Chapter 8

MEMMs (Log-Linear Tagging Models)

8.1 Introduction

In this chapter we return to the problem of tagging. We previously described hidden Markov models (HMMs) for tagging problems. This chapter describes a powerful alternative to HMMs, *log-linear tagging models*, which build directly on ideas from log-linear models. A key advantage of log-linear tagging models is that they allow highly flexible representations, allowing features to be easily integrated in the model.

Log-linear tagging models are sometimes referred to as "maximum entropy Markov models (MEMMs)".[1] We will use the terms "MEMM" and "log-linear tagging model" interchangeably in this chapter. The name MEMM was first introduced by McCallum et al. (2000).

Log-linear tagging models are conditional tagging models. Recall that a generative tagging model defines a joint distribution $p(x_1 \dots x_n, y_1 \dots y_n)$ over sentences $x_1 \dots x_n$ paired with tag sequences $y_1 \dots y_n$. In contrast, a conditional tagging model defines a conditional distribution

$$p(y_1 \dots y_n | x_1 \dots x_n)$$

corresponding to the probability of the tag sequence $y_1 \dots y_n$ conditioned on the input sentence $x_1 \dots x_n$. We give the following definition:

[1] This name is used because 1) log-linear models are also referred to as maximum entropy models, as it can be shown in the unregularized case that the maximum likelihood estimates maximize an entropic measure subject to certain linear constraints; 2) as we will see shortly, MEMMs make a Markov assumption that is closely related to the Markov assumption used in HMMs.

Definition 1 (Conditional Tagging Models) *A conditional tagging model consists of:*

- *A set of words $\mathcal{V}$ (this set may be finite, countably infinite, or even uncountably infinite).*
- *A finite set of tags $\mathcal{K}$.*
- *A function $p(y_1 \dots y_n | x_1 \dots x_n)$ such that:*

  1. *For any $\langle x_1 \dots x_n, y_1 \dots y_n \rangle \in \mathcal{S}$,*

  $$p(y_1 \dots y_n | x_1 \dots x_n) \geq 0$$

  where $\mathcal{S}$ is the set of all sequence/tag-sequence pairs $\langle x_1 \dots x_n, y_1 \dots y_n \rangle$ such that $n \geq 1$, $x_i \in \mathcal{V}$ for $i = 1 \dots n$, and $y_i \in \mathcal{K}$ for $i = 1 \dots n$.

  2. *For any $x_1 \dots x_n$ such that $n \geq 1$ and $x_i \in \mathcal{V}$ for $i = 1 \dots n$,*

  $$\sum_{y_1 \dots y_n \in \mathcal{Y}(n)} p(y_1 \dots y_n | x_1 \dots x_n) = 1$$

  where $\mathcal{Y}(n)$ is the set of all tag sequences $y_1 \dots y_n$ such that $y_i \in \mathcal{K}$ for $i = 1 \dots n$.

Given a conditional tagging model, the function from sentences $x_1 \dots x_n$ to tag sequences $y_1 \dots y_n$ is defined as

$$f(x_1 \dots x_n) = \arg \max_{y_1 \dots y_n \in \mathcal{Y}(n)} p(y_1 \dots y_n | x_1 \dots x_n)$$

Thus for any input $x_1 \dots x_n$, we take the highest probability tag sequence as the output from the model. □

We are left with the following three questions:

- How we define a conditional tagging model $p(y_1 \dots y_n | x_1 \dots x_n)$?
- How do we estimate the parameters of the model from training examples?
- How do we efficiently find

  $$\arg \max_{y_1 \dots y_n \in \mathcal{Y}(n)} p(y_1 \dots y_n | x_1 \dots x_n)$$

  for any input $x_1 \dots x_n$?

The remainder of this chapter describes how MEMMs give a solution to these questions. In brief, a log-linear model is used to define a conditional tagging model. The parameters of the model can be estimated using standard methods for parameter estimation in log-linear models. MEMMs make a Markov independence assumption that is closely related to the Markov independence assumption in HMMs, and that allows the $\arg\max$ above to be computed efficiently using dynamic programming.

