目录

Information and entropy (任何 question 从 16 版视频找答案更细致)

1 Information

Let X be a random variable taking values in $\{1, 2, \ldots, k\}$. Suppose that two people — Alice and Bob — both know its distribution $P(X)$, but only Alice knows the actual value of X. Alice obviously has more information than Bob, but how much more?

Alice 和 Bob 都知道信息 X 的 ditribution

To answer this question, we can ask Alice to tell Bob the value of X. If she can do so using a message of b bits, then we say that Alice has at most b bits more information than Bob does.

Alice 比 Bob 多多少信息？转化成每次 Alice 需要 tell Bob 多少大小的信息。（每次！！！）这就是 entropy！！！

因为 Bob 不知道，Alice 需要告诉他多少。

最后转化成，我们最少用多少长度的信息

For this purpose, Alice and Bob can agree ahead of time on a string of b bits for each possible value of X. Then, after Alice finds out the actual value of X, she can transmit the corresponding string to Bob. The strings are called *code words*, and the assignment of strings to values of X is called a *code*.

对 information X 进行 encode

For example, if X has $k = 4$ possible values, then Alice could assign the code words $00, 01, 10, 11$ to these values, and inform Bob about X using a message of length $b = 2$. So, when $k = 4$, X represents at most 2 bits worth of information.

每次传输最多 2bits！

正负代表传输方向

2. Direction of information transfer

When measuring information, it's important to pay attention to the direction of information transfer. In our example above, Alice has the extra information, and is trying to send it to Bob; the number of bits in the message measures the amount of information.

If we tried to describe Bob's extra information instead of Alice's, we'd get a negative number of bits. This doesn't mean a negative-length message: instead it means that Bob has to receive a message instead of sending one.

如果 Bob 发给 Alice 就是负

3. Not all distributions are created equal

We saw above that Alice can tell Bob about X by sending a two-bit message. Is this the best that Alice can do? The answer depends on the distribution $P(X)$. Intuitively, the two distributions

$$(0.25, 0.25, 0.25, 0.25) \qquad (0.5, 0.25, 0.125, 0.125)$$

are different: in the one on the left, Bob has no idea what value of X to expect, while in the one on the right, Bob knows even before seeing the message that $X = 1$ is more likely than $X = 3$.

To take advantage of this fact, Alice can assign *shorter* code words to *more probable* values of X. Then, on average, she will send fewer bits. We will measure the information in the distribution $P(X)$ by the *expected* number of bits that Alice sends.

左边 uniform，Bob 很难猜到，右边有规律，Bob 有了一部分信息

distribution 决定我们的 encoding

为了 save bits，**频率越高的信息，编码少一点**，**最终的期望长度就是 Alice 需要发送的信息.最终平均长度减少！！**

For example, suppose Alice picks the code words

$$0, 10, 110, 111$$

for the four possible values of X. Then she will send, in expectation,

$$0.5 \cdot 1 + 0.25 \cdot 2 + 0.125 \cdot 3 + 0.125 \cdot 3 = 1.75 < 2$$

bits. We will see below that this value is optimal: no code can use fewer than 1.75 bits (in expectation) for this distribution.

code 的长度分别是 1,2,3,3

每次传输平均少于 2bits

A subtlety is that the code should satisfy the *prefix property* (as the code above does): no code word can be a prefix of any other code word. That way, Bob always knows when the message from Alice is over. Without the prefix property, we'd have to devise some other way to notify Bob when the message is over — and, we'd have to count the information transmitted by these end-of-message notifications.

每一个编码不能出现在**别的编码 prefix**

假设发送编码 10，如果不出现在其他 prefix 就意味结束，如果出现，就不知道是否结束

2 Entropy(low bound)

Suppose we know a code that lets Alice tell Bob about X using b bits (on average). Then, we know that Alice has at most b bits more information than Bob does (on average). But b is only an upper bound: for all we know, there might be a better code that Alice could have used.

每次 code 不得少于 entropy

It turns out that there is a lower bound as well, although we will not prove it here: if P is the distribution of X, then no code can use fewer than

$$H(P) = -\sum_{x=1}^{k} P(x)\log_2(P(x)) \text{ bits.}$$

The value $H(P)$ is called the *entropy* of the distribution P. By convention we also write $H(X)$ even though the entropy is a function of the *distribution* of X rather than of X itself.

For example, the entropy of the distribution $(0.5, 0.25, 0.125, 0.125)$ is

$$0.5 \cdot 1 + 0.25 \cdot 2 + 0.125 \cdot 3 + 0.125 \cdot 3 = 1.75$$

记住 $-\log_2(P(x))$ 是用来计算每个 code 的长度！！！乘以其概率就是期望长度！！

lower bound 就是 entropy(只包含 P(X)分布)

entropy of distribution

entropy 是 distribution P 的函数！！因为式子里只有 distribution 是变量！

The lower bound based on entropy is tight: if Alice needs to tell Bob about a sequence of i.i.d. samples from a distribution $P(X)$, then she can construct a code that uses exactly $H(X)$ bits per sample on average in the limit. (The method of *arithmetic coding* is one way to build such a code.)

For this reason, we can think of entropy as a measure of information: by knowing X, on average, Alice has $H(X)$ bits more information than Bob does. Equivalently, entropy measures Bob's uncertainty or lack of information: it is the expected number of additional bits he needs to become certain about X.

Note: If Alice just needs to tell Bob about a single sample from $P(X)$, she might need more than $H(X)$ bits, since every message has to contain an integer number of bits. (To see the problem: how many bits should Alice assign to a message with probability 0.333?) We'll neglect this integrality effect when calculating information. Equivalently, we'll suppose that there are always lots of messages that Alice needs to send Bob; then she can use a method (like arithmetic coding) that avoids wasting the extra fractional bits.

entropy 代表 Alice 平均比 Bob 多多少 bit 信息

也代表 bob 缺少的信息或者不确定性(uniform entropy 最大，不确

定性最强)

5. Limiting cases

The highest entropy distribution on k outcomes is the uniform distribution, with entropy $H(X) = \log_2 k$ bits. The closer a distribution is to uniform, the higher its entropy.

The lowest entropy distributions are the ones that indicate certainty, with $P(X = x) = 1$ for some x; these distributions have $H(X) = 0$ bits. (By convention, $0 \log_2 0 = 0$; this definition makes the function $p \log_2 p$ continuous as $p \downarrow 0$.)

entropy 代表不确定性！！

uniform 是最不确定的，熵最大

越确定，entropy 越小！！如果其中一个 p(X)=1 其他 X 为 0，因为 log1=0，最终 entropy 为 0！代表确定，不用传输信息，别人已经知道了

6. Why binary?

We have assumed so far that Alice sends Bob a string of bits. This assumption is not necessary: Alice could use an alphabet of some other size a. For example, if Alice can use the digits $0, 1, 2$ instead of $0, 1$, then $a = 3$.

In this case we measure entropy in base a instead of base 2:

$$H_a(X) = -\sum_{x=1}^{k} P(x) \log_a(P(x)) \text{ symbols.}$$

The difference is only a scaling factor, since for any p, $\log_2(p)/\log_a(p) = \log_2(a)$. We can say that s symbols are the same as $s \log_2(a)$ bits.

不一定是二进制编码 ，用多少位编码，a 就等于几

记住 $-\log_2(P(x))$ 是用来计算每个 code 的长度！！！

question？最后一句 symbols?

We can even use non-integer bases such as e via the convention that we convert to an integer base before sending any messages. In base e, we measure information in *nats* instead of bits: 1 nat ≈ 1.4427 bits.

2 joint entropy of X,Y

Now suppose that there are two random variables X and Y, and Alice and Bob both know their joint distribution $P(X,Y)$. Suppose Alice knows X and Y, but Bob does not. How much more information does Alice have than Bob?

This is actually the same question we asked above: define a new random variable Z equal to the tuple (X,Y). Then Alice's extra information is $H(Z)$: that is, she has

$$H(Z) = -\sum_z P(z)\log_2 P(z) \text{ bits}$$

$$-\sum_x \sum_y P(x,y)\log_2 P(x,y) \text{ bits}$$ XY 所有可能组合值相加

sum over all possible values

也是在用 2 进制编码的条件下

3 condition entropy

在多变量的情况下，如果已知其中一个

What if Bob now finds out Y (but not X)? In this case, Bob gets information worth $H(Y)$ on average. Since Alice used to have extra information $H(X,Y)$ compared to Bob, she now has only

$$H(X,Y) - H(Y)$$

extra information on average. We call this value the *conditional entropy* $H(X \mid Y)$. We can derive another expression for the conditional entropy via the definition of conditional probability:

Bob 知道了 Y，就有了 H(Y)

question 为什么减去？？证明

$$\begin{aligned}
&= H(X,Y) - H(Y) \\
&= -\sum_x \sum_y P(x,y)\log_2 P(x,y) + \sum_y P(y)\log_2 P(y) \\
&= -\sum_x \sum_y P(x,y)\log_2 (P(x,y)/P(y)) \\
&= -\sum_x \sum_y P(x,y)\log_2 P(x \mid y)
\end{aligned}$$

(To go from the second line to the third, we used the sum rule: for any possible value y of Y, we have $P(y) = \sum_x P(x, y)$.)

p(y)= sum 带入第二行的第二项第一个 p(y)

然后可以合并，得到上面

换一种思路：

y 是特定 Y 值下的 entropy

p(X|Y=y)其实就是 X 的概率分布，它的 entropy：

We can also write

$$H(X \mid Y = y) = H(P(X \mid Y = y))$$

entropy 和条件概率有关，是条件概率的函数，也就是当 Bob 知道 Y 是 y，此时 Alice 的信息是 H(P(X|Y=y))

用上面上面的公式 H(XY)-H(Y)：$= -\sum_x \sum_y P(x, y) \log_2 P(x \mid y)$ 又

p(xy)=p(x|y)p(y)

带入上面式子得：$-\sum_y \sum_x p(x|y)\log_2 p(x|y)\cdot p(y)$

左边 $\sum_x p(x|y)\log_2 p(x|y)$ 是 H(P(X|Y=y))，因为 P(X|Y=y)是 X 的分布，y 是固定的常数

$-\sum_y \sum_x p(x|y)\log_2 p(x|y)\cdot p(y)$ 是 H(P(X|Y=y))的期望

$P(x, y) = P(x \mid y)\, P(y)$ for all values x, y, we have

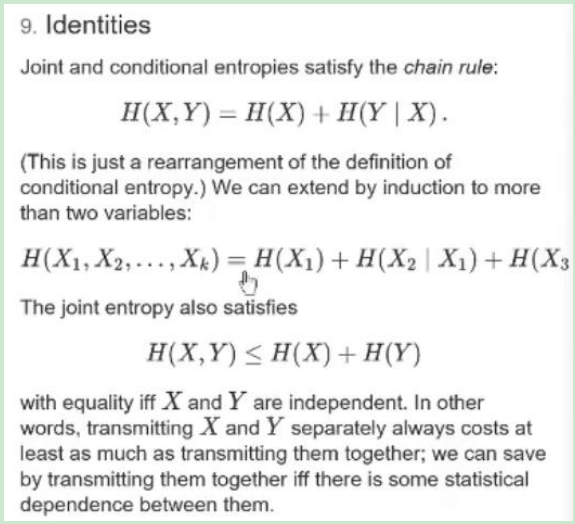
$$H(X \mid Y) = E_Y(H(P(X \mid Y = y))).$$

That is, the conditional entropy is the expectation of the entropy of the conditional distribution.

conditional entropy 的解读

H(X|Y)=H(X,Y)-H(Y)的证明

4 性质



9. Identities

Joint and conditional entropies satisfy the *chain rule*:

$$H(X,Y) = H(X) + H(Y \mid X).$$

(This is just a rearrangement of the definition of conditional entropy.) We can extend by induction to more than two variables:

$$H(X_1, X_2, \ldots, X_k) = H(X_1) + H(X_2 \mid X_1) + H(X_3$$

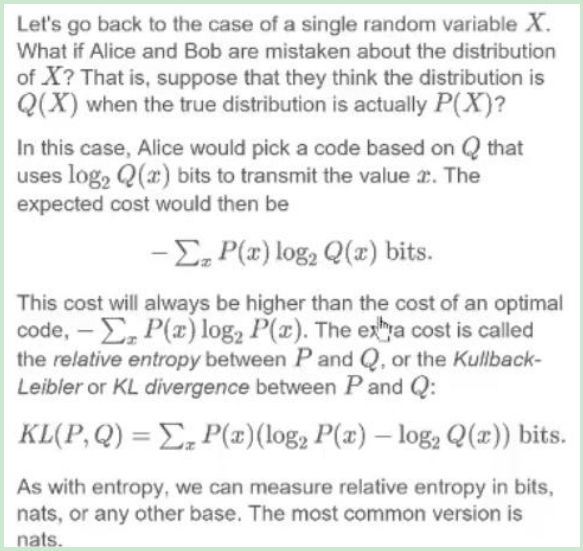
The joint entropy also satisfies

$$H(X,Y) \leq H(X) + H(Y)$$

with equality iff X and Y are independent. In other words, transmitting X and Y separately always costs at least as much as transmitting them together; we can save by transmitting them together iff there is some statistical dependence between them.

分别传送 2 个 message，肯定比 send 一个会长

5 Relative entropy：



Let's go back to the case of a single random variable X. What if Alice and Bob are mistaken about the distribution of X? That is, suppose that they think the distribution is $Q(X)$ when the true distribution is actually $P(X)$?

In this case, Alice would pick a code based on Q that uses $\log_2 Q(x)$ bits to transmit the value x. The expected cost would then be

$$-\sum_x P(x) \log_2 Q(x) \text{ bits.}$$

This cost will always be higher than the cost of an optimal code, $-\sum_x P(x) \log_2 P(x)$. The extra cost is called the *relative entropy* between P and Q, or the *Kullback-Leibler* or *KL divergence* between P and Q:

$$KL(P,Q) = \sum_x P(x)(\log_2 P(x) - \log_2 Q(x)) \text{ bits.}$$

As with entropy, we can measure relative entropy in bits, nats, or any other base. The most common version is nats.

每个信息的实际长度是$-\log Q(x)$，乘以实际概率就是期望长度

得到的实际 enrtopy：$-\sum_x P(x) \log_2 Q(x)$ bits.

会比理想的要长，差就是 relative entropy KL 散度

KL(P,Q)一定大于 0，代表需要付出的额外代价信息

Relative entropy is asymmetric:
$KL(P,Q) \neq KL(Q,P)$ in general. An intuitive description of the asymmetry is that KL divergence grows quickly if P places high probability on outcomes that Q considers unlikely (but not vice versa). Mathematically, this happens because $-p \log_2 q$ grows quickly as $q \to 0$ but shrinks slowly as $p \to 0$. Another way to think of this property is that, if we assign a very long code word to an event that is actually frequent, we will pay a lot.

KL(P,Q)代表不对称 P 是真实，Q 是假

6 Entropy over time

Often in ML, we have a parameter of interest — say θ. We start off with a prior distribution $P(\theta)$, and try to learn about θ from data. In this case we are playing the role of Bob above, and Nature plays the role of Alice: information flows from Nature to us via a sequence of messages (the training examples).

In more detail, over time we see some observations $X_1, X_2, \ldots$, and our distribution evolves as

$$P(\theta), P(\theta \mid X_1), P(\theta \mid X_1, X_2), \ldots$$

Each example changes the amount of information we have about θ, so the entropy of our distribution also evolves: it is

$$H(\theta), H(\theta \mid X_1), H(\theta \mid X_1, X_2), \ldots .$$

theta 的后验概率一直在变，随着 example train
因为分布变了，entropy 也会变

At the beginning we have high entropy (little information), and over time our entropy tends to decrease (our information tends to increase). On average, the decrease will be monotone. But interestingly, it is only monotone on average: if we receive a surprising observation, our entropy may increase.

最开始不确定性最大，enthopy 最大

并不是一直绝对的下降，而是 on average 下降

Model--discrete variable

1model

1. Models

The first step in almost any machine learning problem is to write down a model: a list of the random variables we are interested in, and a structure such as a joint probability distribution that describes how they relate to one another. More precisely, we usually specify a *model class*: our model will have one or more free parameters that we need to estimate using methods like maximum likelihood.

model：就是一些随机变量+他们的分布，最常见是**联合分布**
这些 model 有一些自由参数，我们需要对这些参数进行估计

We'll start with models for a single discrete variable (like the probability for a biased coin, or the number of people who participate in a survey). Later we'll talk about continuous random variables and how to relate multiple random variables to one another.

we get sample from some distribution，这些 distribution 可能在一类家族分布里，然后我们找到具体的那个分布

2 discrete 分布

2.1 分类分布(自由度)

3. Categorical distribution

The simplest discrete distribution is *categorical*: the random variable takes one of k values, and we specify the distribution by listing the probability of each of these outcomes. For example, a coin can come up heads or tails, and so we can specify its distribution by saying

$$P(H) = 0.7 \quad P(T) = 0.3$$

This particular special case (with only two possible outcomes) is called a *Bernoulli* distribution. A more

Bernoulli

If there are k possible outcomes, we have $k-1$ degrees of freedom in a categorical distribution: the last outcome's probability is determined by the requirement that all the probabilities have to sum to 1.

Bernoulli distribution 就是 2 分类分布

参数自由度，因为 sum=1，一个是固定的，其他 k-1 才是真正变的

Multinormial(多项分布)

核心思想

4. Multinomial

If we sample from a categorical distribution n times, and count the number of repetitions of each outcome, we get a *multinomial* distribution. We specify a multinomial distribution by giving the probability of each outcome (i.e., the parameters of the underlying categorical distribution) together with the total number of draws.

A sample from a multinomial distribution is a vector of nonnegative counts: one count for each possible outcome. For example, if we flipped a biased coin 20 times, we could get the sample $(17, 3)$, meaning 17 heads and 3 tails. (This case, where there are only two outcomes, is called a *binomial* distribution.)

核心思想：sample 多次，我们只关心最后每一类的出现频率“”
也就是从 multinomial 里 sample，结果是类长度的 vector，每个元素是该类出现的频率

二项分布推导(联合概率(考虑顺序和不考虑顺序的情况)重要)

最简单的二项分布推导：sample 伯努利分布

重复 5 次，出现两次 A 的情况有很多种！！！很多排列

一次实验的结果：$P(AA\bar{A}A\bar{A})$ ，但任意一种结果的概率相同都是

$P(A)^3P(\bar{A})^2$

对于联合概率多数是不考虑顺序(只看结果)，多个变量同时发生，

没有顺序

p(A,B)= P(A)P(B|A)=P(B)P(A|B)= P(B,A) 只要同时出现，任意顺序分解，只关心结果，不关心过程

而，这里联合概率需要考虑顺序，有明确实验先后顺序

如果事件描述是A发生两次，BC各发生一次，这就是个复合事件就该事件包括：AABC BCAA ABAC ... 需要考虑顺序，因为独立，所有每一个事件的概率一样P(A)P(A)P(B)P(C)

$C_4^2C_2^1$种可能相加(分别给ABC选位置,相同的元素要一次性选A两个人要一次性选)

例 1.1 设某事件 A 在一次试验中发生的概率为 p. 现把这个试验独立地重复 n 次，以 X 记 A 在这 n 次试验中发生的次数，则 X 可取 $0,1,\cdots,n$ 等值. 为确定其概率分布，考虑事件 $\{X=i\}$. 要这个事件发生，必须在这 n 次试验的原始记录

$$A A \bar{A} A \cdots \bar{A} A \bar{A}$$

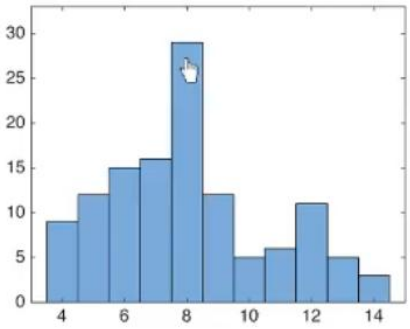
中，有 i 个 A，$n-i$ 个 $\bar{A}$，每个 A 有概率 p，而每个 $\bar{A}$ 有概率 $1-p$. 又“n 次试验独立”表示在每次试验中 A 出现与否与其他次试验的结果独立. 因此，由概率乘法定理给出：每个这样的原始结果序列发生的概率为 $p^i(1-p)^{n-i}$. 又因为在 n 个位置中 A 可以占据任何 i 个位置，故一共有 $\binom{n}{i}$ 种. 由此得出

$$p_i = b(i;n,p) = \binom{n}{i} p^i (1-p)^{n-i} \quad (i = 0,1,\cdots,n). \tag{1.6}$$

X 所遵从的概率分布(1.6)称为二项分布，并常记为 $B(n,p)$. 以后，当随机变量 X 服从某种分布 F 时，我们用 $X \sim F$ 来表达这一点. 例如，X 服从二项分布就记为 $X \sim B(n,p)$.

多项分布(排列组合)

We can display a single sample from a multinomial random variable as a *histogram*. For example, suppose we randomly choose 123 U.S. states (uniformly, with replacement), and count the number of characters in each one's name. The outcome of this experiment is a multinomial random variable, with $n = 123$ and probabilities determined by the table of state name lengths. Here is one possible outcome:



We can read off, for example, that there were more 8-letter states in our sample than any other kind: 29 of the 123 trials. In fact, 12 of the 50 states (24% of them) have 8 letters in their names, so $P(8) = 0.24$; the observed value 29/123 is 0.2358, which is very close to the true $P(8)$.

histgram 柱状图

关于 name 长度的频率分布

The probability of the sample $x = (x_1, x_2, \ldots, x_k)$ given parameters n (the number of trials) and $p_1 \ldots p_k$ (the probability of each outcome) is:

$$\frac{n!}{x_1!\,x_2!\,\ldots\,x_k!}\,p_1^{x_1}p_2^{x_2}\cdots p_k^{x_k}.$$

The terms $p_i^{x_i}$ measure the probability of getting x_i copies of outcome i. The factorial terms in front correct for the fact that we ignore the order in which the outcomes appear.

一共发生 n 次，**让每一类选择其发生的位置：** $C_n^{x1}C_{n-x1}^{x2}C_{n-x1-x2}^{x3}$...得到所有可能的结果，每个结果的概率是 $p^{x1}p^{x2}p^{x3}$...

（2）排列：从 n 个不同元素中，任取 m（m≤n）个元素，按照一定的**顺序**排成一列，叫做从n个不同元素中取出m个元素的一个排列，所有排列的个数记为A_n^m.

$$A_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!}\ (m \le n)$$

规定：$0! = 1$

（3）组合：从 n 个不同元素中任取 m（m≤n）个元素并组成一组，叫做从 n 个不同元素中取出m个元素的一个组合，所有组合个数记为C_n^m.

$$C_n^m = \frac{A_n^m}{A_m^m} = \frac{n(n-1)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}$$

规定：$C_n^0 = 1$

（4）组合数性质：

$$C_n^m = C_n^{n-m},\ C_n^m + C_n^{m-1} = C_{n+1}^m,\ C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$$

排列是先组合(选出)再小排列(2 步)： $A_n^m = C_n^m A_m^m$　排列包括一个选的(组合)过程，如果A_m^m，C_m^m=1 没得选，就纯排列

组合=排列/小排列

破桑 poisson distr

和 Multinomial 的关系：multinormial 是 fixed trials

A sample from a multinomial distribution is a vector of counts: how often did a particular event happen over the course of a fixed number of trials. For example, the events could be how often a coin came up heads, or how often a state's name had 8 letters.

possion 是 fixed time

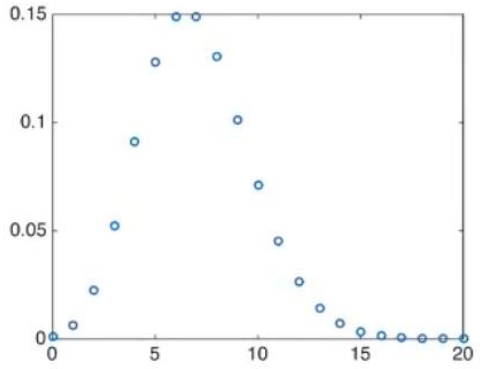
A related situation is when we don't fix the number of trials ahead of time. Instead the events can happen at any time; we fix the total amount of time we observe, and count how often an event happens in this time period. If the events happen independently at a fixed rate, then the result is called a *Poisson* distribution. We could use a Poisson distribution to model the number of customers who arrive at a restaurant between noon and 1pm; or the number of times a neuron fires in 100ms; or the number of photons that fall onto an image pixel during one frame of a movie.

The probability of observing x events under a Poisson distribution is

$$P(X = x) = \lambda^x e^{-\lambda}/x!.$$

每个时间段，某个事件发生的次数(随机变量)的概率

The Poisson distribution has one parameter, λ: the rate. That is, λ tells us how many events we expect to see on average during our observation period. Here's an example of a Poisson distribution with $\lambda = 7$.



The axes are x and $P(X = x)$.

lamda 是事件时间段内**发生的 average 频率.**为 7 时，则 7 发生概率最大，参数需要我们来 estimate

Feature and moment

feature function

A very general way to describe a discrete distribution is via numeric features of the possible outcomes. For example, if the outcomes are integers X, we could use X and X^2 as features; or if the outcomes are U.S.

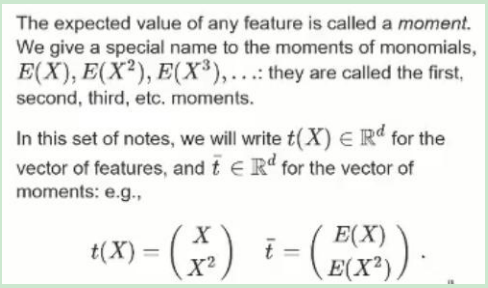
moment

feature 的期望

If we fix a set of outcomes and a list of features, we can (at least partially) describe a probability distribution over the outcomes by giving the expected value of each feature. For example, here are two different distributions over the integers 1..10:

$$P(x) = \frac{x}{55} \qquad Q(x) = \frac{1}{10}.$$

The first one has $E(X) = 7$ and $E(X^2) = 55$. The second has $E(X) = 5.5$ and $E(X^2) = 38.5$.

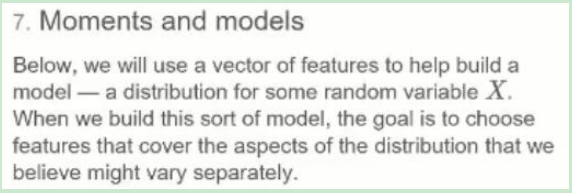

The expected value of any feature is called a *moment*. We give a special name to the moments of monomials, $E(X), E(X^2), E(X^3), \ldots$: they are called the first, second, third, etc. moments.

In this set of notes, we will write $t(X) \in \mathbb{R}^d$ for the vector of features, and $\bar{t} \in \mathbb{R}^d$ for the vector of moments: e.g.,

$$t(X) = \begin{pmatrix} X \\ X^2 \end{pmatrix} \quad \bar{t} = \begin{pmatrix} E(X) \\ E(X^2) \end{pmatrix}.$$

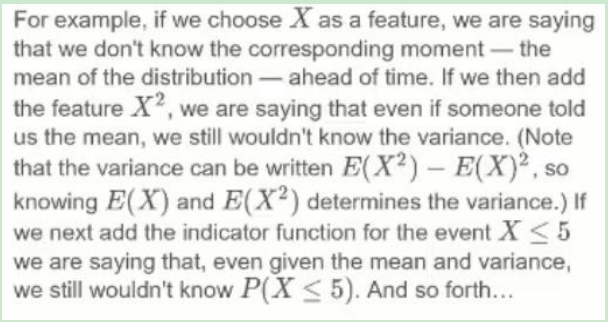
t 是 feature function

t ba 是 moment vector (期望)每个 feature 的期望，是已知的，根据已有数据变换后计算(ba 是估计值)


7. Moments and models

Below, we will use a vector of features to help build a model — a distribution for some random variable X. When we build this sort of model, the goal is to choose features that cover the aspects of the distribution that we believe might vary separately.

choose the feature 是为了更好的 build model,更好的描述这个 model 的特征(model 就是 X 的 distribution)


For example, if we choose X as a feature, we are saying that we don't know the corresponding moment — the mean of the distribution — ahead of time. If we then add the feature X^2, we are saying that even if someone told us the mean, we still wouldn't know the variance. (Note that the variance can be written $E(X^2) - E(X)^2$, so knowing $E(X)$ and $E(X^2)$ determines the variance.) If we next add the indicator function for the event $X \leq 5$ we are saying that, even given the mean and variance, we still wouldn't know $P(X \leq 5)$. And so forth...

通过增加 feature，为了使得 momentom 更好的描述 model distribution 的参数(信息)，build model 其实就是找到 model 的参数

Mximun entropy

我们有了 sample 数据(也已知我们用的 feature function)，我们计算这些 observed feature 的期望(moment)，我们想构建一个

distribution 的参数就是这些 moment

对于 data feature 建模时，我们想将该数据的期望 t ba 作为模型参数。 t ba 已知

根据我们已有的数据 feature 和 momentom，我们去 pick 一个分布(特定参数的分布)

When we build a model based on a vector of features, we'd like to use the corresponding vector of moments $\bar{t}$ as a parameter. That is, we fix a set of outcomes and a list of features ahead of time; then to pick a distribution over the outcomes, we'd like to just specify $\bar{t}$. (This is called the *mean parameterization* of a family of distributions, since the parameter $\bar{t}$ is the mean of the feature vector.) But, in general, there could be many probability distributions that yield the same vector of moments. Which one should we pick?

有这些参数 moment t ba 的模型分布有无数个，我们选择哪一个(但不同的分布可能生成有相同的模型参数)？选择最大信息熵的模型，这使得我们对 distribution 的 assumption 最少(H(distri)越大，需要传递的信息越大，也就是 alice 和 bob 如果对 distribution 越不了解，需要传递的信息越多，信息熵越大！)

A simple rule is to pick the one that assumes the least information beyond what is given. That is, we pick the (unique) distribution with the highest entropy, among the ones that match the given moment vector $\bar{t}$. This is called the *maximum entropy* or *maxent* distribution for $\bar{t}$.

选择让 entropy 最大的分布

maximum entropy distribution(maxent distribution)最大熵模型

The maxent distribution is the one that *maximizes* the expected number of bits in Alice's message, given that they both know $E(t(X)) = \bar{t}$. So, it is the distribution for which Alice has the largest possible amount of extra information compared to Bob. Equivalently, it is the distribution that assumes as little as possible about X: the less we assume about X, the more information we should expect Alice to have to transmit.

在两个人都知道 E(t(X))=t ba 的条件下，也就是该分布的某些期望已知(期望，方差已知)的条件下，其实就是将该未知参数分布的 X 进行 feature 变化后的 moment 等于我们已知数据计算的 momentum！！

对模型的假设越少越好，假设越少，信息熵越大，Alice 需要传递的信息最长 question?为啥这样选

Minimum relative entropy

Often, we'll actually use a slightly more complicated rule: we'll pick the distribution that minimizes relative entropy (KL divergence) from a fixed distribution P_0, called the *base* distribution. This rule is called *minimum relative entropy*. If we pick P_0 to be uniform, we recover maxent.

也就是 P0 是假的分布

当 P0 是 uniform 时，熵最大，如果 minimize 差值，其实结果就是 maxent，肯定会选 entropy 最大的 true p，才能使得 relative entropy 最小

In minimum relative entropy, the role of the base distribution is to let us favor some values of X over others: e.g., we might prefer small numbers, or states close to Pennsylvania, or histograms with a peak near 0.5.

P0 是为了对 X 进行一些约束

A good example of where we need a non-uniform base distribution is to compensate for differences in discretization: if our discretization is denser or less dense than we'd like in some areas, we can compensate by reducing or increasing P_0 in those areas, respectively. (Of course we could also change the discretization itself, but we might have other constraints that limit our choice of how to discretize.)

question

Compute the maxent distribution

In some cases, we can solve explicitly for the maxent or minimum relative entropy distribution. In this section we'll describe the procedure for doing so. Unfortunately the proofs of a number of the facts we'll need are outside the scope of this course, so we'll just state them without proof.

maxent 的 solution：就是我们想得到的最终的分布函数 $P_\theta(x)$

其实就是指数家族分布 $P_\theta(x)$

给定 P0 base distri+t(x) 来求 theta

It turns out that the solution to the maxent or minimum relative entropy problem always has the following form:

$$P_\theta(x) = P_0(x)\exp(t(x)\cdot\theta - g(\theta)).$$

Here θ is a vector of parameters that we will solve for, and g is a function that ensures $\sum_x P_\theta(x) = 1$ for each θ. This form is called an *exponential family distribution*; θ is called the *natural* parameter vector, to contrast with the mean parameter vector $\bar{t}$.

$P_\theta(x)$ 就是我们要找的分布也就是 maxent 分布，形式如上

P_0 是 base distribution

t(x)是 每个 feature vector

theta 是自然参数，和我们的 t ba 对应

1 先要求 g(theta)，2 然后再求 theta

1 先求 g(theta)，是用来保证 $\sum P_\theta(x)$ =1

$$\sum_x P_\theta(x) = 1$$
$$1 = \sum_x P_0(x)\exp(t(x)\cdot\theta - g(\theta))$$
$$= e^{-g(\theta)} \sum_x P_0(x)\exp(t(x)\cdot\theta)$$
$$e^{g(\theta)} = \sum_x P_0(x)\exp(t(x)\cdot\theta)$$

最后两边取 log 得到 g(theta)

指数对数性质关系总结

$a^b = c$ b 就是指数，c 就是对数　　互为反函数

$a^x = y$ 自变量是指数，y 是对数

$\log_a^x = y$　自变量是对数，y 是指数

$a^{\log_a^c}$ =c　这是一种重要变换

$\log_a^{a^b} = b$　只得到最上面的

There are two key difficulties we need to solve to get from the above expression to an explicit representation of P_θ. (We'll see some examples below.) The first difficulty is that we need to derive an expression for the function g. We can see that picking

$$g(\theta) = \ln(\sum_x P_0(x) \exp(t(x) \cdot \theta))$$

makes P_θ satisfy the requirement $\sum_x P_\theta(x) = 1$: we can verify this fact by factoring out $\exp(-g(\theta))$ from

$$\sum_x P_0(x) \exp(t(x) \cdot \theta - g(\theta)) .$$

So, deriving an expression for g means simplifying $\ln(\sum_x P_0(x) \exp(t(x) \cdot \theta))$.

