MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2015

Assignment 3, Due: 9 AM on Friday Oct 30.

This homework contains one programming question for a total of 60 points. You are expected to upload a single zip archive containing the following: your code, 4 output files, and your report as a PDF.

Gene Recognition

In this assignment, we will implement a named entity recognition (NER) system to identify genes in biomedical literature. NER forms an important step for text mining in these types of corpora. In particular, our goal is to train a maximum entropy Markov model that can predict a tag for each word in the text.

Input Format

The input will be a list of instances each consisting of (a) an identifier on the first line, and (b) a sequence of tokens on the second line. Each token is of the form *word_tag* where *tag* can be one of {TAG, GENE1, GENE2}. TAG represents any word that is not part of a gene name, while GENE1 and GENE2 mean the token is part of a gene name. The index 1 and 2 just works as indicators to distinguish two different but adjacent gene names. **Note that the tags are outputs to be predicted and are not to be taken as input for your model.**

An example of one such instance if provided below. You can take a look at either of the **.tag* files in the *data* directory to get a better sense.

```
P00054900A0226
Peroxydase_GENE1 reaction_TAG stains_TAG were_TAG negative_TAG ,_TAG
chloroacetate_GENE2 esterase_GENE2 were_TAG strongly_TAG positive_TAG
 ._TAG
```

The file *train.tag* contains the gold tags for each word. You can use these to train your model. For testing on *dev.tag*, you should only use the words provided in the file to predict tags.

Output Format

The output format of your code should be exactly the same as the input format. For each instance in the input file, output two lines - (a) the identifier (e.g. P00054900A0226) on the first, and (b) the tagged sequence on the second. For each word, output its tag in the format *word_tag*. The output should be sent to the standard stream (e.g. using the *print* function).

Evaluation

The evaluation will be done using measures of *Precision*, *Recall* and *F1 score*. The precision of your model is the percentage of genes detected by your model that are genes according to the gold annotations. The recall is the percentage of genes that are detected by your model among all genes according to the gold annotations. The F1 score is the harmonic mean of precision and recall. You can evaluate the model (on train/dev data) using the evaluation scripts provided in the *eval* folder. Follow these steps:

(a) Extract the entities from the output file *output.tag* (produced by your model). You can use the script eval/format.perl:

```
perl eval/format.perl output.tag > output.format
```

(b) Now run the evaluation script eval/alt_eval.perl using the gold data for the appropriate file. For example, if you are evaluating on the development data *dev.tag*, you can use *dev.gold* for the gold entities. The file *Correct.data* contains alternative taggings. E.g. 'Peroxydase_GENE1 gene_GENE1' and 'Peroxydase_GENE1 gene_TAG' are both correct. You should not modify that file.

```
perl eval/alt_eval.perl data/dev.gold output.format data/Correct.data
```

You should get the precision, recall and F1 scores as outputs.

Framework

You are asked to use the maxent classifier from scikit-learn to implement your model. You can install scikit-learn by following instructions here.

Models Implement these different maximum entropy (log-linear) models and investigate their performance.

1. A simple non-contextual model which only looks at the current word w_t to predict its tag z_t. Note that you can still have many features extracted from this single word. Recall that the conditional probability can be modeled as:

$$p(z_t|w_t) \propto \exp(\theta \cdot \phi(z_t, w_t)) \tag{1}$$

where $\phi(z, w)$ is a feature vector and θ is the corresponding weight vector.

2. A context-sensitive model which selects a tag for a word w_t using features on w_t along with the previous n words $w_{t-n}, w_{t-n+1}, \ldots, w_{t-1}$ and tags $z_{t-n}, \ldots, z_{t-1}$. You are free to experiment with n here.[1] Therefore, we now have:

   $$p(z_t|w_{t-n}, \ldots, w_t, z_{t-n}, \ldots, z_{t-1}) \propto \exp(\theta \cdot \phi(z_t, w_{t-n}, \ldots, w_t, z_{t-n}, \ldots, z_{t-1})) \quad (2)$$

   For prediction, greedily predict a tag for each word in the sentence using your trained model. Therefore, at every time point t, you will use the predicted tags from the previous n time steps to predict z_t.

3. Now implement the Viterbi algorithm for decoding (see Homework 2, section 2) to find the most likely sequence of tags given a sentence. You can use the same context-sensitive model as above.

You can make use of the *train* and *dev* sets for training and tuning/feature selection.

Code Submission [30 points]

For your final submission, **provide the output for the test file *test.tag***[2] in a single file in the output format specified above. You can provide the output using the best set of features based on performance on the development data. We will evaluate your output using F1 score on the withheld gold tags.

Submit the 4 following output files *output_test1.tag* (non-contextual model), and *output_test2.tag* (context-sensitive model without Viterbi), *output_test3.tag* (context-sensitive model with Viterbi), and *output_test_best.tag* (based on your best performing model). The four files should be in *code.zip*'s root folder (along with *maxent_model.py*).

In addition, you are also required to **submit your code**. Your code should be zipped in an archive file called *code.zip*. Your main Python script should be located at the root of the *code.zip* archive (i.e. not in a folder) and be named *maxent_model.py*. It should take the training file and testing file as inputs. In addition, you should take in a parameter to specify the type of model to run (1/2/3). For example, we should be able to run the third model (context-sensitive with Viterbi) as follows:

```
python maxent_model.py data/train.tag data/test.tag 3 > output_test3.tag
```

The code should run in less than 30 minutes (on a reasonably specced computer), with Python 2.7. Don't include the provided data files in your submission.

If your code makes use of any non-standard Python library other than NumPy, SciPy, scikit-learn, matplotlib and NLTK, please add a file *requirements.txt* at the root of the *code.zip* archive: *requirements.txt* should contain the list of any additional package

[1] n=1, 2, 3 are good options.

[2] Note that this file does not contain gold tags (every tag is TAG).

name: one package name per line, and the package name should be installable through *pip*, so that one can run *pip install -r requirements.txt*.

If you wish to compare your results with other students, you are welcome to submit your results for the train and dev set on this Google Spreadsheet. We will give 3 additional points to the submissions that arrive in the top 5 in the ranking based on the test set.

Report [30 points]

You should also submit a short 1-2 page write-up as a PDF file (include it in *code.zip*'s root folder). Address the following points in your report.

(a) Describe all the features you used for each model. Which features were the most useful?

(b) Which model performs the best on the development data in terms of F1 score? Does it also have the highest Precision and Recall?

(c) Provide scatter plots for precision, recall and F1 of the different models.