Natural Language Processing
MEMMs and CRFs

Felipe Bravo-Marquez

September 15, 2019

MEMMs

- The goal of sequence labeling is to assign tags to words, or more generally, to assign discrete labels to discrete elements in a sequence [**?**].
- Well known examples of this problem are: part-of-speech tagging (POS) and Named Entity Recognition (NER).
- Maximum-entropy Markov models (MEMMs) make use of log-linear models for sequence labeling tasks.
- In the early NLP literature, logistic regression was often called maximum entropy classification [**?**].
- Hence, MEMMs will look very similar to the multi-class softmax models seen in the lecture about linear models.
- In contrast to HMMs, here we rely on parameterized functions.

MEMMs

- The goal of MEMMs is model the following conditional distribution:

$$P(s_1, s_2 \ldots, s_m | x_1, \ldots, x_m)$$

- Where each x_j for $j = 1 \ldots m$ is the j-th input symbol (for example the j-th word in a sentence), and each s_j for $j = 1 \ldots m$ is the j-th tag.[1]
- We expecto this P(DET,NOUN,VERB|the,dog,barks) to be higher than P(VERB,VERB,VERB|the,dog,barks)

[1] These slides are based on lecture notes of Michael Collins `http://www.cs.columbia.edu/~mcollins/`. The notation and terminology has been adapted to be consistent with the other material.

MEMMs

- We use S to denote the set of possible tags.
- We assume that S is a finite set.
- For example, in part-of-speech tagging of English, S would be the set of all possible parts of speech in English (noun, verb, determiner, preposition, etc.).
- Given a sequence of words $x_1, \ldots, x_m$, there are k^m possible part-of-speech sequences $s_1, \ldots, s_m$, where $k = |S|$ is the number of possible parts of speech.
- We want to estimate a distribution over these k^m possible sequences.

MEMMs

- In a first step, MEMMs use the following decomposition:

$$\begin{aligned} P(s_1, s_2 \ldots, s_m | x_1, \ldots, x_m) &= \prod_{i=1}^{m} P(s_i | s_1 \ldots, s_{i-1}, x_1, \ldots, x_m) \\ &= \prod_{i=1}^{m} P(s_i | s_{i-1}, x_1, \ldots, x_m) \end{aligned} \tag{1}$$

- The first equality is exact (it follows by the chain rule of conditional probabilities).
- The second equality follows from an independence assumption, namely that for all i,

$$P(s_i | s_1 \ldots, s_{i-1}, x_1, \ldots, x_m) = P(s_i | s_{i-1}, x_1, \ldots, x_m)$$

MEMMs

- Hence we are making a first order Markov assumption similar to the Markov assumption made in HMMs.
- The tag in the i-th position depends only on the tag in the $(i-1)$-th position.
- Having made these independence assumptions, we then model each term using a log-linear (Softmax) model:

$$P(s_i|s_{i-1}, x_1, \ldots, x_m) = \frac{\exp(\vec{w} \cdot \vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i))}{\sum_{s' \in S} \exp(\vec{w} \cdot \vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s'))} \tag{2}$$

MEMMs

Here $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)$ is a feature vector where:

- $x_1, \cdots, x_m$ is the entire sentence being tagged.
- i is the position to be tagged (can take any value from 1 to m)
- s is the previous tag value (can take any value in S).
- s' is the new tag value (can take any value in S)

Example of Features used in Part-of-Speech Tagging

1. $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[1]} = 1$ if s_i = ADVERB and word x_i ends in "-ly"; 0 otherwise.
   If the weight $\vec{w}_{[1]}$ associated with this feature is large and positive, then this feature is essentially saying that we prefer labelings where words ending in -ly get labeled as ADVERB.
2. $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[2]} = 1$ if $i = 1$, s_i= VERB, and x_m=?; 0 otherwise.
   If the weight $\vec{w}_{[2]}$ associated with this feature is large and positive, then labelings that assign VERB to the first word in a question (e.g., "Is this a sentence beginning with a verb?") are preferred.
3. $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[3]} = 1$ if s_{i-1}= ADJECTIVE and l_i= NOUN; 0 otherwise.
   Again, a positive weight for this feature means that adjectives tend to be followed by nouns.
4. $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[4]} = 1$ if s_{i-1}= PREPOSITION and s_i= PREPOSITION.
   A negative weight $\vec{w}_{[4]}$ for this function would mean that prepositions don't tend to follow prepositions.