2 然后再求 theta，　我们的假设是 E(t(X))=t ba

$\sum P_\theta(X)t(X) = tba$

Once we know g, the second difficulty is to find the natural parameter vector θ. We determine θ from the requirement that $E(t(X)) = \bar{t}$. It turns out that this requirement is equivalent to

$$g'(\theta) = \bar{t} .$$

其实就是将该未知参数分布的 X 进行 feature 变化后的 moment 等于我们已知数据计算的 momentum！！

结论是：g 的导数等于 t

证明：

13. Facts about exponential families

To help understand exponential families, it helps to derive some simple facts about them. First, note that the notation P_θ here is consistent with the use of P_0 for the base distribution: if $\theta = 0$ then $\exp(t(x) \cdot \theta - g(\theta))$ is constant with respect to x, so P_θ is exactly the base distribution.

$$P_\theta(x) = P_0(x) \exp(t(x) \cdot \theta - g(\theta)) .$$

1 当 theta=0， $p_\theta = p_0$

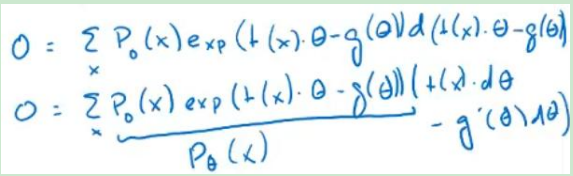
t(x)theta=0　左边 p0 右边 p0 则 exp(-g)一定是 1

2 证明 $tba = \sum P_\theta(X)t(X) = g'(\theta)$

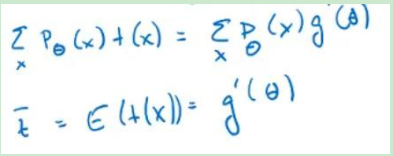
先用 sum=1，然后两边对 theta 取导数(微分),得出 g 的导数形式

应用 linearity 和 chain rule

linearity: $\sum$ 是 add，p(x)是 multiple



左边是 $p_\theta(X)$,移项得：



又右边 $\sum_x P_\theta(x)g'(\theta)$　后面的 g’不依赖 x，sum=1 提出 g’因此 $g'(\theta)\sum_x P_\theta(x)$ =g’

左边 t(x)期望，刚好证明！

$$g'(\theta) = \bar{t} .$$

从而求出 theta

Second, we can see that, if we pick θ so that $g'(\theta) = \bar{t}$, then the distribution P_θ has the desired moments:

$$\begin{aligned}
1 &= \textstyle\sum_x P_\theta(x) \\
&= \textstyle\sum_x P_0(x) \exp(t(x) \cdot \theta - g(\theta)) \\
0 &= d \textstyle\sum_x P_0(x) \exp(t(x) \cdot \theta - g(\theta)) \\
&= \textstyle\sum_x P_0(x) \exp(t(x) \cdot \theta - g(\theta))(t(x) - g'(\theta))\, d\theta \\
&= \textstyle\sum_x P_\theta(x)(t(x) - g'(\theta))\, d\theta \\
&= [E(t(X)) - g'(\theta)]\, d\theta \\
g'(\theta) &= E(t(X))
\end{aligned}$$

In fact, g is always smooth (so that g' exists), and g' is always an invertible function (so that we can always solve for θ in the above equation).

To be more precise, consider the set T of achievable values of $\bar{t}$: that is, $\bar{t} \in T$ if there exists any probability distribution over our set of outcomes that has $E(t(X)) = \bar{t}$. Then we can solve for θ as long as $\bar{t}$ is in the interior of T; if $\bar{t}$ is on the boundary of T, we can get as close as we want, but we may not be able to hit $\bar{t}$ exactly except in the limit of large θ.

Exponential family

11. Exponential family examples

We already saw a simple example of an exponential family: the Bernoulli family of distributions. If our base distribution is the uniform distribution over $\{0, 1\}$ and our feature vector is just the single feature X, then the corresponding moment is the probability of seeing 1, and the maximum entropy distribution is a Bernoulli.

base distribution 是 0,1 的 uniform 分布
feature 只是 X，moment 是出现 1 的期望(概率)
则 maxient 就是伯努利分布

$$P_\theta(x) = P_0(x) \exp(t(x) \cdot \theta - g(\theta)).$$

伯努利分布

同样的，Bernoulli分布 $B(1,\pi)$ 也属于指数族，这是因为它的概率密度函数

$$f(y \mid \pi) = \pi^y (1-\pi)^{1-y}, y \in \{0,1\},$$

可以化为

$$f(y \mid \pi) = \exp\left\{y \log \frac{\pi}{1-\pi} + \log(1-\pi)\right\},$$

再令 $\theta = \log \frac{\pi}{1-\pi}$ 可得

$$f(y \mid \theta) = \exp\left\{y\theta - \log\left(1 + e^\theta\right)\right\},$$

可以化成指数分布形式

A slightly more interesting example is the Poisson family. If our feature vector is just the single feature X and our base distribution is proportional to $1/x!$ over the nonnegative integers, then the minimum relative entropy distribution is a Poisson.

base distribution proportional to 1/x 阶乘

Many other common families of distributions are exponential families. It is an interesting exercise to try to put them in the form given above — e.g., try this exercise for the negative binomial.

More interestingly, we can build examples by choosing feature vectors as described above. For example, suppose that we are building a model of the distribution of a particular type of insect in the area around Pittsburgh. We could divide this area into plots of land, and consider each insect sighting as a sample from a categorical distribution over plots. If we have a lot of plots, we don't want to learn a separate probability for each one. So instead we could assign features to the plots: average temperature and rainfall, variance in temperature and rainfall, seasonal variation in temperature and rainfall, etc. We can then construct an exponential family of distributions based on these features, and fit the parameters of this family by solving for the natural parameter vector θ.

也可以 build 自己的 distribution

总结：

MLE and Maxent EXAMPEL

12. Maximum likelihood

Earlier, we covered the maximum likelihood method for fitting a distribution. Just above, we covered exponential families and how to choose a distribution by maximum entropy or minimum relative entropy. We might ask, which of these two procedures for choosing a distribution is better: maximum likelihood or minumum relative entropy?

In fact, it turns out that they are the same: for an exponential family of distributions with base distribution P_0 and features $t(X)$, we get the *same* value of θ by either:

- maximizing the likelihood of a sequence of observations $X_1, X_2, \ldots, X_n$, or
- computing the observed mean feature vector $\bar{t} = \frac{1}{n}\sum_i t(X_i)$, and picking the distribution with minimum relative entropy to P_0 given the moment vector $\bar{t}$.

MLE 和 max entropy 结果一样，得到一样的 distribution

example：

破森分布　MLE：通过选择参数，让我们 observed data 的概率最大

1 先 observe sample：两个 examples（我们的 data）(已知)

2 而且我们知道都是破森分布，找该分布的参数

想求出分布的参数，从而得出该分布

$$x_1, x_2 \in \{0, 1, 2 \dots\}$$
$$P(x_i = x) = \lambda^x e^{-\lambda} / x!$$
$$P(x_1, x_2) = P(x_1) P(x_2)$$

因为独立，data:x1 x2 同时发生是概率相乘(不考虑顺序)

MLE：　假设我们观察到了两个值，想知道分布的参数 lambda 是多少

我们目标是使得联合概率 p(x1,x2)最大，先取 log，转化成和再求导

然后带入分布：

$$\ln P = \ln P(x_1) + \ln P(x_2)$$
$$= x_1 \ln \lambda - \lambda - \ln x_1! + x_2 \ln \lambda - \lambda - \ln x_2!$$

对 lambda 求导(微分)：

$$d \ln P = \frac{x_1}{\lambda} d\lambda - d\lambda + \frac{x_2}{\lambda} d\lambda - d\lambda$$

令为 0　question 会是极小点吗？？只要是参数的单调递增函数就没问题

$$\lambda = \frac{x_1 + x_2}{2}$$

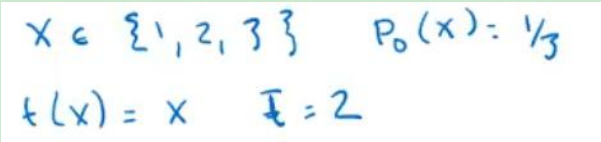
也就是给定我们的观测数据，我们可以得到分布的参数，确定分布

的形式！！

Maximum entropy EXAMPLE

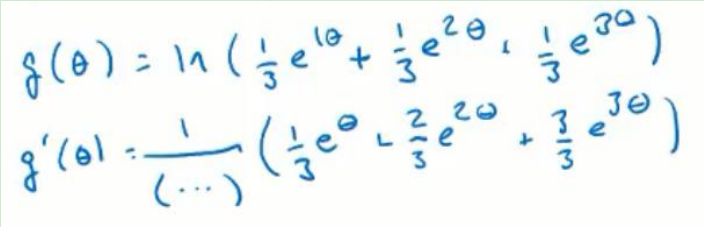
1 choose a moment(from observed data)作为 distribution 的统计量

2 maximize the entropy



P0 是 uniform distribution, t(x)=x

写出 Maximum entropy 的 solution



再求 theta

求导数,并让导数等于我们的 t ba(solve non-linear equation)

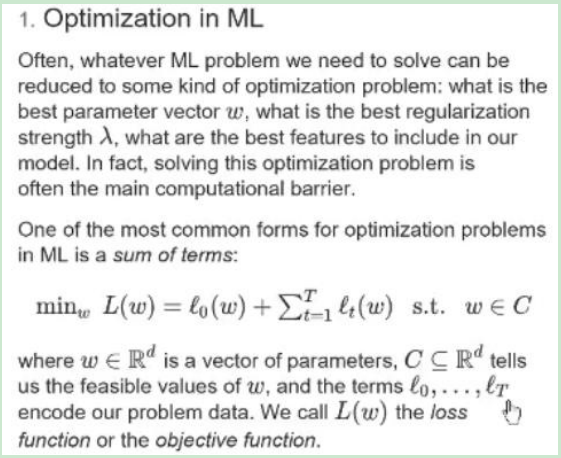
刚好当 theta=0 时，导数等于 t ba

Gradient descent

Cost function 分解

只保证 local optima，除非有些只有 global optima

MLE 和 maximum entropy 都需要 optimize



1. Optimization in ML

Often, whatever ML problem we need to solve can be reduced to some kind of optimization problem: what is the best parameter vector w, what is the best regularization strength λ, what are the best features to include in our model. In fact, solving this optimization problem is often the main computational barrier.

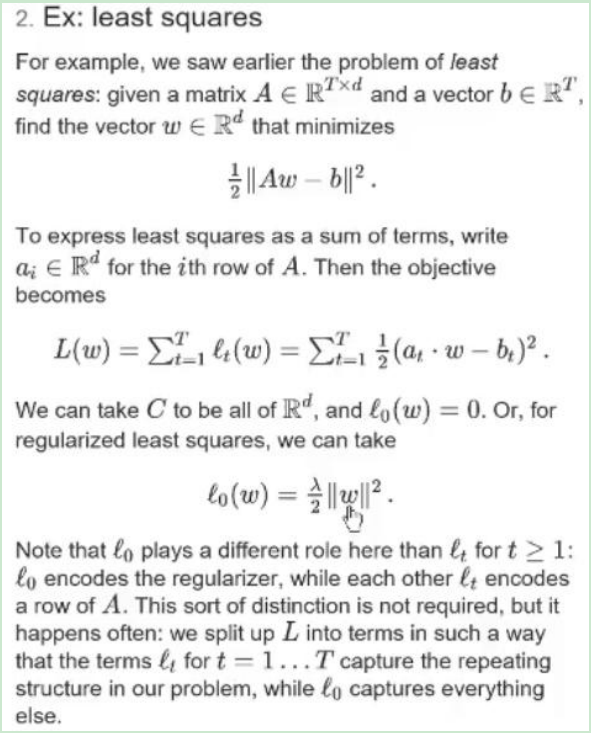
One of the most common forms for optimization problems in ML is a *sum of terms*:

$$\min_w \; L(w) = \ell_0(w) + \sum_{t=1}^{T} \ell_t(w) \quad \text{s.t.} \quad w \in C$$

where $w \in \mathbb{R}^d$ is a vector of parameters, $C \subseteq \mathbb{R}^d$ tells us the feasible values of w, and the terms $\ell_0, \ldots, \ell_T$ encode our problem data. We call $L(w)$ the *loss function* or the *objective function*.

小 L 是将我们的 feature 通过 weight 进行 encode，这个 code 是为了解决特定问题

例子：



2. Ex: least squares

For example, we saw earlier the problem of *least squares*: given a matrix $A \in \mathbb{R}^{T \times d}$ and a vector $b \in \mathbb{R}^T$, find the vector $w \in \mathbb{R}^d$ that minimizes

$$\tfrac{1}{2}\|Aw - b\|^2 .$$

To express least squares as a sum of terms, write $a_i \in \mathbb{R}^d$ for the ith row of A. Then the objective becomes

$$L(w) = \sum_{t=1}^{T} \ell_t(w) = \sum_{t=1}^{T} \tfrac{1}{2}(a_t \cdot w - b_t)^2 .$$

We can take C to be all of $\mathbb{R}^d$, and $\ell_0(w) = 0$. Or, for regularized least squares, we can take

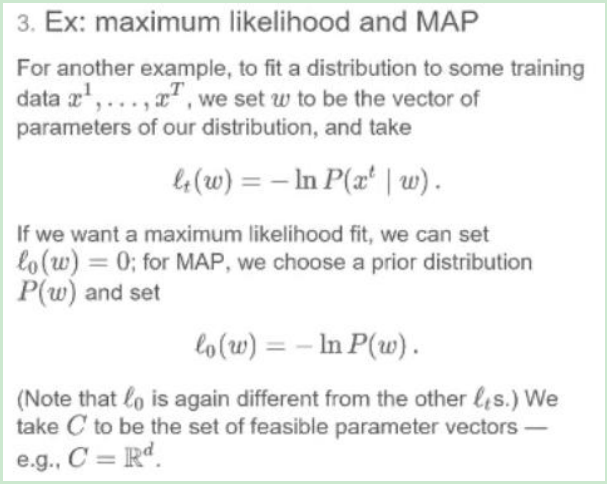
$$\ell_0(w) = \tfrac{\lambda}{2}\|w\|^2 .$$

Note that ℓ_0 plays a different role here than ℓ_t for $t \geq 1$: ℓ_0 encodes the regularizer, while each other ℓ_t encodes a row of A. This sort of distinction is not required, but it happens often: we split up L into terms in such a way that the terms ℓ_t for $t = 1 \ldots T$ capture the repeating structure in our problem, while ℓ_0 captures everything else.

本身 L2 norm 的平方就是平方和的形式，转换成和

Cost function 往往可以分解成小重复的的 loss 情况

MLE 和 MAP



3. Ex: maximum likelihood and MAP

For another example, to fit a distribution to some training data $x^1, \ldots, x^T$, we set w to be the vector of parameters of our distribution, and take

$$\ell_t(w) = -\ln P(x^t \mid w).$$

If we want a maximum likelihood fit, we can set $\ell_0(w) = 0$; for MAP, we choose a prior distribution $P(w)$ and set

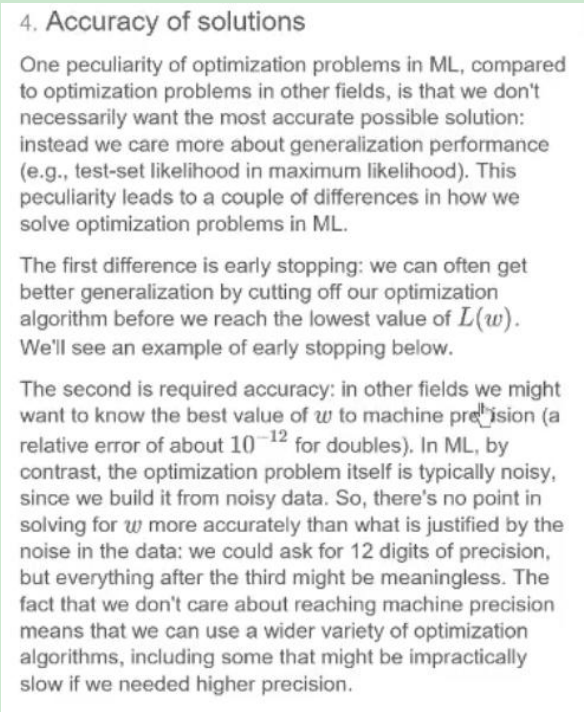
$$\ell_0(w) = -\ln P(w).$$

(Note that ℓ_0 is again different from the other ℓ_ts.) We take C to be the set of feasible parameter vectors — e.g., $C = \mathbb{R}^d$.

上面将联合概率取 log，转换成 SUM

对于 MAP L0 就是先验概率

Accuracy and generalization



4. Accuracy of solutions

One peculiarity of optimization problems in ML, compared to optimization problems in other fields, is that we don't necessarily want the most accurate possible solution: instead we care more about generalization performance (e.g., test-set likelihood in maximum likelihood). This peculiarity leads to a couple of differences in how we solve optimization problems in ML.

The first difference is early stopping: we can often get better generalization by cutting off our optimization algorithm before we reach the lowest value of $L(w)$. We'll see an example of early stopping below.

The second is required accuracy: in other fields we might want to know the best value of w to machine precision (a relative error of about 10^{-12} for doubles). In ML, by contrast, the optimization problem itself is typically noisy, since we build it from noisy data. So, there's no point in solving for w more accurately than what is justified by the noise in the data: we could ask for 12 digits of precision, but everything after the third might be meaningless. The fact that we don't care about reaching machine precision means that we can use a wider variety of optimization algorithms, including some that might be impractically slow if we needed higher precision.

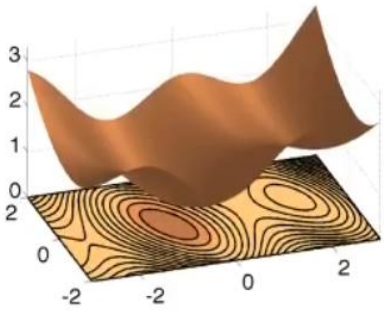
1 solution from training set，我们不关心 train，我们关心 test，我们需要更好的泛化

方法 1：在达到最优之前停止 early stoping，防止过拟合

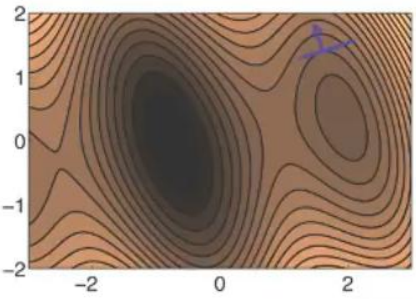
方法 2 data have noise，就算**最优化**也不一定是**我们的结果**

5. Optimization and gradients

Because of the above properties, a lot of useful optimization algorithms for ML are based on a simple fact: the gradient of a function is the direction of instantaneous steepest ascent. That is, the gradient is the direction in which the function locally increases most rapidly. For example, if we look at the following loss function $L(w)$:



and draw its gradient (blue arrow) at a typical point w_0:



朝着 gradient 方向(箭头方向)相反的走

the direction of the gradient. Furthermore, if we look at the hyperplane (blue line) passing through w_0 whose normal vector is the gradient, we can see that $L(w)$ is locally constant if we move along the hyperplane; on one side of the hyperplane, $L(w)$ increases, and on the other side, it decreases.

想象一个平面垂直于 gradient 向量，平面两边分别是使得 L 增大和减小的方向

In other words, the gradient $\nabla L(w_0)$ is normal to the level set $\{w \mid L(w) = L(w_0)\}$. The corresponding hyperplane is tangent to the level set, and it separates local ascent directions from local descent directions. We can explain this behavior by looking at the first-order Taylor expansion of L around w_0:

$$L(w) \approx L(w_0) + (w - w_0) \cdot \nabla L(w_0)\,.$$

This approximation says that $L(w) - L(w_0)$ will be positive exactly when $(w - w_0) \cdot \nabla L(w_0)$ is positive. The hyperplane normal to $\nabla L(w_0)$ and passing through w_0 is the surface

$$\{w \mid (w - w_0) \cdot \nabla L(w_0) = 0\}$$

and so it separates $(w - w_0) \cdot \nabla L(w_0) > 0$ from $(w - w_0) \cdot \nabla L(w_0) < 0$.

一阶泰勒展开：

垂直平面可以表示成 w 的 set(平面就是由参数决定)

我们希望 $L(w)$-$L(w_0)$ 小于 0，也就是从 w0 这一点，我们希望 L 下降

Gradient descent

6. Gradient descent

The interpretation of the gradient as the direction of steepest ascent suggests a simple algorithm: we will decrease $L(w)$ as quickly as possible — at least locally — if we move w in the opposite direction from the gradient,

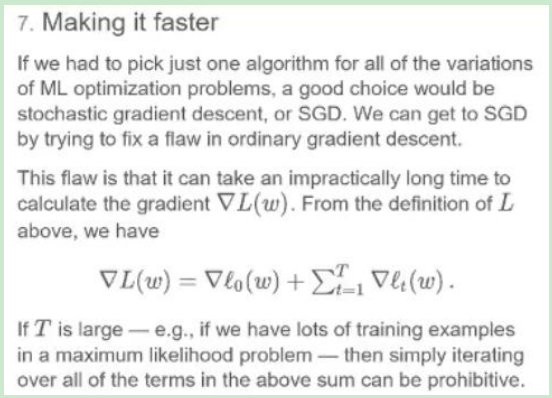
$$w \leftarrow w - \eta \nabla L(w)\,.$$

The *gradient descent* algorithm simply repeats the above update over and over. The parameter $\eta > 0$ is the *learning rate* or the *step size*. We choose η to trade off moving quickly (for which we want large η) vs. making sure that we don't move w so far that $L(w)$ starts to increase again (for which we want small η).

Gradient descent itself is rarely the best optimization algorithm for a given problem. However, a small modification to gradient descent leads to SGD, which can be an excellent choice.

只需让参数都沿其负 gradient 方向 update

Stochastic SGD



7. Making it faster

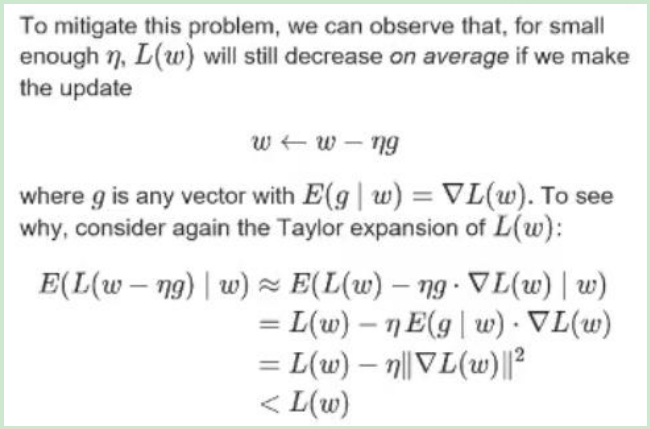
If we had to pick just one algorithm for all of the variations of ML optimization problems, a good choice would be stochastic gradient descent, or SGD. We can get to SGD by trying to fix a flaw in ordinary gradient descent.

This flaw is that it can take an impractically long time to calculate the gradient $\nabla L(w)$. From the definition of L above, we have

$$\nabla L(w) = \nabla \ell_0(w) + \sum_{t=1}^{T} \nabla \ell_t(w) .$$

If T is large — e.g., if we have lots of training examples in a maximum likelihood problem — then simply iterating over all of the terms in the above sum can be prohibitive.

我们求 cost 的 gradient，需要对每个小部分 loss 进行求导，然后相加再平均得到最终的 gradient(每个小 loss 都包含了所有参数)，这样有点慢！！



To mitigate this problem, we can observe that, for small enough η, $L(w)$ will still decrease *on average* if we make the update

$$w \leftarrow w - \eta g$$

where g is any vector with $E(g \mid w) = \nabla L(w)$. To see why, consider again the Taylor expansion of $L(w)$:

$$\begin{aligned} E(L(w - \eta g) \mid w) &\approx E(L(w) - \eta g \cdot \nabla L(w) \mid w) \\ &= L(w) - \eta E(g \mid w) \cdot \nabla L(w) \\ &= L(w) - \eta \|\nabla L(w)\|^2 \\ &< L(w) \end{aligned}$$

g 是总体 gradient 均值 $\nabla L(w)$ 的其中一个，并且 g 的期望是 $\nabla L(w)$

L 也会 decrease，只更新 g 的话

证明：

近似 gradient 是 g，E(g)=总体的 gradient，**完全可以忽略条件 w**(作为条件只是说明 w 是已知，g 是随机变量)

用一阶 taylar 展开：w 是我们固定点，ng 是变动

利用期望的 linearity，L(w)，$\nabla L(w)$ 都是常数，直接对 g 求期望

新的 L 一定小于 L(w),一定会下降

下降会慢一点，因为近似 gradient 有 noise，noise 也有好处，可以 bounce off the local optima

So, if we can find such a vector g more efficiently than evaluating $\nabla L(w)$, we can update w more often than we could with gradient descent. Of course, each update may not be as effective at reducing $L(w)$, since we're introducing noise into our update direction; but the faster updates can more than make up for the loss in accuracy. (And, the noise can sometimes even be beneficial, since it introduces a bit of random exploration.)

符合上面条件的 g 如何获得？

8. Stochastic gradient samples

Fortunately, it turns out to be easy to find such a vector g. All we need to do is choose a single term randomly from the sum, and scale by the number of terms.

Suppose for now that $\ell_0(w) = 0$ for all w. Sample the index i uniformly from the set $\{1, \ldots, T\}$, and take $g = T\nabla\ell_i(w)$. This choice of g yields the desired property: we have

$$E(\nabla\ell_i(w) \mid w) = \frac{1}{T}\sum_{t=1}^{T} \nabla\ell_t(w)\,,$$

since the expectation of any random variable is the sum, over all possible outcomes (integers $1 \ldots T$), of the probability of that outcome ($\frac{1}{T}$) times the value under that outcome ($\nabla\ell_t(w)$). Therefore,

$$E(g \mid w) = \nabla L(w)\,.$$

The result is the stochastic gradient descent algorithm: repeatedly sample i uniformly from $\{1, \ldots, T\}$ and update

$$w \leftarrow w - \eta T\nabla\ell_i(w)\,.$$

随机选一个 single term(loss function),然后对 w 求导，再乘以 scalar T,作为 g，对 g 求和再除以 T(对 g 求平均)结果就是 Cost 的 gradient：E(g)= Cost gradient

可以几乎应用在任何优化场景，就是慢！！！

The result is the stochastic gradient descent algorithm: repeatedly sample i uniformly from $\{1, \ldots, T\}$ and update

$$w \leftarrow w - \eta T\nabla\ell_i(w)\,.$$

To handle $\ell_0 \neq 0$, we can do something only slightly more complicated: again sample an index i uniformly from $\{1 \ldots T\}$, but now take

$$g = \nabla\ell_0(w) + T\nabla\ell_i(w)\,.$$

L0 不参与 sample，每次都加在 g 上！！

我们往往会去掉 T scalar，将其同一放在 learning rate

An alternate convention that you may see is to fold the factor T into the learning rate η, so that the gradient descent update becomes

$$w \leftarrow w - \eta \nabla \ell_i(w)$$

or

$$w \leftarrow w - \eta \left(\frac{1}{T} \nabla \ell_0(w) + \nabla \ell_i(w) \right) .$$

This convention leads to the same algorithm, but the learning rate is scaled differently.

L0 项 需要提出 T factor，保持和其他项的倍数

parameter tuning

9. Parameter tuning

There are two main parameters that we need to tune in SGD: the learning rate η, and the total number of iterations. In fact, we don't just need to select a single η: it turns out that we would like a larger learning rate near the beginning of SGD, and a smaller one near the end.

我们希望一开始 lr 大，后面变小

因为距离 optim 近的时候 step 要小，不然会 miss out

For many problems, such as maximum likelihood, the most common way to tune SGD is via a holdout set: we split our loss function terms randomly into two sets, called the *training set* and the *holdout set*. (In maximum likelihood, the loss function terms correspond to samples from the target distribution, so splitting the terms is the same as splitting our data set.)

通过 holdout set 来调节 lr，split loss function！！

不同的 lr 在 hold out 上的表现，也就是看 holdout set 的 cost 值，降低 lr，直到 hold out cost 会更低，直到不变

Given the split, we run SGD on the training set. We start with a large value of η, and monitor our error $L_{\text{ho}}(w)$ on the holdout set. When $L_{\text{ho}}(w)$ stops decreasing on average, we reduce our learning rate: say, $\eta \leftarrow \eta/2$. And, when $L_{\text{ho}}(w)$ stops decreasing on average even after we reduce η, we stop SGD, and return the value of w that scored the best on $L_{\text{ho}}(w)$. (This is the early stopping method mentioned above.)

计算 hold out 的 cost 计算量大，不要每一次 iteration 计算一次

Because the SGD updates are noisy, we expect that $L_{\text{ho}}(w)$ will fluctuate instead of decreasing monotonically. Additionally, computing $L_{\text{ho}}(w)$ is expensive compared to an iteration of SGD. So, we don't test for a decrease in $L_{\text{ho}}(w)$ on every iteration; instead, every Δ iterations, we compute $L_{\text{ho}}(w)$, and compare its value to the last time we computed it, Δ iterations ago. The parameter Δ should be large compared to the size of the holdout set, so that the cost of computing $L_{\text{ho}}(w)$ is small compared to the cost of Δ iterations of SGD.

Minibatches

Instead of looking at a single term $\nabla \ell_i(w)$ at a time, we can sample B of them at once: e.g., in the case $\ell_0 = 0$,

$$g = \frac{T}{B} \sum_{j=1}^{B} \nabla \ell_{i_j}(w)$$

where the indices i_j are selected uniformly from $\{1, \ldots, T\}$. It's common either to select the i_js independently, or to select them *without replacement*: i.e., we enforce $i_j \neq i_k$ for $j \neq k$.

sample B 个 term(minibatch) ，可以独立单个选取，也可以一次性选取

Such a group of B samples is called a *minibatch*. Minibatches can help average out noise in our gradient samples, reducing the variance of our gradient estimate g and allowing us to use a larger learning rate η.

可以降低 noise，减少了 g 的 variance，可以用更大的 lr

sort.) For another example, we can calculate statistics of our gradient samples within each minibatch; we can use this information to help adjust parameters such as the learning rate.

Conditiona number

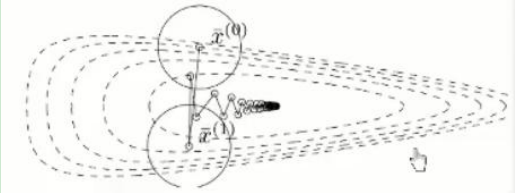
11. Condition number

The single biggest problem with gradient descent and SGD in practice is slow convergence: these algorithms are highly sensitive to the *condition number* of an optimization problem, and bad conditioning (high condition number) can slow down convergence substantially.

For a convex loss function $L(w)$, the condition number at w is defined as the ratio between the largest and smallest eigenvalue of the Hessian matrix $\nabla^2 L(w)$. The condition number of a problem is defined as the largest condition number at any point w. (For a non-convex problem, we can pick a neighborhood where the problem is locally convex, and define the condition number within that neighborhood.)

Because the largest eigenvalue is at least as large as the smallest one, the condition number is always greater than or equal to 1. Informally, good (small) condition numbers mean that the contours of $L(w)$ are close to round:

bad condition：



learning rate 受限于**窄的维度，lr 不能太大**，但对于长的维度，梯度下降又太慢了

2-D 高斯分布例子

12. Ex: 2-d Gaussian MLE

For example, a 2-d Gaussian distribution has parameters $\mu \in \mathbb{R}^2$ and $\Sigma \in \mathbb{R}^{2\times 2}$, corresponding to the mean vector and variance matrix. If we have a batch of T observations from this distribution, $x_t \in \mathbb{R}^2$, we can fit μ and Σ by minimizing

$$L(\mu, \Sigma) = \sum_{t=1}^{T} \ell_t(\mu, \Sigma)$$

where

$$\ell_t(\mu, \Sigma) = \frac{1}{2}\left(\ln|\Sigma| + (x_t - \mu)^T \Sigma^{-1}(x_t - \mu)\right)$$

is the negative log likelihood for a Gaussian distribution, up to an additive constant. (We'll cover this distribution more soon, but for now, it's enough just to trust this formula.) The notation $|\Sigma|$ means the determinant of the matrix Σ, which we defined in the first mini.

将 MLE 转换成优化问题

每一个 loss 都是关于参数的 encoding，这里是每个数据点的-log likehood(这里是 minimize cost，不是 maxi)，

通过观察数据来得到 2-d 高斯分布,来 fit 2-d 高斯分布的拟合(矩阵微分！！)

一个 loss(likelihood term)如上

In this example we wouldn't run SGD in practice, since it's actually faster to calculate the global minimum of L directly. It's still a good example of how to derive the SGD updates, though.

