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/.

The visualization is designed to help you understand log-linear models, a popular and flexible family of probability distributions. As you'll see, a conditional log-linear model defines

$$p(y \mid x) \stackrel{\text{def}}{=} \frac{1}{Z(x)} \exp \sum_{k=1}^{K} \theta_k \cdot f_k(x, y)$$

for some fixed set of real-valued feature functions f_k with adjustable weights $\theta_k \in \mathbb{R}$. For each x, the denominator $Z(x) > 0$ is a scaling factor chosen specifically to ensure that $\sum_y p(y \mid x) = 1$.

If every f_k returns values in $\{0, 1\}$ only, then the above is a scaled product of positive numbers,

$$p(y \mid x) \stackrel{\text{def}}{=} \frac{1}{Z(x)} \prod_{\substack{k \text{ such that} \\ f_k(x,y)=1}} (\exp \theta_k)$$

Playing with the visualization will help you get the hang of how changing the parameters θ_k affects the relative probabilities of different y values, and how those parameters can be set automatically. Our paper http://cs.jhu.edu/~jason/papers/#ferraro-eisner-2013 explains why we built the visualization and what we hope you'll get out of it.

1 Starting Out Concretely

1.1 Formulas for lesson 1

The four shape probabilities in lesson 1 are computed as follows, where θ_{circle} and θ_{solid} are the two parameters that you can control with the sliders. (We'll see later in section 3 that these formulas are examples of a more general pattern, but don't worry about that yet.)

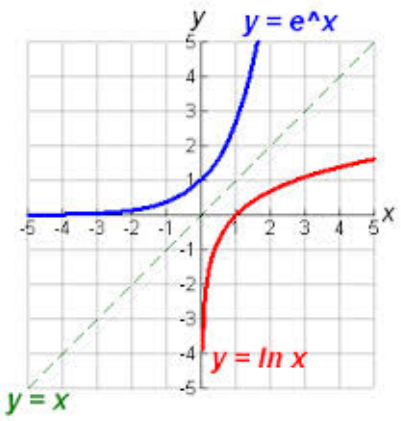
$$p(\text{solid circle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{circle}} + \theta_{\text{solid}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}}) + \exp(\theta_{\text{solid}}) + \exp 0} \tag{1}$$

$$p(\text{striped circle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{circle}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}}) + \exp(\theta_{\text{solid}}) + \exp 0} \tag{2}$$

$$p(\text{solid triangle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{solid}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}}) + \exp(\theta_{\text{solid}}) + \exp 0} \tag{3}$$

$$p(\text{striped triangle}) \stackrel{\text{def}}{=} \frac{\exp 0}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}}) + \exp(\theta_{\text{solid}}) + \exp 0} \tag{4}$$

"exp" denotes the exponential function. Recall that $\exp x$ means e^x where e is a constant (namely $e = 2.71828\ldots$, the base of natural logarithms).[1] From the graph of $y = e^x$ you can see:



- $\exp x$ is always positive (except that $\exp -\infty = 0$). So all of the numerators and denominators in (1)–(4) are ≥ 0. This makes all the probabilities ≥ 0, as required.
- As x gets bigger, so does $\exp x$. So raising θ_{solid} with your slider will raise the numerators of $p(\text{solid circle})$ (1) and $p(\text{solid triangle})$ (3). To be precise, each time you increase θ_{solid} by 1, you multiply those numerators by e. (And decreasing θ_{solid} by 1 divides them by e.)
- Finally, the four probabilities (1)–(4) always sum to 1 (thanks to their common denominator Z). That's required because those are the four possible outcomes:

$$p(\text{solid circle}) + p(\text{striped circle}) + p(\text{solid circle}) + p(\text{striped triangle}) = 1 \tag{5}$$

Hence, when raising θ_{solid} makes the solid probabilities (1) and (3) go up, the striped probabilities (2) and (4) must go down to compensate. This keeps the sum at 1. Lowering θ_{solid} does the reverse.

