目录

2018-4-29

Topics and schedule

The main topics of the course include

- Logic
- Sets and types
- Probability
- Linear algebra and vector spaces
- Functions and function spaces
- Matrices and linear operators
- Matrix differential calculus
- Statistics and inference

Syllabus for mini 1

Lecture outline
Note: this outline is subject to change — future dates are predictions not guarantees

Date	Topic	Notes/materials
	Logic and sets	
8/28	Intro, sets	Sets: Bertsekas, sec 1.1 https://qna.cs.cmu.edu/#/pages/view/144
8/30	Types	https://qna.cs.cmu.edu/#/pages/view/145
9/4	Labor Day — no class	
9/6	Logic	
	Probability	
9/11	Probability	See also Bertsekas ch 1 (on probability)
9/13	Probability	See also Bertsekas ch 2 (on random variables and expecta
	Linear algebra; functions and function spaces	
9/18	Linear algebra; functions and function spaces	See also Ch 0 of Horn & Johnson
9/20	Functions; probability examples	
9/25	Matrices and linear operators	See also Ch 0 of Horn & Johnson
	Matrix calculus	
9/27	Linear algebra review	

10/2	Matrix calculus	
10/4	Matrix calculus	
	Statistics and inference	
10/9	Statistics and inference	
10/11	Statistics and inference	
10/16	**Exam (in class) (no lecture stream today)**	
10/16	Mini-1 last day of classes	

Textbooks

There are no required textbooks for this course. However, the following textbooks are recommended as a way to get an alternate presentation of some of the course material:

- Matrix Analysis (2nd ed.). Roger A. Horn, Charles R. Johnson. Cambridge University Press, 2013.
- Introduction to Probability (2nd ed.). Dimitri P. Bertsekas, John N. Tsitsiklis. Athena Scientific, 2008.
- The Elements of Statistical Learning (2nd ed.). Trevor Hastie, Robert Tibshirani, Jerome Friedman. Springer, 2008.

Logic and se(任何 question 从 16 版视频找答案更细致)

Sets

unordered no repeation collection

notation :

$$\text{primary\_color} = \{\text{red}, \text{green}, \text{blue}\}$$

or by giving a recipe for constructing its elements

$$\text{even\_number} = \{2x \mid x \in \mathbb{Z}\}.$$

The general form of a recipe is

$$S = \{\text{expression} \mid \text{logical property}, \text{logical pro}$$

Z 是整数

N nature num

$$\mathbb{N} = \{x \mid x \in \mathbb{Z}, x \geq 0\} = \{x \in \mathbb{Z} \mid x \geq 0\}$$

python: set 的构建

Python code

{x**2 for x in range(4)}

implements the set builder notation

$$\{x^2 \mid x \in \{0,1,2,3\}\}$$

coding:

1{2*x for x in{1,2,3}}

2 给一个 set 返回不被 2,3,4,5 整除的元素 set

加 if 条件

```
def notdiv(nums):
  return {x for x in nums if (x%2 != 0) and
    (x%3 != 0) and (x%5 != 0)}
```

3 给一个字符串 set，返回字符串只包含 a-m 的字符串集合 正则表达式

{cat, dig, dog, ant, eel}

the function should return

{dig, eel}.

Hint: a good way to test whether a string is OK is to use regular expressions. The following code

re.match(r'^[a-mA-M]*$', 'dig')

也是判断 if

```
import re

def notdiv(nums):
  return {x for x in nums if (x%2 != 0) and (x%3 != 0)
    and (x%5 != 0)}

def onlyAtoM(strs):
  return {x for x in strs if re.match(r'^[a-mA-M]*$',
    x)}
```

5. Union, intersection, difference, complement

The basic operations on sets are union $\cup$ and intersection $\cap$: in set builder notation,

$$X \cap Y = \{x \mid x \in X \wedge x \in Y\},$$

$$X \cup Y = \{x \mid x \in X \vee x \in Y\}.$$

Another useful operation is set difference: everything that's in one set but not another,

$$X \setminus Y = \{x \in X \mid x \notin Y\}.$$

(Note the use of $\notin$ as a shorthand for $\neg(x \in Y)$.)

complement set

Finally, we'll sometimes fix a universe U: a set that contains all objects that we're considering using as elements for other sets. If we have fixed a U, the complement of a set S means everything in U that's not in S:

$$\bar{S} = U \setminus S.$$

Tuple and set product

tuple have order 的 list

python:

```
(1,2,3,4)
(1, 2, 3, 4)

((1,2),(3,4))
((1, 2), (3, 4))
```

2 个元素的 tuple (x,y)

set product: all possible pair in 2 set, set 各取一个元素组成 tuple，但是固定顺序，X 在前，Y 在后！！确定顺序后的所有组合！！

Given sets X and Y, we can write the *set product*

$$X \times Y = \{\langle x, y\rangle \mid x \in X, y \in Y\}$$

X Y 可以不可以相互交换，可以结合：

$$(X \times Y) \times Z = X \times (Y \times Z) = X \times Y \times Z$$

set of tuples

多个 tuple

We will write X^n to mean the set of n-tuples with all elements from X: e.g.,

$$\langle 3, 1, 4, 1, 5\rangle \in \mathbb{Z}^5 .$$

tuple 可以多种类型，因此可以作为函数的参数

```
(1,'k',9.3)
(1, 'k', 9.3)
```

excercise1

DB join set of tupel

数据库每个表没有重复记录，每一行可以当做 tuple(固定表结构，field 顺序)，every record not repeation！每个表是 set of tuple

两个 table，join by same field，create a new tuple

数学表示：最终结果是 set of tuples！！

$$\{\langle x, y, z\rangle \mid \langle x, y\rangle \in A, \langle y, z\rangle \in B\}$$

DataType and function

1 datatype 是 implementation of set

set product:

Programming languages provide operations for combining types: e.g., in C,

struct { int a, float b }

defines the type $\mathbb{Z} \times \mathbb{R}$ of pairs of an integer and a real number. (At least, up to the limits of finite-precision

union:

一种类型既可以当做 float，又可以当做 int，包含两种数

可以近似 set 的 union

union { int a, float b }

defines the type containing (binary representations of) integers together with (binary representations of) reals.

tagged union，每个 element 有属于原先集合的 label，得到的 union，我们知道每个元素的 original label

$$\mathbb{Z} \times \{1\} \cup \mathbb{R} \times \{2\}$$

先做 set product，得到

Function

也是一种 combine different types 的方法

If we have a function $p \in X \to Y$, then for every $x \in X$, there is a unique value $p(x) \in Y$. That is, functions are *complete* (defined for every input value) and *single-valued* (never have ambiguous outputs).

Anonymous Function

Lmbda:

$$\lambda x, y.\ x^2 + 3xy$$

is a function of type

$$\mathbb{R} \times \mathbb{R} \to \mathbb{R};$$

$$\lambda x : \mathbb{R}, y : \mathbb{R}.\ x^2 + 3xy$$

python：可以用于函数的传递

```
lambda x, y: x*x + 3*x*y
```

```
a=lambda x:x*2

a(3)
6
```

excercise ： **function** as parameter (传递函数的引用)
function：(R->R) 作为**函数参数**

```
In [1]: a=lambda x:x*2

In [2]: a(3)
Out[2]: 6

In [3]: def b(a):
   ...:     return a(7)+1
   ...:

In [4]: b(a)
Out[4]: 15

In [5]:
```

可以在函数里调用另一个函数！！！需要 lambda 匿名函数

还可以**函数返回函数**：

```
def plus1(f):
  return lambda x: f(x)+1
```

plus1(lamda x:x) 参数是函数，返回的还是个 function 对象，是 lambda x: （lambda x:x）+1 还可以传参数，因此可以写成下面：

plus1(lamda x:x)(3) 返回 4，参数 x=3 先传递到外层，再传递到内层，返回时先执行返回里面 lamda 函数，然后再返回外层

lambda 就是个函数！！！ ：后面是函数体 ：前面是参数

Propositional logic

logic 是 argument
用于推理模型直到结果

- state assumptions clearly
- describe, for each step, which earlier assumptions or conclusions are inputs, and what conclusion we're now drawing for later use
- organize arguments into digestible (and independently checkable) chunks
- write out everything in enough detail that anyone who comes along later can follow the argument.

2. Propositional logic

The reasoning system called propositional logic is at the heart of many other, more complicated reasoning systems. In propositional logic, variables $a, b, \ldots$ (called *propositions* or *predicates*) represent truth values T or F, and connectives $\vee, \wedge, \neg, \rightarrow$ represent OR, AND, NOT, and IMPLIES.

Alternate notations include $\bar{a}$ or $\sim a$ for $\neg a$; $a \supset b$ for $a \rightarrow b$; $a \,\&\, b$, $a \cdot b$, or ab for $a \wedge b$; $a \mid b$ or $a + b$ for $a \vee b$; $\top$ for T; $\perp$ for F. Falsehood F is sometimes also called *absurdity* or *contradiction*.

1 变量是 proposition(值是 T or F)
2 connectives combine variable，**can compose logical statement**
连接以后的结果还是 True or False
一般提出一个 proposional logic 默认是 true!

inference rule and proof

inference rule ： modus ponens

inference rules. The most famous of these is probably modus ponens: from premises ϕ and $\phi \rightarrow \psi$, conclude ψ. The premises ϕ and $\phi \rightarrow \psi$ are assumptions or previously proven statements; ϕ and ψ can be single variables or more complex statements containing connectives.

从 premise 推 conclude

proof：将 statement 用 inference rule 连接得到 proof

We can put several statements connected by inference rules together into a proof. For example: to say "Socrates is a man; if Socrates is a man then Socrates is mortal; therefore Socrates is mortal", we can write

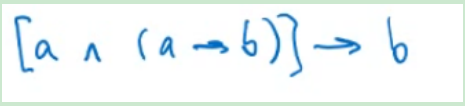
1. $\mathrm{man}(\mathrm{Socrates})$ [assumption]
2. $\mathrm{man}(\mathrm{Socrates}) \rightarrow \mathrm{mortal}(\mathrm{Socrates})$ [assumption]
3. $\mathrm{mortal}(\mathrm{Socrates})$ [modus ponens; 1, 2]

2 column prof

statement 在左 justification 在右(要么是 asumption，要么是 inference rule)

Lemmas

将 proof 的过程抽象化：左边是条件，右边是结论



This process — summarizing a proof or part of a proof by collecting together its assumptions and conclusion — is called "implication introduction" or "proving a lemma".

To go with implication introduction, modus ponens is sometimes called "implication elimination."

collection the assum 和 conlusion 更紧凑

5. Other inference rules

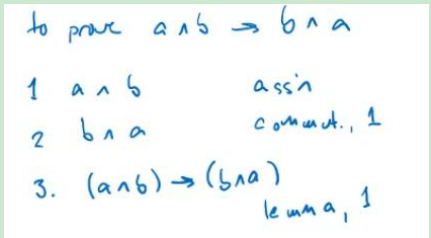
There are lots of useful inference rules besides implication introduction and implication elimination. For example:

- $\wedge$ introduction: if we separately prove ϕ and ψ, then that constitutes a proof of $\phi \wedge \psi$.
- $\wedge$ elimination: from $\phi \wedge \psi$ we can conclude either of ϕ and ψ separately.
- $\vee$ introduction: from ϕ we can conclude $\phi \vee \psi$ for any ψ.
- $\vee$ elimination (also called proof by cases): if we know $\phi \vee \psi$ (the cases) and we have both $\phi \to \chi$ and $\psi \to \chi$ (the case-specific proofs), then we can conclude χ.
- T introduction: we can conclude T from no assumptions.
- F elimination: from F we can conclude an arbitrary formula ϕ. (This rule is sometimes called *ex falso* or *ex falso quodlibet*, from the Latin for "from falsehood, anything.") This rule can be counterintuitive, but one way to think about it is this: we should never be able to prove F, so there's no danger in letting ourselves prove an arbitrary formula given F.
- Associativity: both $\wedge$ and $\vee$ are associative: it doesn't matter how we parenthesize an expression like $a \wedge b \wedge c \wedge d$. (So in fact we often just leave the parentheses out.)

6. Exercise

Use the above inference rules to prove

$$(a \wedge b) \to (b \wedge a).$$



因为操作可以交换 communicate(**inference rule**)，最后得出 **lemmas**

Negation

first way：constructive logic

In both cases we define $\neg\phi$ as a shorthand for $\phi \to F$. That is, $\neg\phi$ is true precisely when ϕ would lead to a contradiction.

another way **classical Propositional logic**

- law of the excluded middle: $\phi \vee \neg\phi$ holds for any statement ϕ.

给定 fi，要么 fi 要么非 fi 没有中间

There are in fact many inference rules that we could add instead of (or in addition to) the law of the excluded middle, including

- De Morgan's laws: $\neg(\phi \vee \psi)$ is equivalent to $\neg\phi \wedge \neg\psi$, and $\neg(\phi \wedge \psi)$ is equivalent to $\neg\phi \vee \neg\psi$.
- double negation elimination: $\neg\neg\phi$ is equivalent to ϕ.
- contraposition: $\phi \to \psi$ is equivalent to $\neg\psi \to \neg\phi$.

还有其他的**推理规则：**

Resolution (inference rule)

Suppose we have two statements of the form

$$\phi \vee \chi \qquad \neg\phi \vee \psi.$$

Here ϕ, χ, ψ may be any formulas of propositional logic. Note that ϕ appears in both formulas: negated in one, but not the other.

Given the above formulas, resolution lets us conclude

$$\chi \vee \psi.$$

That is, we delete ϕ and $\neg\phi$ from our formulas, and join together what's left using $\vee$.

也就是假设上面两个成立，结果都是 true
1 如果 phi true， K(X)无所谓 true false，但 si 是 true
2 如果 phi false， K(X)是 true si 无所谓 T F
也就是 K 和 si 至少有一个是 true，因此 **k V si 也是 true 成立**
通过这个可以将 **fomula k 和 si 组合了！！**

Resolution on clauses

clause（long disjunctions 析取符号）

The result is something like this: from

$$a \vee b \vee \neg c \vee d \vee \neg e \qquad b \vee \neg d \vee f$$

we use resolution on the literal d (which appears positively in the first clause and negatively in the second) to conclude

$$a \vee b \vee \neg c \vee \neg e \vee f\,.$$

Note that we have applied one additional (commonly needed) simplification: the literal b appears in both input clauses, but only shows up once in the output, since $b \vee b \leftrightarrow b$.

跟上面的一样，两边都有 d，然后消去

Scoping rules

对于复杂的 proof，需要有多个 lemma 时，prove 一个 lemma 的过程可能会 prove another lemma，得到一些 **useful 子过程(nested lemma)**

1. Lemma:
    1. Assume: $PB \wedge J \rightarrow Sandwich$
    2. Lemma:
        1. Assume: PB
        2. Lemma:
            1. Assume: J
            2. $PB \wedge J$ [from 2.a, 2.b.i, $\wedge$-introduction]
            3. $Sandwich$ [from 2.b.ii, 1, modus ponens]
        3. End lemma: $J \rightarrow Sandwich$
    3. End lemma: $PB \rightarrow (J \rightarrow Sandwich)$
2. End lemma: $[PB \wedge J \rightarrow Sandwich] \rightarrow [PB \rightarrow (J \rightarrow Sandwich)]$

每个 lemma 一个 scope，每个 lemma 的 assumption 只在当前 scope 有用！！

——》implication 推断

每个 implication 都有 start a new lemma！！！

里面一层：**assume PB**,然后得到 **lemma：J-->Sandwich**

就可以得到下一个 **lemma：PB-->（J-->Sandwich）**

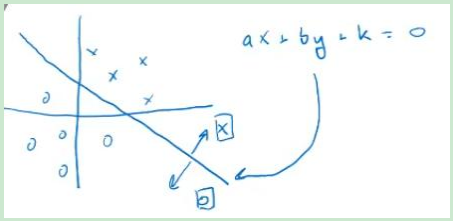
implication 是 assume A 得到 B 就是 A—>B

excercise: perception algorithm(linear discriminator)

为了学习到 linear discriminator 的参数

学习参数的一种方法

linear discriminator（discriminate 2 classes）



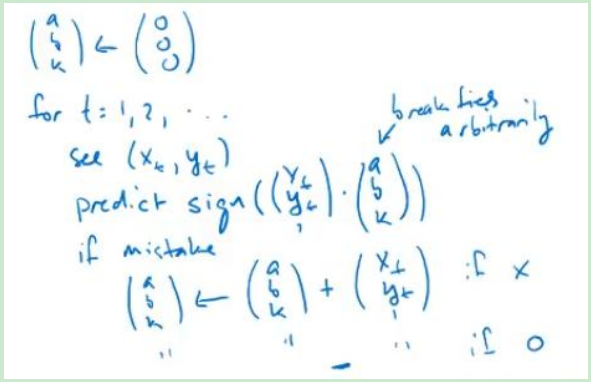
perception algorithm is a way to learn **linear discriminator**

转化成矩阵形式，a,b,k 是我们需要 learn

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \bullet \begin{pmatrix} a \\ b \\ k \end{pmatrix} = 0$$

结果>0 就预测正样本，结果小于 0 就预测负样本

perception algorithm：



先 initialize parameter 为 0 ，遍历所有 dataset，predict，**如果错误，对于正样本参数就加上 example，对于负样本就减(更新参数)，只有错误了才会对 loss 有变化，才会有 gradient，正确预测，没有更新！！**

3. The perceptron algorithm

To learn from streaming data, we can use any of several algorithms. One of the simplest is the *perceptron algorithm*: we initialize our parameters as $w_1 = (0, 0, 0)^T$. As long as we predict correctly, we leave our parameters unchanged, $w_{t+1} = w_t$. Whenever we make a mistake, we update our parameters: if we should have predicted x we set

$$w_{t+1} = w_t + u_t$$

and if we should have predicted o we set

$$w_{t+1} = w_t - u_t$$

This rule makes sense since

$$w_{t+1} \cdot u_t > w_t \cdot u_t$$

in the first case, and

$$w_{t+1} \cdot u_t < w_t \cdot u_t$$

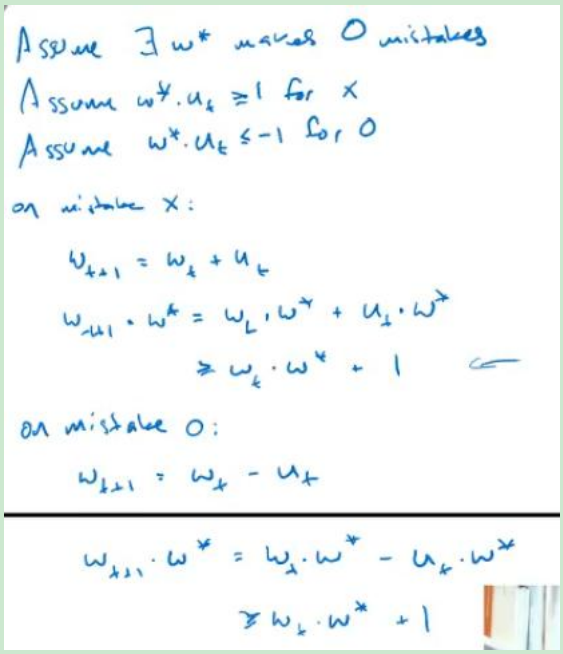
in the second. (To see why, expand out the dot product on the LHS and use $u_t \cdot u_t > 0$.) So, we are more likely to predict the correct label if we see the same example again.

make sence：因为对于正样本判断为负，add example　Wt+1 更大，Wt+1*ut 得到更大的值，**使得分类器 tend to predict 大于 0**

proof it works：Mistake bound

Mistake bound：证明算法在找到最佳参数之前，**最多犯多少次错误(错误次数有上界，就代表参数存在)**

Ut 是 t example
Wt 是 t 时刻的参数



假设存在 w* make 0 mistakes
我们知道 w* ut 对于+样本，一定大于 0，对于-样本，一定小于 0，
又因为 w*可能有很多值如 2+2y=0，2x+4y=0 都是 w*，因此我们 scale w*使得
对于+样本，w* ut>=1 对于-样本，w* ut<=-1

on mistake positive example：
Wt+1=Wt+ut 两边同时乘以 w*，得到如上结论
on mistake negative example：
两种情况下结论一样:都是>=Wt W*+1

因此，我们 deduce：
After M mistakes:
Wt W*>=M
（下面内容继续上面，只是插入几个新的 assumption！）

4. Mistake bound

The perceptron algorithm satisfies many nice properties. Here we'll prove a simple one, called a *mistake bound*: if there exists an optimal parameter vector w^* that can classify all of our examples correctly, then the perceptron algorithm will make at most a small number of mistakes before discovering an optimal parameter vector.

In more detail, suppose that $w^* \cdot u_t > \epsilon$ for all positive examples, and $w^* \cdot u_t < -\epsilon$ for all negative ones. Also assume that our examples are bounded: there is a constant U such that $\|u_t\| \leq U$ for all t.

Then, the perceptron algorithm will make at most

$$\frac{U^2 \|w^*\|^2}{\epsilon^2}$$

mistakes in total. For example, if our examples have norm at most 2, if our optimal parameter vector has norm $\|w^*\| = 3$, and if $\epsilon = \frac{1}{2}$, then the number of mistakes M satisfies

$$M \leq 144 .$$

假设 1 一步系统就是之前的+-1

假设 2 example 不是无限多，是有界的，也就是它的模有上界！！

如果证明 algorithm 最多会犯多少**错**！**有个上界，也就是说明算法有效最终收敛！！**

Proof Part1：

1 先 show $w_t \cdot w^*$ 有 lower bound

u w*>= 一步系统

5. Proof of mistake bound, part I

First we show a lower bound on $w_t \cdot w^*$.

After a mistake on a positive example, we have

$$\begin{aligned} w_{t+1} \cdot w^* &= w_t \cdot w^* + u_t \cdot w^* \\ &\geq w_t \cdot w^* + \epsilon \end{aligned}$$

since, by assumption, $u_t \cdot w^* \geq \epsilon$. Similarly, on a negative example, we have

$$\begin{aligned} w_{t+1} \cdot w^* &= w_t \cdot w^* - u_t \cdot w^* \\ &\geq w_t \cdot w^* + \epsilon \end{aligned}$$

since, by assumption, $u_t \cdot w^* \leq -\epsilon$. So, after M mistakes, we have

$$w_t \cdot w^* \geq \epsilon M$$

by induction: the LHS starts at 0, and increases by at least ϵ with each mistake.

又因为 $w_t \cdot w^*$ start from 0，然后又每次 make mistake 就 add 一个 ε，M 次后得到 εM

Proof Part2：

show upper bound on $\| w^t \|$

6. Proof, part II

Next we show an upper bound on $\|w^t\|$. After a mistake on a positive example, we have

$$\begin{aligned} w_{t+1} \cdot w_{t+1} &= w_t \cdot w_t + 2w_t \cdot u_t + u_t \cdot u_t \\ &\leq w_t \cdot w_t + 0 + U^2 \end{aligned}$$

To see why, note that $w_t \cdot u_t \leq 0$, since we (mistakenly) classified this example as negative. And, $u_t \cdot u_t = \|u_t\|^2 \leq U^2$ by assumption.

Similarly, after a mistake on a negative example, we have

$$\begin{aligned} w_{t+1} \cdot w_{t+1} &= w_t \cdot w_t - 2w_t \cdot u_t + u_t \cdot u_t \\ &\leq w_t \cdot w_t + 0 + U^2 \end{aligned}$$

In this case, $w_t \cdot u_t \geq 0$, since we (mistakenly) classified this example as positive.

So, after M mistakes, we have

$$w_t \cdot w_t \leq MU^2$$

since $w_t \cdot w_t$ starts at zero, doesn't change unless we make a mistake, and increases by at most U^2 on each mistake. Rewriting, we have

$$\|w_t\| \leq U\sqrt{M}$$

for all t.

对于正样本犯错：

$w_{t+1} = w_t + u_t$ 因此 $w_{t+1}w_{t+1}$ 如上

$u_t u_t$ 有 **upper bound** U^2，因此小于等于

正样本犯错 $w_t u_t < 0$，我们用 0 作为其 **upper bound！**

同样对于 negative example，公式结果一样！！

因为 $w_t \cdot w_t$ start from 0,然后又每次 make mistake 就 add U^2，M 次后得到 MU^2

得到 $\|w^t\|$ 的 upper bound

Proof Part3：

7. Proof, part III

From part I we have

$$M \leq \frac{w_t \cdot w^*}{\epsilon}$$

By Hölder's inequality, we therefore have

$$M \leq \frac{\|w_t\|\|w^*\|}{\epsilon}$$

Substituting in the conclusion of part II, we get

$$M \leq \frac{U\sqrt{M}\|w^*\|}{\epsilon}$$

and rearranging we get

$$M \leq \frac{U^2\|w^*\|^2}{\epsilon^2}$$

as desired.

dot product <= norm product

最终得到 M 的 upper bound

Other logical system FOL

之前是 propasitional logic system

object (包含 **boolean properties**)
把相同的 property 变成 boolean value 的 fucntion(**predicate**)

2. First-order logic

One useful extension beyond propositional logic is *first-order logic*. Instead of talking just about unstructured propositions (as we have done so far), first-order logic can group together multiple related propositions into an *object*. Each proposition then becomes a Boolean-valued function (called a *predicate* or a *property*) applied to the object.

For example, we might talk about an object named $Socrates$, with Boolean properties such as man, $mortal$, etc.:

$$man(Socrates) = T$$
$$mortal(Socrates) = T$$
$$Greek(Socrates) = T$$
$$Roman(Socrates) = F$$

We can combine these propositions with the usual logical connectives, like this:

$$(man(Socrates) \wedge mortal(Socrates)) \vee Roman(Socrates)$$

A different object — say $Caesar$ — could have the same properties, but with different truth values:

$$Greek(Caesar) = F$$
$$Roman(Caesar) = T$$

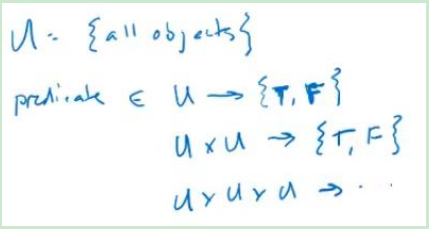
同样的 property，值不同

predicate 断言(程序里的) (一种函数形式)，参数是 object，返回 boolean value

We can also have properties associated with *tuples* of objects instead of just single ones. For example, we could have a predicate $friends$ that operates on pairs of objects, as in

$$friends(Socrates, Plato) = T$$

No matter how many arguments it has, the value of a predicate is either T or F, indicating whether the property holds for the given list of arguments.



函数映射，可以接受多个 object

Function

类似于 predicate，但返回 object

function ∈ $U \to U$
$U \times U \to U$
$U \times U \times U \to U$

3. Functions

FOL also includes *functions* that act on objects or tuples of objects. For example, we could have a function $advisor$ that returns the academic advisor of a student:

$$advisor(Plato) = Socrates$$

The syntax is similar, but don't confuse functions with predicates: a function returns an object like $Socrates$, while a predicate returns a truth value, either T or F. The arguments to either a function or a predicate are always objects — so, it makes sense to nest a function inside a predicate, but not vice versa.

Functions let us write even more complicated expressions like

$$Greek(advisor(Plato))) = T$$

or

$$friends(Socrates, instructor(10\text{-}606))) = F$$

predicate 和 function 混合使用

4. Inference rules

The benefit of introducing objects and functions is that we can reason about many propositions at once. For example, we might say that

$$man(x) \to mortal(x)$$

is true for any object x: this way we can reason simultaneously about the propositions $mortal(Socrates)$, $mortal(Caesar)$, $mortal(Plato)$, ...

To achieve this benefit, first-order logic uses a richer set of inference rules than propositional logic: it upgrades familiar rules such as modus ponens to handle objects, and it adds additional inference rules that are not present in propositional logic. An example of the latter is the *rule of substitution*: roughly, if a statement of FOL is true when applied to some class of objects X, then it is also true when applied to any more specific class of objects $Y \subseteq X$.

同时 reason 多个 object(represent 多个 object)
比 propositional logic 更 powerful

人类是动物，则医生也是动物

5. Reasoning in FOL

Our goal here is not to give a complete introduction to FOL. (In fact we've skipped lots of interesting stuff, including variables, quantifiers, unification, ...) Instead our goal is to point out two things:

- There are many different reasoning systems, each of which can express different facts and different kinds of reasoning. For example, FOL can reason about facts that propositional logic can't even express.
- Even so, reasoning in FOL follows the same basic structure as reasoning in propositional logic: we still express a proof as a sequence of statements, with each statement justified either as an assumption, or as a consequence of previous statements via an inference rule.

