Log-Linear Models

Michael Collins

1 Introduction

This note describes *log-linear models*, which are very widely used in natural language processing. A key advantage of log-linear models is their flexibility: as we will see, they allow a very rich set of features to be used in a model, arguably much richer representations than the simple estimation techniques we have seen earlier in the course (e.g., the smoothing methods that we initially introduced for language modeling, and which were later applied to other models such as HMMs for tagging, and PCFGs for parsing). In this note we will give motivation for log-linear models, give basic definitions, and describe how parameters can be estimated in these models. In subsequent classes we will see how these models can be applied to a number of natural language processing problems.

2 Motivation

As a motivating example, consider again the language modeling problem, where the task is to derive an estimate of the conditional probability

$$P(W_i = w_i | W_1 = w_1 \dots W_{i-1} = w_{i-1}) = p(w_i | w_1 \dots w_{i-1})$$

for any sequence of words $w_1 \dots w_i$, where i can be any positive integer. Here w_i is the i'th word in a document: our task is to model the distribution over the word w_i, conditioned on the previous sequence of words $w_1 \dots w_{i-1}$.

In trigram language models, we assumed that

$$p(w_i | w_1 \dots w_{i-1}) = q(w_i | w_{i-2}, w_{i-1})$$

where $q(w|u, v)$ for any trigram (u, v, w) is a parameter of the model. We studied a variety of ways of estimating the q parameters; as one example, we studied linear interpolation, where

$$q(w|u, v) = \lambda_1 q_{ML}(w|u, v) + \lambda_2 q_{ML}(w|v) + \lambda_3 q_{ML}(w) \quad (1)$$

Here each q_{ML} is a maximum-likelihood estimate, and $\lambda_1, \lambda_2, \lambda_3$ are parameters dictating the weight assigned to each estimate (recall that we had the constraints that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i).

Trigram language models are quite effective, but they make relatively narrow use of the context $w_1 \ldots w_{i-1}$. Consider, for example, the case where the context $w_1 \ldots w_{i-1}$ is the following sequence of words:

> Third, the notion "grammatical in English" cannot be identified in any way with the notion "high order of statistical approximation to English". It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical

Assume in addition that we'd like to estimate the probability of the word *model* appearing as word w_i, i.e., we'd like to estimate

$$P(W_i = \text{model} | W_1 = w_1 \ldots W_{i-1} = w_{i-1})$$

In addition to the previous two words in the document (as used in trigram language models), we could imagine conditioning on all kinds of features of the context, which might be useful evidence in estimating the probability of seeing *model* as the next word. For example, we might consider the probability of *model* conditioned on word w_{i-2}, ignoring w_{i-1} completely:

$$P(W_i = \text{model} | W_{i-2} = \text{any})$$

We might condition on the fact that the previous word is an adjective

$$P(W_i = \text{model} | \text{pos}(W_{i-1}) = \text{adjective})$$

here pos is a function that maps a word to its part of speech. (For simplicity we assume that this is a deterministic function, i.e., the mapping from a word to its underlying part-of-speech is unambiguous.) We might condition on the fact that the previous word's suffix is "ical":

$$P(W_i = \text{model} | \text{suff4}(W_{i-1}) = \text{ical})$$

(here suff4 is a function that maps a word to its last four characters). We might condition on the fact that the word *model* does not appear in the context:

$$P(W_i = \text{model} | W_j \neq \text{model} \quad \text{for } j \in \{1 \ldots (i-1)\})$$

or we might condition on the fact that the word *grammatical* does appear in the context:

$$P(W_i = \text{model} | W_j = \text{grammatical} \text{ for some } j \in \{1 \dots (i-1)\})$$

In short, all kinds of information in the context might be useful in estimating the probability of a particular word (e.g., *model*) in that context.

A naive way to use this information would be to simply extend the methods that we saw for trigram language models. Rather than combining three estimates, based on trigram, bigram, and unigram estimates, we would combine a much larger set of estimates. We would again estimate λ parameters reflecting the importance or weight of each estimate. The resulting estimator would take something like the following form (this is intended as a sketch only):

$$\begin{aligned}
&p(\text{model} | w_1, \dots w_{i-1}) = \\
&\lambda_1 \times q_{ML}(\text{model} | w_{i-2} = \text{any}, w_{i-1} = \text{statistical}) + \\
&\lambda_2 \times q_{ML}(\text{model} | w_{i-1} = \text{statistical}) + \\
&\lambda_3 \times q_{ML}(\text{model}) + \\
&\lambda_4 \times q_{ML}(\text{model} | w_{i-2} = \text{any}) + \\
&\lambda_5 \times q_{ML}(\text{model} | w_{i-1} \text{ is an adjective}) + \\
&\lambda_6 \times q_{ML}(\text{model} | w_{i-1} \text{ ends in ``ical''}) + \\
&\lambda_7 \times q_{ML}(\text{model} | \text{``model'' does not occur somewhere in } w_1, \dots w_{i-1}) + \\
&\lambda_8 \times q_{ML}(\text{model} | \text{``grammatical'' occurs somewhere in } w_1, \dots w_{i-1}) + \\
&\dots
\end{aligned}$$