1.2 Formulas for lesson 2

Now every shape has exactly two features (whereas in lesson 1, some shapes had fewer):

$$p(\text{solid circle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{circle}} + \theta_{\text{solid}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}} + \theta_{\text{striped}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{striped}})}$$

$$p(\text{striped circle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{circle}} + \theta_{\text{striped}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}} + \theta_{\text{striped}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{striped}})}$$

$$p(\text{solid triangle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{triangle}} + \theta_{\text{solid}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}} + \theta_{\text{striped}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{striped}})}$$

$$p(\text{striped triangle}) \stackrel{\text{def}}{=} \frac{\exp(\theta_{\text{triangle}} + \theta_{\text{striped}})}{\exp(\theta_{\text{circle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{circle}} + \theta_{\text{striped}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{solid}}) + \exp(\theta_{\text{triangle}} + \theta_{\text{striped}})}$$

2 Why We Need Log-Linear Models *[read after lesson 2]*

Note: Lessons 1–10 will let you experiment with simple probabilities $p(y)$. Lesson 11 will finally move to the more general case of conditional probabilities $p(y \mid x)$. But this handout will *focus* on conditional probabilities, mainly in order to avoid saying everything twice. If you want to think first about the case of simple probabilities, just ignore x wherever it appears; also see section 3.4.

[1] If $y = e^x$, then $x = \log y$. That's the definition of $\log y$ (sometimes written as $\log_e y$ or $\ln y$), the natural logarithm.

2.1 Modeling probabilities

We often want to model conditional probabilities like $p(y \mid x)$, where y and x are discrete outcomes.

- They can be used to predict y directly, in a context x.
- We can build a more complicated model as a product of such probabilities.[2]

But where do we get these probabilities?

2.2 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.

2.3 Simple methods

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

$$\hat{p}(y \mid x) \stackrel{\text{def}}{=} \frac{\text{count}(x, y)}{\text{count}(x)} \tag{6}$$

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, jumping to the conclusion that y is essentially *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.[3]

3 Log-Linear Models *[read after lesson 2]*

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

[3] 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.

3.1 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 dark striped circle y against a light 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	how strongly y contrasts with background x	2

Thus, your description of this particular (x, y) is given by $\vec{f}(x, y) = (1, 0, 1, 0, 2) \in \mathbb{R}^5$. In English, you can say 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.

3.2 The defining formula

The log-linear probability model computes a contextualized event's probability from its features. 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

$$p(y \mid x) \stackrel{\text{def}}{=} \frac{\tilde{p}(x, y)}{Z(x)} \tag{7}$$

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

$$\tilde{p}(x, y) \stackrel{\text{def}}{=} \exp \sum_{k=1}^{K} (\theta_k \cdot f_k(x, y)) \tag{8}$$

$$= \exp\left(\vec{\theta} \cdot \vec{f}(x, y)\right) > 0 \tag{9}$$

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

The unnormalized probabilities $\tilde{p}(x, y)$ are always positive (because of the exp). Equation (7) has to scale them, in order to get proper probabilities that sum to 1 (i.e., $\sum_y p(y \mid x) = 1$ for each x). As you can see in (7), this is a simple matter of dividing them by a **normalizing constant**, namely

$$Z(x) \stackrel{\text{def}}{=} \sum_y \tilde{p}(x, y) \tag{10}$$

Note that the conditional log-linear modeling framework has allowed us to model $p(y \mid x)$ directly, rather than as a quotient $p(x, y)/p(x)$. We did not have to commit to any model of $p(x)$.

3.3 Feature weights

You can easily see that for any choice of $\vec{\theta}$ and any x, equation (7) 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:

$$\log p(y \mid x) = \vec{\theta} \cdot \vec{f}(x, y) - \log Z(x) \tag{11}$$

In our earlier example, setting the parameter $\theta_5 > 0$ makes contrasting colors more common: the resulting $p(y \mid x)$ 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 (11) 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 weight vector $\vec{\theta}$ that maximizes

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

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

3.4 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,

$$p(y) \stackrel{\text{def}}{=} \frac{\tilde{p}(y)}{Z} \tag{13}$$

$$\tilde{p}(y) \stackrel{\text{def}}{=} \exp\left(\vec{\theta} \cdot \vec{f}(y)\right) > 0 \tag{14}$$

$$Z \stackrel{\text{def}}{=} \sum_y \tilde{p}(y) \tag{15}$$

For example, in the first two lessons, (13) is used to define two probability models over the four shapes $y \in \mathcal{Y} = \{\text{solid circle}, \text{striped circle}, \text{solid triangle}, \text{striped triangle}\}$. Look back at section 1 of this handout to see the concrete formulas for the probabilities of the 4 shapes, which sum to 1.