8.2 Trigram MEMMs

This section describes the model form for MEMMs. We focus on *trigram* MEMMs, which make a second order Markov assumption, where each tag depends on the previous two tags. The generalization to other Markov orders, for example first-order (bigram) or third-order (four-gram) MEMMs, is straightforward.

Our task is to model the conditional distribution

$$P(Y_1 = y_1 \dots Y_n = y_n | X_1 = x_1 \dots X_n = x_n)$$

for any input sequence $x_1 \dots x_n$ paired with a tag sequence $y_1 \dots y_n$.

We first use the following decomposition:

$$\begin{aligned} & P(Y_1 = y_1 \dots Y_n = y_n | X_1 = x_1 \dots X_n = x_n) \\ = & \prod_{i=1}^{n} P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{i-1} = y_{i-1}) \\ = & \prod_{i=1}^{n} P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \end{aligned}$$

The first equality is exact, by the chain rule of probabilities. The second equality makes use of a trigram independence assumption, namely that for any $i \in \{1 \dots n\}$,

$$\begin{aligned} & P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{i-1} = y_{i-1}) \\ = & P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \end{aligned}$$

Here we assume that $y_{-1} = y_0 = *$, where * is a special symbol in the model denoting the start of a sequence.

Thus we assume that the random variable Y_i is independent of the values for $Y_1 \dots Y_{i-3}$, once we condition on the entire input sequence $X_1 \dots X_n$, and the previous two tags Y_{i-2} and Y_{i-1}. This is a trigram independence assumption, where each tag depends only on the previous two tags. We will see that this independence

assumption allows us to use dynamic programming to efficiently find the highest probability tag sequence for any input sentence $x_1 \dots x_n$.

Note that there is some resemblance to the independence assumption made in trigram HMMs, namely that

$$\begin{aligned} & P(Y_i = y_i | Y_1 = y_1 \dots Y_{i-1} = y_{i-1}) \\ = \quad & P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \end{aligned}$$

The key difference is that we now condition on the entire input sequence $x_1 \dots x_n$, in addition to the previous two tags y_{i-2} and y_{i-1}.

The final step is to use a log-linear model to estimate the probability

$$P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1})$$

For any pair of sequences $x_1 \dots x_n$ and $y_1 \dots y_n$, we define the i'th "history" h_i to be the four-tuple

$$h_i = \langle y_{i-2}, y_{i-1}, x_1 \dots x_n, i \rangle$$

Thus h_i captures the conditioning information for tag y_i in the sequence, in addition to the position i in the sequence. We assume that we have a feature-vector representation $f(h_i, y) \in \mathbb{R}^d$ for any history h_i paired with any tag $y \in \mathcal{K}$. The feature vector could potentially take into account any information in the history h_i and the tag y. As one example, we might have features

$$f_1(h_i, y) = \begin{cases} 1 & \text{if } x_i = \texttt{the} \text{ and } y = \texttt{DT} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(h_i, y) = \begin{cases} 1 & \text{if } y_{i-1} = \texttt{V} \text{ and } y = \texttt{DT} \\ 0 & \text{otherwise} \end{cases}$$

Section 8.3 describes a much more complete set of example features.

Finally, we assume a parameter vector $\theta \in \mathbb{R}^d$, and that

$$\begin{aligned} & P(Y_i = y_i | X_1 = x_1 \dots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \\ = \quad & \frac{\exp(\theta \cdot f(h_i, y_i))}{\sum_{y \in \mathcal{K}} \exp(\theta \cdot f(h_i, y))} \end{aligned}$$

Putting this all together gives the following:

Definition 2 (Trigram MEMMs) *A trigram MEMM consists of:*

- *A set of words $\mathcal{V}$ (this set may be finite, countably infinite, or even uncountably infinite).*
- *A finite set of tags $\mathcal{K}$.*
- *Given $\mathcal{V}$ and $\mathcal{K}$, define $\mathcal{H}$ to be the set of all possible histories. The set $\mathcal{H}$ contains all four-tuples of the form $\langle y_{-2}, y_{-1}, x_1 \dots x_n, i \rangle$, where $y_{-2} \in \mathcal{K} \cup \{*\}$, $y_{-1} \in \mathcal{K} \cup \{*\}$, $n \geq 1$, $x_i \in \mathcal{V}$ for $i = 1 \dots n$, $i \in \{1 \dots n\}$. Here * is a special "start" symbol.*
- *An integer d specifying the number of features in the model.*
- *A function $f : \mathcal{H} \times \mathcal{K} \rightarrow \mathbb{R}^d$ specifying the features in the model.*
- *A parameter vector $\theta \in \mathbb{R}^d$.*

Given these components we define the conditional tagging model

$$p(y_1 \dots y_n | x_1 \dots x_n) = \prod_{i=1}^{n} p(y_i | h_i; \theta)$$

where $h_i = \langle y_{i-2}, y_{i-1}, x_1 \dots x_n, i \rangle$, and

$$p(y_i | h_i; \theta) = \frac{\exp(\theta \cdot f(h_i, y_i))}{\sum_{y \in \mathcal{K}} \exp(\theta \cdot f(h_i, y))}$$

□

At this point there are a number of questions. How do we define the feature vectors $f(h_i, y)$? How do we learn the parameters θ from training data? How do we find the highest probability tag sequence

$$\arg \max_{y_1 \dots y_n \in \mathcal{Y}(n)} p(y_1 \dots y_n | x_1 \dots x_n)$$

for an input sequence $x_1 \dots x_n$? The following sections answer these questions.

8.3 Features in Trigram MEMMs

Recall that the feature vector definition in a trigram MEMM is a function $f(h, y) \in \mathbb{R}^d$ where $h = \langle y_{-2}, y_{-1}, x_1 \dots x_n, i \rangle$ is a history, $y \in \mathcal{K}$ is a tag, and d is an integer specifying the number of features in the model. Each feature $f_j(h, y)$ for $j \in \{1 \dots d\}$ can potentially be sensitive to *any* information in the history h in

conjunction with the tag y. This will lead to a great deal of flexibility in the model. This is the primary advantage of trigram MEMMs over trigram HMMs for tagging: a much richer set of features can be employed for the tagging task.

In this section we give an example of how features can be defined for the part-of-speech (POS) tagging problem. The features we describe are taken from Ratnaparkhi (1996); Ratnaparkhi's experiments show that they give competitive performance on the POS tagging problem for English. Throughout this section we assume that the history h is a four-tuple $\langle y_{-2}, y_{-1}, x_1 \ldots x_n, i \rangle$. The features are as follows:

Word/tag features One example word/tag feature is the following:

$$f_{100}(h, y) = \begin{cases} 1 & \text{if } x_i \text{ is } \texttt{base} \text{ and } y = \texttt{VB} \\ 0 & \text{otherwise} \end{cases}$$

This feature is sensitive to the word being tagged, x_i, and the proposed tag for that word, y. In practice we would introduce features of this form for a very large set of word/tag pairs, in addition to the pair base/VB. For example, we could introduce features of this form for all word/tag pairs seen in training data.

This class of feature allows the model to capture the tendency for particular words to take particular parts of speech. In this sense it plays an analogous role to the emission parameters $e(x|y)$ in a trigram HMM. For example, given the definition of f_{100} given above, a large positive value for θ_{100} will indicate that base is very likely to be tagged as a VB; conversely a highly negative value will indicate that this particular word/tag pairing is unlikely.

Prefix and Suffix features An example of a suffix feature is as follows:

$$f_{101}(h, y) = \begin{cases} 1 & \text{if } x_i \text{ ends in } \texttt{ing} \text{ and } y = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

This feature is sensitive to the suffix of the word being tagged, x_i, and the proposed tag y. In practice we would introduce a large number of features of this form. For example, in Ratnaparkhi's POS tagger, all suffixes seen in training data up to four letters in length were introduced as features (in combination with all possible tags).