The problem is that the linear interpolation approach becomes extremely unwieldy as we add more and more pieces of conditioning information. In practice, it is very difficult to extend this approach beyond the case where we small number of estimates that fall into a natural hierarchy (e.g., unigram, bigram, trigram estimates). In contrast, we will see that log-linear models offer a much more satisfactory method for incorporating multiple pieces of contextual information.

3 A Second Example: Part-of-speech Tagging

Our second example concerns part-of-speech tagging. Consider the problem where the context is a sequence of words $w_1 \dots w_n$, together with a sequence of tags, $t_1 \dots t_{i-1}$ (here $i < n$), and our task is to model the conditional distribution over the i'th tag in the sequence. That is, we wish to model the conditional distribution

$$P(T_i = t_i | T_1 = t_1 \dots T_{i-1} = t_{i-1}, W_1 = w_1 \dots W_n = w_n)$$

As an example, we might have the following context:

> Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base from which Spain expanded its empire into the rest of the Western Hemisphere .

Here $w_1 \ldots w_n$ is the sentence *Hispaniola quickly* ... *Hemisphere* ., and the previous sequence of tags is $t_1 \ldots t_5$ = `NNP RB VB DT JJ`. We have $i = 6$, and our task is to model the distribution

$$P(T_6 = t_6 \quad | \quad W_1 \ldots W_n = \textit{Hispaniola quickly} \ldots \textit{Hemisphere} ., \\ T_1 \ldots T_5 = \texttt{NNP RB VB DT JJ})$$

i.e., our task is to model the distribution over tags for the 6th word, *base*, in the sentence.

In this case there are again many pieces of contextual information that might be useful in estimating the distribution over values for t_i. To be concrete, consider estimating the probability that the tag for *base* is `V` (i.e., $T_6 =$ `V`). We might consider the probability conditioned on the identity of the i'th word:

$$P(T_6 = \texttt{V} | W_6 = \textit{base})$$

and we might also consider the probability conditioned on the previous one or two tags:

$$P(T_6 = \texttt{V} | T_5 = \texttt{JJ})$$

$$P(T_6 = \texttt{V} | T_4 = \texttt{DT}, T_5 = \texttt{JJ})$$

We might consider the probability conditioned on the *previous* word in the sentence

$$P(T_6 = \texttt{V} | W_5 = \textit{important})$$

or the probability conditioned on the *next* word in the sentence

$$P(T_6 = \texttt{V} | W_7 = \textit{from})$$

We might also consider the probability based on spelling features of the word w_6, for example the last two letters of w_6:

$$P(T_6 = \texttt{V} | \text{suff2}(W_6) = \textit{se})$$

(here suff2 is a function that maps a word to its last two letters).

In short, we again have a scenario where a whole variety of contextual features might be useful in modeling the distribution over the random variable of interest (in this case the identity of the i'th tag). Again, a naive approach based on an extension of linear interpolation would unfortunately fail badly when faced with this estimation problem.

4 Log-Linear Models

We now describe how log-linear models can be applied to problems of the above form.

4.1 Basic Definitions

The abstract problem is as follows. We have some set of possible *inputs*, $\mathcal{X}$, and a set of possible *labels*, $\mathcal{Y}$. Our task is to model the conditional probability

$$p(y|x)$$

for any pair (x, y) such that $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

For example, in the language modeling task we have some finite set of possible words in the language, call this set $\mathcal{V}$. The set $\mathcal{Y}$ is simply equal to $\mathcal{V}$. The set $\mathcal{X}$ is the set of possible sequences $w_1 \dots w_{i-1}$ such that $i \geq 1$, and $w_j \in \mathcal{V}$ for $j \in \{1 \dots (i-1)\}$.

