Log-Linear Models

Jason Eisner (Johns Hopkins University)

This handout is intended to be used with the interactive visualization at http://cs.jhu.edu/~jason/tutorials/loglin/.

1 The Problem *[read with lesson 1]*

Often we make use of conditional probabilities like $p(y \mid x)$, where y and x are discrete outcomes:

- They can be used to predict y directly.
- We can build a more complicated model as a product of such probabilities.[1]

But where do we get these probabilities?

1.1 Notation

Suppose X denotes a random context and Y is a random event that happens in that context. Let $\hat{p}(Y = y \mid X = x)$ be our estimate of the conditional probability that the event will turn out to be y when the context happens to be x. We'll abbreviate this as $\hat{p}(y \mid x)$. The little "hat" over the p is a reminder that this is just an estimate, not the true probability.

1.2 Simple methods

A simple idea is to estimate each conditional probability separately, using simple count ratios:

$$\hat{p}(y \mid x) \stackrel{\text{def}}{=} \frac{\text{count}(x, y)}{\text{count}(x)} \tag{1}$$

Such a "maximum-likelihood" estimate is *unbiased* (correct on average, when estimated from a *random* dataset). But unfortunately, it may have *high variance* (it is sensitive to the *particular* dataset from which you obtain the counts). It will be unreliable if either count is small:

- Suppose the denominator $\text{count}(x)$ is 2, then the estimate will be either 0, 0.5, or 1. These estimates are quite different, and which one you get depends on your particular dataset.
- It's even worse if the denominator is 1 or 0. What happens in each case?
- If the numerator $\text{count}(x, y)$ is 0, meaning that y was not *observed* in context x, then the estimate will be 0, indicating that y is *impossible* in context x.
- If the numerator is 1, then the estimate might be too high, since perhaps y is just one of many unlikely events—it's the one we happened to see.

"Smoothing" techniques try to reduce the variance of this estimate. However, they still estimate each conditional probability more or less independently of the others.[2]

[1] E.g., Bayes' theorem, Bayesian networks, (hidden) Markov models, probabilistic context-free grammars, ...

[2] Though "backoff smoothing" does jointly estimate $p(y \mid x_1)$ and $p(y \mid x_2)$ if x_1, x_2 are related in a certain way.

2 Log-linear models *[read with lesson 1]*

Log-linear modeling is a very popular and flexible technique for addressing this problem. It has the advantage that it considers *descriptions* of the events. Contextualized events (x, y) with *similar* descriptions tend to have similar probabilities—a form of generalization.

2.1 Feature functions

Since you are the one who knows something about your problem, *you* get to define the function $\vec{f}$ that extracts these descriptions. If you decide that your model will have K features, then $\vec{f}(x, y)$ denotes a vector of K real numbers $(f_1(x, y), f_2(x, y), \ldots, f_K(x, y))$ that *describe* event y in context x. These K numbers are called the **values** or **strengths** of the features.

You might use the following set of 5 features ($K = 5$) to describe colored shapes against backgrounds. The third column gives the feature values for a striped circle y that contrasts strongly with its background x:

feature number	feature meaning	feature value on this example
1	y is circular	1
2	y is square	0
3	y is striped	1
4	y is a striped square	0
5	y contrasts with background x	2

Thus, your description of this particular (x, y) is given by $\vec{f}(x, y) = (1, 0, 1, 0, 2) \in \mathbb{R}^5$. A common figure of speech is that features 1, 3, and 5 **fire** on this contextualized event (x, y), and that feature 5 fires twice (or fires with strength 2).

It is slightly unusual that $f_5(x, y) = 2.0$. In principle a feature value $f_k(x, y)$ can be any positive or negative real number. But most often your feature functions f_k will be boolean functions, so that $f_k(x, y)$ is always either 1 or 0, according to whether (x, y) has feature k or not.

2.2 The defining formula

The log-linear probability model $\hat{p}$ computes a contextualized event's probability from its features.[3] For example, given a light background x, how likely is the shape y to be a dark striped circle?

Specifically, the log-linear model defines

$$\hat{p}(y \mid x) \stackrel{\text{def}}{=} \frac{u(x, y)}{Z(x)} \tag{2}$$

where the interesting part is the **unnormalized probability** defined by

$$u(x, y) \stackrel{\text{def}}{=} \exp \sum_{k=1}^{K} (\theta_k \cdot f_k(x, y)) \tag{3}$$

$$= \exp\left(\vec{\theta} \cdot \vec{f}(x, y)\right) > 0 \tag{4}$$

[3] We are still using the $\hat{p}$ notation because this is only a model that we are trying to fit to the data. In most cases, we will never know whether the true probability distribution p has this form.

