CS134 (Fall 2015) Programming Assignment 3
Linear-Chain Conditional Random Fields

Due November 17, 2015

Back in 2001 in the city of the Brotherly Love, arise Linear-Chain Conditional Random Fields for NLP. And the model is so robust and easy enough to train that it still stays around these days. It is very likely that you will never have to implement CRF ever as there are a lot of good packages out there for it. This assignment will really help you understand the model and diagnose the problems when things go wrong when you use CRF in the future.

Getting Ready

1. Download the starter code and the data.
2. Make sure you understand the forward, backward, Viterbi, and Baum-Welch algorithms.
3. Look at the slides from the tutorial and the slides from lecture.

Datasets

We are giving you all another privilege to check. The privilege to tokenize your language by whitespace (no pun intended). And this privilege ends now. The language that my Papa and Mama taught me has no such convenient word boundaries. We are building a word segmenter for Thai. What does Thai look like?

A Thai sentence looks like this.

เต้นั่งรถไฟฟ้าไปโรงเรียน	Raw string
เ ต ้ น ั ่ ง ร ถ ไ ฟ ฟ ้ า ไ ป โ ร ง เ ร ี ย น	Character segmented
เต้ นั่ง รถ ไฟ ฟ้า ไป โรง เรียน	Syllable segmented
เต้ นั่ง รถไฟฟ้า ไป โรงเรียน	Word segmented
Te sits electric train to school	Literally translated word by word
Te takes the metro to school	Properly translated

A few things to note here

- Each character is a consonant or a vowel or a tone marker (indicating the pitch of the syllable).
- Some vowels can only be in the front of the syllable e.g. เ ไ โ
- Some vowels can never be in the front of the syllable e.g. ั า
- A word can be multi-syllabic (or implicitly multi-word). e.g. the word for metro consists of three meaningful monosyllabic words. รถ ไฟ ฟ้า = car + fire + sky → metro
- Dictionaries cannot solve everything e.g. เต้ ‘Te’ is never in any Thai dictionary.

You are given a bunch of Thai phrases or sentences. Each character is marked with the following:

- First column : The character itself.
- Second column : Character type :
    - t = tone marker
    - w = vowels must be followed by a consonant
    - c = consonant
    - n = consonant that cannot be in the final position
    - p = punctuation
    - e = end of sentence
    - s = silencer
- Third column : BIO tag for the character.
  B = the beginning of a word
  I = the inside of a word
  O = not part of a word. The O tag can be trivially identified. It is associated mostly with punctuation and end of sentence (EOS) mark. The O tag is therefore used a sequence boundary.

Implementing Linear Chain Conditional Random Fields

You are once again welcome to make any changes to the starter code. This is a pretty heavy assignment so we try to give you as much code as we can. That means your implementation will have to comply to *some* format to make it work with the implementation. But again you should really feel free to change things around.

Here are the main things you have to implement

- Forward algorithm
- Backward algorithm
- Viterbi algorithm

Here's what we have done for you. These should work if you comply to what the functions require as input:

- A primitive implementation of minibatch stochastic gradient
- Observed count computation
- Expected count computation (Baum-Welch algorithm) but it needs your implementation of forward and backward algorithms

More instructions are in the starter code. Take your time to read everything and convince yourself you know how the pieces work together.

Experiments

Experimentation helps you develop good classifiers/taggers as it gives some evidence on what features help and/or what part of the model helps. In general, the improvement comes from 1) playing with the model choices e.g. CRF vs MaxEnt, 2) the feature sets, and 3) the label sets. You are asked to conduct one of these experiments:

Option 1 : Model choices
Run a set of experiments to compare CRF and MaxEnt under a few various feature sets and answer these questions based on your experiment results.

- What are the main differences between CRF and MaxEnt in terms of speed and accuracy during training and classification?
- In the context of this task, is it worth it to use CRF instead of MaxEnt?

Option 2 : Feature sets
Features usually have the most impact on the performance but engineering it takes a bit of experimentation and some linguistic knowledge/intuition. Play with a few feature sets and compare the performance. Here are some suggestions of what you can do:

- Use n-gram of characters (position sensitive or not) or n-gram of character types (position sensitive or not). These are sometimes called *conjunctive* features. They essentially make the model memorize some common characters that might form a word.
- Recode character types based on linguistic knowledge. Vowels in Thai have some peculiarity e.g. it can be composed of multiple characters, some vowels must have final consonants. etc. https://en.wikipedia.org/wiki/Thai_alphabet#Vowels
- Use lexicon as a feature. If the previous characters match a word in a dictionary, they should form a word. Email Te for the Thai lexicon.

Run a series of experiments to show the influence and the effectiveness of each combination of feature set.

Option 3 : Label sets
Try modifying the label sets and comparing the performance from a few different label sets. Here are some suggestions:

- The current label set only distinguishes the beginning of the word and the non-beginning of the word. Maybe it might be a good idea to the detect the end of the word separately as another label.
- Or it might be a good idea to detect the second character in the word as a separate label.

Run a series of experiments to show which labeling scheme works the best. Discuss why your experiments turn out the way they do.

Tips and other advice

This is a tricky assignment as each function seems modular but it's not. They will have to work together.

- Take time to read the whole starter code to get the idea of what needs to happen.
- Use the basic test cases that we provide.
- Look at compute_observed_count to see how to use the feature_vector in each timestep.
- Draw pictures and diagrams to visualize the matrices that you are manipulating.
- Each function you need to implement is < 10 lines of code. So you should spend most of the time making sure you understand the algorithms. Once you understand them, it should take almost no time.
- Indices will kill you. Be extra cautious with how you start counting time. What goes into the columns or rows? If something goes wrong in your algorithm, it is most likely because of the indexing issues.
- When computing expected count, be aware of double counting transition matrices. Think about which matrices have been used already at each time step in alpha and beta.