6. Logic with theories

Another common extension of the logics we've discussed so far is to add a *theory* — that is, a list of new objects, predicates, functions, and facts, together with inference rules for reasoning about them. For example, we might design a theory of real numbers, which would introduce objects like 1.4, π, and -27, along with functions like $+$ and $\cdot$, and facts like $0 \in \mathbb{R}$. To work with statements in this theory of reals, we might use inference rules like linear combination of equations: e.g., if we know that $x, y \in \mathbb{R}$ and

$$\begin{aligned} 3x + 2y &= 5 \\ x + y &= 2 \end{aligned}$$

then we could conclude

$$x = 1$$

by subtracting twice the second equation from the first equation.

The new facts in a theory are often called *axioms*; we can think of them as inference rules with no premises. That is, they allow us to conclude a statement like $0 \in \mathbb{R}$ no matter what other statements are present in our current context.

In one sense, a lot of the rest of this course will be teaching you about useful theories: e.g., theories for working with vectors and linear operators, or theories for working with functions and differentiation.

在证明过程中可以 add **theory，**不需要 **premise（前提假设）不需要证明**
上面，**theory** 就 是 **linear combination**

probability

每次考虑概率时，都要思考实验是什么，sample space ，population 是什么!!

首先区分试验和事件的关系！！
每次试验的结果是事件！！且是原子事件
所有可能出现试验的结果集合是 sample space S 样本空间

如抛硬币：
有不同种类的实验，不同实验对应不同的 sample space

假设一次实验是抛两次硬币，sample space：{-+，++，--，+-}(是有先后顺序)
则就有了 4 种原子事件(不是-和+才是原子事件！！！要根据实验来)
实验决定事件！！

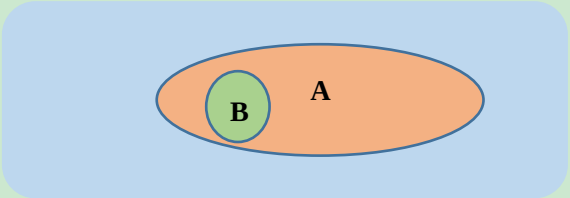
Sample space：定义是假如进行一次实验，能得到所有可能的结果(atomic event 的集合)
但要明确一点的是 sample space **不是实验对象 population**
有时候会对同一群体 population 做不同实验，得到的 sample sapce 不同！！！(同一 population 的不同划分)
population 有些实验是一次实验会有多个观测结果，也是一个事件
population 好像不再是一种实验，而是一种划分？？可以理解为每个人都是一个原子事件？
如 测量一个行人的身高，肤色，**population 就是所有可能的行人**
可以看做两次实验，sample space 不同，但作用的对象 population 一样，此时两种实验的 atomic even 一定有**交叉**，且一定是，这个交叉部分之和一定是 **population**，就可以用 sum rule

当然，**如果不同实验，population 不同时也可以**，其他实验的 event 和某一**特定实验**的所有 event 的交集刚好是这一特定实验的 population，也可以用 sum rule
用 vn 图，来表示 population 会比较清晰！！

因此上面的 VN 图既可以表示为原子事件的集合，又可以表示 population

我理解的是有两类实验，
1 一类是只有一种形式，但可以重复 如抛硬币，这种实验只有固定的 sample space
2 另一类是有很多相近似的实验，如对人测试性别，肤色，不同的实验对应不同的 sample space，但有一个 original space，每个人都是一个 atomic 事件，上面的实验只是对其进行划分而已，**相当于是统一个 sample space 多个实验**

Sum rule :
B 事件可以划分为多个互斥的事件，且多个互斥事件和 A 的交集刚好是 A 的 **population**
上面是将 B 划分为 atomic 事件，这里 B 可以划分为多个复合事件，但互斥就好！！



split A = A,B+A,~B

$$P(A) = P(A, B) + P(A, \neg B)$$

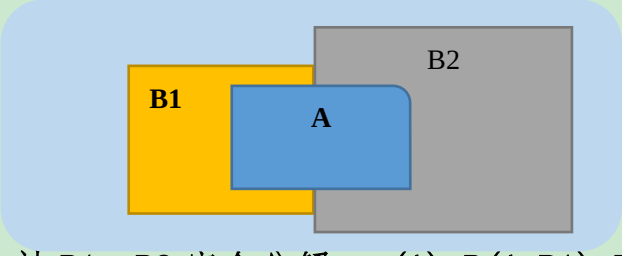
split A into 2 mutually exculsive events

The sum rule also holds if we split into three or more mutually exclusive and exhaustive events: if $B_i \cap B_j = \emptyset$ and $\sum_i P(B_i) = 1$ then

$$P(A) = \sum_i P(A, B_i)\,.$$

当然 B 也不一定包含在 A 里，可以想象多个 B 跟 A 相交的部分刚好组成 A，这样也符合上

面的情况!!!!! 所以事件 B 不一定在 A 里，只是与 A 的交际，将 A 进行了分解！！！



A 被 B1，B2 完全分解，p(A)=P(A,B1)+P(A,B2)

	brown	blue	green
5′3″	.05	.02	.01
5′4″	.06	.025	.02
⋮		⋮	

同一样本空间定义不同的实验和事件，某一个事件如身高=5.3 一定会被其他实验的事件分割并且交集一定等于身高=5.3 的 sample 数量，因为身高=5,3 的人们一定至少属于三种颜色的一类 p(h=5.3)=p(h=5.3 ,green)+ p(h=5.3 ,brown)+ p(h=5.3 ,blue)
因此 sum rule 就可以这样来记忆：同一个 sample space 的不同实验，如果是多个实验：就记住联合分布：

P(A)=P(A, B1,C1)+p(A,B1,C2)+..

其实也就是边缘分布的来源

Atomic event

一次实验中不能再分的事件！
S 是 set of atomic event **sample space(universe)**
可以给每个事件赋予概率！！

1. Atomic events

Many events can't be predicted with total certainty. We use the idea of probability to say how likely they are to happen. Probability builds on top of set theory and logic, which we covered in earlier sets of lecture notes.

We assign probabilities to *simple* or *atomic* events: we think of a *universe* or *sample space* S that is a set of possible atomic events. Often the atomic events will be objects that we can interpret and assign properties to: e.g., the number of spots on a die that we roll, or the card that we draw from a deck.

$P(a) \geq 0$ is the probability that an event $a \in S$ occurs, and $P(\neg a) = 1 - P(a)$ is the probability that it does not. (Notation: we can also write $-a$ or $\bar{a}$ instead of $\neg a$.)

Probability is always between 0 and 1: impossible = 0, certain = 1.

复合事件是 subset of S：
如：第一次出现正面的事件：{+- ， ++}，描述或的关系！
复合事件的概率就是 subset 站 set 的比例！！！ 也就是 atomic event 的概率

计算！

p(+- U ++)=p（+ - ）+p（++）

p(compound event)是各个 p(atomic event)的合（条件是 U 的两端都是原子事件）

如果是复合事件(可能会相交，有重复的原子事件)p(A U B)=P(A)+P(B)-P(A,B)

coumpound event 和 joint event

coumpound event 是或 $A \cup B$ **joint event 是交** $A \cap B$

2. Compound events

Given the probabilities of atomic events, we can compute the probability of a "compound event": a subset of S. The probability of a compound event E is the sum of the probabilities of the atomic events that make up E:

$$P(E) = \sum_{a \in E} P(a)$$

The sum of all atomic events' probabilities must be 1:

$$P(S) = 1 .$$

全部原子事件的概率和=1

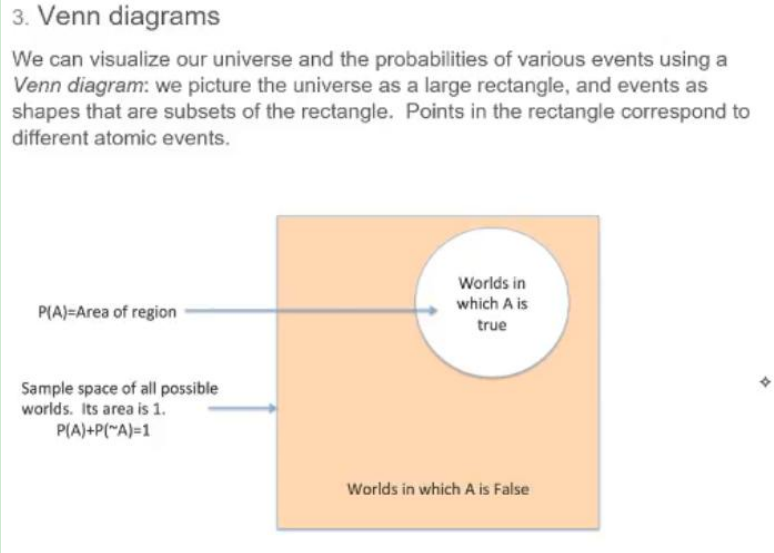
We'll sometimes write events using logical expressions: e.g., $P(\text{die roll} \in \{2, 4, 6\})$ or $P(\text{suit} = \spadesuit)$. We interpret such events using set-builder notation: e.g., the latter event is shorthand for $P(\{x \in S \mid \text{suit}(x) = \spadesuit\})$. The variable x ranges over the atomic events in our universe: here, cards that we might draw from a deck.

有了 conpound event 就有意思了，就会存在交集 A and B,也表示为 A,B,p(A,B)其实就是联合概率！！！

When we're using logic to describe events, there's a straightforward mapping between logical connectives and set connectives: e.g., $A \wedge B$ corresponds to $A \cap B$, $A \vee B$ corresponds to $A \cup B$, and $\neg A$ corresponds to $S \setminus A$.

There's one more notation that you'll commonly see: we often write $P(A, B)$ as a synonym for $P(A \cap B)$ or $P(A \wedge B)$. That is, just like in set-builder notation, we read a comma as "and".

一些常用表示，P（A,B）会省略符号 n

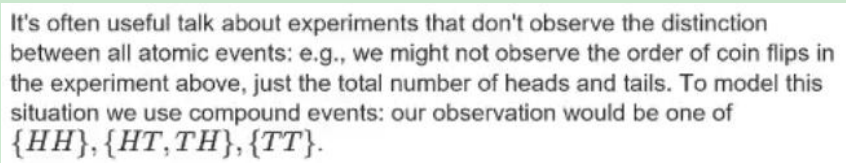


实验：

1 有些实验是**有多个步骤，有先后顺序的值**(多次掷骰子)

全部实验结果=atomic event{HH,TT.HT.TH}

2 也可以忽略其顺序，只看结果，这时候需要定义 compound events{HT,TH}



实验本身不变，只是我们**从新定义了实验的 atomic event，现在是三个**

3 有些实验是**同时观测多个值**(行人调查，身高，肤色)**不存在顺序**，(一次实验多个观测值)

本质上是**做了 2 次独立的实验**，但可以一起**看做一个实验：此时的样本空间是所有可能结果的组合！！！**

	黑	白	黄
1.7			
1.6			
1.4			

每个记录一个 atomic event，一个实验结果

也可以看做 2 次独立的实验

但每个实验结果的 atomic event 有交叉：**白人也有身高 1.7 的**

也可以看做两次独立实验，有交叉？？question

union

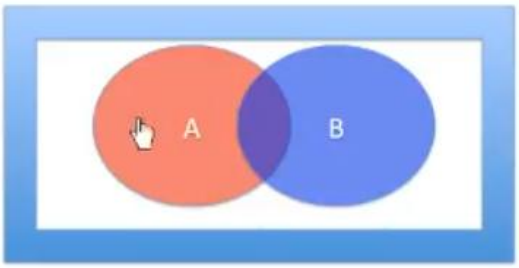
disjoint union
互斥 union

One particularly useful rule is *disjoint union*: if A and B are disjoint events (i.e., $A \cap B = \emptyset$), then

$$P(A \cup B) = P(A) + P(B).$$

non-disjoint union

Here's an example where events A and B can occur at the same time — that is, they are not disjoint, and so $P(A \cap B)$ is not 0.



$$P(A \cup B) = P(A \setminus B) + P(A \cap B) + P(B \setminus A)$$

p(A\B) A-B

$$P(A) = P(A \setminus B) + P(A \cap B)$$

$$P(B) = P(B \setminus A) + P(A \cap B)$$

So,

$$P(A) + P(B) = P(A \setminus B) + P(B \setminus A) + 2P(A \cap B).$$

Moving one copy of $P(A \cup B)$ to the left and combining with the previous result, we get

$$P(A) + P(B) - P(A \cap B) = P(A \cup B).$$

We can also rearrange to get the equally-useful formula

$$P(A) + P(B) - P(A \cup B) = P(A \cap B).$$

Logic rule：

set 的 logic 运算 全部 apply to p(set)

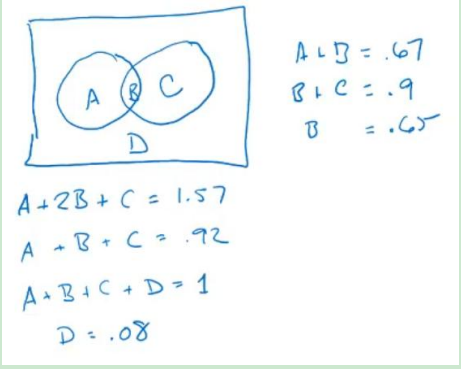
$$P(\neg(A \wedge B)) = P(\neg A \vee \neg B)$$

10. Exercise

Consider the following experiment: you stand on a street corner in Pittsburgh and survey people. You ask two questions: "Do you like computer science?" and "Do you like the Steelers?" You find that 67% of passers by respond "yes" to the first question, and 90% of passers by respond "yes" to the second question. Furthermore, 65% of passers by respond "yes" to both questions.

What is the probability of finding a person who responds "no" to both questions?

2 event,并且有相交



4 个 parts 求 D，方程组解

也可以用概率的方式去解

Random variables：joint marginal and conditional distribution

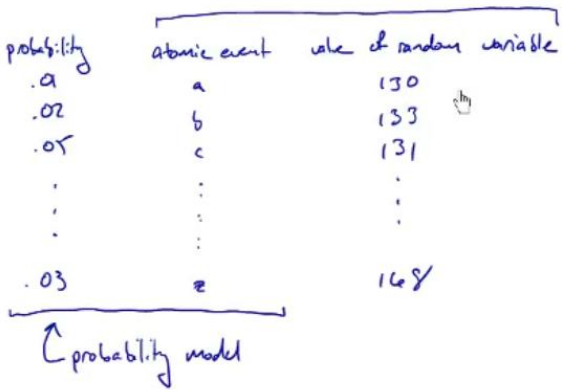
Random variables is a function map a atomic event to a real number，对应的是 atomic event

同一个 sample 的不同 Random variables，其实就是不同的实验(实验结果不同)

random variable 和 probability model 往往在一起：

2. Random variables: the picture

Here's an example of a probability model and a corresponding random variable:



3. Random variables and events

An important use of random variables is to define new events: for example, the event that the height of the next participant is greater than 5'3", or that the next five cards that I draw form a straight flush.

random variable 可以用来定义新的复合事件

很重要的一点；不要忘记我们的 atomic event 和样本空间！

Sometimes we go so far as to forget about the underlying sample space, and talk only about random variables: e.g., we talk about probabilities related to height or eye color without mentioning that the underlying atomic events are survey participants. It's important to remember that there still is an underlying sample space, which we use to define the relevant probability distribution.

joint distribution

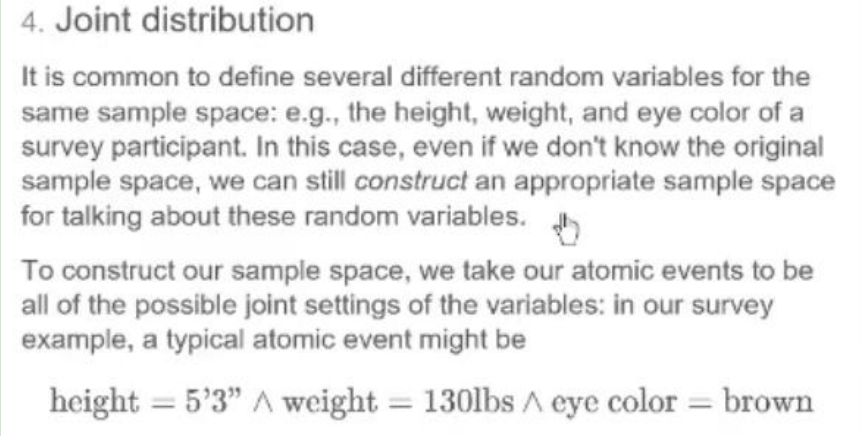
reconstruct the sample space！！

同一个样本空间，定义多个 random variables！！多个变量的概率表就是 joint distribution!

相当于同一个 population，多个独立实验看作一个实验！！一个实验结果多个值

得到的每个事件都是 joint event！！(同时发生)不是 compound

有时候会使得我们忘记我们的 sample space



4. Joint distribution

It is common to define several different random variables for the same sample space: e.g., the height, weight, and eye color of a survey participant. In this case, even if we don't know the original sample space, we can still *construct* an appropriate sample space for talking about these random variables.

To construct our sample space, we take our atomic events to be all of the possible joint settings of the variables: in our survey example, a typical atomic event might be

$$\text{height} = 5'3'' \wedge \text{weight} = 130\text{lbs} \wedge \text{eye color} = \text{brown}$$

random variable 其实就是在原有的 sample sapce 里定义新的复合事件

例如我们调查行人的眼睛，身高，体重**随机变量**

用 3 个随机变量描述同一个 sample space，也就是将三个实验看着一个实验！！！

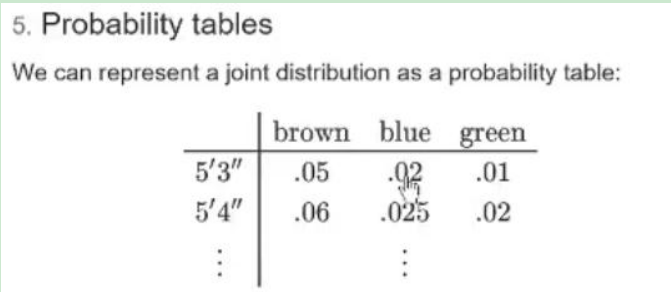
sample space 每次实验的全部可能结果！！

list 所以可能的 random variables joint 列表信息(all possible joint set)

其中，每一条记录：身高=19 and 体重=3 and 眼睛=blue

就是一个 atomic event

每个 atomic event 一个概率，这就是多变量联合分布！！



5. Probability tables

We can represent a joint distribution as a probability table:

	brown	blue	green
5′3″	.05	.02	.01
5′4″	.06	.025	.02
⋮		⋮	

表每个值都是一个 atomic event 的概率！！ table 所有值和为 1

probability table 可能是个 multiple array（tensor）

dimension called modes

probability table and compound event and sum rule

p(h=5.4)是 compound event: (h=5.4,bl),(h=5.4,g),(h=5.4,br)

using the sum rule:

p(h=5.4) equals to sum over all the row(all eye colors)

p(h=5.4)=p(h=5.4,bl)+p(h=5.4,g)+p(h=5.4,br)

用 sum rule 来证明：

此时，把**身高和眼睛**看做同一 population 两次独立实验，得到两个独立 Sample space：

眼睛的 atomic event 跟**身高** event 有交集，

1 交集组成了身高的 population(行人），

2 且颜色 event 都是 exclusive event，就可以用 sum rule

对于多个随机变量

P(A)=P(A, B1,C1)+p(A,B1,C2)+..

B 和 C 的全部组合. BC 的组合也是 atomic event 互斥，且和 A 有交叉，组成 A 全部 population

以上是一种证明，现实用的时候只记住上层结论就好了！！

7. Alternate formatting for probability tables

To write out a multidimensional probability table on a 2d piece of paper, we can list out all of its entries one after the other, like this:

height	weight	eye color	P
$5'3''$	130	brown	.03
$5'3''$	130	blue	.02
$5'3''$	130	green	.01
$5'3''$	131	brown	.04
$5'3''$	131	blue	.03
$5'3''$	131	green	.02
	⋮		⋮

This version is also called a probability table, even though its formatting is different.

3 modes(random variables) probalibity table 来展示 joint distribution

Marginal distribution

marginal of the joint distribution, 只计算一个 random variable 的概率值,只需将概率表里全部的其他 random variables 的所有可能的概率值相加(sum rule)

Conditional distribution(conditional probability 计算)

If we condition on an event A, the distribution of X can change.

需要重新计算

Given a joint distribution, we can also ask what happens if we fix the value of one or more of the variables. The result is called a *conditional* distribution; we say that we are *conditioning on* or *observing* the random variables that we are fixing. For example, we could ask: suppose that we know that the height of the person is 5'3"; then what is the distribution over eye color?

是在 joint distribute 的基础上计算，固定一个 random variable 的值(observation)，求另一个 random variable 的概率

需要对 joint distribute table 进行变化(**概率不再是之前的值**)，但概率值都不同了！！

计算：

	brown	blue	green
$5'3''$	.05	.02	.01
$5'4''$	.06	.025	.02
⋮		⋮	

Again it is straightforward to find the probability table for a conditional distribution: we throw away all the parts of the joint table that are inconsistent with our observations, then normalize (divide each remaining element by the sum). For example, when we condition on height being 5'3", we get:

brown	blue	green
.625	.25	.125

Here, for example, $0.625 = .05/(.05 + .02 + .01)$.

1 将 p-table 的不符合 observation 的值全部去掉

2 normalize sum to 1

因为身高固定情况下，三种颜色的眼睛的概率和一定为 1！！

每个值除以和，可以归一化！！！

其实就是条件概率定义：$p(brown|5.3) = \frac{p(brown,5.3)}{p(5.3)}$

其实也是对条件概率的另一种解释

p(A|B)还是 2 维的 table

notation:

If we write $P(X, Y \mid Z, W)$, we mean the conditional distribution of the random variables X and Y, given the values of the random variables Z and W. As usual, we can fill in specific values of the variables: for example, if X and Y are Boolean while Z and W are integers, we could ask for $P(X = T, Y = F \mid Z = 3, W = 1)$, the probability that $X = T$ and $Y = F$ given that $Z = 3$ and $W = 1$.

We can do the same thing for conditionals and marginals: e.g., start from $P(X, Y \mid Z)$, which is a table for the conditional distribution of X and Y for all possible values of Z. Then $P(X, Y \mid Z = 1)$ is a table for the conditional distribution of X and Y, given that $Z = 1$; this is a slice of our larger conditional probability table, and sums to 1 over X and Y. For another example, $P(X, Y = 1 \mid Z)$ is also a slice of our larger table; but, this slice does not sum to 1, since it doesn't correspond to a single probability distribution.

P(X, Y|Z)是已经 normal 好的三维 table，每一个 Z，sum P(XY)都是 1

P(X,Y|Z=1)是上面表的一个 slice，就成了 2 维了

$$P(W \mid O, R) = \left[\begin{array}{cc} .9 & .05 \\ .1 & .05 \\ \hline .1 & .95 \\ .9 & .95 \end{array}\right].$$

p(W|O,R)是 2*2*2 三维 table tensor 有 2 layer，每个 layer 2*2

上下代表 W，行 O，列 R

O,R 选定后，和是 1，全部 table 的合一定不为 1

If we don't fill in specific values for the random variables, then we mean a (conditional) probability table: $P(X, Y \mid Z, W)$ is a four-dimensional table, indexed by the values of X, Y, Z, W. We can also fill in some random variables but not others: e.g., $P(X, Y \mid Z = 3, W = 1)$ is a two-dimensional table, indexed by X, Y.

p(XY|ZW)，是一个 4 random variables 的 table 4 维

p(XY|Z=3,W=1)，是一个 2 random variables 的 conditional ditribute table 2 维

A conditional probability distribution is a distribution over the variables on the left of the conditioning bar (here X and Y). So, for example, $\sum_X \sum_Y P(X, Y \mid Z = 3, W = 1)$ must always be 1. A conditional probability distribution is *not* a distribution over

注意：conditional 分布是条件左边变量的 distribute！

因此对左边变量联合概率相加才为 1，条件是只有 Z，W 先取一次 slice，sum XY 才等于 1 ！！！！

因此，$P(X,Y|Z=3,W=1)$，是 XY 的 p-table，全部相加值为 1

对于 $P(X,Y|Z,W)$，是 4 个变量的 p-table，如果 sum XYZW 一定不等于 1，

be 1. A conditional probability distribution is *not* a distribution over the variables on the right of the bar. For example, $\sum_Z \sum_W P(X=T,Y=F \mid Z,W)$ will typically not be 1.

$P(\text{Gender} = \text{male}) = 0.3313 + 0.0972 + 0.1341 + 0.1059 =$

Gender	Hours Worked	Wealth	Probability
female	< 40.5	poor	0.2531
female	< 40.5	rich	0.0246
female	>= 40.5	poor	0.0422
female	>= 40.5	rich	0.0116
male	< 40.5	poor	0.3313
male	< 40.5	rich	0.0972
male	>= 40.5	poor	0.1341
male	>= 40.5	rich	0.1059

marginal p

$P(\text{Wealth} = \text{rich} \wedge \text{Gender} = \text{female}) = 0.0246 + 0.0116 =$

Gender	Hours Worked	Wealth	Probability
female	< 40.5	poor	0.2531
female	< 40.5	rich	0.0246
female	>= 40.5	poor	0.0422
female	>= 40.5	rich	0.0116
male	< 40.5	poor	0.3313
male	< 40.5	rich	0.0972
male	>= 40.5	poor	0.1341
male	>= 40.5	rich	0.1059

joint event p

Statictics feature

random variable 的 statictics feature 是总体的，是未知的，我们只能估计

以下都是假设每个 random variable 已经有了观测值，来计算其统计 feature(是抽样对总体的估计！！)

Mean and Variance

都是针对一个 list 数据，得到一个统计量

mean(expection)

1 ramndom variable 的 feature

$$E(X) = \sum_x x \cdot P(X = x).$$

另一种 notion $E(X) = \bar{X}.$

2 function of ramndom variable

X 是 ramndom variable，f(X)也是 ramndom variable

同样可以计算：$E(f(X)) = \sum_x f(X) \cdot P(X = x)$ ，概率是自变量的概率分布！

3 linearity of expection

$$E(aX + b) = a\,E(X) + b$$

直接穿透

linearity of expectation 证明

对于随机向量，也适用：假设 A 是系数矩阵

$$E[AXX^TA^T] = AE[XX^T]A^T$$

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix} \qquad X \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$AX = \begin{pmatrix} 1x_1 + 2x_2 \\ 2x_1 + 0x_2 \end{pmatrix}$$

$$EAX = E\begin{pmatrix} 1x_1 + 2x_2 \\ 2x_1 + 0x_2 \end{pmatrix} = \begin{pmatrix} E[1x_1 + 2x_2] \\ E[2x_1 + 0x_2] \end{pmatrix} = \begin{pmatrix} Ex_1 + 2Ex_2 \\ 2Ex_1 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix}\begin{pmatrix} Ex_1 \\ Ex_2 \end{pmatrix} = AEX$$

A 其实是对 X 每个元素做了线性变换，A 可以提出来，做完线性变换后求均值和 先求均值再线性变换 等价！！

Variance

$$\mathrm{Var}(X) = E[(X - \bar{X})^2].$$

其实**方差**也是期望：f(X)的期望，此时 f(X)=$(X - \tilde{X})^2$

随机变量是 与期望的差的平方

The moments of $X - \bar{X}$ are called the *central moments* of X. So, the variance is the second central moment of X.

central moment　　方差是 2th central moment

3th central moment : skew　4 th ?

variance 的单位是**单位的平方**，std 的**单位还是跟 X 一样**

exercise:

5. Exercise: variance of a coin flip

A good example is to compute the variance of a biased coin flip: if we count heads as $X = 1$ and tails as $X = 0$, and if we have probability p of heads, then it's clear that $E(X) = p$. It's somewhat longer but also straightforward to derive

$$\mathrm{Var}(X) = p(1 - p).$$

Standardizing

z-score

$$Z = \frac{X-\mu}{\sigma}$$

$E[X-\mu] = E[X] - \mu = 0$ 使得变量期望为 0

证明 Zscore 后期望 0 和方差 1： linearity

$E[\frac{X-\mu}{\sigma}] = \frac{1}{\sigma}E[X] - \frac{\mu}{\sigma} = 0$ 也是 0

$\text{var}[\frac{X-\mu}{\sigma}] = E[(\frac{X-\mu}{\sigma})^2] = \frac{1}{\sigma^2}E[(X-\mu)^2] = \frac{1}{\sigma^2}\cdot\sigma^2 = 1$

对于多个 random variable，每个变量独立的进行标准化！：

Covariance and correlation

协方差 针对两个 list 数据，得到一个统计量

Suppose we have two random variables, X and Y. The covariance of X and Y is defined as

$$\text{Cov}(X,Y) = E[(X-\bar{X})(Y-\bar{Y})].$$

协方差矩阵计算：

X	Y
1	2
3	6
4	2
5	2

$\Sigma_{11} = (1-3.25,3-3.25,4-3.25,5-3.25)\times(1-3.25,3-3.25,4-3.25,5-3.25)^T = 8.75$

$\Sigma_{12} = (1-3.25,3-3.25,4-3.25,5-3.25)\times(2-3,6-3,2-3,2-3)^T = -1$

$\Sigma_{21} = (2-3,6-3,2-3,2-3)\times(1-3.25,3-3.25,4-3.25,5-3.25)^T = -1$

$\Sigma_{22} = (2-3,6-3,2-3,2-3)\times(2-3,6-3,2-3,2-3)^T = 12$

所以，按照定义，给定的4个二维样本的协方差矩阵为：

$$\Sigma = \begin{pmatrix} 8.75 & -1 \\ -1 & 12 \end{pmatrix}$$

计算协方差不需要排序，就是要看 XY 有没有线性相关关系，主对角线是方差，其他是协方

差，Cov(XY)=Cov(YX)因为都是两个变量的相关关系，肯定一样！！

Covariance 有正负，代表 X 和 Y 的趋势关系

The covariance is a measure of how much big values of X tend to predict big values of Y.

