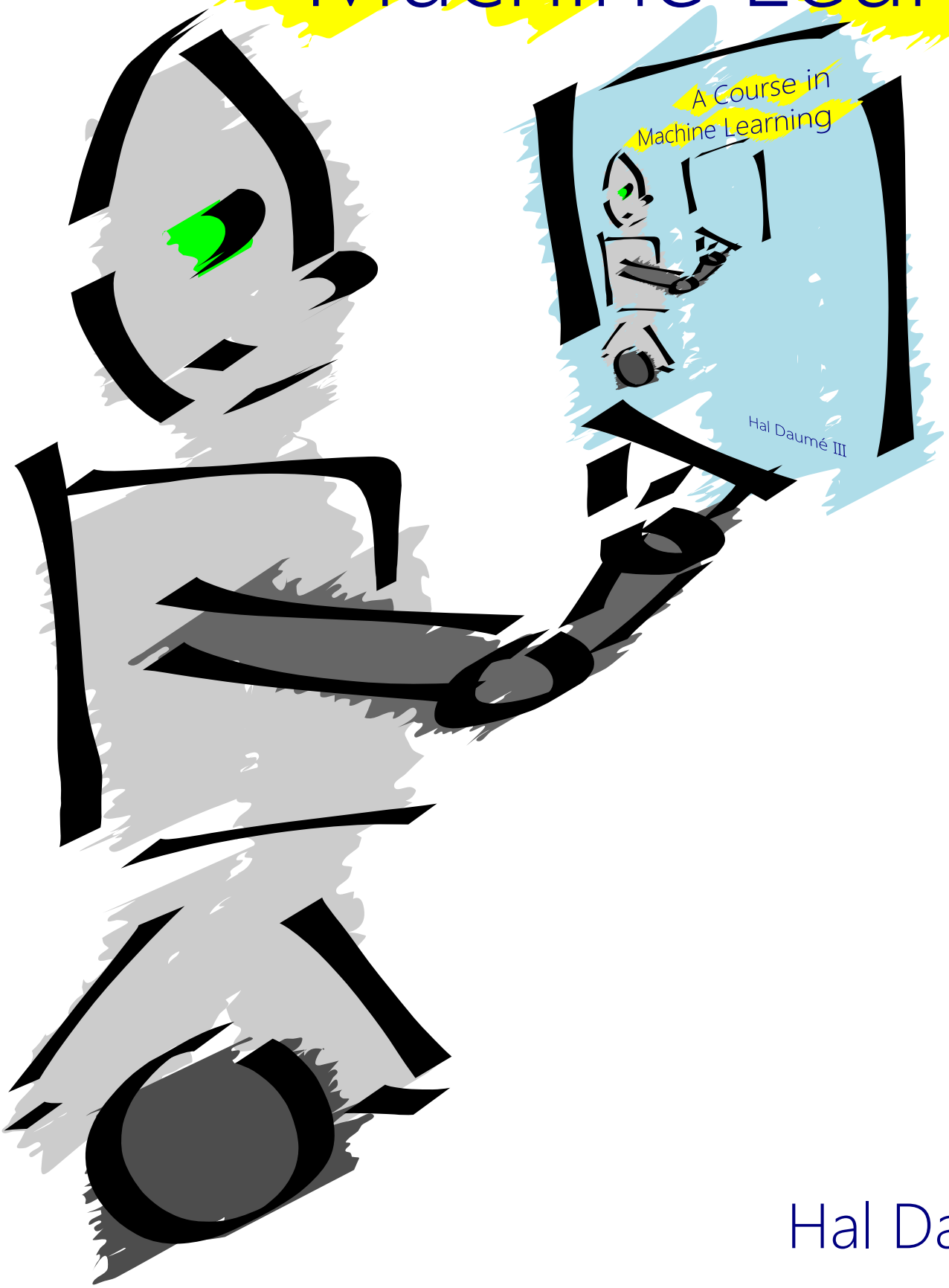
A Course in Machine Learning



Hal Daumé III

7 | Probabilistic Modeling

–

Learning Objectives:
- Define the generative story for a naive Bayes classifier.
- Derive relative frequency as the solution to a constrained optimization problem.
- Compare and contrast generative, conditional and discriminative learning.
- Explain when generative models are likely to fail.
- Derive logistic loss with an ℓ_2 regularizer from a probabilistic perspective.

Dependencies:

Many of the models and algorithms you have learned about thus far are relatively *disconnected*. There is an alternative view of machine learning that unites and generalizes much of what you have already learned. This is the **probabilistic modeling** framework, in which you will explicitly think of learning as a problem of **statistical inference**.

