ASSIGNMENT 3: NAMED ENTITY RECOGNITION

Motivation: The motivation of this assignment is to get practice with sequence labeling tasks such as Named Entity Recognition. More precisely you will experiment with the HMM and/or CRF models and various features on a subset for a medical corpus with a natural language processing package called MALLET.

Problem Statement: The goal of the assignment is to build an NER system for diseases and treatments. The input of the code will be a set of tokenized sentences and the output will be a label for each token in the sentence. Labels can be D, T or O signifying disease, treatment or other.

Training Data: We are sharing a training dataset of labeled sentences. The format of each line in the training dataset is "token label". There is one token per line followed by a space and its label. Blank lines indicate the end of a sentence. It has a total of 3655 sentences.

The Task: You need to write a sequence tagger that labels the given sentences in a tokenized test file. The tokenized test file follows the same format as training except that it does not have the final label in the input. Your output should label the test file in the same format as the training data.

To accomplish this, first download and install MALLET. Get Familiar with the "sequence tagging" part of MALLET by reading about command line interface for sequence tagging and the developer's guide for sequence tagging. This documentation is very short and incomplete, but that's all there is.

Run Mallet's SimpleTagger on training.txt with the following command:

- java -cp "/home/username/mallet-2.0.7/class:/home/username/mallet-2.0.7/lib/mallet-deps.jar" cc.mallet.fst.SimpleTagger --train true --test lab --threads 2 training.txt

The above command is meant for linux so you may need to adapt the syntax for other operating systems. Also, make sure to correct the path. Mallet seems to be buggy if you use a single thread so make sure to set the --threads option to at least 2. Mallet will optimize a CRF model on half of the data and test it on the other half. If MALLET takes too long, increase the number of threads based on the number of cores that your computer has. Also, use the --iterations option to reduce the number of iterations from the 500 default to something smaller like 50.

In order to improve the tagging accuracy create additional features that might be useful for the task. For example, if you wish to add features of capitalization and whether the current token is a body word, it may look like this for the disease "chronic coronary Edwards complex":

chronic D

coronary BODYWORD D

Edwards CAPITAL D

complex D

…

You may have multiple features space separated before the final label of the token. Insert only the features that are on before the label. The word itself is treated as a feature. The order of the features does not matter.

Here are some suggestions on features:

1. Try features from lower level syntactic processing like POS tagging or shallow chunking.
2. Try features that try to assign a semantic label to the current token. A well-known generic ontology is Wordnet. Another famous medical ontology is MESH. Features based on these ontologies are likely to help.
3. You may use existing word embeddings as features. One possibility is to use word2vec embeddings that are trained on a general corpus.
4. You may also train your own word embeddings using unlabeled data. I have collected some sample in-domain unlabeled data here.
5. You may define word shape features.
6. Your idea here…

You may also experiment with the order of the Markov Chain in CRF model by using the --orders option. You may also modify the source code of SimpleTagger to experiment with an HMM instead of a CRF.

Methodology and Experiments:

Option 1: Remove 20% of data randomly as test set and 10% as development set. Train on 70% of the data. If needed, do parameter fitting on the devset and finally test on the test set.

Option 2: Do a 10-fold cross validation. Train on 8 folds, use 9^{th} as devset and 10^{th} as test set. Repeat this 10 times with different folds as test set. This is a more robust option.

As you work on improving your baseline system document its performance. Perform error analysis on a subset of data and think about what additional knowledge could help the classifier the most. That will guide you in picking the next feature to add.

Perform ablation study by switching off subsets of your features and see the degradation of performance. You can perform an alternative experiment by incrementally adding sets of features. Either way the goal is to identify the most useful features (and the value of each feature) for this task. In addition to quantitative results, also look at specific examples and try to qualitatively understand value of each feature by noticing which examples each feature helps in.

What to submit?

1. Submit your best code (best if trained on all training data and not just on a subset) by Tuesday, 4th November 2014, 11:55 PM. The code should **not** need to train again. You should submit only the testing code, after the models have been trained. That is, you should not need to access the training data anymore.

   Submit your code is in a .zip file named in the format **<EntryNo>.zip.** Make sure that when we run "unzip yourfile.zip" the following files are produced in the present working directory:

   compile.sh
   run.sh
   Writeup.pdf (and not writeup.pdf, Writeup.doc, etc)

   You will be penalized if your submission does not conform to this requirement.

   Your code will be run as ./run.sh inputfile.txt outputfile.txt. The outputfile.txt should have the same number of lines as inputfile.txt. And it should have two additional characters per token line (space and labeling). Here is a format checker. Make sure your code passes format checker before final submission.

   Your code should run on Baadal machines with 2 GB RAM.

2. Your writeup (at most 2 pages, 10 pt font) should describe how you created your best NER system (about 1 page). Explain any interesting ideas that you used. Describe your ablation study detailing the value of each feature quantitatively. Give specific examples to describe value of each feature qualitatively. Also mention the names of people you discussed the assignment with. The writeup will judged on clarity and innovation as well as experimental results.

Evaluation Criteria

(1) 30 points are for description of the system. Anything innovative may yield bonus points.
(2) 60 points for performance of your code.
(3) 10 points bonus given to outstanding write-ups and best performing systems.

What is allowed? What is not?

1. The assignment is to be done individually.
2. You may use Java, or Python for this assignment.

3. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
4. Feel free to search the Web for papers or other websites describing how to build named entity recognizers. Cite the references in your writeup. However, you should not use (or read) other people's NER code.
5. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.
6. Your code will be automatically evaluated. You get a zero if it is does not conform to output guidelines. Make sure it satisfies the format checker before you submit.