一般是对同一个总体，同时进行两次独立实验，看这两次实验结果有没有相关性

计算：只能先用抽样估计，主要每一行 X，Y 都是一次实验(同时进行)

X	Y	X_i-X_{avg}	Y_i-Y_{avg}	Product
1	8	-3.89	2.56	-9.96
3	6	-1.89	0.56	-1.06
2	9	-2.89	3.56	-10.29
5	4	0.11	-1.44	-0.16
8	3	3.11	-2.44	-7.59
7	3	2.11	-2.44	-5.15
12	2	7.11	-3.44	-24.46
2	7	-2.89	1.56	-4.51
4	7	-0.89	1.56	-1.39

$$\text{Covariance} = \frac{\sum(x_i-x_{avg})(y_i-y_{avg})}{n-1}$$

是一个值

相关系数

$$\rho = \frac{Cov(X,Y)}{\sqrt{D(X)}*\sqrt{D(Y)}} = \frac{E[(X-\bar{X})*(Y-\bar{Y})]}{\sqrt{D(X)}*\sqrt{D(Y)}} = E[\frac{(X-\bar{X})}{\sqrt{D(X)}}*\frac{(Y-\bar{Y})}{\sqrt{D(Y)}}]$$

利用 linearity

其实就是：在计算 corivance 之前先标准化 X Y，此时得到的 corivance 就是 correlation！

If we standardize X and Y before computing their covariance, the result is called the *correlation* between X and Y. In equations:

$$\text{Corr}(X,Y) = E\left[\frac{X-\bar{X}}{\sigma_X}\frac{Y-\bar{Y}}{\sigma_Y}\right]$$

correlation 本质就是协方差！！！

Correlation is always in the range $[-1, 1]$: a correlation of 1 means that Y is a linear function of X with positive slope, while a correlation of –1 means that Y is a linear function of X with negative slope.

反应的是 linear 关系

Covariance, correlation, and independence are all distinct concepts: if two random variables are independent, then they have zero covariance and zero correlation, but the reverse is not true.

Exercise: give an example of two random variables that are dependent but uncorrelated.

Conditional mean and var

If we condition on an event A, the distribution of X can change. In this new distribution, X will have a mean and variance; these are called the *conditional* mean and variance of X given A, written $E(X \mid A)$ and $\mathrm{Var}(X \mid A)$.

有了 condition X 会有新的 mean 和 var

因为 X 的范围变小了

以上是 condition 是个确定的 event！、

下面 condition 是 random variables：

Similarly, we can condition on a random variable Y, and compute the mean and variance of X as a function of the value of Y.

Y 取不同的值，得到不同的 distribution p(X|Y=i)

每个 distribution 可以计算 X 的 mean 和 variance

不同的 Y，得到不同的 conditional mean 和 var of p(X|Y=i)

E(X|Y)

Vector-valued random variable（随机向量）

random variable value 是 vector(多个值)

相当于是把之前的**多次试验合并成一个试验**，或者一次实验多个结果：身高，肤色..

mean vector

If $X \in \mathbb{R}^n$ is a vector-valued random variable, then we define its expected value *componentwise*: that is,

$$E(X) = \begin{pmatrix} E(X_1) \\ E(X_2) \\ \vdots \\ E(X_n) \end{pmatrix} \in \mathbb{R}^n.$$

X 是 Rn，EX 也是 Rn　维度不变，期望的维度和随机变量的维度一样

每个元素单独求均值

也符合常理(身高，肤色单独求)

Variance/Covariance matrix

$$\operatorname{Var}(X) = E\left[(X - \bar{X})(X - \bar{X})^T\right] \in \mathbb{R}^{n \times n}.$$

The diagonal elements of $\operatorname{Var}(X)$ are the variances of the individual components of X; the off-diagonal elements are the covariances of pairs of elements of X.

主对角线是方差，非对角线是协方差

对于 1 维随机变量方差;E[(X-u)^2]　(X-u)^2 也是一维

$X - \tilde{X}$ 是向量

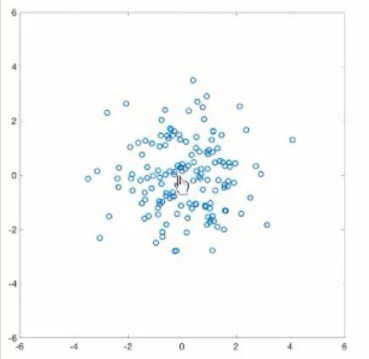
$(X - \tilde{X})(X - \tilde{X})^T$ 是向量的外积得矩阵(只是一个组合矩阵，也就是假设 X 三个取值 ABC,得到 AB　AC CC..所有组合的期望,以矩阵形式排列)

矩阵 n*n 个元素(组合),则期望：E[n*n] 等于 每个组合的期望 ，最终得到 n*n 的矩阵

Interpreting variance matrix

variance matrix 是用于描述多元变量的特点，1 各维度数据的方差 2 各维度数据之间的关系

假设 X 2 维，两个元素　画出 X

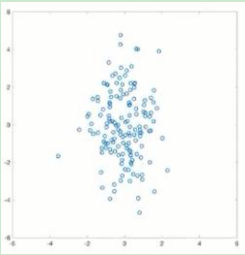


center(0,0),round shape,每个变量方向

In this example, the variance matrix is

$$\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

diagonal variance matrix，且对角元素相等，两个元素的方差相等



By contrast, for the variance matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$$

其中一个元素方差大，波动范围大

以上都是平行于坐标轴的，只有横线或水平线，一个变量不随另一个变量变！！协方差为 0

以上都是 Var(X)是特殊的矩阵，协方差都为 0

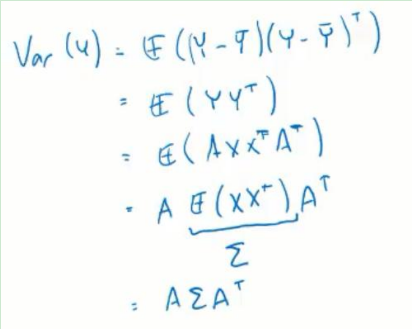
如何是一般的矩阵如何解读？ 下面

transform rule: make var simple to interpret

1 linear tranformation rule

X 的 var 是∑，Y=AX 假设 EX 的是 0 向量，$EY = E[AX] = AEX = 0$

$\text{var}(Y) = A\Sigma A^T$ 方差的性质证明：



linearity

2 对协方差矩阵 SVD 分解

假如

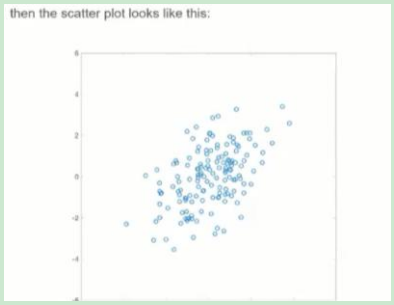
12. Interpreting variances: general matrices

To interpret a general variance matrix, it helps to have a factorization. The most useful is the so-called *singular value decomposition* (which we'll see more of later). In this case, the SVD is

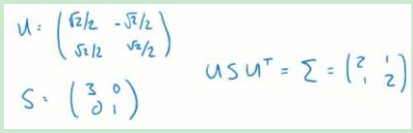
$$\Sigma = USU^T$$

where the matrix U is orthonormal and the matrix S is diagonal. To visualize the variance Σ we can then use the transform trick above. Example: when

$$\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$



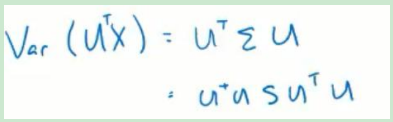
factorize the var Sigma



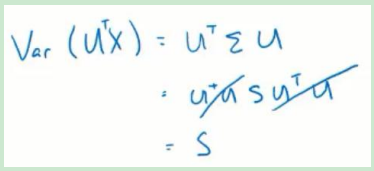
$\Sigma = Var(X)$

transform rule + SVD

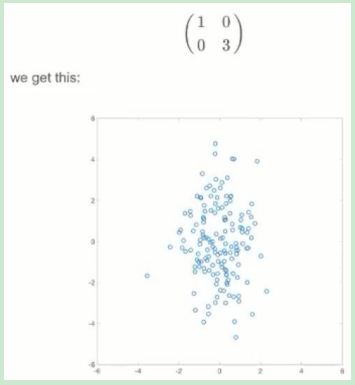
$Var(U^T X)$



又正交矩阵乘是 I



而我们已经知道$U^T X$ 的协方差矩阵是 S,$U^T X$ 的图像如下：



$U^T X$ 是对上面 X rotate，并没有改变 shape

X 乘以 orthoronal 矩阵，只 rotate

因此我们也知道了 X 的图像，只需要 rotate 下

以上是对一个 X 拥有 genneral matirx(非特殊矩阵)的解读方法，

通过 SVD 分解，将 genneral matirx 转成特殊矩阵的 X 形状(容易直观理解 shape)，反向推出 general matrix 的形状

Table and slicing

p(X,Y,Z)其实就是个三维的 tensor，可以切片！！(fix 其他 mode)

We can also fill in an event for only one or two of our random variables. For example, $P(X, Y = 1, Z)$ denotes a *slice* of our probability table. This is a 2-mode tensor, with one mode for each of X and Z. The (i, k) element of this slice is $P(X = i, Y = 1, Z = k)$.

fill one event 相当于 slice(降了一维) 2-mode tensor

We can do the same thing for conditionals and marginals: e.g., start from $P(X, Y \mid Z)$, which is a table for the conditional distribution of X and Y for all possible values of Z. Then $P(X, Y \mid Z = 1)$ is a table for the conditional distribution of X and Y, given that $Z = 1$; this is a slice of our larger conditional probability table, and sums to 1 over X and Y. For another example, $P(X, Y = 1 \mid Z)$ is also a slice of our larger table; but, this slice does not sum to 1, since it doesn't correspond to a single probability distribution.

P(XY|Z) 3 维 tensor

P(X,Y|Z=1)二维 conditional distribution of(XY)，sum over XY equals 1

p(X,Y=1|Z)也是 slice，不是 single dirtibution(每个 Z 取值对应一个 ditribution！) sum 不为 1

joint distirbution Factored form

如果 feature 太多，tensor 太大

需要用 **factored form** 表示（**分解成乘积**）compacted reprensation

form. The most common one is *factored form*: we write the full joint probability distribution as a product of several terms, with each individual term depending only on a subset of the variables.

将联合分布分解，但会损失精度

For example, suppose we have five binary random variables:

- O: whether our robot is Outside
- M: whether our robot is made of Metal
- R: whether it is Raining
- W: whether our robot is Wet
- U: whether our robot is rUsty

The joint table has 32 entries. But, if we represent it in factored form as

整个表的 **entry 个数**=每个 random variable 个数的**乘积**

将 p(OMRWU)分解成小的 **p-table**

$$P(O)\,P(M)\,P(R)\,P(W \mid O,R)\,P(U \mid W,M)$$

P(o)是 marginal of o

这样每个 table 含有的 entries 个数(参数个数)是和：

then the individual tables have

$$2+2+2+8+8=22$$

entries total.

question?为什么是和？

semantic for expresion involving table

It's worth looking a bit closer at the semantics of the equation

$$P(O,M,R,W,U)=P(O)\,P(M)\,P(R)\,P(W \mid O,R)\,P(U \mid V$$

左边和右边的 table 的 shape 是一样的，维度一样

右边每个 factor 是个小 table

To interpret $P(O)\,P(M)\,P(R)\,P(W \mid O, R)\,P(U \mid W, M)$ (or any other expression involving multiple tables):

- Look for all of the random variables that appear free anywhere in the expression. (Free means not inside an event like $Y = 1$.) The expression as a whole represents a big table, with one mode for each free variable, and one entry for each possible assignment to all of these free variables. In our case, the free variables are O, M, R, W, U, so we have a five-mode table.
- To get a particular entry of the big table — say $O = F, M = F, R = T, W = T, U = F$ — use the corresponding assignment to fill in all the random variables in the expression. Each smaller table now corresponds to a single scalar, such as $P(W = T \mid O = F, R = T)$. Each smaller table might not use all of the random variables in our assignment. That's OK; we just ignore any variables that don't appear in that table.
- Now evaluate the expression according to ordinary scalar arithmetic.

In the first step above, we are looking only for *free* variables: for example, the expression $P(W \mid O = F, R)$ would represent a 2×2 table, since two binary variables (W and R) appear free. The remaining variable, O, is bound to a specific value, F.

Refactoring： the chain rule

目的也是为了避免出现 uncompressed form(joint distri) of 联合概率，**将其分解**

$$P(A, B) = P(A \mid B)\,P(B)$$

可以用 chain rule 进行 refactor，**再分解，再组合**

For example, we can use the chain rule to refactor as follows:

$$\begin{aligned} P(A)\,P(B \mid A)\,P(C \mid B) &= P(A, B)\,P(C \mid B) \\ &= P(A \mid B)\,P(B)\,P(C \mid B) \\ &= P(A \mid B)\,P(B, C) \\ &= P(A \mid B)\,P(B \mid C)\,P(C) \end{aligned}$$

Bayes rule

From the chain rule, we have

$$P(A \mid B)\,P(B) = P(A, B) = P(B \mid A)\,P(A)\,.$$

Dividing through by $P(B)$, we get

$$P(A \mid B) = P(B \mid A)\,P(A) \,/\, P(B)\,.$$

条件反转， 分母只是个常数，用作归一化

general form 左右两边可以全部加个 condition C

distributions, named after the Rev. Thomas Bayes. The most general form of Bayes' rule allows us to condition on side information C throughout:

$$P(A \mid B, C) = P(B \mid A, C)\,P(A, C) \,/\, P(B, C)\,.$$

P(A|B,C)P(B,C)=P(ABC)

P(B|A,C)P(A,C)=P(ABC)

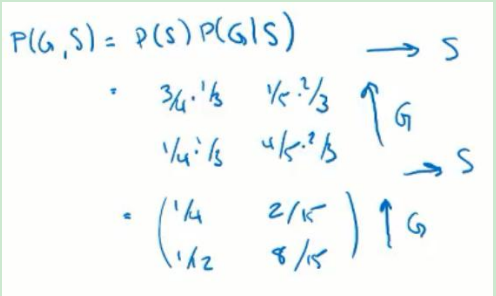
exercise： question 维度约定！！

G：草是干 or 湿 S:浇水

$$P(S) = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \end{bmatrix} \qquad P(G \mid S) = \begin{bmatrix} \frac{3}{4} & \frac{1}{5} \\ \frac{1}{4} & \frac{4}{5} \end{bmatrix}$$

条件概率变成 2*2 了矩阵，S column， G row

用 broadcasting：

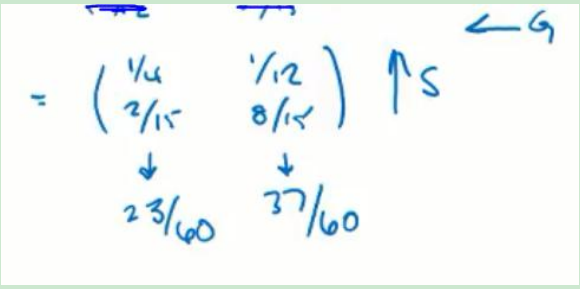


条件概率：就是一维的乘以 2 维的是 broadcasting

转置 S row G column，因为我们约定 p(X,Y)first on the row，Y in column

(Here we are following the convention that the first random variable that appears in an expression corresponds to the rows of a table, while the second corresponds to columns.)

转置一下



得到 p(S,G),比上 p(G)（列的和）=P(S|G)

Independent（marginal independent）

p(X,Y)=P(X)P(Y) ， 也就是 p(X|Y)=p(X)　　条件可以去掉

Two random variables X and Y are said to be *independent* if knowledge of one does not tell us anything about the value of the other — that is, if $P(X \mid Y) = P(X)$ and $P(Y \mid X) = P(Y)$.

互不影响

then X and Y are independent. If no such factorization is possible, they are dependent.

For example, if

$$P(X,Y) = \begin{bmatrix} \frac{1}{12} & \frac{1}{4} \\ \frac{1}{6} & \frac{1}{2} \end{bmatrix}$$

then we can write

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

where

$$P(X) = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \end{bmatrix} \qquad P(Y) = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \end{bmatrix}.$$

So, X and Y are independent.

marginal independent, we can factorize:

P(X) P(Y)=P(XY) 向量的外积

$P(X,Y) = [\text{terms that depend only on } X]\,[\text{terms that depend on}$

Y 如果可以这样分解,XY 一定独立！！判断条件

Conditional independence 条件独立(见专题！)

最常见的例子：

同一个 hidden state s1，观测两次 o1 o2

因为 o1 和 o2 有可能是两个 state 生成，state 之间有关联，o1 o2 一定不独立(不是因为因为由同一个 s1 生成(同一个 state 观测还是独立的))

P(o1,o2)不等于 p(o1)p(o2)！

但是给定条件 s1 下，就独立，每次观测都是独立的(不受上一次观测影响)

给定 s1 条件下独立：

p(o1,o2|s1)=p(o1|s1)p(o2|s1)

也就是 p(o1|s1,o2)= p(o1|s1)

注意：两个事件如果本身独立的，对任何条件都条件独立！

hmm:

p(s2)=p(s2|o1) state 只受前面 state 影响，跟观测独立

则 p(s2|s1,o1)=p(s2|s1)

We can also ask about independence in a conditional distribution: if X and Y are independent after conditioning on an event $Z = z$ (that is, in the distribution $P(X, Y \mid Z = z)$ then we say X and Y are *conditionally independent* given $Z = z$. If X and Y are conditionally independent given $Z = z_1$, $Z = z_2$, and so on for all possible values of the random variable Z, then we say that X and Y are conditionally independent given Z.

given Z=z 和 general given Z

P(X,Y|Z)=P(X|Z)P(Y|Z) XY 在给定 Z 条件下独立

等价于：P(X|Y,Z)=P(X|Z)　　条件可以去掉(就独立)!!!

但 p(X|Y)不等于 p(X)

also，p(X,Y,Z)can fatorize：待看

if XY conditional independent on Z:

$= [\text{terms depending only on } X, Z]\,[\text{terms depending only on } Y, Z]$

then X and Y are conditionally independent given Z. If we can't, then they are conditionally dependent.

能按照上面分解，就一定条件独立！！！

P(XYZ)的 general 分解：

P(XYZ)=P(Y)P(XZ|Y)=P(Y)P(Z|Y)P(X|YZ)

又条件独立：P(X|YZ)=P(X|Z),就得到下面：

For example, given a distribution factored as

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

we can write

$$P(X,Y,Z) = [P(X \mid Z)]\,[P(Z \mid Y)\,P(Y)]$$

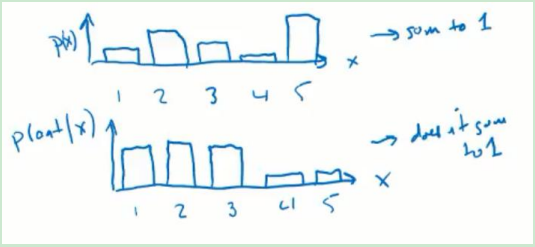
demonstrating that X and Y are conditionally independent given Z.

例如上面左边只有 XZ，右边只有 YZ 可以写成这样，如果可以这样分解,XY 一定条件独立！！判断条件

A slightly subtle point: if we're given only a symbolic representation of a distribution like $P(X,Y,Z) = P(X \mid Z)\,P(Z \mid Y)\,P(Y)$ (rather than its complete probability table), we might not be able to decide whether an independence or conditional independence relationship holds. Typically, from this sort of partial knowledge, we can prove one direction but not the other: if we find a factorization of the symbolic representation we can prove independence (or conditional independence). But if we fail to find a factorization, that doesn't necessarily mean that the variables are dependent (or conditionally dependent): a factorization might be possible if we knew more about the distribution.

充分但不必要

ercercise bayes rule 先验



p(x) sum to 1

p(cat|x)对条件 sum 不为 0

cat 在后，也叫后验概率

cat 在前，先验概率 p(x|cat)

bayes rule 用后验计算先验概率

p(x|cat)=p(cat|x)p(x)/p(cat)

只有 p(cat)未知，但 p(x|cat) sum x=1，可以解出：

右边令 $p(cat|x)p(x)\eta$ choose η，so that $\sum_x p(cat|x)p(x)\eta=1$

1/p(cat)= η 可以解出！！

最终得到 p(x|cat)先验概率分布

就是后验分布相乘，上面两个图的相乘

Bayes filters

X hidden state (GPS) 假设只有这一个状态

Y obs（得出有用信息）

假如一次观测：

1 我们有 P(X) prior ditribution(已知)(before 观测)

2 p(Y|X)是 2 d table 已知，也就是发射概率已知，该 state 得到的不同观测的概率分布已知

p(Y=y|X) 1d table observation likehood(已知 evidence)

3 计算 posterior p(X|Y=y)由观察来推断 state

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

bayes rule p(Y=y) 对观测数据的先验 未知

左右求 sum

$$\underbrace{\sum_x P(x \mid y=y)}_{1} = \sum_x P(y=y \mid x) P(x) / P(y=y)$$

$$= \sum_x P(y=y \mid x) P(x)$$

左边等于 1，p(Y=y)是常量跟 X 无关，两边乘以 p(Y=y)

$$\underbrace{\sum_x P(x \mid y=y)}_{P(y=y)} = \sum_x P(y=y \mid x) P(x) / P(y=y)$$

$$= \sum_x P(y=y \mid x) P(x)$$

左边 $\sum_x P(X \mid Y=y) P(Y=y) = \sum_x P(X, Y=y) = P(Y=y)$,marginal probability

然后右边都已知，可以求出 p(Y=y)，然后带入原公式，得到后验 P(X|Y=y)

如果有新的 evidence(observation)Z Z=z

在已经有了 Y=y 的条件下，又观察到 Z=z(独立的观察)

Y 和 Z 相当于是对同一个 X state 的两次独立观测(如：维度，海拔)

需要 update 后验 p(X|y,z)

再用 bayes rule，条件反转：只反转我们关心的条件 Z=z，不反转已经发生的 Y，Y 已发生作为条件

$$P(X|Z=z,Y=y)=P(Z=z|X,Y=y)P(X|Y=y)/P(Z=z,Y=y)$$

1 P(X|Y=y)是上一次计算的后验，本次的 prior（已知）

2 假设;Z 和 Y conditional independ on X(符合我们 intition)，Z 和 Y 都是 state 的观测值，因此条件独立

$$P(Z=z|X)=P(Z=z|X,Y=y)$$

P(Z=z|X)是 observation likehood(已知 evidence)

这种假设的目的是让 p(Z=z|X)不要那么大，size 不会增加,因为 p(Z|X,Y)会增加一个维度！

$$P(X|Z=z,Y=y)=P(Z=z|X)P(X|Y=y)/P(Z=z,Y=y)$$

两边 sum 为 1，两边同时乘以 P(Z=z,Y=y)是常数不依赖 X

左边 $\sum_x P(X,Z=z,Y=y)=P(Z=z,Y=y)$，右边已知，可以求出 P(Z=z,Y=y),最终得到后验：p(X|Y=y,Z=z)

P(X)最开始有个先验，是我们对 state 的假设，然后随着观测，不断的更新先验概率（上一次计算的后验 P(X|Y=y)成为下一次的先验），得到 X 的分布

every time apply bayes rule,to incoporate evidence,to update the posterior

Prior & Posterior

Every time we apply Bayes rule to incorporate evidence as above, the distribution before the update is called the *prior*, and the distribution after the update is called the *posterior*. Unfortunately, if

bayse rule 其实就是使得先验变后验(计算**后验**)

每用一次 bayes rule，上一次的 posterior 变成下一次的 prior

and so forth. When there's a danger of confusion, it's better to be specific about exactly which distribution we mean: say, $P(X \mid Y = y)$.

前面是两次观测：现在同一个 state 是多次观测

同一个 state X，Multiple observation（多次观测）

观测多次得到 observation sequence

- the initial distribution $P(X)$, and
- the observation likelihoods $P(Y_t \mid X)$; often these are the same for every time step (i.e., do not depend on t).

我们假设 gps 观测器稳定，**因此发射概率 p(Yt|X)不会随着 t 变，**stationary，是固定的分布。

bayes rule:写出后验概率 p(X|Y1=y1 Y2=y2…Yt=yt)

条件反转(只和我们关心的条件(当前 t 时刻的 Y)反转，其他条件不动)，写出 bayes:

记：Ht 是 obs history of t time(not include t):

Y1=y1 Y2=y2…Yt-1=yt-1

$$P(X \mid Y_t = yt, H_t) = P(Y_t = yt \mid X, H_t) * P(X \mid H_t) / Normalization$$

$$P(X \mid Y_t = yt, H_t) = P(Y_t = yt_t \mid X, H_t) * P(X \mid H_t) / P(Y_t = yt \mid H_t)$$

注意最后一个式子是条件概率，因为总体是条件联合概率

又因为多次观测在给定 X 下条件独立：

Note that, as above, we've made the simplifying assumption that $P(Y_t = y_t \mid X, H_t) = P(Y_t = y_t \mid X)$.

$$P(X \mid Y_t = yt, H_t) = P(Y_t = yt_t \mid X) * P(X \mid H_t) / P(Y_t = yt \mid H_t)$$

同样的方法先求分母，再求后验。

ML

X 可以是模型的参数，Y 是 train example

This situation, where we receive similar observations repeatedly, happens quite often in ML: in addition to the car example, we can imagine that X is a parameter vector (such as the weight vector in a classifier) and each Y_t is a training example (a pair of a feature vector and a true label).

每增加一个 example，就可以更新 poterior of X

Bayes filter (add Transition 转移概率)

之前假设 X 只有一个状态，现在 X 也会随时间 t change(robot 可以 move)，X1 可以 transit to X2…

我们已知：

- the initial distribution $P(X_1)$,
- the observation likelihoods $P(Y_t \mid X_t)$,
- and the *transition probabilities* $P(X_{t+1} \mid X_t)$.