In each of these models, a shape's features are computed by functions like $f_{\text{solid}}(y)$, which returns 1 on the solid circle and solid triangle, and returns 0 on the other two shapes. Therefore, the parameters of the model are feature weights like $\theta_{\text{solid}} \in \mathbb{R}$. The larger θ_{solid} is, the more the probability distribution favors the solid circle and solid triangle.

3.5 Features versus Attributes

In practical settings, the outcome y and the context x are characterized by their *attributes* (or *properties*). Each feature's value is then computed from one or more of these attributes. f_4 and f_5 from section 3.1 would be defined more formally as

$$f_4(x,y) \stackrel{\text{def}}{=} (\textsc{fill}(y) = \text{striped}) \wedge (\textsc{shape}(y) = \text{square}) \qquad \textit{"y is a striped square"} \tag{16}$$

$$f_5(x,y) \stackrel{\text{def}}{=} |\textsc{brightness}(y) - \textsc{brightness}(x)|/100 \qquad \textit{"how strongly } y \textit{ contrasts with background } x\textit{"} \tag{17}$$

which refer to the FILL, SHAPE, and BRIGHTNESS attributes of y and/or x.

You will sometimes hear people refer to individual attributes as the "features" or "basic features" of x and y. That's confusing. I recommend that you use "features" only to refer to feature functions (like f_4) or to the feature values that they return (like $f_4(x,y)$).

3.6 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 3.1, 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 3.1, $\vec{\theta} \cdot \vec{f}(x,y)$ 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**
    $s \leftarrow s + \theta_{\mathsf{striped\ square}}$
$s \leftarrow s + \theta_{\mathsf{contrast}} \cdot f_{\mathsf{contrast}}(x,y)$ ▷ where $f_{\mathsf{contrast}}(x,y) \equiv ||\text{color}(x) - \text{color}(y)||$ is not binary

3.7 Remark: Other names for log-linear models

Log-linear models $p(y)$ and conditional log-linear models $p(y \mid x)$ 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.

4 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?

4.1 An intuition

It turns out that maximizing equation (12) 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.,

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

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!

4.2 The justification

The setting where $E = O$ for all features is actually the setting of $\vec{\theta}$ that maximizes equation (12), or equivalently, the setting that maximizes the logarithm of (12):[4]

$$F(\vec{\theta}) \stackrel{\text{def}}{=} \log \prod_{i=1}^{N} p(y_i \mid x_i) = \sum_{i=1}^{N} \log p(y_i \mid x_i) \tag{18}$$

4.3 Formula for the gradient

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

[4] It just happens that the logarithm is easier to work with mathematically, because of the exp in (8). 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).

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \sum_{i=1}^{N} \frac{\partial}{\partial \theta_k} \log p(y_i \mid x_i) \tag{19}$$

$$= \sum_{i=1}^{N} \frac{\partial}{\partial \theta_k} \log \tilde{p}(x_i, y_i) - \frac{\partial}{\partial \theta_k} \log Z(x_i) \tag{20}$$

$$\vdots$$

$$= \underbrace{\left(\sum_{i=1}^{N} f_k(x_i, y_i) \right)}_{\text{observed count } O_k \text{ of feature } k} - \underbrace{\left(\sum_{i=1}^{N} \sum_{y} p(y \mid x_i) f_k(x_i, y) \right)}_{\text{expected count } E_k \text{ of feature } k} \tag{21}$$

At the max of $F(\vec{\theta})$, this derivative must be 0 for each θ_k, and hence $E_k = O_k$ for each feature k.

Clarification: So far in the online toy, you have only been modeling $p(y)$. However, soon you'll try $p(y \mid x)$ as defined in section 3.2. What does "expected count" mean in this setting? (21) defined it as $\sum_{i=1}^{N} \sum_{y} p(y \mid x_i) f_k(x_i, y)$. This counts 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 $p(y \mid x_i)$.

