Report of Homework 3: Chinese Event Extraction

Name: Jialun Shen
Sid: 16307110030

I. Introduction

In this homework, I used two sequence labeling models for Chinese event extraction. I implemented Hidden Markov Model (HMM) by myself, and used the CRF++ package[1] for Conditional Random Fields (CRF).

II. HMM

HMM is implemented in *extraction.py*. Recall that an HMM is defined by:

1. States X; in this task, labels
2. Observations E; in this task, words
3. Initial distribution: $P(X_1)$
4. Transitions: $P(X|X_{-1})$
5. Emissions: P(E|X)

Our task is most likely explanation: given a sequence of words $e_1, ..., e_t$, we need to find the sequences of labels $x^*_1, ..., x^*_t$ that is most likely to have generated those observations, i.e.,

$$x^*{}_{1:t} = \underset{x_{1:t}}{\operatorname{argmax}} \mathrm{P}(x_{1:t}|e_{1:t})$$

In my implementation, I used Viterbi decoding, which uses dynamic programming and can get the global best sequence of labels. In contrast, greedy decoding simple assigns a locally best symbol at each position depending on previous labeling decisions as well as observed data. Therefore, greedy decoding cannot guarantee the sequence to be optimal.

The function for Viterbi decoding in *extraction.py* is *test_sent_viterbi*(*words, model*). In the function, *Mlst* is a a probability lattice M (a list of dictionaries), and *Mlst*[*i*][*j*] is the max probability of states ending with state j at time i M[i, j]. For each sequence of evidence *words*, we iterate from the first word to the last, update M by

$$\mathrm{M[i, j]} = \max_{\mathrm{k}} \mathrm{M[i-1, k]P(j|k)P}(e_{\mathrm{i}}|x_{\mathrm{j}}) \quad 1 \le \mathrm{i} \le \mathrm{n}$$

and we store the best previous label k* in a list of dictionary as *backpointerlst*[*i*][*j*]. Finally, we pick the maximizer label k* of M[n, k], and backtrack to get the best sequence for the whole sentence.

III. CRF

HMM is a generative model, which learns the joint probability distribution P(X, E); while CRF is a discriminative model, which learns the conditional distribution P(X|E). For a sequence of evidence $e_1, ... ,e_n$ and a sequence of labels $x_1, ..., x_n$, we have

---

[1] http://taku910.github.io/crfpp/

$$P(x_1 ... x_n | e_1 ... e_n) = \frac{\exp(W \cdot F(x_1 ... x_n | e_1 ... e_n))}{Z_W(e_1 ... e_n)}$$

where $F(x_1 ... x_n | e_1 ... e_n)$ is the features, W is the weights, and

$$Z_W(e_1 ... e_n) = \sum_{x' \in X^n} \exp(W \cdot F(x' | e_1 ... e_n))$$

is the normalizing constant.

For the decoding problem, we can use a variant form of the Viterbi algorithm, in a very similar way to the decoding algorithm for HMMs, and I will skip the details.

CRF++ is a command line tool. The version I used is 0.58.

The commands I used to train the models are:

```
$ ./crf_learn template argument_train.txt argument_model
$ ./crf_learn template trigger_train.txt trigger_model
```

and the commands I used to generate the result files are:

```
$ ./crf_test -m argument_model argument_test.txt >> argument_result_crf.txt
$ ./crf_test -m trigger_model trigger_test.txt >> trigger_result_crf.txt
```

The template I used is a simple one as follows:

```
# Unigram
U00:%x[-2,0]
U01:%x[-1,0]
U02:%x[0,0]
U03:%x[1,0]
U04:%x[2,0]
U05:%x[-1,0]/%x[0,0]
U06:%x[0,0]/%x[1,0]
# Bigram
B
```

Also, I found that an inappropriate template can cause failure to train a model.

IV. Performance Comparison

The evaluation results of HMM and CRF are given below.

```
=====trigger labeling result=====
type_correct:  1.0
accuracy:  0.9159
precision:  0.5241
recall:  0.127
F1:  0.2045
=====argument labeling result=====
type_correct:  0.039
accuracy:  0.4276
precision:  0.4065
recall:  0.9281
F1:  0.5654
```

Fig. 1 Result of HMM

```
=====trigger labeling result=====
type_correct:  0.9712
accuracy:  0.9481
precision:  0.9619
recall:  0.4058
F1:  0.5708
=====argument labeling result=====
type_correct:  0.8316
accuracy:  0.8395
precision:  0.9228
recall:  0.6546
F1:  0.7659
```

Fig. 2 Result of CRF

We can see that generally CRF has a better performance over HMM. There are 9 and 36 different labels in trigger and argument labeling task (including O) respectively. However, I notice that HMM has a strange performance -- it has high type_correct and low recall in trigger labeling, but low type_correct and high recall in argument labeling.

A possible problem may be underflow caused by multiplication of small probabilities. Therefore, I revised the M in Viterbi decoding to be

$$\mathrm{M}[\mathrm{i},\mathrm{j}] = \max_{\mathrm{k}} \mathrm{M}[\mathrm{i}-1,\mathrm{k}] + \log \mathrm{P}(\mathrm{j}|\mathrm{k}) + \log \mathrm{P}(e_{\mathrm{i}}|x_{\mathrm{j}}) \quad 1 \le \mathrm{i} \le \mathrm{n}$$

and I assign a small probability *p*=1e-50 to zero probability so as to prevent math error. The new result is

```
=====trigger labeling result=====
type_correct:  0.9681
accuracy:  0.9035
precision:  0.4226
recall:  0.3664
F1:  0.3925
=====argument labeling result=====
type_correct:  0.229
accuracy:  0.5837
precision:  0.4704
recall:  0.299
F1:  0.3656
```

Fig. 3 Result of HMM with revised Viterbi

HMM performs poor indeed. As a generative model, HMM may require more training samples to improve its performance.

The results for *task* are saved in *task*_result_hmm.txt and *task*_result_crf.txt.