Independent Project 2: Sequence Labeling

CS 6501-005 Natural Language Processing
Instructor: Yangfeng Ji

Deadline: Nov. 2nd, 2018
Please download the data from Collab, and don't further distribute them.

1 Hidden Markov Models

1.1 A Toy Problem

In this part, let's first consider a toy sequence labeling problem with Viterbi decoding. The problem is about decoding a DNA sequence. Unlike POS tagging, it only has two hidden states H and L, and four possible observations values `A, C, G` and `T`. The transition probability between `H` and `L` is specified in Table 1. Similarly, the emission probability from hidden states to observations is given in Table 2

	H	L	END
START	0.6	0.4	0.0
H	0.4	0.4	0.2
L	0.2	0.5	0.3

Table 1: Transition probability

	A	C	G	T
H	0.2	0.3	0.3	0.2
L	0.3	0.2	0.2	0.3

Table 2: Emission probability

For the following questions, consider the sequence `GCACTG`

1. (3 points) Please review section 7.3 and algorithm 11 in [], and fill out the following trellis table with the values of Viterbi variable v_m and backpointer b_m in *each* cell.

START	G	C	A	C	T	G	end
0							
0							

Attention: Algorithm 11 describe the Viterbi algorithm in log space.

2. (2 points) Based on the answer to the previous question, decode the hidden states of the given DNA sequence.

1.2 POS Tagging

In this section, you will work on a problem POS tagging. From the attachment, you can find three files for this part.

- `trn.pos`
- `dev.pos`
- `tst.pos`

In both `trn.pos` and `dev.pos`, each line is one sentence, and each token consists of a word and its POS tag, as

`word`/POS

As you may notice, the POS tag set is much smaller than the conventional POS tag set in the Penn Treebank. There are only 10 tags in total

A, C, D, M, N, O, P, R, V, W

Remember, there are also two special tags for POS tagging: START and END.

The task for this section is to estimate both transition and emission probabilities first, then use them to build a POS tagger for the data in dev and test sets.

1. (1 point) Preprocessing. Scan the training examples and map the words with frequency less than K into a special unknown token `Unk`. Specify the value of K that you used in your implementation and the vocab size V.
2. (2 points) From the training examples in `trn.pos`, estimate the transition probability and make sure the following equation holds
$$\sum_{y_t \in \mathcal{Y}} P(y_t \mid y_{t-1}) = 1 \tag{1}$$
where y_{t-1} is the previous hidden state and y_t is the current hidden state.
Please write the estimated probabilities into a file named `[computingID]-tprob.txt` with the CSV format, i.e., in each line
$$y_{t-1}, y_t, P(y_t \mid y_{t-1})$$
3. (2 points) From the training examples in `trn.pos`, estimate the emission probability and make sure the following equation holds
$$\sum_{x_t \in \mathcal{V}} P(x_t \mid y_t) = 1 \tag{2}$$
where x_t is the current word and y_t is the corresponding POS tag.
Please write the estimated probabilities into a file named `[computingID]-eprob.txt` with the CSV format,
$$y_t, x_t, P(x_t \mid y_t)$$
4. (1 point) To avoid the numeric issue caused by zero probability, one simple solution is add a small number to the MLE equations. For example, the equation of estimated $P(x_t \mid y_t)$ now becomes
$$P(x_t \mid y_t) = \frac{c(x_t, y_t) + \alpha}{c(y_t) + V\alpha} \tag{3}$$
where V is the vocab size and set $\alpha = 1$ for now.

The similar idea can also be applied to the estimation of $P(y_t \mid y_{t-1})$ as

$$P(y_t \mid y_{t-1}) = \frac{c(y_{t-1}, y_t) + \beta}{c(y_{t-1}) + N\beta} \tag{4}$$

where N is the number of the all the possible tags and set $\beta = 1$ for now.

Re-estimate the probabilities using Equations 3 and 4, and write them into the `[computingID]-eprob-smoothed.txt` and `[computingID]-tprob-smoothed.txt` respectively

5. (3 points) Convert the estimated probabilities from the previous step into log space and implement the Viterbi decoding algorithm as in Algorithm 11. Report the accuracy of your decoder on the dev data.

6. Run the decoder on the test data, and write the decoding results into a file named `[computingID]-viterbi.txt` with the same format as in `trn.pos`.

7. (3 points, extra credits) Fine tune the values of α and β based on the decoding performance on the development set. Report the best values of α, β and corresponding accuracy.

8. (2 points, extra credits) Run the decoder on the test data, and write the decoding results into a file named `[computingID]-viterbi-tuned.txt` with the same format as in `trn.pos`.

2 Conditional Random Fields

The task of this section is to build a POS tagger with conditional random fields. In the file `crf.py`, you can find a very simple implementation about using CRF to build a POS tagger. In the class `CRF`, two methods `train` and `get_word_feature` are particular important for this assignment.

There are two data files used for this part

- `trn-tweet.pos`
- `dev-tweet.pos`

You don't need to worry about the data format. The data will be automatically loaded when you try to run `crf.py`.

First of all, run the code with `python crf.py` to make sure everything works fine. In order to build a CRF model, you need to install the Python package `sklearn_crfsuite`.

1. (1 point) As you can see from the `get_word_feature` function, only the word itself is used as feature. Try to use the POS tag of the corresponding word as an additional feature, and report the performance on dev data. (**Hint**: you may need the `update` function in the Python built-in type `dict`: `https://docs.python.org/3/library/stdtypes.html#dict`)

2. (2 points) Please augment the model by adding more features into `get_word_feature`. Here are some examples
   - the first letter of the word
   - the last letter of the word
   - the first two letters of the word
   - the last two letters of the word
   - the previous word
   - the next word

- $\cdots$

In your report, explain what features you added and report the performance on the dev data.

3. In the `train` function, there are some pre-defined parameter values of `crfsuite.CRF`. More detail explanation of these parameters can be found in `https://sklearn-crfsuite.readthedocs.io/en/latest/api.html`. Specifically, there is a parameter called `algorithm`, which is the training algorithm used for CRFs.

   - (1 point) Switch the value of `algorithm` from "lbfgs" to "averaged perceptron", report the performance on the dev data.
   - (2 points) Based on our lectures on CRF learning and averaged perceptron, explain how it is used to train CRF models. (**hint**: (i) think about the difference between training a perceptron and a logistic regression model; (ii) think about where you can get $\hat{\boldsymbol{y}}$ for a given sequence $\boldsymbol{x}$.)