4.4 Algorithm for maximizing the log-probability

It can be shown that F is a concave function (see appendix B), 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**,[5] a simple function maximization algorithm that increases θ_k at a rate proportional to $\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = O_k - E_k$, for all k in parallel. These partial derivatives are positive when $E_k < O_k$ or negative when $E_k > O_k$. In the interactive visualization, they will be displayed as "hints" on the sliders if you check the "Show gradient" box (see lesson 6). You can see that as you adjust the $\vec{\theta}$ sliders, these partial derivatives also gradually change—eventually reaching 0 at the top of the hill, where $E_k = O_k$ exactly for all k.

The final value of $\vec{\theta}$—at the top of the hill—is our estimate of the "true" $\vec{\theta}$ value that we have assumed to exist. (It might sometimes be denoted by $\widehat{\vec{\theta}}$, since a "hat" indicates an estimate (as in section 2.2). The true value would be denoted by $\vec{\theta}^*$.)

[5] For large N, it is popular nowadays to use **stochastic gradient ascent**, which is faster in practice because it does not have to fully evaluate the sum $F(\vec{\theta}) = \sum_{i=1}^{N} \cdots$ or $\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \sum_{i=1}^{N} \cdots$ before updating $\vec{\theta}$. Rather, it updates to improve each summand in turn. Bottou (2012) gives a good presentation with practical advice.

4.5 Detailed derivation for the curious

We got to (21) above by applying basic rules of differentiation:

$$\frac{\partial}{\partial \theta_k} \log \tilde{p}(y \mid x) = \frac{\partial}{\partial \theta_k} \left(\sum_{k=1}^{K} \theta_k \cdot f_k(x, y) \right) \tag{22}$$

$$= \frac{\partial}{\partial \theta_k} \left(\theta_k \cdot f_k(x, y) + \text{constant not involving } \theta_k \right) = f_k(x, y) \tag{23}$$

$$\frac{\partial}{\partial \theta_k} \log Z(x) = \frac{1}{Z(x)} \frac{\partial}{\partial \theta_k} Z(x) = \frac{1}{Z(x)} \sum_y \frac{\partial}{\partial \theta_k} \tilde{p}(x, y) \tag{24}$$

$$= \frac{1}{Z(x)} \sum_y \frac{\partial}{\partial \theta_k} \exp(\vec{\theta} \cdot \vec{f}(x, y)) \tag{25}$$

$$= \frac{1}{Z(x)} \sum_y \left(\exp(\vec{\theta} \cdot \vec{f}(x, y)) \right) \left(\frac{\partial}{\partial \theta_k} (\vec{\theta} \cdot \vec{f}(x, y)) \right) \tag{26}$$

$$= \frac{1}{Z(x)} \sum_y \tilde{p}(x, y) \cdot f_k(x, y) \tag{27}$$

$$= \underbrace{\sum_y p(y \mid x) \cdot f_k(x, y)}_{\textit{expected value } e_k(x) \textit{ of feature } k \textit{ in context } x} \tag{28}$$

hence

$$\frac{\partial}{\partial \theta_k} \log p(y \mid x) = \frac{\partial}{\partial \theta_k} \log \tilde{p}(y \mid x) - \frac{\partial}{\partial \theta_k} \log Z(x) = f_k(x, y) - e_k(x) \tag{29}$$

Equation (21) simply sums this over all training pairs (x_i, y_i) to find that $\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \sum_i f_k(x_i, y_i) - e_k(x_i) = O_k - E_k$.

Cool remark: The function $-\frac{1}{N} F(\vec{\theta})$ gives the average surprisal of a training example. Differentiating with respect to θ_k, we get $\text{mean}_i\, e_k(x_i) - f_k(x_i, y_i)$ (easy to see from the above). This can be regarded as the model's *bias*—its average error in predicting feature k when using the current parameters. And appendix B shows that differentiating twice with respect to θ_j and θ_k, we get the *covariance* of the model's predictions of features j and k!

5 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?

5.1 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 2.3, 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.[6]

5.2 How many parameters?

What are the available parameters? In section 2.3 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 (12) will then get you exactly the same bad probabilities as in section 2.3.

