CS 388 Natural Language Processing
Homework 2: Part-of-Speech Tagging with HMMs and CRFs

Due: March 9, 2015

In this homework you will explore the performance of Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) on the POS tagging task, in particular using some real-world data from the Penn Treebank, replicating some of the experiments in the original paper on CRFs (Lafferty *et al*., 2001) (using less data to reduce computational demands). Follow the steps outlined below and then turn in any code you write electronically and an electronic copy write-up describing your methods, your experimental results, and your interpretation and explanation of these results. Specific information regarding how to submit the electronic part is here.

1. Obtain and compile the source code download the latest version (known working versions: 2.0.6, 2.0.7) and copy the simple HMM interface from `/u/mooney/cs388-code/nlp/pos/HMMSimpleTagger.java` into your source folder `mallet-2.0.7/src/cc/mallet/fst/`.
2. Compile it using the built-in build.xml file (just run "ant" while in the mallet-2.0.7 directory).
3. The first task in any NLP project is to convert the raw data into a format suitable for software being used. In our case, we need to convert from the Penn Treebank format to the Mallet format. The two datasets we will use in this project are Airline Travel Information Service (ATIS) and Wall Street Journal (WSJ). ATIS is a much smaller corpus covering a more limited domain (airline reservations vs. general and financial news). The raw POS-tagged ATIS data is located in `/projects/nlp/penn-treebank3/tagged/pos/atis/atis3.pos`. It looks something like this:

```
 [ @0y0012sx-a-11/CD ]

======================================

List/VB
[ the/DT flights/NNS ]
from/IN
[ Baltimore/NNP ]
to/TO Seattle/NNP
[ that/WDT stop/VBP ]
in/IN
[ Minneapolis/NNP ]

======================================

[ @0y0022sx-d-5/CD ]

======================================

Does/VBZ
[ this/DT flight/NN ]
serve/VB
[ dinner/NN ]

======================================
```

Mallet requires the input data to contain one word and label per line, space separated, with blank lines between sentences, e.g.:

```
List VB
the DT
flights NNS
from IN
Baltimore NNP
to TO
Seattle NNP
that WDT
stop VBP
in IN
Minneapolis NNP
```

```
Does VBZ
this DT
flight NN
serve VB
dinner NN
```

Also, for ATIS, occurrences like `[ @0y0022sx-d-5/CD ]` should be removed during conversion, since these identifiers are not useful English text. The POS-tagged WSJ data is located in `/projects/nlp/penn-treebank3/tagged/pos/wsj/` and is separated into multiple files called "sections". Write preprocessing code to convert a Penn Treebank POS file into a file appropriate for Mallet.

4. See the sparse documentation on the Mallet `SimpleTagger` here. The Mallet source comes with a built-in CRF tagger, `SimpleTagger`, and the class version adds a HMM tagger, `HMMSimpleTagger`. You may wish to examine the source code for these files for more command line arguments (however, be careful using the `--threads` command for the CRF SimpleTagger-- it might actually take longer than without threads). For ATIS you should train on 80% of the data and test on the remaining 20% using the commands:

```
$ java -cp "mallet-2.0.7/class:mallet-2.0.7/lib/mallet-deps.jar"
       cc.mallet.fst.HMMSimpleTagger
       --train true --model-file model_file
       --training-proportion 0.8
       --test lab train_file

$ java -cp "mallet-2.0.7/class:mallet-2.0.7/lib/mallet-deps.jar"
       cc.mallet.fst.SimpleTagger
       --train true --model-file model_file
       --training-proportion 0.8
       --test lab train_file
```

to run the `HMMSimpleTagger` and `SimpleTagger` (uses CRFs) respectively. The directory containing your copy of Mallet must be in your `CLASSPATH` environment. `train_file` refers to the formatted training set generated above, and `model_file` refers to where the trained model output will be stored. The argument "--test lab" tells it to measure the token labeling accuracy during testing.

Change the training and testing code in Mallet to also measure accuracy specifically for out-of-vocabulary items (OOV, as in Lafferty, et al., 2001, section 5.3). This requires storing all of the words encountered during training and efficiently checking each test word to determine whether or not it appeared in training, recording errors on OOV items separately. Both HMMs and CRFs use Mallet's `TokenAccuracyEvaluator` class to measure accuracy, logging results through the Java logger. The `evaluateInstanceList` method of this class is called during both testing and training. One strategy is to record all seen training instances in this class during the first training iteration and reference it during testing iterations. Although Mallet does not make use of Java 1.5 features, you are welcome to use them in your code, provided you update the `build.xml` file.

