Variance and Standard Deviation

Expectation summarizes a lot of information about a random variable as a single number. But no single number can tell it all.

Compare these two distributions:

- Distribution 1:

$$\Pr(49) = \Pr(51) = 1/4; \quad \Pr(50) = 1/2.$$

- Distribution 2: $\Pr(0) = \Pr(50) = \Pr(100) = 1/3$.

Both have the same expectation: 50. But the first is much less "dispersed" than the second. We want a measure of *dispersion*.

- One measure of dispersion is how far things are from the mean, on average.

Given a random variable X, $(X(s) - E(X))^2$ measures how far the value of s is from the mean value (the expectation) of X. Define the *variance* of X to be

$$Var(X) = E((X - E(X))^2) = \Sigma_{s \in S} \Pr(s)(X(s) - E(X))^2$$

The *standard deviation* of X is

$$\sigma_X = \sqrt{Var(X)} = \sqrt{\Sigma_{s \in S} \Pr(s)(X(s) - E(X))^2}$$

- Why not use $|X(s) - E(X)|$ as the measure of distance?
- $(X(s) - E(X))^2$ turns out to have nicer mathematical properties.
- In R^n, the distance between $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ is $\sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$

Example:

- The variance of distribution 1 is
$$\frac{1}{4}(51 - 50)^2 + \frac{1}{2}(50 - 50)^2 + \frac{1}{4}(49 - 50)^2 = \frac{1}{2}$$
- The variance of distribution 2 is
$$\frac{1}{3}(100 - 50)^2 + \frac{1}{3}(50 - 50)^2 + \frac{1}{3}(0 - 50)^2 = \frac{5000}{3}$$

Note: We are not covering 6.6, 6.7, or the negative binomial and Poisson distributions discussed in 6.4.

Logic

Logic is a tool for formalizing reasoning. There are lots of different logics:

- probabilistic logic: for reasoning about probability
- temporal logic: for reasoning about time (and programs)
- epistemic logic: for reasoning about knowledge

The simplest logic (on which all the rest are based) is *propositional logic*. It is intended to capture features of arguments such as the following:

> Borogroves are mimsy whenever it is brillig. It is now brillig and this thing is a borogrove. Hence this thing is mimsy.

Propositional logic is good for reasoning about

- conjunction, negation, implication ("if . . . then . . .")

Amazingly enough, it is also useful for

- circuit design
- program verification

Proposition Logic: Syntax

To formalize the reasoning process, we need to restrict the kinds of things we can say. Propositional logic is particularly restrictive.
The *syntax* of propositional logic tells us what are legitimate formulas.

We start with *primitive propositions*. Think of these as statements like

- It is now brilling
- This thing is mimsy
- It's raining in San Francisco
- n is even

We can then form more complicated *compound propositions* using connectives like:

- $\neg$: not
- $\wedge$: and
- $\vee$: or
- $\Rightarrow$: implies
- $\Leftrightarrow$: equivalent (if and only if)

Examples:

- $\neg P$: it is not the case that P
- $P \wedge Q$: P and Q
- $P \vee Q$: P or Q
- $P \Rightarrow Q$: P implies Q (if P then Q)

Typical formula:

$$P \wedge (\neg P \Rightarrow (Q \Rightarrow (R \vee P)))$$

Wffs

Formally, we define *well-formed formulas* (*wffs* or just *formulas*) inductively (remember Chapter 2!):
The wffs consist of the least set of strings such that:

1. Every primitive proposition $P, Q, R, \ldots$ is a wff
2. If A is a wff, so is $\neg A$
3. If A and B are wffs, so are $A \wedge B$, $A \vee B$, $A \Rightarrow B$, and $A \Leftrightarrow B$

Disambiguating Wffs

We use parentheses to disambiguate wffs:

- $P \vee Q \wedge R$ can be either $(P \vee Q) \wedge R$ or $P \vee (Q \wedge R)$

Mathematicians are lazy, so there are standard rules to avoid putting in parentheses.

- In arithmetic expressions, $\times$ binds more tightly than $+$, so $3 + 2 \times 5$ means $3 + (2 \times 5)$
- In wffs, here is the precedence order:
  - $\neg$
  - $\wedge$
  - $\vee$
  - $\Rightarrow$
  - $\Leftrightarrow$

  Thus, $P \vee Q \wedge R$ is $P \vee (Q \wedge R)$;
  $P \vee \neg Q \wedge R$ is $P \vee ((\neg Q) \wedge R)$
  $P \vee \neg Q \Rightarrow R$ is $(P \vee (\neg Q)) \Rightarrow R$
- With two or more instances of the same binary connective, evaluate left to right:
  $P \Rightarrow Q \Rightarrow R$ is $(P \Rightarrow Q) \Rightarrow R$

Translating English to Wffs

To analyze reasoning, we have to be able to translate English to wffs.
Consider the following sentences:

1. Bob doesn't love Alice
2. Bob loves Alice and loves Ann
3. Bob loves Alice or Ann
4. Bob loves Alice but doesn't love Ann
5. If Bob loves Alice then he doesn't love Ann

First find appropriate primitive propositions:

- P: Bob loves Alice
- Q: Bob loves Ann

Then translate:

1. $\neg P$
2. $P \wedge Q$
3. $P \vee Q$
4. $P \wedge \neg Q$ (note: "but" becomes "and")
5. $P \Rightarrow \neg Q$

Evaluating Formulas

Given a formula, we want to decide if it is true or false.