我们可以直接求出 global minimum，不需要 SGD

For this purpose, we need the gradient of each loss term ℓ_t with respect to μ and Σ. Calculating these gradients is a good way to practice the matrix differential calculus that we covered in the first mini: for μ, we have

$$d\ell_t(\mu, \Sigma) = \mu^T \Sigma^{-1} d\mu - x_t^T \Sigma^{-1} d\mu$$

by expanding out the matrix product, using linearity and the product rule, and collecting terms. (This is similar to the least squares example that we did in the first mini.) The gradient is the coefficient of $d\mu$, which we transpose to get a column vector:

$$\nabla_\mu \ell_t = \Sigma^{-1}(\mu - x_t).$$

只需考虑一个 likehood term 求导

对 u 求导，因为 sigma 是常数，直接忽略第一项，展开第二项

$$\ell_t(\mu, \Sigma) = \tfrac{1}{2}\left(\ln|\Sigma| + (x_t - \mu)^T \Sigma^{-1}(x_t - \mu)\right)$$

$$dl = \frac{1}{2} d(x^T - u^T)(\Sigma^{-1}x - \Sigma^{-1}u) = \frac{1}{2} d(x^T\Sigma^{-1}x - x^T\Sigma^{-1}u - u^T\Sigma^{-1}x + u^T\Sigma^{-1}u) = \frac{1}{2} d(x^T\Sigma^{-1}x - 2x^T\Sigma^{-1}u + u^T\Sigma^{-1}u)$$

用到了 scalar 的转置不变

第一项忽略

$$= -x^T\Sigma^{-1}du + \frac{1}{2}u^T\Sigma^{-1}du + \frac{1}{2}du^T\Sigma^{-1}u$$

注意：$du^T = (du)^T$　linearity　transpose

再用 scalar 的转置不变(且协方差矩阵对称 T 不变)，后面两项合并

$$= u^T \Sigma^{-1} du - x^T \Sigma^{-1} du$$
$$= \Sigma^{-1}(u-x)du$$

gradient 就是; $\Sigma^{-1}(u-x)$

2 对∑求导：

$$\ell_t(\mu, \Sigma) = \tfrac{1}{2}\left(\ln|\Sigma| + (x_t-\mu)^T\Sigma^{-1}(x_t-\mu)\right)$$

ln determinate 的导数：用到了**行列式求导**，以及**逆矩阵求导**，因为都是 scalar， scalar 的 trace 等于本身

$$\frac{1}{2}tr(\Sigma^{-1}d\Sigma) + (x-u)^T d(\Sigma^{-1})(x-u) = \frac{1}{2}tr(\Sigma^{-1}d\Sigma) + tr((x-u)^T d(\Sigma^{-1})(x-u))$$

$$d(\Sigma^{-1}) = \Sigma^{-1}d\Sigma\Sigma^{-1}$$

tr 和 tr 合并，都是 scalar

$$\frac{1}{2}tr(\Sigma^{-1}d\Sigma) + (x-u)^T d(\Sigma^{-1})(x-u) = \frac{1}{2}tr(\Sigma^{-1}d\Sigma) + tr((x-u)^T\Sigma^{-1}d\Sigma\Sigma^{-1}(x-u))$$

tr(ABC)=tr(BCA)=tr(CAB)**可以任意 rotate(把左边的转到右边，移动多个时左边要以总体移动)，但要保持相对顺序，ACB 不对！**

目的把 d sigma 移动到右边

$$= \frac{1}{2}tr(\Sigma^{-1}d\Sigma) + \Sigma^{-1}(x-u)(x-u)^T\Sigma^{-1})d\Sigma$$

最终：l 对于 sigma 的导数如下：令 0 求 sigma

$$\nabla_\Sigma \ell_t = \Sigma^{-1} - \Sigma^{-1}(x_t-\mu)(\Sigma^{-1}(x_t-\mu))^T$$

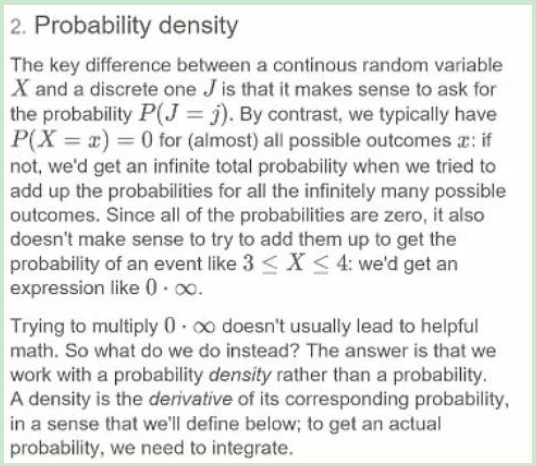
以上只是一个 example 的 gradient

以上如果用 SGD 来求，会先给参数 u 和 sigma 初始值，然后带入 gradient 来更新参数！！！

连续变量

连续变量取值太多了(无限)因此其概率分布，每一点的值几乎为 0

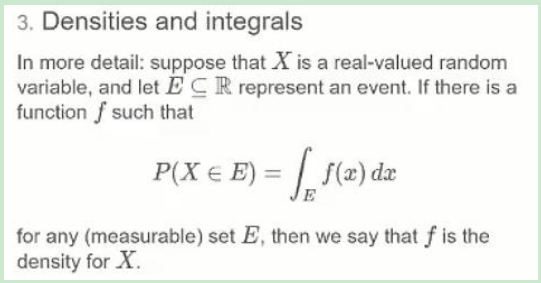
解决方法：probability density



2. Probability density

The key difference between a continous random variable X and a discrete one J is that it makes sense to ask for the probability $P(J = j)$. By contrast, we typically have $P(X = x) = 0$ for (almost) all possible outcomes x: if not, we'd get an infinite total probability when we tried to add up the probabilities for all the infinitely many possible outcomes. Since all of the probabilities are zero, it also doesn't make sense to try to add them up to get the probability of an event like $3 \leq X \leq 4$: we'd get an expression like $0 \cdot \infty$.

Trying to multiply $0 \cdot \infty$ doesn't usually lead to helpful math. So what do we do instead? The answer is that we work with a probability *density* rather than a probability. A density is the *derivative* of its corresponding probability, in a sense that we'll define below; to get an actual probability, we need to integrate.



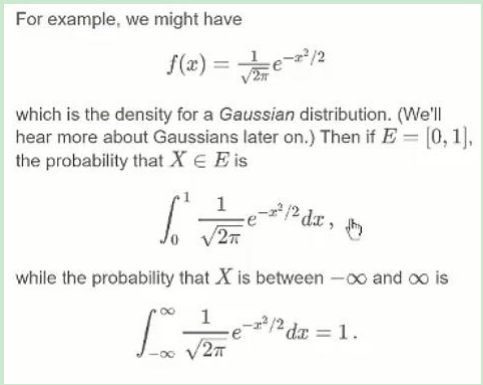
3. Densities and integrals

In more detail: suppose that X is a real-valued random variable, and let $E \subseteq \mathbb{R}$ represent an event. If there is a function f such that

$$P(X \in E) = \int_E f(x)\,dx$$

for any (measurable) set E, then we say that f is the density for X.

注意这里的概率分布 X 只能取区间，此时概率分布的导数就是 **密度函数**



For example, we might have

$$f(x) = \tfrac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

which is the density for a *Gaussian* distribution. (We'll hear more about Gaussians later on.) Then if $E = [0, 1]$, the probability that $X \in E$ is

$$\int_0^1 \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx,$$

while the probability that X is between $-\infty$ and ∞ is

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx = 1.$$

E 就是个 interval

density 的定义：只要在区间内积分等于该区间的概率！这个函数

就是 density

以上是概率分布函数

分布函数(累计分布函数)

4. CDF

Another way to describe the distribution of a continuous random variable is with its *cumulative distribution function* or *CDF*. If $f(x)$ is the density, then the CDF is defined as:

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u)\,du\,.$$

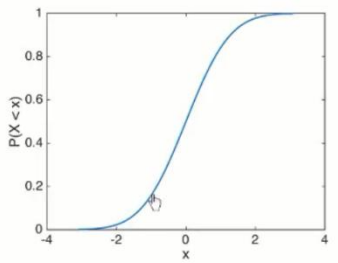
(We can actually define the CDF even in some situations where the density is undefined — e.g., if there is a *point mass*, a single point x such that $P(X = x) > 0$. But in this course we won't need that extra generality.)

CDF 永远是**单调递增函数 F(x)**

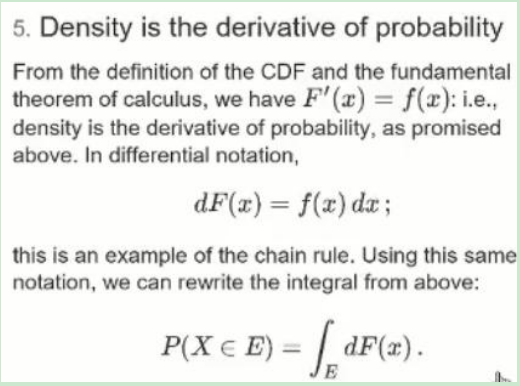
如果求[a,b]区间的值就直接 F(a)-F(b)

CDF 被称为**分布函数，分布函数和概率分布其实等价，知道其一就知道另一个：例如知道概率函数可以计算累计分布，知道累计分布，可以计算某个区间的概率分布**

For example, the CDF of the Gaussian distribution from above is:



The CDF is always monotone nondecreasing, and has $\lim_{x\to\infty} F(x) = 1$ and $\lim_{x\to-\infty} F(x) = 0$.



5. Density is the derivative of probability

From the definition of the CDF and the fundamental theorem of calculus, we have $F'(x) = f(x)$: i.e., density is the derivative of probability, as promised above. In differential notation,

$$dF(x) = f(x)\,dx\,;$$

this is an example of the chain rule. Using this same notation, we can rewrite the integral from above:

$$P(X \in E) = \int_E dF(x)\,.$$

question 微积分

概率分布函数和累计分布函数的导数都是密度函数！！！都可以用来求其两个函数

概率分布只能取区间，累计分布其实也是取区间

概率单调函数变换性质

$$P(X \leq x) = P(F(X) \leq F(x))$$

F(x)连续单调函数变换，不影响概率

P(x)=f(x)经常混在一起用 ，MLE 也会用 f(x)

但 f(x)的值可能大于 1

此时 p 不再代表概率，而是**概率密度**

Conditional density

7. Conditional densities

Just like a conditional probability, we can write a conditional probability density:

$$dF(x \mid y) = f(x, y)\, dx\,.$$

The conditional density is a function of both x and y, but it only makes sense to integrate with respect to x, not y. For example, we have

$$\int_{\mathcal{X}} f(x, y)\, dx = 1$$

(where $\mathcal{X}$ is the set of all possible outcomes x), but no such identity holds for y.

y 已经固定

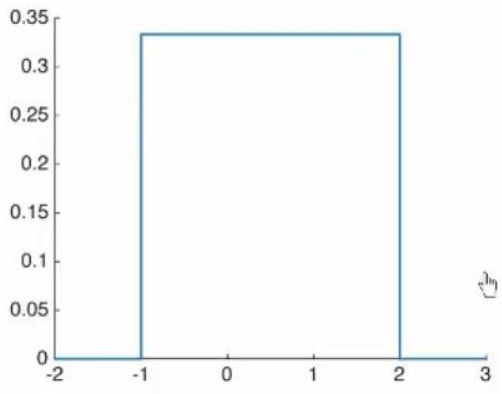
distribution

2. Uniform distribution

Perhaps the simplest density is the *uniform*:

$$P(x) = \begin{cases} 0 & x < a \\ \frac{1}{b-a} & a \le x \le b \\ 0 & x > b \end{cases}$$

In a uniform distribution, the random variable is equally likely to fall anywhere between the left and right limits, a and b. The parameters are just a and b. Here is a uniform density on the interval $[-1, 2]$:



3. Normal distribution

By far the most commonly-used continuous density is the *normal* or *Gaussian* distribution. There are a couple of reasons for its popularity: first is that it is analytically very nice to work with, and second is that it arises naturally via the *central limit theorem* (see below). In addition, the family of normal distributions is closed under addition (the sum of two normals is normal) and scalar multiplication (k times a normal is normal).

The normal distribution has two parameters: mean μ and variance σ^2. The square root of the variance, σ, is called the standard deviation. The mean shifts the distribution left or right, while the standard deviation scales the distribution wider or narrower around the mean.

If we add two or more random variables, the mean of the sum is the sum of the means; and, if the variables are independent, the variance of the sum is the sum of the variances. If we take a scalar multiple of a random variable, we scale the mean and the standard deviation (not the variance!): the mean of kX is $k\mu$, and the standard deviation is $k\sigma$. These properties hold for any random variables, but they're particularly useful for normally distributed ones, since they tell us the parameters of the resulting distribution.

normal advantage

closed under additation 和 scalar，还是 normal

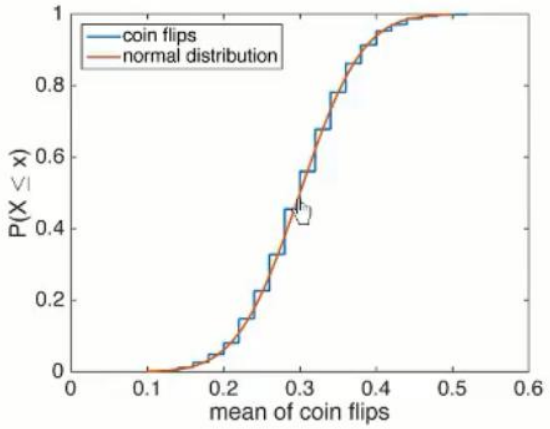
$$P(x \mid \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}.$$

对于标准正态：

This particular plot is a *standard normal*, with mean zero and variance 1.

A good rule of thumb is to remember that about 95% of the probability of a normal distribution falls between $\pm 2\sigma$, and about 2/3 of the probability falls between $\pm\sigma$. (The actual range for achieving 95% coverage is closer to $\pm 1.96\sigma$, and for 2/3 coverage is $\pm 0.96\sigma$, if you want some more significant digits.)

For example, suppose that we flip a biased coin many times. Each individual flip follows a Bernoulli distribution; a normal distribution would be a very bad approximation. But, if we take the average of many flips, the result will be much closer to normal. Here's a comparison:

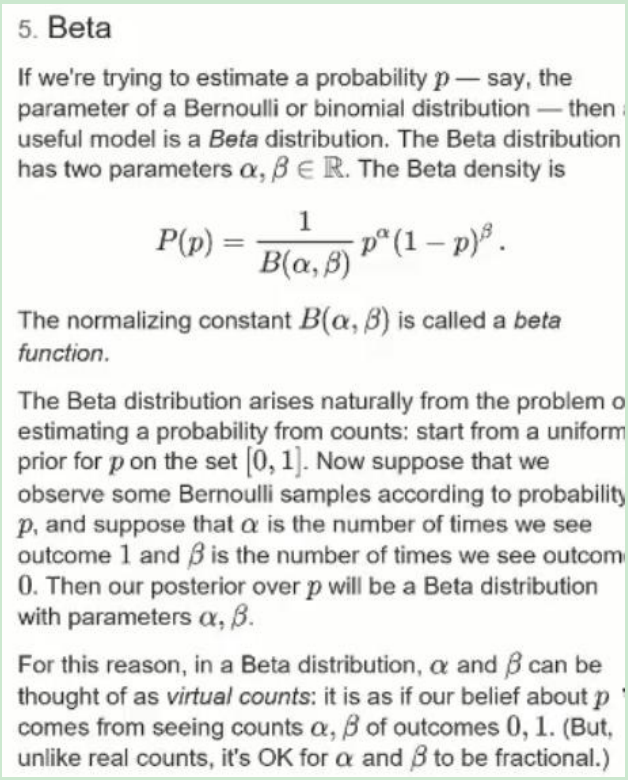


The blue line is an empirical approximation to the CDF of the mean of 50 flips of a coin with $P(\text{heads}) = 0.3$. (See below for how to construct such an approximation.) The red line is the CDF of a normal distribution with matching mean and variance.

For a more real-world example, a person's height is determined by a wide variety of genetic and environmental factors; only a few of these factors (like gender) have an individually significant effect. So, if we condition on the few individually-significant factors, the resulting distribution of heights will be very close to normal.

只有很少的显著的影响到的因素如 gender

这种条件分布，很接近正态分布(受很多因素的影响，没有决定性因素)



5. Beta

If we're trying to estimate a probability p — say, the parameter of a Bernoulli or binomial distribution — then useful model is a *Beta* distribution. The Beta distribution has two parameters $\alpha, \beta \in \mathbb{R}$. The Beta density is

$$P(p) = \frac{1}{B(\alpha, \beta)} p^{\alpha}(1-p)^{\beta}.$$

The normalizing constant $B(\alpha, \beta)$ is called a *beta function*.

The Beta distribution arises naturally from the problem o estimating a probability from counts: start from a uniform prior for p on the set $[0, 1]$. Now suppose that we observe some Bernoulli samples according to probability p, and suppose that α is the number of times we see outcome 1 and β is the number of times we see outcom 0. Then our posterior over p will be a Beta distribution with parameters α, β.

For this reason, in a Beta distribution, α and β can be thought of as *virtual counts*: it is as if our belief about p comes from seeing counts α, β of outcomes $0, 1$. (But, unlike real counts, it's OK for α and β to be fractional.)

BETA 是从 0-1 里 sample，可以用来估计一个分布的概率

Beta 是 p 的概率密度 P(p)，distibution of probality！！

假设伯努利实(我们不知道硬币有没有动过手脚，因此 p 未知)，需要根据观测来判断，我们先对 p 的先验是 uniform(**也就是 p 是任何值的概率都一样，现在模型的 p 已经已知**)，实验几次后，head 是 alpha，tail 是 beta，此时对 p 的后验 posterior 概率，Beta 就是 p 的概率密度 P(p)

alpha 和 beta 其实是对应两种实验结果

如果没有我们想要的分布：

6. Empirical estimates

Sometimes we don't know ahead of time which distribution to use, or sometimes we can't find a predefined family of distributions that matches our knowledge about the random variable. In this case it can make sense to try to use an *empirical estimate*: an estimate based on a random sample. We'll describe two methods to accomplish this: a piecewise linear CDF, and a kernel density estimate. Both of these methods use the random sample to build an approximate CDF for our density.

The benefit of an empirical estimate is that we need very little prior knowledge about our random variable. The difficulty is that we need a large enough random sample to build an accurate CDF. For one-dimensional continuous random variables (like the ones we are discussing in this set of lecture notes) it can be quite practical to collect a large enough random sample. Unfortunately, as we move to multiple variables the required sample size increases sharply; so, while the empirical methods we describe here generalize well to two or three random variables at once, they become impractical very quickly after that.

但是需要太多的数据，尤其是多维的变量

两种方法：

1 piecewise linear CDF

Given a sample $X_1, \ldots, X_n$ from our random variable X, we can build a piecewise-linear approximation to the CDF for X as follows: first, sort our sample, so that $X_1 \leq X_2 \leq \ldots \leq X_n$. Then, assign each point a probability from 0 to 1 in increments of $\frac{1}{n-1}$: that is, assign the ith point the probability

$$p_i = \frac{i-1}{n-1}.$$

将我们的 sample 排序

Finally, build our CDF by connecting each pair of adjacent points with a line:

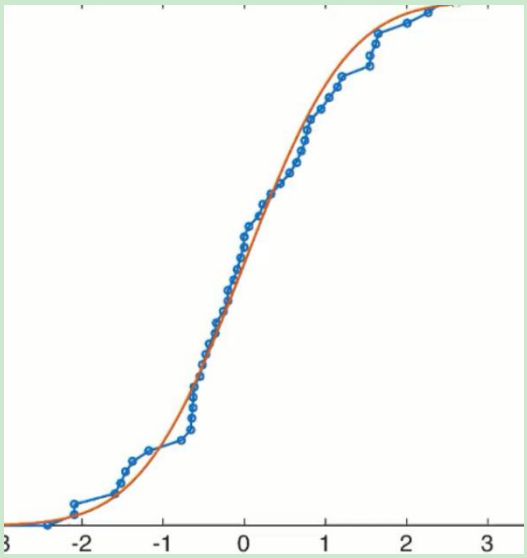
$$\hat{F}(x) = \begin{cases} 0 & x < X_1 \\ \frac{i-1}{n-1} + s_i(x - X_i) & X_i \le x < X_{i+1} \\ 1 & X_n \le x \end{cases}$$

where the slope s_i is determined by the distance between points $i-1$ and i:

$$s_i = \frac{1}{X_{i+1} - X_i} \frac{1}{n-1}.$$

Here's an example of a piecewise linear CDF built from a sample of 50 points from a Gaussian distribution:

然后每两个点 X1,X2 之间连上直线



第二种方法：

8. Kernel density estimate

Another simple way to build an empirical CDF is a *kernel density estimate*. We start from the CDF of a base distribution, called the *kernel*: the kernel should be centered at 0 and fairly narrow. For example, we could use a Gaussian distribution with a small variance; or, we could even use a point mass distribution, whose CDF is a step function. (Note: this use of the word "kernel" is different from the way we previously used it, in the context of function spaces and support vector machines.)

从一个 **kernel**(CDF of **base distribution**)出发，这个 distribution 是 **center=0 narrow (例如一个高斯分布，variance 很小)**

If F is the CDF of our kernel, then we build the estimate

$$\hat{F}(x) = \frac{1}{n}\sum_i F(x - X_i).$$

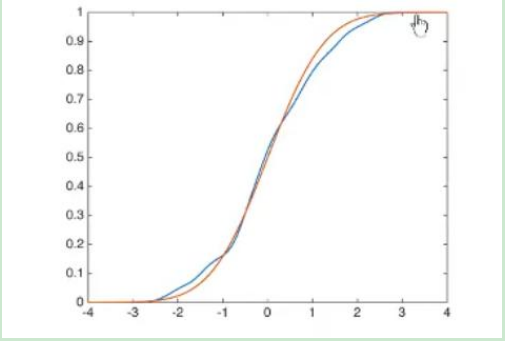
That is, we average together n copies of our kernel's CDF, one for each of our samples. The ith copy is shifted so that it is centered at X_i.

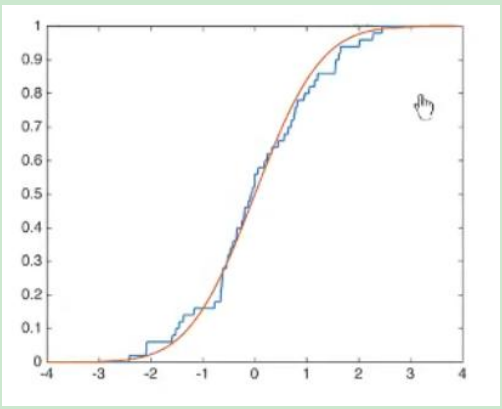
假如有 n points，我们先 copy n 个 our kernel，然后 shift 每个 kernel centered at ith points，然后求平均

每一个 X 一个 cdf(center=X)，**然后求平均**

例子：高斯 kernel 和 step function kernel

Here are two examples, built from the same 50 points as above. The first plot uses a Gaussian kernel with $\sigma = \frac{1}{5}$, while the second uses a point mass kernel (step function CDF).





QQ plot

CHECK whether we have the right CDF

或者验证我们的分布是某个分布！！

9. Q-Q plot

There are some problems with empirically estimating a CDF: one is that it takes a lot of data to get a really accurate answer, and another is that the resulting CDF is not very easy to work with analytically. So, one common strategy is to build an empirical CDF $\hat{F}$ and compare it to common families of distributions, to see if there's a match. If there is, we can use the known CDF of the matching family, and get the benefits of high accuracy and analytic tractability.

需要判断一下我们估计的 CDF 是否和已知的 distribution 一致，如果一致就用已知的，这样更精确！！

A common tool for comparing two CDFs is the Q-Q or *quantile-quantile* plot. Given a CDF $F(x)$ and a probability $\alpha \in [0,1]$, the α *quantile* of F is defined to be $F^{-1}(\alpha)$: that is, the value of x such that $P(X \leq x) = \alpha$. (We also sometimes say the $100\% \cdot \alpha$ *percentile*.)

quantile-quantile plot

因为 CDF，值域是 0-1

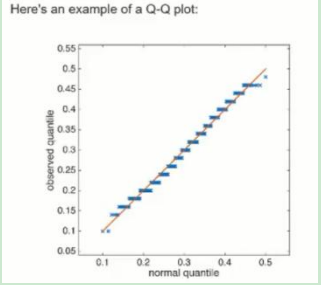
20%是 CDF 的 y 值 对应 quantile 也就是 CDF 的 x 值(x 是 CDF 的分位数 quantile)也就是 CDF 反函数 $F^{-1}(\alpha)$ alpha 是 y

QQ 图来比较两个 CDF 的 $F^{-1}(\alpha)$ 是否一致

To make a Q-Q plot, we start from two CDFs that we want to compare: say an empirical CDF $\hat{F}$ and the analytic CDF F for some known distribution like a normal. We then vary α from 0 to 1, and compare the corresponding quantiles of F and $\hat{F}$ by plotting $\hat{F}^{-1}(\alpha)$ vs. $F^{-1}(\alpha)$. If the plot is (approximately) a straight line, we declare a match.

question：如何转换成 qq plot 的点？

Here's an example of a Q-Q plot:



如果是 45 度过原点直线说明两个 CDF 一样

A nice feature of Q-Q plots is that they are robust to variations in location and scale: e.g., if we get the mean and variance of a Gaussian distribution wrong, the plot will still be a straight line. The only difference will be that the slope and intercept of the line will change: if we have the perfect match our line will have slope 1 and go through the origin, while if we are off on location or scale our line will have some other intercept or slope.

只针对于：location(均值)和 scale(方差)

不管分布如何**平移和乘以 scalar**，都可以比较

只要是**同一类分布**都是**直线**，比如不同的**方差，均值** of normal distribution，也会在一条直线，只是斜率和截距不同

On the other hand, if we are looking at a family of distributions that has a parameter that is not a location or scale parameter, we need to estimate this parameter first in order to use a Q-Q plot. For example, we could estimate the parameter by maximum likelihood, under the assumption that the family in question is correct: if it is, then the Q-Q plot will show a straight line, and if it isn't, then it won't matter that the parameter estimate is meaningless.

比如 beta 的参数，就不是 **location(均值)和 scale(方差)，需要先估计出来**

Maximum entropy on continuous variables

$H(X) = \sum_x -p(x)\ln p(x)$ ← discrete

$H(X) = \int_{\mathcal{X}} -p(x)\ln p(x)\,dx$ ← contin.

此时 p 是 density

$$g(\theta) = \ln \int_{\mathcal{X}} P_0(x) \exp(t(x) \cdot \theta)\, d\theta\,.$$

Unlike the finite case, for some values of θ it might be impossible to normalize the distribution: the integral over x might be infinite. We rule out such values of θ: we define Θ to be set of the valid parameter vectors,

$$\Theta = \{\theta \mid \int_{\mathcal{X}} P_0(x) \exp(t(x) \cdot \theta)\, dx \text{ is finite}\}$$

and we require $\theta \in \Theta$. The set Θ is always convex.

The normal and Beta families defined above are examples of exponential families. The normal is based on the statistics $t(X) = (X, X^2)^T$, while the Beta uses the log-probabilities of 0 and 1 as statistics, $t(P) = (\ln(P), \ln(1-P))^T$.

question normalize?

multivariate probility density

实质是多元函数

The same idea works for multiple variables: if X and Y are random variables, and there is a function $f(x, y)$ such that

$$P(a \le X \le b, c \le Y \le d) = \int_a^b \int_c^d f(u, v)\, dv\, du$$

then we call f the *joint density* of X and Y. Similarly, for three variables, the joint density must satisfy

$$P(E) = \int_a^b \int_c^d \int_e^f f(u, v, w)\, dw\, dv\, du$$

where

$$E = \{(x, y, z) \mid a \le X \le b, c \le Y \le d, e \le Z \le f\}.$$

Note that event probabilities now correspond to multiple integrals, since we have multiple dimensions.

joint density

Note that event probabilities now correspond to multiple integrals, since we have multiple dimensions.

Similarly, we can generalize the cumulative distribution function (CDF) to two or more variables as

$$F(x, y) = P(X \le x, Y \le y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f(u, v)\, dv\, du.$$

As before, the fundamental theorem of calculus tells us that differentiating the CDF F repeatedly gives us the density f.

dF 先对 x 求导，再对 y 求导，得到 f density

2. Identities

All of the identities that we're used to — the sum rule, the chain rule, etc. — still hold for probability densities. They look slightly different, though. Write $p(x, y)$ for the joint density of X and Y. Write $q(x)$ and $r(y)$ for the marginal densities of X and Y respectively. Write $s(x \mid y)$ for the conditional density of X given $Y = y$. Then we have:

- Sum rule: $\int_{\mathcal{Y}} p(x, y)\, dy = q(x)$, where $\mathcal{Y}$ is the set of possible values of the random variable Y.
- Chain rule: $p(x, y) = s(x \mid y)\, g(y)$.

It can be annoying to assign a separate name to each probability density function, so a common shortcut is just to use p or P for all of them:

- Sum rule: $\int_{\mathcal{Y}} P(x, y)\, dy = P(x)$.
- Chain rule: $P(x, y) = P(x \mid y)\, P(y)$.

sum rule: marginal density 是对另一个变量求积分

chain rule: conditional density

就把 density 当做**概率(统一用 P 表示)**，来使用，但不是概率！

density 也适用于 **bayes rule**

3. Bayes rule

Maybe the most important identity for working with densities is the Bayes rule:

$$P(y \mid x)\, P(x) = P(x \mid y)\, P(y)\,.$$

or

$$P(y \mid x) = P(x \mid y)\, P(y) / P(x)\,.$$

As above, these look just like the corresponding equations for discrete random variables; but, the terms are densities instead of probabilities.

多元变量 density 的 entropy

4. Information and entropy

We can define entropy and related quantities for multivariate probability densities as well, although the usual caveats apply: we need to be careful about coordinate systems and about non-smooth densities. We define

$$H(X) = -\int_{\mathcal{X}} P(x) \ln P(x)\, dx$$

$$H(X, Y) = -\int_{\mathcal{X}} \int_{\mathcal{Y}} P(x, y) \ln P(x, y)\, dx\, dy$$

$$H(X \mid Y) = -\int_{\mathcal{X}} \int_{\mathcal{Y}} P(x, y) \ln P(x \mid y)\, dx\, dy$$

$$KL(P, Q) = \int_{\mathcal{X}} (\ln P(x) - \ln Q(x))\, P(x)\, dx$$

where $\mathcal{X}$ is the domain of X, $\mathcal{Y}$ is the domain of Y, and we have arbitrarily chosen the natural logarithm.

It is difficult to interpret a continuous entropy in terms of information, since it would take infinitely much information to convey the exact value of a continuous random variable. Instead, what makes sense is to *compare* the entropy of one continuous random variable to that of another. The difference in entropies $H(X) - H(Y)$ tells us how many more bits it would take to transmit X than Y, if we use the same (high enough) accuracy for both.

In more detail, suppose that X and Y have the same range. Discretize this range into k equal bins. Measure the entropy of the discrete versions of X and Y, and subtract. Now take the limit as $k \to \infty$: the result is the difference in entropies of the original continuous variables.

The zero point of continuous entropy is defined to be the uniform distribution on the interval $[0, 1]$. That is, if Y follows this distribution, then $H(Y) = 0$, and $H(X)$ is the number of extra bits we need to use for X compared to Y.

直接看 H(X)意义不大，因为 continue value 无限大，无限长的，

如果要精确传递，信息量也**无限大**

H(X)- H(Y)可以比较两种信息的差

The zero point of continuous entropy is defined to be the uniform distribution on the interval $[0, 1]$. That is, if Y follows this distribution, then $H(Y) = 0$, and $H(X)$ is the number of extra bits we need to use for X compared to Y.

question?

Modeling-multiple variables

joint model 多元变量

想对多个变量 model

1. Joint models

In the previous sets of lecture notes, we looked at how to model *individual* variables, either discrete or continuous. Of course, in most ML problems, we need more than one random variable. In particular we need a way to model *interactions* among variables: situations in which knowing the value of one random variable gives us information about the likely values of another random variable.

As a motivating example, imagine that we record the height, weight, gender, and age of a patient entering a doctor's office. Knowing one of these variables (say, gender) tells us something about the likely values of other variables (e.g., height and weight). We can use this information in either direction: taller patients are more likely to be male, and male patients are more likely to be taller.

多个变量，有关联，想 model 多个变量，可以用随机向量的

distribution 或者 Graphic model

There are two common (related) ways to build these joint models: vector-valued random variables and graphical models. This set of notes covers vector-valued random variables; we'll return to graphical models later.

2. Vector-valued random variables

To build a joint model, we can collect all of our random variables together into a single vector-valued random variable: e.g., we could record height (in cm), weight (in kg), gender, and age (in years) as a vector

$$\begin{pmatrix} 190 \\ 86 \\ M \\ 47 \end{pmatrix} \in \mathbb{R}_+ \times \mathbb{R}_+ \times \{M, F\} \times \mathbb{R}+ .$$

Once we've done so, we can model this vector using multivariate analogs of the distributions we've already discussed. We'll discuss three:

- Dirichlet,
- multivariate normal, and
- multivariate maxent.

其实就是多元联合分布，都是 sample 得到多个值，都是这多个值的概率分布

joint model 例子

1Dirichlet distribution

3. Dirichlet distribution

Earlier, we used a Beta distribution to model a random variable that represents a probability — e.g., the probability p that a coin will land as heads. (Recall that p in this case plays two roles: first, it helps us specify the probability distribution of a random variable, namely a coin flip, via the Bernoulli(p) distribution. Second, it is itself a random variable: it is a sample from from a Beta(α, β) distribution.)

The Dirichlet distribution generalizes the Beta distribution to more than two outcomes: e.g., a die roll or a Unicode character. If there are k possible outcomes, we need to model k probabilities that sum to 1; call them $p_1, p_2, \ldots, p_k$. The Dirichlet distribution has k parameters, $\alpha_1, \alpha_2, \ldots, \alpha_k$; its density function is

$$\frac{1}{B(\alpha_1, \ldots, \alpha_k)} p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$$

where $B(\ldots)$ is a normalizing constant, called a Beta function.

beta 的延伸

beta 是两种结果，只有 1 个概率 p

Dirichlet 现在有**多个结果**，每种结果一个概率，需要多个 alpha(多个结果的 count)，每次 sample 产生多个 p(每个分类一个

p)，并且 sum 为 1(dirichlet 就是这个 p 组合的概率分布(多个概率的概率分布))

Just as a Beta distribution arises naturally from inference about a Bernoulli distribution, a Dirichlet distribution arises naturally from inference about a categorical distribution: suppose we start from a uniform prior over the parameters $p_1, p_2, \ldots, p_k$ of a categorical distribution (supported on the simplex $p_i \geq 0$, $\sum_i p_i = 1$). Suppose we then observe some trials, and see outcome 1 of the categorical distribution on α_1 trials, outcome 2 on α_2 trials, ..., and outcome k on α_k trials. Then our posterior distribution over $p_1 \ldots p_k$ will be Dirichlet with parameters $\alpha_1, \ldots, \alpha_k$.

Because of this fact, we can think of $\alpha_1, \ldots, \alpha_k$ as "virtual counts," just as we did for the parameters of a Beta distribution: if our belief about $p_1 \ldots p_k$ is represented by a Dirichlet distribution with parameters $\alpha_1, \ldots, \alpha_k$, it's as if we've seen α_1 copies of outcome 1, α_2 copies of outcome 2, etc. (But it's fine in this case for α_i to be fractional, unlike with real counts.)

The Beta distribution is a special case of the Dirichlet distribution: if p_1, p_2 follow a $\text{Dirichlet}(\alpha_1, \alpha_2)$ distribution, then p_2 follows a $\text{Beta}(\alpha_1, \alpha_2)$ distribution, and $p_1 = 1 - p_2$ follows $\text{Beta}(\alpha_2, \alpha_1)$.

dirichlet 是对 **category distribution 的推断**(uniform prior)

Beta 分布是 dirichlet 分布的**特例(分类数为 2)**

Mulitivarate normal distribution

其实就是联合分布，只是这里是 density，density 是多元函数！！

The multivariate normal (or Gaussian) distribution in k dimensions has two parameters: the mean vector $\mu \in \mathbb{R}^d$ and the variance matrix $\Sigma \in \mathbb{R}^{d \times d}$. These parameters represent (unsurprisingly) the mean and variance of X: $\mu_i = E(X_i)$, $\Sigma_{ii} = \text{Var}(X_i)$, and $\Sigma_{ij} = \text{Cov}(X_i, X_j)$. Just as the variance of a univariate normal distribution is always positive, the variance matrix of a multivariate normal is always positive definite. Σ is also called the *covariance* matrix; it is the same matrix either way, and its elements are the variances and covariances of the elements of X.

It is also common to specify the inverse of the covariance matrix, instead of the covariance matrix itself: $\Omega = \Sigma^{-1}$. This parameterization is equivalent since Σ has to be positive definite and therefore invertible. We call Ω the *precision* matrix.