In this chapter, you will learn about two flavors of probabilistic models: generative and conditional. You will see that many of the approaches (both supervised and unsupervised) we have seen already can be cast as probabilistic models. Through this new view, you will be able to develop learning algorithms that have inductive biases closer to what you, as a designer, believe. Moreover, the two chapters that follow will make heavy use of the probabilistic modeling approach to open doors to other learning problems.

7.1 Classification by Density Estimation

Our underlying assumption for the majority of this book is that learning problems are characterized by some unknown probability distribution $\mathcal{D}$ over input/output pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Suppose that someone *told* you what $\mathcal{D}$ was. In particular, they gave you a Python function computeD that took two inputs, x and y, and returned the probability of that x, y pair under $\mathcal{D}$. If you had access to such a function, classification becomes simple. We can define the **Bayes optimal classifier** as the classifier that, for any test input $\hat{x}$, simply returns the $\hat{y}$ that maximizes computeD$(\hat{x}, \hat{y})$, or, more formally:

$$f^{(\mathrm{BO})}(\hat{x}) = \arg\max_{\hat{y} \in \mathcal{Y}} \mathcal{D}(\hat{x}, \hat{y}) \tag{7.1}$$

This classifier is optimal in one specific sense: of *all possible* classifiers, it achieves the smallest zero/one error.

Theorem 8 (Bayes Optimal Classifier)**.** *The Bayes Optimal Classifier $f^{(BO)}$ achieves minimal zero/one error of any deterministic classifier.*

MATH REVIEW | RULES OF PROBABILITY

chain rule, marginalization and Bayes' rule

Figure 7.1:

This theorem assumes that you are comparing against *deterministic* classifiers. You can actually prove a stronger result that $f^{(\text{BO})}$ is optimal for randomized classifiers as well, but the proof is a bit messier. However, the intuition is the same: for a given x, $f^{(\text{BO})}$ chooses the label with highest probability, thus *minimizing* the probability that it makes an error.

Proof of Theorem 8. Consider some other classifier g that claims to be better than f. Then, there must be some x on which $g(x) \neq f(x)$. Fix such an x. Now, the probability that f makes an error on this particular x is $1 - \mathcal{D}(x, f^{(\text{BO})}(x))$ and the probability that g makes an error on this x is $1 - \mathcal{D}(x, g(x))$. But $f^{(\text{BO})}$ was chosen in such a way to *maximize* $\mathcal{D}(x, f^{(\text{BO})}(x))$, so this *must* be greater than $\mathcal{D}(x, g(x))$. Thus, the probability that f errs on this particular x is smaller than the probability that g errs on it. This applies to any x for which $f(x) \neq g(x)$ and therefore f achieves smaller zero/one error than any g. □

The **Bayes error rate** (or **Bayes optimal error rate**) is the error rate of the Bayes optimal classifier. It is the best error rate you can ever hope to achieve on this classification problem (under zero/one loss).

The take-home message is that if someone gave you access to the data distribution, forming an *optimal* classifier would be trivial. Unfortunately, no one gave you this distribution, but this analysis suggests that good way to build a classifier is to try to *estimate* $\mathcal{D}$. In other words, you try to learn a distribution $\hat{\mathcal{D}}$, which you hope to very similar to $\mathcal{D}$, and then use this distribution for classification. Just as in the preceding chapters, you can try to form your estimate of $\mathcal{D}$ based on a finite training set.

The most direct way that you can attempt to construct such a probability distribution is to select a *family* of parametric distributions. For instance, a Gaussian (or Normal) distribution is parametric: it's parameters are its mean and covariance. The job of learning is then to infer which parameters are "best" as far as the observed training data is concerned, as well as whatever inductive bias you bring. A key assumption that you will need to make is that the training data you have access to is drawn **independently** from $\mathcal{D}$. In particular, as you draw examples $(x_1, y_1) \sim \mathcal{D}$ then $(x_2, y_2) \sim \mathcal{D}$ and so on, the nth draw (x_n, y_n) is drawn from $\mathcal{D}$ and *does not* otherwise depend

on the previous $n-1$ samples. This assumption is usually false, but is also usually sufficiently close to being true to be useful. Together with the assumption that all the training data is drawn from the same distribution $\mathcal{D}$ leads to the **i.i.d. assumption** or **independently and identically distributed** assumption. This is a key assumption in almost all of machine learning.

7.2 Statistical Estimation

Suppose you need to model a coin that is possibly biased (you can think of this as modeling the *label* in a binary classification problem), and that you observe data `HHTH` (where `H` means a flip came up heads and `T` means it came up tails). You can assume that all the flips came from the same coin, and that each flip was independent (hence, the data was i.i.d.). Further, you may choose to believe that the coin has a fixed probability β of coming up heads (and hence $1-\beta$ of coming up tails). Thus, the parameter of your model is simply the scalar β.

