Natural Language Processing
MEMMs and CRFs

Felipe Bravo-Marquez

September 30, 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 [Eisenstein, 2018].
- 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 multi-class models for sequence labeling tasks.
- In the early NLP literature, logistic regression was often called maximum entropy classification [Eisenstein, 2018].
- 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/crf.pdf`. The notation and terminology has been adapted to be consistent with the rest of the course.

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 (s_0 has always a special tag $*$):

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

- 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 multiclass log-linear (softmax) model:

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

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

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 ends 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” [Goodfellow et al., 2016].

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,

$$\arg\max_{s_1,\ldots,s_m} P(s_1, \ldots, s_m | x_1, \ldots, x_m). \tag{4}$$

- 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 $s \in S$.

Decoding with MEMMs

The algorithm is as follows:

- Initialization: for $s \in S$

$$\pi[1, s] = P(s | s_0, x_1, \ldots, x_m)$$

  where s_0 is a special “initial” state.

- For $j \in \{2, \ldots, m\}$, $s \in \{1, \ldots, k\}$

$$\pi[j, s] = \max_{s' \in S} \left[\pi[j-1, s'] \times P(s | s', x_1, \ldots, x_m)\right]$$

Finally, having filled in the $\pi[j, s]$ values for all j, s, we can calculate

$$\max_{s_1, \ldots, s_m} = \max_{s} \pi[m, s].$$

Decoding with MEMMs

- Finally, having filled in the $\pi[j, s]$ values for all j, s, we can calculate

$$\max_{s_1,\ldots,s_m} = \max_{s} \ \pi[m, s].$$

- The algorithm runs in $O(mk^2)$ time (i.e., linear in the sequence length m, and quadratic in the number of tags k).
- As in the Viterbi algorithm for HMMs, we can compute the highest-scoring sequence using backpointers in the dynamic programming algorithm.

Comparison between MEMMs and HMMs

- So what is the motivation for using MEMMs instead of HMMs?
- Note that the Viterbi decoding algorithms for the two models are very similar.
- In MEMMs, the probability associated with each state transition s_{i-1} to s_i is

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

- In HMMs, the probability associated with each transition is:

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

Comparison between MEMMs and HMMs

- The key advantage of MEMMs is that the use of feature vectors $\vec{\phi}$ allows much richer representations than those used in HMMs.
- For example, the transition probability can be sensitive to any word in the input sequence $x_1, \ldots, x_m$.
- In addition, it is very easy to introduce features that are sensitive to spelling features (e.g., prefixes or suffixes) of the current word x_i , or of the surrounding words.
- These features are useful in many NLP applications, and are difficult to incorporate within HMMs in a clean way.

Conditional Random Fields (CRFs)

- We now turn to Conditional Random Fields (CRFs).
- Notation: for convenience, we'll use $x_{1:m}$ to refer to an input sequence $x_1, \ldots, x_m$, and $s_{1:m}$ to refer to a sequence of tags $s_1, \ldots, s_m$.
- The set of all possible tags is again S.
- The set of all possible tag sequences is S^m.
- In conditional random fields we'll again build a model of

$$P(s_1, \ldots, s_m | x_1, \ldots, x_m) = P(s_{1:m} | x_{1:m})$$

Conditional Random Fields (CRFs)

- A first key idea in CRFs will be to define a feature vector that maps an entire input sequence $x_{1:m}$ paired with an entire tag sequence $s_{1:m}$ to some d-dimensional feature vector:

$$\vec{\Phi}(x_{1:m}, s_{1:m}) \in \mathcal{R}^d$$

- We'll soon give a concrete definition for $\vec{\Phi}$.
- For now just assume that some definition exists.
- We will often refer to $\vec{\Phi}$ as being a "global" feature vector.
- It is global in the sense that it takes the entire state sequence into account.