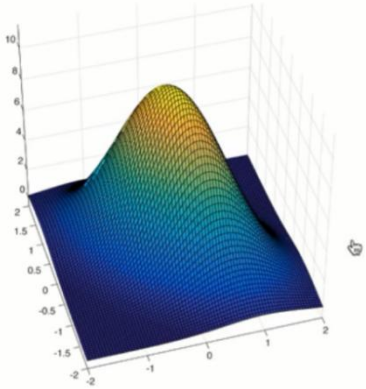
Given these parameters, the Gaussian density is

$$P(x \mid \mu, \Omega) = \sqrt{|2\pi\Omega|} \exp(-\tfrac{1}{2}(x-\mu)^T \Omega (x-\mu)) .$$

d 元高斯分布，X 是 d 维向量

u 是对应的 d 维向量($u_i = E(X_i)$**)** $\sum$ **是 d*d 维的矩阵(方差和协方差)，用 precision matrix 代替，代表多个变量之间的关联**

Here's an example of a Gaussian with mean $(0, 0)^T$ and covariance $\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{pmatrix}$.



2-d

The marginal and conditional distributions of a Gaussian are also Gaussian. That is, when a vector X follows a multivariate Gaussian distribution, each of the sub-vectors of X follows a lower-dimensional Gaussian distribution. (By a sub-vector we mean a vector that we get by picking a subset of the coordinates of X.) And, if we condition on one or more components of X, the remaining components follow a Gaussian distribution. In particular, if we marginalize or condition down to a single remaining variable, it will follow a univariate Gaussian distribution.

marginal/condition，其实都是将 X 降维， X 的 low dimension 仍然是高斯分布

marginal 是将 p(x,y,z)变成 p(x,y)，仍是高斯分布

Just like its cousin the univariate Gaussian, the multivariate Gaussian is ubiquitous. The reasons for its popularity are similar: the Gaussian family is closed under addition and multiplication by a matrix; it is analytically tractable; and there is a multivariate analog of the central limit theorem.

In addition, the multivariate Gaussian has one more nice property: it is a very convenient way to specify interactions among variables. We'll take advantage this property later, when we discuss Gaussian graphical models.

addition 和 scalar 仍然 closed！！

而且 interaction with variables 方便

multivariate maxent distribution

Just as in the discrete and continuous cases, we can define a vector of features $t(X)$ that depends on a vector-valued random variable X: for example, we might set $t(X) = (X_1, X_2, X_1^2, X_2^2, X_1X_2)^T$. We can then describe the distribution of X using the mean parameter $\bar{t} = E(t(X))$.

本来是 X: X1,X2， t(X)变成上面的更多的变量

t ba 可以用来描述 X 的分布

Given $\bar{t}$, the maximum entropy or maxent distribution is the one that maximizes $H(X)$ subject to the constraint $E(t(X)) = \bar{t}$. And, the minimum relative entropy distribution (with base distribution P_0) is the distribution that minimizes KL divergence from P_0.

在分布满足 E(t(X))=t ba 的情况下，选择 max entropy 的分布

Just as in the previous cases, there is an explicit expresion for the maxent or minimum relative entropy distribution: it is

$$P(x) = P_0(x)\exp(t(x)\cdot\theta - g(\theta))\,.$$

Here θ is the natural parameter, and $g(\theta)$ is a normalizing constant. As before, we can compute $g(\theta)$ by summing and/or integrating $P_0(x)\exp(t(x)\cdot\theta)$ over all feasible values of X, then taking the log. Also as before, we have $g'(\theta) = \bar{t}$.

By varying $\bar{t}$ (or equivalently by varying θ) we trace out a family of distributions, called an exponential family.

A good example of a multivariate maxent distribution is the multivariate normal: we get this distribution by taking $t(X)$ to be the vector of all first- and second-degree monomials based on the components of X. For example, in the 2-d case, we take $t(X) = (X_1, X_2, X_1^2, X_2^2, X_1X_2)^T$.

computation in maxent 很高！！！

power law 幂 distribution

If the set of allowed values of X is a subset of $(0,\infty)$, and if we use the statistic $t(X) = -\ln(X)$, we get an important class of distributions called *power laws*. These distributions have densities that look like

$$\exp(-\theta\ln(x) - g(\theta)) \propto x^{-\theta}\,.$$

The power law density is defined if $\theta > 1$, since the integral over x is infinite otherwise. The name "power law" comes from its functional form: the dependence on x is a (negative) power of x. Many natural phenomena have been claimed to follow power law distributions: for example the strength of an earthquake, or the property damage due to a hurricane.

X 是 positive number ， theta>1

和 $x^{-\theta}$ 成正比， $x^{-\theta}$ 就是长尾函数（幂函数）

特点是有一个 heavy tail ，很常见的现象

We can derive the CDF of a power law from the fact that $\int x^{-\theta} dx$ is proportional to $x^{-(\theta-1)}$: if the range of X is $(0, \infty)$, then the CDF is

$$F(x) = 1 - x^{-(\theta-1)}.$$

In particular, $1 - F(x)$ also follows a power law, but with a different exponent.

Log-log plot

来验证 polynomial 函数形式（幂函数）指数固定

12. Log-log plot

There's a useful trick that can help us identify a power law distribution. Recall that the CDF of a power law is of the form $1 - \text{poly}(x)$. A well-known way to identify polynomials is with a log-log plot: if we plot $\ln(x)$ vs. $\ln(x^k)$, the result will be a straight line of slope k, since $\ln(x^k) = k \ln(x)$.

比较 x 和 x^k　　ln(x)　　k ln(x)　是线性关系

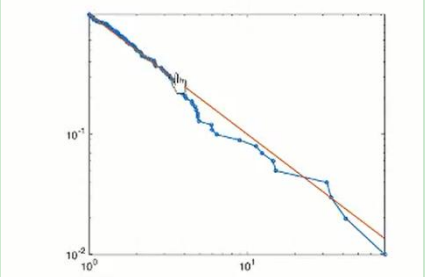
可以 identify polynomials x 和多项式取完 log 一定是线性的

So, to identify a power law distribution, we can build an empirical estimate of the CDF $\hat{F}(x)$, and make a log-log plot of x vs. $1 - \hat{F}(x)$. If we have a true power law distribution with parameter θ, we'll get a line with slope $-(\theta - 1)$.

如我们经验估计 cdf，我们想验证是否是 power law distribution，

因为 1-cdf 是 polynomial(power law 幂)和 x 取 log，画图 slope 是-(theta-1)，因为 $F(x) = 1 - x^{-(\theta-1)}$.

The result looks something like this:



Here we're showing an example (in blue) based on a sample of 100 points from a power law distribution with $\theta = 2$. Because $\theta = 2$, the approximate slope of the blue curve is -1; we show a line with slope -1 for comparison (in red).

Note that the more-common small values of x are bunched together at the top left of the plot, while the long tail of larger values of x is spread out to the bottom right. For this reason, the empirical CDF gets a lot more noisy toward the bottom right: that portion of the curve is based on fewer points.

Transformation

如果 X 不 follow 任意我们上面的 distribution，需要对 X 进行变换，变换后会服从某个分布

Given a random variable X, any function of X is a random variable too: e.g., $\exp(X)$ or $\ln(X)$ or X^3. If the distribution of X isn't well modeled by one of the above families, then perhaps the distribution of some invertible transformation of X is; if so, and if we can find the appropriate transformation, we are in good shape. This strategy is often used in conjunction with tools like Q-Q plots to diagnose whether we have found a good transformation and a good model for the transformed version of X.

QQ plot 来 check，我们是否找到合适的变换

CDF Transformation

In fact, it turns out that we can transform *any* density into any other density, so long as we know the CDFs for both. (By saying we have a density, we are assuming that our CDF doesn't have any vertical jumps in it — that is, there are no single points x such that $P(X = x)$ is positive.)

This transformation is based on the following observation: suppose we have a CDF F, and suppose U is distributed uniformly on the unit interval $[0, 1]$. Then if we define

$$X = F^{-1}(U)$$

we will have

$$X \sim F,$$

that is, X will follow the distribution whose CDF is F.

想把 U 变成 CDF ，上面变换后，X 就服从 distribution，whose

CDF is F(不是直接服从 CDF)

证明：此时我们得到的 X 的 CDF 是 F: 只需证明 P(X<=x)=F(x)

To see why, note that

$$\begin{aligned} P(X \leq x) &= P(F(X) \leq F(x)) \\ &= P(U \leq F(x)) \\ &= F(x)\,. \end{aligned}$$

The first line is true because we made the same transformation of X and x. The second is by the definition of X in terms of U. The last line follows from a property of uniform random variables: the CDF of a uniform random variable is just a straight line.

U 是 uniform random variable(也是一个区间)

1 第一行：单调函数变换不影响概率 2 又 F(X)=U(根据上面的 define，U 就是 F 的值域)3 因为 U 是 uniform random variable on 0-1，小于某个值的概率就是那个值(面积比)

The above derivation shows how to transform a uniform variable into some other known distribution. We can also reverse the process: since we have $U = F(X)$, we know that applying the CDF of X to X results in a uniform random variable.

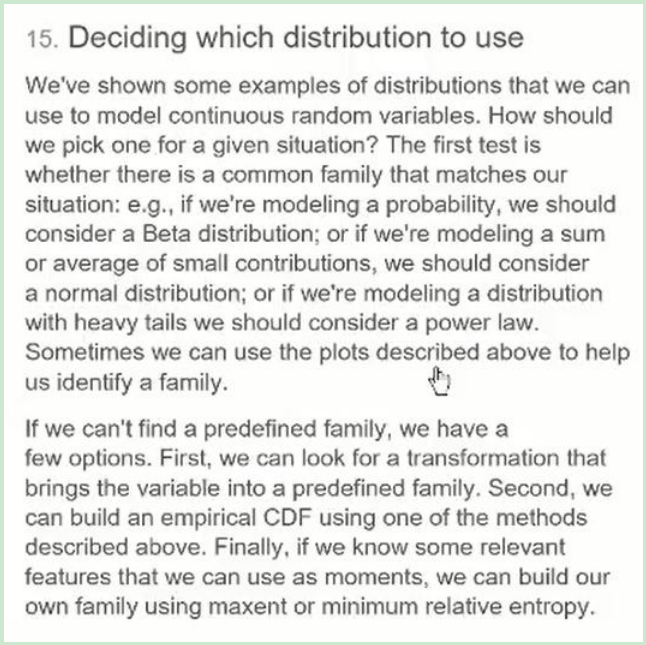
可以将任意 distribution 转成 uniform

Finally, we can chain the above two transformations together: if F and G are two different CDFs, and if X is sampled from the distribution with CDF F, then

$$Y = G^{-1}(F(X))$$

follows the distribution with CDF G: we first transform X into a uniform random variable $F(X)$, and then transform the uniform random variable into a variable Y with CDF G.

两个任意 CDF 转换，通过“中间变量”转成 uniform，再转



15. Deciding which distribution to use

We've shown some examples of distributions that we can use to model continuous random variables. How should we pick one for a given situation? The first test is whether there is a common family that matches our situation: e.g., if we're modeling a probability, we should consider a Beta distribution; or if we're modeling a sum or average of small contributions, we should consider a normal distribution; or if we're modeling a distribution with heavy tails we should consider a power law. Sometimes we can use the plots described above to help us identify a family.

If we can't find a predefined family, we have a few options. First, we can look for a transformation that brings the variable into a predefined family. Second, we can build an empirical CDF using one of the methods described above. Finally, if we know some relevant features that we can use as moments, we can build our own family using maxent or minimum relative entropy.

1 先看有没有现成的分布符合我们的 data

use plot 来判断

2 没有现成的就是变换 或者 经验 CDF

Conditional model(生成和判别)

LINEAR MODEL(linear 高斯模型)

很多情况下是最好 model

So far we've been looking at ways to model the joint distribution of several random variables. Sometimes that might be more work than we need to do, though: we might only need to model the distribution of some subset of the variables (say $Y_1, Y_2, \ldots, Y_d$) conditioned on the distribution of the remaining variables (say $X_1, X_2, \ldots, X_p$). The X_i are called *inputs* or *independent variables*. The Y_j are called *outputs*, *targets*, *labels*, or *dependent variables*.

Write $X = (X_1, \ldots, X_p)$ for the vector of inputs, and $Y = (Y_1, \ldots, Y_d)$ for the vector of outputs. The joint distribution $P(X, Y)$ can be factored as

$$P(X,Y) = P(X)\,P(Y \mid X)\,.$$

If we only need to model $P(Y \mid X)$, we save the effort (and the data) required to come up with a model of $P(X)$.

对于多个变量建模，有时候只需要 model 一部分变量 Y..的分布，condition on 剩余变量 X..(已知)，X 就是 input 变量，Y 是 output

P(X,Y)可以被分解 P(X)P(Y|X),**我们只需要 model P(Y|X) conditional model**

P(X,Y)是 joint model(generative model 生成模型) 可以 generate pairs(X,Y)

P(Y|X) conditional model 如果 Y 是离散分类，P(Y|X) 是 discriminative model(判别模型)，判别是哪个 Y

以上，X Y 都是多变量

A model of $P(Y \mid X)$ is called a *conditional* model, by contrast with a model of $P(X, Y)$, which is called a *joint* model. A joint model is also sometimes called a *generative* model: if we have the full joint distribution we can sample (generate) pairs (X_i, Y_i), while if we only have a conditional model, we cannot. In the special case that Y is discrete, we call $P(Y \mid X)$ a *discriminative* model, since it lets us choose (discriminate) among different values of Y.

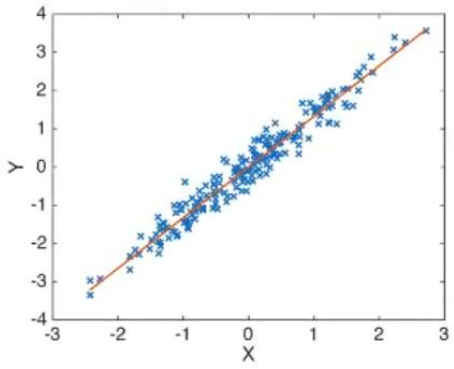
最常见的判别模型是 linear model

By far the most common conditional model is the linear model. In the case $d = p = 1$ (i.e., only one input X and one output Y, called the *univariate* case), the linear model is

$$Y \approx \hat{Y} = Xw + b\,.$$

Here $\hat{Y}$ is our prediction of Y. $\hat{Y}$ is a linear function of X, with slope $w \in \mathbb{R}$ and intercept $b \in \mathbb{R}$.

linear model 是 **X Y 都是单变量**



Estimating a linear model is called *linear regression*. The rest of this set of notes will focus on univariate linear regression; later, we'll generalize to multiple inputs and outputs.

estimate 参数 w 和 b

3. The sample

To fit a conditional model, we need a sample drawn i.i.d. from the joint distribution $P(X, Y)$. Note that this is the same type of data we would use to fit a joint model; the conditional model doesn't change the type of data we need, even if it can reduce the complexity of our model and the total number of samples we need.

We will write our sample as

$$(X^1, Y^1), (X^2, Y^2), \ldots, (X^n, Y^n)$$

where n is the sample size. Note the use of superscripts instead of subscripts to index samples: this convention will later help us avoid confusion between X^1 (the first sample of X) and X_1 (the first component of X). So, for example, X_3^7 refers to the third component of the seventh sample of X.

如果想根据 sample data 来 fit model(distribution) **需要假设 data 是从原有的 joint distribution 里 IID 获取)**

4. Minimizing residuals

Given a sample $\left((X^i, Y^i)\right)_{i=1}^n$, we can determine the best values of the parameters w and b by minimizing the *residuals* or *errors* $Z^i = Y^i - \hat{Y}^i$. In more detail, suppose $\ell(z)$ is a function that has a minimum at $z = 0$ and increases monotonically as z moves away from 0 — we call such an ℓ a *loss function*. Then, we can find good values of w and b by minimizing the sum of losses

$$L = \sum_{i=1}^n \ell(Z^i) = \sum_{i=1}^n \ell(Y^i - (X^i w + b)).$$

By far most common loss function is the *squared loss*, $\ell(z) = z^2$; minimizing squared loss is called *least squares* (a problem that we've seen before).

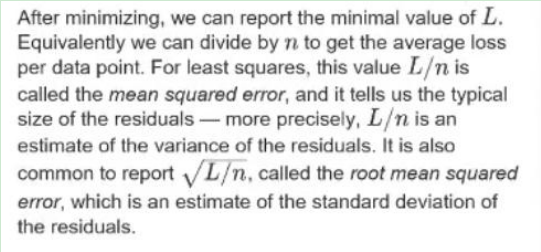
目标使得 residual 最小！！

define loss function:

需要定义一个通用的 loss 函数，input 是 residual，residual 越

小，loss 越小，只需使得全部的 loss sum 最小！！

最常用的是取平方

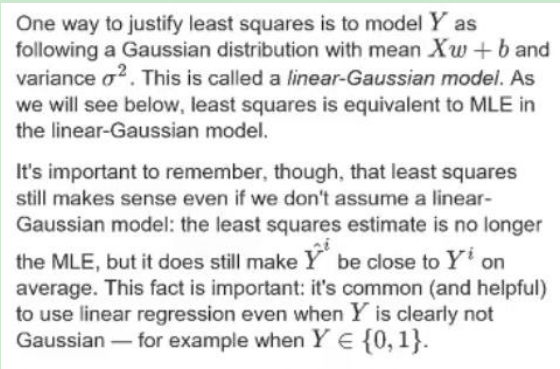

After minimizing, we can report the minimal value of L. Equivalently we can divide by n to get the average loss per data point. For least squares, this value L/n is called the *mean squared error*, and it tells us the typical size of the residuals — more precisely, L/n is an estimate of the variance of the residuals. It is also common to report $\sqrt{L/n}$, called the *root mean squared error*, which is an estimate of the standard deviation of the residuals.

如果我们 assume p(Y|X)是高斯分布(linear 高斯分布)，loss 就是 square loss！！！

此时是 linear-高斯 model： $Y = X\mathrm{w} + \mathrm{b} + \varepsilon$

其实就是假设 ε 高斯分布，X 是已知的，再已知 X 的情况下加上随机波动的误差，得到 Y. $p(\varepsilon)$ 和 $p(Y \mid X)$ 都是高斯分布，$p(Y \mid X)$ 其实就是 $p(\varepsilon)$ 均值移动 Xw+b，方差不变，在这个假设下去做 MLE，就会发现等同于 least square

linear 高斯模型 and MLE 等价


One way to justify least squares is to model Y as following a Gaussian distribution with mean $Xw + b$ and variance σ^2. This is called a *linear-Gaussian model*. As we will see below, least squares is equivalent to MLE in the linear-Gaussian model.

It's important to remember, though, that least squares still makes sense even if we don't assume a linear-Gaussian model: the least squares estimate is no longer the MLE, but it does still make $\hat{Y}^i$ be close to Y^i on average. This fact is important: it's common (and helpful) to use linear regression even when Y is clearly not Gaussian — for example when $Y \in \{0, 1\}$.

MLE for linear-高斯 model 的 solution 就是 least square

To show that least squares yields the MLE in a linear-Gaussian model, we can start from the log likelihood (the log of the probability of our data (X^i, Y^i)):

$$L = \sum_{i=1}^{n} \ln P(X^i, Y^i)$$
$$= \sum_{i=1}^{n} (\ln P(Y^i \mid X^i) + \ln P(X^i))$$

MLE:

写出 log likelihood（永远是联合概率的乘积），由于假设 conditional model，所以可以分解，又假设 P(Y|X)高斯分布，带入：

$$\sum_{i=1}^{n} \ln P(X^i, Y^i)$$
$$\sum_{i=1}^{n} (\ln P(Y^i \mid X^i) + \ln P(X^i))$$
$$-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (Y^i - (X^i w + b))^2 - \frac{n}{2} \ln 2\pi\sigma^2 + \text{constant}$$

$\ln P(Y^i \mid X^i)$, dropping the terms $\ln P(X^i)$; this is called *maximum conditional likelihood estimation*, and as we can see from above, it is equivalent to MLE if we assume that $P(X^i)$ doesn't depend on our parameters.)

P(X)全部 data 的概率(已知)，是个常数，跟参数 w，b 独立(假设)

minimize L 其实就是 minimize 第一项 square，就是 least square!

最终得到参数，得到 conditional model P(Y|X)

又被称为 MCLE

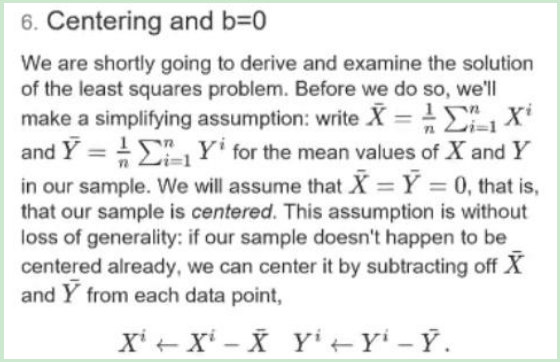
Solution of least square

Center and Scale

因为是 regression，需要对 X Y 都 standardized

center 数据归 0 化

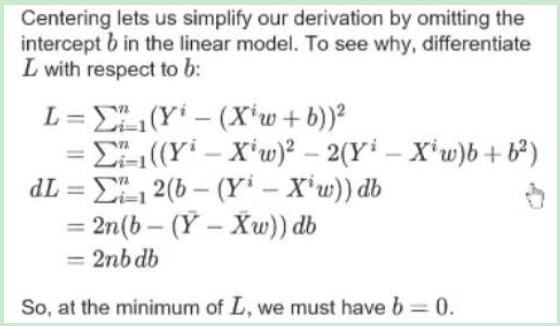
X Y 减去其均值后其合为 0，均值也是 0



6. Centering and b=0

We are shortly going to derive and examine the solution of the least squares problem. Before we do so, we'll make a simplifying assumption: write $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X^i$ and $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y^i$ for the mean values of X and Y in our sample. We will assume that $\bar{X} = \bar{Y} = 0$, that is, that our sample is *centered*. This assumption is without loss of generality: if our sample doesn't happen to be centered already, we can center it by subtracting off $\bar{X}$ and $\bar{Y}$ from each data point,

$$X^i \leftarrow X^i - \bar{X} \quad Y^i \leftarrow Y^i - \bar{Y}.$$



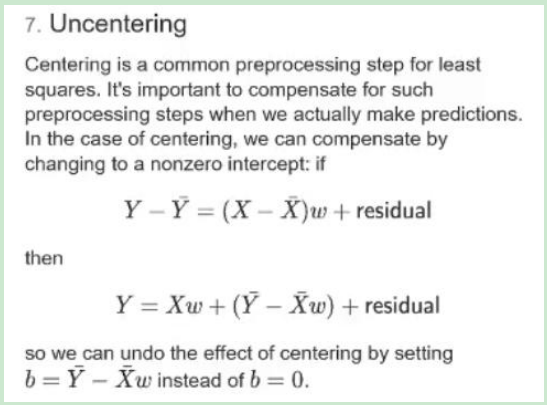
Centering lets us simplify our derivation by omitting the intercept b in the linear model. To see why, differentiate L with respect to b:

$$\begin{aligned} L &= \textstyle\sum_{i=1}^{n}(Y^i - (X^i w + b))^2 \\ &= \textstyle\sum_{i=1}^{n}((Y^i - X^i w)^2 - 2(Y^i - X^i w)b + b^2) \\ dL &= \textstyle\sum_{i=1}^{n} 2(b - (Y^i - X^i w))\, db \\ &= 2n(b - (\bar{Y} - \bar{X}w))\, db \\ &= 2nb\, db \end{aligned}$$

So, at the minimum of L, we must have $b = 0$.

上面的和转成均值，X,Y 均值都是 0，Loss function 导数 b 永远为 0，所以 b 就可以为 0

当 prediction 是需要 compensate：预测前也要减去 XY 均值



7. Uncentering

Centering is a common preprocessing step for least squares. It's important to compensate for such preprocessing steps when we actually make predictions. In the case of centering, we can compensate by changing to a nonzero intercept: if

$$Y - \bar{Y} = (X - \bar{X})w + \text{residual}$$

then

$$Y = Xw + (\bar{Y} - \bar{X}w) + \text{residual}$$

so we can undo the effect of centering by setting $b = \bar{Y} - \bar{X}w$ instead of $b = 0$.

其实就增加了一个 $\bar{Y} - \bar{X}w$ 作为 b

最常见的错误：

It's important in this case to compute $\bar{X}$ and $\bar{Y}$ on the *training data* only. That is, we don't want to try to use our test data to help estimate these means: doing so would require us to know our test data before making a prediction, which doesn't make sense. (And if we do know our test data ahead of time for some reason, then using it this way could give us incorrect answers. E.g., if we're trying to use separate test data to measure how well our learned regression model can predict Y, then this unrealistic use of the test data can cause us to get an inaccurate estimate of our MSE.)

只能在 train 上计算 X Y 的均值！！！

如果用了 test，模型就包含了 test 的信息！

8. The least squares slope

We can now solve for the value of w that minimizes squared error. This is a special case of a derivation that we've done before, but it helps to write out the univariate case for intuition.

Write $\mathbf{X}$ for the vector whose ith element is X^i, and write $\mathbf{Y}$ for the vector whose ith element is Y^i. Then

$$\begin{aligned} L &= \tfrac{1}{2}\|\mathbf{Y} - \mathbf{X}w\|^2 \\ &= \tfrac{1}{2}(\mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T\mathbf{X}w + \mathbf{X}^T\mathbf{X}w^2) \end{aligned}$$

and we can find the least squares slope by setting the derivative of L with respect to w to 0:

$$\begin{aligned} dL &= (-\mathbf{Y}^T\mathbf{X} + \mathbf{X}^T\mathbf{X}w)\,dw \\ w &= \mathbf{Y}^T\mathbf{X} \,/\, \mathbf{X}^T\mathbf{X} \end{aligned}$$

least square 的 solution 就是直线的 slope

对 loss function 对 w 求导，令 0，得到 w

Since we have assumed $\bar{X} = \bar{Y} = 0$, we have $\frac{1}{n}\mathbf{X}^T\mathbf{X} = \mathrm{Var}(X)$ and $\frac{1}{n}\mathbf{Y}^T\mathbf{X} = \mathrm{Cov}(X, Y)$. So, another way to write the least squares slope is

$$w = \mathrm{Cov}(X, Y) \,/\, \mathrm{Var}(X)\,.$$

In words, the least squares slope is proportional to the covariance between inputs and outputs, and inversely proportional to the variance of the inputs.

又 XY 均值都是 0，w 可写成方差和协方差的形式

9. Scaling X and Y

It is easy to check that scaling X or Y will scale w to compensate — as if we stretched the entire plot horizontally or vertically. For example, if we scale

$$X^{\text{new}} = kX$$

then, since $\text{Var}(kX) = k^2 \text{Var}(X)$ and $\text{Cov}(kX, Y) = k\,\text{Cov}(X, Y)$,

the slope updates as

$$w^{\text{new}} = \frac{\text{Cov}(kX,Y)}{\text{Var}(kX)} = \frac{k}{k^2}\frac{\text{Cov}(X,Y)}{\text{Var}(X)} = \frac{w}{k}\,.$$

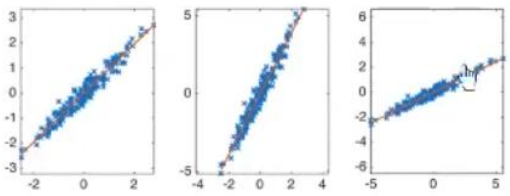
The factor $1/k$ in w^{new} cancels the factor k in X^{new}, meaning that $X^{\text{new}} w^{\text{new}} = Xw$. A similar derivation holds if we scale Y.

Scale X 会影响 w 的值！！也就是斜率会变

scale X 会使得 w 反向的 scale

Scale Y 会使得 w 相同的 scale

Here are some examples of scaling X and Y:



From left to right, the plots show the original X and Y; leaving X alone while scaling $Y \leftarrow 2Y$; and leaving Y alone while scaling $X \leftarrow 2X$.

standardize:

10. Interpreting w

Because the least squares slope changes predictably when we rescale X and Y, we can use rescaling to help us understand how regression behaves. In particular, it will help to scale so that $\text{Var}(X) = \text{Var}(Y) = 1$. We call the resulting quantities (which have mean zero and variance 1) the *standardized* versions of X and Y.

With X and Y standardized, the above expression for w simplifies to

$$w = \text{Cov}(X, Y) = \text{Corr}(X, Y)\,.$$

That is, when X and Y are standardized, the slope of the least squares fit is exactly the correlation between X and Y. This observation, together with our earlier discussions of centering and scaling, leads to perhaps the best way to understand univariate linear regression:

实际 slope 就是 XY 的 corr 相关系数(XY 方差都是 1)

The regression fit is a line whose slope is the correlation between X and Y, scaled to compensate for the actual variances of X and Y, and shifted to compensate for the actual means of X and Y.

slope 就是 correlation！！

XY 的方差会 scale w

XY 的均值会使得线截距

outliner+heteroscedastic+nonlinearity

如果 data 不是 linear-高斯分布，就会有问题

11. Problems to watch out for in linear regression

In practice, linear models and least squares often work quite well even when our data are not linear-Gaussian. There are three common problems that can trip us up, though: *outliers*, *heteroscedasticity*, and of course *nonlinearity*.

1 outliner: high residual

If the conditional distribution of residuals is heavy-tailed, then we will often see a few residuals that are much larger than all of the others. The corresponding points are called *outliers*. (Note: an outlier in this context is an

outliner 很影响 regression

Unfortunately, outliers can dominate the result of least squares: to see why, note that the optimal w makes $\frac{dL}{dw} = 0$. We can split up the derivative of L into one term per example:

$$0 = \frac{dL}{dw} = \sum_i \frac{d}{dw}(Z^i)^2/2$$

where Z_i is the ith residual. If one of the terms in the sum is much larger than the others, then it can greatly influence the place where the sum is zero. Since the derivative of squared loss grows as Z^i grows, outliers will have a disproportionate influence on the optimal value of w.

会对 loss function 产生影响

2 heteroscedastic

If all conditional distributions $P(Y^i \mid X^i)$ are Gaussian (or close to Gaussian), outliers will be rare, since the Gaussian distribution has light tails. Even in this case, it might happen that the variances of the different conditional distributions are different. Distributions with different variance are called *heteroscedastic*. Heteroscedasticity leads to a similar problem as outliers: the higher-variance points can dominate the result of least squares.

数据确实服从高斯分布，但可能是从多个方差不同分布的来

如果高斯分布的 var 太大，也会有很多 outlier

或者是 有两个峰的分布

3 不是线性关系

Finally, if the relationship between X and Y is not linear, we can still sometimes discover qualitative properties like the overall direction of the relationship; but clearly our predictions would become more accurate if we took account of the nonlinearity. Nonlinearity can sometimes masquerade as outliers or heteroscedasticity: as the true value of $E(Y^i \mid X^i)$ gets closer to or farther from the best linear model, it makes the residuals appear smaller or larger.

针对上面问题，诊断方法：residual plot(Y hat vs residual)

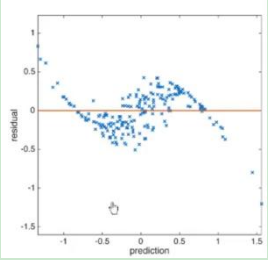
可以帮助我们：

检验 1 outliner 以及 2 residual 是否符合高斯分布假设 3 heteroscedastic 4 non-linearity(curve 而不是直线)

To diagnose problems with a linear model, it's often a good idea to make a residual plot — that is, a scatter plot of prediction $\hat{Y}^i$ vs. residual Z^i. A residual plot can help us discover whether our residuals follow a Gaussian distribution; in particular, it can reveal the presence of outliers. (We can also discover outliers by plotting a histogram of Z^i, or by explicitly estimating a CDF of Z^i via kernel density estimation.)

A scatter plot can also help us discover heteroscedasticity, by showing whether the typical size of residuals changes for different values of $\hat{Y}_i$. Finally, a scatter plot can help us discover some kinds of nonlinearities: in this case the residuals will tend to be positive in some areas and negative in others, so that the scatter plot looks like a curve instead of a straight line.

non-linear 非线性关系



x 是 Y hat　y 是 Y-Y hat(residual)

Y 预测的越小，真实值越大，Y 预测的越大，真实值越小

本来是应该是条直线，**所有预测值 residual 差不多**而现在是根据预测值的不同，residual 不同

We've described residual plots for univariate regressions, but these plots also work well in multivariate regression. In fact, they can be even more important there, since it's harder to visualize what's going on when there are more than a few dimensions.

fix outlier

13. Fixing outliers

There are a couple of well-known heuristics that can help reduce the effects of outliers. The first is to use a *robust* loss function: one that is not as sensitive to large residuals. For example, we could use the *absolute loss*, $\ell(z) = |z|$, or the *Huber loss*,

$$\ell(z) = \begin{cases} z^2 & |z| \leq 1 \\ 2|z| - 1 & \text{otherwise} \end{cases}.$$

These loss functions help deal with outliers because their derivative stays bounded even for large values of z. (Recall that large derivatives can strongly perturb the place where $\frac{dL}{dw} = 0$.) There are efficient algorithms to minimize both Huber loss and absolute loss, although not quite so efficient as for least squares.

对 loss 改进，是的 outlier 不影响它，不会很大,如果 residual 用另外的 loss

最好是 remove residual：

The second way to reduce the effects of outliers is simply to remove them. Of course, we don't know ahead of time which points are outliers. But, it often works to use an iterative procedure to identify them: first we fit a linear model to all of our data. Then we sort our points by the absolute value of their residual, and remove the top few (the worst few outliers). Then we re-fit our model, and iterate: keep removing outliers a few at a time until the remaining residuals are all small enough.

识别 outlier: 1 fit a liner 2 将其 residual 的绝对值排序

3 remove top few 4 refit--循环上面几次 直到 residual 都不是很大

可以通过 qq plot 决定什么时候 stop

This procedure works as long as the outliers are not so common or so large as to completely alter the regression line: as long as the initial fit is moderately close to correct, we can still use it to identify outliers. We can improve the robustness of this procedure by working slowly: we only remove a few outliers at a time, then re-fit the line so that we can more accurately identify remaining outliers. (Of course, going too slowly wastes computation time.) We can decide when to stop by testing to see that the remaining residuals are close to normal (e.g., using a Q-Q plot), or by cross-validation (which we'll describe in more detail later).

上面得到的最终线性模型，可以用于对 test 的 outlier 的检测

如果 test 的 true Y 跟带入模型得到的 Y hat 在+-3sigma 之外，

就说明是这个 true Y 是 outlier

If desired, once the above iteration converges, we can use it to decide on an outlier detection rule for test data: for example, any test point that falls more than $\pm 3\sigma$ from our prediction $\hat{Y}$ could be considered an outlier. (This could be useful, for example, when we are evaluating the performance of our learned prediction rule.) As with the centering procedure described above, we need to be sure to design our outlier detection rule *using the training data only*: e.g., if we somehow remove outliers from the test set before making predictions, we'll get an unrealistically low estimate of MSE.