? Describe a case in which at least one of the assumptions we are making about the coin flip is false.

The most basic computation you might perform is **maximum likelihood estimation**: namely, select the paramter β the maximizes the probability of the data under that parameter. In order to do so, you need to compute the probability of the data:

$$p_\beta(D) = p_\beta(\texttt{HHTH}) \qquad \text{definition of } D \qquad (7.2)$$
$$= p_\beta(\texttt{H})p_\beta(\texttt{H})p_\beta(\texttt{T})p_\beta(\texttt{H}) \qquad \text{data is independent} \qquad (7.3)$$
$$= \beta\beta(1-\beta)\beta \qquad (7.4)$$
$$= \beta^3(1-\beta) \qquad (7.5)$$
$$= \beta^3 - \beta^4 \qquad (7.6)$$

Thus, if you want the parameter β that maximizes the probability of the data, you can take the derivative of $\beta^3 - \beta^4$ with respect to β, set it equal to zero and solve for β:

$$\frac{\partial}{\partial\beta}\left[\beta^3 - \beta^4\right] = 3\beta^2 - 4\beta^3 \qquad (7.7)$$
$$4\beta^3 = 3\beta^2 \qquad (7.8)$$
$$\iff 4\beta = 3 \qquad (7.9)$$
$$\iff \beta = \frac{3}{4} \qquad (7.10)$$

Thus, the maximum likelihood β is 0.75, which is probably what you would have selected by intuition. You can solve this problem more generally as follows. If you have H-many heads and T-many tails, the probability of your data sequence is $\beta^H(1-\beta)^T$. You can try to take the derivative of this with respect to β and follow the same recipe, but all of the products make things difficult. A more

friendly solution is to work with the **log likelihood** or **log probability** instead. The log likelihood of this data sequence is $H \log \beta + T \log(1-\beta)$. Differentiating with respect to β, you get $H/\beta - T/(1-\beta)$. To solve, you obtain $H/\beta = T/(1-\beta)$ so $H(1-\beta) = T\beta$. Thus $H - H\beta = T\beta$ and so $H = (H+T)\beta$, finally yeilding that $\beta = H/(H+T)$ or, simply, the fraction of observed data that came up heads. In this case, the maximum likelihood estimate is nothing but the relative frequency of observing heads!

? How do you know that the solution of $\beta = H/(H+T)$ is actually a maximum?

Now, suppose that instead of flipping a coin, you're rolling a K-sided die (for instance, to pick the label for a multiclass classification problem). You might model this by saying that there are parameters $\theta_1, \theta_2, \ldots, \theta_K$ specifying, respectively, the probabilities that any given side comes up on a role. Since these are themselves probabilities, each θ_k should be at least zero, and the sum of the θ_ks should be one. Given a data set that consists of x_1 rolls of 1, x_2 rolls of 2 and so on, the probability of this data is $\prod_k \theta_k^{x_k}$, yielding a log probability of $\sum_k x_k \log \theta_k$. If you pick some particular parameter, say θ_3, the derivative of this with respect to θ_3 is x_3/θ_3, which you want to equate to zero. This leads to... $\theta_3 \to \infty$.

This is obviously "wrong." From the mathematical formulation, it's correct: in fact, setting all of the θ_ks to ∞ *does* maximize $\prod_k \theta_k^{x_k}$ for any (non-negative) x_ks. The problem is that you need to constrain the θs to sum to one. In particular, you have a constraint that $\sum_k \theta_k = 1$ that you forgot to enforce. A convenient way to enforce such constraints is through the technique of **Lagrange multipliers**. To make this problem consistent with standard minimization problems, it is convenient to minimize negative log probabilities, instead of maximizing log probabilities. Thus, the *constrainted* optimization problem is:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & -\sum_k x_k \log \theta_k \\ \text{subj. to} \quad & \sum_k \theta_k - 1 = 0 \end{aligned} \tag{7.11}$$

The Lagrange multiplier approach involves adding a new variable λ to the problem (called the **Lagrange variable**) corresponding to the constraint, and to use that to move the constraint into the objective. The result, in this case, is:

$$\max_{\lambda} \min_{\boldsymbol{\theta}} \quad -\sum_k x_k \log \theta_k - \lambda \left(\sum_k \theta_k - 1 \right) \tag{7.12}$$

Turning a constrained optimization problem into it's corresponding **Lagrangian** is straightforward. The mystical aspect is why it works. In this case, the idea is as follows. Think of λ as an adversary: λ

is trying to maximize this function (you're trying to minimize it). If you pick some parameters $\boldsymbol{\theta}$ that actually satisfy the constraint, then the green term in Eq (**??**) goes to zero, and therefore λ does not matter: the adversary cannot do anything. On the other hand, if the constraint is even *slightly* unsatisfied, then λ can tend toward $+\infty$ or $-\infty$ to blow up the objective. So, in order to have a non-infinite objective value, the optimizer *must* find values of $\boldsymbol{\theta}$ that satisfy the constraint.