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).

5.3 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 (18), let's maximize the **regularized log-likelihood**,

$$F(\vec{\theta}) = \left(\sum_{i=1}^{N} \log p(y_i \mid x_i) \right) - C \cdot R(\vec{\theta}) \tag{30}$$

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.

[6] Technically, the **likelihood** of the parameter vector $\vec{\theta}$ is defined by (12), 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 (12) or equivalently the log-likelihood (18).

5.4 Types of regularization

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

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \text{observed count of } k - \text{expected count of } k \underbrace{- 2C \cdot \theta_k}_{new\ term} \tag{31}$$

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 $R(\vec{\theta}) = \sum_i |\theta_i|$. Then

$$\frac{\partial}{\partial \theta_k} F(\vec{\theta}) = \text{observed count of } k - \text{expected count of } k \underbrace{- C \cdot \text{sign}(\theta_k)}_{new\ term} \tag{32}$$

This has a similar effect. See appendix A for the differences.

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 from estimating $p(y \mid x) = 0$ as happened in lesson 8.

With either ℓ_2 or ℓ_1, the function $F(\vec{\theta})$ in (30) is concave—in fact strictly concave.[7] So it has a single global maximum that can be found efficiently by greedily climbing the hill.

[7] Why? Because $F(\vec{\theta})$ is a sum of concave functions. We noted earlier that log-likelihood is a concave function (see appendix B for a proof). The regularizer $R(\vec{\theta})$, it is a strictly convex function, which becomes strictly concave when we "turn it upside down" by multiplying it by $-C < 0$. Their sum is therefore strictly concave.

A More Remarks on Regularization *[read if curious]*

A.1 Which regularizer to use?

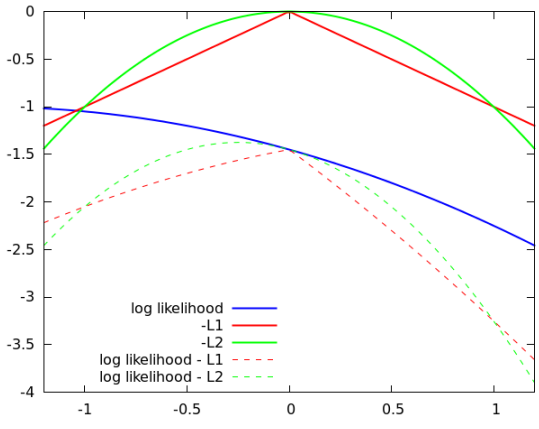
Optimization is relatively easy with ℓ_2 because the regularizer is differentiable. This lets you use standard optimization techniques that rely on the gradient (e.g., L-BFGS or stochastic gradient ascent).

Advantage of ℓ_1: Sparse solutions. The main reason to consider ℓ_1 instead is that it is a "sparsity-inducing regularizer." As we saw in lesson 9, the optimal $\vec{\theta}$ will tend to be a *sparse vector* with many zero elements, especially if C is large.

You may prefer to have many zero elements: you won't need to compute $f_k(x, y)$ at test time if $\theta_k = 0$. So this regularizer does **feature selection** for you: it identifies a small set of useful features that deserve to get nonzero weights. The resulting model may be easier for you to interpret, and is small and fast at test time.

Why does ℓ_1 regularization have this effect? This graph shows a particular parameter θ_k on the x axis. Suppose that $\theta_k = 0$. We could increase the log-likelihood by decreasing θ_k. However, this wouldn't help the log-likelihood (blue curve) as much as it would hurt the regularizer (red curve)! We therefore prefer to keep θ_k at 0, to avoid decreasing the *total* function $F(\vec{\theta})$ (dashed red curve).

In general, moving θ_k away from 0 *in either direction* will increase the ℓ_1 regularization penalty at rate C (where $C = 1$ in this drawing). Hence we will not be willing to do this unless it increases the log-likelihood at an even greater rate, that is, $|\frac{\partial}{\partial \theta_k} F(\vec{\theta})| > C$. That's not the case in the situation shown.



