Chinese Event Extraction Report

Shihan Ran - 15307130424

***Abstract*—This project is aimed at doing sequence labeling to extract Chinese event using Hidden Markov Models and Conditional Random Field, which can be separated as two subtasks: trigger labeling (for 8 types) and argument labeling (for 35 types). During this project, for reading and saving data, I use libraries like *pickle* and *codecs*. In terms of tokenization and tagging Part-Of-Speech for preparation for the CRF toolkit, I choose *Jieba*. To achieve higher accuracy rate for HMM, I've used several smoothing methods, and implemented both bigram and trigram models. Talking about training and testing models, I divided the Development Set into Training Set and Dev-Test Set. Finally, the best accuracy was achieved at 71.65% for argument, 94.68% for trigger with CRF, 96.15% for argument, 71.88% for trigger with HMM.**

I. Introduction

EVENT Extraction has always been a popular topic in Natural Language Processing. With the amount of text files on the Internet increasing exponentially each day, the volume of information available online continues expanding. Sequence labeling, where the task is to map a sentence $x_1, \cdots, x_n$ to a tag sequence $y_1, \cdots, y_n$, is an important component in information extraction tasks.

In this project, we focus on Chinese event extraction, which can be separated into two aspects: identify the trigger word in the sentence, classify it to the 8 types and identify all the arguments in the sentence, classify them to 35 types.

For the following parts of the report, we will present the mathematics of the HMM firstly, beginning with the Markov chain. Then we will discuss the principles of Viterbi algorithms by introducing the design and implementation of the Chinese event extractor. Finally, we analyze the testing results, draw a conclusion and discuss the future work.

II. Hidden Markov Models

Hidden Markov Models (HMM) [1] is a sequence model. A sequence model or sequence classifier is a model whose job is to assign a label or class to each unit in a sequence, that is mapping a sequence of observations to a sequence of labels. A HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), they compute a probability distribution over possible sequences of labels and choose the best label sequence by maximum likelihood estimation.

A. Markov chain

To define it properly, we need to introduce the Markov chain first, sometimes called the observed Markov model. A Markov chain is a special case of a weighted automaton in which weights are probabilities (the probabilities on all arcs leaving a node must sum to 1) and in which the input sequence uniquely determines which states the automaton will go through.

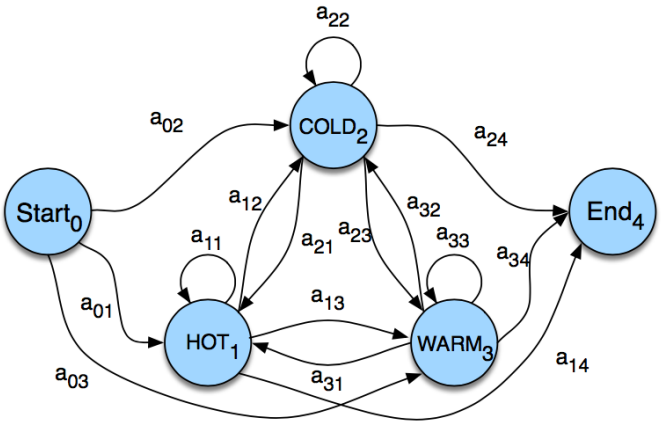

Fig. 1. A Markov chain for weather, which can be specified by the graph structure, the transition between states, and the distribution over starting state probabilities.

This Markov chain should be familiar; in fact, it represents a bigram language model. Given models in Fig. 1, we can assign a probability to any sequence from our vocabulary. A Markov chain is specified by the following components from Table I.

TABLE I
A Markov chain's components

S= $s_1 \cdots s_n$	a set of n **states**
A = $a_{01}a_{02} \cdots a_{n1} \cdots a_{nn}$	a **transition** probability matrix A, each a_{ij} representing the probability of moving from state i to state j
π	an initial probability distribution over states

B. Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of events that we can observe in the world. In many cases, however, the events we are interested in may not be directly observable in the world. For example, we didnt observe Part-Of-Speech tags in the world; we saw words and had to infer the correct tags from the word sequence. We call the Part-Of-Speech tags hidden because they are not observed.