If we solve the *inner* optimization of Eq (**??**) by differentiating with respect to θ_1, we get $x_1/\theta_1 = \lambda$, yielding $\theta_1 = x_1/\lambda$. In general, the solution is $\theta_k = x_k/\lambda$. Remembering that the goal of λ is to enforce the sums-to-one constraint, we can set $\lambda = \sum_k x_k$ and verify that this is a solution. Thus, our optimal $\theta_k = x_k / \sum_k x_k$, which again completely corresponds to intuition.

7.3 Naive Bayes Models

Now, consider the binary classification problem. You are looking for a *parameterized* probability distribution that can describe the training data you have. To be concrete, your task might be to predict whether a movie review is positive or negative (label) based on what words (features) appear in that review. Thus, the probability for a *single* data point can be written as:

$$p_{\boldsymbol{\theta}}((y, \boldsymbol{x})) = p_{\boldsymbol{\theta}}(y, x_1, x_2, \ldots, x_D) \tag{7.13}$$

The challenge in working with a probability distribution like Eq (7.13) is that it's a distribution over a *lot* of variables. You can try to simplify it by applying the **chain rule** of probabilities:

$$\begin{aligned} p_{\boldsymbol{\theta}}(x_1, x_2, \ldots, x_D, y) &= p_{\boldsymbol{\theta}}(y) p_{\boldsymbol{\theta}}(x_1 \mid y) p_{\boldsymbol{\theta}}(x_2 \mid y, x_1) p_{\boldsymbol{\theta}}(x_3 \mid y, x_1, x_2) \\ &\quad \cdots p_{\boldsymbol{\theta}}(x_D \mid y, x_1, x_2, \ldots, x_{D-1}) && (7.14) \\ &= p_{\boldsymbol{\theta}}(y) \prod_d p_{\boldsymbol{\theta}}(x_d \mid y, x_1, \ldots, x_{d-1}) && (7.15) \end{aligned}$$

At this point, this equality is *exact* for any probability distribution. However, it might be difficult to craft a probability distribution for the 10000th feature, given the previous 9999. Even if you could, it might be difficult to accurately estimate it. At this point, you can make assumptions. A classic assumption, called the **naive Bayes assumption**, is that *the features are independent, conditioned on the label.* In the movie review example, this is saying that *once you know that it's a positive review*, the probability that the word "excellent" appears is independent of whether "amazing" also appeared. (Note that this does *not* imply that these words are independent when you

don't know the label—they most certainly are not.) Formally this assumption states that:

$$\textbf{Assumption:} \quad p(x_d \mid y, x_{d'}) = p(x_d \mid y) \quad , \quad \forall d \neq d' \tag{7.16}$$

Under this assumption, you can simplify Eq (7.15) to:

$$p_{\theta}((y, \boldsymbol{x})) = p_{\theta}(y) \prod_d p_{\theta}(x_d \mid y) \qquad \text{naive Bayes assumption} \tag{7.17}$$

At this point, you can start parameterizing p. Suppose, for now, that your labels are binary *and* your features are also binary. In this case, you could model the label as a biased coin, with probability of heads (e.g., positive review) given by θ_0. Then, for each label, you can imagine having one (biased) coin for each feature. So if there are D-many features, you'll have $1 + 2D$ total coins: one for the label (call it θ_0) and one for each label/feature combination (call these $\boldsymbol{\theta}_{+1}$ and as $\boldsymbol{\theta}_{-1}$). In the movie review example, we might expect $\theta_0 \approx 0.4$ (forty percent of movie reviews are positive) and also that θ_{+1} might give high probability to words like "excellent" and "amazing" and "good" and $\boldsymbol{\theta}_{-1}$ might give high probability to words like "terrible" and "boring" and "hate". You can rewrite the probability of a single example as follows, eventually leading to the log probability of the entire data set:

$$p_{\theta}((y, \boldsymbol{x})) = p_{\theta}(y) \prod_d p_{\theta}(x_d \mid y) \qquad \text{naive Bayes assumption} \tag{7.18}$$

$$= \left(\theta_0^{[y=+1]} (1 - \theta_0)^{[y=-1]}\right) \prod_d \theta_{(y),d}^{[x_d=1]} (1 - \theta_{(y),d})^{[x_d=0]} \qquad \text{model assumptions} \tag{7.19}$$