Here $\vec{\theta} \in \mathbb{R}^K$ is a vector of feature **weights**, which are the adjustable parameters of the model. You will control these with sliders in the interactive visualization.

The unnormalized probabilities $u(x, y)$ are always positive (because of the exp). Equation (2) has to scale them, in order to get proper probabilities that sum to 1 (i.e., $\sum_y \hat{p}(y \mid x) = 1$ for each x). This is a simple matter of dividing them by a **normalizing constant**, namely

$$Z(x) \stackrel{\text{def}}{=} \sum_y u(x, y) \tag{5}$$

2.3 Feature weights

You can easily see that for any choice of $\vec{\theta}$ and any x, equation (2) gives a genuine probability distribution over the events y that could occur in context x. You can also see now why the model is "log-linear"—for each context x, the log-probability is a linear function of the feature vector:

$$\log \hat{p}(y \mid x) = \vec{\theta} \cdot \vec{f}(x, y) - \log Z(x) \tag{6}$$

In our earlier example, setting the parameter $\theta_5 > 0$ makes contrasting colors more common: the resulting $\hat{p}(y \mid x)$ says that dark shapes y are probable when the background x is light, and vice-versa. Alternatively, setting $\theta_5 < 0$ makes contrasting colors less common.

In general, the linear coefficients $\vec{\theta}$ in (6) are incredibly important: the way you pick them will determine all the probabilities in the model, and even the normalizing constants! Fortunately, to help you set $\vec{\theta}$, you'll have a training set of events in their contexts, $((x_1, y_1), (x_2, y_2), \ldots (x_N, y_N))$. Then the $\vec{\theta}$ that maximizes

$$\prod_{i=1}^{N} \hat{p}(y_i \mid x_i) \tag{7}$$

does the best possible job of predicting these events in their contexts. More on this later.

2.4 Unconditioned Models

The initial lessons in the interactive visualization deal with the simple case where there is no information about context. So we can omit x. We now have a simple distribution over possible y values,

$$\hat{p}(y) \stackrel{\text{def}}{=} \frac{u(y)}{Z} \tag{8}$$

$$u(y) \stackrel{\text{def}}{=} \exp\left(\vec{\theta} \cdot \vec{f}(y)\right) > 0 \tag{9}$$

$$Z \stackrel{\text{def}}{=} \sum_y u(y) \tag{10}$$

2.5 Remark: Named Features and Efficient Dot Products

In the notation above, we assigned numbers $1, 2, \ldots, K$ to the features. This lets you represent the parameter vector $\vec{\theta}$ (as well as each feature vector $\vec{f}(x, y)$) by an *array* of K real numbers.

However, it is often more convenient to refer to the features by names instead of numbers. The names can be strings or other objects that describe the feature. For example, in the table from

section 2.1, the first feature function might be called f_{circle} instead of f_1, with corresponding weight θ_{circle} instead of θ_1. Then the weight vector $\vec{\theta}$ can be implemented as a hash table instead of an array: it still maps the index k to the weight θ_k, but now k is a name instead of a number.

The dot product $\vec{\theta} \cdot \vec{f}(x, y)$ is now defined by $\sum_k \theta_k \cdot f_k(x, y)$ where k ranges over the set of feature names. In computing this sum, it is fine to skip features k where $f_k(x, y) = 0$. In many cases, this is a big speedup because only a few of the K features will fire on a given contextualized event (x, y). For example, $f_k(x, y)$ will be 1 for only one of the shapes $k \in \{\mathsf{circle}, \mathsf{square}, \ldots\}$, and 0 for all other shapes. So the sum includes just one of the weights $\theta_{\mathsf{circle}}, \theta_{\mathsf{square}}, \ldots$.