An example of a *prefix* feature is as follows:

$$f_{102}(h, y) = \begin{cases} 1 & \text{if } x_i \text{ starts with } \texttt{pre} \text{ and } y = \texttt{NN} \\ 0 & \text{otherwise} \end{cases}$$

Again, a large number of prefix features would be used. Ratnaparkhi's POS tagger employs features for all prefixes up to length four seen in training data.

Prefix and suffix features are very useful for POS tagging and other tasks in English and many other languages. For example, the suffix `ing` is frequently seen with the tag `VBG` in the Penn treebank, which is the tag used for gerunds; there are many other examples.

Crucially, it is very straightforward to introduce prefix and suffix features to the model. This is in contrast with trigram HMMs for tagging, where we used the idea of mapping low-frequency words to "pseudo-words" capturing spelling features. The integration of spelling features—for example prefix and suffix features—in log-linear models is much less ad-hoc than the method we described for HMMs. Spelling features are in practice very useful when tagging words in test data that are infrequent or not seen at all in training data.

Trigram, Bigram and Unigram Tag features An example of a trigram tag feature is as follows:

$$f_{103}(h, y) = \begin{cases} 1 & \text{if } \langle y_{-2}, y_{-1}, y \rangle = \langle \text{DT, JJ, VB} \rangle \\ 0 & \text{otherwise} \end{cases}$$

This feature, in combination with the associated parameter θ_{103}, plays an analogous role to the $q(\text{VB}|\text{DT, JJ})$ parameter in a trigram HMM, allowing the model to learn whether the tag trigram $\langle \text{DT, JJ, VB} \rangle$ is likely or unlikely. Features of this form are introduced for a large number of tag trigrams, for example all tag trigrams seen in training data.

The following two features are examples of bigram and unigram tag features:

$$f_{104}(h, y) = \begin{cases} 1 & \text{if } \langle y_{-1}, y \rangle = \langle \text{JJ, VB} \rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(h, y) = \begin{cases} 1 & \text{if } \langle y \rangle = \langle \text{VB} \rangle \\ 0 & \text{otherwise} \end{cases}$$

The first feature allows the model to learn whether the tag bigram `JJ VB` is likely or unlikely; the second feature allows the model to learn whether the tag `VB` is likely or unlikely. Again, a large number of bigram and unigram tag features are typically introduced in the model.

Bigram and unigram features may at first glance seem redundant, given the obvious overlap with trigram features. For example, it might seem that features f_{104} and f_{105} are subsumed by feature f_{103}. Specifically, given parameters $\theta_{103}, \theta_{104}$ and θ_{105} we can redefine θ_{103} to be equal to $\theta_{103}+\theta_{104}+\theta_{105}$, and $\theta_{104} = \theta_{105} = 0$,

giving exactly the same distribution $p(y|x;\theta)$ for the two parameter settings.[2] Thus features f_{104} and f_{105} can apparently be eliminated. However, when used in conjunction with regularized approaches to parameter estimation, the bigram and unigram features play an important role that is analogous to the use of "backed-off" estimates $q_{\text{ML}}(\text{VB}|\text{JJ})$ and $q_{\text{ML}}(\text{VB})$ in the smoothed estimation techniques seen previously in the class. Roughly speaking, with regularization, if the trigram in feature f_{103} is infrequent, then the value for θ_{103} will not grow too large in magnitude, and the parameter values θ_{104} and θ_{105}, which are estimated based on more examples, will play a more important role.

Other Contextual Features Ratnaparkhi also used features which consider the word before or after x_i, in conjunction with the proposed tag. Example features are as follows:

$$f_{106}(h,y) = \begin{cases} 1 & \text{if previous word } x_{i-1} = \textit{the} \text{ and } y = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(h,y) = \begin{cases} 1 & \text{if next word } x_{i+1} = \textit{the} \text{ and } y = \text{VB} \\ 0 & \text{otherwise} \end{cases}$$

Again, many such features would be introduced. These features add additional context to the model, introducing dependencies between x_{i-1} or x_{i+1} and the proposed tag. Note again that it is not at all obvious how to introduce contextual features of this form to trigram HMMs.