Conditional Random Fields (CRFs)

- In CRFs we build a giant log-linear model:

$$P(s_{1:m}|x_{1:m};\vec{w}) = \frac{\exp(\vec{w}\cdot\vec{\Phi}(x_{1:m},s_{1:m}))}{\sum_{s'_{1:m}\in S^m}\exp(\vec{w}\cdot\vec{\Phi}(x_{1:m},s'_{1:m}))}$$

- This is “just” another log-linear model, but it is is “giant”.
- The space of possible values for $s_{1:m}$ is huge S^m.
- The normalization constant (denominator in the above expression) involves a sum over all possible tag sequences S^m.
- These issues might seem to cause severe computational problems.
- Under appropriate assumptions we can train and decode efficiently with this type of model.

Conditional Random Fields (CRFs)

- We define the global feature vector $\vec{\Phi}(x_{1:m}, s_{1:m})$ as follows:

$$\vec{\Phi}(x_{1:m}, s_{1:m}) = \sum_{j=1}^{m} \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)$$

  where $\vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)$ are the same as the feature vectors used in MEMMs.
- Example:
  $\vec{\Phi}(\text{[the,dog,barks]}, \text{DET,NOUN,VERB]}) = \vec{\phi}(\text{[the,dog,barks]}, 1, *, \text{DET}) + \vec{\phi}(\text{[the,dog,barks]}, 2, \text{DET}, \text{NOUN}) + \vec{\phi}(\text{[the,dog,barks]}, 3, \text{NOUN}, \text{VERB})$
- Essentially, we are adding up many sparse vectors.

Conditional Random Fields (CRFs)

- We are assuming that for any dimension of $\vec{\Phi}_{[k]}, k = 1, \ldots, d$, the k'th global feature is:

$$\vec{\Phi}(x_{1:m}, s_{1:m})_{[k]} = \sum_{j=1}^{m} \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)_{[k]}$$

- Thus $\vec{\Phi}(x_{1:m}, s_{1:m})_{[k]}$ is calculated by summing the "local" feature vector $\vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)_{[k]}$ over the m different tag transitions in $s_1, \ldots, s_m$.
- We would expect each local vector to encode relevant information about the tag transition by turning on some vector dimensions (setting the value to one).
- We now turn to two critical practical issues in CRFs: first, decoding, and second, parameter estimation (training).

Decoding with CRFs

- The decoding problem in CRFs is as follows.
- For a given input sequence $x_{1:m} = x_1, x_2, \ldots, x_m$, we would like to find the most likely underlying state sequence under the model, that is,

$$
\begin{aligned}
arg \max_{s_{1:m} \in S^m} P(s_{1:m} | x_{1:m}; \vec{w}) &= arg \max_{s_{1:m} \in S^m} \frac{\exp(\vec{w} \cdot \vec{\Phi}(x_{1:m}, s_{1:m}))}{\sum_{s'_{1:m} \in S^m} \exp(\vec{w} \cdot \vec{\Phi}(x_{1:m}, s'_{1:m}))} \\
&= arg \max_{s_{1:m} \in S^m} \exp(\vec{w} \cdot \vec{\Phi}(x_{1:m}, s_{1:m})) \\
&= arg \max_{s_{1:m} \in S^m} \vec{w} \cdot \vec{\Phi}(x_{1:m}, s_{1:m}) \\
&= arg \max_{s_{1:m} \in S^m} \vec{w} \cdot \sum_{j=1}^{m} \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j) \\
&= arg \max_{s_{1:m} \in S^m} \sum_{j=1}^{m} \vec{w} \cdot \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)
\end{aligned} \tag{5}
$$

Decoding with CRFs

- We have shown that finding the most likely sequence under the model is equivalent to finding the sequence that maximizes:

$$arg \max_{s_{1:m} \in S^m} \sum_{j=1}^{m} \vec{w} \cdot \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)$$

- This problem has a clear intuition. Each transition from tag s_{j-1} to tag s_j has an associated score: $\vec{w} \cdot \vec{\phi}(x_{1:m}, j, s_{j-1}, s_j)$
- This score could be positive or negative.
- Intuitively, this score will be relatively high if the state transition is plausible, relatively low if this transition is implausible.
- The decoding problem is to find an entire sequence of states such that the sum of transition scores is maximized.
- We can again solve this problem using a variant of the Viterbi algorithm, in a very similar way to the decoding algorithm for HMMs or MEMMs.