In our example from section 2.1, $\vec{\theta} \cdot \vec{f}(x, y)$ could be efficiently computed as

$s \leftarrow 0$ ▷ initialize sum
$s \leftarrow s + \theta_{\text{shape}(y)}$ ▷ add weight of the single binary shape feature that fires on y: e.g., θ_{circle}
if y is striped **then**
  $s \leftarrow s + \theta_{\mathsf{striped}}$
  if y is square **then**
    $s \leftarrow s + \theta_{\mathsf{striped\ square}}$
  end if
end if
$s \leftarrow s + \theta_{\mathsf{contrast}} \cdot f_{\mathsf{contrast}}(x, y)$ ▷ where $f_{\mathsf{contrast}}(x, y) \equiv ||\text{color}(x) - \text{color}(y)||$ is not binary

2.6 Remark: Other names for log-linear models

Log-linear models $\hat{p}(y)$ and conditional log-linear models $\hat{p}(y \mid x)$ are so prevalent that they have been rediscovered many times! So you will see lots of names for them. The following are all roughly equivalent: log-linear models, Gibbs distributions, undirected graphical models, Markov Random Fields or Conditional Random Fields, exponential models, and (regularized) maximum entropy models. Special cases include logistic regression and Boltzmann machines.

In all these cases, you define a probability by multiplying some positive numbers together and dividing by a constant Z so that your probabilities will sum to 1.

3 Finding the Parameters *[read with lesson 6]*

We will try some examples in the interactive visualization where it's tricky to adjust $\vec{\theta}$ by hand. So let's ask, how *should* one adjust it?

3.1 An intuition

It turns out that maximizing equation (7) is not too hard. Suppose your training set has $N = 100$ events of which 80 are circles (some solid, some striped). But your current $\vec{\theta}$ says that circles only happen 0.7 of the time, i.e.,

$$\hat{p}(\text{striped circle}) + \hat{p}(\text{solid circle}) = 0.7$$

so your model predicts that only 70 of the 100 events should be circles.

In that case, you should increase the weight of the "circle" feature, because the **expected count** 70 is smaller than the **observed count** 80 ($E < O$) and so you need to make circles more likely in your model. Similarly, if $E > O$ then you should decrease this weight. Either way, by gradually

adjusting this weight you can make $E = O$, so that your model correctly predicts the observed number of circles.

As you've probably noticed, the feature weights do interact. Once you've made $E = O$ for the circle feature, trying to make $E = O$ for the striped feature may mess up your first move, so that $E \neq O$ again for the circle feature. However, if you keep gradually adjusting all of the features in this way, either simultaneously or one at a time, then you will eventually reach a setting where $E = O$ for all features!

3.2 The justification

The unique setting where $E = O$ for all features is actually the setting of $\vec{\theta}$ that maximizes equation (7), or equivalently, the setting that maximizes the logarithm of (7):[4]

$$F(\vec{\theta}) \stackrel{\text{def}}{=} \log \prod_{i=1}^{N} \hat{p}(y_i \mid x_i) \tag{11}$$

$$= \sum_{i=1}^{N} \log \hat{p}(y_i \mid x_i) \tag{12}$$

3.3 Formula for the gradient

You can see this by computing the partial derivatives of $F(\vec{\theta})$ with respect to the weights:

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \sum_{i=1}^{N} \frac{\partial}{\partial \theta_k} \log \hat{p}(y_i \mid x_i) \tag{13}$$

$$= \sum_{i=1}^{N} \frac{\partial}{\partial \theta_k} \log u(y_i \mid x_i) - \frac{\partial}{\partial \theta_k} \log Z(x_i) \tag{14}$$

$$\vdots$$

$$= \underbrace{\left(\sum_{i=1}^{N} f_k(x_i, y_i) \right)}_{\textit{observed count } O \textit{ of feature } k} - \underbrace{\left(\sum_{i=1}^{N} \sum_{y} \hat{p}(y \mid x_i) f_k(x_i, y) \right)}_{\textit{expected count } E \textit{ of feature } k} \tag{15}$$

At the maximum of $F(\vec{\theta})$, this derivative must be 0 for each θ_k, and therefore $E = O$ for each feature k.

Clarification: So far in the online toy, you have only been modeling $\hat{p}(y)$. However, soon you'll try $\hat{p}(y \mid x)$, and we handled that case above. What does "expected count" mean in this setting? (15) defined it as $\sum_{i=1}^{N} \sum_{y} \hat{p}(y \mid x_i) f_k(x_i, y)$. This measures how many times f_k would have fired on average if we had still observed exactly the same training contexts $x_1, \ldots x_n$, but if the corresponding training events $y_1, \ldots y_n$ were respectively replaced by random draws from the conditional distributions $\hat{p}(y \mid x_i)$.

[4] It just happens that the logarithm is easier to work with mathematically, because of the exp in (3). But why is it equivalent to maximize the logarithm? Because $p_1 > p_2$ if and only if $\log p_1 > \log p_2$. That is, $\log p$ is an increasing function of p (look at the graph of the log function).