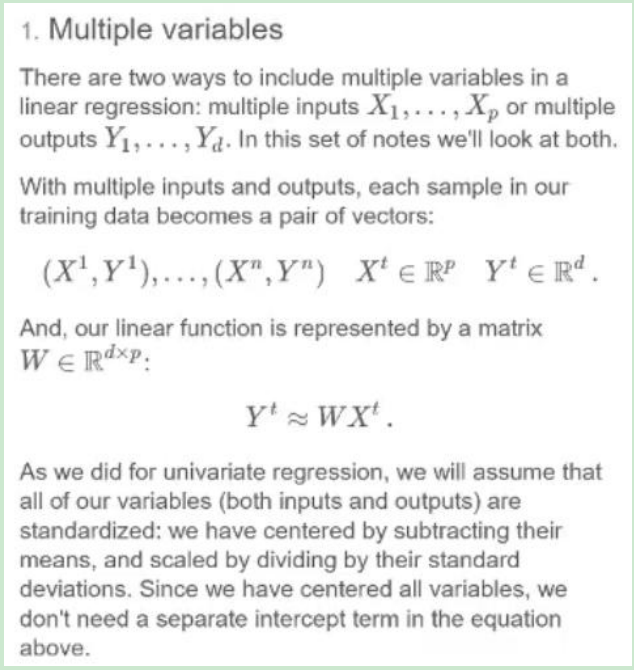
Fix heteroscedasticity 或 nonlinearity

14. Fixing heteroscedasticity or nonlinearity

The fixes for heteroscedasticity and nonlinearity are similar to each other: we want to transform our data and/or add new features so that a linear model with constant residual variance becomes a better fit. We'll discuss feature transforms and feature generation later, when we talk about multivariate regression.

transform 或 add new feature

Multiple variables(多元回归)



1. Multiple variables

There are two ways to include multiple variables in a linear regression: multiple inputs $X_1, \ldots, X_p$ or multiple outputs $Y_1, \ldots, Y_d$. In this set of notes we'll look at both.

With multiple inputs and outputs, each sample in our training data becomes a pair of vectors:

$$(X^1, Y^1), \ldots, (X^n, Y^n) \quad X^t \in \mathbb{R}^p \quad Y^t \in \mathbb{R}^d.$$

And, our linear function is represented by a matrix $W \in \mathbb{R}^{d \times p}$:

$$Y^t \approx W X^t.$$

As we did for univariate regression, we will assume that all of our variables (both inputs and outputs) are standardized: we have centered by subtracting their means, and scaled by dividing by their standard deviations. Since we have centered all variables, we don't need a separate intercept term in the equation above.

X 或 Y 都是可以是多个变量(线性神经网络)

每个 example(XY)都是 a pair of vector(两个 vector 组成：X，Y)

W 此时是矩阵！！

我们已经假设了所有变量 XY 都标准化了，都 centered，所以没有 intercept

2. Least squares

Just as for univariate regression, we'd like to find the value of W that minimizes our total loss — typically squared error, but we can also use other loss functions such as Huber or absolute loss. That is, we want to minimize

$$L = \frac{1}{2}\sum_t \|Y^t - \hat{Y}^t\|^2 = \frac{1}{2}\sum_t \|Y^t - WX^t\|^2 .$$

Write $\mathbf{X} \in \mathbb{R}^{p\times n}$ for the matrix whose columns are the inputs X^t, and write $\mathbf{Y} \in \mathbb{R}^{d\times n}$ for the matrix whose columns are the outputs Y^t. Then we have

$$L = \frac{1}{2}\|\mathbf{Y} - W\mathbf{X}\|_F^2$$

where $\|\cdot\|_F^2$ is the squared *Frobenius* norm, i.e., the sum of squared elements of a matrix. Another way to write this expression is

$$\begin{aligned} L &= \frac{1}{2}\mathrm{tr}((\mathbf{Y} - W\mathbf{X})^T(\mathbf{Y} - W\mathbf{X})) \\ &= \frac{1}{2}\mathrm{tr}(\mathbf{Y}^T\mathbf{Y} - 2\mathbf{Y}^T W\mathbf{X} + \mathbf{X}^T W^T W\mathbf{X}) \end{aligned}$$

可以将**所有 t 个 exmaple X Y 向量**变成**矩阵**

每个 example 的 loss 定义：Y 向量之差的 L2 norm，cost 就是所有 example loss 之合

多个向量 L2 norm 之合可以转化成**矩阵 Frobenius norm ，矩阵每一列都是向量**

squared **Frobenius norm 有个规律，等于展开矩阵相乘后的 trace！！**

In the first line we used the identity $\|Z\|_F^2 = \mathrm{tr}(Z^T Z)$, which we mentioned in the first mini (and which is easy to verify by writing out both sides as sums over the components of Z).

3. Solving for W

To find the optimal W, we can differentiate and set to 0:

$$\begin{aligned} 0 &= dL \\ &= \tfrac{1}{2}\mathrm{tr}(-2\mathbf{Y}^T dW\,\mathbf{X} + 2\mathbf{X}^T W^T dW\,\mathbf{X}) \\ &= \mathrm{tr}((-\mathbf{X}\mathbf{Y}^T + \mathbf{X}\mathbf{X}^T W^T) dW) \\ W\mathbf{X}\mathbf{X}^T &= \mathbf{Y}\mathbf{X}^T \end{aligned}$$

In the third line we used *trace rotation*, $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, which we also discussed in the first mini (and which is also easy to verify by writing out both sides as sums).

trace 是 linear operator，and d 可以穿透

The last line above is a system of linear equations for W, called the *normal equations*. So, linear regression is only as hard as constructing and solving a system of linear equations.

As always with linear systems, it's not usually a good idea to solve the normal equations by inverting $\mathbf{X}\mathbf{X}^T$ explicitly; instead we can use any standard technique for solving linear systems. A good one is to factor $\mathbf{X}\mathbf{X}^T = LL^T$ where L is lower triangular. Then we can backsolve for each row of W separately: each row of W depends only on the corresponding row of $\mathbf{Y}\mathbf{X}^T$.

一般情况下会 LUD 分解，不用逆矩阵

每一行的 W 是独立的，可以单独去解

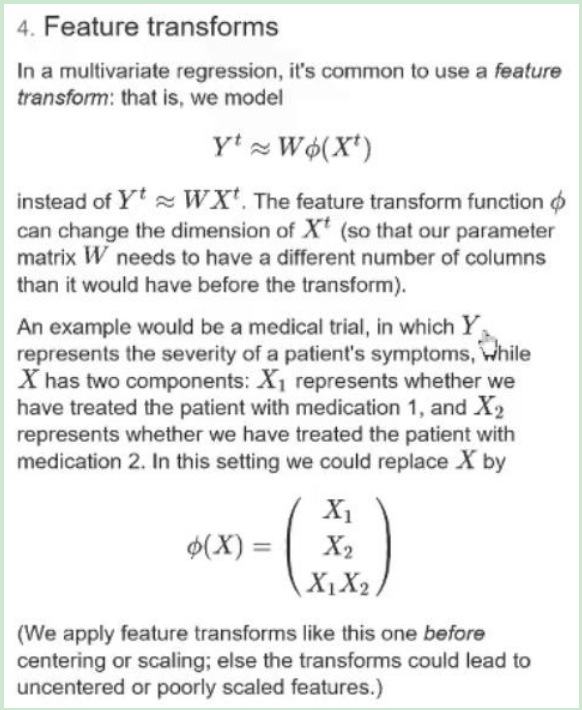
Feature transform

注意，transform 要放在第一位！！再标准化

如果归 0 化以后，再 transform，会影响归 0 化！！！

对于 test data，也要同样 feature transform

cross feature



4. Feature transforms

In a multivariate regression, it's common to use a *feature transform*: that is, we model

$$Y^t \approx W\phi(X^t)$$

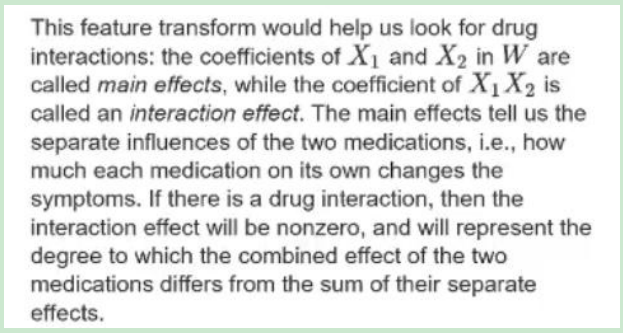
instead of $Y^t \approx WX^t$. The feature transform function ϕ can change the dimension of X^t (so that our parameter matrix W needs to have a different number of columns than it would have before the transform).

An example would be a medical trial, in which Y represents the severity of a patient's symptoms, while X has two components: X_1 represents whether we have treated the patient with medication 1, and X_2 represents whether we have treated the patient with medication 2. In this setting we could replace X by

$$\phi(X) = \begin{pmatrix} X_1 \\ X_2 \\ X_1X_2 \end{pmatrix}$$

(We apply feature transforms like this one *before* centering or scaling; else the transforms could lead to uncentered or poorly scaled features.)

cross feature 可以增加 interaction effect



This feature transform would help us look for drug interactions: the coefficients of X_1 and X_2 in W are called *main effects*, while the coefficient of X_1X_2 is called an *interaction effect*. The main effects tell us the separate influences of the two medications, i.e., how much each medication on its own changes the symptoms. If there is a drug interaction, then the interaction effect will be nonzero, and will represent the degree to which the combined effect of the two medications differs from the sum of their separate effects.

系数：X1 X2 main effect　X1*X2 interaction effect

经过非线性 transform 后，最终还是 linear model

feature selection

另一种 transform 是 feature selection：

Another example of a useful feature transform is *variable selection*: if some components of X are not helpful for predicting Y, then we can use $\phi(X)$ to drop them. For example, if $X \in \mathbb{R}^{42}$, we could take

$$\phi(X) = \begin{pmatrix} X_3 \\ X_{17} \\ X_{25} \end{pmatrix}$$

Variable selection can help if X is high-dimensional: it lets us avoid spending data to estimate the coefficients of irrelevant features. (Of course, it might be hard to tell ahead of time which features are irrelevant — so, there are lots of algorithms that try to help us decide. For example, two such algorithms are Lasso and forward stagewise regression.)

throw away some feature : lasso or forward stagewise

feature transform 需要自己手动去 try

Non-linear transform

5. Nonlinear transforms

One of the most important uses of a feature transform is to handle the case when $E(Y \mid X)$ is a *nonlinear* function of X — for example, if $X, Y \in \mathbb{R}$, we could model

$$E(Y \mid X) = X^2 - 3X + 1 .$$

To learn a nonlinear function like this one, we use a feature transform to replace the single input X by multiple inputs, each one of which is a transformed version of X. For example, the transform

$$\phi(X) = \begin{pmatrix} X \\ X^2 \end{pmatrix}$$

will let us learn a quadratic function like the one above. We could also include a constant feature, making $\phi(X) = (1, X, X^2)^T$. The constant feature would be necessary to fit a general quadratic function if we weren't centering our data, but since we are centering, the constant feature is redundant.

如果没有 center X，constant 1 也是必要的

If there are multiple inputs, each feature can depend on any subset of them: e.g.,

$$\phi(X) = \begin{pmatrix} X_1 \\ X_2 \\ X_1^2 \\ X_2^2 \\ X_1 X_2 \end{pmatrix}$$

lets us learn a quadratic function on two inputs.

当 Y 是 X 的非线性函数，也就是 E(Y|X)给定 X 下 Y 的期望是非线性

也可以用 transformation，然后再 learn linear model

Even after a potentially-nonlinear feature transform, we still call the problem of estimating W "linear regression," for two reasons: our prediction is linear in our parameters W (or alternately in our features $\phi(X)$); and, algorithms that find W tend to work equally well whether we start from X or $\phi(X)$.

Dummy coding

6. Dummy coding

Another important example of a feature transform is *dummy coding*. Dummy coding is designed to handle categorical inputs: e.g., a day of the week, a gender, or a U.S. state. Often, when we receive a data set, each categorical input is represented as a small integer: say 1 = Alabama, 2 = Alaska, 3 = Arizona, etc. Linear regression will produce unexpected results if we leave categorical inputs represented this way: suppose we learn a weight w for such an input. Then, moving from Alamaba to Alaska produces a change of w in $E(Y \mid X)$ (which is fine). But now moving from Arizona to Alaska has to produce a change of $-w$ (which is not fine: there's no reason for these changes to be related in this way). In fact, for any three states, knowing $E(Y \mid X)$ for two of them will completely constrain the third.

To avoid this behavior, dummy coding expands a single categorical input into several binary inputs, each one corresponding to a possible value of the original categorical input. In the states example, we would construct a binary feature that's 1 for Alabama and zero otherwise; another that's 1 for Alaska and 0 otherwise; and so on. After dummy coding, we learn one weight for Alabama, another for Alaska, and so on; therefore, we are not enforcing any particular relationship among the predictions for different states.

When we construct a dummy code, we will typically choose one of the possible values as a "default" value, and omit its corresponding dummy variable. This practice can help with interpretation, and it also helps avoid the problem of rank deficiency, discussed below.

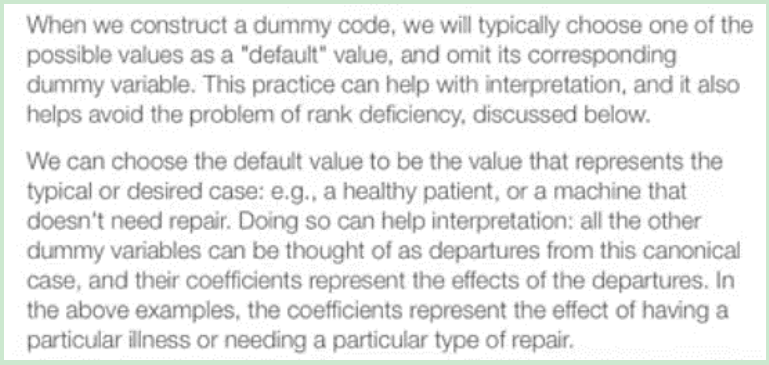
We can choose the default value to be the value that represents the typical or desired case: e.g., a healthy patient, or a machine that doesn't need repair. Doing so can help interpretation: all the other dummy variables can be thought of as departures from this canonical case, and their coefficients represent the effects of the departures. In the above examples, the coefficients represent the effect of having a particular illness or needing a particular type of repair.

不可以对分类变量进行 0,1,2,3,4 编号

会学习到各个分类的大小，但每个分类是平等地位

需要 onehot-encoding，一个 feature 转成多个 feature

default value



When we construct a dummy code, we will typically choose one of the possible values as a "default" value, and omit its corresponding dummy variable. This practice can help with interpretation, and it also helps avoid the problem of rank deficiency, discussed below.

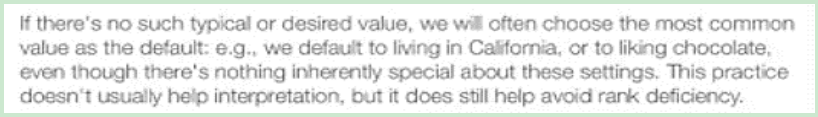
We can choose the default value to be the value that represents the typical or desired case: e.g., a healthy patient, or a machine that doesn't need repair. Doing so can help interpretation: all the other dummy variables can be thought of as departures from this canonical case, and their coefficients represent the effects of the departures. In the above examples, the coefficients represent the effect of having a particular illness or needing a particular type of repair.

choose default value 然后去掉这一列

dumy code 每一行所有 feature 的和是 1

做完归一化之后，**每一行所有 feature 和是 0，feature 之间 linear dependent ，这样回归会有问题**

just drop one

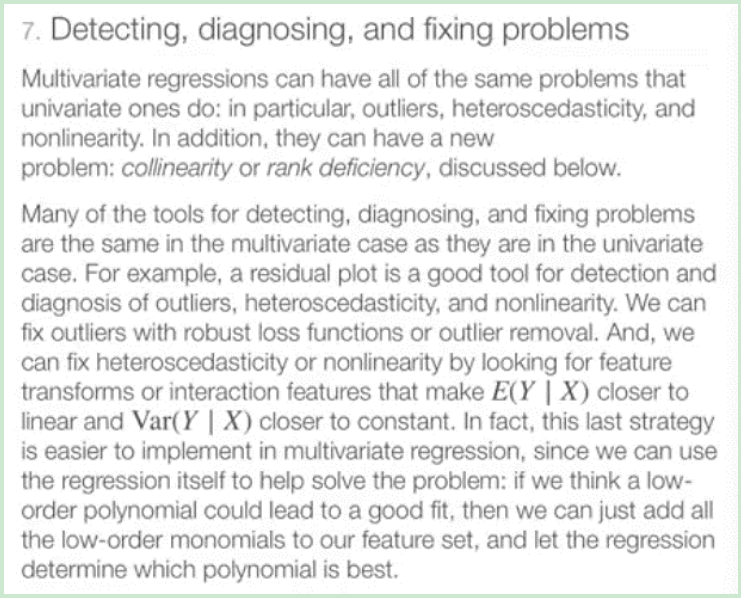


If there's no such typical or desired value, we will often choose the most common value as the default: e.g., we default to living in California, or to liking chocolate, even though there's nothing inherently special about these settings. This practice doesn't usually help interpretation, but it does still help avoid rank deficiency.

choose most common value

question 具体怎么用？

Detecting diagnose

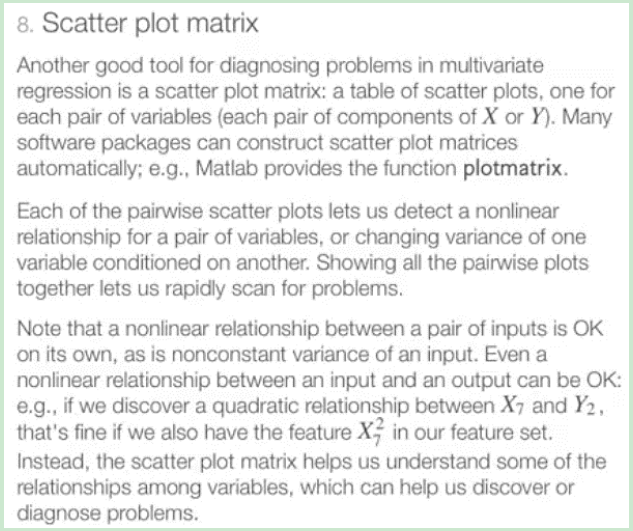


7. Detecting, diagnosing, and fixing problems

Multivariate regressions can have all of the same problems that univariate ones do: in particular, outliers, heteroscedasticity, and nonlinearity. In addition, they can have a new problem: *collinearity* or *rank deficiency*, discussed below.

Many of the tools for detecting, diagnosing, and fixing problems are the same in the multivariate case as they are in the univariate case. For example, a residual plot is a good tool for detection and diagnosis of outliers, heteroscedasticity, and nonlinearity. We can fix outliers with robust loss functions or outlier removal. And, we can fix heteroscedasticity or nonlinearity by looking for feature transforms or interaction features that make $E(Y \mid X)$ closer to linear and $\mathrm{Var}(Y \mid X)$ closer to constant. In fact, this last strategy is easier to implement in multivariate regression, since we can use the regression itself to help solve the problem: if we think a low-order polynomial could lead to a good fit, then we can just add all the low-order monomials to our feature set, and let the regression determine which polynomial is best.

feature transform 对于处理 hetero 和 nonlinearity 有效



8. Scatter plot matrix

Another good tool for diagnosing problems in multivariate regression is a scatter plot matrix: a table of scatter plots, one for each pair of variables (each pair of components of X or Y). Many software packages can construct scatter plot matrices automatically; e.g., Matlab provides the function **plotmatrix**.

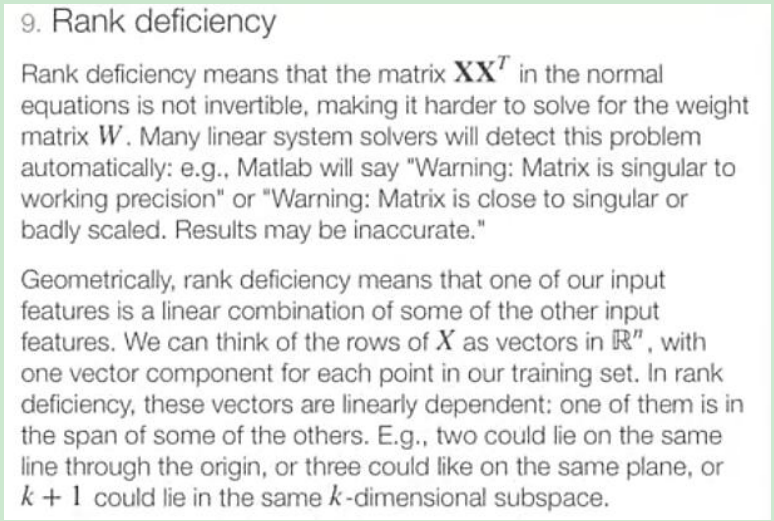
Each of the pairwise scatter plots lets us detect a nonlinear relationship for a pair of variables, or changing variance of one variable conditioned on another. Showing all the pairwise plots together lets us rapidly scan for problems.

Note that a nonlinear relationship between a pair of inputs is OK on its own, as is nonconstant variance of an input. Even a nonlinear relationship between an input and an output can be OK: e.g., if we discover a quadratic relationship between X_7 and Y_2, that's fine if we also have the feature X_7^2 in our feature set. Instead, the scatter plot matrix helps us understand some of the relationships among variables, which can help us discover or diagnose problems.

每一个 X,Y 都两两的作散点图，看有没有 non-linear

如果发现了某个 X 和 Y 的二次关系，就可以包含该 X 平方作为 feature

Rank deficiency



9. Rank deficiency

Rank deficiency means that the matrix $\mathbf{X}\mathbf{X}^T$ in the normal equations is not invertible, making it harder to solve for the weight matrix W. Many linear system solvers will detect this problem automatically: e.g., Matlab will say "Warning: Matrix is singular to working precision" or "Warning: Matrix is close to singular or badly scaled. Results may be inaccurate."

Geometrically, rank deficiency means that one of our input features is a linear combination of some of the other input features. We can think of the rows of X as vectors in $\mathbb{R}^n$, with one vector component for each point in our training set. In rank deficiency, these vectors are linearly dependent: one of them is in the span of some of the others. E.g., two could lie on the same line through the origin, or three could like on the same plane, or $k+1$ could lie in the same k-dimensional subspace.

不是可逆矩阵，不能直接解方程

rank deficiency 本质原因：其中一个 feature 可以被某些其他的 feature 线性表示

delete feature 最好，找到最少的 feature

Rank deficiency can happen if we accidentally add a duplicate feature in our feature transform: e.g., two copies of X_1. More subtly, the rank deficiency can be spread out across several features. For example, if we make a complete dummy code for a categorical feature (i.e., if we don't omit the variable for the default or most common value of the feature), then the sum of all of the dummy variables is constant. So, after centering, the sum of the dummy variables is zero for every example, meaning that the set of dummy variables is linearly dependent.

如果不删掉一个 dumming feature 就会出现上面问题

Another source of rank deficiency is insufficient data. For example, we could have a binary feature that is more often 1 than 0: say, "likes chocolate". If we don't have a very large sample, we might happen to see only people who like chocolate, in which case the feature appears constant in our sample (even though it

is not constant in the larger world). Similarly, we could have two binary features that are correlated: perhaps "likes pecans" and "likes walnuts". Then, even if we have enough data that we see both values of each feature, we might never see an example where they differ. So, these two features will be linearly dependent in our sample (even though they are not dependent in the larger world).

2 train set 太少,使得一个 feature 全部一个值

10. Fixing rank deficiency

Rank deficiency can cause two problems with a linear regression. The first is computational: we have trouble finding the optimal W, and in fact there may be more than one optimal W. This problem is relatively easy to fix: we can simply drop features (rows of $\mathbf{X}$) in an arbitrary order until the problem goes away ($\mathbf{X}$, and therefore $\mathbf{X}\mathbf{X}^T$, is full rank). Then, for each dropped feature, we set the corresponding column of W to zero.

A more complicated strategy that can work better in some cases is to use an SVD: we factor $\mathbf{X} = USV^T$, and drop the zero diagonal elements of S along with the corresponding columns of U and V. This strategy can cause fill-in if $\mathbf{X}$ is a sparse matrix; but, it can also make the resulting normal equations better-conditioned.

drop feature 或者 矩阵分解

The second problem is with interpretation: if two or more input features are linearly dependent, then there's no way to determine which subset of them is "truly" associated with the outputs. It's worth stressing that this is *only* a problem with interpretation: our predictions are the same no matter which subset of features we use in our regression. But, since one of the common uses of regression is to discover associations among variables, this problem can be troublesome.

Fixing the interpretation problem can be hard: it requires us to understand the underlying meaning or cause of the rank deficiency. Sometimes we can quickly discover a bug in our feature construction, such as accidentally including the dummy variables for all values of a categorical variable instead of omitting the default value. Other times we are stuck trying to figure out more about the underlying meanings of our features, a process that is inherently hard to automate.

Interpretation can actually be a problem even if our features are only *approximately* linearly dependent instead of exactly linearly dependent: in this case we will need very large amounts of data to resolve the component of W along any direction W_0 such that $W_0\mathbf{X} \approx 0$. We can diagnose approximate rank deficiency via an SVD: we look for small diagonal elements of S instead of zero ones. Fixing approximate rank deficiency follows the same process as fixing exact rank deficiency: if we can't find a bug in feature construction, we are stuck trying to figure out more about the underlying meanings of our features.

当**矩阵接近 not inverse**(not full rank)是也比较麻烦，需要大量数据

standardize

11. Standardizing, revisited

It's important to remember that we standardize using means and variances calculated from the training data: we don't know the means and variances of our test data ahead of time, so it doesn't make sense to include them in our learned linear function.

In more detail, at training time, we first compute the training set means and standard deviations: $\bar{X}_i$, σ_i^X, $\bar{Y}_j$, σ_j^Y. We then standardize the training data:

$$X_i^t \leftarrow (X_i^t - \bar{X}_i)/\sigma_i^X \qquad Y_j^t \leftarrow (Y_j^t - \bar{Y}_j)/\sigma_j^Y$$

for $i \in 1 \ldots p$, $j \in 1 \ldots d$, $t \in 1 \ldots n$. Finally, we record the means and variances so that we can use them later for prediction.

At test time, when a new input vector X comes in, our first step is to standardize: $X_i \leftarrow (X_i - \bar{X}_i)/\sigma_i^X$ just as above. (Here $\bar{X}_i$ and σ_i^X are the values we recorded from the training data.) Then we multiply by W to get a prediction of the standardized version of Y. Finally we reverse the standardization for Y:

$$Y_j \leftarrow \sigma_j^Y Y_j + \bar{Y}_j$$

(again using the mean and standard deviation from the training data) so that we have a prediction of the unstandardized version of Y.

多元，每个 feature，单独做 standardize

只能用 train 的信息来做标准化，同样对 test 做标准化

记录 train set 的 mean 和 std

如果是 regression，需要将 Y 变换成我们原本的 scale，乘以 std+mean

前面都是 regression

model choosing

1. Model choice

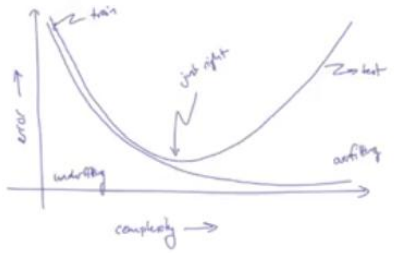
In the last few sets of notes, we've discussed the different choices to be made in designing a conditional linear model. Some of these choices are:

- What data to collect, and what to measure about each example.
- How to clean up the data: remove outliers, fill in missing feature values, or more complex processing.
- What feature transforms to use: which subset of features to include, which interaction terms, which nonlinearities.
- Which loss function to use: squared loss, absolute loss, Huber loss, etc.
- What type of regularization to use, and what value of the regularization parameter.
- Whether and how to use automatic feature selection, such as forward stagewise regression.

Often there's no good way to make these choices ahead of time. So, we choose based on data: we look at what choices lead to accurate predictions on the data we have. Unfortunately, doing so leads to a problem: overfitting.

2. Overfitting

Overfitting is a catch-all term for the problems that result from fitting too complex a model on too little data. The following graph shows the typical symptoms of overfitting: as our model gets more complex, we fit our training data better and better. Initially, the prediction performance on test data tracks the performance on training data fairly well — usually slightly worse than training performance. But, at some point, our performance on new test data stops tracking training performance, and starts to get much worse.



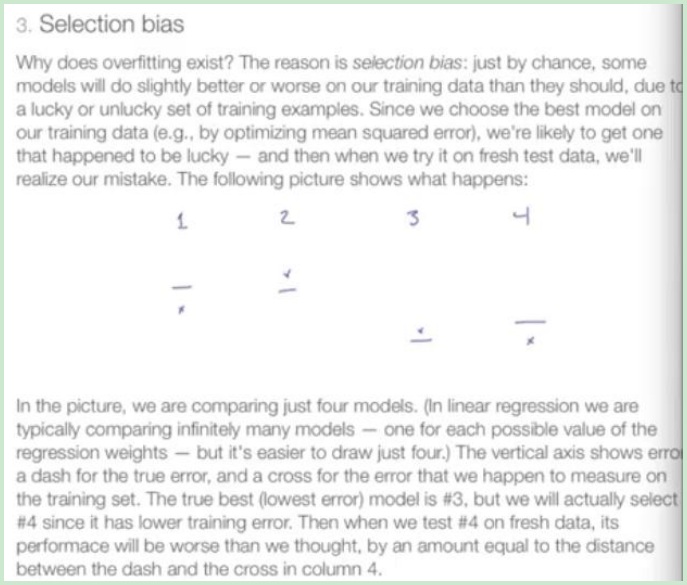
High model complexity can mean things like too many parameters to learn or not enough regularization. Importantly, though, model complexity also includes *all of the choices that we made* while designing our model — including all of the choices listed above, such as which variables to include.

少的数据上 fit 太复杂的 model(太多参数或没有正则化：weight 越

大，预测波动越大）

select bias

choose from different models(choose **perform best**)



3. Selection bias

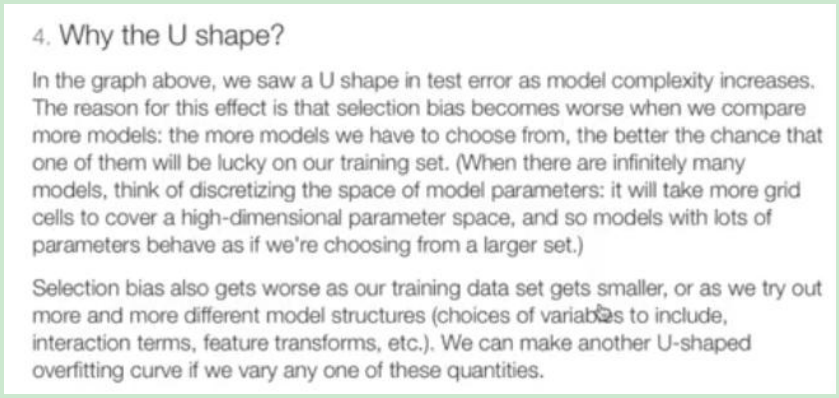
Why does overfitting exist? The reason is *selection bias*: just by chance, some models will do slightly better or worse on our training data than they should, due to a lucky or unlucky set of training examples. Since we choose the best model on our training data (e.g., by optimizing mean squared error), we're likely to get one that happened to be lucky – and then when we try it on fresh test data, we'll realize our mistake. The following picture shows what happens:

In the picture, we are comparing just four models. (In linear regression we are typically comparing infinitely many models – one for each possible value of the regression weights – but it's easier to draw just four.) The vertical axis shows error, a dash for the true error, and a cross for the error that we happen to measure on the training set. The true best (lowest error) model is #3, but we will actually select #4 since it has lower training error. Then when we test #4 on fresh data, its performace will be worse than we thought, by an amount equal to the distance between the dash and the cross in column 4.

selection bias 有些 model **可能幸运的在我们 train 上表现好，我们选择他(不能只用 train set 来检验模型准确性)，但 test 很差**

true error of model --- model 表现的 error **x**

model3 的真实 error 应该是最低，但我们只按照**模型表现的 error**，选 model4，但 model4 实际的 error 高于 model3



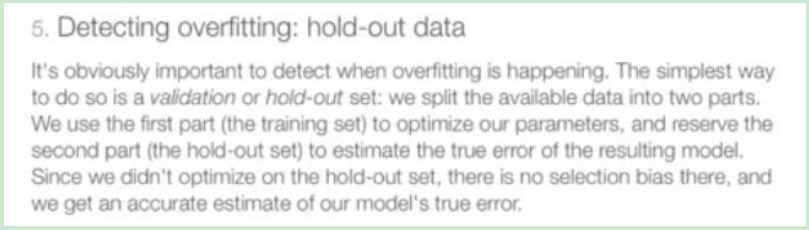
4. Why the U shape?

In the graph above, we saw a U shape in test error as model complexity increases. The reason for this effect is that selection bias becomes worse when we compare more models: the more models we have to choose from, the better the chance that one of them will be lucky on our training set. (When there are infinitely many models, think of discretizing the space of model parameters: it will take more grid cells to cover a high-dimensional parameter space, and so models with lots of parameters behave as if we're choosing from a larger set.)

Selection bias also gets worse as our training data set gets smaller, or as we try out more and more different model structures (choices of variables to include, interaction terms, feature transforms, etc.). We can make another U-shaped overfitting curve if we vary any one of these quantities.

U shape 的原因(随着模型复杂度的增加)model 越 complex，**可选择的模型越多(线性，二次线，三次)**，越会 select bias

hold out 其实就是 split train 和 test，第一部分训练模型。第二部分说明模型真实准确率



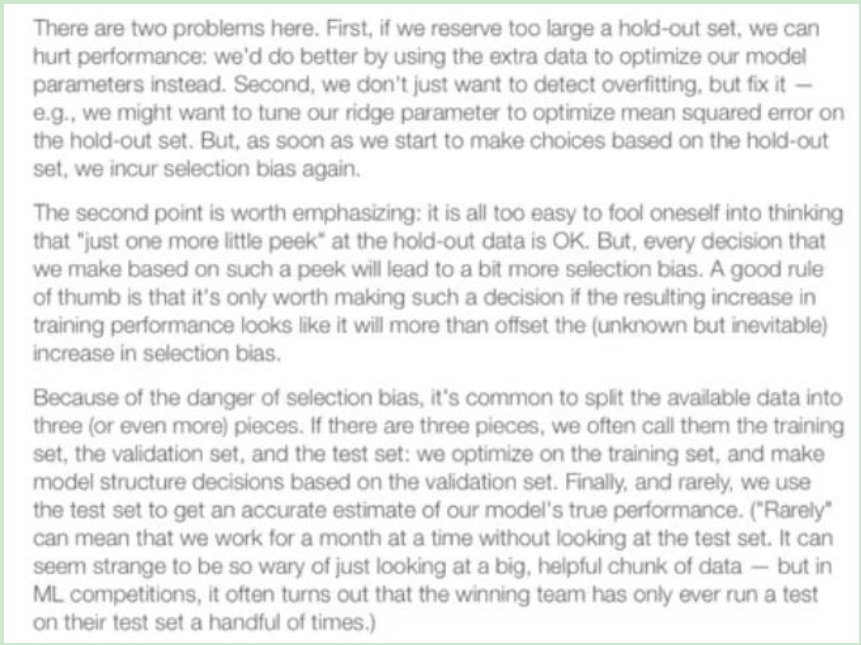
5. Detecting overfitting: hold-out data

It's obviously important to detect when overfitting is happening. The simplest way to do so is a *validation* or *hold-out* set: we split the available data into two parts. We use the first part (the training set) to optimize our parameters, and reserve the second part (the hold-out set) to estimate the true error of the resulting model. Since we didn't optimize on the hold-out set, there is no selection bias there, and we get an accurate estimate of our model's true error.

detect overfit:

用 hold-out 来检测是否过拟合，**计算 model true error**

hold out set 不参与建模，因此**模型不会 lucky**，不存在 selection bias



There are two problems here. First, if we reserve too large a hold-out set, we can hurt performance: we'd do better by using the extra data to optimize our model parameters instead. Second, we don't just want to detect overfitting, but fix it — e.g., we might want to tune our ridge parameter to optimize mean squared error on the hold-out set. But, as soon as we start to make choices based on the hold-out set, we incur selection bias again.