Parameter Estimation in CRFs (training)

- For parameter estimation, we assume we have a set of n labeled examples, $\{(x^i_{1:m}, s^i_{1:m})\}^n_{i=1}$. Each $x^i_{1:m}$ is an input sequence $x^i_1, \ldots, x^i_m$ each $s^i_{1:m}$ is a tag sequence $s^i_1, \ldots, s^i_m$.
- We again set the negative log-likelihood (or cross-entropy) as the loss function L as optimize parameters using gradient descent.
- The main challenge here is that gradient calculations $\frac{\partial L}{\partial \vec{w}_{[k]}}$ involve summing over S^m (a very large set containing all possible tag sequences).
- This sum can be computed efficiently using the Forward-backward algorithm[3].
- This is another dynamic programming algorithm that is closely related to the Viterbi algorithm.

[3] `http://www.cs.columbia.edu/~mcollins/fb.pdf`

CRFs and MEMMs

- CRFs and MEMMS are discriminative sequence labeling models: they model the conditional probability directly via a parameterized log-linear multi-class function (softmax).
- HMMs, on the other hand, are are generative models.
- In MEMM the normalization (denominator of the softmax) is local: it happens at each tag step (the sum runs over all posible tag values S).
- In CRFs the normalization is global: the sum runs over all possible tag sequences S^m.
- Training a MEMM is quite easy: just train a multi-class log-linear model for for a given word to the label. This classifier is used at each word step to predict the whole sequence.
- Training CRF is more complex. The objective is to maximize the log probability of the most likely sequence.

CRFs and MEMMs: the label bias problem

- MEMMs end up making up decision at each time step independently.
- This leads to a problem called label bias: in some tag space configurations, MEMMs essentially completely ignore important aspects of the context.
- Example: The right POS labeling of sentence “will to fight” (la voluntad de pelear) is “NN TO VB”. [4]
- Here NN stands for “noun”, TO stands for “infinitive to”, and VB stands for “verb base form”.
- Modals (MD) show up much more frequently at the start of the sentence than nouns do (e.g., questions).
- Hence, tag “MD” will receive a higher score than tag “NN” when x_0=“will” : $P(s_1 = MD|s_0 = *, x_1 = \text{“will”}, ...) > P(s_1 = NN|s_{i-1} = *, x_1 = \text{“will”})$.
- But we know that MD + TO is very rare: “... can to eat”, “... would to eat”.

[4] Here we are using the PENN Treebank tagset: `https://www.eecis.udel.edu/~vijay/cis889/ie/pos-set.pdf`

CRFs and MEMMs: the label bias problem

- The word “to” is relatively deterministic (almost always has tag TO) so it doesn’t matter what tag precedes it.
- Because of the local normalization of MEMMs, $P(s_i = TO | s_{i-1}, x_1, \ldots, x_i = \text{“to”}, \ldots, x_n)$ will always be 1 when $x_i =$“to” regardless of the value of s_{i-1} (MD or NN).
- That means our prediction for “to” can’t help us disambiguate “will”.
- We lose the information that MD + TO sequences rarely happen.
- As a consequence: a MEMMS would likely label the first word to “MD”.
- CRF overcomes this issue by doing a global normalization: it considers the score of the whole sequence before normalizing to make it a probability distribution.

Links

- http://people.ischool.berkeley.edu/~dbamman/nlpF18/slides/11_memm_crf.pdf
- http://people.ischool.berkeley.edu/~dbamman/nlpF18/slides/12_neural_sequence_labeling.pdf
- 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
- http://www.davidsbatista.net/blog/2017/11/13/Conditional_Random_Fields

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.