Hidden Markov model (HMM) allows us to talk about both observed events (like words that we see in the input) and hidden events (like Part-Of-Speech tags) that we think of as causal factors in our probabilistic model.

An HMM is specified by the following components from Table II.

A first-order hidden Markov model instantiates **two simplifying assumptions**. First, as with a first-order Markov

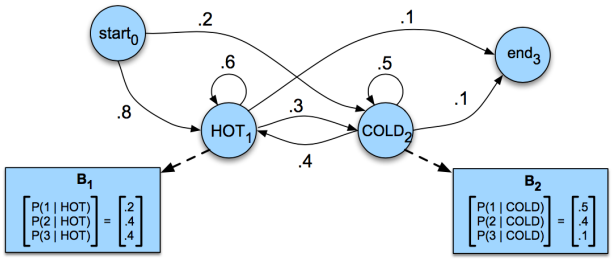

Fig. 2. A hidden Markov model for relating numbers of ice creams eaten by Caroline (the observations) to the weather (H or C, the hidden variables).

TABLE II
HMM'S COMPONENTS

S= $s_1 \cdots s_n$	a set of n **states**
A = $a_{01}a_{02} \cdots a_{n1} \cdots a_{nn}$	a **transition** probability matrix A, each a_{ij} representing the probability of moving from state i to state j
O = $o_1 o_2 \cdots o_T$	a sequence of T observations
B=$b_i(o_t)$	a sequence of observation likelihoods, also called emission probabilities, each expresses the probability of an observation o_t being generated from a state i
π	initial probability distribution over states

chain, the probability of a particular state depends only on the previous state:

$$P(s_i|s_1 \cdots s_{i1}) = P(s_i|s_{i1})$$

Second, the probability of an output observation o_i depends only on the state that produced the observation s_i and not on any other states or any other observations:

$$P(o_i|s_1 \cdots s_i, \cdots, s_T, o_1, \cdots, o_i, \cdots, o_T) = P(o_i|s_i)$$

Notice that in the HMM in Fig. 2, there is a (non-zero) probability of transitioning between any two states. Such an HMM is called a **fully connected** or ergodic HMM. Sometimes, however, we have HMMs in which many of the transitions between states have **zero probability**, and we should do smoothing ourselves.

For Chinese event extraction tasks, it's a **Decoding problem**, i.e. given an observation sequence O and an HMM $\lambda = (A, B, \pi)$, discover the best hidden state sequence Q.

We use the Viterbi Algorithm to to do decoding problems.

C. *Viterbi Algorithm*

Viterbi is a kind of dynamic programming that makes uses of a dynamic programming trellis. The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $s_0, s_1, \cdots, s_{t1}$, given the automaton λ . The value of each cell $v_t(j)$ is computed by **recursively** taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max P(s_0, s_1, \cdots, s_{t-1}, o_1, o_2, \cdots, o_t, s_t = j|\lambda)$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences. Given that we had already computed the probability of being in every state at time t-1, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state s_j at time t, the value $v_t(j)$ is computed as

$$v_t(j) = \max v_{t-1}(i) a_{ij} b_j(ot)$$

The three factors that are multiplied in the above equation for extending the previous paths to compute the Viterbi probability at time t are as the following Table III.

TABLE III
VITERBI'S COMPONENTS

$v_{t-1}(i)$	the **previous Viterbi path probability**
a_{ij}	the transition probability from previous state s_i to current state s_j
$b_i(o_t)$	the state observation likelihood of the observation o_t given the current state j

III. THINGS BEFORE MODELING

Before we get started, we should consider several questions:

1) What algorithms should we choose?
2) How can we find the balance between using rules and improving the programs' speed?
3) How can we find the balance between accuracy and speed?
4) Are we over-fitting dataset?

We'll answer those questions in the following sections.

IV. DESIGN OF EVENT EXTRACTOR

As a supervised algorithm, the Chinese event extractor needs a training set with pairs of (states, observations) to do probability calculation, and we should pay extra attention to smoothing in order to avoid zero values.

The training process of the Chinese event extractor includes following steps:

A. *Preparation of Training Set*

Reading *argument_train.txt* and *trigger_train.txt* from disk, simply store words and count into dicts. Then I use *pickle* package to store the already processed data into disk in order to facilitate training process.