[2]Source: `https://blog.echen.me/2012/01/03/introduction-to-conditional-random-fields/`

Feature Templates

It is possible to define more general feature templates covering unigrams, bigrams, n-grams of words as well as tag values of s_{i-1} and s_i.

1. A word unigram and tag unigram feature template:
   $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[hash(j,k)]} = 1$ if s_i = TAG$_{[j]}$ and x_i = WORD$_{[k]}$; 0 otherwise $\forall j, k$.
   Notice that j is and index spanning all possible tags in S and k is another index spanning the words in the vocabulary V.
2. A word bigram and tag bigram feature template:
   $\vec{\phi}(x_1, \cdots, x_m, i, s_{i-1}, s_i)_{[hash(j,k,u,v)]} = 1$ if s_{i-1} = TAG$_{[j]}$ and s_i = TAG$_{[k]}$ and x_{i-1} = WORD$_{[u]}$ and x_i = WORD$_{[v]}$; 0 otherwise $\forall j, k, u, v$.

The function $hash(j, k, ...)$ will map each different feature to a unique index in the feature vector.

Notice that the resuling vector will be very high-dimensional and sparse.

MEMMs and Multi-class Softmax

- Notice that the log-linear model from above is very similar to the multi-class softmax model presented in the lecture about linear models.
- A general log-linear model has the following form:

$$P(y|x;\vec{w}) = \frac{\exp(\vec{w}\cdot\vec{\phi}(x,y))}{\sum_{y'\in Y}\exp(\vec{w}\cdot\vec{\phi}(x,y'))}$$

- A multi-class softmax model has the following form:

$$\begin{aligned}\hat{\vec{y}} &= \text{softmax}(\vec{x}\cdot W + \vec{b}) \\ \hat{\vec{y}}_{[i]} &= \frac{e^{(\vec{x}\cdot W+\vec{b})_{[i]}}}{\sum_j e^{(\vec{x}\cdot W+\vec{b})_{[j]}}}\end{aligned} \tag{3}$$

MEMMs and Multi-class Softmax

- Difference 1: in the log-linear model we have a fixed parameter vector $\vec{w}$ instead of having multiple vectors (one column of W for each class value).
- Difference 2: the feature vector of the log-linear model $\vec{\phi}(x, y)$ includes information of the label y, whereas the input vecotr $\vec{x}$ of the softmax model is independent of y.
- Log-linear models allow using features that consider the interaction between x and y (e.g., x end in "ly" and y is an ADVERB).

Training MEMMs

- Once we've defined the feature vectors $\vec{\phi}$, we can train the parameters $\vec{w}$ of the model in the usual way for linear models.
- We set the negative log-likelihood as the loss function and optimize parameters using gradient descent from the training examples.
- This is equivalent as using the cross-entropy loss.
- "Any loss consisting of a negative log-likelihood is a cross-entropy between the empirical distribution defined by the training set and the probability distribution defined by model" [**?**].

Decoding with MEMMs

- The decoding problem is as follows.
- We are given a new test sequence $x_1, \ldots, x_m$.
- Our goal is to compute the most likely state sequence for this test sequence,

$$\text{argmax}_{s_1,\ldots,s_m} P(s_1, \ldots, s_m | x_1, \ldots, x_m). \tag{4}$$

- There are k^m possible state sequences, so for any reasonably large sentence length m brute-force search through all the possibilities will not be possible.
- We can use the Viterbi alogrithm in a similar way as used for HMMs.

Decoding with MEMMs

- The basic data structure in the algorithm will be a dynamic programming table π with entries $\pi[j, s]$ for $j = 1, \ldots, m$, and $s \in S$.
- $\pi[j, s]$ will store the maximum probability for any state sequence ending in state s at position j.
- More formally, our algorithm will compute

$$\pi[j, s] = \max_{s_1, \ldots, s_{j-1}} \left(P(s | s_{j-1}, x_1, \ldots, x_m) \prod_{k=1}^{j-1} P(s_k | s_{k-1}, x_1, \ldots, x_m) \right)$$

for all $j = 1, \ldots, m$, and for all $s \in S$.

Links

- https://www.depends-on-the-definition.com/sequence-tagging-lstm-crf/
- https://www.quora.com/What-are-the-pros-and-cons-of-these-three-sequence-models-MaxEnt-
- https://people.cs.umass.edu/~mccallum/papers/crf-tutorial.pdf

Questions?

Thanks for your Attention!

References I

Eisenstein, J. (2018).
Natural language processing.
Technical report, Georgia Tech.

Goodfellow, I., Bengio, Y., and Courville, A. (2016).
Deep learning.
MIT press.