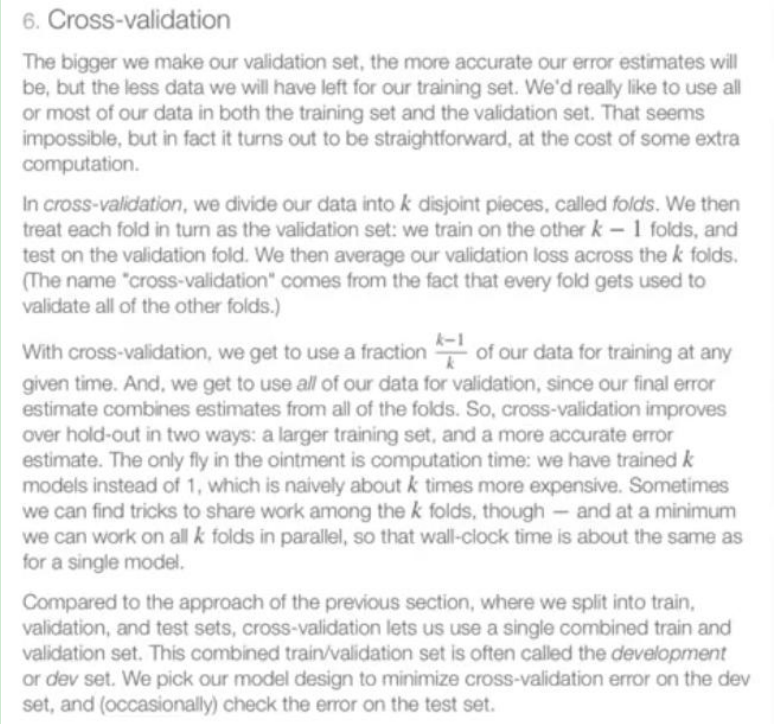
The second point is worth emphasizing: it is all too easy to fool oneself into thinking that "just one more little peek" at the hold-out data is OK. But, every decision that we make based on such a peek will lead to a bit more selection bias. A good rule of thumb is that it's only worth making such a decision if the resulting increase in training performance looks like it will more than offset the (unknown but inevitable) increase in selection bias.

Because of the danger of selection bias, it's common to split the available data into three (or even more) pieces. If there are three pieces, we often call them the training set, the validation set, and the test set: we optimize on the training set, and make model structure decisions based on the validation set. Finally, and rarely, we use the test set to get an accurate estimate of our model's true performance. ("Rarely" can mean that we work for a month at a time without looking at the test set. It can seem strange to be so wary of just looking at a big, helpful chunk of data — but in ML competitions, it often turns out that the winning team has only ever run a test on their test set a handful of times.)

1 hold out set 不能太多，影响建模

2 每用一次 hold out set，就用了**信息**，会 over fit the hold out set，**相当于模型的建立利用了 hold set 的信息，在 test 的准确率就不准了**

更优化的办法：cv



6. Cross-validation

The bigger we make our validation set, the more accurate our error estimates will be, but the less data we will have left for our training set. We'd really like to use all or most of our data in both the training set and the validation set. That seems impossible, but in fact it turns out to be straightforward, at the cost of some extra computation.

In *cross-validation*, we divide our data into k disjoint pieces, called *folds*. We then treat each fold in turn as the validation set: we train on the other $k-1$ folds, and test on the validation fold. We then average our validation loss across the k folds. (The name "cross-validation" comes from the fact that every fold gets used to validate all of the other folds.)

With cross-validation, we get to use a fraction $\frac{k-1}{k}$ of our data for training at any given time. And, we get to use *all* of our data for validation, since our final error estimate combines estimates from all of the folds. So, cross-validation improves over hold-out in two ways: a larger training set, and a more accurate error estimate. The only fly in the ointment is computation time: we have trained k models instead of 1, which is naively about k times more expensive. Sometimes we can find tricks to share work among the k folds, though — and at a minimum we can work on all k folds in parallel, so that wall-clock time is about the same as for a single model.

Compared to the approach of the previous section, where we split into train, validation, and test sets, cross-validation lets us use a single combined train and validation set. This combined train/validation set is often called the *development* or *dev* set. We pick our model design to minimize cross-validation error on the dev set, and (occasionally) check the error on the test set.

最后需要平均模型的表现结果，average validation error

10 fold 好处 1 用了 90% data train model 2 用了所有的 data validite

跟 hold-out 比：更精确的 error，而且 train set 更多

选择 validation error 最小的模型

如何选择最终的 model：

K-FOLD 跑完假设平均准去率我们满意了，这 K 个模型怎么选？选哪个？

7. Picking a final model; model ensembles

After k-fold cross-validation, we have k sets of parameters W_j: one for each fold j. Which set do we report?

One simple strategy is to try to pick the best parameters — e.g., by testing each W_j on a small amount of additional data. This strategy, though, requires us to save some of our data for this purpose; we then can't use this data for training, which is unfortunate.

方案 1 选在 test 上表现最好的模型： 需要预留一部分数据 test

方案 2 最常用：

Another strategy, and perhaps the most common one, is to *re-optimize*. That is, once we are satisfied with our model design — once we've forced down the cross-validation error as much as we can without risking overfitting — we fix the model design and optimize our model parameters again using the entire dev set.

re-optimize

通过调整模型超参数，使得最终的 average validat error 很小，然后用全部的数据从新训练模型(前提是保证不 overfitting，还是需要 test)

方案 3：ensemble(model averaging) k 个模型！！

But, perhaps the best strategy is *model averaging*: we report all of the parameter sets instead of picking just one. (Such a set of models is sometimes called an *ensemble*.) Then, when it comes time to make predictions on a new example X, we predict once using each model, and average the predictions:

$$\hat{Y} = \frac{1}{k}\sum_{j=1}^{k} W_j X .$$

For linear regression, this prediction is the same as the one one we get by averaging the parameters from each fold, and reporting the average parameters $\frac{1}{k}\sum_{j=1}^{k} W_j$. Model averaging also works for many other ML methods, not just linear regression. But, in the general case, we can't just average the parameters; instead we have to predict separately using each parameter set, and average the predictions.

ensemble：其实就是将多个 model 结果平均(这里是多个 cross validation model)

对于 linear regression，是将多个模型参数取平均

对于分类，是分类预测结果进行 average

It's also interesting to skip averaging our predictions together, and instead report the entire list of predictions. We can view this list as a representation of how certain we are of our prediction: if all of the predictions are nearly the same, we're highly confident, while if they vary a lot, we're not confident at all. For a low-dimensional prediction, we can even fit a simple distribution (such as a Gaussian or a kernel density estimate) to our list of predictions, and report this distribution instead of our list. Such a distribution is called a *predictive distribution*, and it summarizes our certainty (or lack thereof) about our prediction.

也可以 report 全部的 prediction 结果，将 prediction 变成 list，可以代表我们的 confidence

甚至可以 fit **predictive distribution(fit 成一个分布)，每个结果可以计算一个概率或概率密度，代表我们对结果的确定性，**每个结果一个概率

beyond prediction error

8. Beyond prediction error

So far in this set of notes we've been concerned with accurately estimating and reporting our prediction error. There are lots of other questions we could ask, particularly if we want to interpret the regression weights W. For example, we could ask whether there is evidence that some element W_{ij} is nonzero — i.e., whether one of our inputs is useful for predicting one of our outputs. Or, we could ask whether it is more likely that W_{ij} is positive or negative — i.e., whether the input tends to be associated with increases or decreases in the output.

我们还想知道其他的信息：对 weight 的解释(是否是 0，正负)，input 的贡献，和 output 的相关关系

An ensemble of models can help us answer questions like these. For example, if W_{ij} is positive in a large fraction of our models, then the ensemble provides evidence that W_{ij} is truly positive — that is, that we would still find $W_{ij} > 0$ if we had a much larger amount of data available to train on.

In fact, we can (at least informally) interpret our model ensemble as a sample of size k from our posterior distribution over W, that is, from

$$P(W \mid X^1, \ldots, X^n, Y^1, \ldots, Y^n).$$

So, if we want to answer a question about W, we can ask what fraction of times the answer is yes for the models in our ensemble, and report back that fraction as an estimate of the posterior probability $P(\textbf{yes} \mid \textbf{data})$. E.g., if $W_{ij} > 0$ in 9 of the 15 models in our ensemble, then we say $P(W_{ij} > 0 \mid \textbf{data}) \approx \frac{9}{15}$.

In the case of cross-validation, the interpretation of an ensemble as a sample from the posterior is only informal. We will shortly see a related method called the *bootstrap* for which we can formalize this interpretation.

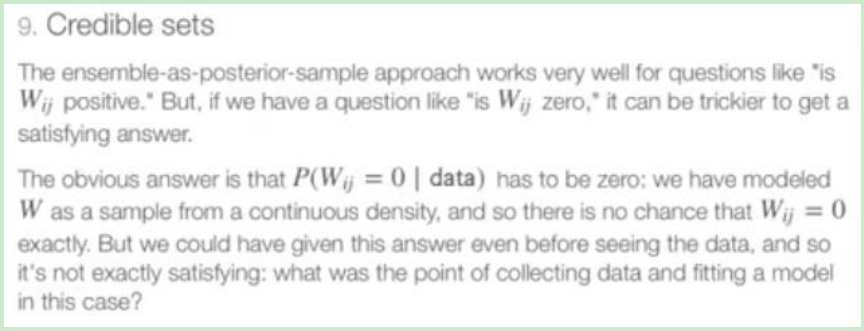
ensemble model 可以帮我们准确的回答(看每个 model 该参数 w 的

值)

k-fold k 个 model 可以理解是从后验概率 P(W|observed X,Y) 里 sample

假如 15 个 model，有 9 个 W>0， 9/15 ，可以作为后验概率 p(W>0|data)的 estimate 估计

Credible sets

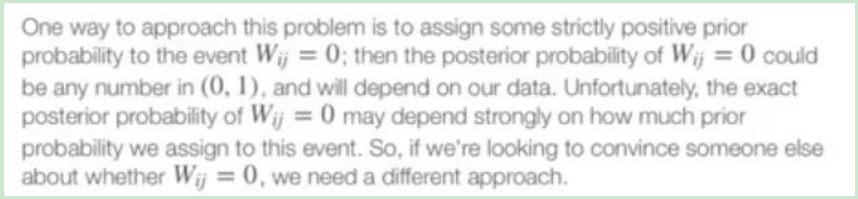


9. Credible sets

The ensemble-as-posterior-sample approach works very well for questions like "is W_{ij} positive." But, if we have a question like "is W_{ij} zero," it can be trickier to get a satisfying answer.

The obvious answer is that $P(W_{ij} = 0 \mid \text{data})$ has to be zero: we have modeled W as a sample from a continuous density, and so there is no chance that $W_{ij} = 0$ exactly. But we could have given this answer even before seeing the data, and so it's not exactly satisfying: what was the point of collecting data and fitting a model in this case?

如果问参数精确到是否 0 更加复杂，因为理论上 input 对 output 没关系 weight 应该为 0，但实际上我们的 data 只是 **sample**，不代表总体，得到的实际 model，weight 可以会普遍小，但不一定精准的是 0，不好验证了(而且因为我们的 data 是 sample，weight 不会完全为 0)



One way to approach this problem is to assign some strictly positive prior probability to the event $W_{ij} = 0$; then the posterior probability of $W_{ij} = 0$ could be any number in $(0, 1)$, and will depend on our data. Unfortunately, the exact posterior probability of $W_{ij} = 0$ may depend strongly on how much prior probability we assign to this event. So, if we're looking to convince someone else about whether $W_{ij} = 0$, we need a different approach.

一个方法是给该参数一个先验 p(w=0)，然后得到后验 p(w=0| **observed** data)

但实际，**我们给的先验概率越大，后验概率也越大，不足以信服**

credible set：贝叶斯置信区间 给 w 一个置信区间(取值区间，是不是 0 一目了然)

One such approach is called a *credible set*. Credible sets are the Bayesian version of *confidence sets* or *confidence intervals*, which you may have encountered before; in most cases, we can use a credible set anywhere we can use a confidence interval.

An α-credible set is a set that covers a fraction $1-\alpha$ of our posterior probability. The parameter α is called the *size* or *level* of the set. For example, if 95% of our posterior samples have $W_{ij} \in [3.72, 4.05]$, then $[3.72, 4.05]$ is a size-0.05 credible set for W_{ij}. We can use credible sets to answer the question of whether W_{ij} is zero or nonzero: we say that we have evidence that $W_{ij} \neq 0$ iff the credible set does not contain zero. The strength of the evidence depends on α: small values of α correspond to strong evidence.

这个 set of w 涵盖了 1-alpha 的后验概率

例如：假如置信水平 alph=0.05，也就是后验概率分布

p(w|observed data)的 samples 里有 95%的 w 在这个区间内(0.5 credible set)

如果区间不包含 0，我们认为 w 不会是 0(have strong evidence 不是 0)，alpha 越小，confidence 越大

Bootstrapping

可以帮助我们更接近 poterior of model parameter

Bootstrapping 可以当做是 sample of posterior P(W|data)

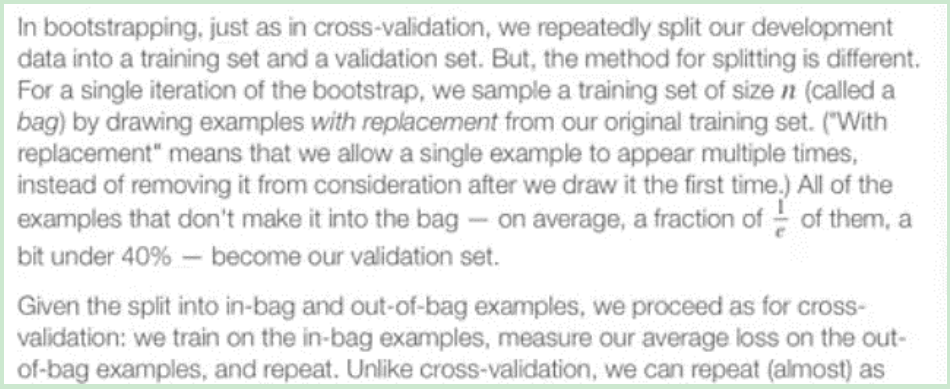
Can treat fitted parameter vectors as a sample from ***posterior*** distribution over parameters (given data)

Cross-validation has a couple of unfortunate limitations: first, it only gives us a heuristic approximation to the posterior distribution over our parameters W. And second, we can only have up to n folds if we have n training examples — we might want more folds if we need an accurate estimate of a posterior probability. To fix both of these limitations, we can switch to a method called the *bootstrap*.

limitation:

1 heuristic 探索的，不是很精确对于后验概率

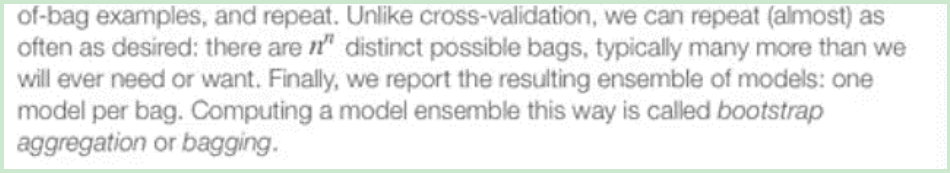
2 n-fold，有时候 n 太少，sample 太少，很难去估计后验概率


In bootstrapping, just as in cross-validation, we repeatedly split our development data into a training set and a validation set. But, the method for splitting is different. For a single iteration of the bootstrap, we sample a training set of size n (called a *bag*) by drawing examples *with replacement* from our original training set. ("With replacement" means that we allow a single example to appear multiple times, instead of removing it from consideration after we draw it the first time.) All of the examples that don't make it into the bag — on average, a fraction of $\frac{1}{e}$ of them, a bit under 40% — become our validation set.

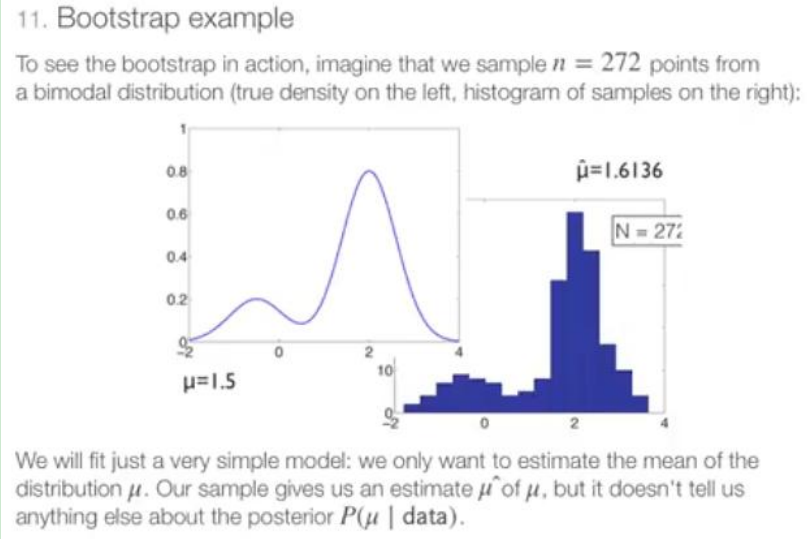
Given the split into in-bag and out-of-bag examples, we proceed as for cross-validation: we train on the in-bag examples, measure our average loss on the out-of-bag examples, and repeat. Unlike cross-validation, we can repeat (almost) as

bootstrap 同样也是将 data split 成 train 和 validation，但 split 方法不同：

bootstrap 每次获取 train 的方法是：有放回(replace)的抽取 train set(in-bag)从 original train set 里 sample（有些 example 永远不会进入 train set），那些没有出现在 train 里的 data 作为 validation set(out of bag)


of-bag examples, and repeat. Unlike cross-validation, we can repeat (almost) as often as desired: there are n^n distinct possible bags, typically many more than we will ever need or want. Finally, we report the resulting ensemble of models: one model per bag. Computing a model ensemble this way is called *bootstrap aggregation* or *bagging*.

我们可以得到更多的 split 组合，这样可以得到更多的 model 来 ensemble

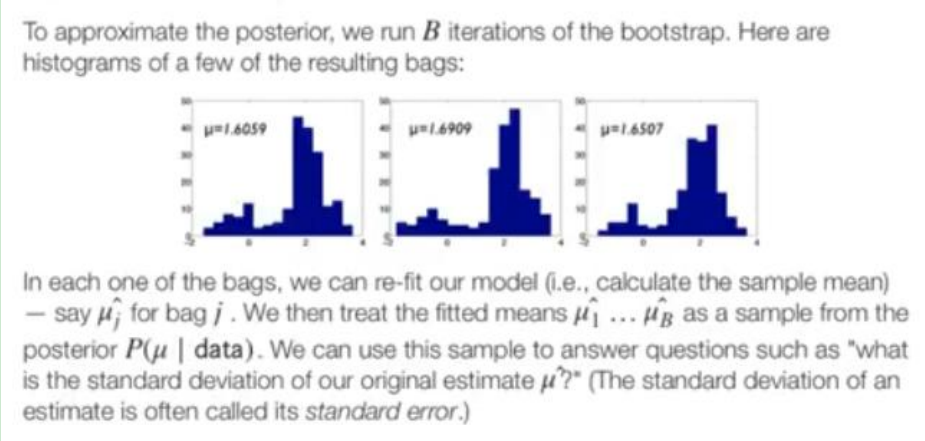


从已知的分布中 sample 272 数据

我们想从数据估计真实分布的**均值**

我们的 sample 只能给我们一个 estimate 总体的均值，是对 true mean 的估计，但我们想知道后验概率 p(u |270data) 真实 u 的后验分布

run B 次 bootstrap：还在 272data 里 bootstrap



我们将每个 bag 的 mean 可以看做后验概率 p(u|270data)的 sample 来回答我们总体 u (question)的标准差是多少

经过计算发现我们 estimate u 的标准差和真实 u 的标准差差不多

For example, on one run of the bootstrap with $B = 100,000$ we get an estimated standard error of 0.0818, which differs from the true standard error of 0.0825 by less than 0.001. (In this simple example we can calculate the exact standard error, but in general we can't, and have to rely on tools like the bootstrap.) With many fewer repetitions we can get a coarser estimate of the standard error: e.g., $B = 10$ leads to an estimate that is off by about 0.1.

这些 mean 的方差，可以告诉我们 confidence

12. Bootstrap asymptotic behavior

Under appropriate assumptions, it is possible to prove that each iteration of the bootstrap gives us an accurate sample from the posterior distribution $P(W \mid \text{data})$. The most important assumptions are:

- We need a large enough original sample. In practice the bootstrap can start working well with very reasonable-sized samples; e.g., the example above has $n = 272$ samples.
- The model class needs to be "smooth enough". Intuitively, this means that the model class shouldn't be able to react strongly to the exact placement of the samples: for example, the bootstrap produces bags with the (highly unrealistic) property of exactly-repeated samples, and we're in trouble if the model class can tell the difference between these and real samples.
- The (original) sample needs to be iid. If not, the bootstrap (and in fact many other methods) will give us overconfident answers: it will say that the standard error of our estimates is smaller than it truly is. This sort of overconfidence can be a significant problem: it can be very bad to be confidently wrong.

每一个 bag 都是从后验分布里的 sample **并且是 IID sample**

prerequisite 向量的膜和向量差的膜

永远和原来元素单位一致

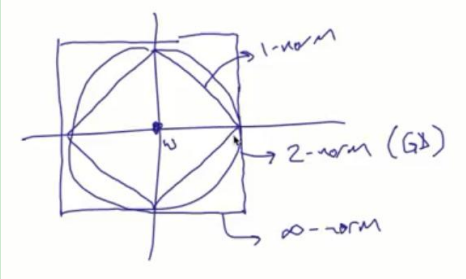
2-norm 欧氏距离 $\|\bullet\|_2$ **元素平方和开根号**

再平方 $\|\bullet\|_2^2$

1-norm: $\|\bullet\|_1$ 元素绝对值和， $\|\bullet\|_1^2$ **是再平方**

两个向量差 v-w 的 norm

假设向量是 2 维度，w 是圆心(0,0)向量，



1-norm 为例 假设 v 是(x,y) 且和 w 差的是 1

x，y 都是正时，1-nrom 是 x+y=1

xy 都负：-x-y=1，就对应上面的二维四边形

2-norm 是个**球**

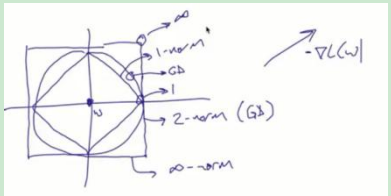
infinite-norm 是正方体

也就是不同的 norm，v 的取值范围不同！！！

也就是 v 的取值范围不同！！！

因此对应 gradient，v 的变化范围也就不同：

small update 不同，change direction(path)



改变了 parameter value space！！！

take a different route toward the minimum of L

Incremental Forward stagewise regression

stagewise 分段回归；用于 feature selection

1. Variable selection

We mentioned earlier that an important type of feature transform for linear regression is *variable selection*: reducing the number of dimensions in our input vector X to make better use of available data. If we don't know ahead of time which dimensions are relevant, we can use a machine learning algorithm to help us decide: the algorithm will try to discover a small set of dimensions that together lead to an accurate prediction of our output vector Y.

In this set of lecture notes we'll describe *incremental forward stagewise regression (IFSR)*, an effective and simple algorithm for variable selection in linear regression. IFSR also provides a nice way to introduce a useful optimization tool, *steepest descent*.

IFSR Incremental forward stagewise regression

可以 feature selection

gradient descend 另一种解释

2. Gradient descent revisited

As a reminder, if we are trying to minimize the function $L(w)$, gradient descent repeatedly sets

$$w \leftarrow w - \eta \nabla L(w) .$$

That is, it repeatedly evaluates the gradient of L at the current parameter vector w, and subtracts a multiple of this gradient from w.

We can rewrite the exact same update as

$$w \leftarrow \arg\min_v \left[\frac{1}{2\eta} \|v - w\|_2^2 + (L(w) + \nabla L(w)^T (v - w)) \right]$$

(where we have written $\| \cdot \|_2$ to emphasize that we are referring to the ordinary Euclidean norm, also called the 2-norm).

This expression says that we choose the next value of our parameter vector to trade off two terms: first we want to make a small update (the term $\frac{1}{2\eta}\|v - w\|_2^2$), and second we want to minimize a Taylor approximation to L at w (the term $L(w) + \nabla L(w)^T (v - w)$). The parameter η controls this tradeoff: small values of η emphasize the first term, while large values of η emphasize the second.

变成一种优化，**参数是最终的更新值**，使得目标函数最小！

假设最终更新后的参数值是 v

1 左边是 small update term：使 2-norm 最小化也就是让 v 接近 w

2 右边泰勒展开：右边就是 Loss(v)，目的就是更新参数使其最小

steepest，同时还要使得 step v 尽量(we want small update)

We can see that this new expression leads to the same update as before. Write $F(v)$ for the term inside square brackets above, so that our update becomes $w \leftarrow \arg\min_v F(v)$. Then by differentiating and setting to zero we have

$$\begin{aligned} 0 &= dF \\ &= \frac{1}{\eta}(v-w)^T dv + \nabla L(w)^T dv \\ 0 &= (v-w) + \eta \nabla L(w) \\ v &= w - \eta \nabla L(w) \end{aligned}$$

上面的函数如果是 F(v)，求导并取最小值后,得到的 v 就是

跟上面的形式一样

Steepest descent

change the small update term to 1 norm

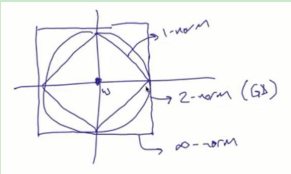
We can generalize gradient descent by changing the "small update" term above to use a different norm, such as $\|v-w\|_1$ (the sum of absolute values of components of $v-w$) or $\|v-w\|_\infty$ (the maximum absolute value across components of $v-w$). For example, using the 1-norm, we get

$$w \leftarrow \arg\min_v \left[\frac{1}{2\eta}\|v-w\|_1^2 + L(w) + \nabla L(w)^T (v-w) \right].$$

This new algorithm is called *steepest descent*; as we saw above, steepest descent with the ordinary Euclidean norm is the same as gradient descent, but if we use a different norm the updates can be different.

改变了 parameter space

1-norm space 最 sparce（sparce update 因为它最接近原点，也就是最接近 w，也就是距离几乎为 0，update 也几乎为 0），最终得到的参数也是 sparce 参数(几乎为 0 ，值接近 0)



By changing the norm, we change what we mean by a small update: e.g., if we use the 1-norm, we are more willing to go in directions that have a small 1-norm, as compared to gradient descent, which prefers directions that have a small 2-norm. We have effectively changed the geometry of our parameter space, and therefore changed what we mean by the shortest path between two points: steepest descent will take a different route toward the minimum of L, depending on which norm we choose.

take a different route toward the minimum of L

This change could change the behavior of optimization in several ways:

- In a problem with multiple local minima, we could get to a different one depending on which norm we use.
- Even in a strictly convex problem (which has no local minima, only a single global minimum), it could take more or fewer iterations for the optimization algorithm to reach the minimum.
- If we stop before reaching a minimum (as in the regularization strategy of early stopping), we will get a different answer depending on which norm we use.

会得到不同的 local optima ,iteration 次数不同

early stoping 的结果不同

5. Computation

In order to run steepest descent efficiently, we need to solve minimization problems of the form

$$\arg\min_d \left[\frac{1}{2\eta}\|d\|^2 + g^T d\right]$$

for whichever norm $\|\cdot\|$ we pick. (Here we have written g for the gradient and d for $v - w$, and we have dropped the constant term $L(w)$ since it doesn't affect the argmin.)

In general this minimization problem could be difficult; but, for some common norms (including $\|\cdot\|_1$ and $\|\cdot\|_\infty$) there are fast algorithms to solve it. We'll describe a couple of these algorithms without proof. We'll describe the solutions for $\eta = 1$; if $\eta \neq 1$, we just scale these solutions by η.

For $\|\cdot\|_1$, the optimal d will have a single nonzero component, with the same position and opposite sign as the largest-magnitude component of g. The absolute value of this component will be $\|g\|_1$. For example, if $g = (-1, 2, 0)^T$, then $d = (0, -3, 0)^T$. If two or more components of g are equally large, we can break ties arbitrarily.

For $\|\cdot\|_\infty$, the optimal d will have all components equal to $\pm\|g\|_\infty$. Each component's sign will be opposite to the corresponding component of g. For example, if $g = (1, -3, -1)^T$, then $d = (-3, 3, 3)^T$. If some component of g is zero, we can choose the sign of the corresponding component of d arbitrarily.

对于 1-norm 和 infinite-norm 会计算量很容易！！！有技巧

question

IFSR

6. Incremental forward stagewise regression

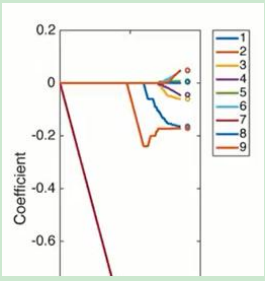
Given the above definitions, it's simple to describe IFSR. We start from a regression problem — i.e., from some training data (X^t, Y^t). We build the least squares objective for our training data, and we minimize this objective by steepest descent, using the 1-norm and a small learning rate. We stop before we reach the global minimum of our objective: in particular, we stop while our parameter vector w is still sparse. (For example, we can choose when to stop by looking at some holdout data or by bootstrapping.) The nonzero components of w correspond to the variables we have selected, since we can drop all of the other components of w and X without changing our prediction.

但如果我们用 1-norm +steepest gradient descent+小 learning

rate+early stopping(保证 w 还是 sparce)用 hold out set 来决定什么时候停止(防止过拟合) 其实就是：IFSR

1-norm +steepest gradient descent 如果 run 很久，最终得到的是 least square solution,w 还是一样的(question?)，但我们 early stopping(w 就 sparce 了)

A nice way to visualize the output of IFSR is to graph the *regularization path*, i.e., all the parameter vectors w that we pass through on the way to the optimum. The plot below is an example: here $w \in \mathbb{R}^9$. The horizontal axis is the length of the path so far (considered as a curve in $\mathbb{R}^9$), and the vertical axis shows the values of each component of w. For comparison, the least-squares w is shown with $\circ$ symbols at the far right.



Graphic model

GM

1. Interactions

The key question when modeling multiple random variables is to describe their interactions: how knowledge about some of the variables influences our beliefs about the others. We've talked so far about two types of models that capture these interactions: multivariate maxent distributions, and linear regression. In the following lecture notes we'll talk about *graphical models*, which generalize both of the above model types. We'll say only a little bit about graphical models in general, and then move to an interesting special case: *Gaussian graphical models*, in which the joint distribution of all variables is Gaussian.

(A full treatment of graphical models would take an entire course. If you're interested, that course is 10-708 Probabilistic Graphical Models.)

多个变量的交互关系(direct 和 indirect)

2. Direct and indirect interactions

Graphical models are based on distinguishing *direct* from *indirect* interactions among variables. In a direct interaction between two variables X and Y, there are no intermediaries: that is, there are no other variables that can completely explain the effect of X on Y or vice versa. In an indirect interaction, the entire effect of X on Y (or vice versa) can be explained by some set of other variables $Z_1 \ldots Z_k$.

GM 是用来区分 direct 和 indirect interaction

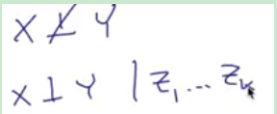
direct X 直接影响 Y，没有任何中间变量，没有 Z 就

indirect X 对 Y 的影响，可以被其他变量解释，或通过其他变量来影响

More precisely, suppose that there exist some variables $Z_1 \ldots Z_k$ such that X and Y are independent in the conditional distribution $P(X, Y \mid Z_1 \ldots Z_k)$. In this case any interaction between X and Y is indirect. On the other hand,

也就是 X Y 条件独立！

X Y 本身不独立，因为还是会间接影响！！，但条件独立！！



For example, suppose we have three variables:

- X indicates whether a person is infected with the influenza virus;
- Y indicates whether the person has symptoms of flu;
- Z indicates whether the person stays home from work or school.

We could make a model where X and Y interact directly, and Y and Z interact directly, but all interaction between X and Z runs through Y.

X 感染病毒　Y 流感症状 Z 是否在家

X 和 Y direct　　Y 和 Z direct

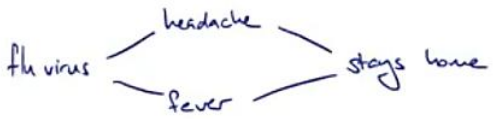
X 和 Z indirect

3. Graphical models

We can draw a graph that represents the above interaction structure:

$$X — Y — Z$$

More generally, we can draw a graph where each of our random variables is a node, and there are edges between pairs of variables that (at least potentially) interact directly. For example:



This kind of graph is called a *Markov random field* or *MRF*, and it is one of the most common types of graphical models.

边界**直接相连的** edge 是 direct interaction，不直接相连但有 path 的是 indirect，没有 path 的是无关

MRF：马尔科夫随机场

More generally, a graphical model is any way of specifying properties of a probability distribution using a graph. The graph can be directed or undirected, and it can have nodes for other entities besides just random variables. Some common graphical models (besides MRFs) include *Bayes nets*, *factor graphs*, and *junction trees*; here we'll focus only on MRFs.

The advantage of a graphical model is that it can make it much easier to work with and reason about a probability distribution. For a simple example, in a Markov random field, there can be an interaction between two variables only if there is a path in the graph between these variables. This interaction can be direct if there is a path with just one edge; otherwise it can only be indirect.

帮助我们 reason **概率**

GM and maxent

4. Graphical models and maxent

Recall that a maximum entropy distribution takes the form

$$P(X = x) = \exp(t(x) \cdot \theta - g(\theta))$$

where $t(x)$ is the vector of statistics, θ is the natural parameter vector, and $g(\theta)$ is a normalizer.