Solving for θ_0 is identical to solving for the biased coin case from before: it is just the relative frequency of positive labels in your data (because θ_0 doesn't depend on $\boldsymbol{x}$ at all). For the other parameters, you can repeat the same exercise as before for each of the $2D$ coins independently. This yields:

$$\hat{\theta}_0 = \frac{1}{N} \sum_n [y_n = +1] \tag{7.20}$$

$$\hat{\theta}_{(+1),d} = \frac{\sum_n [y_n = +1 \wedge x_{n,d} = 1]}{\sum_n [y_n = +1]} \tag{7.21}$$

$$\hat{\theta}_{(-1),d} = \frac{\sum_n [y_n = -1 \wedge x_{n,d} = 1]}{\sum_n [y_n = -1]} \tag{7.22}$$

In the case that the features are *not* binary, you need to choose a different model for $p(x_d \mid y)$. The model we chose here is the **Bernouilli distribution**, which is effectively a distribution over independent

MATH REVIEW | COMMON PROBABILITY DISTRIBUTIONS

remove people about discrete, bernoulli, binomial, multinomial and gaussian distributions

Figure 7.2:

coin flips. For other types of data, other distributions become more appropriate. The die example from before corresponds to a **discrete distribution**. If the data is continuous, you might choose to use a **Gaussian distribution** (aka **Normal distribution**). The choice of distribution is a form of **inductive bias** by which you can inject your knowledge of the problem into the learning algorithm.

7.4 Prediction

Consider the *predictions* made by the naive Bayes model with Bernoulli features in Eq (7.18). You can better understand this model by considering its decision boundary. In the case of probabilistic models, the decision boundary is the set of inputs for which the likelihood of $y = +1$ is precisely 50%. Or, in other words, the set of inputs $\boldsymbol{x}$ for which $p(y = +1 \mid \boldsymbol{x})/p(y = -1 \mid \boldsymbol{x}) = 1$. In order to do this, the first thing to notice is that $p(y \mid \boldsymbol{x}) = p(y, \boldsymbol{x})/p(\boldsymbol{x})$. In the ratio, the $p(\boldsymbol{x})$ terms cancel, leaving $p(y = +1, \boldsymbol{x})/p(y = -1, \boldsymbol{x})$. Instead of computing this ratio, it is easier to compute the **log-likelihood ratio** (or LLR), $\log p(y = +1, \boldsymbol{x}) - \log p(y = -1, \boldsymbol{x})$, computed below:

$$\text{LLR} = \log\left[\theta_0 \prod_d \theta_{(+1),d}^{[x_d=1]}(1-\theta_{(+1),d})^{[x_d=0]}\right] - \log\left[(1-\theta_0)\prod_d \theta_{(-1),d}^{[x_d=1]}(1-\theta_{(-1),d})^{[x_d=0]}\right] \quad \text{model assumptions} \tag{7.23}$$

$$= \log\theta_0 - \log(1-\theta_0) + \sum_d [x_d = 1]\left(\log\theta_{(+1),d} - \log\theta_{(-1),d}\right) + \sum_d [x_d = 0]\left(\log(1-\theta_{(+1),d}) - \log(1-\theta_{(-1),d})\right) \quad \text{take logs and rearrange} \tag{7.24}$$

$$= \sum_d x_d \log\frac{\theta_{(+1),d}}{\theta_{(-1),d}} + \sum_d (1-x_d)\log\frac{1-\theta_{(+1),d}}{1-\theta_{(-1),d}} + \log\frac{\theta_0}{1-\theta_0} \quad \text{simplify log terms} \tag{7.25}$$

$$= \sum_d x_d\left[\log\frac{\theta_{(+1),d}}{\theta_{(-1),d}} - \log\frac{1-\theta_{(+1),d}}{1-\theta_{(-1),d}}\right] + \sum_d \log\frac{1-\theta_{(+1),d}}{1-\theta_{(-1),d}} + \log\frac{\theta_0}{1-\theta_0} \quad \text{group } x\text{-terms} \tag{7.26}$$

$$= \boldsymbol{x}\cdot\boldsymbol{w} + b \tag{7.27}$$

$$w_d = \log \frac{\theta_{(+1),d}(1-\theta_{(-1),d})}{\theta_{(-1),d}(1-\theta_{(+1),d})} \quad , \quad b = \sum_d \log \frac{1-\theta_{(+1),d}}{1-\theta_{(-1),d}} + \log \frac{\theta_0}{1-\theta_0} \tag{7.28}$$

The result of the algebra is that the naive Bayes model has precisely the form of a linear model! Thus, like perceptron and many of the other models you've previous studied, the decision boundary is linear.