In the part-of-speech tagging example, we have some set $\mathcal{V}$ of possible words, and a set $\mathcal{T}$ of possible tags. The set $\mathcal{Y}$ is simply equal to $\mathcal{T}$. The set $\mathcal{X}$ is the set of contexts of the form

$$\langle w_1 w_2 \dots w_n, t_1 t_2 \dots t_{i-1} \rangle$$

where $n \geq 1$ is an integer specifying the length of the input sentence, $w_j \in \mathcal{V}$ for $j \in \{1 \dots n\}$, $i \in \{1 \dots (n-1)\}$, and $t_j \in \mathcal{T}$ for $j \in \{1 \dots (i-1)\}$.

We will assume throughout that $\mathcal{Y}$ is a finite set. The set $\mathcal{X}$ could be finite, countably infinite, or even uncountably infinite.

Log-linear models are then defined as follows:

Definition 1 (Log-linear Models) *A log-linear model consists of the following components:*

- *A set $\mathcal{X}$ of possible inputs.*
- *A set $\mathcal{Y}$ of possible labels. The set $\mathcal{Y}$ is assumed to be finite.*
- *A positive integer d specifying the number of features and parameters in the model.*
- *A function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$ that maps any (x, y) pair to a* feature-vector *$f(x, y)$.*
- *A parameter vector $v \in \mathbb{R}^d$.*

For any $x \in \mathcal{X}$, $y \in \mathcal{Y}$, the model defines a condtional probability

$$p(y|x; v) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(v \cdot f(x, y'))}$$

Here $\exp(x) = e^x$, and $v \cdot f(x, y) = \sum_{k=1}^{d} v_k f_k(x, y)$ is the inner product between v and $f(x, y)$. The term $p(y|x; v)$ is intended to be read as "the probability of y conditioned on x, under parameter values v". □

We now describe the components of the model in more detail, first focusing on the feature-vector definitions $f(x, y)$, then giving intuition behind the model form

$$p(y|x; v) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(v \cdot f(x, y'))}$$

5 Features

As described in the previous section, for any pair (x, y), $f(x, y) \in \mathbb{R}^d$ is a feature vector representing that pair. Each component $f_k(x, y)$ for $k = 1 \ldots d$ in this vector is referred to as a *feature*. The features allows us to represent different properties of the input x, in conjunction with the label y. Each feature has an associated parameter, v_k, whose value is estimated using a set of training examples. The training set consists of a sequence of examples $(x^{(i)}, y^{(i)})$ for $i = 1 \ldots n$, where each $x^{(i)} \in \mathcal{X}$, and each $y^{(i)} \in \mathcal{Y}$.

In this section we first give an example of how features can be constructed for the language modeling problem, as introduced earlier in this note; we then describe some practical issues in defining features.

5.1 Features for the Language Modeling Example

Consider again the language modeling problem, where the input x is a sequence of words $w_1 w_2 \ldots w_{i-1}$, and the label y is a word. Figure 1 shows a set of example features for this problem. Each feature is an indicator function: that is, each feature is a function that returns either 1 or 0. It is extremely common in NLP applications to have indicator functions as features. Each feature returns the value of 1 if some property of the input x conjoined with the label y is true, and 0 otherwise.

The first three features, f_1, f_2, and f_3, are analogous to unigram, bigram, and trigram features in a regular trigram language model. The first feature returns 1 if the label y is equal to the word *model*, and 0 otherwise. The second feature returns 1 if the bigram $\langle w_{i-1}\ y \rangle$ is equal to $\langle$*statistical model*$\rangle$, and 0 otherwise. The third feature returns 1 if the trigram $\langle w_{i-2}\ w_{i-1}\ y \rangle$ is equal to $\langle$*any statistical model*$\rangle$,

$$
\begin{aligned}
f_1(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model} \\ 0 & \text{otherwise} \end{cases} \\
f_2(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and } w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases} \\
f_3(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, w_{i-2} = \texttt{any}, w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases} \\
f_4(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, w_{i-2} = \texttt{any} \\ 0 & \text{otherwise} \end{cases} \\
f_5(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, w_{i-1} \text{ is an adjective} \\ 0 & \text{otherwise} \end{cases} \\
f_6(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, w_{i-1} \text{ ends in ``ical''} \\ 0 & \text{otherwise} \end{cases} \\
f_7(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, \text{``model'' is not in } w_1, \ldots w_{i-1} \\ 0 & \text{otherwise} \end{cases} \\
f_8(x,y) &= \begin{cases} 1 & \text{if } y = \texttt{model}, \text{``grammatical'' is in } w_1, \ldots w_{i-1} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 1: Example features for the language modeling problem, where the input x is a sequence of words $w_1 w_2 \ldots w_{i-1}$, and the label y is a word.

and 0 otherwise. Recall that each of these features will have a parameter, v_1, v_2, or v_3; these parameters will play a similar role to the parameters in a regular trigram language model.