$P(Y_t \mid X_t)$，每个不同的 state 对应不同的分布(发射概率)

$$P(X_t \mid Y_t = y_t, H_t) = P(Y_t = y_t \mid X_t, H_t) * P(X_t \mid H_t) / P(Y_t = y_t \mid H_t)$$

假设 $P(Y_t = y_t \mid X_t, H_t) = P(Y_t = y_t \mid X_t)$ 当前观测 t，在给定 Xt 条件下，条件独立于前面所有的观察，也就是不仅是相同 state 下的 obs 独立，

不同的 state 下的 obs 也独立

including) time t. Again we've made a conditional independence assumption to keep things manageable: we are assuming that the transition probabilities don't depend on previous states or observations.

$$P(X_t \mid Y_t = y_t, H_{t+1}) = P(Y_t = y_t \mid X_t) * P(X_t \mid H_t) / P(Y_t = y_t \mid H_t)$$

先求和，再两边乘以分母：左边 $\sum_{X_t} P(X_t, Y_t = y_t \mid H_t) = P(Y_t = y_t \mid H_t)$，

可以求出分母，进而求出后验 $P(X_t \mid Y_t = y_t, H_{t+1}) = P(X_t \mid H_{t+1})$

But now, in between observations, we need to apply the sum rule to handle a transition:

$$P(X_t \mid H_t) = \sum_{X_{t-1}} P(X_t \mid X_{t-1}) P(X_{t-1} \mid H_t).$$

先验概率的计算(利用上一次的后验)：sum rule：

$$P(X_t \mid H_t) = \sum_{X_{t-1}} P(X_t, X_{t-1} \mid H_t) = \sum_{X_{t-1}} P(X_t, \mid X_{t-1}) P(X_{t-1} \mid H_t)$$

每个不同的也就是由 t-1 个观测，

We can continue this process — alternately handling observations and transitions — as long as we like. This process is called a *Bayes filter*. (Some well-known special cases of Bayes filters include Kalman filters and the forward algorithm for hidden Markov models.)

bayes filter 也包括 HMM

联合概率分解+independ+condition independ 大总结！！

概率分解相乘的实质：将我们**观测到的事件结果(A,B,C,D。。)分解成一步一步**，如果不考虑顺序的话，分解顺序不影响最终联合概率结果！**每一步和每一步之间**是概率相乘，如果每一步独立就是**直接概率相乘**，如果不独立就**条件概率相乘**！！

基本公式：

$$P(A,B) = P(A)P(B \mid A) = P(B)P(A \mid B)$$

通常情况我们不考虑顺序(只考虑联合概率，一起发生)，只关系最后结果，任何顺序结果概率一样

如果前一次事件发生不影响下一次事件发生，事件就独立：

$$P(A \mid B) = P(A)$$

$P(A,B)=P(B)P(A)$

联合概率分解公式： (**万能分解公式，这样分解永远成立！**)

$$P(A,B,C)=P(A,B)P(C\mid A,B)=P(A)P(B\mid A)P(C\mid A,B)$$

也就是先一点一点的发生 A,B,C….不考虑顺序，不同的顺序概率一样
关键：**每次事件发生**都受**前面全部所有事件影响！！！**
p(C|A,B):AB 已经先发生

换一种思路！
P(ABC)=P(A)P(B,C|A)=P(A)P(B|A)P(C|AB)

$P(B,C\mid A)$ **条件联合分布也是联合概率，也可以分解 P(B,C|A)=P(B|A)P(C|B,A)**

第一步的条件也要加到第二步(A)！！
而不是：P(B,C|A)=P(B|A)P(C|B)，这里假设了 P(C|B)=P(C|B,A),也就是 A 和 C 条件独立

如果 BC **条件独立：P(B,C|A)=P(B|A)P(C|A)**
从上面看出，条件不影响乘法法则**(运算时可以当 A 不存在,最后每个式子再加 A 条件)**

P(B|A)P(C|B,A) =P(B,C|A)
反过来看：概率相乘**相同的条件可以合并，前一步发生的 B 会消掉下一步的条件**变成联合概率**(也是可以忽略已有条件 A)**

规律：
1 当两个乘子，有相同事件条件时，可以将其忽略再运算
2 同一个事件 B,在一个乘子中已发生，另一个乘子中是条件，可以约去**条件**，转为已发生(条件概率)
同时符合上面的才可以运算合并，说明是由联合分布(或条件联合分布)分解而来
少一个中间乘子都不行

下面是无效的分解：假如抛硬币 A 是正面，B 是反面，没有意义，不可能同时发生

$$P(D\mid B)P(C\mid A)=P(D,C\mid A,B)$$
$$P(C\mid B)P(C\mid A)=P(C\mid A,B)$$

相同条件可以合并，但**不相同条件也不能随便合并，**没有任何法则支撑，没办法运算
还是要像上面那样

以上是联合分布的正常分解：P(ABC)=P(A)P(B,C|A)=P(A)P(B|A)P(C|AB)
特殊的分解：

HMM，状态转移概率: $P(A,B,C)=P(A)P(B\mid A)P(C\mid B)$

HMM 认为 $P(C\mid B)=P(C\mid A,B)$ 也就是 C 跟 A 条件独立

$P(D|C) = P(D|A,B,C)$ D 和 A,B 也条件独立！！

也就是在当前状态条件下，下一个状态和前面状态独立(**条件独立**)，只受前面事件影响

HMM 分解

hmm:是个很好的 conditional independ 和概率分解的例子：

了解HMM的人们，都知道HMM有五个基本要素，三个假设和解决的三个问题：
首先看下HMM的**五个基本要素**：
HMM是个五元组$\lambda = (S, O, \pi, A, B)$

S:状态值集合，O：观察值集合，π：初始化概率，A：状态转移概率矩阵，B：给定状态下，观察值概率矩阵

其次，回忆下HMM的**三个假设**：
1、有限历史性假设，p(si|si-1,si-2,...,s1) = p(si|si-1)
2、齐次性假设，（状态与具体时间无关），P(si+1|si)=p(sj+1,sj)
3、输出独立性假设，输出仅与当前状态有关，P(o1,...ot|s1,...st) = P(ot|qt)

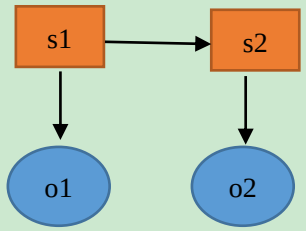
最后，来说下HMM解决的**三个问题**：
1：评估问题，已知模型参数λ=(A, B, π),计算某个观测序列发生的概率，即求P(O|λ)
2：解码问题，给出观测序列O和模型μ，怎样选择一个状态序列S(s1,s2,...st+1),能最好的解释观测序列O
3：学习问题，如何调整模型参数 λ=(π, A, B)，使得P(O|λ)最大?

都是先万能公式分解，然后根据假设进行修改@!

两个事件独立，则任何条件下也这两个事件也会条件独立！

分解方式 1：分成两大步：

1 先生成全部隐状态　2 后全部观察状态(也可以每生成一个 hidden，产生 obs)



$p(s1,s2,o1,o2)=p(s1,s2)p(o1,o2|s1,s2)$
$=p(s1)p(s2|s1)p(o1|s1,s2)p(o2|o1,s1,s2)$
$=p(s1)p(s2|s1)p(o1|s1)p(o2|s2)$

o1 和 s2 无关，则任何条件也条件无关 p(o1|s1,s2)=p(o1|s1)
o2 和 o1 一定无关

假设 o2 只与 s2 有关，o1 和 o2 无关 p(o2|o1,s1,s2)=p(o2|s2)

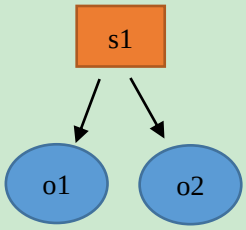
只需要前面发生过一次 p(s1)，后面乘再多以 s1 为条件的概率，都可以！p(s2|s1)p(o1|s1)都以 s1 为条件，乘积里只需要出现一次 p(s1)
再如前面出现 p(s2|s1) 有了 s2 后面都可以 s2 为条件 p(o2|s2)

p(o1,o2|s1,s2)=p(o1|s1)p(o2|s2)

条件独立的另一种形式，两个独立的条件

3 另一种可能是一个 state，观测两次：

hidden state s1，观测两次 o1 o2



p(o1,o2,s1)=p(s1)p(o1,o2|s1)=p(s1)p(o1|s1)p(o2|o1,s1)

=p(s1)p(o1|s1)p(o2|s1)

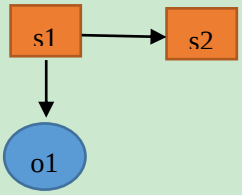
假设 o1 o2 条件独立 p(o2|o1,s1)=p(o2|s1)

其实上面就推出了条件独立的两个公式：

o1 o2 本身不独立，因为 o1 和 o2 有可能是两个 state 生成，state 之间有关联，o1 o2 一定不独立(不是因为因为由同一个 s1 生成(同一个 state 观测还是独立的))

分解方式 2：

因为步奏不影响结果，我们可以考虑另外一个顺序：先 s1，o1 再 s2



p(s1,o1,s2)=p(s1)p(o1|s1)p(s2|s1,o1)=p(s1)p(o1|s1)p(s2|s1)

p(s2)=p(s2|o1) state 只受前面 state 影响，跟观测独立

只需要前面发生过一次 p(s1)，后面乘再多以 s1 为条件的概率，都可以！

HMM general case:

X，Y 都是 300 个 random variables can take value from1-300

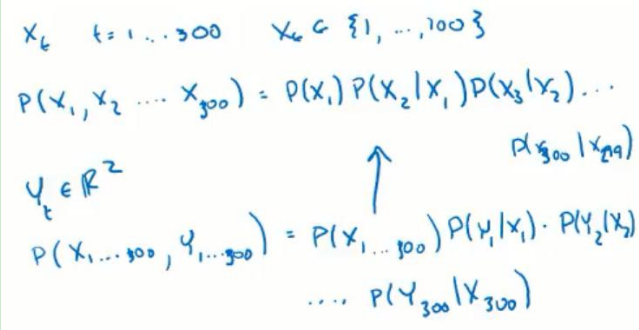
X 是隐状态 Y 是观测状态

X 只是依赖前一个状态

Y 只依赖当前 X P(Y|X)

求我们生成的 300 个数据的概率：

其实就是联合概率的步奏分解（不关心步奏顺序）



分解方式 1:

表示先有状态，再进行观测

P(X1,X2...,Y1,Y2..)=P(X1,X2..)P(Y1,Y2...|X1,X2...)

P(Y1,Y2...|X1,X2...)=P(Y1|X1,X2..)P(Y2,Y3.|Y1,X1,X2.)

=P(Y1|X1,X2..)P(Y2|Y1,X1,X2.)P(Y3..|Y1,Y2,X1,X2...)

又根据假设:

=P(Y1|X1)P(Y2|X2)P(Y3|X3)

最终:

P(X1,X2...,Y1,Y2..)=P(X1,X2..)P(Y1|X1)P(Y2|X2)P(Y3|X3)

分解方式 2:

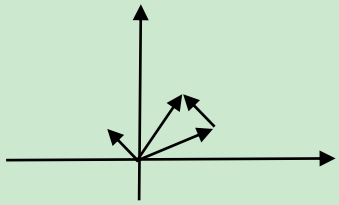
每产生一个 state，观测一次..

所以 p(X1..100,Y1..100)=P(X1)P(Y1|X1)P(X2|X1)P(Y2|X2)..

一步一步这样来的!

只需要前面发生过一次 p(X)，后面乘再多以 X 为条件的概率，都可以!

linear algerbra



a 向量减去 b 向量，得到连线的向量，这个**向量的值**，其实就是将该向量的尾部移动到原点，**箭头指向的坐标就是该向量的值**

Vector space R(向量空间)

向量构成的空间

R space :the space of n element vectors of real number(就是 n 维向量的 space)

property:

It supports addition and scalar multiplication of vectors:
$ax + y \in \mathbb{R}^n$ for $a \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$.

任何两个向量的线性组合：addition 和 scalar multiplication，仍在该 space

这个 property make the vector space(以二维为例，两个向量的线性组合将整个二维平面填充了)

- $\mathbb{R}^n$ has an inner product, written $x \cdot y$ or $\langle x, y \rangle$. We have $x \cdot y \in \mathbb{R}$ for $x, y \in \mathbb{R}^n$.
- The inner product satisfies the usual properties: e.g., it is bilinear (linear in each argument) and symmetric, and $x \cdot x = 0$ iff $x = 0$.
- If we use the inner product to define a norm $\|x\| = \sqrt{x \cdot x}$, then every Cauchy sequence converges. That is, if $\|x_i - x_{i+1}\| \to 0$ as $i \to \infty$ then $x = \lim_{i \to \infty} x_i$ exists and $x \in \mathbb{R}^n$.

The first two of these properties make $\mathbb{R}^n$ an *inner product space*. The last one makes it a *complete* inner product space. (Both of these are stronger properties than just being a vector space.)

向量内积是□，所有可能的内积组成了 inner product space

向量的 norm 膜

Cauchy sequence：当 i 趋向无穷，xi 和 xi+1 非常接近，这个序列 x1，x2，x3…就是

converge：i 趋向无穷 x 存在

complete inner product space question

Other vector space(question)

vector 不只是 list of number，而是任何可以 addition 和 scalar multiplication 的东西

All of these properties make $\mathbb{R}^n$ a nice structure to work with. But sometimes we need to work with objects that are not in $\mathbb{R}^n$: e.g., matrices or functions. We can still use some of the same tricks when working with these objects, by abstracting out the important properties that we like from $\mathbb{R}^n$.

This sort of abstraction is very important in ML: it lets us take algorithms that are designed to learn an element of $\mathbb{R}^n$ (often called a *parameter vector*), and generalize them to work on other classes of objects such as matrices or functions. For some classes of objects, it would be very difficult to design effective learning algorithms any other way.

make the matrix and function a vector space（a vector set）

向量的集合

We start by defining a *vector space* to be a set V (whose elements are called *vectors*), together with operations of addition and scalar multiplication that satisfy the usual axioms: e.g., $ax + y \in V$ for $a \in \mathbb{R}$ and $x, y \in V$. For example:

- The set of matrices $\mathbb{R}^{m \times n}$ is a vector space, if we interpret addition and scalar multiplication elementwise.
- We can similarly make the set of functions $\mathbb{R} \to \mathbb{R}$ into a vector space: we define addition and scalar multiplication to operate separately on each possible argument to our functions. For example, $(f + 3g)(x) = f(x) + 3g(x)$.

One vector space can be contained inside another one: $U \subseteq V$. In this case U is called a *subspace* of V.

1 set of matrices R (m,n)，也可以作为一个 vector space，因为 R(m,n)矩阵的线性组合还是 R (m,n)矩阵

2 set of function R->R,也可以作为一个 vector space，f,g 是空间的向量，f+3g 结果还是 R->R

subspace

subspace 也是个 vector space，并且是 contain 在另一个 vector space 里

既然也是 vector space(vector set),也需要满足上面的线性表示的条件！

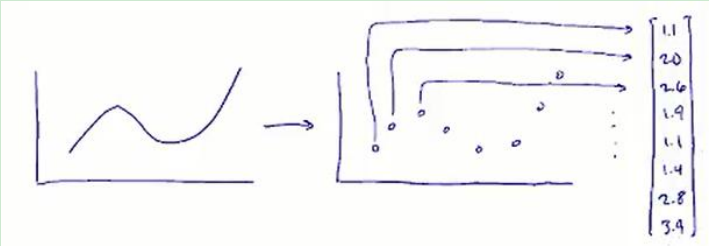
1 从一个子空间 V，获取的向量，线性组合一定还在这个子空间

2 子空间必须过原点，

如 R3 的 subspace，就像 3 维空间的一个过原点的平面(不是 R2)满足上面条件

Vector space of function(function space)

evaluate $f(x_i)$ on a grid of points x_i. If there are n grid points, then we can collect the function values $\tilde{f}_i = f(x_i)$ into a vector $\tilde{f} \in \mathbb{R}^n$.



sample the function y 值，on n grid x，用 list of y 代表 function，

function 是一个 infinite-dimensional vector

function vector space，也满足两个 function 的线性组合还在 function vector space

This vector $\tilde{f}$ behaves a lot like the original function: e.g., we can add together two functions f and g by adding their vectors $\tilde{f}$ and $\tilde{g}$ componentwise, and we can multiply by a scalar $a \in \mathbb{R}$ by taking $a\tilde{f}$ componentwise. These componentwise operations mimic what we would have gotten by first building the function $af + g$ and then sampling it at our grid $x_1 \ldots x_n$.

funciton space 是有多个 function(infinite vector)组成 space

perceptron alg 应用在 function space

feature function：假设是：

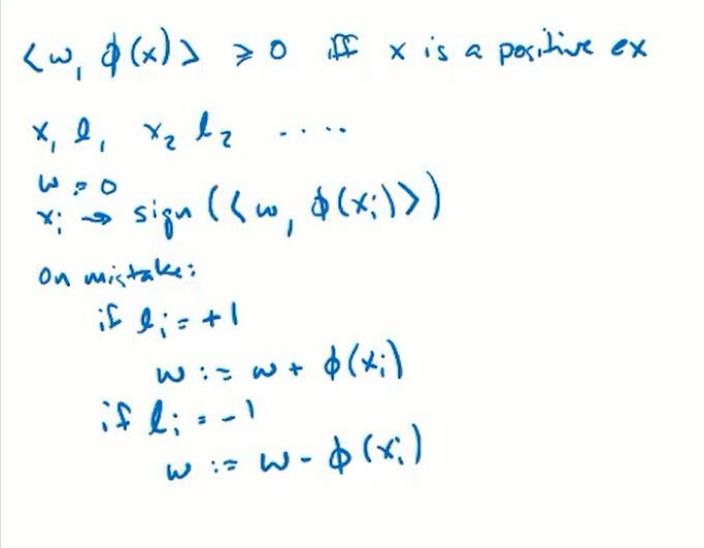
$$\phi(x_i) = q_{x_i} \qquad q_{x_i}(x) = e^{-\|x_i - x\|^2/2\sigma^2}$$

(σ: parameter)

我们希望 phi 和 w 的内积的符号决定 prediction

当 x 是 positive example 时，phi 和 w 的内积为正

每个不同的 x 对应不同的 fucntion (infinite vector)



我们只需 define 这个 inner product 就好了，使得结果是 R

上面 l 是 label

w initial to 0 function(in phi function space)

Complete inner product spaces question？

Above we described how to think of functions or matrices as vectors in a vector space V. We can upgrade V to an *inner product space* by defining an inner product $\langle x, y \rangle \in \mathbb{R}$ for $x, y \in V$; again the inner product must satisfy the properties above like bilinearity and symmetry. For general inner product spaces we typically use the notation $\langle x, y \rangle$; we reserve $x \cdot y$ for the standard inner product in $\mathbb{R}^n$.

For example:

- A useful inner product for matrices is $\langle X, Y \rangle = \sum_{ij} X_{ij} Y_{ij} = \text{tr}(X^T Y)$.
- A useful inner product for functions $f, g \in (X \to \mathbb{R})$ is $\langle f, g \rangle = \int_X f(x) g(x) dx$.
- There are often multiple possible ways to define an inner product for a given vector space. For example, the usual inner product on $\mathbb{R}^n$ is $\langle x, y \rangle = \sum_i x_i y_i = x^T y$. But for any fixed $n \times n$ matrix A, we can define $M = A^T A$ and set $\langle x, y \rangle = x^T M y$ instead; this is called a *Mahalonobis* inner product.

其他内积的定义

As always we define $\|x\| = \sqrt{\langle x, x \rangle}$. Given this norm, we reuse the usual definitions related to convergence:

- A sequence $x_1, x_2, \ldots$ is a *Cauchy sequence* iff $\|x_i - x_{i+1}\| \to 0$ as $i \to \infty$.
- A sequence $x_1, x_2, \ldots$ *converges* to x iff $\|x_i - x\| \to 0$ as $i \to \infty$.

We say V is *complete* if any Cauchy sequence $x_1 \in V, x_2 \in V, \ldots$ converges to some vector $x \in V$. (A complete inner product space is also called a *Hilbert space*, after the mathematician David Hilbert.)

Fisher linear discriminat(generalize our algorithm to another space)

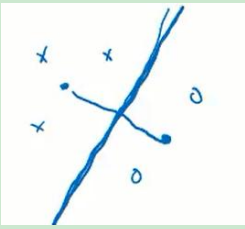
generalize our algorithm to another space form

discriminate function

another way to learn

There are lots of ways to build such a classifier, but one simple one is the *Fisher linear discriminant*:

$$w = E(x \mid \text{class is x}) - E(x \mid \text{class is o})$$
$$b = w \cdot [E(x \mid \text{class is x}) + E(x \mid \text{class is o})]/2$$



linear discirminator 就是两个 class mean 连线的垂直线

Non-linear classifier:

用更 complicated vector space

Then we need to use a more complicated vector space than $\mathbb{R}^2$. For example, suppose we take our two-dimensional points $x = (x_1, x_2)$ and apply a function to each of them, such as:

$$\phi(x) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2) \quad \phi \in \mathbb{R}^2 \to \mathbb{R}^5.$$

将我们每个 2 维的 example 点，变成 5 维的点在 5 维里面(non-linear mapping)，用 linear discriminator（本身是 nolinear function）

Now if we take a vector $w \in \mathbb{R}^5$ and a threshold $b \in \mathbb{R}$, the function $w \cdot \phi(x) - b$ is nonlinear: it consists of a nonlinear mapping from $\mathbb{R}^2$ to $\mathbb{R}^5$, followed by a linear mapping from $\mathbb{R}^5$ to $\mathbb{R}$. In fact, by varying

不再是 linear function，先是 non-linear mapping from 2 维 to 5，然后 linear mapping from 5 维 to 1 维

此时，对于 2 维来说，我们的 non-linear function(classifier boundary)可以是任何曲线

w and b, we can make $w \cdot \phi(x) - b$ be any quadratic function: for example, if we take $w = (0, 0, 0, 1, 1)$ and $b = 0$ we get the function $x_1^2 + x_2^2$.

其实就是二维曲线了，对于高维还是 linear classification

可以用相同的方法学习：

Fortunately, nearly the same recipe as above works just fine: we take

$$w = E(\phi(x) \mid \text{class is x}) - E(\phi(x) \mid \text{class is o})$$
$$b = w \cdot [E(\phi(x) \mid \text{class is x}) + E(\phi(x) \mid \text{class is o})]/2$$

That is, we take the perpendicular bisector between the class means *after applying* the function $\phi(x)$.

思想：feature transform

我们在原本 2 维里，用原本的算法，不能解决问题，将其 mapping 到高维，用同样的算法就可以解决，用同样简单的算法，解决更复杂的问题！！

coding:

```
% givens
xplus = [-1, -1, 1, 1];
yplus = [-1, 1, -1, 1];
xminus = [.25, 0];
yminus = [0, .25];

% calculate phi
phiplus = [xplus; yplus; xplus.^2; yplus.^2];
phiminus = [xminus; yminus; xminus.^2; yminus.^2];
```

2 维 map to 4 维

```
% calculate conditional expectations of phi
ephiplus = mean(phiplus, 2);
ephiminus = mean(phiminus, 2);
```

```
% definitions of w and b from notes
w = ephiplus - ephiminus
b = 0.5 * w' * (ephiplus + ephiminus)
```

计算出 w，b

换一种方法，不去映射，而是直接学习到我们想要的 function

The above example shows how to use a feature transform to learn from a simple class of nonlinear functions, such as polynomials of a fixed degree. What if we want to choose our discriminant function from a larger class of functions — say, all infinitely differentiable functions?

不只是我们变化的这几种 feature 形式 x_1x_2, x_1^2, x_2^2,让算法自己选择

更多的特征

仍然可以用 fisher discriminate

This is where it helps to think of vector spaces whose elements are functions. It turns out that we can make a vector space whose elements are (essentially) the infinitely differentiable functions in $[-1,1] \times [-1,1] \to \mathbb{R}$; and, we can design ML algorithms that work directly on this vector space.

需要用到函数空间，元素是函数，然后让 ML 算法作用在该空间上面

Geometry of function

linear function

$$f_{w,b}(x) = \langle w, x \rangle - b$$

To simplify, we'll start with $b = 0$, so that we are looking just at the effect of w. We can think of w in several ways: w is a vector in V, but it also represents the linear function

$$f_w(x) = \langle w, x \rangle$$

as well as the plane

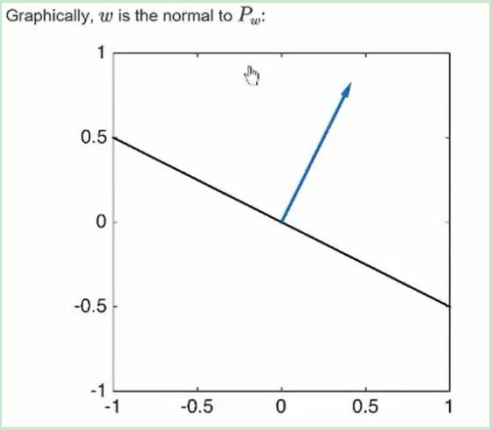
$$P_w = \{x \mid f_w(x) = 0\}.$$

P_w is the zero level set of f_w, and the decision boundary of our classifier.

对 w 的解释：

1 w 是向量

2 decision boundary（plane）的法向量，$3x+2y+z=0$是三维的一个平面，(3,2,1)是 w， plane 也是由 w 决定的



上图注意：因为 b 为 0，plane 必须过原点

w 是 plane 的法向量(norml to plane) (3,2,1)是个向量，垂直于 plane：

证明：

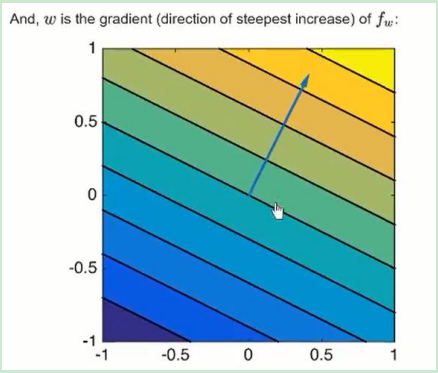
$$3x_1+2y_1+z_1=0$$
$$3x_2+2y_2+z_2=0$$

$$3(x_1-x_2)+2(y_1-y_2)+(z_1-z_2)=0$$

两向量相减$(x_1-x_2,y_1-y_2,z_1-z_2)$，得到的向量一定在该平面(也就是平面上两点的连线一定还在平面上) 与 w 相乘为 0，w 垂直于平面

3 w 方向是 gradient 的方向指向函数下降最快的方向

And, w is the gradient (direction of steepest increase) of f_w:

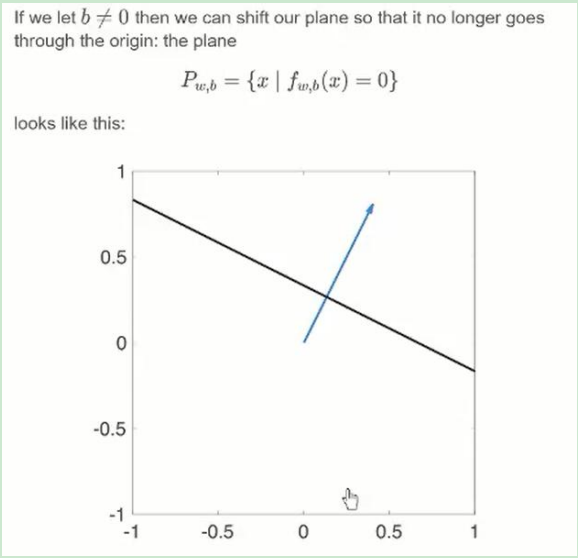


当 b 不是 0

If we let $b \neq 0$ then we can shift our plane so that it no longer goes through the origin: the plane

$$P_{w,b} = \{x \mid f_{w,b}(x) = 0\}$$

looks like this:



plane 不过原点！

同时 scale w,b，不会 change plane，等式两边乘以任何值，等式不变

如果单独 scale：

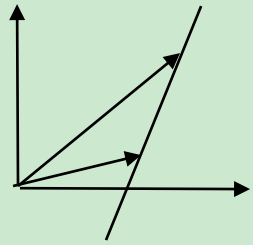
- Scaling up b moves the plane farther away from the origin
- Scaling down b moves the plane closer to the origin.
- Scaling w has the reverse behavior: making w bigger moves the plane closer to the origin, and making w smaller moves it farther away from the origin.

Vector space and dimension and span

1 向量 in vector space

向量其实就是空间中的一个点！！！

向量在一个空间，就是点在这个空间



假设直线，直线上的两个点就代表这两个向量！！

但上面直线不是 vector space，vector space 必须过原点，才能保证上面的任意两个向量的线性组合还在这个 space，上面不满足！

2 span 和 subspace

1. Linear combination; span; linear independence

If $x_1, x_2, \ldots x_k \in V$ are vectors and $a_1, a_2, \ldots a_k \in \mathbb{R}$ are scalars, then the vector

$$x = a_1x_1 + a_2x_2 + \ldots + a_kx_k \in V$$

is called a *linear combination* of the x_i. The set of all such x (for all possible ways to set $a_1, \ldots, a_k$) is called the *span* of the x_i.