TODO: MBR

7.5 Generative Stories

A useful way to develop probabilistic models is to tell a **generative story**. This is a *fictional* story that explains how you believe your training data came into existence. To make things interesting, consider a multiclass classification problem, with continuous features modeled by independent Gaussians. Since the label can take values $1 \dots K$, you can use a discrete distribution (die roll) to model it (as opposed to the Bernoilli distribution from before):

1. For each example $n = 1 \dots N$:
   (a) Choose a label $y_n \sim \mathcal{D}isc(\boldsymbol{\theta})$
   (b) For each feature $d = 1 \dots D$:
      i. Choose feature value $x_{n,d} \sim \mathcal{N}or(\mu_{y_n,d}, \sigma^2_{y_n,d})$

This generative story can be directly translated into a likelihood function by replacing the "for each"s with products:

$$p(D) = \overbrace{\prod_n \underbrace{\theta_{y_n}}_{\text{choose label}} \underbrace{\prod_d \underbrace{\frac{1}{\sqrt{2\pi\sigma^2_{y_n,d}}} \exp\left[-\frac{1}{2\sigma^2_{y_n,d}}(x_{n,d}-\mu_{y_n,d})^2\right]}_{\text{choose feature value}}}_{\text{for each feature}}}^{\text{for each example}} \tag{7.29}$$

You can take logs to arrive at the log-likelihood:

$$\log p(D) = \sum_n \left[\log \theta_{y_n} + \sum_d -\frac{1}{2}\log(\sigma^2_{y_n,d}) - \frac{1}{2\sigma^2_{y_n,d}}(x_{n,d}-\mu_{y_n,d})^2\right] + const \tag{7.30}$$

To optimize for $\boldsymbol{\theta}$, you need to add a "sums to one" constraint as before. This leads to the previous solution where the θ_ks are proportional to the number of examples with label k. In the case of the μs

you can take a derivative with respect to, say $\mu_{k,i}$ and obtain:

$$\frac{\partial \log p(D)}{\partial \mu_{k,i}} = \frac{\partial}{\partial \mu_{k,i}} - \sum_n \sum_d \frac{1}{2\sigma^2_{y_n,d}} (x_{n,d} - \mu_{y_n,d})^2 \qquad \text{ignore irrelevant terms} \tag{7.31}$$

$$= \frac{\partial}{\partial \mu_{k,i}} - \sum_{n:y_n=k} \frac{1}{2\sigma^2_{k,d}} (x_{n,i} - \mu_{k,i})^2 \qquad \text{ignore irrelevant terms} \tag{7.32}$$

$$= \sum_{n:y_n=k} \frac{1}{\sigma^2_{k,d}} (x_{n,i} - \mu_{k,i}) \qquad \text{take derivative} \tag{7.33}$$

Setting this equal to zero and solving yields:

$$\mu_{k,i} = \frac{\sum_{n:y_n=k} x_{n,i}}{\sum_{n:y_n=k} 1} \tag{7.34}$$

Namely, the sample mean of the ith feature of the data points that fall in class k. A similar analysis for $\sigma^2_{k,i}$ yields:

$$\frac{\partial \log p(D)}{\partial \sigma^2_{k,i}} = \frac{\partial}{\partial \sigma^2_{k,i}} - \sum_{y:y_n=k} \left[\frac{1}{2} \log(\sigma^2_{k,i}) + \frac{1}{2\sigma^2_{k,i}} (x_{n,i} - \mu_{k,i})^2 \right] \qquad \text{ignore irrelevant terms} \tag{7.35}$$

$$= - \sum_{y:y_n=k} \left[\frac{1}{2\sigma^2_{k,i}} - \frac{1}{2(\sigma^2_{k,i})^2} (x_{n,i} - \mu_{k,i})^2 \right] \qquad \text{take derivative} \tag{7.36}$$

$$= \frac{1}{2\sigma^4_{k,i}} \sum_{y:y_n=k} \left[(x_{n,i} - \mu k, i)^2 - \sigma^2_{k,i} \right] \qquad \text{simplify} \tag{7.37}$$

You can now set this equal to zero and solve, yielding:

$$\sigma^2_{k,i} = \frac{\sum_{n:y_n=k} (x_{n,i} - \mu_{k,i})^2}{\sum_{n:y_n=k} 1} \tag{7.38}$$

Which is just the sample variance of feature i for class k.

