it will not be able to grade your work.**
4. Understanding the scoring:

    1. Accuracy = (correct BIO tags)/Total BIO Tags
    2. Precision, Recall and F-measure measure Noun Group performance: A noun group is correct if it in both the system output and the answer key.
        1. Precision = Correct/System_Output
        2. Recall = Correct/Answer_key
        3. F-measure = Harmonic mean of Precision and Recall
5. Evaluation: A simple system should achieve about 90% F-measure. It is possible, but difficult to obtain 95-96%. Your grade will be judged as follows:
    1. Your system's score
    2. The innovativeness of your features
    3. Any interesting analysis that you do