The form of processed dicts as the following Table IV.

TABLE IV
PRE-PROCESSING

states	{'Rainy', 'Sunny'}
observations	{'walk', 'shop', 'clean'}
start_probability	{'Rainy': 6, Sunny': 4}
transition_probability	'Rainy': {'Rainy': 7, 'Sunny': 3}, 'Sunny': {'Rainy': 4, 'Sunny': 6}
emission_probability	'Rainy':{'walk': 1, 'shop': 4, 'clean': 5}, 'Sunny': {'walk': 6, 'shop': 3, 'clean': 1}

Notice: We should pay extra attention to the encoding and decoding problems in Chinese.

B. *Probability Calculation*

Considering the HMM we have is not a fully connected one, in which many of the transitions between states have zero probability. Hence the probability calculation isn't simply doing $P = \frac{n}{N}$, we should do smoothing as well as add **'UNK'** item into the dict.

We choose **add** λ **smoothing** methods.

V. Training and Testing

A testing set is a set of samples with pre-tagged label values as the training set and is used to determine the sequence labeling accuracy of the event extractor. By comparing the label value assigned by the extractor with the pre-tagged label value, we can compute the average accuracy, type_correct, precision, recall, and F1 score of the extractor.

A. *Use Dev-set*

The corpus is divided into Development set and Test set, and to go a step further, we divided the Development set into Training set and Dev-Test set.

That is, we do training on Training set, testing, adjusting parameters and choosing the best classifier base on the results from Dev-Test set. At last, we use our trained best extractor to do test on testing set.

B. *Viterbi algorithm*

The training of the Chinese event extraction is the process of computing probabilities require by:

$$v_t(j) = \max P(s_0, s_1, \cdots, s_{t-1}, o_1, o_2, \cdots, o_t, s_t = j|\lambda)$$

which can be presented as

$$v(s_0, s_1, \cdots, s_t) = \arg\max \prod_{i=1}^{t} A(s_i|s_{i-1}) \prod_{i=1}^{t} B(o_i|s_i)$$

The algorithm procedure can be described as following:

Input: a sentence $x_1 \ldots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.
Initialization: Set $\pi(0, *, *) = 1$, and $\pi(0, u, v) = 0$ for all (u, v) such that $u \neq *$ or $v \neq *$.
Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{K}$, $v \in \mathcal{K}$,

$$\pi(k, u, v) = \max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

- **Return** $\max_{u \in \mathcal{K}, v \in \mathcal{K}} (\pi(n, u, v) \times q(\text{STOP}|u, v))$

Fig. 3. The basic Viterbi Algorithm.

VI. Using CRF Toolkit

A. *Generating POS lists*

Jieba tokenizer creates a word list and Part-Of-Speech list for every sentence by tokenizing it. Hence I simply use *pseg.cut(sentence)* and write the processed line into disk.

B. *Write my own template*

Since I've added a column containing Part-Of-Speech features, special template needed to be implemented.

VII. Result Analysis

A. *Different smoothing methods for Bigram HMM*

*1) **trigger extraction**:* By changing λ, I got:

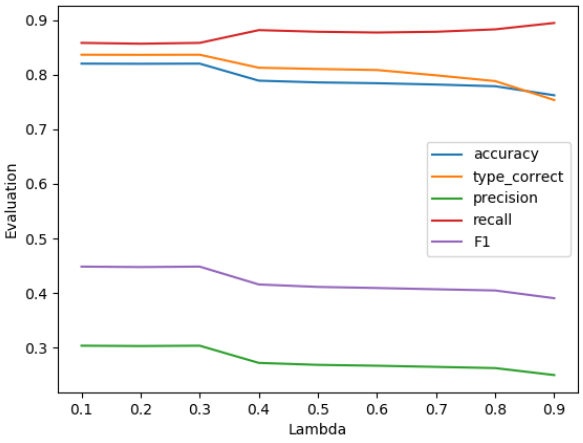

Fig. 4. The accuracy using add λ smoothing method

We can see, the change of λ doesn't affect the performance of bigram HMM much and it reached its convergence early, I assume the reason is as the followings:

- The types of trigger are not large (only 8 types)
- Only one word will be given a trigger tag in a sentence, hence there are a lot of 'O' tag, which means it's more likely for HMM to classify the right one to label and label a correct type

The best performances of HMM for Trigger was achieved at $\lambda = 0.1$

TABLE V
Best performance of bigram HMM for Trigger

accuracy	type_correct	precision	recall	F1
0.8204	0.835	0.3036	0.8584	0.4485

*2) **argument extraction**:* By changing λ, I find the performance varies enormously.

