MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2017

Assignment 3, Due: 9am on Oct 24rd. Upload pdf or scan to Stellar.

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. We will train two models, one will be the maximum entropy Markov model, and the other will be a Recurrent Neural Network (RNN).

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. Indices 1 and 2 work only as indicators to distinguish two different but adjacent gene names.

An example of one such instance is 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 io 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.

Code Submission [30 points]

For your code submission, **provide the output for the test file *test.tag*[1]** in a single file in the output format specified above. You can provide the output using the model with 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 5 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), *output_test_RNN.tag* (RNN) and *output_test_best.tag* (based on your best performing model). The five files should be in *code.zip*'s root folder (along with *maxent_model.py* and *rnn_model.py*).

In addition, you are also required to **submit your code** for each model. Your code files should be zipped in *code.zip* mentioned above. Your main Python scripts should be located at the root of the *code.zip* archive (i.e. not in a folder).

Maxent model: The maxent model should 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
```

RNN model: The RNN model should be named: *rnn_model.py*. Please include instructions in a text file called *run_rnn_model.txt* that includes instructions on how to run your model.

If your code makes use of any non-standard Python library other than NumPy, SciPy, Pytorch, 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 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*.

Maxent Model Framework

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

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

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.[2] Therefore, we now have:

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

   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 should use of the *train* and *dev* sets for training and tuning/feature selection.

RNN Framework

Next, you will implement a neural model on the same dataset. A simple recurrent neural network has 3 layers: input layer, hidden layer, and output layer. Define w_t to be the features for the word at position t, $\hat{y}_t$ to be the predicted probabilities at position t, and h_t to be the hidden feature representation for the word at position t. The layers are then defined as follows:

- Input layer: The input x_t is a concatenation of the word level features and the previous hidden representation:

$$x_t = \begin{cases} [w_t; \vec{0}] & t = 0 \\ [w_t; h_{t-1}] & t > 0 \end{cases} \tag{3}$$

  For the initial word, you should use a zero vector with the size of the hidden state.

- Hidden layer: The hidden layer h_t is defined as:

$$h_t = \sigma(W_h \cdot x_t + b_h)$$

  Where σ is a nonlinear activation function. The rectified linear unit (ReLU) is often used because its gradient is easy and fast to compute.

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

- Output layer:

$$y_t = \text{softmax}(W_y \cdot h_t + b_y)$$

  Where the softmax function provides a probability distribution over the label classes.

The loss function for classification tasks is the cross entropy loss. Let y_t^i be the true label at position t for class i, and $\hat{y}_t^i$ be the predicted probability at position t for class i. The loss at position t is then defined as (this is the same as the NLLLoss function in torch):

$$l_t = -\sum_i \left[y_t^i \log(\hat{y}_t^i) + (1 - y_t^i) \log(1 - \hat{y}_t^i) \right]$$

You should consider which features from the maxent model you want to use for the RNN model. You may try different augmentations to the model (for instance beam search vs greedy decoding), and describe your model in the question section. Feel free to try GRU or LSTM if you think it will improve your model performance (Pytorch provides libraries to do this easily).

IMPORTANT: Because the cross entropy loss/ negative log likelihood is not directly correlated to the F1 score, you will have to re-balance the classes while training. (See the ”weight” option of NLLLoss).

Questions [35 points]

Please answer the following questions regarding the models that you ran.

1. Please provide your results for your 4 models (Precision, Recall, F1 Score).
2. Describe all the features you used for the maxent models. Which features were the most useful?
3. Please describe the RNN model you used in detail, including the feature space (if different from the maxent model), the decoding method, or any other augmentations you used for the model.
4. What are the advantages of using a RNN compared to a maxent model? Disadvantages?
5. Please describe how your RNN model performance changes as you change the hidden layer size. You can use the cross entropy loss as a measure of performance for this question.
6. Which maxent model performs the best on the development data in terms of F1 score? Does it also have the highest Precision and Recall? How does your RNN performance compare?

   Please answer the following questions regarding the exact RNN model described in the homework in the context of the gene tagging problem.
7. Suppose that the last word in the sentence was always a gene name. How easily can the described RNN model learn this positional bias? Explain your answer.
8. How can you modify this model such that the RNN has context of both words later in the sentence as well as the words earlier in the sentence?

Reading Questions [20 points]

This problem is for graduate students only.

Please read the following paper on Named Entity Recognition: Neural Architectures for Named Entity Recognition (NAACL 2016, http://www.aclweb.org/anthology/N16-1030), and answer the following questions.

1. Write down the loss function of conditional random fields and briefly describe the similarities and differences between CRF and MaxEnt.
2. Write down the formulation of LSTM, and specify the difference between LSTM and vanilla RNN.
3. How did the authors deal with the unknown words, i.e. the words in test data that are absent in the training data.
4. What are the advantages and disadvantages of the transition-based model compared with the LSTM-CRF model.

Submission details

```
Answers.pdf
code
 |__maxent_model.py
 |__rnn_model.py
 |__output_test1.tag
 |__output_test2.tag
 |__output_test3.tag
 |__output_testRNN.tag
 |__output_test_best.tag
```

1. Create and submit report with answers to questions **as a PDF named Answers.pdf**.
2. Submit report and code to Stellar **independently**. Submit the code in a zipped file. Do not include the data directory in the zipped file.