The features $f_4 \ldots f_8$ in figure 1 consider properties that go beyond unigram, bigram, and trigram features. The feature f_4 considers word w_{i-2} in conjunction with the label y, ignoring the word w_{i-1}; this type of feature is often referred to as a "skip bigram". Feature f_5 considers the part-of-speech of the previous word (assume again that the part-of-speech for the previous word is available, for example through a deterministic mapping from words to their part-of-speech, or perhaps through a POS tagger's output on words $w_1 \ldots w_{i-1}$). Feature f_6 considers the suffix of the previous word, and features f_7 and f_8 consider various other features of the input $x = w_1 \ldots w_{i-1}$.

From this example we can see that it is possible to incorporate a broad set of contextual information into the language modeling problem, using features which are indicator functions.

5.2 Feature Templates

We now discuss some practical issues in defining features. In practice, a key idea in defining features is that of *feature templates*. We introduce this idea in this section.

Recall that our first three features in the previous example were as follows:

$$f_1(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and } w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases}$$

$$f_3(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model}, w_{i-2} = \texttt{any}, w_{i-1} = \texttt{statistical} \\ 0 & \text{otherwise} \end{cases}$$

These features track the unigram $\langle$*model*$\rangle$, the bigram $\langle$*statistical model*$\rangle$, and the trigram $\langle$*any statistical model*$\rangle$.

Each of these features is specific to a particular unigram, bigram or trigram. In practice, we would like to define a much larger class of features, which consider all possible unigrams, bigrams or trigrams seen in the training data. To do this, we use *feature templates* to generate large sets of features.

As one example, here is a feature template for trigrams:

Definition 2 (Trigram feature template) *For any trigram* (u, v, w) *seen in train-*

ing data, create a feature

$$f_{N(u,v,w)}(x,y) = \begin{cases} 1 & \textit{if } y = w,\ w_{i-2} = u,\ w_{i-1} = v \\ 0 & \textit{otherwise} \end{cases}$$

where $N(u,v,w)$ is a function that maps each trigram in the training data to a unique integer.

A couple of notes on this definition:

- Note that the template only generates trigram features for those trigrams seen in training data. There are two reasons for this restriction. First, it is not feasible to generate a feature for every possible trigram, even those not seen in training data: this would lead to V^3 features, where V is the number of words in the vocabulary, which is a very large set of features. Second, for any trigram (u,v,w) not seen in training data, we do not have evidence to estimate the associated parameter value, so there is no point including it in any case.[1]

- The function $N(u,v,w)$ maps each trigram to a unique integer: that is, it is a function such that for any trigrams (u,v,w) and (u',v',w') such that $u \neq u'$, $v \neq v'$, or $w \neq w'$, we have

  $$N(u,v,w) \neq N(u',v',w')$$

  In practice, in implementations of feature templates, the function N is implemented through a hash function. For example, we could use a hash table to hash strings such as `trigram=any_statistical_model` to integers. Each distinct string is hashed to a different integer.

Continuing with the example, we can also define bigram and unigram feature templates:

Definition 3 (Bigram feature template) *For any bigram (v,w) seen in training data, create a feature*

$$f_{N(v,w)}(x,y) = \begin{cases} 1 & \textit{if } y = w,\ w_{i-1} = v \\ 0 & \textit{otherwise} \end{cases}$$

where $N(v,w)$ maps each bigram to a unique integer.

[1] This isn't quite accurate: there may in fact be reasons for including features for trigrams (u,v,w) where the bigram (u,v) *is* observed in the training data, but the trigram (u,v,w) is *not* observed in the training data. We defer discussion of this until later.

Definition 4 (Unigram feature template) *For any unigram (w) seen in training data, create a feature*

$$f_{N(w)}(x, y) = \begin{cases} 1 & \text{if } y = w \\ 0 & \text{otherwise} \end{cases}$$

where $N(w)$ maps each unigram to a unique integer.

We actually need to be slightly more careful with these definitions, to avoid overlap between trigram, bigram, and unigram features. Define T, B and U to be the set of trigrams, bigrams, and unigrams seen in the training data. Define

$$N_t = \{i : \exists (u, v, w) \in T \text{ such that } N(u, v, w) = i\}$$

$$N_b = \{i : \exists (v, w) \in B \text{ such that } N(v, w) = i\}$$

$$N_u = \{i : \exists (w) \in U \text{ such that } N(w) = i\}$$

Then we need to make sure that there is no overlap between these sets—otherwise, two different n-grams would be mapped to the same feature. More formally, we need

$$N_t \cap N_b = N_t \cap N_u = N_b \cap N_u = \emptyset \tag{2}$$

In practice, it is easy to ensure this when implementing log-linear models, using a single hash table to hash strings such as `trigram=any_statistical_model`, `bigram=statistical_model`, `unigram=model`, to distinct integers.