ATIS is a fairly small corpus and should run fairly quickly. Average your results over 10 random training/test splits of the data (changing the random seed in each trial using the parameter `--random-seed` to produce different train/test splits). The WSJ data is much larger and training on the standard 20 sections requires about a week. Therefore, just train on section 00 and test on section 01 (which should take the CRF tagger about 2-4 hours or so, on the Condor pool). An extension of this experiment would involve training on two sections and testing on two other sections. Study the effects of increasing training data, and compare and contrast between the results of training using HMMs and CRFs. Compare the test accuracy, training accuracy, and run time (you can use the Unix command `time` for this or condor's timing statistics from the job completion e-mail) of the CRF and HMM taggers on both ATIS and WSJ data. You might also try training the HMM tagger on a larger number of sections. Other experiments would include changing the number of iterations for the CRF tagger. How does varying the number of iterations affect HMMs and CRFs? This can be answered in your report as well.

5. Next try adding extra orthographic features to the data and rerunning the CRF tagger (since HMMs are generative, they can not handle such arbitrary features, unless p(x) is modeled as well). `SimpleTagger`(CRF) allows extra features to be included on each line, separated by spaces. Therefore, your new input data might look like this:

```
leaving ing VBG
from IN
Baltimore caps NNP
making ing VBG
a DT
stop NN
in IN
Minneapolis caps s NNP
```

In this case, orthographic features have been added indicating capitalization (`caps`) as well as common English suffixes (e.g. `-ing` and `-s`). Also include a feature for words containing a hyphen (`"-"`) or words that start with a number. To find a good set of suffixes to detect, just search the web to find a site with a good table of common English suffixes. Rerun the earlier experiments on ATIS and WSJ with the CRF tagger using these additional features.

Using Condor

We encourage you to run your longer experiments using the department's Condor pool. An overview of the system is available here. Because Condor does not have native support for checkpointing Java jobs, you may find it easiest to run in the `vanilla` universe. Be aware that your jobs may be terminated by Condor if they are competing for resources and plan ahead for this if you choose to use Condor.

Your condor submit file(s) may follow this template:

```
universe = vanilla
environment = CLASSPATH=path to Mallet/mallet-2.0.7/class:path to Mallet/mallet-2.0.7/lib/mallet-deps

Initialdir = path to experiment
Executable = /usr/bin/java

+Group   = "GRAD"
+Project = "INSTRUCTIONAL"
+ProjectDescription = "CS388 Homework 2"

Log = path to experiment/experiment.log

Notification = complete
Notify_user = your email

Arguments = cc.mallet.fst.SimpleTagger arguments to Mallet

Output = experiment.out
Error  = experiment.err
Queue 1

Additional experiments can go here
```

Report

Your report should contain a problem statement, the approach that you took to solve the problem, a brief discussion of the experiments you ran, including a nicely formatted table(s) of results, with training and test accuracy (both overall and just for OOV items), number of OOV items as a percentage of total test tokens, and run time for each approach for each corpus. Also include a discussion of your results that answers at least the following questions, where possible, explaining the results in terms of underlying algorithmic properties.

- How does the overall test accuracy of CRF and HMM differ (when using only tokens) and why?
- How does the test accuracy for OOV items for CRF and HMM differ (when using only tokens) and why?
- How does the training accuracy of HMM and CRF differ and why?
- How does the run time of HMM and CRF differ and why?
- How does adding orthographic features affect the accuracy (both overall and OOV) and runtime of the CRF and why?
- Which features helped the most? (i.e. try only including some feature types and not others)

If you are ambitious and have time, you might try adding additional types of features of your own design and discuss your results.

Code

Be sure to turn in any code you write using the steps here. Also be sure to include a README file including the specific commands you used for this assignment and a description of your changes to Mallet. Please do not submit all of Mallet -- simply electronically submit your report, README file, your preprocessing code, and any files that you changed in Mallet.

References

J. Lafferty, A. McCallum, and F. Pereira. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. Proceedings of the International Conference on Machine Learning (ICML-2001)

C. Sutton and A. McCallum. *An Introduction to Conditional Random Fields for Relational Learning*. Introduction to Statistical Relational Learning. Edited by Lise Getoor and Ben Taskar. MIT Press. 2006