CS4740: Introduction to Natural Language Processing
Project 3: Sequence tagging and its applications

Due electronically by 11:59 PM, Tuesday, Apr. 17, 2012

1 Overall Goal

This project is an open-ended assignment in which you are to implement a hidden Markov model (HMM) or experiment with a package/tookit that implements one or more of the sequence-tagging algorithms covered in class: HMMs, Maximum Entropy Markov Models (MEMMs) or even conditional random fields (CRFs).

Then you will be applying the sequence-tagging method of your choice to **part-of-speech tagging** or to **named entity recognition**. Although we will provide the data for both of these tasks, you need only address one of the tasks in this project.

If you decide to use an existing package/toolkit rather than implement a sequence tagger from scratch, you are expected to include more extensive experiments. As usual, you will write up your findings in a short 5-6pg report.

We suggest that you work in groups of 3-5 students.

2 Programming Portion

1. We will provide two data sets in CMS, one for each task. Each data set is divided into a training set and a test set.

2. For the p-o-s tagging task, each token in the sentences is associated with a p-o-s tag, and you are expected to predict the p-o-s tag for each token. The NE task uses the IOB encoding scheme. Each token is associated with a label "O" if it is outside the entity, label "B-xxx" if it is the head of entity "xxx", and "I-xxx" if it is within entity "xxx" but not the head of the entity. We also provide the p-o-s tag of each token which you can consider in your sequence tagging approach.

3. You should then **either**:

(a) Implement the sequence tagging algorithm yourself. We very strongly suggest implementing HMMs rather than MEMMs. If you decide to try both, start with HMMs. You may use any programming language that you'd like and any preprocessing tools that you can find. It might be possible, for example, to use a toolkit just to extract n-gram information, but to write the HMM code by yourself.

(b) Use an existing HMM/MEMM/CRF toolkit/package to run your experiments. It is up to you to figure out how to use the packages. Some suggestions are available from the course home page. **In both cases, you will need to think up front about what kind of experiments would be interesting to run given your choice of algorithm and data set.**

4. **Part-Of-Speech tagging** The **Part-Of-Speech tagging** training data is provided with both words and the correct part of speech tags. For example,

   NNP Pierre
   NNP Vinken
   , ,
   ... ...

   is a snapshot of the training data, where the first column contains the p-o-s tags and the second column contains the words. The test data only has words. In order to evaluate your system's performance, you must go to
   `http://inclass.kaggle.com/c/cornell-cs4740-part-of-speech-tagging`
   and upload your predictions.

   You submission to Kaggle should be in the same format as the training file. When you upload your predictions to Kaggle you will see the evaluation results on a subset of the test data. You will be able to see your score on the full test set after the submission deadline. You might also want to reserve some of the training data for test purposes during development.

   The Kaggle competition ends at 8:00pm on Sunday, Apr. 15. You should include the final evaluation result in your final report. Note

that once the Kaggle competition closes, you cannot make additional submissions to Kaggle. The gold standard predictions for the test set will be released via CMS at the close of the competition. These could be of use as you analyze system performance or characterize the kinds of errors that your system made for inclusion in the project report.

5. **Named Entity Recognition** The training data is organized by sentences. Each sentence contains three lines, where the first line contains the words, the second line contains the corresponding p-o-s tags, and the third line contains the correct labels. For example,

   The following bond was announced by lead manager Toronto Dominion .
   DT VBG NN VBD VBN IN NN NN NNP NNP .
   O O O O O O O O B-PER I-PER O
   ... ...

   specifies an example sentence in the training data. "Toronto Dominion" is labeled as a "PER" (Person) entity in the sentence. The test data only has words and p-o-s tags. In order to evaluate your system's performance, you must go to `www.cs.cornell.edu/w8/~bishan/cs4740` and upload your predictions.

   You submission should be in the same format as the training file. When you upload your predictions you will see the evaluation results on a subset of the test data. You will be able to see your score on the full test set after the submission deadline (it is the same as the Kaggle submission deadline). You might also want to reserve some of the training data for test purposes during development.