In this expression, it's useful to look at the *sparsity structure* of $t(x)$: each component of $t(x)$ might depend on only some components of x. In this case, we call that set of components of x a *neighborhood*. For example, if

$$t(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_1 x_2 \\ x_2 x_3 \end{pmatrix}$$

then the neighborhood of t_1 is $\{x_1\}$, while the neighborhood of t_5 is $\{x_2, x_3\}$.

X 是三维变量 R3

每个 component 所包含变量的 set 叫做 neighborhood

t depend 的 x 元素**就是 neighborhood**

We can build an MRF using only this sparsity structure: there is an edge between two variables if and only if they are in some neighborhood together. (We won't prove this statement, but the proof is a good exercise in working with probability distributions and independence.) For example, the sparsity structure of $t(x)$ defined above leads to the MRF

$$X_1 - X_2 - X_3$$

This MRF tells us about the structure of the distribution $P(X)$: e.g., if we condition on X_2, then X_1 and X_3 are independent.

在 maximum entropy distribution P(X)**两个变量如果在同一个 neighborhood 里，他们的交互是 directly(上面只有最后两个)**

为什么 question 没有给证明

5. Gaussian graphical models; regression

Unfortunately, computation in general maxent distributions (or their corresponding MRFs) can be prohibitively expensive. However, there is one important exception: if the joint distribution of all variables is Gaussian, then we can do many operations efficiently, such as conditioning on a set of variables or marginalizing out a set of variables. We'll see more about GGMs in the next set of lecture notes.

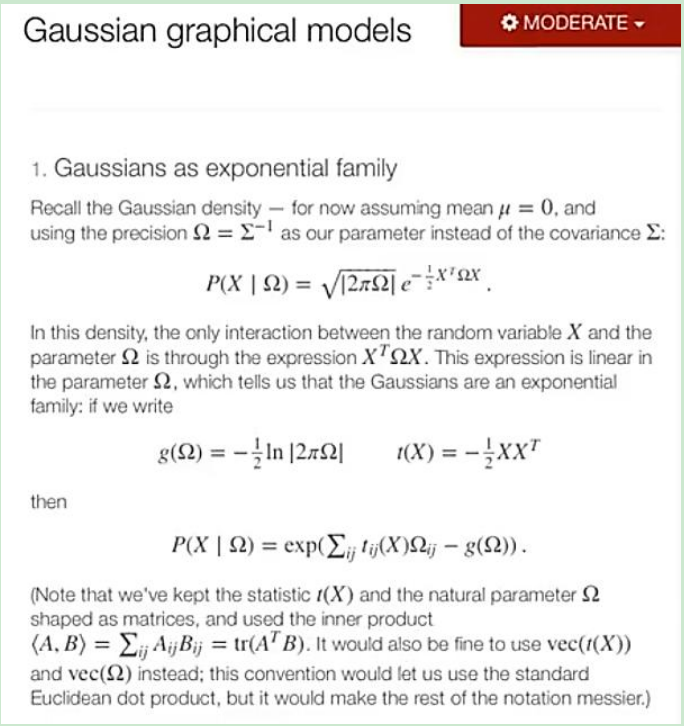
In particular, we'll see that linear regression can be interpreted as computing a conditional distribution in a GGM — showing that, as promised, graphical models generalize linear regression.

每个 gm 都有一个对应的 maximum entropy distribution

计算量太大

但如果**联合分布是高斯分布，计算量就是小了**

Gaussain graphical model



Gaussian graphical models

MODERATE

1. Gaussians as exponential family

Recall the Gaussian density — for now assuming mean $\mu = 0$, and using the precision $\Omega = \Sigma^{-1}$ as our parameter instead of the covariance Σ:

$$P(X \mid \Omega) = \sqrt{|2\pi\Omega|}\, e^{-\frac{1}{2}X^T\Omega X}.$$

In this density, the only interaction between the random variable X and the parameter Ω is through the expression $X^T\Omega X$. This expression is linear in the parameter Ω, which tells us that the Gaussians are an exponential family: if we write

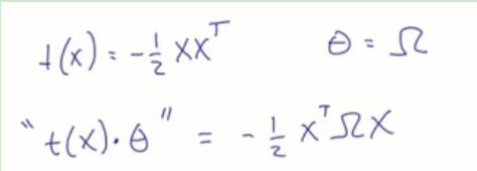
$$g(\Omega) = -\tfrac{1}{2}\ln|2\pi\Omega| \qquad t(X) = -\tfrac{1}{2}XX^T$$

then

$$P(X \mid \Omega) = \exp(\textstyle\sum_{ij} t_{ij}(X)\Omega_{ij} - g(\Omega)).$$

(Note that we've kept the statistic $t(X)$ and the natural parameter Ω shaped as matrices, and used the inner product $\langle A, B\rangle = \sum_{ij} A_{ij}B_{ij} = \mathrm{tr}(A^TB)$. It would also be fine to use $\mathrm{vec}(t(X))$ and $\mathrm{vec}(\Omega)$ instead; this convention would let us use the standard Euclidean dot product, but it would make the rest of the notation messier.)

Ω precision matrix 是参数



t(X)和 precision 是矩阵元素相乘和

高斯分布写成 maxent(指数分布族)形式

t(X)是 X 的外积也就是把**一个 X 向量变化成了一个矩阵特征，也**就是假如 X 有 X1，X2

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix}[X1 \quad X2] = \begin{bmatrix} X1X1 & X1X2 \\ X1X2 & X2X2 \end{bmatrix}$$

变化后的 feature，包含了全部变量的交互，GM 会全部连接！如下：

3. Gaussians as MRFs

Each one of our features or statistics $t_{ij}(X)$ depends on only a small subset of the coordinates of X — in particular, only on X_i and X_j. Because the features are sparsely connected like this, it makes sense to draw a Gaussian distribution as a Markov random field: a graph with one node per coordinate of X, and one edge for each feature t_{ij}. Here's an example where $X \in \mathbb{R}^6$:

Now, the above graph is not very interesting: it's just a complete graph. But, things start to get interesting if we start to remove edges, as we will below.

t 的每个 component 只依赖部分的变量 X

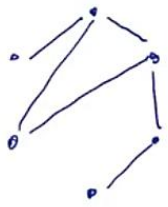
假设 X 是 6 维，可以画出每个 Xi 之间的图关系：如果变量 Xi 都在一个 neighborhood 里，就有 direct 关系

假如Ω部分元素为 0，就少了些交互项：

$$P(X \mid \Omega) = \exp(\textstyle\sum_{ij} t_{ij}(X)\Omega_{ij} - g(\Omega)).$$

4. Sparse precision

If we constrain some of our precision matrix entries Ω_{ij} to be zero, that corresponds to ignoring some of our features $t_{ij}(X)$, which corresponds to removing some of the edges from our graphical model:



The result is called a *(sparse) Gaussian graphical model*, and it is a surprisingly powerful tool for modeling multivariate continuous distributions. Like any graphical model, we can use the graph structure to help us answer questions about the structure of the distribution, such as which variables are (conditionally) independent of one another. But unlike most graphical models, there are poly-time algorithms for many useful operations on GGMs. This combination of expressiveness and tractability can be quite handy.

Note: we assume that all diagonal elements of Ω are allowed to be nonzero. If we constrained a diagonal element of Ω to be zero, Ω would not be invertible.

GM 对于 model 多个连续变量分布很有用

5. Conditioning

Conditioning in a GGM is straightforward: we we start from our prior density, fill in the values of the variables (components of X) that we have observed, and renormalize to get our posterior density. Since the prior density is of the form

$$\frac{1}{\text{normalizing constant}} e^{-\text{quadratic in } X}$$

our posterior will also be of this form: we are just fixing some of the variables in the quadratic, thereby reducing the degree of some of the terms.

$P(X \mid \Omega) = \sqrt{|2\pi\Omega|}\, e^{-\frac{1}{2}X^T\Omega X}$ 这是我们的 joint distribution

condition 就是 fill 几个 variable already know

然后再 normalize，使得相加为 1

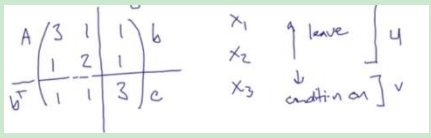
In more detail, suppose that the quadratic part of the prior density is

$$\tfrac{1}{2}X^T\Omega X$$

(that is, our prior is Gaussian with mean 0 and precision Ω). Suppose that we observe the last component of X. Partition X and Ω according to our observation:

$$X = \begin{pmatrix} U \\ v \end{pmatrix} \qquad \Omega = \begin{pmatrix} A & b \\ b^T & c \end{pmatrix}$$

where $A \in \mathbb{R}^{n\times n}$ is a matrix, $b, U \in \mathbb{R}^n$ are vectors, and $c, v \in \mathbb{R}$ are scalars. Our goal is now to calculate the posterior density over U (the unobserved components of X).



prior 是高斯分布均值 0

我们观测到了 v 是 X 的最后一个元素，将向量和矩阵分块，为了区分出观测和未观测的元素

我们想计算 U 的后验(未被观测的数据的后验概率)

We can split up the quadratic part of the prior as

$$\tfrac{1}{2}X^T\Omega X = \tfrac{1}{2}(U^TAU + 2U^Tbv + v^2c).$$

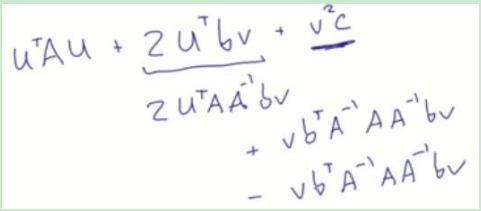
The last term, v^2c, doesn't depend on U; we can therefore ignore it. We can complete the square for the remaining terms: rewrite U^Tbv as $U^TAA^{-1}bv$, and add and subtract $b^TA^{-1}bv^2$, and we get

$$\tfrac{1}{2}X^T\Omega X = \tfrac{1}{2}(U + A^{-1}bv)^TA(U + A^{-1}bv) + \text{constant}.$$

As promised, this is a quadratic in U. By pattern matching, we can see that it corresponds to the quadratic part of a Gaussian log-density with precision A and mean $-A^{-1}bv$. So, if we fix the last component of X to v, our posterior density for U becomes a Gaussian with precision A and mean $-A^{-1}bv$.

计算 $X^T\Omega X$,中间增加 AA^{-1},也就是 I 不变

然后把它组成平方式：question **complete the square**



得到未被观测的数据的后验概率

会有了新的 mean 和 precision ：

To summarize: to condition on an element of X in a zero-mean Gaussian,

- delete the corresponding row and column of the precision matrix Ω, and
- set the mean for all other elements of X to $-A^{-1}bv$, where A and b are the appropriate blocks of Ω, and v is the observed value of the element of X.

对于 mean 不为 0 的高斯分布，先归 0 化，conditional 完，再加上 mean

We can handle a nonzero-mean Gaussian by first subtracting the mean, then conditioning, then adding the mean back in. The resulting update to the precision matrix is the same, since the precision is not affected by changes to the mean. If we partition the mean as

$$\mu = \begin{pmatrix} \mu_U \\ \mu_v \end{pmatrix}$$

(where $\mu_U \in \mathbb{R}^n$ and $\mu_v \in \mathbb{R}$) then the resulting update to the mean is

$$\mu_U \leftarrow \mu_U - A^{-1}b(v - \mu_v)\,.$$

question 没看懂

If we need to condition on several elements of X at once, we can condition on them one at a time; or, there are block versions of all of the above formulas. For example, the final formula becomes

$$\mu_U \leftarrow \mu_U - A^{-1}B(V - \mu_V)$$

where we partition X and Ω as

$$X = \begin{pmatrix} U \\ V \end{pmatrix} \qquad \Omega = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$$

and observe the V block. (Here the dimensions are $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times k}$, $C \in \mathbb{R}^{k\times k}$, $U \in \mathbb{R}^n$, $V \in \mathbb{R}^k$.) As always with block matrix formulas, we can handle observations of any components of X by permuting so that the observed components are last, updating, and unpermuting (or by doing the equivalent updates in place).

6. Conditioning and linear regression

We mentioned above that we can derive linear regression as (conditional) maximum likelihood estimation under a Gaussian assumption. That is, if we use regression to predict U from V, then we get a coefficient matrix W such that WV is the mean of U in the conditional distribution $P(U \mid V)$.

We just computed the conditional distribution $P(U \mid V)$ — making the same Gaussian assumption, so long as the observed covariance matrix of U and V is Ω^{-1}. Working with centered variables for simplicity, the mean of U under this distribution is $-A^{-1}BV$. Therefore, we must have $W = -A^{-1}B$.

We won't show it here, but this formula is equivalent to the one we derived earlier. The difference is that here, we've written W in terms of blocks of the precision matrix Ω; previously we wrote it in terms of blocks of the covariance matrix Ω^{-1}. To show equivalence, we'd use the *block matrix inverse* formula — that is, a formula for the inverse of a matrix (here Ω) in terms of its blocks (here A, B, and C).

question

7. Marginalization

Marginalization is in some ways simpler than conditioning, although the actual expression for the updated parameters is more complicated. It is straightforward to prove (although we will not do so) that all of the marginals of a Gaussian distribution are Gaussian. Given this fact, all we need to do to marginalize is compute the mean and variance of the marginal distribution.

But, it is clear that $E(X_i)$ is not affected if we drop a variable X_k, for any $k \neq i$. Similarly, $\mathrm{Cov}(X_i, X_j)$ is not affected if we drop X_k, for any $k \neq i, j$. So, computing the mean and variance of the marginal distribution is easy: they are the same as the mean and variance of the original distribution, except that we drop the components corresponding to variables that we are marginalizing out.

marginal 高斯 还是高斯

mean 和 cov 只需剔除我们 marginalize 的那个变量的成分，就得到新的高斯的 mean 和 cov

That is, to marginalize out X_k given the mean μ and precision Ω, we do the following:

- Compute the covariance $\Sigma = \Omega^{-1}$.
- Drop the kth row and column of Σ, and the kth element of the mean vector μ. Write Σ' for the resulting smaller covariance matrix, and μ' for the resulting smaller mean vector.
- Compute the updated precision $\Omega' = (\Sigma')^{-1}$.

Marginalization and conditioning are therefore very similar operations: to marginalize we drop rows and columns of the covariance, while to condition we drop rows and columns of the precision.

上面计算效率不高：

8. Schur complements

While it's simplest to think about marginalization as inverting, deleting variables, and inverting again, this is not the most computationally efficient algorithm: instead, we can avoid taking two matrix inverses by using the *Schur complement formula*. Suppose we have an $(n+1)\times(n+1)$ matrix Ω partitioned as

$$\Omega = \begin{pmatrix} A & b \\ b^T & c \end{pmatrix}$$

where $A \in \mathbb{R}^{n\times n}$ is a matrix, $b \in \mathbb{R}^n$ is a vector, and $c \in \mathbb{R}$ is a scalar. (We are implicitly assuming that Ω is symmetric, as all covariance and precision matrices are.) The Schur complement formula says that, if we marginalize out the last row and column of Ω using the above procedure (i.e., invert Ω, drop the last row and column, and invert again), the result will be

$$\Omega' = A - bc^{-1}b^T,$$

called the *Schur complement* of c in Ω. (Note that we have assumed without loss of generality that the variable we want to marginalize out is the last one; if not, we can permute the variables so that it is.)

假如我们想 marginalize 最后一行的变量

新的Ω可以有上面公式计算，不用去求 A 的逆

There is also a block version of the Schur complement formula that lets us drop several rows/columns at once: if we partition an $(n+k)\times(n+k)$ symmetric matrix Ω as

$$\Omega = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$$

so that $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times k}$ and $C \in \mathbb{R}^{k\times k}$, then the exact same formula applies:

$$\Omega' = A - BC^{-1}B^T.$$

加入我们想 marginalize 多个变量 C

In this block version, we still need to compute a matrix inverse; but as long as C is much smaller than Ω, we will save computation compared to the naive approach.

Alternately, if we want to maintain the covariance Σ instead of the precision Ω, marginalization becomes the cheap operation, and conditioning naively seems to require matrix inversions. However, there is an analogous way to use Schur complements to update the covariance matrix efficiently when we condition on one or more variables.

我们需要计算 C 的逆，但是 C 很小，容易计算

MLE Gaussain graphical model

Let's assume we have an i.i.d. sample $X^1, X^2, \ldots, X^N$ from a GGM with mean vector $\mu \in \mathbb{R}^d$ and precision matrix $\Omega \in \mathbb{R}^{d \times d}$. Let's suppose for now that we are given the structure of the GGM. That is, suppose we know which off-diagonal elements of Ω are constrained to be zero: say $\Omega_{ij} = 0$ for $(i, j) \in Z$. We'd like to estimate μ and Ω by maximum likelihood.

It's straightforward to derive that the maximum likelihood estimate for μ is the empirical mean

$$\hat{\mu} = \bar{X} = \frac{1}{N} \sum_{t=1}^{N} X^t$$

no matter what the structure of Ω is. So, we will assume that $\hat{\mu} = 0$ — which we can ensure by subtracting $\hat{\mu}$ from each sample X^t — and focus on estimating Ω.

假设我们有了 idd sample，并且知道 gm 的**结构，也就是知道Ω的非对角元素哪些不为 0**

我们想去 estimate u 和Ω参数用 MLE

MLE 的 mean 的结果就是 empirical mean，我们减去均值，然后重点去 estimate Ω！！！

With this assumption, our log likelihood is

$$\begin{aligned} L &= \ln P(X^1, \ldots, X^N \mid \Omega) \\ &= N \ln \sqrt{|2\pi\Omega|} - \frac{1}{2} \sum_t (X^t)^T \Omega X^t \\ &= \frac{Nd}{2} \ln 2\pi + \frac{N}{2} \ln |\Omega| - \frac{N}{2} \operatorname{tr}(C^T \Omega) \end{aligned}$$

where we have written

$$C = \frac{1}{N} \sum_t X^t (X^t)^T$$

for the empirical covariance matrix. We can find the unconstrained MLE (i.e., ignoring the structure defined by Z) by differentiating and setting to 0:

$$dL = \frac{N}{2} \operatorname{tr}(\Omega^{-T} d\Omega) - \frac{N}{2} \operatorname{tr}(C^T d\Omega)$$

(Here we used the fact that the differential of $\ln |\Omega|$ is the inner product of Ω^{-1} with $d\Omega$; we mentioned this fact in mini 1, or we can find it by looking it up in a table of differentials.) Setting $dL = 0$, we get

$$\Omega^{-1} = C.$$

求微分令 0

C 是我们 empirical covriance matrix

That is, the unconstrained MLE for the precision is the inverse of the empirical covariance C.

！！！！

10. Constrained maximum likelihood

To enforce the constraints $\Omega_{ij} = 0$ for $(i,j) \in Z$, we need a technique from constrained optimization called *Lagrange multipliers*. We'll cover this technique in the optimization section of this course. For now, we'll just skip to the result: instead of setting $\Omega = C^{-1}$, we set

$$\Omega = (C + \Lambda)^{-1}$$

where $\Lambda \in \mathbb{R}^{d \times d}$ is a matrix with $\Lambda_{ij} = 0$ for $(i,j) \notin Z$. (That is, the zero pattern of Λ is the complement of the zero pattern of Ω.)

We can find Λ by gradient descent: we repeatedly set

$$\Lambda \leftarrow \Lambda + \eta\Delta$$

where $\eta > 0$ is a learning rate, and the step matrix $\Delta \in \mathbb{R}^{d \times d}$ is given by

$$\Delta_{ij} = \begin{cases} [(C + \Lambda)^{-1}]_{ij} & (i,j) \in Z \\ 0 & \text{o/w} \end{cases}$$

This is not the most efficient algorithm for this purpose, but it is simple to code (half a dozen lines in Matlab) and works fine for small d.

constrain 某些元素=0

没细讲

假如我们不知道变量间的 structure：

11. Structure learning in GGMs

In our discussion of parameter learning, we assumed that we knew the structure of the GGM ahead of time — that is, which precision entries were allowed to be nonzero. Of course, we would also like to learn the structure of our GGM. This is an important problem, so there have been a lot of methods proposed to solve it; but, one of them follows nicely from ideas we have covered in this course, so that is the one we will cover here.

In particular we will use a connection to linear regression: we can optimize the jth row and column of Ω by fitting a linear regression model to predict X_j from all of the other components of X. Because of this connection, GGM structure learning reduces to variable selection: choosing the nonzero elements of the jth row/column of Ω is just like choosing the nonzero elements of the weight vector that we use to predict X_j from the other random variables.

Therefore, we can learn the structure of Ω by applying any of the variable selection methods that we've discussed. For example, we can run forward stagewise regression separately for each X_j, and use a bootstrap error estimate to determine when to stop adding variables to the model. Then we can let Ω_{ij} be nonzero if X_i had a nonzero weight in predicting X_j, or if X_j had a nonzero weight in predicting X_i. (There are lots of variants on this approach: for example, we could use another variable selection method (such as lasso) instead of forward stagewise regression; or we could let Ω_{ij} be nonzero only when *both* X_i and X_j have nonzero weights in predicting each other.)

Preconditioning and Momentum(待看 16 年视频，牛顿方法和 BFGS)

画二次型 contour

1. Review: sketching quadratics

We looked earlier at how to sketch the contours of a Gaussian PDF with precision matrix P,

$$\sqrt{|P/2\pi|}\,\exp\left(-\tfrac{1}{2}x^T Px\right).$$

(Recall that the precision matrix is the inverse of the covariance matrix.)

The contours aren't affected by a monotone (increasing or decreasing) transformation — such a transformation only affects the height of a contour, not its shape or position. So, we can drop the leading constant $\sqrt{|P/2\pi|}$, take the log, and negate, leaving

$$\tfrac{1}{2}x^T Px.$$

In other words, sketching the contours of a Gaussian PDF is the same problem as sketching the contours of a quadratic function.

高斯的 contour 的 shape 不受 scalar 加或者乘的影响，只跟 precision 有关

也就是为了画 $\exp(-\frac{1}{2}x^T Px)$ 其实就是等价于画 $\frac{1}{2}x^T Px$

To sketch these contours, we can take the SVD of the matrix P,

$$P = USU^T$$

where U is orthonormal and S is diagonal and nonnegative. The contours of the quadratic will be concentric ellipses (or ellipsoids in 3D or higher), whose shapes are given by the matrices U and S. Each column U_i points along one axis of the ellipsoids; the relative length of this axis is $\frac{1}{\sqrt{S_{ii}}}$. Scaling all of the axes together determines which contour we plot.

P 决定了 contour 形状，分解后 U S 决定了形状

$\frac{1}{2}x^T Px$ 应该类似于高维中的椭圆　U 是椭圆的每个轴，S 决定椭圆的半径

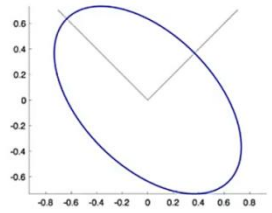
For example, when

$$P = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

then we can factor P as USU^T where

$$U = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \quad S = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$$

A representative contour then looks like this:



The contour is the blue line; the two black lines are U_1 and U_2.

precision matrix：元素如果较小，说明 A 和 B 变量的关联小

conditioning

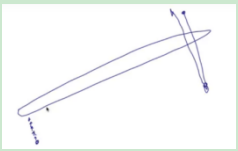
conditioning 反应就是 shape of contour 的情况

2. Review: conditioning

As we discussed earlier, the convergence speed of SGD depends on the shape of the contours of our loss function $L(w)$ in the neighborhood of a minimum. If the contours are close to round (a well-conditioned problem) then SGD will converge fairly quickly; if they are long and skinny (a poorly-conditioned problem), then SGD can take a very long time to converge.

The reason for slow convergence is that, in a poorly conditioned problem, it's impossible to set a good learning rate: a large learning rate will mean that we bounce back and forth across the shorter axes of the contours, leading to high variance in w along those directions. On the other hand, a small learning rate will mean that we take a long time to progress along the longer axes of the contours. If we knew the axis directions ahead of time, we could set a separate learning rate in each direction — but it can be prohibitively expensive to compute these directions.

SGD 速度受 contour 影响



lr 大和小都无法解决问题，太小就太慢，太大就波动

3. Review: condition number

If our loss function is quadratic, e.g., $L(w) = \frac{1}{2}w^T Hw$, then the tools from above for sketching a quadratic function can help us tell how well-conditioned our problem is: we can factor $H = USU^T$. If all the diagonal elements of S are approximately equal, then the contours of L will be close to spherical, while if some diagonal elements of S are much larger than others, then the contours of L will have some axes much longer than others.

对于 quadratic loss function contour condition 的量化

对 H 分解：S 的对角元素大小相当，contour 是球，如果差别大，就会畸形

For smooth but non-quadratic loss functions, we can apply the exact same tools: make a second-order Taylor approximation, $L(w) \approx \frac{1}{2} w^T H w + \ldots$, where H is the Hessian (second derivative) of L at w. Near w, the contours of L will be approximately the same as the contours of its Taylor approximation; and, we can take the SVD of H and use the tools from above to examine the latter contours.

如果不是 quadratic loss function，用二阶展开来近似(也是二次函数)，此时 H 是 Hessian Matrix，用同样的方法分解

loss function 是多个参数一个返回值

The condition number at w is defined as the ratio between the largest and smallest eigenvalue of the Hessian matrix $\nabla^2 L(w)$. The

Hessian Matrix

Hessian Matrix，就是对 quadratic loss function f 先求 w 的导数(gradient 向量)
再求一次导数就是矩阵，也就是 loss function 的二阶导数！！！

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

We can summarize the shape of the contours by looking at the largest and smallest diagonal elements of S. Their ratio is called the *condition number*; large condition numbers correspond to poorly conditioned problems, while condition numbers near 1 correspond to well-conditioned problems.

condition number=最大/最小(element of Hessian Matrix)

condition number=1 时，contour 是好的

知道了 condition number 如何调整我们的 lr?

Preconditioning

12. Preconditioning

In most applications of SGD, it's too expensive to compute $\nabla^2 L(w)$ exactly: remember that SGD is designed to avoid computing even the gradient vector exactly. But, if we can come up with a cheap-yet-accurate approximation $AA^T \approx (\nabla^2 L(w))^{-1}$, then we can still replace w by $w' = A^{-1}w$, and run SGD on the transformed problem. SGD will converge faster on the transformed problem if A is a good enough approximation to the square root of the inverse Hessian. This strategy — linearly transforming the variable w to improve the problem's condition number — is called *preconditioning*.

最好的方法改善 contour，可以先 transform variable，使得 contour 更好

$H = D^2 L$ (sencond derivative)　　$\mathrm{H} = \mathrm{AA}^{\mathrm{T}}$

As mentioned above, it's typically too expensive to compute and factor the Hessian H. But what if we could? Set $A = U\sqrt{S^{-1}}$. (By the square root of a diagonal matrix, we mean to take the square root of each diagonal element.) Now change variables to $v = A^{-1}w$, so that

$$\begin{aligned}\tfrac{1}{2}w^T H w &= \tfrac{1}{2}(Av)^T H (Av) \\ &= \tfrac{1}{2}(v^T\sqrt{S^{-1}}U^T)(USU^T)(U\sqrt{S^{-1}}v)\end{aligned}$$

The pairs $U^T U$ cancel out, as do the terms $\sqrt{S^{-1}}S\sqrt{S^{-1}}$, leaving

$$\tfrac{1}{2}w^T H w = \tfrac{1}{2}v^T v\,.$$

In other words, our change of variables has given us a perfectly-conditioned problem: one in which the Hessian is the identity matrix.

All of this is a thought experiment, though, since we've assumed that it's too expensive to compute and factor H. But, a lot of practical methods try to approximate the strategy above: that is, they look for a change of variables that improves conditioning, even if it doesn't lead to perfect conditioning. Such a change of variables is called a *preconditioner*.

One of the simplest possible preconditioners is based on the diagonal elements of the Hessian: it's often cheap to compute these, and ignoring the off-diagonal elements of the Hessian sometimes leads to a good-enough approximation. With this approximation, our preconditioner is just to scale each component w_i by $\sqrt{1/H_{ii}}$ (one over the square root of the corresponding diagonal element of the Hessian), then run SGD as usual. Or equivalently, we can leave w_i alone, and scale the learning rate for w_i by $\sqrt{H_{ii}}$.

There are lots of more-complicated ways to precondition; some prominent options are Adagrad, incomplete Cholesky factorization, (L-)BFGS, and natural gradients. We won't cover these here, but recommend looking them up if you need them.

Momentum (最好的解决！！)

另一种方法改善，更简单

As mentioned above, the failure mode for SGD in a poorly-conditioned problem is that the gradient updates behave differently along different directions in parameter space. Write U_1 for the column of U that corresponds to the largest diagonal element of S, and U_n for the column of U that corresponds to the smallest diagonal element of S. (So, the contours of our loss function are wide in the direction U_n, and narrow in the direction U_1.) The gradient of L will tend to be small in the direction of U_n, and large in the direction of U_1. If we set the learning rate too small, we'll make little progress in the direction U_n: the gradient will always point in more or less the same direction, but our steps will be so small that we barely move. On the other hand, if we set the learning rate high enough to make progress in the direction U_n, the direction U_1 becomes a problem: the gradient updates will tend to skip back and forth over the minimum in this direction, so that the projection of $\nabla L(w)$ on U_1 will have high variance.

每个 direction 的 gradient 情况不同，不好去设定 lr
如果 lr 设置过大，在 U1 方向上，波动就会很大,因为 U1 本身 gradient 就大，因此每个 step 就大，也就是每个 step 在 U1 方向的投影大

Momentum distinguishes between the directions U_1 and U_n by taking advantage of the above properties of our gradient updates: namely that the projections of our gradient estimates on U_n tend to be small but point in the same direction, while the projections on U_1 tend to be larger but point in different directions. These properties mean that, if we *average* our gradient updates over a window of a few iterations of SGD, the projections on U_n will tend to pass through unaffected, while the projections on U_1 will tend to cancel out. This is exactly the desired effect: we will effectively reduce our learning rate in the direction of U_1 compared to our learning rate in the direction of U_n. And, it works even though we don't know U_1 or U_n ahead of time — or even if the Hessian changes slowly, so that U_1 and U_n change as well.

Un 的 gradient 小，但是方向几乎一致(step 小)
U1 的 gradient 大，但方向都不一致(step 大)
如果对每个方向的 step 都取一段时间的均值，再作为 step，对 Un 影响不大，但对于 U1，这种随机性波动还会被 cancel out，step 方向会更稳定，我们就可以用大的 lr
如果方向不稳定，用大的 lr 比较危险

In more detail, where plain SGD just remembers the parameter vector w across steps, SGD with momentum keeps track of an averaged gradient h as well:

$$h \leftarrow \theta h + (1-\theta)g$$
$$w \leftarrow w - \eta h$$

h: average of gradient,用来更新 w　　g 是普通 gradient

where g is a stochastic gradient estimate, just like in plain SGD. If we set the momentum parameter θ to 0 we get plain SGD: the averaged gradient h will always equal the most recent stochastic gradient estimate. If we set θ closer to 1 we effectively average across a longer window, and each individual stochastic gradient estimate has a smaller effect. So, our updates to w tend to keep on going in the same direction as they have been going — hence the name "momentum."

Momentum works well in combination with preconditioning. For example, the popular Adam optimizer combines momentum with a diagonal preconditioner.

以上是 unconstrained optimization

constrained optimization and non-smooth optimization

Intuitive

$$L(w) = \ell_0(w) + \sum_{t=1}^{T} \ell_t(w).$$

We'd also like to solve constrained optimization problems, i.e., problems of the form

$$\min_w L(w) \text{ s.t. } w \in C$$

for some constraint set C. Similarly, we'd like to solve non-smooth problems, i.e., problems where some of the terms ℓ_t are not differentiable everywhere.

也就是对参数有约束，constrain the w in set C

很多常见例子：参数约束

Optimization problems that can be expressed with constraints or non-smooth terms are ubiquitous in ML: some examples are

- Lasso, a version of linear regression where we penalize $\|w\|_1$ to encourage sparsity;
- Support vector machines, a classifier similar to the perceptron except that we penalize $\|w\|_2^2$ to encourage large margins; and
- Nonnegative least squares (NNLS), a version of linear regression where we enforce $w \geq 0$ componentwise, to encode a monotone relationship between inputs and outputs.

We will see more about some of these problems below.

如何 solving

用 SGD 解决 constrained optimize

1 Projected SGD

2. Projected SGD

We'll start with a method for constrained optimization, called *projected SGD*. Projected SGD works when we can efficiently project onto the constraint set C: i.e., find the v that solves

$$\min_v \|v - w\|^2 \text{ s.t. } v \in C.$$

Projected SGD is similar to SGD, except that it projects w back onto C after every iteration: it repeatedly sets

$$w \leftarrow \Pi_C(w - \eta g)$$

where η is a learning rate, Π_C stands for projection onto C, and g is a stochastic gradient estimate. As with SGD, it's a good idea to use preconditioning, decreasing learning rates, early stopping, and minibatches.

Projected SGD 是个函数，参数是 SGD 更新后的参数 w，返回一个 1 满足 C, 2 且距离 w 小

其实就是计算出了 w，但 w 不一定满足 C，于是找一个 v，**1 和 w 距离尽量的小，2 并且 v 满足 C**

after every iteration(参数更新完后)，project on constrain set

每一个 step 如果**超越了 constrain set**，然后**再投影回到 constrain set**，也就是乘以一个 π

Minibatches work particularly well with projected SGD, since they can help save on the cost of computing Π_C: we only need to project after every minibatch, instead of after every stochastic gradient sample. For this reason, minibatches are even more the norm for projected SGD than for regular SGD.

Because we need to project often, projected SGD only works well if we can project efficiently. For example, we can project efficiently onto a box constraint like $w \in [0, 1]^d$ by handling each component independently:

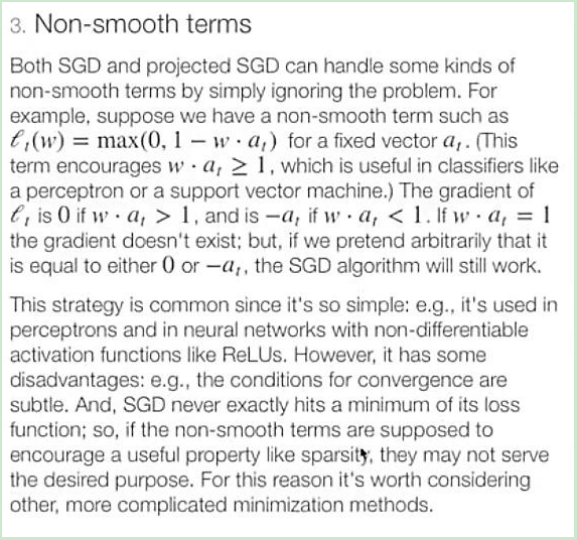
$$v_j = \begin{cases} w_j & w_j \in [0, 1] \\ 0 & w_j < 0 \\ 1 & w_j > 1 \end{cases}$$

Unfortunately, there's no sufficiently-efficient algorithm to project onto more interesting sets of constraints such as a system of linear equations $Aw = b$; we'll discuss below how to handle these sorts of constraints.

project operation 计算量大，需要 projection efficiently!

project 需要高效 for example：只要超过边界就等于边界 如上

但目前对于 Aw=b 没有特别高效的 projection 算法！！



3. Non-smooth terms

Both SGD and projected SGD can handle some kinds of non-smooth terms by simply ignoring the problem. For example, suppose we have a non-smooth term such as $\ell_t(w) = \max(0, 1 - w \cdot a_t)$ for a fixed vector a_t. (This term encourages $w \cdot a_t \geq 1$, which is useful in classifiers like a perceptron or a support vector machine.) The gradient of ℓ_t is 0 if $w \cdot a_t > 1$, and is $-a_t$ if $w \cdot a_t < 1$. If $w \cdot a_t = 1$ the gradient doesn't exist; but, if we pretend arbitrarily that it is equal to either 0 or $-a_t$, the SGD algorithm will still work.