假设从 V 里获取 k 个向量 x(set of vectors)，这些向量全部的线性组合 x 就是 span of the vector set！记为 span(x1,x2..xk)

Given a set of vectors $\mathcal{X} = \{x_1, x_2, \ldots, x_k\}$ in a vector space V, the set of linear combinations

$$S = \text{span}(\mathcal{X}) = \{a_1x_1 + \ldots + a_kx_k \mid a_i \in \mathbb{R} \text{ for each } i\}$$

is a subset of V. It is also itself a vector space, called a *subspace* of V: it's straightforward to show that S is closed under addition and scalar multiplication. If the vectors in $\mathcal{X}$ are linearly independent, they form a basis for S.

span 一定等于 V 或是 V 的 subspace(因为只要是线性组合，一定是 space)，比如三维空间，两个向量(001)(101),span 只是个三维空间的平面，是 subspace!

如果这个 set 的向量 linear independent，是这个 subspace(span)的 basis，数量不多不少

2. Basis; dimension

Suppose that $\{x_1, \ldots, x_k\}$ is a linearly independent set of vectors in V. We might be able to extend the set: that is, there might exist another vector x_{k+1} that we can add to the set while preserving linear independence. Or, we might not: it could be that any other vector that we add causes the set to become linearly dependent.

In this latter case, the set $\{x_1, \ldots, x_k\}$ is called a *basis* for the vector space V, and the size k of the set is called the *dimension* of V. (This is the same idea of dimension that we're used to: a line is a one-dimensional vector space, a plane is a two-dimensional one, etc.) If $\{x_1, \ldots, x_k\}$ is a basis for V, then any $x \in V$ can be expressed (uniquely) as a linear combination of the x_i.

Dimension is geometrically related to linear independence: for a set of vectors to be linearly independent, no k of them can lie in any $(k-1)$-dimensional subspace. For example, no two vectors can lie on the same line, and no three vectors can lie on the same plane.

orthogonal comlement of S

如果 S 是 strict subspace：V 如果存在向量,但不在 S

If S is a strict subspace of V, we can add more vectors to $\mathcal{X}$ to make a basis for V; call the set of additional vectors $\mathcal{Y} = \{y_1, \ldots, y_\ell\}$. If V has an inner product, we can always choose $\mathcal{Y}$ so that all of its vectors are orthogonal to the vectors in $\mathcal{X}$. If we do so, $\text{span}(\mathcal{Y})$ is called the orthogonal complement of S, written $S^\perp$.

S(X)是 V 的 subspace，我们可以 add more vector Y(vector set)to X，使得 X 成为 V 的 basis

如果所有的 Y 里的 vector orthogonal to the X，span(Y)就是正交互补 of span(X)　也就是 Y 组成的 subspace 和 X 组成的 subspace 正交互补

DEFINITION The orthogonal complement of a subspace V contains every vector that is perpendicular to V.

Taking the orthogonal complement twice gives us back the original subspace: $S^{\perp\perp} = S$. And, if we define

做两次的话，又回到 S

subspace: $S^{\perp\perp} = S$. And, if we define

$$S + S^{\perp} = \{x + y \mid x \in S, y \in S^{\perp}\}$$

then $S + S^{\perp} = V$.

question ?如何相加

两个正交补子空间相加就是完整该维度的空间

By definition, the span of an empty set of vectors is equal to $\{0\}$. And, the orthogonal complement of the entire vector space V is defined to be $V^{\perp} = \{0\}$.

Range and nullspace and orthogonal complement

Range

If $A \in \mathbb{R}^{m \times n}$ is a matrix, we write $\text{span}(A)$ for the span of the columns of A. For any $x \in \mathbb{R}^n$, the vector Ax is a linear combination of the columns of A, with coefficients equal to the components of x; so, $Ax \in \text{span}(A)$. For this reason, we call $\text{span}(A)$ the *range* of A.

span（matrix）　Ax 是 A column vector 的 linear combination

span(A)的另一个名字：range of martrix A

Null space

The *nullspace* of A is the set

$$\text{null}(A) = \{x \mid Ax = 0\}.$$

The nullspace is also called the *kernel*.

kernel of A

A famous theorem of linear algebra says that, if we define

$$R = \text{span}(A) \quad N = \text{null}(A^T)$$

then

$$R = N^\perp$$

which means also that $N = R^\perp$ and $R + N = \mathbb{R}^m$. That is, the range of A and the nullspace of A^T are two orthogonal subspaces that sum to the entire vector space $\mathbb{R}^m$.

R 和 N 互相 orthogonal complement，向量互相两两垂直

The range and nullspace of a linear operator $A \in U \to V$ are defined similarly, and an analogous theorem holds: the range of A and the nullspace of A^* are orthogonal and sum to all of V.

linearly dependent and independent

A 的列向量 如果向量之间存在线性关系，也就是一个向量被其他向量表示，也就**存在一组不为 0 的 x**，使得 Ax=0，则 **A 的列向量线性相关**

一个向量 set 线性关系，也就是有多余向量

linearly independent

如果，不存在，Ax=0 只有 0 解，线性无关！

Dimension is geometrically related to linear independence: for a set of vectors to be linearly independent, no k of them can lie in any $(k-1)$-dimensional subspace. For example, no two vectors can lie on the same line, and no three vectors can lie on the same plane.

几何意义：k 个向量，如果都在 k-1 个平面就说明线性相关

如:三个向量如果都在一个平面就线性相关

check:

rank(vector set V)得到最大 size of linearly independent subset of V

Basis & dimension

Suppose that $\{x_1, \ldots, x_k\}$ is a linearly independent set of vectors in V. We might be able to extend the set: that is, there might exist another vector x_{k+1} that we can add to the set while preserving linear independence. Or, we might not: it could be that any other vector that we add causes the set to become linearly dependent.

假设一个线性独立的 vector set，如果存在一个 vector 添加进 set 后仍然线性独立，如果不存在，说明这个 set of vector V,可以表示 V 这个维度任意向量了，再添加一个就多余了(就 linear dependent 了)

enough independent vectors to span （填充）the space！！

In this latter case, the set $\{x_1, \ldots, x_k\}$ is called a *basis* for the vector space V, and the size k of the set is called the *dimension* of V. (This is the same idea of dimension that we're used to: a line is a

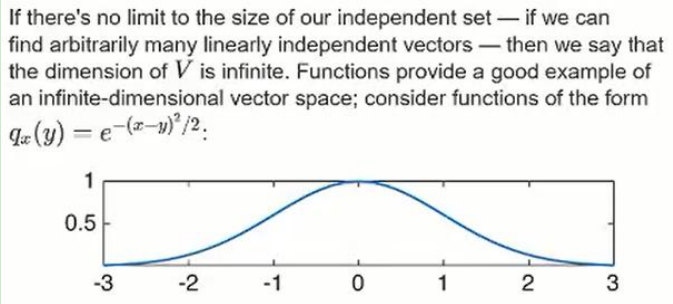
这个线性独立的 vector se 的 span 是 V,是 Basis for V ， basis 的个数是 V 的 dimension

就像 R3 三维平面里的一个二维 subspace，假如只有一个向量，自己是 linearly independent，再增加一个向量(也在子空间)这两个向量，就是子空间的 basis，这个 subspace dimension 是 2 维度

其实就是我们理解的空间维度，n 维空间至少需要，n 个向量的 basis

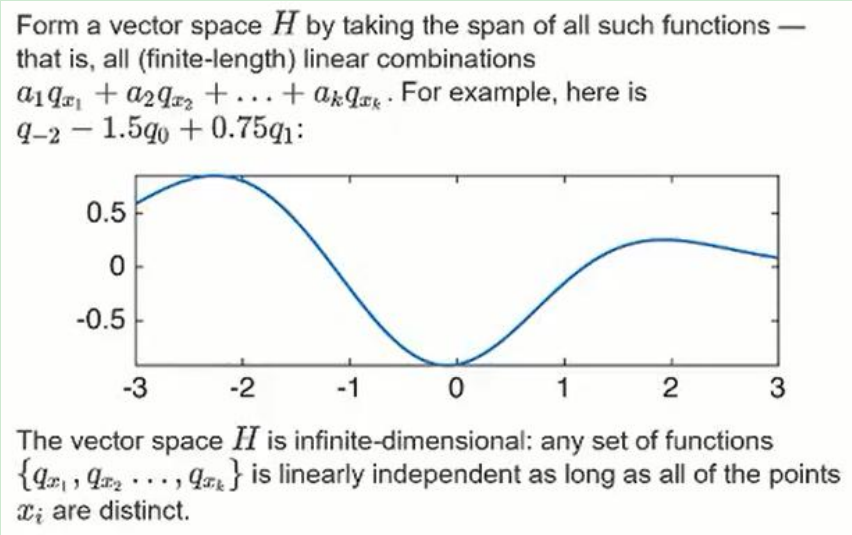
infinite dimensional vector space(question)

space of function vector has infinite dimension



无论增加多少向量，**都不可能被其他向量线性表示**

是关于 y 的函数，x 是参数，不同的 x 取值，y 函数不同，对应不同的函数向量 q，不同的 x 其实是平移该曲线



每个函数，都不可能被其他函数线性表示，只要参数 x 不同

上面的例子，看出线性组合的图像，跟**原来的函数完全不同了**，不能被线性表示了

证明

(It's nontrivial to prove the above fact, but it's easy to check it: for any natural number k, we can build a set of k such functions $q_{x_1}, q_{x_2}, \ldots$ using widely spaced points x_i. If we sample each one of these functions at k distinct points — say $x_1, x_2, \ldots$ — we'll get a set of k vectors in $\mathbb{R}^k$. We can use a language such as Matlab to check that these vectors are linearly independent. And, if the vectors are linearly independent, then so must be the functions: since the vectors are linearly independent, we can't get a linear combination of our functions to be zero at the k sample points. So, we certainly can't get a linear combination of our functions to be zero everywhere.)

inner product of fuction(question)

To work with a vector space of functions, we'll need to pick an inner product. We saw one above: $\langle f, g \rangle = \int_{\mathbb{R}} f(x)\, g(x)\, dx$. This is a perfectly good inner product for many purposes, but it turns out that it's not great for learning; so, for learning, we'll define a different inner product.

因为是连续值，所以只能求积分

可以定义不同形式的 inner product，（只是几何意义改变）

这是更深的另一门课程！！！ 对特殊的向量定义 inner product，然后 generalize to 一般的向量

that we have an inner product on H, and that it satisfies the usual properties like being symmetric and bilinear.

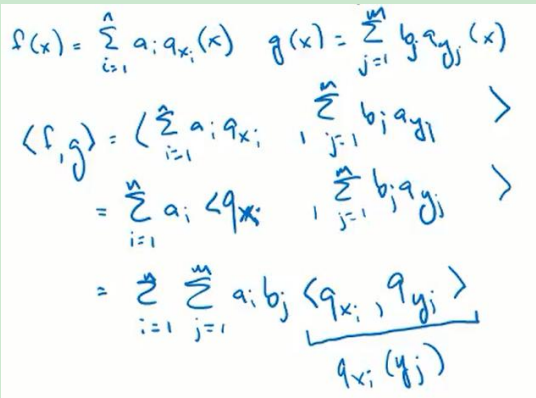
We'll start by defining the inner product of two basis vectors q_x and q_y:

$$\langle q_x, q_y \rangle = q_x(y) = q_y(x)\,.$$

qx qy 是 basis 的任意两个 vecter 先定义特殊向量的 inner product basis vector 的内积：q function 是对称，并且 bilinear 双线性函数

We can then work out the inner product of any pair of functions $f, g \in H$ by expressing f and g as linear combinations of basis vectors: use the equation above and the fact that $\langle f, g \rangle$ is bilinear in f and g.

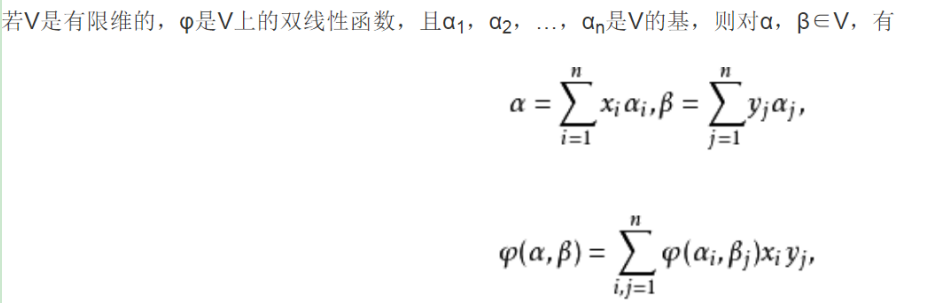
可以求 H space 里任意两个函数的 inner product，只需将这两个函数写成 2 个不同 basis vectors 的线性组合(H 的两个 basis)



假如我们有两个函数，每个函数都是两个 basis 的线性组合

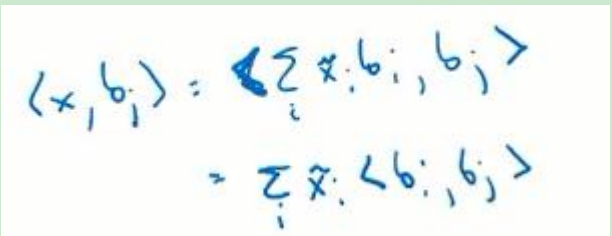
上面的 qxi qyi 只是为了区分两个函数 f,g,

上面最关键一步，是利用了 inner product 的 bilinear 双线性的性质：



可以提出来线性部分

另一种：其中一个是 sum，也可以单独提出



Linear classification in infinite demension

Earlier we learned a nonlinear classifier via a feature function:
by taking

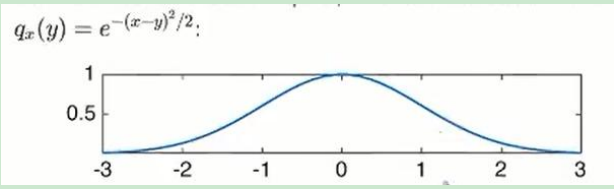
$$\phi(x) = (x_1, x_2, x_1x_2, x_1^2, x_2^2)$$

we learned a discriminant function of the form $w \cdot \phi(x) - b$, which allowed us to implement any quadratic boundary between classes.

Now let's consider the feature function

$$\phi(x) = q_x \in \mathbb{R} \to \mathbb{R}$$

如果选 feature function 如下图的 fucntion:



把 x 映射成 function vector（infinite vector）属于 H

map R to H

$q_x(y) = e^{-(x-y)^2/2}$. This feature function has type $\phi \in \mathbb{R} \to H$, where $H \subset (\mathbb{R} \to \mathbb{R})$ is our infinite-dimensional vector space of functions. So, for any real argument x, the value of $\phi(x)$ is a function from reals to reals.

对于任意的 x 都对应一个 infinie y vector

也就是将 x R map 成 vector H(将一个 x 通过 feature transform 变成无数个 feature)

一个 fucntion vector，其实就是将其所有的函数值变成向量 (infinite vector)

还需要一个 weight vector in same function space H

space. Why would we do this? Well, if we pick a vector $w \in H$ and a threshold $b \in \mathbb{R}$, we can write down a discriminant function

$$f_{w,b}(x) = \langle w, \phi(x) \rangle - b.$$

在 H 空间是 linear function，在 x 空间是非常 non-linear！！

This discriminant function is extremely flexible: it turns out that, by choosing w, b appropriately, we can realize any discriminant function in the entire infinite-dimensional vector space H. So, if we can find an algorithm to learn this kind of classifier, it will be very useful: we won't have to worry about whether our discriminant function should be quadratic or cubic or something else, since we can learn just about any discriminant function that Nature might throw at us.

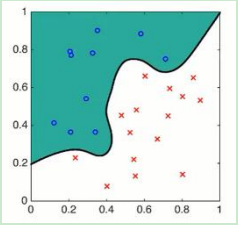
我们可以 learn 近似**任意 function，**不用提前定义 feature 函数形式，quaqactic，cos，cube

而且任然可以用我们之前的 fisher disciminant 算法

6. Infinite-dimensional Fisher discriminants

In fact, the same algorithm that we used before still works! The Fisher linear discriminant is defined to be the perpendicular bisector between the class means *in our vector space* H. In equations:

$$w = E(\phi(x) \mid \mathrm{x}) - E(\phi(x) \mid \mathrm{o})$$
$$b = w \cdot [E(\phi(x) \mid \mathrm{x}) + E(\phi(x) \mid \mathrm{o})]/2$$



学习到更复杂的边界！！！

例如：

In this example, we used points $x \in \mathbb{R}^2$, and we used the feature function

$$\phi_x(y) = e^{-\|x-y\|^2/2\sigma^2},$$

也就是将 R2 转成 函数向量 infinite vector

The parameter σ is called the *kernel width*. It influences how "wiggly" our function will be: the length scale in x of the wiggles will be on the order of σ. (So, large values of σ make us learn very smooth functions, and small values let us learn less-smooth functions.) In the two plots above we used $\sigma = 0.07$; in the second plot you can see that the bumps in the function have width on the order of σ. Because of this, the bends in the class boundary in the first plot have radius of curvature on the order of σ.

sigama：**kernel width 越大，上下波动的成分越少**

String

Another interesting example is strings: let Σ be a finite alphabet, and take our generating set G to be the set of all finite-length strings over Σ (that is, sequences of zero or more characters from Σ). For example, if we take Σ to be the set of ASCII characters, vectors are linear combinations of the form

$$3 \cdot \text{"hello"} + 5 \cdot \text{"world"} - 2 \cdot \text{"!"}$$

To define an inner product, we set $\langle x, y \rangle$ for $x, y \in G$ to be the number of common substrings in x and y. For example,

$$\langle \text{"aard"}, \text{"vark"} \rangle = 4$$

because the strings "aard" and "vark" have four substrings in common (the empty string and the strings "a", "r", and "ar"). Or,

$$\langle \text{"lolly"}, \text{"pop"} \rangle = 2$$

because the strings "lolly" and "pop" have two substrings in common (the empty string and the string "o").

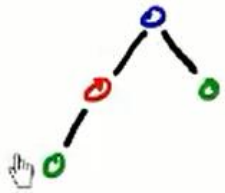
We could also make variants of this inner product by counting substrings with repetition, or by allowing substrings that skip over characters. In the latter case, for example, "loan" and "loon" would share the substring "lo*n".

This vector space may seem weird, but it is actually pretty useful for learning text classifiers: e.g., to determine whether an email is spam, we might want to look at how many substrings it has in common with known-spam emails that we've seen previously.

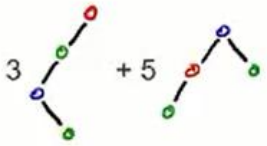
tree

5. Trees

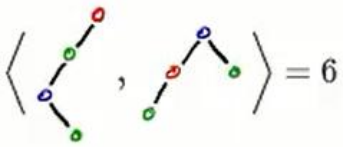
We can do something similar for trees: let Σ again be a finite set, and take our generating set G to be the set of all binary trees whose nodes are labeled with elements of Σ. For example, if we let $\Sigma = \{\text{red}, \text{green}, \text{blue}\}$, elements of G might look like this:



As always, vectors are linear combinations of elements of G:



We can define an inner product on elements of G by counting common subtrees: for example,



because they share the subtrees

Function (operator)on a vector space

map vector in one vector space to another vector in another vector space

例子：

- We could define a function $r \in \mathbb{R}^2 \to \mathbb{R}^2$ that takes a vector and rotates it 30° clockwise.

Or, we could define a function $p \in \mathbb{R}^5 \to \mathbb{R}^2$ that extracts the first and fourth components of a vector (discarding the second, third, and fifth).

oprator

2. Functionals, operators

We've mostly looked so far at real-valued functions on a vector space: that is, $f \in V \to \mathbb{R}$ for some vector space V. Such functions are sometimes called *functionals*.

It's also interesting to look at functions from one vector space to another: $f \in U \to V$ for vector spaces U, V. These functions are sometimes called *operators*. Note that a functional is a special case of an operator: $\mathbb{R} = \mathbb{R}^1$ is itself a vector space.

operator 是向量版的 function，map one vector space to another

linear function(operator)

3. Linear functions

A function $f \in U \to V$ (either a functional or an operator) is called *linear* if it satisfies

$$f(ax + by) = af(x) + bf(y)$$

for all vectors $x, y \in U$ and scalars $a, b \in \mathbb{R}$. We've already spent some time working with linear functionals, such as a linear discriminant function. Now we're going to take a look at linear operators.

linear function 可以将 x 维度变成另一个维度的 y，只需要满足：

pull addtion and scale out！！

f 可以穿透，直达变量

example:

A well-known example of a linear operator $f \in \mathbb{R}^n \to \mathbb{R}^m$ is multiplication by an $m \times n$ matrix, $f(x) = Ax$. A less-well-known

矩阵乘以向量！ 这个矩阵就是 linear operator

example:

example is a derivative operator: for example, we can define an operator $\frac{d}{dx}$ that acts on differentiable functions in a subspace of $\mathbb{R} \to \mathbb{R}$, and we have $\frac{d}{dx}(af + bg) = a\frac{d}{dx}f + b\frac{d}{dx}g$. This latter example shows that we can have linear operators that act on infinite-dimensional vector spaces.

Coordinates

我们对于**向量越来越模糊，模糊的对象**

So far we've mostly been talking about vectors as opaque objects: the only operations they support are addition and scalar multiplication (and sometimes inner product). This is in contrast to the usual practice for the vector space $\mathbb{R}^n$, where we think of a vector as having a concrete representation as a tuple of real numbers. In this latter view, vectors support an additional operation, *indexing*: we can ask for (say) the second coordinate of a vector.

The first view above is called *coordinate-free* or *abstract*, while the second is called *concrete*. The abstract view of vectors can help simplify proofs; but, if we want to manipulate vectors on a computer, a concrete representation in terms of coordinates is necessary.

一种获得抽象向量坐标的方法：

To go from abstract to concrete, we need a basis $B = \{b_1, \ldots, b_n\}$ for our vector space. Recall that we can express any vector x uniquely as a linear combination of basis vectors, $x = \bar{x}_1 b_1 + \bar{x}_2 b_2 + \ldots + \bar{x}_n b_n$. We can take the scalars $\bar{x}_1, \ldots, \bar{x}_n$ as coordinates for x:

$$x \in V \leftrightarrow \begin{pmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_n \end{pmatrix} \in \mathbb{R}^n .$$

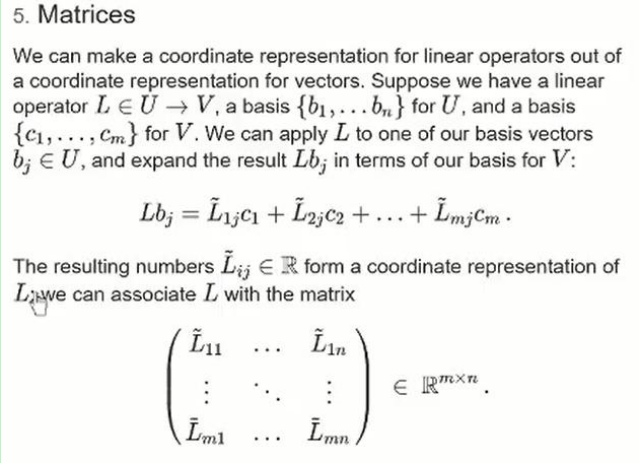
This coordinate representation depends on which basis we pick: if we choose a different basis B', the representation of a given vector x can look completely different. It's important to remember that the vector x hasn't changed, only our representation of it.

choose a basis，用线性组合的**每个系数**列表代表每个向量！！

向量从此有了坐标

coordinate for operation

operation 用矩阵来表示



5. Matrices

We can make a coordinate representation for linear operators out of a coordinate representation for vectors. Suppose we have a linear operator $L \in U \to V$, a basis $\{b_1, \ldots b_n\}$ for U, and a basis $\{c_1, \ldots, c_m\}$ for V. We can apply L to one of our basis vectors $b_j \in U$, and expand the result Lb_j in terms of our basis for V:

$$Lb_j = \tilde{L}_{1j}c_1 + \tilde{L}_{2j}c_2 + \ldots + \tilde{L}_{mj}c_m\,.$$

The resulting numbers $\tilde{L}_{ij} \in \mathbb{R}$ form a coordinate representation of L; we can associate L with the matrix

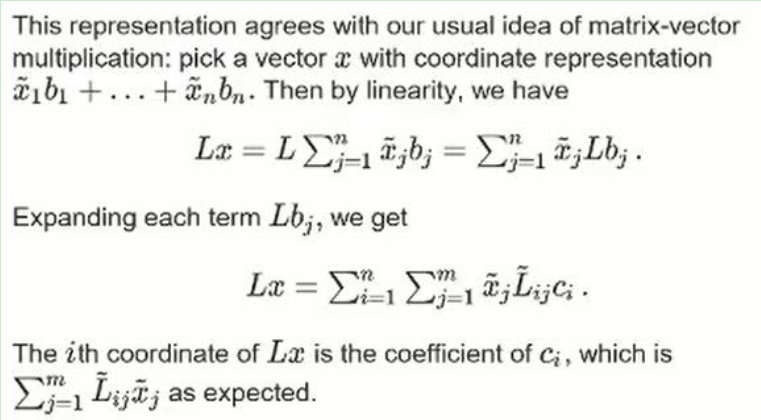
$$\begin{pmatrix} \tilde{L}_{11} & \ldots & \tilde{L}_{1n} \\ \vdots & \ddots & \vdots \\ \tilde{L}_{m1} & \ldots & \tilde{L}_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

linear Operator L map U to V

从 U 和 V 各选一个 basis

对 U 的 basis 向量进行线性操作：

Lb 应该是 V 的**一个向量**，可以用 V 的 basis 表示，系数是 L tilder **向量**。所有 b，都这样做，得到个 L tiled 矩阵(如上)，这个矩阵就是 representation of L



This representation agrees with our usual idea of matrix-vector multiplication: pick a vector x with coordinate representation $\tilde{x}_1 b_1 + \ldots + \tilde{x}_n b_n$. Then by linearity, we have

$$Lx = L\sum_{j=1}^{n} \tilde{x}_j b_j = \sum_{j=1}^{n} \tilde{x}_j Lb_j\,.$$

Expanding each term Lb_j, we get

$$Lx = \sum_{i=1}^{n}\sum_{j=1}^{m} \tilde{x}_j \tilde{L}_{ij} c_i\,.$$

The ith coordinate of Lx is the coefficient of c_i, which is $\sum_{j=1}^{m} \tilde{L}_{ij}\tilde{x}_j$ as expected.

随机在 U 里取一个向量 x，可以用 basis b 表示

Lx ，因为 L 是 linear operator，可以将系数和 add 提出，直达 b

又 Lb $Lb_j = \tilde{L}_{1j}c_1 + \tilde{L}_{2j}c_2 + \ldots + \tilde{L}_{mj}c_m\,.$

由是 V 向量，可以由 V basis 表示，带入(上面公式有误)

应该是$\sum_{j=1}^{n}\sum_{i=1}^{m}$ $Lx = \sum_{i=1}^{n}\sum_{j=1}^{m}\tilde{x}_j\tilde{L}_{ij}c_i\,.$

用 coordinate 矩阵乘以 x 的 corodinate 向量,得到的坐标刚好是 V 的坐标,也就得到 V 向量！！(也就是用矩阵乘等价于 L operator：Lx)

As was the case for vectors, picking different bases will result in a completely different matrix to represent L. As before, the operator L has not changed, only our representation of it.

It's sometimes helpful to think of the matrix $\tilde{L} = (\tilde{L}_{ij})$ as a linear operator in $\mathbb{R}^n \to \mathbb{R}^m$, by contrast with L, which is a linear operator in $U \to V$. Our coordinate representations associate each vector $x \in U$ with a vector $\tilde{x} \in \mathbb{R}^n$ and each vector $y \in V$ with a vector $\tilde{y} \in \mathbb{R}^m$ in such a way that

$$Lx = y \quad \text{is equivalent to} \quad \tilde{L}\tilde{x} = \tilde{y}\,.$$

On the left, Lx means to apply the linear function L to the vector $x \in U$; on the right, $\tilde{L}\tilde{x}$ means to multiply the matrix $\tilde{L} \in \mathbb{R}^{m\times n}$ by the vector $\tilde{x} \in \mathbb{R}^n$.