   The standard measures to report for NE identification are *recall, precision, and F1 score(also called F-measure)* evaluated on the entity level. Precision is $\frac{|C \cap P|}{|P|}$ and Recall is $\frac{|C \cap P|}{|C|}$ and F1 is $\frac{2Prec \times Recall}{Prec + Recall}$, where P and C are the sets of predicted and correct name entity spans respectively. The gold standard predictions for the test set will be released via CMS at 8:00pm on Sunday, Apr. 15. These could be of use as you analyze system performance or characterize the kinds of errors that your system made for inclusion in the project report.

6. **Develop a baseline system.** For p-o-s tagging, for example, relatively good performance (approximately 90% accuracy) can be obtained by simply choosing the most frequent tag. This means, for each token as determined in the training set, a simple but dumb baseline would assign the most frequent tag in the training corpus to all words. For NE tagging, one option is to assume that all proper names (i.e. words that begin with a capital letter) are assigned the most likely NE tag.

3 Ideas for comparisons/experiments

Note that you do not have to do anything from this list if you have other ideas for interesting experiments to run using your system.

1. Will bigrams be adequate or will trigrams (or 4-grams, etc.) be better?
2. What smoothing method will you employ? How do different smoothing methods affect the results?
3. How does your handling of unknown words affect the results?
4. For MEMMs, you can try to encode different features or you can add P(tag—UNK) to your emission model.
5. If you're using toolkits, you might compare one sequence-tagging method with another.

4 The Report

You should submit a short document (5-6 pages will suffice) that contains the following sections. (You can include additional sections if you wish.):

1. The Sequence Tagging Approach. Make clear which sequence tagging method(s) that you selected. Make clear which parts were implemented from scratch vs. obtained via an existing package. Explain and motivate any preprocessing and design choices.
2. Experiments. Motivate and describe the experiments that you ran. What were your hypotheses as to what would happen?

3. Results. Summarize the performance of your system on both the training and test data. Minimally, you should compare your results to the baseline. Please put the results into clearly labeled tables or diagrams and include a written summary of the results.

4. Discussion of the results. This section should include any observations regarding your system's performance — e.g. When did it work well, when did it fail, why? How might you improve the system? Additional analysis is always good. You can do an error analysis — what sorts of errors occurred, and why? You can provide a learning curve — how does your model's performance change with the amount of training data provided. You can come up with a better baseline and compare your systems against that.

5 Grading Guide

- design and implementation of the sequence-tagging system (40% of the grade if you implemented the tagger from scratch, 25% if you used existing tagger(s))
- clarity and quality of the report including the quality of the experiments/comparisons selected (60% of the grade if you implemented the tagger from scratch; 75% of the grade if you used existing tagger(s))

5.1 Things to avoid

- Don't be ridiculously inefficient. You're not expected to spend time optimizing your code, but it shouldn't take forever either. Bigram Viterbi is $O(sm^2)$ where s is the length of the sentence and m is the number of tags. Your implementation should have this property. A very good implementation of trigram viterbi takes about 10 seconds to train and test on the Part-Of-Speech data you'll be given. You don't need to aim for that sort of speed, but if your code is taking all day, something's wrong. Come ask for help.

6 What to submit

Part One:
By 11:59pm, Sunday, Apr. 8, you need to submit:

1. A short document that indicates your plan for the project: which data set(s) and algorithm(s) will you include? will you implement the algorithm(s) from scratch or use a toolkit/package? if the latter, which toolkit/package will you use?

2. The document should describe the baseline system, report the baseline predictions for the task, and explain how you got them(e.g. wrote your own code, used a package, used the web-based scorer).

3. The code for the baseline system **if you are implementing from scratch**. If you are using a package, do **not** submit the code.

Full Project:
By 11:59pm, Tuesday, Apr. 17, you need to submit the full assignment to CMS. This consists of a .zip or .tar — one submission per group, containing the following:

1. source code and executables **We do not want you to submit code from existing toolkits/packages - only any code that you wrote yourself!!**

2. a README file that explains how to run your system

3. the report