3.4 Algorithm for maximizing the log-probability

It can be shown that F is a convex function, so it is like a simple hill with only one local maximum. There are many reasonable ways to climb to the top of this hill. One option is **gradient ascent**, a simple function maximization algorithm that increases θ_k at a rate proportional to $\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = O - E$, for all k in parallel. These rates of increase are positive or negative for $E < O$ or $E > O$ respectively. As $\vec{\theta}$ is adjusted, these rates will also gradually change, eventually reaching 0 at the top of the hill.

3.5 Details for the curious

We got to (15) above by applying basic rules of differentiation:

$$\frac{\partial}{\partial \theta_k} \log u(y \mid x) = \frac{\partial}{\partial \theta_k} \left(\sum_{k=1}^{K} \theta_k \cdot f_k(x, y) \right) \tag{16}$$

$$= \frac{\partial}{\partial \theta_k} \left(\theta_k \cdot f_k(x, y) + \text{constant not involving } \theta_k \right) = f_k(x, y) \tag{17}$$

$$\frac{\partial}{\partial \theta_k} \log Z(x) = \frac{1}{Z(x)} \frac{\partial}{\partial \theta_k} Z(x) = \frac{1}{Z(x)} \sum_y \frac{\partial}{\partial \theta_k} u(x, y) \tag{18}$$

$$= \frac{1}{Z(x)} \sum_y \frac{\partial}{\partial \theta_k} \exp(\vec{\theta} \cdot \vec{f}(x, y)) \tag{19}$$

$$= \frac{1}{Z(x)} \sum_y \left(\exp(\vec{\theta} \cdot \vec{f}(x, y)) \right) \left(\frac{\partial}{\partial \theta_k} (\vec{\theta} \cdot \vec{f}(x, z)) \right) \tag{20}$$

$$= \frac{1}{Z(x)} \sum_y u(x, y) \cdot f_k(x, y) \tag{21}$$

$$= \sum_y \hat{p}(y \mid x) \cdot f_k(x, y) \tag{22}$$

4 Regularizing the parameter estimates *[read with lesson 8]*

Problems sometimes arise when we simply maximize the log-probability. Why does this happen? Can we fix it?

4.1 Overfitting, underfitting, and the bias/variance tradeoff

You saw in lesson 8 that maximizing $F(\vec{\theta})$ might actually be a bad idea. The resulting estimate of $\vec{\theta}$ makes the *training* data very probable, but it might not generalize well to *test* data. That is called **overfitting** the training data.

In fact, there is a close connection to the bad unsmoothed estimates of section 1.2, which also overfitted. Both cases use **maximum-likelihood estimates**—that is, they model the training data as well as they can, using the available parameters.[5]

[5] Technically, the **likelihood** of the parameter vector $\vec{\theta}$ is defined by (7), which says how probable the training data would be if we used *that* $\vec{\theta}$. We have been finding the **maximum-likelihood** estimate of $\vec{\theta}$ by maximizing the likelihood (7) or equivalently the log-likelihood (12).

4.2 How many parameters?

What are the available parameters? In section 1.2 we were able to choose the probabilities individually, so we could model the training data perfectly (a bad idea!). But now we can only manipulate the probabilities indirectly through the $\vec{\theta}$ parameters of our log-linear model.

If the log-linear model has only a few parameters, that may save us from overfitting—we will be unable to fit the training data too closely, which forces us to generalize. You saw this in lesson 3 of the interactive visualization. On the other hand, if the log-linear model has enough features, as in lesson 4, then you can again fit the training data perfectly. Maximizing (7) will then get you exactly the same bad probabilities as in section 1.2.

So perhaps we should simply build a simple log-linear model with few parameters. But wait ... do we really want to break our good model and get rid of potentially useful features?? Then we might have a problem of **underfitting**—not matching the training data closely enough.

In general, it's reasonable for the number of parameters to increase with the size of the training data. We see exactly the same issue when choosing between bigram and trigram models. We should be able to match true English better with a trigram model (or better yet, a 50-gram model), but we may not have enough training data to estimate trigram probabilities accurately. In that case we fall back to a bigram model, which reduces variance (to avoid overfitting) but unfortunately increases bias (leading to underfitting).

4.3 The regularization idea

We'd prefer not to make a strict choice between pure bigram and pure trigram models. The training corpus may have plentiful evidence about *some* trigrams while lacking evidence about others. This line of thinking leads to backoff smoothing—the topic of the next assignment.