How do we deal with a complicated formula like:

$$P \wedge (\neg P \Rightarrow (Q \Rightarrow (R \vee P)))$$

The truth or falsity of such a formula depends on the truth or falsity of the primitive propositions that appear in it. We use *truth tables* to describe how the basic connectives ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$) work.

Truth Tables

For $\neg$:

P	$\neg P$
T	F
F	T

For $\wedge$:

P	Q	$P \wedge Q$
T	T	
T	F	
F	T	
F	F	

For $\vee$:

P	Q	$P \vee Q$
T	T	
T	F	
F	T	
F	F	

This means $\vee$ is *inclusive* or, not *exclusive* or.

Exclusive Or

What's the truth table for "exclusive or"?

P	Q	$P \oplus Q$
T	T	F
T	F	T
F	T	T
F	F	F

$P \oplus Q$ is equivalent to $(P \wedge \neg Q) \vee (\neg P \wedge Q)$

P	Q	$\neg P$	$\neg Q$	$P \wedge \neg Q$	$Q \wedge \neg P$	$(P \wedge \neg Q) \vee (\neg P \wedge Q)$
T	T	F	F	F	F	F
T	F	F	T	T	F	T
F	T	T	F	F	T	T
F	F	T	T	F	F	F

Truth Table for Implication

For $\Rightarrow$:

P	Q	$P \Rightarrow Q$
T	T	
T	F	
F	T	
F	F	

Why is this right? What should the truth value of $P \Rightarrow Q$ be when P is false?

- This choice is mathematically convenient
- As long as Q is true when P is true, then $P \Rightarrow Q$ will be true no matter what.

For $\Leftrightarrow$:

P	Q	$P \Leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

How many possible truth tables are there with two primitive propositions?

P	Q	?
T	T	
T	F	
F	T	
F	F	

By the product rule, there are 16.

We've defined connectives corresponding to 4 of them: $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$.

- Why didn't we bother with the rest?
- They're definable!

Other Equivalences

It's not hard to see that $P \oplus Q$ is also equivalent to $\neg(P \Leftrightarrow Q)$

Thus, $P \Leftrightarrow Q$ is equivalent to $\neg(P \oplus Q)$, which is equivalent to

$$\neg((P \wedge \neg Q) \vee (\neg P \wedge Q))$$

Thus, we don't need $\Leftrightarrow$ either.

We also don't need $\Rightarrow$:

$P \Rightarrow Q$ is equivalent to $\neg P \vee Q$

We also don't need $\vee$:

$P \vee Q$ is equivalent to $\neg(\neg P \wedge \neg Q)$

Each of the sixteen possible connectives can be expressed using $\neg$ and $\wedge$ (or $\vee$)

Tautologies

A *truth assignment* is an assignment of T or F to every proposition.

- How hard is it to check if a formula is true under a given truth assignment?
- Easy: just plug it in and evaluate.
  - Time linear in the length of the formula

A *tautology* (or *theorem*) is a formula that evaluates to T for *every* truth assignment.

Examples:

- $(P \vee Q) \Leftrightarrow \neg(\neg P \wedge \neg Q)$
- $P \vee Q \vee (\neg P \wedge \neg Q)$
- $(P \Rightarrow Q) \vee (Q \Rightarrow P)$
  - It's necessarily true that if elephants are pink then the moon is made of green cheese or if the moon is made of green cheese, then elephants are pink.

How hard is it to check if a formula is a tautology?

- How many truth assignments do we have to try?

Arguments

Definition: An argument has the form

A_1
A_2
$\vdots$
A_n

B

$A_1, \ldots, A_n$ are called the *premises* of the argument; B is called the *conclusion*. An argument is *valid* if, whenever the premises are true, then the conclusion is true.

Logical Implication

A formula A *logically implies* B if $A \Rightarrow B$ is a tautology.

Theorem: An argument is valid iff the conjunction of its premises logically implies the conclusion.

Proof: Suppose the argument is valid. We want to show $(A_1 \wedge \ldots \wedge A_n) \Rightarrow B$ is a tautology.

- Do we have to try all 2^k truth assignments (where $k = \#$primitive propositions in $A_1, \ldots, A_n, B$).

It's not that bad.

- Because of the way we defined $\Rightarrow$, $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ is guaranteed to be true if $A_1 \wedge \ldots \wedge A_n$ is false.
- But if $A_1 \wedge \ldots \wedge A_n$ is true, B is true, since the argument is valid.
- Thus, $(A_1 \wedge \ldots \wedge A_n) \Rightarrow B$ is a tautology.

For the converse, suppose $(A_1 \wedge \ldots \wedge A_n) \Rightarrow B$ is a tautology. If $A_1, \ldots, A_n$ are true, then B must be true. Hence the argument is valid.