We could of course define additional templates. For example, the following is a template which tracks the length-4 suffix of the previous word, in conjunction with the label y:

Definition 5 (Length-4 Suffix Template) *For any pair (v, w) seen in training data, where $v = \textit{suff4}(w_{i-1})$, and $w = y$, create a feature*

$$f_{N(\textit{suff4}=v,w)}(x, y) = \begin{cases} 1 & \text{if } y = w \text{ and } \textit{suff4}(x) = v \\ 0 & \text{otherwise} \end{cases}$$

where $N(\textit{suff4} = v, w)$ maps each pair (v, w) to a unique integer, with no overlap with the other feature templates used in the model (where overlap is defined analogously to Eq. 2 above).

5.3 Feature Sparsity

A very important property of the features we have defined above is feature sparsity. The number of features, d, in many NLP applications can be extremely large. For example, with just the trigram template defined above, we would have one feature for each trigram seen in training data. It is not untypical to see models with 100s of thousands or even millions of features.

This raises obvious concerns with efficiency of the resulting models. However, we describe in this section how feature sparsity can lead to efficient models.

The key observation is the following: for any given pair (x, y), the number of values for k in $\{1 \ldots d\}$ such that

$$f_k(x, y) = 1$$

is often very small, and is typically much smaller than the total number of features, d. Thus all but a very small subset of the features are 0: the feature vector $f(x, y)$ is a very sparse bit-string, where almost all features $f_k(x, y)$ are equal to 0, and only a few features are equal to 1.

As one example, consider the language modeling example where we use only the trigram, bigram and unigram templates, as described above. The number of features in this model is large (it is equal to the number of distinct trigrams, bigrams and unigrams seen in training data). However, it can be seen immediately that for any pair (x, y), at most three features are non-zero (in the worst case, the pair (x, y) contains trigram, bigram and unigram features which are all seen in the training data, giving three non-zero features in total).

When implementing log-linear models, models with sparse features can be quite efficient, because there is no need to explicitly represent and manipulate d-dimensional feature vectors $f(x, y)$. Instead, it is generally much more efficient to implement a function (typically through hash tables) that for any pair (x, y) computes the indices of the non-zero features: i.e., a function that computes the set

$$Z(x, y) = \{k : f_k(x, y) = 1\}$$

This set is small in sparse feature spaces—for example with unigram/bigram/trigram features alone, it would be of size at most 3. In general, it is straightforward to implement a function that computes $Z(x, y)$ in $O(|Z(x, y)|)$ time, using hash functions. Note that $|Z(x, y)| \ll d$, so this is much more efficient than explicitly computing all d features, which would take $O(d)$ time.

As one example of how efficient computation of $Z(x, y)$ can be very helpful, consider computation of the inner product

$$v \cdot f(x, y) = \sum_{k=1}^{d} v_k f_k(x, y)$$

This computation is central in log-linear models. A naive method would iterate over each of the d features in turn, and would take $O(d)$ time. In contrast, if we make use of the identity

$$\sum_{k=1}^{d} v_k f_k(x, y) = \sum_{k \in Z(x,y)} v_k$$

hence looking at only non-zero features, we can compute the inner product in $O(|Z(x, y)|)$ time.

6 The Model form for Log-Linear Models

We now describe the model form for log-linear models in more detail. Recall that for any pair (x, y) such that $x \in \mathcal{X}$, and $y \in \mathcal{Y}$, the conditional probability under the model is

$$p(y \mid x; v) = \frac{\exp(v \cdot f(x, y))}{\sum_{y' \in \mathcal{Y}} \exp(v \cdot f(x, y'))}$$

The inner products

$$v \cdot f(x, y)$$

play a key role in this expression. Again, for illustration consider our languge-modeling example where the input $x = w_1 \ldots w_{i-1}$ is the following sequence of words:

> Third, the notion "grammatical in English" cannot be identified in any way with the notion "high order of statistical approximation to English". It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical

The first step in calculating the probability distribution over the next word in the document, conditioned on x, is to calculate the inner product $v \cdot f(x, y)$ for each possible label y (i.e., for each possible word in the vocabulary). We might, for example, find the following values (we show the values for just a few possible words—in reality we would compute an inner product for each possible word):

$$\begin{aligned} v \cdot f(x, model) &= 5.6 & v \cdot f(x, the) &= -3.2 \\ v \cdot f(x, is) &= 1.5 & v \cdot f(x, of) &= 1.3 \\ v \cdot f(x, models) &= 4.5 & \ldots & \end{aligned}$$

Note that the inner products can take any value in the reals, positive or negative. Intuitively, if the inner product $v \cdot f(x, y)$ for a given word y is high, this indicates that the word has high probability given the context x. Conversely, if $v \cdot f(x, y)$ is low, it indicates that y has low probability in this context.