How does this idea apply to log-linear models? In general, we'd like to include a whole lot of potentially useful features in our model, but use only the ones that we can estimate accurately on our particular training data. Statisticians call this idea **regularization**. In the log-linear setting, a feature with weight 0 has no effect on the model, and a feature with low weight has little effect. So the regularization principle is "All weights should stay close to 0, except where really needed to model the data."

So instead of maximizing the log-likelihood (12), let's maximize the **regularized log-likelihood**,

$$F(\vec{\theta}) = \left(\sum_{i=1}^{N} \log \hat{p}(y_i \mid x_i) \right) - C \cdot R(\vec{\theta}) \tag{23}$$

where $R(\vec{\theta})$ is a penalty for weights that get far away from 0. A parameter $C \geq 0$, which can be tuned via cross-validation, will control the size of this penalty and hence the strength of regularization.

4.4 Types of regularization

A common choice is called ℓ_2 (or L_2) regularization. Define $R(\vec{\theta}) = C \cdot ||\vec{\theta}||^2 = C \cdot \sum_i \theta_i^2$. Then

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \text{observed count of } k - \text{expected count of } k \underbrace{- 2C \cdot \theta_k}_{\textit{new term}} \tag{24}$$

Because of the new term, gradient ascent will tend to push θ_k toward 0 (increase it if it s negative, decrease it if it's positive). For large values of C, the optimal θ_k will be usually be close to 0.

However, if feature k is sufficiently common in the training data, then $\frac{\partial}{\partial \theta_k} F(\vec{\theta})$ will be dominated by the counts, and the regularization term will have comparatively little influence. In this case, θ_k will be able to pull farther away from 0 in order to model the observations—which is exactly what we wanted to happen for well-observed features.

Another common choice is ℓ_1 (or L_1) regularization. Here $R(\vec{\theta}) = C \sum_i |\theta_i|$. Then

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \text{observed count of } k - \text{expected count of } k \underbrace{- C \cdot \text{sign}(\theta_k)}_{\textit{new term}} \tag{25}$$

This has a similar effect, but with the nice property that the optimal θ_k often equals 0 exactly. (Can you figure out why? Hint: use the formula above to graph how $F(\vec{\theta})$ varies with θ_k when θ_k is close to 0. What happens when C is large enough?)

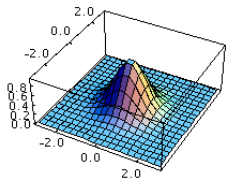
Notice that for any $C > 0$, regularization will always prevent θ_k from zooming off to ∞ or $-\infty$ (because that would make $R(\vec{\theta})$ infinitely bad). Happily, this prevents us estimating $\hat{p}(y \mid x) = 0$ as happened in lesson 8.

4.5 Remark: A Bayesian interpretation

One way to interpret this regularization is in terms of Bayes' Theorem. Suppose we want to find the most likely value of $\vec{\theta}$, given the training data. That is, instead of choosing $\vec{\theta}$ to maximize $p(\text{data} \mid \vec{\theta})$ (the **maximum likelihood** principle), we'll more sensibly choose it to maximize $p(\vec{\theta} \mid \text{data})$ (the **maximum a posteriori (MAP)** principle). By Bayes' Theorem, $p(\vec{\theta} \mid \text{data})$ is proportional to

$$\underbrace{p(\text{data} \mid \vec{\theta})}_{\textit{likelihood}} \cdot \underbrace{p(\vec{\theta})}_{\textit{prior}} \tag{26}$$

And maximizing the log of this is precisely the same as maximizing (23), provided that we define $p(\vec{\theta})$ to be proportional to $\exp -R(\vec{\theta})$. In other words, our prior distribution says that if $R(\vec{\theta})$ is large then $\vec{\theta}$ has low probability. From this point of view, ℓ_2 regularization corresponds to a Gaussian prior,[6] and ℓ_1 regularization corresponds to an exponential prior. Both of these say that *a priori*, we expect each θ_k to be close to 0.

[6] Since the exp of a quadratic function is a Gaussian function. This looks like a bell curve in k dimensions, with most of the probability near the origin: 

This choice explicitly defines the prior probability of $\vec{\theta}$ to be proportional to $\exp -\frac{1}{2\sigma^2} \sum_k \theta_k^2$, where $\sigma^2 (= 1/2C)$ is called the *variance* of the Gaussian. Clearly, this is just a convenient assumption about where the true value of $\vec{\theta}$ is most likely to fall.