This strategy is common since it's so simple: e.g., it's used in perceptrons and in neural networks with non-differentiable activation functions like ReLUs. However, it has some disadvantages: e.g., the conditions for convergence are subtle. And, SGD never exactly hits a minimum of its loss function; so, if the non-smooth terms are supposed to encourage a useful property like sparsity, they may not serve the desired purpose. For this reason it's worth considering other, more complicated minimization methods.

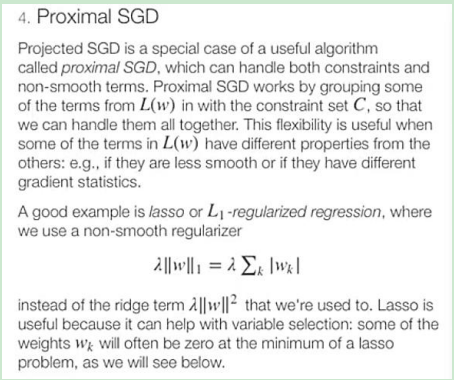
non-smooth 点不平滑就没有导数，可以把这个点导数赋予任意值，

再用 SGD

缺点：SGD 可能没办法 hit minimum of loss function

proximal SGD

解决：最接近的 proximal SGD 处理 non-smooth



4. Proximal SGD

Projected SGD is a special case of a useful algorithm called *proximal SGD*, which can handle both constraints and non-smooth terms. Proximal SGD works by grouping some of the terms from $L(w)$ in with the constraint set C, so that we can handle them all together. This flexibility is useful when some of the terms in $L(w)$ have different properties from the others: e.g., if they are less smooth or if they have different gradient statistics.

A good example is *lasso* or *L_1-regularized regression*, where we use a non-smooth regularizer

$$\lambda \|w\|_1 = \lambda \sum_k |w_k|$$

instead of the ridge term $\lambda \|w\|^2$ that we're used to. Lasso is useful because it can help with variable selection: some of the weights w_k will often be zero at the minimum of a lasso problem, as we will see below.

处理既有 constrain 和 non-smooth term 的优化，将 L 里的需要 constrain 的组织在一起处理 (提出来)

proximal SGD 会写如下形式：

For proximal SGD, we will write

$$L(w) = \ell_0(w) + \sum_{t=1}^{T} \ell_t(w)$$

where ℓ_0 is the term with different properties (e.g., the L_1 regularizer). We will group the constraint set C in with ℓ_0 by setting

$$\ell_0(w) = \infty \text{ if } w \notin C.$$

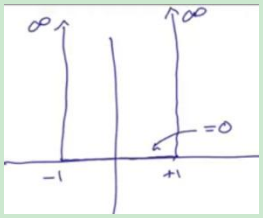
给 L0 term 增加一个约束(一旦不满足约束就变 infinite)，然后使得 L(w)最小

这样可以去掉约束条件，合并到我们的一个优化问题，只需解决这个优化问题，就能同时满足约束(消去了约束和拉格朗日思想相近)

注意：

1 L0 可能是 non-smooth（如 L0 term）

2 把约束，加在 L0 上(这样可以去掉约束条件)；一旦超过 C 就 infinite



如果 cost function 这样，约束在-1,1 之间

For proximal SGD, we will write

$$L(w) = \ell_0(w) + \sum_{t=1}^{T} \ell_t(w)$$

where ℓ_0 is the term with different properties (e.g., the L_1 regularizer). We will group the constraint set C in with ℓ_0 by setting

$$\ell_0(w) = \infty \text{ if } w \notin C.$$

其实就是让 L(w)最小的同时也要满足 C，如果不满足 C 就找离 L(w)最近的 w

上面公式其实等价于下面：

We'll also define a new concept, the *proximal* or *prox* operator: the prox operator for a function f is

$$\Pi_f(w) = \min_{w'} \tfrac{1}{2}\|w - w'\|^2 + f(w').$$

The prox operator generalizes projection: if we set f to be the indicator function of a set C

$$f(w) = \begin{cases} 0 & w \in C \\ \infty & w \notin C \end{cases}$$

then Π_f is exactly projection onto C.

prox operator 是个函数，参数是 SGD 更新后的 weight，返回选择符合条件的新的 weight

w 是 SGD 更新后的 weight w’是新的 weight，左边 term 还是让新的 weight 距离 w 尽量的近，右边只要不满足 constrain 就大大惩罚

最终返回满足这样的 w’

其实 $\min_{w'} \frac{1}{2}\|w - w'\|^2 + f(w')$ 等价于 $\min_{w'} \frac{1}{2}\|w - w'\|^2 \; s.t. v \in C$，而后者是 Projected SGD 的定义

2. Projected SGD

We'll start with a method for constrained optimization, called *projected SGD*. Projected SGD works when we can efficiently project onto the constraint set C: i.e., find the v that solves

$$\min_v \|v - w\|^2 \text{ s.t. } v \in C.$$

因此 Projection SGD 是 proximal 的特例，如果将 f 设置 indicate 函数

With this notation, proximal SGD repeatedly updates

$$w \leftarrow \Pi_{\eta \ell_0}(w - \eta g)$$

where g is a stochastic gradient estimate such as

$$g = T\nabla \ell_i(w) \quad i \sim \text{Uniform}(1, \ldots, T)$$

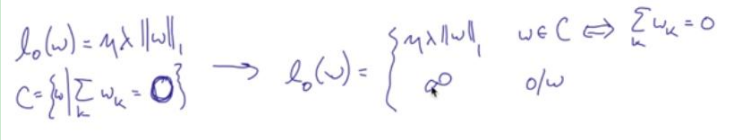
(or the equivalent estimate for a minibatch), and $\Pi_{\eta \ell_0}$ is the prox operator for the function $\eta \ell_0(w)$. Note that the learning rate η multiplies both the gradient estimate g and the function ℓ_0 that appears in the prox operator.

同样方法计算我们的更新后的参数 w-ng，然后给 proximal function，然后作为我们的新的参数值

Proximal SGD generalizes projected SGD: if we set ℓ_0 to be the indicator of the constraint set C, the proximal SGD update reduces to the projected SGD update. So, proximal SGD inherits many of the properties of projected SGD and plain SGD: it's a good idea to use preconditioning, decreasing learning rates, early stopping, and minibatches, and minibatches can save computation by letting us apply the prox operator less often. Also, proximal SGD is most useful when we can compute the prox operator efficiently.

proximal SGD Example： L0

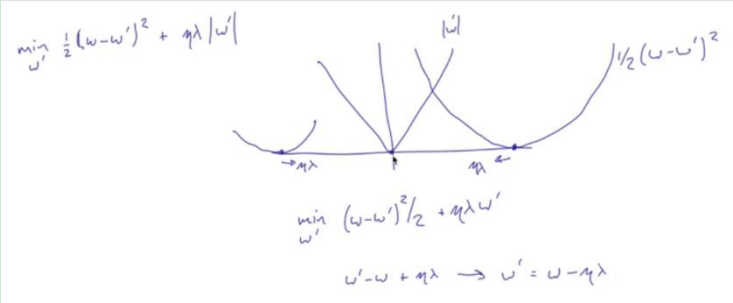
L0 term 本身是 non-smooth，然后再给它加上约束(proximal SGD)



将 constrain 合并在 L0 term
如果满足 constrain，term 不变，如果不满足，就 infinite，再将 L0 加上 $\min_{w'} \frac{1}{2}\|w - w'\|^2$，会找到尽量离 w 近的满足 C 的点

proximal SGD for Losso problem

如下图直观解释：



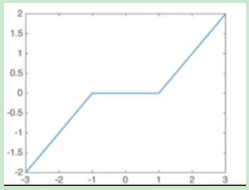
对于 L0
如果 w'大于 0，是右边情况，可以直接求导，w'=w-正数
w 是右边函数的最小值(w 是 constant,w' 是变量)，会使得 w'向 0 靠拢
同理如果 w'小于 0，会向右边 0 靠拢
最终 w'会接近 0

To see how proximal SGD generalizes projected SGD, we can derive the proximal SGD update for the lasso problem mentioned above. For this purpose, we need the prox operator for the function $\eta\lambda\|w\|_1$. A short derivation shows that, for each component separately:

$$[\Pi_{\eta\lambda\|\cdot\|_1}(w)]_k = \begin{cases} 0 & |w_k| \le \eta\lambda \\ w_k - \eta\lambda & w_k > \eta\lambda \\ w_k + \eta\lambda & w_k < -\eta\lambda \end{cases}$$

So, the proximal SGD update for lasso sets $w \leftarrow w - \eta g$, then *soft thresholds* each element of w independently: it moves w_k closer to 0 by $\eta\lambda$, and if it hits zero, keeps it there instead of changing sign. Here's the soft-threhold function for $\eta\lambda = 1$:

其实也是需要一个 prox operator，对于 SDG 更新后的参数 w，函数图形：



并且是对 w 的每个 component 独立的进行操作！！

The above expression shows that $\Pi_{\lambda\|\cdot\|_1}$ is efficient to compute. In addition, it shows why lasso encourages the weights w_k to remain at zero: there is a range of inputs w_k for which $[\Pi_{\lambda\|\cdot\|_1}(w)]_k = 0$, so unless the gradient is strong enough to move w_k out of this range, it will remain at zero.

Lagrange multiplier 解决更复杂的约束优化

Play games(SGD for minmax(0-sum) problem)

6. Playing games

So far, we've used SGD to solve unconstrained and simply-constrained optimization problems. It turns out that SGD is also a good algorithm for minimax problems, also called zero-sum games: that is, for finding w and α in

$$\min_w \max_\alpha L(w, \alpha)$$

where

$$L(w, \alpha) = \sum_{t=1}^{T} \ell_t(w, \alpha)$$

is a sum of terms. We still call L an objective function, just as in an ordinary optimization problem. Or, from the w player's perspective, we call L a loss function, and from the α player's perspective, we call it a *gain* or *payoff* function.

To solve such a game, the w player and the α player can run SGD at the same time: we repeatedly select an index i uniformly from $\{1, \ldots, T\}$, and update in parallel

$$w \leftarrow w - \eta T \frac{\partial}{\partial w} \ell_i(w, \alpha)$$
$$\alpha \leftarrow \alpha + \eta T \frac{\partial}{\partial \alpha} \ell_i(w, \alpha)$$

2 player w try min L + alpha try max L

loss function 和 gain function　　w 和 alpha run SGD same time，然后 w 和 alpha 同时更新！

只需一个朝 gradient 反方向和正方向

(By "update in parallel," we mean that we first compute both of the right-hand sides, then simultaneously assign to both of the left-hand sides.) Note that the α player is actually running stochastic gradient *ascent*, since she is trying to maximize $L(w, \alpha)$; for conciseness we'll still call it SGD for both players.

Just as for minimization, we can use projected SGD to implement constraints on w or α. We can also add separate regularizers for each player, and handle them with proximal SGD:

$$L(w, \alpha) = \ell_0(w) - \ell'_0(\alpha) + \sum_{t=1}^{T} \ell_t(w, \alpha).$$

For this purpose, the w player ignores the regularizer for the α player, and vice versa. As always, it's a good idea to use decaying learning rates, preconditioning, and minibatches. For a minibatch, we compute all of the gradients with respect to both w and α first, then update w and α in parallel.

分别的加独立的 constrain term

7. Convergence

The convergence guarantee for SGD is a bit more subtle in games than it was for minimization problems. The reason is that we sometimes need *randomization*: e.g., imagine if we always played rock in rock-paper-scissors. The guarantee is

对于 SGD 的 convergence 有点复杂

石头 剪刀 布--每个 player 一个自己的 distribution

想赢比赛不能只出一种，需要 randomize！

always played rock in rock-paper-scissors. The guarantee is this: if we randomly select an iteration and play the w from the end of that iteration, then our distribution over w converges to (locally) optimal play. Similarly, if we randomly select another iteration and play the α from the end of that iteration, then our distribution over α converges to (locally) optimal play.

convergence guarantee：每个 iteration 更新参数时(每个 iter 一个 w 值)，从前面的所有 iteration random choose w，作为本次 iteration 的 w 值，就能保证 converge 到最好的 local optimal

In case the objective is convex in w, the situation is slightly simpler: we can play the *average* of all of the w vectors, across all iterations, instead of randomizing. Similarly, if the objective is concave in α, we can play the average of all α vectors.

对于 concave 和 convex 函数，只需 play average of 全部 w

Lagrange multiplier 乘子

拉格朗日用来解决更加复杂的 constrain，proxmal SGD 已经不行了

将 constrain 看作博弈论！！

8. Lagrange multipliers

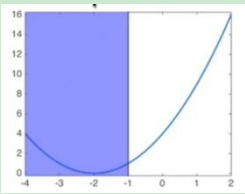
We can use SGD on zero-sum games as a tool to help us solve constrained optimization problems — especially, problems with complicated constraints for which projected SGD is too inefficient.

To do so, we invent an α player to enforce the constraints. That is, we set up the α player so that, if the w player goes outside the constraint set C, she is guaranteed to suffer high loss. To see how this works, consider the (very simple) constrained optimization problem

$$\min_w (w+2)^2 \text{ s.t. } w \geq -1 .$$

对于复杂的 constrain，project SGD 计算量太大

引入另一个 player 来惩罚 go outside constrain C



$L(w)$ is shown as a solid curve; the constraint forbids the shaded region. Without the constraint, the minimum would be at $w = -2$; with the constraint, the minimum is at $w = -1$.

换一种形式： 2player！ 用 game 解决优化问题

Now consider the game

$$\min_w \max_\alpha (w+2)^2 - \alpha(w+1) \text{ s.t. } \alpha \geq 0 .$$

把**约束放在式子里**，如果 w=-2 w+1=-1 alpha 为了 max 会无穷大

w 如果是 0， w+1=1 **alpha 会取 0**

w player 必须大于等于-1,等价于上面条件约束

如果 w+1<-1,alpha 正无穷，式子也是正无穷，**w 会非常不满意**，所有 w 一定不会选 w+1<0，只会选 w+1>=0 也就是 w>=-1 ，**使得 alpha(w+1)=0**

Therefore, the game implements our original optimization problem: in any solution, the w player must pick w to minimize $(w+2)^2$ while obeying $w \geq -1$. The new variable α is called a *Lagrange multiplier*, after the mathematician who invented this idea.

这种构建方法一定有规律，alpha 称为 langrange multiplier

multiple inequality constrain：

9. Lagrange multipliers for inequality constraints

We can extend the Lagrange multiplier idea to handle multiple linear inequality constraints

$$A_j \cdot w \geq b_j \quad j \in \{1, \ldots, J\}.$$

To keep notation concise, we collect all of the constraint vectors, one per row, into a matrix A, and all of the constants into a vector b, and write

$$Aw \geq b$$

where the inequality is interpreted componentwise.

To implement multiple inequality constraints, we define a vector α of Lagrange multipliers, with one element per constraint (i.e., per row of A). We then add

$$\alpha^T(b - Aw)$$

to the objective $L(w, \alpha)$, and constrain

$$\alpha \geq 0$$

componentwise.

拉格朗日乘子 alpha 一定是正数！！(唯一的约束)

多个约束组织成矩阵和向量，定义一个拉格朗日乘子向量将 $\alpha^T(b - Aw)$ **加在目标函数上，alpha 让目标函数 max 最大！！**

规律：如果是大于号就将参数右移，乘以拉格朗日乘子

It is straightforward to verify that the same reasoning as in the previous section holds for each component of α: α_j will be 0 if the jth constraint is strictly satisfied, $\alpha_j \to \infty$ if the jth constraint is violated, and α_j doesn't matter if the jth constraint is satisfied with equality. So, the w player will play so that $Aw \geq b$, and the product $\alpha^T(b - Aw)$ will vanish from the objective.

It may seem that we haven't made any progress: we replaced one set of inequalities, $Aw \geq b$, with another, $\alpha \geq 0$. But intuitively the second set of constraints seems simpler; and in fact, we will see below that this difference leads to an efficient algorithm.

只要 w 满足 Aw>b，拉格朗日部分就为 0，因为 b-Aw 小于 0 了，为了让目标函数最大，alpha 只能是 0，这部分就是 0

Note that we have written all of our inequalities with $\geq$ instead of $\leq$. This convention is without loss of generality, since we can negate any $\leq$ constraint to turn it into a $\geq$ constraint.

将约束全部统一为大于号(如果是小于号，就乘以-1 变成大于)

equality constraint

10. Lagrange multipliers for equality constraints

We can extend the Lagrange multiplier idea to handle linear equality constraints as well:

$$Aw = b.$$

To do so, we add

$$\alpha^T(b - Aw)$$

to the objective $L(w, \alpha)$, and leave the Lagrange multiplier vector α unconstrained. The reasoning is similar to the constrained case: $\alpha_j \to \infty$ if jth constraint is violated with $[b - Aw]_j > 0$; $\alpha_j \to -\infty$ if the jth constraint is violated with $[b - Aw]_j < 0$; and α_j doesn't matter if the jth constraint is satisfied. So, the w player will play so that $Aw = b$, and the product $\alpha^T(Aw - b)$ will vanish from the objective.

还是右移，但 alpha 不再约束为正 alpha unconstrained

因为如果 b-Aw 小于 0，alpha 会取负无穷大，如果 b-Aw 大于 0，alpha 会取正无穷大，为了 max lossmax loss，这样 w 就非常不愿意，w 只能取 b-Aw=0 的情况，此时 alpha 取 0

只有等于 0，才最优

We can of course mix the two types of constraints: the only difference is whether we require $\alpha_j \geq 0$ (for inequality constraints) or leave α_j free to vary (for equality constraints).

也可以混合约束，equality 的 alpha unconstrained ，inequality 的 alpha，constrain>0

我们最终的问题变成：

12. Solving it

Now we have a problem of the form

$$\min_w \max_\alpha L(w) + \alpha^T(b - Aw) \text{ s.t. } \alpha \in C$$

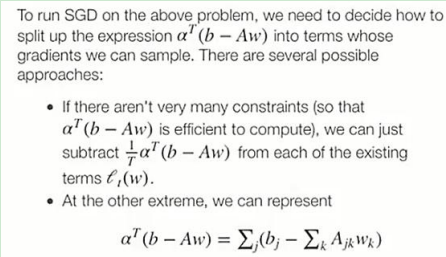
where the constraint set C requires some components of α to be nonnegative, and leaves other components of α free.

Why did this help? Before, we had constraints of the form $Aw = b$ or $Aw \geq b$; these constraints were difficult to project onto. Now we only have constraints of the form $\alpha_j \geq 0$; these are trivial to project onto, since all we have to do is set α_j to zero if it was negative. So, we can now solve our game (and therefore our original optimization problem) by running projected SGD simultaneously for the two players.

上面等式约束和不等式约束的最优都是在 w 满足约束的情况下，alpha=0，因此只需 project alpha to 0，project algorithm 高效

也就是从复杂的 constrain set，转换成简单的 constrain set 就可以 run projected SGD 了

因此，可以用 projection SGD 来优化！！！


To run SGD on the above problem, we need to decide how to split up the expression $\alpha^T(b - Aw)$ into terms whose gradients we can sample. There are several possible approaches:

- If there aren't very many constraints (so that $\alpha^T(b - Aw)$ is efficient to compute), we can just subtract $\frac{1}{T}\alpha^T(b - Aw)$ from each of the existing terms $\ell_t(w)$.
- At the other extreme, we can represent

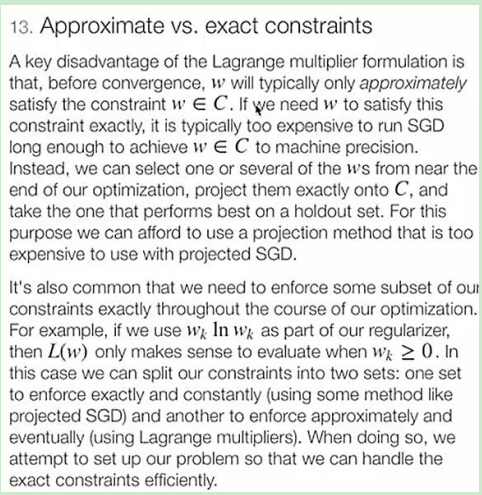
$$\alpha^T(b - Aw) = \sum_j (b_j - \sum_k A_{jk} w_k)$$

为了 SGD,需要每次 sample 部分 term，需要把约束项也分解到每个 term 里

方法 1 当约束少时，直接分成 T 份(add 不是 subtract)

方法 2：可以将向量或矩阵变成合的形式

update both alpha and w

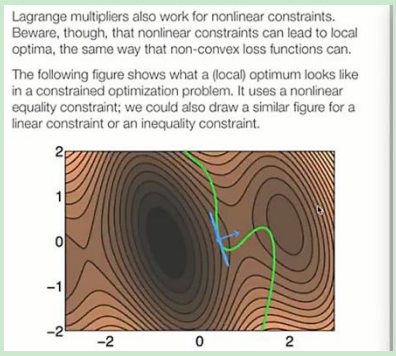

13. Approximate vs. exact constraints

A key disadvantage of the Lagrange multiplier formulation is that, before convergence, w will typically only *approximately* satisfy the constraint $w \in C$. If we need w to satisfy this constraint exactly, it is typically too expensive to run SGD long enough to achieve $w \in C$ to machine precision. Instead, we can select one or several of the ws from near the end of our optimization, project them exactly onto C, and take the one that performs best on a holdout set. For this purpose we can afford to use a projection method that is too expensive to use with projected SGD.

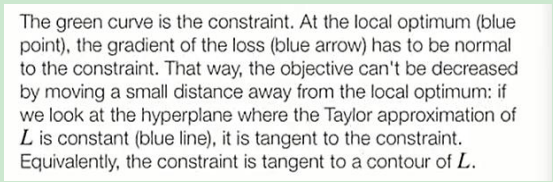
It's also common that we need to enforce some subset of our constraints exactly throughout the course of our optimization. For example, if we use $w_k \ln w_k$ as part of our regularizer, then $L(w)$ only makes sense to evaluate when $w_k \geq 0$. In this case we can split our constraints into two sets: one set to enforce exactly and constantly (using some method like projected SGD) and another to enforce approximately and eventually (using Lagrange multipliers). When doing so, we attempt to set up our problem so that we can handle the exact constraints efficiently.

在 convergence 之前，w 不一定满足 C 约束

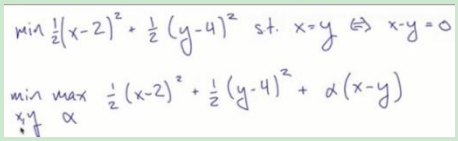
Non-linear constrain



Non-linear constrain 会导致 local optima



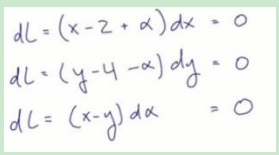
例子：



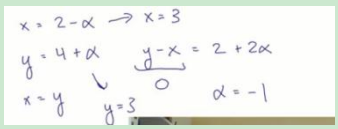
我们不需要优化算法，直接解：

分别求 x y lapha 三个参数的微分，但对于复杂问题，没办法 solve

只能 SGD



因为 d 微分不会是 0，因此括号应该是 0



Dicrete optimization

1. Discrete optimization in ML

In the last few sets of lecture notes, we've focused on continuous optimization. Many ML problems also require discrete or mixed optimization: that is, all or some of the variables are discrete. Unfortunately, discrete and mixed optimization problems tend to be harder than purely continuous ones. Fortunately, though, it's often possible to find effective heuristics for solving these problems in practice.

variable 是 discrete 或 category，需要单独的考虑和 decision

Another example of a mixed discrete/continuous optimization problem is a mixture model: instead of learning one model that fits all of our data, we learn several models, each of which fits a subset of our data. We create new discrete variables to indicate which model is responsible for each example.

There are lots of different kinds of mixture models, depending on how we model each subset of our data. If the individual models are Gaussian distributions, the result is called a Gaussian mixture model, and is related to the popular k-means clustering method. Or, outlier detection is another example of a mixture model: the outliers form one subset, and the inliers form another.

mixed modle：split data into subset，每个 subset learn 不同的 model，就会多一个离散变量，代表每个 example 属于哪个 model

例子：高斯混合模型：fit 多个高斯模型，需要为每个 data 决定属于哪个高斯分布

One good example of a mixed optimization problem in ML is regression with outlier detection: for each of our examples, we keep a binary variable that indicates whether it is an outlier. Then, we fit our regression model to the inliers; we fit the outliers with a different, simpler model such as a uniform distribution with a wide support.

每个 example 两类：outliner　or　nooutlier

outlier detect 也是**混合模型**，对 inlier 建立 regression model

对 outlier 建立一个简单的 model，如 uniform

coordinate descent 对于有 global optima 的 single 变量更有效！！

example lasso

A good example of a problem where coordinate ascent works well is lasso:

$$\min_w \tfrac{1}{2}\|Y - w^T X\|_F^2 + \lambda\|w\|_1 .$$

(Since it's a minimization problem, we're actually using coordinate *descent*, but the algorithm is essentially the same.) In lasso the discrete optimization aspect comes from the discrete choice of whether to set each weight w_k to zero.

因为需要 discrete 的单独的考虑 whether set 每个 weight to 0，因此刚好用 coordinate descent 来解决

If we hold all but one component w fixed in the above objective, we get a problem of the form

$$\min_{w_k} Q(w_k) + \lambda|w_k|$$

where $Q(w_k)$ is a one-dimensional quadratic function. So, we can solve for w_k exactly and efficiently. Naively we would have to recompute the one-dimensional quadratic function from scratch in each new iteration, but it turns out that with careful bookkeeping we can update the 1-D quadratics efficiently and incrementally. So, coordinate descent for lasso is a fast algorithm — in fact, one of the best for solving large-scale lasso problems.

每次只考虑一个 weight，然后 optimize objective，会变成 1 dimension objective check 取值正，负，0 对 objective 的影响

example Kmean

8. Spherical clustering (k-means)

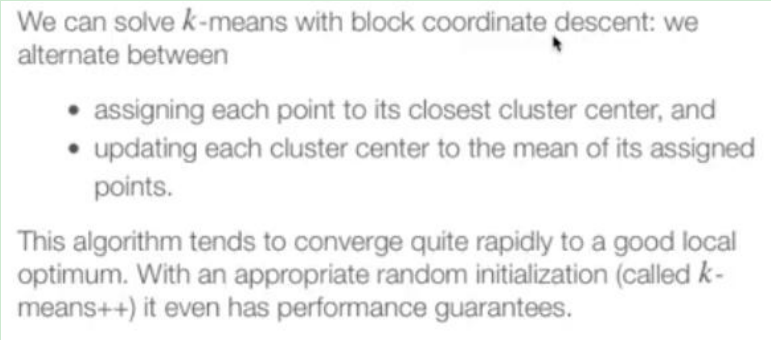
Another good example problem is k-means clustering: we have a sample $X^1, \ldots, X^T$. We want to pick cluster centers $\mu_1, \ldots, \mu_k$ and assign each point X^i to a cluster center in a way that minimizes the total distance of points from their assigned centers. If we write $Z_{ij} = 1$ if X^i is assigned to μ_j, and $Z_{ij} = 0$ otherwise, then we want to solve

$$\min_{\mu_j, Z_{ij}} \sum_{i=1}^{T} \sum_{j=1}^{k} Z_{ij}\|X^i - \mu_j\|^2 .$$

This problem is called k-means since the optimal value for each of the k cluster centers is the mean of its assigned points.

另一个 coordinate descent 例子：Kmean 聚类，pick the center u

Z 是 indicate variable 如果 Xi assign to uj，因此使得 Xi 到 uj 的距离最小
最终得到我们 pick 的 uj 和，并得到 Z

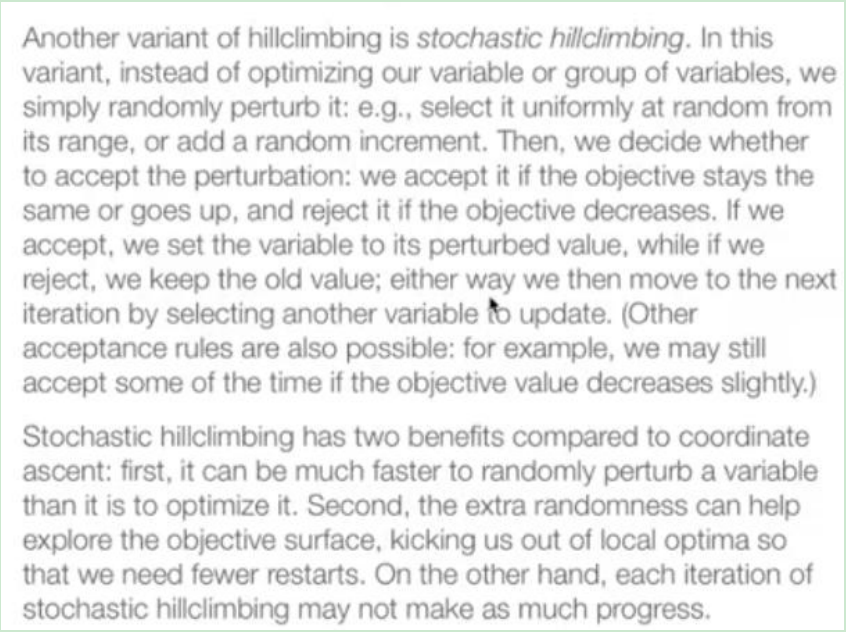

We can solve k-means with block coordinate descent: we alternate between

- assigning each point to its closest cluster center, and
- updating each cluster center to the mean of its assigned points.

This algorithm tends to converge quite rapidly to a good local optimum. With an appropriate random initialization (called k-means++) it even has performance guarantees.

可以用 **block coordinate descent**，也就是 hold the Z，优化 u(average enter of 每个 cluster)
或者 hold the u，然后再确定每个 Z，也就是每个 X 的类别

Stochastic hillclimbing


Another variant of hillclimbing is *stochastic hillclimbing*. In this variant, instead of optimizing our variable or group of variables, we simply randomly perturb it: e.g., select it uniformly at random from its range, or add a random increment. Then, we decide whether to accept the perturbation: we accept it if the objective stays the same or goes up, and reject it if the objective decreases. If we accept, we set the variable to its perturbed value, while if we reject, we keep the old value; either way we then move to the next iteration by selecting another variable to update. (Other acceptance rules are also possible: for example, we may still accept some of the time if the objective value decreases slightly.)

Stochastic hillclimbing has two benefits compared to coordinate ascent: first, it can be much faster to randomly perturb a variable than it is to optimize it. Second, the extra randomness can help explore the objective surface, kicking us out of local optima so that we need fewer restarts. On the other hand, each iteration of stochastic hillclimbing may not make as much progress.

不是做 optimize，而是**随机的给值 perturb value**，然后值逐渐的增大，如果 object 增加就 accept，如果减少就忽略，继续，这样会使得 objective 一直增加！！！
1 随机赋值比 optimize 更快
2 更好的 explore，脱离 local optimal
就是速度慢一点！！！

概率推断
一些是 discrete 随机变量一些连续随机变量

10. Probabilistic inference

In ML, one way that discrete optimization problems often arise is *probabilistic inference*: we start from a probability distribution

$$P(X_1, X_2, \ldots, X_n)$$

over discrete (or mixed discrete and continuous) variables $X_1 \ldots X_n$. We observe (condition on) some of the variables; then we want to marginalize out some of the other variables, and maximize the probability of the remaining ones. We'll write C for the indices of the variables we want to condition on, M for the indices of the variables we want to marginalize out, and O for the indices of the variables that we want to optimize: so, we want to compute

$$\max_{X_O} \sum_{X_M} P(X_O, X_M \mid X_C) .$$

我们观察到了一些变量，是已知条件 C
M 是想 marginalize 去掉 对 M 求和
O 是想 optimize

EXAMPLE 高斯混合模型

A good example of a probabilistic inference problem is a Gaussian mixture model — the probabilistic version of k-means. If we think of a point's cluster assignment as a categorical random variable, and its distance from its cluster center as a Gaussian random variable with variance σ^2, then the log-probability of X^i, μ_j, and Z_{ij} is

$$-\frac{1}{\sigma^2} \sum_{i=1}^{T} \sum_{j=1}^{k} Z_{ij} \|X^i - \mu_j\|^2 + \text{constant} .$$

(Here we have assigned uniform prior probabilities to the cluster centers and cluster assignments, and interpreted $(Z_{ij})_{j=1\ldots k}$ as a

高斯混合模型 是分类随机变量 question
也类似于 kmean

为了解决上面的问题，用的算法：

11. Gibbs sampling

We will start from the case where there are no variables to optimize: $O = \emptyset$, and we simply want to know the conditional distribution of the variables in M,

$$P(X_M \mid X_C).$$

For this purpose we can use a method similar to stochastic hillclimbing, called *Gibbs sampling*. In Gibbs sampling, we repeatedly pick a variable from x_M, and set it to be a sample from its conditional distribution given the remaining variables.

类似于爬山算法

For example, in the Gaussian mixture model, it is easy to derive that

$$P(Z_{ij} \mid X^i, \mu_1 \ldots \mu_k) \propto e^{-\frac{1}{2\sigma^2}\|X_i - \mu_j\|^2}$$

and that the conditional distribution of μ_j is Gaussian with mean

$$\frac{\sum_i Z_{ij} X^i}{\sum_i Z_{ij}}$$

and variance

$$\frac{\sigma^2}{\sum_i Z_{ij}}.$$

So, to run Gibbs sampling, we can alternate in blocks: first we repeatedly select an example X^i, resample its cluster membership, and update Z_{ij} for all j. Then, we run through each of the clusters and resample μ_j.

Unlike hillclimbing, we can't use the values of the variables to select which variable to update next: we can only use information that we know before starting the sampler, such as the connectivity of the graph. For example, it's OK to pick a variable if lots of its neighbors have been recently updated; it's not OK to pick a variable if its neighbors have recently had *large* updates.