Other Features We have described the main features in Ratnaparkhi's model. Additional features that he includes are: 1) spelling features which consider whether the word x_i being tagged contains a number, contains a hyphen, or contains an upper-case letter; 2) contextual features that consider the word at x_{i-2} and x_{i+2} in conjunction with the current tag y.

8.4 Parameter Estimation in Trigram MEMMs

Parameter estimation in trigram MEMMs can be performed using the parameter estimation methods for log-linear models described in the previous chapter. The training data is a set of m examples $(x^{(k)}, y^{(k)})$ for $k = 1 \ldots m$, where each $x^{(k)}$ is a sequence of words $x_1^{(k)} \ldots x_{n_k}^{(k)}$, and each $y^{(k)}$ is a sequence of tags $y_1^{(k)} \ldots y_{n_k}^{(k)}$.

[2] To be precise, this argument is correct in the case where for every unigram and bigram feature there is at least one trigram feature that subsumes it. The reassignment of parameter values would be applied to all trigrams.

Here n_k is the length of the k'th sentence or tag sequence. For any $k \in \{1 \dots m\}$, $i \in \{1 \dots n_k\}$, define the history $h_i^{(k)}$ to be equal to $\langle y_{i-2}^{(k)}, y_{i-1}^{(k)}, x_1^{(k)} \dots x_{n_k}^{(k)}, i \rangle$. We assume that we have a feature vector definition $f(h, y) \in \mathbb{R}^d$ for some integer d, and hence that

$$p(y_i^{(k)} | h_i^{(k)}; \theta) = \frac{\exp\left(\theta \cdot f(h_i^{(k)}, y_i)\right)}{\sum_{y \in \mathcal{K}} \exp\left(\theta \cdot f(h_i^{(k)}, y)\right)}$$

The regularized log-likelihood function is then

$$L(\theta) = \sum_{k=1}^{m} \sum_{i=1}^{n_k} \log p(y_i^{(k)} | h_i^{(k)}; \theta) - \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$$

The first term is the log-likelihood of the data under parameters θ. The second term is a regularization term, which penalizes large parameter values. The positive parameter λ dictates the relative weight of the two terms. An optimization method is used to find the parameters θ^* that maximize this function:

$$\theta^* = \arg\max_{\theta \in \mathbb{R}^d} L(\theta)$$

In summary, the estimation method is a direct application of the method described in the previous chapter, for parameter estimation in log-linear models.

8.5 Decoding with MEMMs: Another Application of the Viterbi Algorithm

We now turn to the problem of finding the most likely tag sequence for an input sequence $x_1 \dots x_n$ under a trigram MEMM; that is, the problem of finding

$$\arg \max_{y_1 \dots y_n \in \mathcal{Y}(n)} p(y_1 \dots y_n | x_1 \dots x_n)$$

where

$$p(y_1 \dots y_n | x_1 \dots x_n) = \prod_{i=1}^{n} p(y_i | h_i; \theta)$$

and $h_i = \langle y_{i-2}, y_{i-1}, x_1 \dots x_n, i \rangle$.

First, note the similarity to the decoding problem for a trigram HMM tagger, which is to find

$$\arg \max_{y_1 \dots y_n \in \mathcal{Y}(n)} q(\text{STOP} | y_{n-1}, y_n) \times \prod_{i=1}^{n} q(y_i | y_{i-2}, y_{i-1}) e(x_i | y_i)$$

Putting aside the $q(\text{STOP}|y_{n-1}, y_n)$ term, we have essentially replaced

$$\prod_{i=1}^{n} q(y_i|y_{i-2}, y_{i-1}) \times e(x_i|y_i)$$

by

$$\prod_{i=1}^{n} p(y_i|h_i; \theta)$$

The similarity of the two decoding problems leads to the decoding algorithm for trigram MEMMs being very similar to the decoding algorithm for trigram HMMs. We again use dynamic programming. The algorithm is shown in figure 8.1. The base case of the dynamic program is identical to the base case for trigram HMMs, namely

$$\pi(*, *, 0) = 1$$

The recursive case is

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times p(v|h; \theta))$$

where $h = \langle w, u, x_1 \dots x_n, k \rangle$. (We again define $\mathcal{K}_{-1} = \mathcal{K}_0 = *$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \dots n$.)