用矩阵来 represent linear operator

linear operator (x)等价于用这个矩阵去乘 x！！！！！

Find coordinate representation！！

1. Calculating coordinates

Suppose we have a vector space V and a basis $B = \{b_1, b_2, \ldots, b_n\} \subseteq V$ for V. We noted above that any $x \in V$ has a unique representation in terms of B:

$$x = \sum_{i=1}^{n}\tilde{x}_i b_i$$

It makes sense to ask: how can we find this representation given x and B? That is, how can we calculate the coordinates $\tilde{x}_i$?

有了 B 和 x 之后，如何计算 cooradinate of vector

这就是 system of equation 方程组

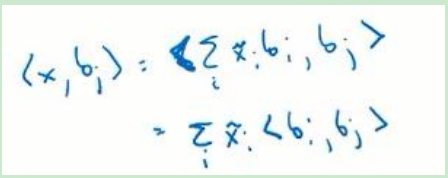
2. A system of equations

We will start by writing down a system of linear equations that the coordinates must satisfy. We will then solve this system to find the desired coordinates.

To get our equations, suppose we take the inner product between x and one of the basis vectors b_j. By linearity we have

$$\langle x, b_j \rangle = \sum_{i=1}^{n} \tilde{x}_i \langle b_i, b_j \rangle$$

为了得到 x 的坐标，我们定义 x 和 basis vector 的内积，再带入上上面的 x，得到 **inner product**



Define $g_j = \langle x, b_j \rangle$ and $G_{ji} = \langle b_i, b_j \rangle$. We then have

$$g_j = \sum_{i=1}^{n} \tilde{x}_i G_{ji}$$

for all j. This is our desired system of linear equations. For convenience, we can represent the same system in matrix form:

$$g = G\tilde{x}$$

where g is the vector with components g_j, $\tilde{x}$ is the vector with components $\tilde{x}_i$, and G is the matrix with entries G_{ij}. We know G and g; solving the system will give us $\tilde{x}$. (Note that G is symmetric, $G_{ij} = G_{ji}$, by symmetry of the inner product. So, this is a symmetric system of linear equations.)

上面是个 matrix multiplication
记住 inner product 都是 R scalar

构建了 **x tilde** 的一个方程组，**解出 x 就好**

g 和 G 都是内积(已经定义好了)，我们可以计算，然后解出 x tilde

4. What if B is not a basis?

What happens if we accidentally start from a set B that is not really a basis for V? There are two ways that B could fail to be a basis:

- B might be linearly dependent: we might have accidentally included a vector that can be expressed as a linear combination of the other vectors in B. (At least, to machine precision.)
- B might not include enough vectors: there might be vectors in V that cannot be expressed as a linear combination of vectors in B.

In the first case, our software library will typically complain when we ask it to solve the system of equations: e.g., it might throw a division-by-zero exception, or it might warn of numerical problems (roundoff errors). In some cases, our library might try to be clever: e.g., it could set some of the coefficients $\tilde{x}_i$ to zero, perhaps in combination with a warning. This sort of cleverness is usually OK: we'll still have $x = \sum_{i=1}^{n} \tilde{x}_i b_i$, even if n is not as small as it could be. Sometimes, the library might fail silently: it thinks that it found the correct $\tilde{x}$, but due to numerical problems, we do not have $x = \sum_{i=1}^{n} \tilde{x}_i b_i$.

In the second case, we will always get a silent failure: everything will seem to go fine, and we will get a back a vector of coordinates $\tilde{x}$. But we will have nonzero residual:

$$r = x - \sum_{i=1}^{n} \tilde{x}_i b_i \neq 0$$

For both of the above cases, it's wise to check at the end whether $r = 0$ to sufficiently high precision, in order to catch possible silent failures.

Matrix:

1 Slice operation

slice of matrix

Transpose and adjoint(共轭)

transpose on matrix

ajoint on operater(operator 也可以矩阵表示)，将 一个 linear operator U to V,变成 linear function V to U (adjoint 也是线性 operator)，是 linear operator 的逆运算

transpos 和 ajoint 两者本质一样

Let U and V be Hilbert spaces (complete inner product spaces). For any linear operator $L \in U \to V$, we define the *adjoint* operator $L^* \in V \to U$ by the property

$x \in U, y \in V$

$< Lx, y >=< x, L^* y >$ 左边都是 V 的 vector 右边是 U 的 vector inner product 一样 question

每个 liner operator L 都会有一个 L*,两者的关系:将坐标矩阵 transpose

L*就是将 L 的坐标矩阵转置！！！

If we have a coordinate representation $\tilde{L} \in \mathbb{R}^{m \times n}$ of L, then the coordinate representation of the adjoint L^* is the matrix transpose

$$\tilde{L}^T = (\tilde{L}_{ji}) = \begin{pmatrix} \tilde{L}_{11} & \dots & \tilde{L}_{m1} \\ \vdots & \ddots & \vdots \\ \tilde{L}_{1n} & \dots & \tilde{L}_{mn} \end{pmatrix}.$$

The columns of $\tilde{L}^T$ are the rows of $\tilde{L}$, and vice versa.

所有 adjoint tanspose 都是 linear operator，都是转置

We can view adjoint and transpose themselves as operators: the adjoint operator maps $L \in U \to V$ to $L^* \in V \to U$, while the transpose operator maps $\tilde{L} \in \mathbb{R}^{m \times n}$ to $\tilde{L}^T \in \mathbb{R}^{n \times m}$. The adjoint and transpose operators are linear: e.g., $(A + B)^* = A^* + B^*$.

执行两次，还是原来的 linear operator

If we take the adjoint twice, we get back to the original operator: $L^{**} = L$. Similarly, taking the transpose twice gets back to the original matrix: $\tilde{L}^{TT} = \tilde{L}$.

对称矩阵：转置不变

If $U = V$ we can potentially have $L = L^*$; such an operator is called *self-adjoint*. Similarly, we can have $\tilde{L} = \tilde{L}^T$; such a matrix is called *symmetric*. Self-adjoint operators correspond to symmetric matrices.

Inverse operator

可以操作 matrix 和 linear operator

inverse 和 adjoint 不同，但是有相同的 input output space

inverse 不一定存在，adjoint 一定存在

Let U and V be Hilbert spaces (complete inner product spaces). For a linear operator $L \in U \to V$, we say that $L^{-1} \in V \to U$ is the *inverse* of L if it satisfies

$$x = L^{-1}y \text{ whenever } y = Lx$$

or equivalently

$$x = L^{-1}Lx, \quad y = LL^{-1}y \text{ for all } x \in U, y \in V.$$

inverse 也就是抵消还原 L 的操作

x 属于 U，y 属于 V y=Lx

如果存在 inverse，L is invertble

如果不存在 inverse，L is singular 奇异的 矩阵

If L is not invertible, it is called *singular*. We might still want an operator that acts like the inverse "whenever that makes sense." Such an operator is called the *pseudoinverse*, written $L^\dagger$. Formally, we define $L^\dagger$ to be the operator such that, if we take $x = L^\dagger b$, the error $\|Lx - b\|$ is as small as possible.

pseudo inverse matrix 先计算出一个 x ba，然后让 x ba 接近 x，

不断的更新 pseudo inverse matrix

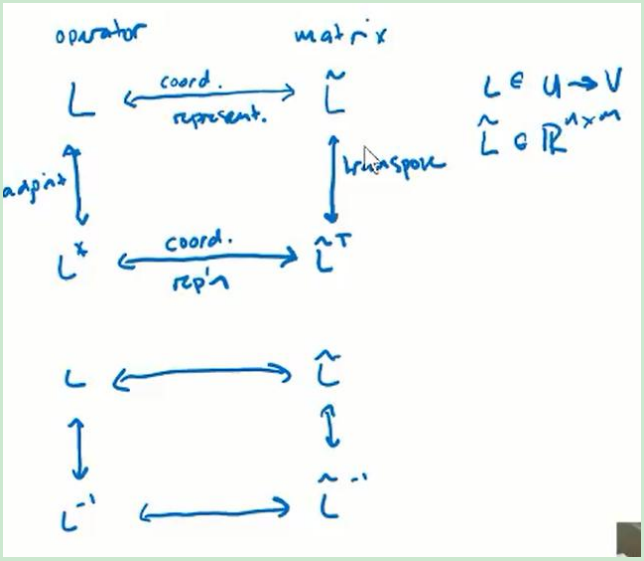
inverse 可以 on operator ，也可以 on matrice

We can define an inverse operation on matrices as well: if A and A^{-1} are matrices that satisfy

$$AA^{-1}x = x \qquad A^{-1}Ay = y$$

for all vectors $x, y \in \mathbb{R}^n$, then we say that A^{-1} is the inverse of A.

linear operation 小结



linear operator 都有对应的 coordinate matrix tiled

adjoint operator 是操作 linear operator(x, y 域变换)

adjoint operator 也有对应的 **coordinate matrix** tiled, 是将 linear operator matrix transpose

inverse operator 是操作 linear operator(x, y 域变换)

inverse operator 也有对应的 coordinate **matrix** tiled,是对 linear operator matrix 的 inverse

matrix pattern

1 diagonal matrix 对角线不为 0, 其他全部 0

2 low triangular 左下角不为 0 upper….其他全 0

3 identity matrix 对角线全为 1 其他全 o

4 genaral sparce matrix

A matrix with lots of zero entries but no particular pattern is called *sparse*. Typically we need a high fraction of zero entries to make it worthwhile to think of a matrix as sparse: say at least 80% zeros. It's common, though, to work with matrices that are even sparser: say, 99.9% zero entries.

一个矩阵上面的特性如果都满足，就更容易操作(操作时不需要考虑每一个元素)

If a matrix follows multiple patterns, it can be even more convenient to work with: e.g., a sparse lower-triangular matrix is more convenient than a matrix that is just sparse or a matrix that is just lower-triangular.

Orthogonal matrix and vector

orthogonal vector:

In an inner product space, two vectors x and y are called *orthogonal* if $\langle x, y\rangle = 0$. Geometrically, orthogonal vectors form a right angle with one another.

orthogonal **matrix: column vector** is orthogonal to 其他 colunm vector(两两垂直)

An orthogonal *matrix* is one where $A^T A = D$ with D diagonal. In an orthogonal matrix, pairs of columns are orthogonal vectors: $\langle A_{:,i}, A_{:,j}\rangle = 0$ for $i \neq j$. If A is orthogonal and square, then so is A^T.

必须把转置放前面，才能保证 A 的列相乘

$A^T A = D$ 是对角矩阵(其他元素 0)，只有跟自己 column 乘才不为 0

如果 A 是 square 方阵，且 orthogonal，A transpose 也是！！

如果 D 是 identity matix

If $A^T A = I$ where I is the identity matrix, we say that A is *orthonormal*. An orthonormal matrix is orthogonal. In addition, each column has unit length: $\langle A_{:,i}, A_{:,i}\rangle = 1$. If A is orthonormal and square, then so is A^T. It's particularly easy to compute the inverse of a square orthonormal matrix: we have $A^{-1} = A^T$.

A orthonormal matrix，每个 column vector 都是 unit length，所有平方和是 1

A inverse 就是 A transpose

orthogonal 使得我们的矩阵更优美，更多的形状，更容易计算！

square orthonormal matrix ： inverse=transpose

orthonormal matrix 可以 rotate 和 flip transformation 向量

Positive(semi) definite operator and matrix

A linear operator $A \in U \to U$ is called *positive semidefinite* or PSD if $\langle x, Ax \rangle \geq 0$ for all vectors x. The operator A is called *positive definite* or PD if the inequality is strict: $\langle x, Ax \rangle > 0$ for all vectors $x \neq 0$. We also apply the terms PSD and PD to the corresponding matrix $\tilde{A}$.

a operator A（coordinate matrix）对于所有 vector x,使得 x 和 Ax 的内积大于或等于 0 PSD PD

To test PD-ness or PSD-ness in the non-self-adjoint or asymmetric case, we can use the following fact: a non-self-adjoint linear operator A is PSD or PD iff $A + A^*$ is PSD or PD. Similarly, an asymmetric matrix $\tilde{A}$ is PSD or PD iff $\tilde{A} + \tilde{A}^T$ is. (Exercise: prove this fact.)

exercise:

linear equation

1 Factorization

A=BC,分解以后，如果 B，C 具备了前面特殊 pattern 矩阵，就便于计算了

Two of the most common and useful factorizations are the *LU decomposition* and the *singular value decomposition*. Both of these factorizations always exist, and there are efficient algorithms to find them. (You can find implementations of these algorithms in Matlab or NumPy.)

LU 和 SVD 最常用的两个分解

LU

In the LU decomposition we write

$$A = LU$$

where L is lower triangular and U is upper triangular. (Here and in the rest of this set of lecture notes, we're assuming that $A \in \mathbb{R}^{n \times n}$ is a square matrix. So, its factors are also square, and have the same dimensions as A.)

A 是方阵 LU 也是方阵

可以 padding 0 to 不是方阵，再进行分解

2 SVD

In the singular value decomposition or SVD we write

$$A = USV^T$$

where U and V are orthonormal and S is diagonal with nonnegative elements. The columns of U and V are called the left and right singular vectors of A, while the diagonal elements of S are called singular values.

两边是正交标准矩阵，中间是对角矩阵(非负)

singular vector 和 values

如果 A 是特殊 pattern 矩阵：

If A is symmetric (that is, $A = A^T$) then the LU factorization can be simplified to

$$A = LDL^T$$

where L is lower triangular and D is diagonal. If in addition A is PSD, the matrix D is unnecessary: we can factor $A = LL^T$.

A 如果是对称矩阵，如果还正定：

也就是 $U = DL^T$

如果 A 是 PSD $A = LL^T$ D 是 identity

For symmetric A, the SVD can be simplified by taking $V = U\Sigma$, where Σ is a diagonal matrix of signs: that is, each diagonal element of Σ is ± 1. That leads to the factorization

$$A = US\Sigma U^T.$$

If in addition A is PSD, the matrix Σ is unnecessary: we can factor $A = USU^T$.

如果是对称矩阵，SVD sigma 是符号对角阵

sigma 是对角符号矩阵(元素只是-1 或 1 代表正负号)

如果 A 是 PSD 因为符号都为正 sigma 是 identity 矩阵，$A = USU^T$

Factorizations are useful for lots of tasks, but one example is that they can help us test whether a symmetric matrix is PSD or PD: in the LU factorization $A = LDL^T$, the matrix A is PD iff the diagonal elements of D are strictly positive, while A is PSD if the diagonal elements are nonnegative. Or, in the SVD $A = US\Sigma U^T$, the matrix A is PD (PSD) if all of the diagonal elements of $S\Sigma$ are strictly positive (nonnegative).

检验是否 PSD PD

solve linear equation(矩阵分解)

1 不应该求 inverse 来解决

Let $A \in U \to V$ be an invertible linear operator and $b \in V$ be a vector. It's often useful to solve the linear equation

$$Ax = b \quad x \in U.$$

Mathematically, the solution is simple to write: we have $x = A^{-1}b$. But, this is typically a bad way to solve linear equations in practice!

1 计算量大　2 计算误差太大

Why? It can be expensive to compute A^{-1}, since A^{-1} will typically not have the same pattern of nonzeros that A does. And, even if we can afford to compute A^{-1}, doing so can lead to bad roundoff errors. Double precision real numbers have a relative error of about 10^{-15} when we first store them in a computer, and a typical relative error (after we put them through a few computations) of perhaps 10^{-12}. But, it's common to see much larger errors if we try to compute A^{-1} and multiply out $A^{-1}b$. The computed components of $A(A^{-1}b)$ can easily differ from the components of b by a lot more than a factor of 10^{-12} — say by a factor of 10^{-3} or even 10^{+3}.

So, ***don't solve equations with*** $\mathbf{A^{-1}}$ unless you're absolutely sure it won't cause a problem.

用矩阵分解来解

Instead, it's much better to use matrix factorizations to solve linear equations. The most common version of this approach is to use the LU factorization: if $A = LU$ then, multiplying first by L^{-1} and then by U^{-1}, we have

$$x = U^{-1}L^{-1}b.$$

$A = LU$

$LU\mathrm{x} = \mathrm{b}$

$\mathrm{x} = U^{-1}L^{-1}\mathrm{b}$

It's typically efficient and well-conditioned to find L and U. We still don't form L^{-1} or U^{-1}, since doing so would lead to the same problems as before. But, there is an algorithm (called *backsubstitution*) that takes advantage of the triangular structure of L and U to solve

$$Ly = b \quad \text{followed by} \quad Ux = y$$

寻找 U 和 L，更高效，没有 round off

也不能直接求 inverse of U 和 L,而是用 backsubstitution 来求： $LU\mathrm{x} = \mathrm{b}$

把 Ux 看作新的变量 y，Ly=b　$\mathrm{y} = \mathrm{L}^{-1}b = Ux$

先 solve 出 y，再 solve 出 x，就得到解 x 了

2 也可以用 SVD 来解方程

It's typically a bit slower, but also a bit more numerically stable, to solve linear equations by using the SVD: if $A = USV^T$ then

$$x = VS^{-1}U^Tb.$$

更慢，但是 numeral stable!

square orthonormal matrix : inverse=transpose

least square solution

A not invertible，我们要找 pseudoinverse A(近似于 A invertible)

If A is not invertible, we can still find an approximate solution to $Ax = b$ by using the pseudoinverse: we set

$$x = A^\dagger b.$$

If we are lucky and b is in the column span of A, we will have $Ax = b$; if not, x will minimize the norm of the *residual* vector $Ax - b$. The value of x we get this way is sometimes called the *least squares* solution, since we can equivalently say that x minimizes $\|Ax - b\|^2$, the sum of squared elements of the residual vector.

找到 pseudoinverse，进而得到的 x，使得 Ax-b=0 接近

用 svd 做比较好,帮助找到 pseudoinverse A

For a singular matrix A, it's still possible (and numerically stable) to form the singular value decomposition $A = USV^T$. U and V will be invertible, since they're orthonormal. But, one or more diagonal elements of S will be zero, so S will not be invertible.

奇异矩阵的 SVD 分解， S 不可逆！！

如何找到 pseudoinverse A？只需找到 pseudoinverse S

The SVD can help us find the pseudoinverse of A. The pseudoinverse of a diagonal matrix S is easy to compute: $S^\dagger$ is a diagonal matrix with S_{ii}^{-1} on the diagonal where possible, and zero where $S_{ii} = 0$. (It's not often that we get to set $1/0 = 0$.) Using this fact, it's easy to compute the pseudoinverse of a A as well: it is $VS^\dagger U^T$.

先 SVD 分解，然后得到 pseudoinverse S，再得到 pseudoinverse A

将 S 对角线的不为 0 元素**取倒数**，为 0 的元素设为 0，组成的对角矩阵就是 pseudoinverse S

If our matrix A is close to singular, then some of the diagonal elements of S will be close to zero. In this case (say, when some diagonal elements of S are 10^{10} times smaller than others), it can be worth pretending that these elements are exactly zero: we won't be able to get a numerically accurate solution to our original linear equation, but our residual $Ax - b$ can be smaller if we give up on the problematic singular values and vectors.

如果 A 近似 singular，则 S 的对角元素几乎接近 0，**此时我们可以近似将 S 对角线全部设为 0**

Matrix differentials

导数

导数及性质

1. Scalar derivatives — notation

We'll start with a summary/reference for scalar derivatives. First, notation:

- For a function $f \in \mathbb{R} \to \mathbb{R}$, we write $f' \in \mathbb{R} \to \mathbb{R}$ for its derivative with respect to its argument. If the argument is called x, we also can write $\frac{df}{dx}$, df/dx, or $\frac{d}{dx}f$.
- Second and higher derivatives are f'', f''', ... or $\frac{d^2f}{dt^2}$, $\frac{d^3f}{dt^3}$, ...
- If a function depends on more than one variable (say x and y), we write $\frac{\partial}{\partial x}f$ or $\frac{\partial}{\partial y}f$ to indicate the derivative with respect to one of the variables while holding the others constant. This is called *partial* differentiation.
- Evaluation: given a function $f(x)$, we write $f(x)\big|_{\hat{x}}$ or $f(x)\big|_{x=\hat{x}}$ as a synonym for $f(\hat{x})$. This notation is useful when we want to take a bunch of derivatives and then evaluate the result at a point — the usual $f(x)$ notation doesn't support x being both a variable (for differentiation) and a constant (for function evaluation) at the same time.

导数 linear operator 性质

2. Scalar derivatives — identities

Some of the most common identities for working with scalar derivatives:

- Differentiation and partial differentiation are linear operators: for example, $(af + bg)' = af' + bg'$.
- Chain rule: if we want $\frac{d}{dx}f(g(x))$, then we can write

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}.$$

  As a mnemonic we can "cancel the dg." (But since $\frac{df}{dg}$ is not really division, this is just a mnemonic.) Another way to write the chain rule is

$$\frac{d}{dx}f(g(x)) = f'(g(x))\,g'(x).$$

- Product rule: $(fg)' = fg' + f'g$.

3. Common functions

Derivatives of some common functions are below. We are treating each expression as a function of x; all other symbols represent constants.

- The derivative of a constant is zero.
- The derivative of a monomial x^k is kx^{k-1}. This identity works for negative and fractional values of k as well; the only exception is $k = 0$, in which case x^k is a constant and its derivative is zero.
- The derivative of $\sin(x)$ is $\cos(x)$; the derivative of $\cos(x)$ is $-\sin(x)$.
- The derivative of e^{ax} is ae^{ax}. If we're using some other base, as in b^x, we rewrite $b^x = e^{x \ln b}$ before using the above identity.
- The derivative of $\ln(x)$ is x^{-1}. With another base we rewrite $\log_b(x) = \ln(x)/\ln(b)$ before using the above identity.

Vector derivative

两种函数：参数是向量 或者 返回向量值，他们的求导方法

4. Vector derivatives

It's also useful to think about functions that return vectors or take vectors as arguments. For a vector-valued function $f \in \mathbb{R} \to \mathbb{R}^n$,

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix},$$

we collect all of the partial derivatives of the component functions into a vector the same shape as $f(x)$:

$$\frac{df}{dx} = \begin{pmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \\ \vdots \\ \frac{\partial f_n}{\partial x} \end{pmatrix}.$$

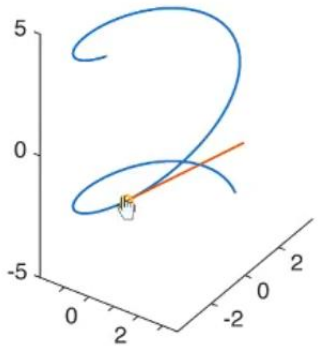
1 返回向量的函数

f map R to Rn

其实本质是有多个 function 组成,分别返回值,同一个 scalar input

We can think of this sort of function as representing a curve in $\mathbb{R}^n$. The derivative $\frac{df}{dx}$ represents a tangent vector to this curve: the instantaneous velocity of a point moving along the curve as the argument x changes. The length of the tangent vector tells us the speed of the point, and the components tell us the slope of the vector in each direction.

Here's an example of a function in $\mathbb{R} \to \mathbb{R}^3$, together with its tangent vector computed at the point marked with a circle:



derivative 代表切线 of curve

x 是速度，derivative 是加速度

因为 y 是三维向量，所以画在三维空间，每个 y 值都只是三维空间的

一个点，连乘一条曲线

derivative 是 vector 刚好和曲线相切！

2 向量作为参数的函数 多元函数

For a multiple-argument function $f \in \mathbb{R}^n \to \mathbb{R}$, we collect all of the arguments into a column vector

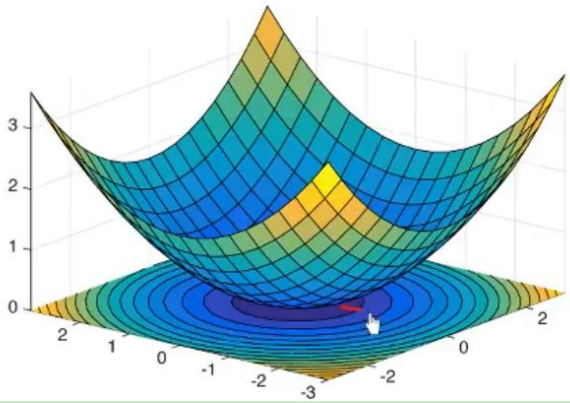
$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Then we write df/dx for the row vector of partial derivatives:

$$\frac{df}{dx} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n} \right).$$

We can think of this type of function as representing a surface in $\mathbb{R}^{n+1}$: the argument x varies across $\mathbb{R}^n$ while $f(x)$ determines the height of the function. The tangent vector tells us the direction of steepest increase of the function.

Here's an example of a function in $\mathbb{R}^2 \to \mathbb{R}$, together with its tangent vector computed at the point marked in red:



空间中的曲面

Chain rule for vector

With the above notation, the chain rule looks just like it did for scalar-to-scalar functions. Suppose f takes multiple arguments and g returns multiple values. We can collect the multiple values into a

f vector 参数 g 返回 vector f(g(x))

returns multiple values. We can collect the multiple values into a vector and write (as before) $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$. But now, $\frac{df}{dg}$ is a row vector and $\frac{dg}{dx}$ is a column vector, so when we multiply them we get their dot product. A more familiar way to write this rule may be

$$\frac{df}{dx} = \frac{\partial f}{\partial g_1}\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\frac{\partial g_2}{\partial x} + \ldots + \frac{\partial f}{\partial g_n}\frac{\partial g_n}{\partial x}.$$

还可以写成原来的形式，只不过现在的 factor 是 vector，是向量的元素相乘(dot product)

First order taylor series

7. First-order Taylor series

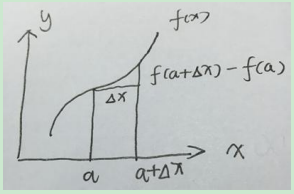
We can approximate any nonlinear function $y = f(x)$ around a point $(\hat{x}, \hat{y})$ with a linear function called the *first-order Taylor series*:

$$y \approx \hat{y} + A(x - \hat{x}) \quad y, \hat{y} \in \mathbb{R}^m;\ x, \hat{x} \in \mathbb{R}^n;\ A \in \mathbb{R}^{m \times n}$$

Here the matrix A is called the *Jacobian* of f at $\hat{x}$, or the first derivative of y with respect to x evaluated at $\hat{x}$:

$$A_{ij} = \frac{\partial y_i}{\partial x_j}\Big|_{\hat{x}}$$

推导：



$$\frac{f(a+\Delta x) - f(a)}{\Delta x} = \tan\alpha$$

又有$f'(a) = \tan\alpha$ (3)

将式(3)代入式(2)就得出了线性近似，也就是泰勒的一阶展开：

$$f(a + \Delta x) = f(a) + f'(a)\Delta x$$

也就是 a 附近任意一点的 y 值都可以近似用 a 的函数值和导数来表示！！

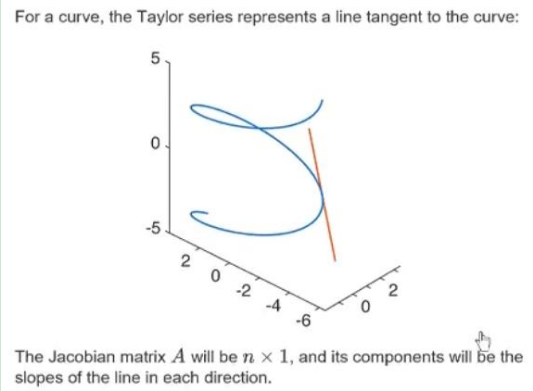
用 linear function 近似**任何 non-linear function 的任意一点的**值，first-order Taylor 展开

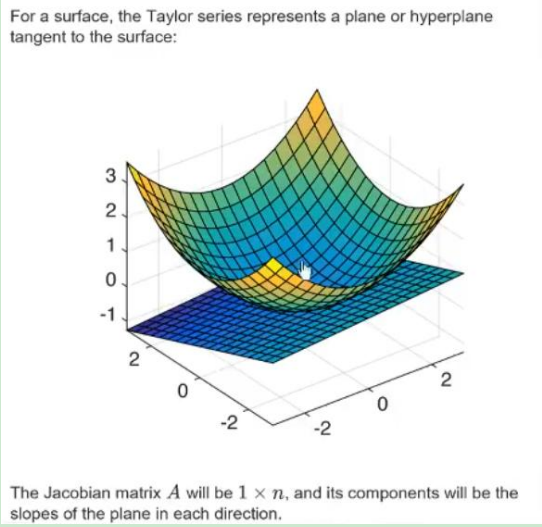
假设 nonlinear function 有两个点$(\hat{x}, \hat{y})$，可以用右边来近似$(\hat{x}, \hat{y})$邻域周围点(x)带入近似 f(x)，**只是对这一固定点附近邻域 x 的近似**

上面 x 和 y 都向量，这里是 general case，我们可以将 m 或 n 设置为 1，也符合上面的定理

因为 x 是向量,y 也是向量,因此 $A_{ij} = \frac{\partial y_i}{\partial x_j}\Big|_{\dot{x}}$ 偏导数是矩阵:A(jacobian)

其实就是是一阶导数 derivative!





differentials 微分

矩阵微分最好的书

1. Using identities

Derivatives are important, so people have proved more identites about them than we could possibly cover here. Below are a few of the most useful.