The inner products $v \cdot f(x, y)$ can take any value in the reals; our goal, however, is to define a conditional distribution $p(y|x)$. If we take

$$\exp\left(v \cdot f(x, y)\right)$$

for any label y, we now have a value that is greater than 0. If $v \cdot f(x, y)$ is high, this value will be high; if $v \cdot f(x, y)$ is low, for example if it is strongly negative, this value will be low (close to zero).

Next, if we divide the above quantity by

$$\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x, y')\right)$$

giving

$$p(y|x; v) = \frac{\exp\left(v \cdot f(x, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x, y')\right)} \tag{3}$$

then it is easy to verify that we have a well-formed distribution: that is,

$$\sum_{y \in \mathcal{Y}} p(y|x; v) = 1$$

Thus the denominator in Eq. 3 is a normalization term, which ensures that we have a distribution that sums to one. In summary, the function

$$\frac{\exp\left(v \cdot f(x, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x, y')\right)}$$

performs a transformation which takes as input a set of values $\{v \cdot f(x, y) : y \in \mathcal{Y}\}$, where each $v \cdot f(x, y)$ can take any value in the reals, and as output produces a probability distribution over the labels $y \in \mathcal{Y}$.

Finally, we consider where the name log-linear models originates from. It follows from the above definitions that

$$\begin{aligned} \log p(y|x; v) &= v \cdot f(x, y) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x, y')\right) \\ &= v \cdot f(x, y) - g(x) \end{aligned}$$

where

$$g(x) = \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x, y')\right)$$

The first term, $v \cdot f(x, y)$, is linear in the features $f(x, y)$. The second term, $g(x)$, depends only on x, and does not depend on the label y. Hence the log probability $\log p(y|x; v)$ is a linear function in the features $f(x, y)$, as long as we hold x fixed; this justifies the term "log-linear".

7 Parameter Estimation in Log-Linear Models

7.1 The Log-Likelihood Function, and Regularization

We now consider the problem of parameter estimation in log-linear models. We assume that we have a training set, consisting of examples $(x^{(i)}, y^{(i)})$ for $i \in \{1 \ldots n\}$, where each $x^{(i)} \in \mathcal{X}$, and each $y^{(i)} \in \mathcal{Y}$.

Given parameter values v, for any example i, we can calculate the log conditional probability

$$\log p(y^{(i)}|x^{(i)}; v)$$

under the model. Intuitively, the higher this value, the better the model fits this particular example. The log-likelihood considers the sum of log probabilities of examples in the training data:

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) \tag{4}$$

This is a function of the parameters v. For any parameter vector v, the value of $L(v)$ can be interpreted of a measure of how well the parameter vector fits the training examples.

The first estimation method we will consider is maximum-likelihood estimation, where we choose our parameters as

$$v_{ML} = \arg\max_{v \in \mathbb{R}^d} L(v)$$

In the next section we describe how the parameters v_{ML} can be found efficiently. Intuitively, this estimation method finds the parameters which fit the data as well as possible.

The maximum-likelihood estimates can run into problems, in particular in cases where the number of features in the model is very large. To illustrate, consider the language-modeling problem again, and assume that we have trigram, bigram and unigram features. Now assume that we have some trigram (u, v, w) which is seen only once in the training data; to be concrete, assume that the trigram is *any statistical model*, and assume that this trigram is seen on the 100'th

training example alone. More precisely, we assume that

$$f_{N(any,statistical,model)}(x^{(100)}, y^{(100)}) = 1$$

In addition, assume that this is the only trigram (u, v, w) in training data with $u =$ *any*, and $v =$ *statistical*. In this case, it can be shown that the maximum-likelihood parameter estimate for v_{100} is $+\infty$,[2], which gives

$$p(y^{(100)}|x^{(100)}; v) = 1$$

In fact, we have a very similar situation to the case in maximum-likelihood estimates for regular trigram models, where we would have

$$q_{ML}(\textit{model}|\textit{any, statistical}) = 1$$

for this trigram. As discussed earlier in the class, this model is clearly under-smoothed, and it will generalize badly to new test examples. It is unreasonable to assign

$$P(W_i = \textit{model}|W_{i-1}, W_{i-2} = \textit{any, statistical}) = 1$$

based on the evidence that the bigram *any statistical* is seen once, and on that one instance the bigram is followed by the word *model*.