? What would the estimate be if you decided that, for a given class k, all features had equal variance? What if you assumed feature i had equal variance for each class? Under what circumstances might it be a good idea to make such assumptions?

7.6 Conditional Models

In the foregoing examples, the task was formulated as attempting to model the **joint** distribution of $(\boldsymbol{x}, y)$ pairs. This may seem wasteful: at prediction time, all you care about is $p(y \mid \boldsymbol{x})$, so why not model it directly?

Starting with the case of regression is actually somewhat simpler than starting with classification in this case. Suppose you "believe"

that the relationship between the real value y and the vector $\boldsymbol{x}$ should be linear. That is, you expect that $y = \boldsymbol{w} \cdot \boldsymbol{x} + b$ *should hold* for some parameters $(\boldsymbol{w}, b)$. Of course, the data that you get does not exactly obey this: that's fine, you can think of deviations from $y = \boldsymbol{w} \cdot \boldsymbol{x} + b$ as *noise*. To form a probabilistic model, you must assume some distribution over noise; a convenient choice is zero-mean Gaussian noise. This leads to the following generative story:

1. For each example $n = 1 \dots N$:
   (a) Compute $t_n = \boldsymbol{w} \cdot \boldsymbol{x}_n + b$
   (b) Choose noise $e_n \sim \mathcal{N}or(0, \sigma^2)$
   (c) Return $y_n = t_n + e_n$

In this story, the variable t_n stands for "target." It is the noiseless variable that you do not get to observe. Similarly e_n is the error (noise) on example n. The value that you actually get to observe is $y_n = t_n + e_n$. See Figure 7.3.

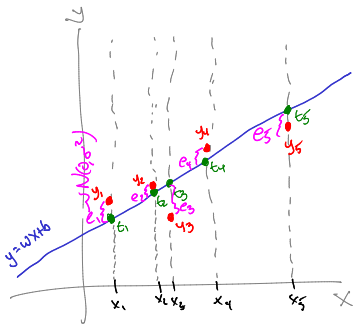


Figure 7.3: pictorial view of targets versus labels

A basic property of the Gaussian distribution is additivity. Namely, that if $a \sim \mathcal{N}or(\mu, \sigma^2)$ and $b = a + c$, then $b \sim \mathcal{N}or(\mu + c, \sigma^2)$. Given this, from the generative story above, you can derive a shorter generative story:

1. For each example $n = 1 \dots N$:
   (a) Choose $y_n \sim \mathcal{N}or(\boldsymbol{w} \cdot \boldsymbol{x}_n + b, \sigma^2)$

Reading off the log likelihood of a dataset from this generative story, you obtain:

$$\log p(D) = \sum_n \left[-\frac{1}{2} \log(\sigma^2) - \frac{1}{2\sigma^2} (\boldsymbol{w} \cdot \boldsymbol{x}_n + b - y_n)^2 \right] \quad \text{model assumptions} \tag{7.39}$$

$$= -\frac{1}{2\sigma^2} \sum_n (\boldsymbol{w} \cdot \boldsymbol{x}_n + b - y_n)^2 + const \quad \text{remove constants} \tag{7.40}$$

This is precisely the linear regression model you encountered in Section 6.6! To minimizing the *negative* log probability, you need only solve for the regression coefficients $\boldsymbol{w}, b$ as before.

In the case of binary classification, using a Gaussian noise model does not make sense. Switching to a Bernoulli model, which describes binary outcomes, makes more sense. The only remaining difficulty is that the parameter of a Bernoulli is a value between zero and one (the probability of "heads") so your model must produce such values. A classic approach is to produce a real-valued target, as before, and then transform this target into a value between zero and

one, so that $-\infty$ maps to 0 and $+\infty$ maps to 1. A function that does this is the logistic function[1], defined below and plotted in Figure **??**:

[1] Also called the **sigmoid function** because of it's "S"-shape.

$$\text{Logistic function:} \quad \sigma(z) = \frac{1}{1+\exp[-z]} = \frac{\exp z}{1+\exp z} \tag{7.41}$$

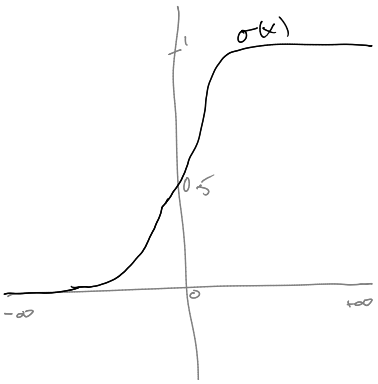


Figure 7.4: sketch of logistic function

The logistic function has several nice properties that you can verify for yourself: $\sigma(-z) = 1 - \sigma(z)$ and $\partial\sigma/\partial z = z\sigma^2(z)$.