Recall that the recursive case for a trigram HMM tagger is

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$

Hence we have simply replaced $q(v|w, u) \times e(x_k|v)$ by $p(v|h; \theta)$.

The justification for the algorithm is very similar to the justification of the Viterbi algorithm for trigram HMMs. In particular, it can be shown that for all $k \in \{0 \dots n\}$, $u \in \mathcal{K}_{k-1}$, $v \in \mathcal{K}_k$,

$$\pi(k, u, v) = \max_{y_{-1} \dots y_k \in S(k,u,v)} \prod_{i=1}^{k} p(y_i|h_i; \theta)$$

where $S(k, u, v)$ is the set of all sequences $y_{-1} y_0 \dots y_k$ such that $y_i \in \mathcal{K}_i$ for $i = -1 \dots k$, and $y_{k-1} = u$, $y_k = v$.

8.6 Summary

To summarize, the main ideas behind trigram MEMMs are the following:

Input: A sentence $x_1 \dots x_n$. A set of possible tags $\mathcal{K}$. A model (for example a log-linear model) that defines a probability

$$p(y|h;\theta)$$

for any h, y pair where h is a history of the form $\langle y_{-2}, y_{-1}, x_1 \dots x_n, i\rangle$, and $y \in \mathcal{K}$.
Definitions: Define $\mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \dots n$.
Initialization: Set $\pi(0, *, *) = 1$.
Algorithm:

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

$$\pi(k, u, v) = \max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times p(v|h;\theta))$$

$$bp(k, u, v) = \arg\max_{w \in \mathcal{K}_{k-2}} (\pi(k-1, w, u) \times p(v|h;\theta))$$

    where $h = \langle w, u, x_1 \dots x_n, k\rangle$.

- Set $(y_{n-1}, y_n) = \arg\max_{u \in \mathcal{K}_{n-1}, v \in \mathcal{K}_n} \pi(n, u, v)$
- For $k = (n-2) \dots 1$,

$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \dots y_n$

Figure 8.1: The Viterbi Algorithm with backpointers.

1. We derive the model by first making the independence assumption

$$\begin{aligned} & P(Y_i = y_i | X_1 = x_1 \ldots X_n = x_n, Y_1 = y_1 \ldots Y_n = y_n) \\ = \quad & P(Y_i = y_i | X_1 = x_1 \ldots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \end{aligned}$$

and then assuming that

$$\begin{aligned} & P(Y_i = y_i | X_1 = x_1 \ldots X_n = x_n, Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \\ = \quad & p(y_i | h_i; \theta) \\ = \quad & \frac{\exp\{\theta \cdot f(h_i, y_i)\}}{\sum_y \exp\{\theta \cdot f(h_i, y)\}} \end{aligned}$$

where $h_i = \langle y_{i-2}, y_{i-1}, x_1 \ldots x_n, i \rangle$, $f(h, y) \in \mathbb{R}^d$ is a feature vector, and $\theta \in \mathbb{R}^d$ is a parameter vector.

2. The parameters θ can be estimated using standard methods for parameter estimation in log-linear models, for example by optimizing a regularized log-likelihood function.

3. The decoding problem is to find

$$\arg \max_{y_1 \ldots y_n \in \mathcal{Y}(n)} \prod_{i=1}^{n} p(y_i | h_i; \theta)$$

This problem can be solved by dynamic programming, using a variant of the Viterbi algorithm. The algorithm is closely related to the Viterbi algorithm for trigram HMMs.

4. The feature vector $f(h, y)$ can be sensitive to a wide range of information in the history h in conjunction with the tag y. This is the primary advantage of MEMMs over HMMs for tagging: it is much more straightforward and direct to introduce features into the model.