If you have to differentiate a complex expression, tables or books of identities can be very helpful. Probably the best known book of identities is The Matrix Cookbook.

A word of warning, though: beware of identities you don't understand. Subtle differences in notation or definitions can (and frequently do) lead to errors when using such identities.

微分的定义

其实就是 derivative 给所有变量加个 d,函数的微分定义：

df(x)=f(x)’ dx

1 复合函数微分：df(g(x))=f(u)’ d g(x)=…dx

2 多元函数全微分：z=f(x,y) $dz=\frac{\partial z}{\partial x}dx+\frac{\partial z}{\partial y}dy$ 也就是分别求出每个偏导数加上 d 再相加

3. With two or more variables

If we have multiple variables $x, y, \ldots$, then a differential represents a function $df(x, y, \ldots; dx, dy, \ldots)$. This function has two sets of arguments: it is linear in $dx, dy, \ldots$, but it can be nonlinear in $x, y, \ldots$. For example, if

$$f = \cos(x + y)$$

then

$$df = \sin(x + y)\,(dx + dy)\,.$$

This expression represents the Taylor series

$$f(x, y) - f(\hat{x}, \hat{y}) \approx \sin(\hat{x} + \hat{y})\,((x - \hat{x}) + (y - \hat{y}))\,.$$

多元函数的泰勒展开式

3 微分 d 支持线性：d(ax+by)=adx+bdy

We can also write the first-order Taylor series using a slightly different notation:

$$dy = A\,dx$$

Here dy is short for $y - \hat{y}$, and dx is short for $x - \hat{x}$. The quantities dy and dx are called *differentials*, and we can read the equation above as "the change in y is A times the change in x."

Taylor 的另一种形式，其实就是导数的形式

change of y= A* change of x

Our new notation hasn't changed the meaning at all (it's still a first-order Taylor series), but it is suggestive of a derivative: if we "divide through by dx" we get

$$dy/dx = A\,,$$

which states that A is the first derivative of y with respect to x. (It's only suggestive since dy/dx isn't really division.) The difference between the new notation and the old is that the new notation gives the symbols dy and dx their own individual meanings, letting us separate them and manipulate each one on its own.

更接近 derivative 的形式

A 是 first derivative

但实际上不能直接 dy/dx 两个都是不同的向量维度

Dependence of A on x

实际上 A 是 depend on x(我们选取的固定点)，也就是一阶导数在该点的函数值，不同的 x 不同的 A，更准确的写法如下：

In the expression

$$(y - \hat{y}) \approx A\,(x - \hat{x})\,,$$

the matrix A implicitly depends on the point where we evaluate the Jacobian. This dependence will matter later when we start taking multiple derivatives; so, we will be more precise by writing

$$(y - \hat{y}) \approx A(\hat{x})\,(x - \hat{x})\,.$$

Similarly, instead of $dy = A\,dx$ we write

$$dy = A(x)\,dx\,.$$

用微分的形式比求导的形式更好

Total partial derivative

因为就不区分 partial derivative ∂和 total derivative d 了，不会 apparently 写∂和 d

In our new notation, we won't explicitly distinguish partial derivatives ($\partial/\partial x$) from total derivatives (d/dx). Instead we'll use the following convention: we're considering simultaneous changes to any variable that appears in a differential $dx, dy, \ldots$. Any other variables (those that don't appear in differentials) are considered to be held constant.

也就是只要出现 **d 变量，就代表这些变量在变，其他变量不变**

So for example, in the expression

$$dL = f(x, y, w)dx + g(x, y, w)dy$$

we are relating changes in L to changes in x and y while holding w constant. If x and y are both functions of another variable t, then if

上面的就是对 xy 求 partial derivative，也用 d 表示，w 固定

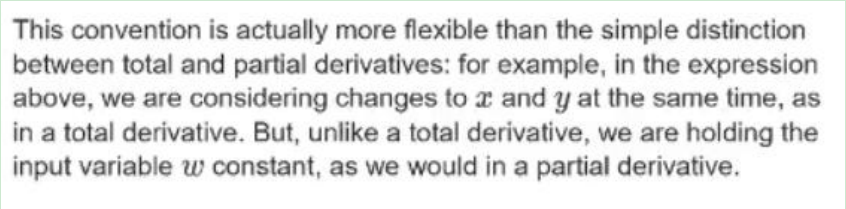
假设下 x，y 又是关于 t 自变量的函数，w 跟 t 无关

constant. If x and y are both functions of another variable t, then if we "divide through" by dt we get

$$\frac{dL}{dt} = f(x, y, w)\frac{dx}{dt} + g(x, y, w)\frac{dy}{dt}$$

which is the correct expression for the total derivative of L with respect to t (under the assumption that w does not depend on t.)

同时除以 dt，左边得到 t 的 total derivative，右边其实也是


This convention is actually more flexible than the simple distinction between total and partial derivatives: for example, in the expression above, we are considering changes to x and y at the same time, as in a total derivative. But, unlike a total derivative, we are holding the input variable w constant, as we would in a partial derivative.

矩阵微分的例子：

If we have

$$y = f(x) = \|Ax - b\|^2/2$$

(where x, b are vectors, A is a matrix, and y is a scalar) then

$$\begin{aligned} y &= \sum_i \left(\sum_j A_{ij}x_j - b_i\right)\left(\sum_k A_{ik}x_k - b_i\right)/2 \\ \partial y/\partial x_\ell &= \sum_i \tfrac{A_{i\ell}}{2}\left(\sum_k A_{ik}x_k - b_i\right) + \left(\sum_j A_{ij}x_j - b_i\right)\tfrac{A_{i\ell}}{2} \\ &= \sum_i A_{i\ell}\left(\sum_j A_{ij}x_j - b_i\right) \\ dy/dx &= (Ax - b)^T A \end{aligned}$$

中间两个括号相乘是向量的内积(两个括号的式子是一样的)

如果要展开向量成元素，再分别求偏导数 太麻烦！！

(The first line follows by expanding the norm and the matrix-vector product as sums. The second line follows from the product rule. The third line collapses terms. And in the last line, we recognize that the sums can be written as matrix-vector products.)

用 matrix differential 方法：

We can write the last line in our new notation as

$$dy = (Ax - b)^T A\, dx$$

Later we'll see how to derive this expression directly, without having to go through the work of expanding everything into sums over individual components as above.

working with differentials

Above we saw how to write a first-order Taylor series in matrix differential notation. That's fine, but not very useful unless we can work with expressions directly in this notation. As always with derivatives, the main tools for working with differentials are linearity, the chain rule, and the product rule; we'll take a look at each in turn below. We'll see that some derivatives become *much* easier to find using the new notation.

linearity 和 chain rule ，product rule 是三个求微分的法宝

Type checking

make sure all the **types of variable**, the function, intermediate thing

involving differentials. We will start from an equation that looks like

$$f = \dots x \dots$$

which describes how a variable $f \in V$ depends on a variable $x \in U$. Therefore, the right-hand side of the equation represents a function in $U \to V$.

The differential will then look like

$$df = \dots x \dots dx \dots$$

Here $x \in U$, $dx \in U$, and $df \in V$. So, the right-hand side of this equation represents a function in $U \times U \to V$: given $x \in U$ and $dx \in U$, we can compute $df \in V$. This function is always linear in the dx argument, but can be (and often is) nonlinear in x.

x 和 dx 维度一样 ， df 可能不一样

This is an abuse of notation: we are using the same name both for the function $f \in U \to V$ and the output of that function $f \in V$. However, this kind of abuse of notation is very common (not just in this class), so it's worth watching out for.

f 既代表**函数**，又代表**函数** output

Similarly, we can write $df(x; dx)$ to emphasize that df depends on both x and dx. (The semicolon separates the nonlinear argument from the linear one.) This is an abuse of notation for the same reason: we are using df for both a function of type $U \times U \to V$ and the function's output of type V.

df：代表微分的 output 结果

加了分号就代表已经执行，得到最终的函数形式：是已经求好微分了

df，参数是 x 和 dx，也就是该微分函数的自变量是 x dx，加;的目的是为了突出这个函数的参数或自变量是谁(函数形式已经出现)

df 既是 函数的形式，又是微分的 output 结果

Another possible notation confusion: the expression $df(x)$ means $d(f(x))$, or "the differential of the expression $f(x)$." By contrast, $df(x;dx)$ means $(df)(x;dx)$; here df refers to the function we get by taking the differential. We can always tell which way the parentheses are intended to go by looking at the number of arguments: in $df(x)$ there is not (yet) an explicit dx argument, so it means we haven't taken the differential yet.

df(x)仅仅代表对 f(x)求微分，还没有求

df(x;dx) 是(df)(x:dx) 加了分号就代表已经执行,得到最终的函数形式，肯定包含 dx 了

prime:

Another common notation for derivatives is $f'(x)$. We can relate this notation to the differential $df(x;dx)$: for each $x, dx \in U$,

$$f'(x)\,dx = df(x;dx)\,.$$

That is, $f'(x) \in U \to V$ refers to the (linear) function that maps dx to df; we get this function by fixing the x argument of $df \in U \times U \to V$.

A similar convention is common whenever we work with differentials: when reading an expression like

$$df = A(x)\,dx + B(x,y)\,dy$$

that contains both variables $x, y, \ldots$ and differentials $dx, dy, \ldots$, the convention is that we consider $x, y, \ldots$ fixed, and interpret all expressions as functions of $dx, dy, \ldots$. So for example, in the above expression, we are considering $B(x,y)$ to represent a linear function (a matrix) for each fixed x, y. We apply this function to (multiply this matrix by) the vector dy.

f’(x)的 x 是 fixed point　dx 的 x 是变量 x-fixed point

Differential and linearity(穿透)

linearity

Because taking the derivative is a linear operator, differentials pass through addition and multiplication-by-constant: e.g., if $X, Y, A, B \in \mathbb{R}^{n \times n}$, with A, B constant, then

$$d(AX + BY) = A\,dX + B\,dY\,.$$

The same works for sums as well: for functions $f_i \in (U \to V)$ on vector spaces U, V, we have

$$d\sum_i f_i(x) = \sum_i df_i(x)\,.$$

As usual, $dA = 0$ if A is constant.

例子：

$\sum_x p(x) f(g(\theta))$ 对 theta 求微分

微分可以穿透任何 linear 部分：$\mathrm{d}\sum_x p(x) f(g(\theta)) = \sum_x p(x)\mathrm{d}f(g(\theta))$

但只穿透到关于参数的最外层函数(包括复合函数)

不一定非要穿到 theta，可以穿到中间如 theta 的函数

chain rule 复合函数

Suppose $f \in (V \to W)$ and $g \in (U \to V)$ for vector spaces U, V, W. Then

$$d(f(g(x))) = f'(g(x))\,dg(x)\,.$$

再穿一层，但是要乘以外层函数的导数

The above expression uses several of the shorthands we've defined above, so let's unpack it to understand it better. For this purpose, it helps to write $u = g(x)$, so that our expression becomes

$$d(f(u)) = f'(u)\,du\,.$$

这是最好的解释，函数的微分=函数的导数乘以 dx

Rewriting f' according to the definition from earlier, we have

$$d(f(u)) = df(u; du)\,.$$

$df(u) = f'(u)du$ 为了突出已经得到微分的式子，并且**两个参数 u 和 d**

Substituting back in $u = g(x)$, we get

$$d(f(g(x)) = df(g(x); d(g(x)))\,.$$

We can expand $d(g(x)) = dg(x; dx)$ to get

$$d(f(g(x)) = df(g(x); dg(x; dx))\,,$$

最终的参数还 x 和 dx

type checking

The expression on the right-hand side of the above equation defines a function that takes arguments $x \in U$ and $dx \in U$, and produces an output in W. We can type-check to be sure: we are composing $df \in V \times V \to W$ with $g \in U \to V$ and $dg \in U \times U \to V$, matching on type V. And, we are using x and dx to provide the inputs to g and dg, matching on type U.

参数还 x 和 dx 都属于 V

结果是 W 因为 f 输出的维度是 W

product rule

两个函数(同一参数)的乘积的形式，对参数求导

Suppose that $f(x) = g(x) \cdot h(x)$, where $\cdot$ stands for matrix multiplication or composition of linear operators. That is, for each fixed $x \in X$, we have we have $g(x) \in V \to W$ and $h(x) \in U \to V$, so $f(x) \in U \to W$.

Then

$$df = dg(x) \cdot h(x) + g(x) \cdot dh(x)\,.$$

As usual, we can easily extend to three (or more) terms in the product:

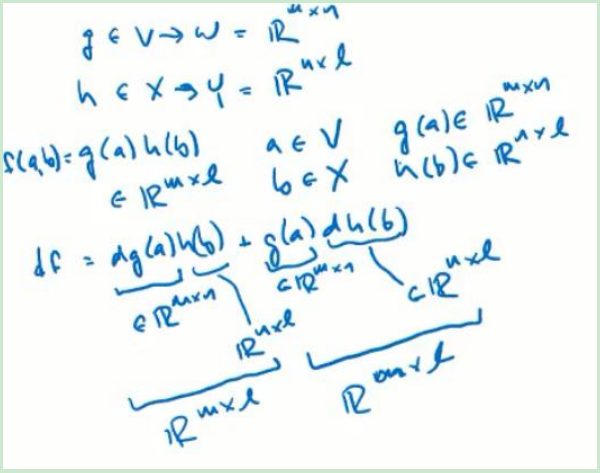
$$\begin{aligned} d(f \cdot g \cdot h) &= df \cdot g \cdot h + f \cdot d(g \cdot h) \\ &= df \cdot g \cdot h + f \cdot (dg \cdot h + g \cdot dh) \\ &= df \cdot g \cdot h + f \cdot dg \cdot h + f \cdot g \cdot dh\,. \end{aligned}$$

每个 factor 函数都单独求一次微分

注意上面函数相乘的顺序不能变，如果涉及到矩阵和向量的话

$dg(x) \cdot h(x) + g(x) \cdot dh(x)$. 都要按照原函数乘积的顺序！！

type checking：



扩展：

8. Multiplication-like operators

The product rule actually works for any multiplication-like operator — meaning any operator that is linear in each of its arguments separately. (Such an operator is called *bilinear*, or *multilinear* if it has more than two arguments.)

Matrix multiplication, and its cousin composition of linear operators, are common examples, but there are lots of others:

- Ordinary multiplication of scalars
- Inner product of vectors (including dot product)
- Cross product of vectors
- Componentwise product of vectors, matrices, or tensors (also called Hadamard product)
- Componentwise product with broadcasting (as we used for probability tables)
- And lots of others that we haven't worked with yet: tensor product, Kronecker product, Khatri-Rao product, …

So for example, if $f(x) \circ g(x)$ represents the componentwise product of two vectors, then

$$d(f(x) \circ g(x)) = df(x) \circ g(x) + f(x) \circ dg(x).$$

The multiplication-like operator doesn't have to be commutative — for example, matrix multiplication is not. In this case the order of the factors matters: while we often write $(fg)' = fg' + gf'$ for the product of scalars (using commutativity to reorder the factors), this sort of reordering would not work for matrix multiplication.

只是要注意，矩阵的话不支持交换，所以写的顺序很重要！

least squares(带正则化项)求微分

example：范数

都是要保证和原来的数的单位一样

1-范数：‖x‖1=|x1|+|x2|+...+|xn|

2-范数：‖x‖2=(|x1|^2+|x2|^2+...+|xn|^2)^1/2

$$y = \|Ax - b\|^2/2 = (Ax - b)^T(Ax - b)/2$$

|| || 是 12 norm 也是 squared error

12 norm 的平方(其实就是向量各个元素平方和)=向量的内积

turn vector norm squre to vector T * vector

9. Example: squared error

Returning to our example of squared error

$$y = \|Ax - b\|^2/2 = (Ax - b)^T(Ax - b)/2$$

we can write

$$dy = d(Ax - b)^T(Ax - b)/2 + (Ax - b)^T d(Ax - b)/2$$

第 1 步：微分 chain rule

第 2 步：乘以矩阵 A 和 transpose 都是 linear operator，transpose 也可以当做一个矩阵 B 相乘

$$d(Ax-b)^T = dB(Ax-b) = Bd(Ax-b)$$

而微分和求导又支持 linear operator，可以

$$Bd(Ax-b) = B(Adx)$$

$= \frac{1}{2}(A\Delta x)^T(Ax-b) + (Ax-b)^T A\Delta x$

因为 y 是 scalar 因此两个相加的部分也都是 scalar

transpose scalar 还是 scalar 本身

因此我们让第二个部分 transpose 得 $dy = \min \|Ax-b\|$，刚好和第一部分一样

$dy = (Ax-b)^T Adx$

least squares

使得 $y = \|Ax-b\|^2/2$ 最小，也就让导数为 0

find the best possible approximation to b within the range of A 也就是需要让 dy=0 就是导数等于 0

我们要求 x，就不能直接展开，再利用 scalar 的 transpose 性质！

$dy = (Ax-b)^T Adx$

$A^T(Ax-b) = 0$

$A^TAx = A^Tb$ normal equalition 用我们之前的方法 LU 分解

The last line above is a system of linear equations, called the normal equations. We can solve the normal equations using the techniques that we discussed earlier, so long as A^TA is nonsingular. For example, we can use LU factorization and backsubstitution. (More precisely, we would use Cholesky factorization, which is the special case of LU factorization for a symmetric psd matrix.)

regularized least square

如果A^TA是 singular?

regularize the solution

so long as the matrix A^TA is nonsingular. But what if A^TA is singular?

A singular system of equations can have either no solutions or infinitely many solutions, depending on whether the right-hand-side vector is in the range of the coefficient matrix. In our case, A^Tb is in the range of A^TA by inspection (since the range of A^TA is the same as the range of A^T); so, there are infinitely many solutions.

A common way to proceed in this case is to add another goal to the problem: instead of just trying to minimize squared error, we also try to keep the coefficient vector x close to zero. That is, instead of minimizing $y = \frac{1}{2}\|Ax - b\|^2$, we minimize

$$z = \frac{1}{2}\|Ax - b\|^2 + \frac{1}{2}\lambda\|x\|^2$$

where $\lambda \geq 0$ is a parameter. If $\lambda = 0$ we get back the original least squares problem. As λ increases, we put more weight on keeping x small, called *regularizing* the solution.

equation system 可能会无解或者无数解

看 右边的向量是否 in the **range of 系数矩阵**

有无数解，类似于我们的 weight 可以同时线性增加

我们只取最小接近 0 的解（regularize the solution）

重新求微分

一 先整理 loss fucntion z

2. Regularized least squares: differential

Just as with ordinary least squares, we can solve the regularized least squares problem by setting the differential to zero:

$$\begin{aligned} z &= \tfrac{1}{2}(Ax - b)^T(Ax - b) + \tfrac{1}{2}\lambda x^T x \\ &= \tfrac{1}{2}(x^T A^T Ax - 2b^T Ax + b^T b + \lambda x^T x) \end{aligned}$$

1 turn **vector norm squre** to **vector T * vector**

2 因为合是 scalar，add 两部分都是 scale，先不管这个，先展开 z

$$= ((Ax)^T - b^T)(Ax - b) + \lambda x^T x$$
$$= (Ax)^T Ax - (Ax)^T b - b^T Ax + b^T b + \lambda x^T x$$
$$= x^T A^T Ax - x^T A^T b - b^T Ax + b^T b + \lambda x^T x$$

因为合是 scalar，因此每一项都是 scalar

$$x^T A^T b = (x^T A^T b)^T = b^T Ax$$

最终

$$z = x^T A^T Ax - 2b^T Ax + b^T b + \lambda x^T x$$
$$d(x^T A^T Ax) = d[(x^T) \cdot (A^T Ax)] = dx^T A^T Ax + x^T A^T Adx$$

二 再求微分对 x

product rule

$$d(x^T A^T Ax) = d[(x^T) \cdot (A^T Ax)] = dx^T A^T Ax + x^T A^T Adx$$

$$\begin{aligned} z &= \tfrac{1}{2}(Ax - b)^T (Ax - b) + \tfrac{1}{2}\lambda x^T x \\ &= \tfrac{1}{2}(x^T A^T Ax - 2b^T Ax + b^T b + \lambda x^T x) \\ dz &= \tfrac{1}{2}(dx^T A^T Ax + x^T A^T A\,dx - 2b^T A\,dx + \lambda dx^T x + \lambda x^T dx \\ &= x^T A^T A\,dx - b^T A\,dx + \lambda x^T dx \\ 0 &= A^T Ax - A^T b + \lambda x \end{aligned}$$

Using $\lambda x = \lambda I x$, where I is the identity matrix, we get

$$(A^T A + \lambda I)x = A^T b$$

As before, we can solve this system of equations by techniques like Cholesky factorization. But now, as long as $\lambda > 0$, the system is guaranteed to be nonsingular.

Gram matrix formation

A common way for $A^T A$ to be singular is if A is wider than it is tall: $A \in \mathbb{R}^{m \times n}$ with $m < n$. If n is large enough, it might be computationally difficult to solve for x: for example, it's expensive to solve a $10^6 \times 10^6$ system of equations.

对于普通方程 Ax，如果矩阵 A m<n 代表 feature 多，example 少，

奇异矩阵

直接解上面 $(A^TA-\lambda I)x=A^Tb$ **比较难** A^TA **是 n*n** 太大

In this case, we can save time by solving for $u = Ax$ instead of for x itself. This strategy makes sense because u is an even simpler representation of the answer we're looking for: x lets us express the best approximation to b within the range of A, but u is itself this best approximation.

To solve for u, multiply both sides of the regularized normal equations by A:

$$\begin{aligned} A(A^TA+\lambda I)x &= AA^Tb \\ AA^TAx+\lambda Ax &= AA^Tb \\ (AA^T+\lambda I)Ax &= AA^Tb \\ (AA^T+\lambda I)u &= AA^Tb \end{aligned}$$

This system is called the Gram matrix formulation of the (regularized) normal equations; $G = AA^T$ is called the Gram matrix. Again, we can find u via techniques like Cholesky factorization. But now the system is smaller, only $m \times m$ instead of $n \times n$.

转化成 gram matrix AA^T m*m 方程组会小很多

不直接解 x，解出 u(Ax)

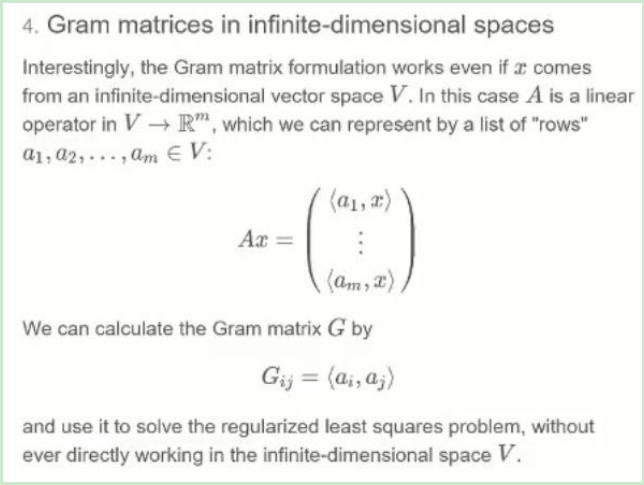
$$z = \frac{1}{2}\|Ax-b\|^2 + \frac{1}{2}\lambda\|x\|^2$$

找 u 来逼近 b

In this case, we can save time by solving for $u = Ax$ instead of for x itself. This strategy makes sense because u is an even simpler representation of the answer we're looking for: x lets us express the best approximation to b within the range of A, but u is itself this best approximation.

solve u 而不是 x，因为 A 是常数,**解 x 和结 Ax 本质一样，目的都一样让 Ax 都逼近 b，最后得出 x**

Gram matrix in infinite-dimensional space



4. Gram matrices in infinite-dimensional spaces

Interestingly, the Gram matrix formulation works even if x comes from an infinite-dimensional vector space V. In this case A is a linear operator in $V \to \mathbb{R}^m$, which we can represent by a list of "rows" $a_1, a_2, \ldots, a_m \in V$:

$$Ax = \begin{pmatrix} \langle a_1, x \rangle \\ \vdots \\ \langle a_m, x \rangle \end{pmatrix}$$

We can calculate the Gram matrix G by

$$G_{ij} = \langle a_i, a_j \rangle$$

and use it to solve the regularized least squares problem, without ever directly working in the infinite-dimensional space V.

即使 x 是 infinite dimension，无限个 feature：

A 是 m*infinite　　x 是 infinite*1　　Ax=m*1

也可以这样解！！！

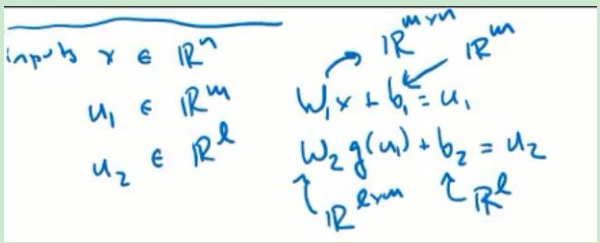
G 还是 AA^T 得到的是 **m*m 维度矩阵** AA^T

同样用 $(AA^T + \lambda I)u = AA^T b$

也就是 feature 再多也可以这样解

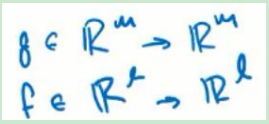
neural network example 微分

input x Rn



x input　　u 是每个 layer 的 units 个数

g 和 f ： componentwise non-linear，对 u 的每个元素进行非线性变化



y 是 Rl　y=f(u2)

最终 map from Rn to Rl，最终 y：

5. Neural networks

We can write a two-layer neural network as

$$y = f(W_2\, g(W_1 x + b_1) + b_2)$$

Here W_1, W_2 are weight matrices, and b_1, b_2 are weight vectors. The functions f and g are componentwise nonlinearities: the ith component of each function will depend only on the ith component of its input vector, via a scalar nonlinearity like

$$f_i(z_i) = \ln(1 + \exp(z_i))$$

(This particular example is a smooth version of a ReLU.)

微分

We can differentiate using the chain rule and linearity. Write $u_1 = W_1 x + b_1$, and $u_2 = W_2\, g(u_1) + b_2$. Then

$$\begin{aligned} dy &= f'(u_2)\, d(W_2\, g(W_1 x + b_1) + b_2) \\ &= f'(u_2)\, (W_2\, dg(W_1 x + b_1) + b_2) \\ &= f'(u_2)\, (W_2\, g'(u_1)\, (W_1 dx + b_1) + b_2) \end{aligned}$$

In the above, f' and g' are functions that produce a diagonal matrix from an input vector: for example, the (i,i) entry of $f'(z)$ is $f_i'(z_i)$.

第一步第三步都是 chain rule，先求外层导数，再求里面层的微分

第二步是用 linearity，写错了 b2 不应该出现

第三步，又用了 linearity！ b1，b2 都应该删掉！

可以 type check 看看维度相乘之后，是否等于最终的维度

linear operator 微分运算诀窍

2. Linear operations

We already know that the differential passes straight through linear operations. Here are a few examples where this property is useful:

- Matrix transpose is linear, as is its cousin, adjoint of a linear operator. So, $d(A^T) = (dA)^T$ for any matrix A.
- Matrix trace is linear, so $d\,\mathrm{tr}\,A = \mathrm{tr}\,dA$. (The trace of a matrix is the sum of its diagonal entries. See below for common uses of trace.
- Reshaping a vector, matrix, or tensor is linear (as in Matlab's reshape() or Python's numpy.reshape() function).
- A special case of the previous example is reshaping a matrix into a vector by stacking its columns. Similarly, so is the inverse operation of reshaping a vector into a matrix by filling in matrix entries from the vector in order. A common notation for these operations is $v = \mathrm{vec}(M)$ (to reshape a matrix into a vector) and $M = \mathrm{unvec}(v)$ (to invert this operation). If the matrix M is not square, then we need to assume that its dimensions are clear from context in order to use unvec. Matlab writes `M(:)` for $\mathrm{vec}(M)$; Python uses `numpy.ravel(M)`.