Using the logistic function, you can write down a generative story for binary classification:

1. For each example $n = 1 \dots N$:
   (a) Compute $t_n = \sigma(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)$
   (b) Compute $z_n \sim \mathcal{B}er(t_n)$
   (c) Return $y_n = 2z_n - 1$ (to make it ± 1)

The log-likelihood for this model is:

$$\log p(D) = \sum_n \Big[[y_n = +1] \log \sigma(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) + [y_n = -1] \log \sigma(-\boldsymbol{w} \cdot \boldsymbol{x}_n + b) \Big] \qquad \text{model and properties of } \sigma \tag{7.42}$$

$$= \sum_n \log \sigma(y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)) \qquad \text{join terms} \tag{7.43}$$

$$= -\sum_n \log[1 + \exp(-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b))] \qquad \text{definition of } \sigma \tag{7.44}$$

$$= -\sum_n \ell^{(\log)}(y_n, \boldsymbol{w} \cdot \boldsymbol{x}_n + b) \qquad \text{definition of } \ell^{(\log)} \tag{7.45}$$

As you can see, the log-likelihood is *precisely* the negative of (a scaled version of) the logistic loss from Chapter 6. This model is the **logistic regression** model, and this is where logisitic loss originally derived from.

TODO: conditional versus joint

7.7 Regularization via Priors

In the foregoing discussion, parameters of the model were selected according to the maximum likelihood criteria: find the parameters $\boldsymbol{\theta}$ that maximize $p_{\boldsymbol{\theta}}(D)$. The trouble with this approach is easy to see even in a simple coin flipping example. If you flip a coin twice and it comes up heads both times, the maximum likelihood estimate

for the bias of the coin is 100%: it will *always* come up heads. This is true even if you had only flipped it once! If course if you had flipped it one million times and it had come up heads every time, *then* you might find this to be a reasonable solution.

This is clearly undesirable behavior, especially since data is expensive in a machine learning setting. One solution (there are others!) is to seek parameters that balance a tradeoff between the likelihood of the data and some prior belief you have about what values of those parameters are likely. Taking the case of the logistic regression, you might a priori believe that small values of $\boldsymbol{w}$ are more likely than large values, and choose to represent this as a Gaussian prior on each component of $\boldsymbol{w}$.

The **maximum a posteriori** principle is a method for incoporating both data and prior beliefs to obtain a more balanced parameter estimate. In abstract terms, consider a probabilistic model over data D that is parameterized by parameters θ. If you think of the parameters as just another random variable, then you can write this model as $p(D \mid \theta)$, and maximum likelihood amounts to choosing θ to maximize $p(D \mid \theta)$. However, you might instead with to maximize the probability of the *parameters*, given the data. Namely, maximize $p(\theta \mid D)$. This term is known as the **posterior** distribution on θ, and can be computed by Bayes' rule:

$$\underbrace{p(\theta \mid D)}_{\text{posterior}} = \frac{\overbrace{p(\theta)}^{\text{prior}}\overbrace{p(D \mid \theta)}^{\text{likelihood}}}{\underbrace{p(D)}_{\text{evidence}}} \quad \text{, where} \quad p(D) = \int \mathrm{d}\theta p(\theta) p(D \mid \theta) \tag{7.46}$$

This reads: the **posterior** is equal to the **prior** times the **likelihood** divided by the **evidence**.[2] The evidence is a scary-looking term (it has an integral!) but note that from the perspective of seeking parameters θ than maximize the posterior, the evidence is just a constant (it does not depend on θ) and therefore can be ignored.

[2] The evidence is sometimes called the **marginal likelihood**.

Returning to the logistic regression example with Gaussian priors on the weights, the **log posterior** looks like:

$$\log p(\theta \mid D) = -\sum_n \ell^{(\log)}(y_n, \boldsymbol{w} \cdot \boldsymbol{x}_n + b) - \sum_d \frac{1}{2\sigma^2} w_d^2 + const \qquad \text{model definition} \tag{7.47}$$

$$= -\sum_n \ell^{(\log)}(y_n, \boldsymbol{w} \cdot \boldsymbol{x}_n + b) - \frac{1}{2\sigma^2} ||\boldsymbol{w}||^2 \tag{7.48}$$

and therefore reduces to a regularized logistic function, with a squared 2-norm regularizer on the weights. (A 1-norm regularizer

can be obtained by using a Laplace prior on w rather than a Gaussian prior on w.)

7.8 Exercises

Exercise 7.1. TODO...