A very common solution for log-linear models is to modify the objective function in Eq. 4 to include a *regularization term*, which prevents parameter values from becoming too large (and in particular, prevents parameter values from diverging to infinity). A common regularization term is the 2-norm of the parameter values, that is,

$$||v||^2 = \sum_k v_k^2$$

(here $||v||$ is simply the length, or Euclidean norm, of a vector v; i.e., $||v|| = \sqrt{\sum_k v_k^2}$). The modified objective function is

$$L'(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) - \frac{\lambda}{2} \sum_k v_k^2 \tag{5}$$

[2]It is relatively easy to prove that v_{100} can diverge to ∞. To give a sketch: under the above assumptions, the feature $f_{N(any,statistical,model)}(x, y)$ is equal to 1 on only a single pair $x^{(i)}, y$ where $i \in \{1 \ldots n\}$, and $y \in \mathcal{Y}$, namely the pair $(x^{(100)}, y^{(100)})$. Because of this, as $v_{100} \to \infty$, we will have $p(y^{(100)}|x^{(100)}; v)$ tending closer and closer to a value of 1, with all other values $p(y^{(i)}|x^{(i)}; v)$ remaining unchanged. Thus we can use this one parameter to maximize the value for $\log p(y^{(100)}|x^{(100)}; v)$, independently of the probability of all other examples in the training set.

where $\lambda > 0$ is a parameter, which is typically chosen by validation on some held-out dataset. We again choose the parameter values to maximize the objective function: that is, our optimal parameter values are

$$v^* = \arg\max_v L'(v)$$

The key idea behind the modified objective in Eq. 5 is that we now balance two separate terms. The first term is the log-likelihood on the training data, and can be interpreted as a measure of how well the parameters v fit the training examples. The second term is a penalty on large parameter values: it encourages parameter values to be as close to zero as possible. The parameter λ defines the relative weighting of the two terms. In practice, the final parameters v^* will be a compromise between fitting the data as well as is possible, and keeping their values as small as possible.

In practice, this use of regularization is very effective in smoothing of log-linear models.

7.2 Finding the Optimal Parameters

First, consider finding the maximum-likelihood parameter estimates: that is, the problem of finding

$$v_{ML} = \arg\max_{v \in \mathbb{R}^d} L(v)$$

where

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v)$$

The bad news is that in the general case, there is no closed-form solution for the maximum-likelihood parameters v_{ML}. The good news is that finding $\arg\max_v L(v)$ is a relatively easy problem, because $L(v)$ can be shown to be a *convex* function. This means that simple gradient-ascent-style methods will find the optimal parameters v_{ML} relatively quickly.

Figure 2 gives a sketch of a gradient-based algorithm for optimization of $L(v)$. The parameter vector is initialized to the vector of all zeros. At each iteration we first calculate the gradients δ_k for $k = 1 \ldots d$. We then move in the direction of the gradient: more precisely, we set $v \leftarrow v + \beta^* \times \delta$ where β^* is chosen to give the optimal improvement in the objective function. This is a "hill-climbing" technique where at each point we compute the steepest direction to move in (i.e., the direction of the gradient); we then move the distance in that direction which gives the greatest value for $L(v)$.

Simple gradient ascent, as shown in figure 2, can be rather slow to converge. Fortunately there are many standard packages for gradient-based optimization,

Initialization: $v = 0$

Iterate until convergence:

- Calculate $\delta_k = \frac{dL(v)}{dv_k}$ for $k = 1 \ldots d$
- Calculate $\beta^* = \arg\max_{\beta \in \mathbb{R}} L(v + \beta\delta)$ where δ is the vector with components δ_k for $k = 1 \ldots d$ (this step is performed using some type of line search)
- Set $v \leftarrow v + \beta^* \delta$

Figure 2: A gradient ascent algorithm for optimization of $L(v)$.

which use more sophisticated algorithms, and which give considerably faster convergence. As one example, a commonly used method for parameter estimation in log-linear models is LBFGs. LBFGs is again a gradient method, but it makes a more intelligent choice of search direction at each step. It does however rely on the computation of $L(v)$ and $\frac{dL(v)}{dv_k}$ for $k = 1$ at each step—in fact this is the only information it requires about the function being optimized. In summary, if we can compute $L(v)$ and $\frac{dL(v)}{dv_k}$ efficiently, then it is simple to use an existing gradient-based optimization package (e.g., based on LBFGs) to find the maximum-likelihood estimates.