An ℓ_2 regularizer does *not* promote sparsity. If $\theta_k = 0$, we can again increase log-likelihood (blue curve) by decreasing θ_k—but this time, we may as well do so, since the regularizer (green curve) has a flat derivative at $\theta_k = 0$ and so will suffer only negligible harm if we move a tiny bit leftward. Notice that the total function $F(\theta)$ (dashed green curve) is differentiable and happens to have its maximum at $\theta_k < 0$. Once we use ℓ_2, there is nothing very special about 0 anymore—it would be surprising if the maximum were achieved at $\theta_k = 0$ exactly.

Computational disadvantage of ℓ_1: More complicated optimization. We already mentioned that with either ℓ_2 or ℓ_1, the objective function is concave. However, with ℓ_1, it is somewhat harder to perform the maximization. The practical difficulty is that $F(\vec{\theta})$ (dashed red line) is non-differentiable, so methods like gradient ascent and L-BFGS no longer work.[8]

[8] The simplest approach is the **subgradient method**. However, this is not very good at achieving sparsity: even when the optimum $\vec{\theta}$ is sparse, so that some elements of $\vec{\theta}$ will converge to 0 *in the limit*, these elements may never hit 0 exactly *in finite time*. A popular approach that fixes this is the **proximal gradient** algorithm, which is explained along with other approaches in the monograph of Bach et al. (2012). **Stochastic gradient ascent** (footnote 5) can also be adapted to this setting, as explained by Tsuruoka et al. (2009) and Shalev-Shwartz and Tewari (2011).

Statistical disadvantage of ℓ_1 (and of sparsity in general): Forced choices. Consider the setting where you have two features that fire in very similar circumstances on training data: $f_1(x, y) \approx f_2(x, y)$. In that case, which feature should get the higher weight? Consider 3 of the options that must be chosen among when maximizing (30):

- $\theta_1 = b,\ \theta_2 = b$ (for some constant b)
- $\theta_1 = 2b,\ \theta_2 = 0$
- $\theta_1 = 0,\ \theta_2 = 2b$

In all three cases, the model will assign pretty much the same probabilities, since $\vec{\theta} \cdot \vec{f}(x, y)$ is not changed when we shift weight between the similar features f_1 and f_2. So it is the regularizer (not the log-likelihood) that will make the choice.

The ℓ_2 regularizer prefers the first option, which divides the weight evenly between the two features. (That's because $b^2 + b^2$ is smaller than $(2b)^2 + 0^2$ or $0^2 + (2b)^2$.) Thus, it doesn't commit to one feature over the other. Both features will be considered equally at test time.

The ℓ_1 regularizer thinks all three of these cases are *equally* good. (That's because $b + b = 2b + 0 = 0 + 2b$.) In practice, however, real numbers are rarely *exactly* equal. Since the features f_1 and f_2 are not *quite* identical, something will break the tie. The choice will probably be more like this:

- $\theta_1 = 1.5,\ \theta_2 = 1.7$ (for some constant b)
- $\theta_1 = 0,\ \theta_2 = 3.4$
- $\theta_1 = 3.0,\ \theta_2 = 0$

and in this case, the ℓ_1 regularizer will prefer the third option (since it wants to minimize $|\theta_1| + |\theta_2|$). This shouldn't be surprising since ℓ_1 likes a sparse weight vector. However, ℓ_1 is putting all its eggs in one basket (as any feature selection method does). It "arbitrarily" selects f_1 as important and f_2 as irrelevant, even though they had very similar behavior on training data. It might turn out that this was the wrong decision. Perhaps f_2 is actually more helpful for this problem in general, even though f_1 appeared to be slightly more helpful on the limited sample of training data.