Trace(linear operator)

所以 Trace 主要是操作方阵，方阵才有对角线！！

The trace of a matrix is defined to be

$$\mathrm{tr}\,M = \sum_i M_{ii}\,.$$

As mentioned above (and as can be seen from this definition), trace is a linear operator. The trace often finds its way into matrix algebra for a number of reasons; perhaps the most common is that

$$\mathrm{tr}\,A^T B = \sum_{ij} A_{ij} B_{ij}\,.$$

That is, $\mathrm{tr}\,A^T B$ is the analog of dot product for matrices: it is the sum of products of corresponding entries.

trace 是矩阵对角线元素合是 scalar

$trA^T B$ 类似于两个矩阵的点乘：假如 A 和 B 维度一样，就是对应元素乘积的和

性质：

1 tr scalar=scalar　2 $tr(A^T)=tr(A)$　转置不改变 diagnal 元素!!

3 trace rotation

A last useful trick for working with traces is *trace rotation*: $\mathrm{tr}\, AB = \mathrm{tr}\, BA$. Trace rotation works as long as AB is square, even if A and B are not (so that AB and BA are different-size matrices).

只能右边的移动到左边

tr A(BC)=tr (BC)A

tr (AB)C=tr C(AB)

inverse and determinant 微分

We can get a lot of power from differentiating common matrix operations such as inverse and determinant directly. (We haven't yet defined the determinant, written $|A|$, but one way to do so is to use the LU factorization: the determinant of a triangular matrix is the product of its diagonal entries, and the determinant of a product of matrices is the product of their determinants. So, if $A = LU$, then $|A| = |L|\,|U|$ = the product of all diagonal entries of L and U.)

determinant of A　先分解 A=LU　$|A|=|L|\cdot|U|$=product of L 和 U 的对角元素(L 对角元素和 U 对角元素的内积)

Let F be an $n \times n$ matrix. At points where F is invertible, we have

$$d(F^{-1}) = -F^{-1} dF\, F^{-1}\,.$$

Similarly, at points where F is invertible,

$$d|F| = |F|\,\mathrm{tr}(F^{-1}dF)\,.$$

And, if F is positive definite (and therefore invertible),

$$d\ln|F| = \mathrm{tr}(F^{-1}dF)\,.$$

At points where F is not positive definite, the differential of $\ln|F|$ does not exist (the slope becomes infinite as we cross the boundary of the set of psd matrices). Similarly, at points where F is not invertible, the differential of F^{-1} does not exist. The differential of $|F|$ exists even if F is not invertible, but the expression becomes more complicated — in fact it involves operations we haven't defined in this class.

different shapes

In the following table, we show some typical forms of differentials involving variables of different shapes: scalars, vectors, or matrices. Write s, t for scalars, v, w for vectors, and M, N for matrices. Then the typical forms are:

$ds = a\,dt$	$ds = v^T\,dw$	$ds = \mathrm{tr}(M^T dN)$
$dv = w\,ds$	$dv = M\,dw$	—
$dM = N\,ds$	—	—

- In $ds = a\,dt$, the scalar a is the ordinary derivative.
- In $ds = v^T dw$, the vector v is the gradient of a real-valued function.
- In $dv = w\,ds$, the vector w is the tangent to a curve.
- In $dv = M\,dw$, the matrix M is the Jacobian.

第 2 个 s 是 scalar,s 是 vector 参数 w 的 fucntion,v 是 gradient(多参数函数)向量

第 3 个 v 函数参数是 scalar s，返回 vector v(返回多个值)，w 也是 vector

4：v 和 w 都是 vector　函数个参数都是 vector，导数是 Matrice(decobean matrix)

Senond and higher differential

1. The first diffferential (review)

If $f \in U \to V$ is an operator between vector spaces U, V, the differential is a function $df(x; dx) \in U \times U \to V$. This function is linear in the dx argument, and potentially nonlinear in the x argument.

For example, if U and V are both real vector spaces, the differential takes the form

$$df(x; dx) = A(x)\,dx$$

where $A(x)$ is a matrix for each fixed x, and dx is a vector.

We can also write just $df = A(x)\,dx$. That is, we can use df to represent the output of the differential function, rather than the function itself. This more-compact notation is more common, but in this set of notes we'll use the longer notation as well, in order to be precise.

f input output 都是向量，因此 A 是矩阵，df 也可以表示成微分结

果

一阶微分，是 dx 的线性函数

二阶微分，是 dx 的二次函数

If we like, we can differentiate again: we can take the differential of df with respect to its x argument. The result is the *second differential*, written $d^2 f(x; dx)$. The second differential is a *quadratic* function of dx.

其实是；$d(df) = d^2 f = d(f(x)'dx) = f(x)''dxdx = f(x)''dx^2$

In more detail, start from $d(f(x)) = df(x; dx)$. Next fix $dx = u$ for some vector $u \in U$, and write

$$g(x) = df(x; u)\,.$$

Then we can take the differential of g,

$$d(g(x)) = dg(x; dx)\,.$$

The second differential of f in the direction u is defined to be

$$d^2 f(x; u) = dg(x; u)\,.$$

df(x,dx)是一个关于 x 和 dx 的 fucntion

此时再对 x 求微分，dx 作为常量

Note that we supply the same value $dx = u$ in both df and dg; this is what makes the second differential a quadratic function of dx.

二阶微分的形式：

For example, if $U = \mathbb{R}^n$ and $V = \mathbb{R}$, we have

$$df(x; dx) = a(x)\,dx$$

where the gradient $a(x) \in \mathbb{R}^{1\times n}$ is a row vector, and

$$d^2 f(x; dx) = dx^T A(x)\,dx$$

where the Hessian $A(x) \in \mathbb{R}^{n\times n}$ is a square matrix. We can also write simply

$$df = a(x)\,dx \quad d^2 f = dx^T A(x)\,dx$$

(taking df and $d^2 f$ to mean the outputs of the differential functions, rather than the functions themselves).

假设 f input Rn output R ，**a(x)是 f 一阶导数是 Rn**

二阶导数是 海森矩阵， $dx^T dx$ 是 dx 的二次项

3. Second-order Taylor expansions

Just as the first differential $df(x)$ represents the linear part of the change in f as a function of the change in x, the second differential $d^2f(x)$ represents the quadratic part. So, just as we can use the first differential to make a first-order Taylor expansion, we can use the first and second differentials to make a second-order Taylor expansion.

For example, if

$$f(t) = \begin{pmatrix} 2\cos(t) \\ 4\sin(t) \\ t \end{pmatrix},$$

then

$$df(t) = \begin{pmatrix} -2\sin(t) \\ 4\cos(t) \\ 1 \end{pmatrix} dt = f'(t)\,dt$$

and

$$d^2f(t) = \begin{pmatrix} -2\cos(t) \\ -4\sin(t) \\ 0 \end{pmatrix} dt^2 = f''(t)\,dt\,.$$

So, the second-order Taylor expansion is

$$df = f'(t)\,dt + \tfrac{1}{2}f''(t)\,dt^2$$

f 是一个参数，3 个 output

上面二阶微分右边少个 square

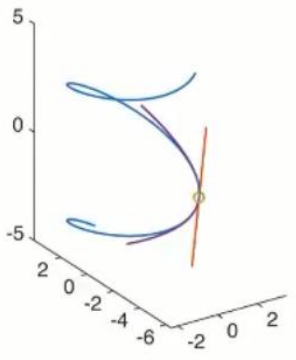
二阶泰勒展开式

等价于：

$$\hat{f}(t) \approx f(\hat{t}) + f'(\hat{t})(t-\hat{t}) + \tfrac{1}{2}f''(\hat{t})(t-\hat{t})^2\,.$$

对 t hat 附近进行近似

A graph:



The original function is in blue; the first-order Taylor approximation is in red; and the second-order Taylor approximation is in purple.

近似成的二次曲线，可以看出 order 越高，近似程度会越大！！

二阶微分例子

4. Computing the second differential

Computing the second differential works almost the same way as computing the first differential: the most important tools are linearity, the chain rule, and the product rule. The one additional rule for the second differential is that we treat pre-existing copies of dx as constant.

For example, we saw earlier that

$$d(\|Ax-b\|^2/2) = (Ax-b)^T A\,dx\,.$$

So, we have

$$\begin{aligned} d^2(\|Ax-b\|^2/2) &= d(((Ax-b)^T A\,dx) \\ &= (d(Ax-b))^T A\,dx \\ &= (A\,dx)^T A\,dx \\ &= dx^T A^T A\,dx\,. \end{aligned}$$

The second line uses linearity, together with the rule that we treat the previously-existing copy of dx as a constant.

treat 上一次的 dx 作为常数

Muliple variables 二阶微分

多元函数微分，其实就是分别求偏导数加 d

$$dz = \frac{\partial z}{\partial x}dx + \frac{\partial z}{\partial y}dy$$

$$f(x,y) = \cos(x)\sin(y)$$

then

$$df(x,y) = -\sin(x)\sin(y)\,dx + \cos(x)\cos(y)\,dy$$

二阶多元微分，其实就是二阶偏导数(把 d 当常数)，再对 xy 求一次微分

(by the product rule) and

$$\begin{aligned} d^2 f(x,y) = &-\cos(x)\sin(y)\,dx^2 - \sin(x)\cos(y)\,dx\,dy \\ &- \sin(x)\cos(y)\,dx\,dy - \cos(x)\sin(y)\,dy^2. \end{aligned}$$

The first two terms in $d^2 f$ come from the first term of df via the product rule, while the second two terms come from the second term of df via the product rule.

Statistical inference

目的

1. A fair coin?

Let's suppose we flip a coin 20 times, and it comes up heads 15 times. Do you think the coin is fair? Why or why not?

Statistical inference is the process of finding numerical answers to this sort of question about randomness: quantitatively, how much evidence do we have to support or reject a possible conclusion?

从数据结果做出推论

model

Statistical inference typically involves *models*: guesses we make about how the world works. These models will typically have numerical parameters, and our job in statistical inference is to figure out everything we can about the model parameters from our observations.

统计推断是为了建立 model 来 guess world 现象，需要从 observation(数据)中推断出我们 model 的参数

我们的 model 一般是概率分布模型，关于 p 的 model

For example, if we want to model a coin flip, we might say that the coin has probability p of coming up heads; this is called a *Bernoulli* distribution, and it's pretty much the only model we can use for such a simple experiment. The probability p is our parameter.

对投硬币进行建模，模型是概率模型

More interestingly, we might want to model repeated coin flips. In this case, we have more choices. We might say that each coin flip has the same probability p, or we might say that the first 10 flips have probability p_1 while the next 10 flips have probability p_2. We might even say that the first flip has probability p, but subsequent flips depend on the outcomes of previous flips: every time we get heads, the probability of future flips turning up heads increases a bit. (This would be an odd coin, but statisticians like to consider all possibilities.)

By far the most common way to model repeated events is called *i.i.d.*, or *independent identically distributed*. This is the "every flip has probability p" option above: we imagine that every repetition follows the same model, and that its outcome doesn't depend on any previous outcomes.

对于**重复试验**或重复事件的 model，可以有很多假设，**最常见的是 i.i.d.每一次试验都是相互独立的，并且都有相同的概率分布**

independent identical distribution

continue parameter vs discrete parameter

model 的 参数是连续的，我们需要离散化它 question

Often the parameters of our model are continuous: e.g., probabilities, or the weights of a linear separator. So far in this class we've only said how to work with probabilities for discrete random variables. So, for the purposes of this set of notes, we'll imagine that we have discretized our parameters finely: e.g., we could represent a continuous parameter as a 32-bit floating-point number. The probability for each individual 32-bit number will typically be tiny, but if we sum over all of them, we will get a total probability of 1.

For notation, we'll write $\mathbb{D}$ for a finite-precision discrete subset of the real numbers — this could be all 32-bit floating point numbers, or all multiples of 10^{-12} in the range $[-10^8, 10^8]$. We then allow each parameter to take values in a subset of $\mathbb{D}$: for example, a parameter that represents a probability would take values in $[0, 1] \cap \mathbb{D}$.

Caution: our choice of discretization matters. For example, if we include more resolution near zero (as the 32-bit floating point numbers do), then a uniform distribution over $\mathbb{D}$ will assign more probability to answers near zero. A similar effect happens for a parameter $\theta > 0$ if we decide to discretize evenly in θ vs. if we decide to discretize evenly in $\ln \theta$. There's no single correct way to discretize, so the only advice we can give is to be aware of this potential problem.

Note: a similar but harder-to-understand version of this problem would still happen if we stuck with a continuous representation of θ. See the Borel-Kolmogorov paradox.

如果让 parameter continue 非常复杂

我们只能选择一种更合适的 dicrete 方法

prior posterior

Piror

也就是对 paramter 的先验判断，before see the data

Even if we don't know the parameters in advance, we'll typically have some information about them: e.g., the likely range of values, or that some values are more plausible than others. We distill this information into a *prior distribution*:

$$\text{prior} = P(\text{parameters}).$$

The prior distribution is typically fairly broad, reflecting our lack of information: e.g., we might start knowing only that all values of p between 0 and 1 are equally likely, and therefore choose a uniform distribution.

不同的 prior 会影响我们最终的结论

例如 choose uniform distri

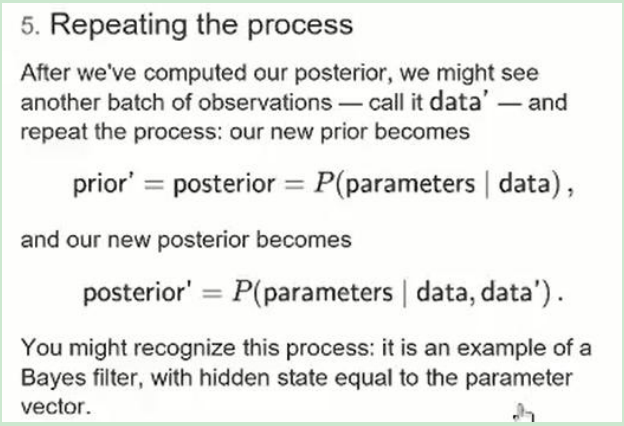
Posterior

Once we see the data, we will (hopefully) know more about the parameters: e.g., if we see 15 out of 20 flips come up heads, we might think that $p = 0.75$ is relatively likely, while $p = 0.01$ is relatively unlikely. We summarize our new state of information in a *posterior* distribution over our parameters:

$$\text{posterior} = P(\text{parameters} \mid \text{data}).$$

一旦我们 see the data(observation)得到我们的**后验概率(信息)，反推我们的 theta 的概率分布，进而对我们的参数可以更准确的估计**

对之前**参数分布的信息的更新**



5. Repeating the process

After we've computed our posterior, we might see another batch of observations — call it data' — and repeat the process: our new prior becomes

$$\text{prior}' = \text{posterior} = P(\text{parameters} \mid \text{data}),$$

and our new posterior becomes

$$\text{posterior}' = P(\text{parameters} \mid \text{data}, \text{data}').$$

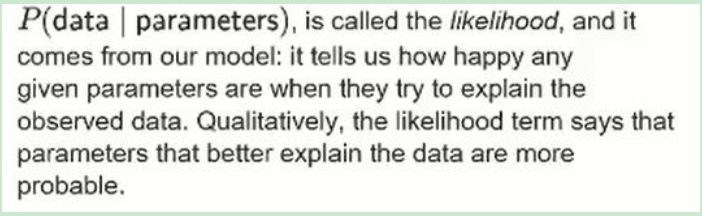
You might recognize this process: it is an example of a Bayes filter, with hidden state equal to the parameter vector.

observe likehood

利用 bayes filter 不断更新参数信息

find the posterior with bayes rule

$$p(parameters \mid data) = p(data \mid parameters)\, p(parameters) / p(data)$$



$P(\text{data} \mid \text{parameters})$, is called the *likelihood*, and it comes from our model: it tells us how happy any given parameters are when they try to explain the observed data. Qualitatively, the likelihood term says that parameters that better explain the data are more probable.

obersve likelihood $p(data \mid parameters)$ (不是先验，先验只是 p(参数))

likelihood 是在给定参数下，出现这些 observed data 的概率(可能性)

参数越能解释这些 data，概率越大

例如：我们假如**参数 p=0.1(正面)** 投 10 次硬币，p(8 次 head)<p(1

次 head)

p(data)

The other new term, $P(\text{data})$, presents a problem: we usually don't have an easy way to compute it. We can sometimes discover it from the constraint that the posterior has to sum to 1. But, in many cases even this method is computationally intractable: it may be difficult to sum over a high-dimensional parameter space.

如果 paramter 很多就不能用 sum =1 了，计算量太大

Fortunately, if we're primarily concerned about the relative probability of different sets of parameters under the posterior, we are in luck: the $P(\text{data})$ term is independent of the parameters, and so we can write

因为我们主要关心 $p(parameters|data)$，也就是**哪一个参数概率最大**

但 p(data)不依赖 parameter，可以忽略作为一个**常数** ,basye rule 可以写成比例形式：

$$p(parameters|data) \propto p(data|parameters)p(parameters)$$

In more detail, write $X = (X_1, X_2, \ldots, X_n)$ for our data: n i.i.d. repetitions of an experiment. Write $\theta \in \Theta \subseteq \mathbb{D}^k$ for the vector of parameters of our model.

Each X_i is a random variable, and therefore so is the entire data set X. θ could be just a single number (like p in our coin-flip example), but more-interesting models often have many parameters — thousands or even millions.

With this notation, Bayes' rule translates to:

$$P(\theta \mid X) \propto P(\theta)\, P(X \mid \theta)\,.$$

We can simplify further using conditional independence: by the i.i.d. assumption we can write $P(X \mid \theta) = \prod_i P(X_i \mid \theta)$, so Bayes' rule becomes:

$$P(\theta \mid X) \propto P(\theta) \prod_i P(X_i \mid \theta)\,.$$

利用 iid 假设和条件独立(**在给定 thate 下的多次独立观测**)：

likelihood obs data 的联合概率可以写成乘积形式(在给定参数下每个 obs X 值的概率)，bayse rule 的最终形式

例子：

In the coin flip example, we can write

$$P(X_i \mid \theta) = \begin{cases} \theta & \text{if } X_i = 1 \\ 1-\theta & \text{if } X_i = 0 \end{cases}$$

our likelyhood 概率，给定参数下

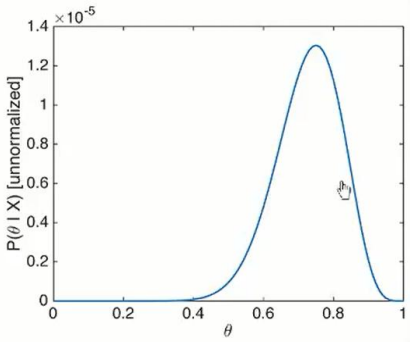
With our data of 15 heads and 5 tails, and assuming a uniform prior, our posterior is

$$P(\theta \mid X) \propto \theta^{15}(1-\theta)^5 \quad \text{for } \theta \in [0,1] \cap \mathbb{D}.$$

因为 prior p(theta) 是 a uniform distribution，that is, let all possible outcomes of theta have the same probability

此时P(theta)是常数，可以忽略，最终得到后验！！

If we plot this function, we get



From this plot, we can see that the coin is highly likely (but not certain) to be biased towards heads. To get the exact value, we can normalize $P(\theta \mid X)$ to sum to 1, and find that

$$P(\theta \geq 0.5 \mid X) = \frac{\sum_{\theta \geq 0.5} \theta^{15}(1-\theta)^5}{\sum_{0 \leq \theta \leq 1.0} \theta^{15}(1-\theta)^5} = 0.987.$$

可以只 report theta 的一个范围
分子只是上面右边的比例项，normalize

theta=0.7 时 p(theta)最高，做过手脚

MAP(假设有了先验)

我们这里特殊给先验 uniform，一般情况下不这样

report theta 的 MAP estimate 值

Sometimes it's convenient to report just a single value for θ instead of the entire posterior. Which θ should we report? It makes sense to pick the most probable value, i.e., the one that maximizes our posterior $P(\theta \mid X)$. This value is called the *maximum a posteriori* or *MAP* estimate of θ.

估计参数时选择后验概率最大的那个参数是 MAP estimate

MAP:也可以选择概率最大的值，这时候假设 theta 是连续变量

The easiest way to find the MAP estimate is often to revert back to a continuous representation of the posterior distribution. We can then differentiate the posterior and set the differential to zero. (This approach works best if we used an even discretization; if we did not, we need to compensate with an extra term in the continuous representation of the posterior.)

$$P(\theta \mid X) \propto \theta^{15}(1-\theta)^5$$

求导令为 0

To make the math easier, we typically maximize $\ln(P(\theta \mid X))$ instead of $P(\theta \mid X)$ itself — the result is equivalent since log is a monotone function.

In our example,

$$\begin{aligned}
\ln P(\theta \mid X) &= 15\ln\theta + 5\ln(1-\theta) + \text{constant} \\
0 &= d\ln P(\theta \mid X) \\
&= d(15\ln\theta + 5\ln(1-\theta)) \\
&= 15\frac{d\theta}{\theta} - 5\frac{d\theta}{1-\theta} \\
5\theta &= 15(1-\theta) \\
\theta &= 15/20 = 0.75
\end{aligned}$$

This answer certainly seems reasonable: with 15 heads out of 20 flips, the most likely value for $P(\text{heads})$ is 15/20.

maximize the log of posterior，更简单！！！

乘积转化成和，而且 log 容易求导

结果其实就是我们观测的结果！！！

This answer certainly seems reasonable: with 15 heads out of 20 flips, the most likely value for $P(\text{heads})$ is 15/20.

We can see now why taking the log of the posterior makes the derivation easier: we need to differentiate a sum of terms rather than a product of terms, so we can use linearity instead of the product rule.

MLE

10. Maximum likelihood

Different people might reasonably disagree on which prior to use for our parameters. The posterior (and therefore the MAP estimate) will be different under different priors. For example, in the coin flip problem, if our prior were uniform over $[0.2, 0.4] \cap \mathbb{D}$ instead of $[0, 1] \cap \mathbb{D}$, we certainly wouldn't get 0.75 as the most likely value of θ, since 0.75 is impossible according to our prior distribution.

不同的 prior，会造成我们最终的 MAP 的几个不同参数估计结果

Can we come up with a single estimate of θ that everyone can agree on? One good candidate is the *maximum likelihood estimate* or *MLE*.

To get the MLE, we simply omit the prior from Bayes' rule — or equivalently, pretend that the prior is uniform over some large range of possible values for θ. So, instead of maximizing (likelihood * prior), we just maximize likelihood: a uniform distribution is constant with respect to θ, so it drops out when we take the derivative.

忽略 prior，或者让 prior 都是 uniform，对于所有 theta 都是常数(可以忽略)，其实跟上面的过程一样

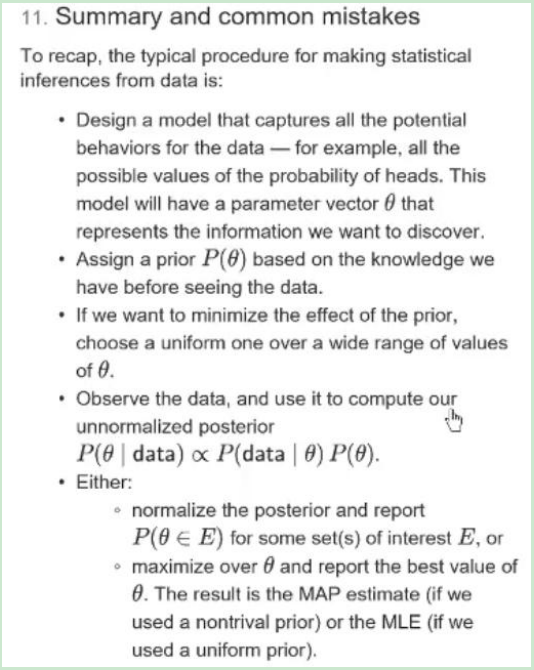
Using a uniform prior makes sense because it is the least informative possible choice — that is, any other choice of prior would favor some values of θ over others. This is a reasonable compromise: while nobody wants to give up their own biases, they would prefer to do so rather than accept someone else's.

In the coin flip example, we actually used a uniform prior already — so, our MAP estimate is already an MLE as well.

当 MAP 的先验是 uniiform 时，此时 MAP 等于 MLE

(Of course, the meaning of a uniform distribution depends on how we discretized θ — or equivalently in the continuous case, on which transform of θ we decided to use. So we haven't really avoided all arguments, just pushed them farther down the line. Fortunately, the more data we have, the less this issue matters: as the number of observations goes to infinity, the limit of the MLE is the same, no matter which discretization or transform of θ we decided to use.)

对 theta 不同的离散化，也会影响最终的结果，但随着数据的增多，这个影响会越来越小



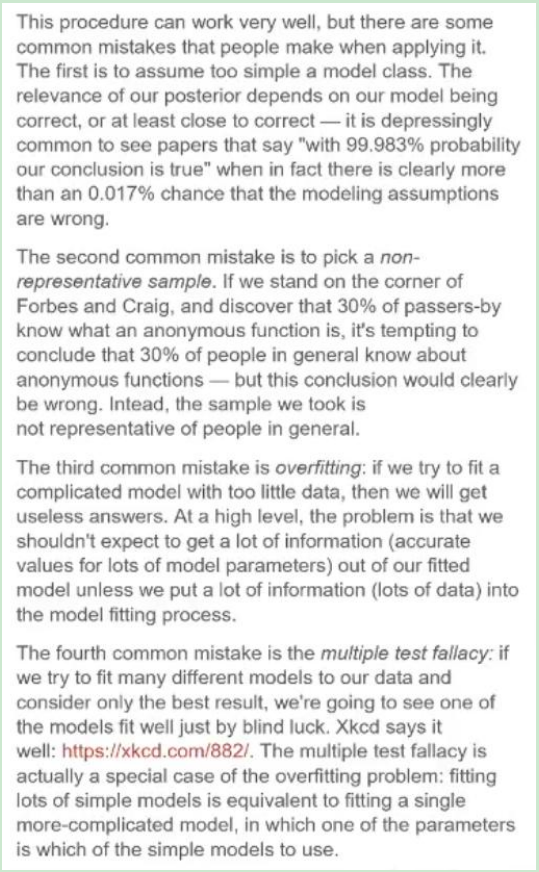
11. Summary and common mistakes

To recap, the typical procedure for making statistical inferences from data is:

- Design a model that captures all the potential behaviors for the data — for example, all the possible values of the probability of heads. This model will have a parameter vector θ that represents the information we want to discover.
- Assign a prior $P(\theta)$ based on the knowledge we have before seeing the data.
- If we want to minimize the effect of the prior, choose a uniform one over a wide range of values of θ.
- Observe the data, and use it to compute our unnormalized posterior $P(\theta \mid \text{data}) \propto P(\text{data} \mid \theta)\, P(\theta)$.
- Either:
  - normalize the posterior and report $P(\theta \in E)$ for some set(s) of interest E, or
  - maximize over θ and report the best value of θ. The result is the MAP estimate (if we used a nontrival prior) or the MLE (if we used a uniform prior).

后验概率的两种 reprot

mistake:



This procedure can work very well, but there are some common mistakes that people make when applying it. The first is to assume too simple a model class. The relevance of our posterior depends on our model being correct, or at least close to correct — it is depressingly common to see papers that say "with 99.983% probability our conclusion is true" when in fact there is clearly more than an 0.017% chance that the modeling assumptions are wrong.

The second common mistake is to pick a *non-representative sample*. If we stand on the corner of Forbes and Craig, and discover that 30% of passers-by know what an anonymous function is, it's tempting to conclude that 30% of people in general know about anonymous functions — but this conclusion would clearly be wrong. Intead, the sample we took is not representative of people in general.

The third common mistake is *overfitting*: if we try to fit a complicated model with too little data, then we will get useless answers. At a high level, the problem is that we shouldn't expect to get a lot of information (accurate values for lots of model parameters) out of our fitted model unless we put a lot of information (lots of data) into the model fitting process.

The fourth common mistake is the *multiple test fallacy:* if we try to fit many different models to our data and consider only the best result, we're going to see one of the models fit well just by blind luck. Xkcd says it well: https://xkcd.com/882/. The multiple test fallacy is actually a special case of the overfitting problem: fitting lots of simple models is equivalent to fitting a single more-complicated model, in which one of the parameters is which of the simple models to use.

1 question **model 太 simple,**

2 数据 sample 不代表总体

3 overfitting model 太复杂，数据太少