Optimization of the regularized objective function,

$$L'(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) - \frac{\lambda}{2} \sum_k v_k^2$$

can be performed in a very similar manner, using gradient-based methods. $L'(v)$ is also a convex function, so a gradient-based method will find the global optimum of the parameter estimates.

The one remaining step is to describe how the gradients

$$\frac{dL(v)}{dv_k}$$

and

$$\frac{dL'(v)}{dv_k}$$

can be calculated. This is the topic of the next section.

7.3 Gradients

We first consider the derivatives

$$\frac{dL(v)}{dv_k}$$

where

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v)$$

It is relatively easy to show (see the appendix of this note), that for any $k \in \{1 \dots d\}$,

$$\frac{dL(v)}{dv_k} = \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} p(y|x^{(i)}; v) f_k(x^{(i)}, y) \tag{6}$$

where as before

$$p(y|x^{(i)}; v) = \frac{\exp\left(v \cdot f(x^{(i)}, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)}$$

The expression in Eq. 6 has a quite intuitive form. The first part of the expression,

$$\sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)})$$

is simply the number of times that the feature f_k is equal to 1 on the training examples (assuming that f_k is an indicator function; i.e., assuming that $f_k(x^{(i)}, y^{(i)})$ is either 1 or 0). The second part of the expression,

$$\sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} p(y|x^{(i)}; v) f_k(x^{(i)}, y)$$

can be interpreted as the expected number of times the feature is equal to 1, where the expectation is taken with respect to the distribution

$$p(y|x^{(i)}; v) = \frac{\exp\left(v \cdot f(x^{(i)}, y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)}$$

specified by the current parameters. The gradient is then the difference of these terms. It can be seen that the gradient is easily calculated.

The gradients

$$\frac{dL'(v)}{dv_k}$$

where

$$L'(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v) - \frac{\lambda}{2} \sum_k v_k^2$$

are derived in a very similar way. We have

$$\frac{d}{dv_k} \left(\sum_k v_k^2 \right) = 2v_k$$

hence

$$\frac{dL'(v)}{dv_k} = \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} p(y|x^{(i)}; v) f_k(x^{(i)}, y) - \lambda v_k \qquad (7)$$

Thus the only difference from the gradient in Eq. 6 is the additional term $-\lambda v_k$ in this expression.

A Calculation of the Derivatives

In this appendix we show how to derive the expression for the derivatives, as given in Eq. 6. Our goal is to find an expression for

$$\frac{dL(v)}{dv_k}$$

where

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)}|x^{(i)}; v)$$

First, consider a single term $\log p(y^{(i)}|x^{(i)}; v)$. Because

$$p(y^{(i)}|x^{(i)}; v) = \frac{\exp\left(v \cdot f(x^{(i)}, y^{(i)})\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)}$$

we have

$$\log p(y^{(i)}|x^{(i)}; v) = v \cdot f(x^{(i)}, y^{(i)}) - \log \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)$$

The derivative of the first term in this expression is simple:

$$\frac{d}{dv_k} \left(v \cdot f(x^{(i)}, y^{(i)})\right) = \frac{d}{dv_k} \left(\sum_k v_k f_k(x^{(i)}, y^{(i)}) \right) = f_k(x^{(i)}, y^{(i)}) \qquad (8)$$

Now consider the second term. This takes the form

$$\log g(v)$$

where

$$g(v) = \sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)$$

By the usual rules of differentiation,

$$\frac{d}{dv_k} \log g(v) = \frac{\frac{d}{dv_k}(g(v))}{g(v)}$$

In addition, it can be verified that

$$\frac{d}{dv_k} g(v) = \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \exp\left(v \cdot f(x^{(i)}, y')\right)$$

hence

$$\begin{aligned}
\frac{d}{dv_k} \log g(v) &= \frac{\frac{d}{dv_k}(g(v))}{g(v)} && (9)\\
&= \frac{\sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \exp\left(v \cdot f(x^{(i)}, y')\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)} && (10)\\
&= \sum_{y' \in \mathcal{Y}} \left(f_k(x^{(i)}, y') \times \frac{\exp\left(v \cdot f(x^{(i)}, y')\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x^{(i)}, y')\right)} \right) && (11)\\
&= \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y'|x; v) && (12)
\end{aligned}$$

Combining Eqs 8 and 12 gives

$$\frac{dL(v)}{dv_k} = \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y \in \mathcal{Y}} p(y|x^{(i)}; v) f_k(x^{(i)}, y)$$