Why ℓ_1 and not ℓ_0? If your goal is to achieve sparsity, then ℓ_1 is not quite appropriate. You should really use ℓ_0, where $\ell_0(\vec{\theta})$ returns the number of nonzero elements of $\vec{\theta}$. Keeping this small is equivalent to keeping the vector sparse. If you really think that only a few features are relevant, then ℓ_0 assesses whether your weight vector is plausible. However, using ℓ_0 would make $F(\vec{\theta})$ really hard to maximize. So ℓ_1 is commonly used as a convex approximation to ℓ_0.[9]

In fact, what you want is probably $\ell_0 + \ell_2$, which says "Try to have a small set of nonzero weights, and keep those small." A common choice is to approximate this by $\ell_1 + \ell_2$ (the "elastic net" regularizer), giving a concave $F(\vec{\theta})$. Note that when you add two regularization terms in this way,

[9] What would it take to maximize exactly using ℓ_0 regularization? Notice that it is straightforward to maximize the log-likelihood *after* you have done feature selection. For example, you can find the values of $\theta_2, \theta_5, \theta_7$ that are optimal when you require all other weights to be 0. (The ℓ_0 regularizer would be pretty happy with this $\vec{\theta}$, since $R(\vec{\theta}) = 3$ is nice and small, but the log-likelihood may not be so great since your model only has 3 features.) You could repeat this convex maximization for *each* subset of the K features, and return the model for which the regularized log-likelihood (30) is maximized. Unfortunately, this method involves looping over all 2^K subsets! An approximate version is called **forward selection**. Instead of trying all 2^K subsets, greedily expand the feature set by one feature at a time (starting with the empty set), in a way that increases (30) as fast as possible until it can't be increased anymore. This no longer takes exponential time, but it is still rather slow.

you probably want to give them separate coefficients (both tuned on dev data). A nice trick is to optimize with $\ell_1 + \ell_2$ to accomplish feature selection, and then remove the weight-0 features and re-optimize the other weights with ℓ_2 only ("debiasing").

A.2 A interpretation of regularization as MAP estimation

One way to interpret 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

$$\underbrace{p(\text{data} \mid \vec{\theta})}_{\textit{likelihood}} \cdot \underbrace{p(\vec{\theta})}_{\textit{prior}} \tag{33}$$

And maximizing the log of this is precisely the same as maximizing (30), 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,[10] 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.

A.3 The full Bayesian treatment

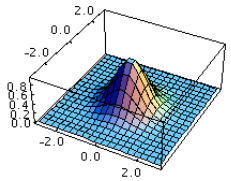
MAP estimation as in the previous section is sometimes called **empirical Bayes**. But a real Bayesian wouldn't just use the single best choice of $\vec{\theta}$. After all, Bayes' Theorem doesn't pick one $\vec{\theta}$ for us; it gives us a whole posterior probability distribution over models, $p(\vec{\theta} \mid \text{data})$. This represents our uncertainty about what the true $\vec{\theta}$ is. So ideally, we would average over all of these models, like this!

$$\hat{p}(y \mid x) = \int_{\vec{\theta} \in \mathbb{R}^K} p(y, \vec{\theta} \mid x)\, d\vec{\theta} \tag{34}$$

$$= \int_{\vec{\theta} \in \mathbb{R}^K} p(y \mid x, \vec{\theta}) \cdot p(\vec{\theta} \mid \text{data})\, d\vec{\theta} \tag{35}$$

where $p(y \mid x, \vec{\theta})$ is the conditional distribution given by a log-linear model with parameters $\vec{\theta}$.

This **fully Bayesian** approach gives a principled solution to our feature selection dificulty from appendix A.1. Suppose you have a Bayesian prior that explicitly prefers a sparse solution, for example $\exp -R(\theta)$ where R is the $\ell_0 + \ell_2$ regularizer. Then the $\vec{\theta}$ vectors that are most probable (*a posteriori* as well as *a priori*) will generally be sparse. But since we're not sure which one is

[10] 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.

correct, we average over their predictions. We are no longer choosing just one sparse vector from which to make our predictions; that is, we no longer put all our eggs in one basket.

The fully Bayesian approach to log-linear models is the "right way" to use them. Unfortunately, it is highly computationally intensive. There is some work on it, but it is not widely used in practice.

B Second derivatives *[read if curious]*

Recall equation (29):

$$\frac{\partial}{\partial \theta_k} \log p(y \mid x) = f_k(x, y) - e_k(x)$$

You can differentiate this a second time to get the second derivative $\frac{\partial}{\partial \theta_j} \frac{\partial}{\partial \theta_k} \log p(y \mid x)$, which also turns out to have a nice form:

$$\frac{\partial}{\partial \theta_j} \frac{\partial}{\partial \theta_k} \log p(y \mid x) = \underbrace{-\sum_{y'} p(y' \mid x) \left(f_j(x, y') - e_j(x)\right) \cdot \left(f_k(x, y') - e_k(x)\right)}_{\textit{covariance of features } j \textit{ and } k \textit{ in context } x} \tag{36}$$

where $e_j(x)$ and $e_k(x)$ are defined by (28). Isn't that cute?