TABLE VI
Different performance of argument labeling

λ	accuracy	type_correct	precision	recall	F1
10^{-1}	0.4266	0.0505	0.4111	0.9929	0.5815
10^{-2}	0.6076	0.1986	0.5064	0.8678	0.6396
10^{-3}	0.6812	0.2803	0.5781	0.7599	0.6567
10^{-3}	0.6902	0.2989	0.5916	0.7357	0.6558
10^{-4}	0.6913	0.2997	0.5945	0.7254	0.6534
10^{-5}	0.6921	0.3001	0.5955	0.7244	0.6537
10^{-6}	**0.6923**	**0.3002**	0.5959	0.7241	0.6538
10^{-7}	0.6923	0.3002	0.5959	0.7241	0.6538
10^{-8}	0.6923	0.3002	0.5959	0.7241	0.6538

We can see, its convergence was reached at **0.3002**, and then the decrease of λ doesn't affect the performance. I assume this phenomenon is due to the limitation of **add** λ **smooth**.

B. Trigram HMM

Intuitively we take it that trigram models will outperform bigram models, hence apart from bigram HMM, I implemented another trigram HMM.

Notice: We should pay extra efforts to deal with the initial state, which is represented by *****, and the final state, which is represented by **'STOP'**.

This time I use **back-off smoothing**, and performances vary from the different combination of $\lambda_1, \lambda_2, \lambda_3$, by changing the relative magnitude of λ, we got results as follows:

TABLE VII
PERFORMANCE OF TRIGRAM HMM FOR TRIGGER

λ_1:λ_2:λ_3	accuracy	type_correct	precision	recall	F1
1:1:1	0.9615	0.9662	0.8284	0.6905	0.7532
7:2:1	0.9615	0.9662	0.8284	0.6905	0.7532
1:2:7	0.9617	0.9684	0.7959	0.7401	0.7671
1:1:8	0.9624	**0.9686**	0.7959	0.7431	0.7706

TABLE VIII
PERFORMANCE OF TRIGRAM HMM FOR ARGUMENT

λ_1:λ_2:λ_3	accuracy	type_correct	precision	recall	F1
1:1:1	0.7188	0.4117	0.6853	0.5530	0.6121
7:2:1	0.7265	0.4047	0.6907	0.5763	0.6283
1:2:7	0.7081	0.4200	0.6881	0.4981	0.5779
1:1:8	0.7067	**0.4367**	0.6926	0.4834	0.5694

It is natural for us to draw a conclusion that trigram HMM has better performances than bigram HMM.

However, it is interesting to see that different combinations of λ will affect the performances for trigger and argument differently.

C. CRF Toolkit

After add the Part-Of-Speech features, I've tried several command line parameters like *-f 3 -c 1.5* and find the performances are as followings:

TABLE IX
BEST PERFORMANCE OF CRF

	accuracy	type_correct	precision	recall	F1
trigger	0.9468	0.9547	0.9707	0.3869	0.5532
argument	0.7165	0.4334	0.7529	0.4369	0.5529

CRF is more stable and has higher performances for both trigger and argument tasks.

VIII. CONCLUSION

The main objective of this project is to apply HMM models on Chinese event extraction. We explored Markov chains, both bigram and trigram Hidden Markov models and Viterbi algorithm for event extraction. Also, we explored methods used for text preprocessing, probability computation, dealing with zero values and so on.

In this project, we have learned the advantage and easy-to-use feature of HMM and CRF. However, there does remain scope for improving the performance of the Chinese event extractor. For example, we can use Neural networks like LSTM to do sequence to sequence tasks.

REFERENCES

[1] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of IEEE.