Equation (36) is the negated covariance of $f_j(x, y')$ and $f_k(x, y')$ when y' is randomly distributed (given x) according to the model. Note that it does not depend on y.

The Hessian of $\log p(y \mid x)$ is therefore a negated covariance matrix, which means that it must be negative semidefinite. And since $F(\vec{\theta}) = \sum_i \log p(y_i \mid x_i)$, its Hessian is a sum of such matrices, so it is also negative semidefinite. This proves the claim in section 4.4 that $F(\vec{\theta})$ is concave.

Alternatively, the Hessian of $-\frac{1}{N} F(\vec{\theta})$ can be regarded directly as a covariance matrix. To be precise, it gives the covariances of the features $\vec{f}(x, y)$ when x is distributed according to the dataset and y given x is distributed according to the model. This means that $-\frac{1}{N} F(\vec{\theta})$ must be convex, again proving that $F(\vec{\theta})$ is concave. (As noted in section 4.5, the function $-\frac{1}{N} F(\vec{\theta})$ can be interpreted as "average surprisal" and its gradient can be regarded as a bias vector, which goes nicely with the observation that its Hessian can be regarded as a covariance matrix.)

B.1 Detailed derivation for the *really* curious

We can check that equation (36) is correct:

$$-\sum_{y'} p(y' \mid x) \left(f_j(x,y') - e_j(x)\right) \cdot \left(f_k(x,y') - e_k(x)\right) \tag{37}$$

$$= -\sum_{y'} p(y' \mid x) \underbrace{\left(f_j(x,y') - e_j(x)\right)}_{see\ (29)} \cdot f_k(x,y')$$

$$+ \underbrace{\left(\sum_{y'} p(y' \mid x) \left(f_j(x,y') - e_j(x)\right)\right)}_{=0\ by\ definition\ of\ e_j(x)} \cdot e_k(x) \tag{38}$$

$$= -\sum_{y'} p(y' \mid x) \underbrace{\left(\frac{\partial}{\partial \theta_j} \log p(y' \mid x)\right)}_{differentiate\ by\ chain\ rule} \cdot f_k(x,y') = -\sum_{y'} p(y' \mid x) \frac{1}{p(y' \mid x)} \frac{\partial}{\partial \theta_j} p(y' \mid x) \cdot f_k(x,y') \tag{39}$$

$$= -\sum_{y'} \left(\frac{\partial}{\partial \theta_j} p(y' \mid x)\right) \cdot f_k(x,y') \text{ where each } f_k(x,y') \text{ is a constant} \tag{40}$$

$$= -\frac{\partial}{\partial \theta_j} \underbrace{\sum_{y'} \left(p(y' \mid x) \cdot f_k(x,y')\right)}_{see\ (28)} \text{ i.e., the derivative of a linear combination} \tag{41}$$

$$= -\frac{\partial}{\partial \theta_j} e_k(x) \quad = \frac{\partial}{\partial \theta_j} \underbrace{\left(f_k(x,y) - e_k(x)\right)}_{see\ (29)\ again} \text{ for any } y, \text{ since } f_k(x,y) \text{ is a constant} \tag{42}$$

$$= \frac{\partial}{\partial \theta_j} \frac{\partial}{\partial \theta_k} \log p(y \mid x) \tag{43}$$

Step (38) notes that the difference of f_j from its expectation is 0 on average—that's in the nature of an expectation. You could write this out formally as $\sum_{y'} p(y' \mid x) \left(f_j(x,y') - e_j(x)\right) = \left(\sum_{y'} p(y' \mid x) f_j(x,y')\right) - \left(\sum_{y'} p(y' \mid x)\right) \cdot e_j(x) = e_j(x) - 1 \cdot e_j(x) = 0$.