Remember:

Borogroves are mimsy whenever it is brillig.
It is now brillig and this thing is a borogrove.
Hence this thing is mimsy.

Suppose

- P: It is now brilling
- Q: This thing is a borogrove
- R: This thing is mimsy

This becomes:

$P \Rightarrow (Q \Rightarrow R)$
$P \wedge Q$

R

This argument is valid if

$$[(P \Rightarrow (Q \Rightarrow R)) \wedge (P \wedge Q)] \Rightarrow R$$

is a tautology.

Natural Deduction

Are there better ways of telling if a formula is a tautology than trying all possible truth assignments.

- In the worst case, it appears not.
  - The problem is co-NP-complete.
  - The *satisfiability* problem—deciding if at least one truth assignment makes the formula true—is NP-complete.

Nevertheless, it often seems that the reasoning is staightforward:
Why is this true:

$$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$$

We want to show that if $P \Rightarrow Q$ and $Q \Rightarrow R$ is true, then $P \Rightarrow R$ is true.

So assume that $P \Rightarrow Q$ and $Q \Rightarrow R$ are both true. To show that $P \Rightarrow R$, assume that P is true. Since $P \Rightarrow Q$ is true, Q must be true. Since $Q \Rightarrow R$ is true, R must be true. Hence, $P \Rightarrow R$ is true.

We want to codify such reasoning.

Formal Deductive Systems

A *formal deductive system* (also known as an *axiomatization*) consists of

- *axioms* (special formulas)
- *rules of inference*: ways of getting new formulas from other formulas. These have the form

  $$\begin{array}{l} A_1 \\ A_2 \\ \vdots \\ A_n \\ \hline B \end{array}$$

  Read this as "from $A_1, \ldots, A_n$, infer B."

Think of the axioms as tautologies, while the rules of inference give you a way to derive new tautologies from old ones.

Standard question in logic:

> Can we come up with a nice *sound and complete axiomatization*: a (small, natural) collection of axioms and inference rules from which it is possible to derive all and only the tautologies?

- *Soundness* says you get only tautologies
- *Completeness* says you get all the tautologies

It's amazing how many logics have nice sound and complete axiomatizations.

- Gödel's famous incompleteness theorem says that there is no sound and complete axiomatization for arithmetic.

Some Rules of Inference

Rule 1: Modus Ponens

$A \Rightarrow B$
A

B

It's easy to see that if $A \Rightarrow B$ is true and A is true, then B is true.

Rule 2: Conditional Proofs:
If, by assuming $A_1, \ldots, A_n$, you can derive B, then you can conclude $(A_1 \wedge \ldots \wedge A_n) \Rightarrow B$.

- Conditional proofs can be nested
- When writing a proof, if you assume A, indent it and all subsequent steps until you conclude B, then write $A \Rightarrow B$ unindented.

Prove that $((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$ is a tautology.

1. $P \Rightarrow Q$ (assumption)
2. $Q \Rightarrow R$ (assumption)
    3. P (nested assumption)
    4. Q (1,3,modus ponens)
    5. R (2,4,modus ponens)
6. $P \Rightarrow R$ (3–5,Rule 2)
7. $((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$ (1–6,Rule 2)

Some axioms (these aren't mentioned in the book, but you can use them):

- $(A \Leftrightarrow B) \Rightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$
- $(A \wedge B) \Rightarrow A$
- $(A \wedge B) \Rightarrow B$
- $\neg\neg A \Leftrightarrow A$

Rule 3: Modus Tollens

$A \Rightarrow B$
$\neg B$

$\neg A$

Example: Prove that $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$ is a tautology.

1. $P \Rightarrow Q$	(assumption)
2. $\neg Q$	(nested assumption)
3. $\neg P$	(modus tollens,1,2)
4. $\neg Q \Rightarrow \neg P$	(Rule 2)
5. $(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$	(Rule 2)

The Rules of the Game

What are you allowed to use when you're proving something:

- That depends.
- We have to specify (in advance) what are the axioms and rules of inference
- For the purposes of this course, you're allowed to use Rules 1–3, and *nothing else* unless the problem says you can.

Using Rule 2

- You get to make *any assumptions you want.*
- Every time you make an assumption, indent the line with the assumption (and the following lines), so you can see it visually.
- At the end of the day, if, by assuming $A_1, \ldots, A_n$, you can prove B, then you can *discharge* the assumption and write

$$A_1 \wedge \ldots \wedge A_n \Rightarrow B$$

- You must discharge all the assumptions you've made
- It's often a good idea to assume the left-hand side of a statement like $A \Rightarrow B$ that you're trying to prove.
- What the book calls "premises" are really assumptions

Example: Prove $A \wedge B \Rightarrow A$.

1. A	(assumption)
2. B	(assumption)
3. $(A \wedge B) \Rightarrow A$	(1,2, Rule 2)

Algorithm Verification

This is (yet another) hot area of computer science.

- How do you prove that your program is correct?
  - You could test it on a bunch of instances. That runs the risk of not exercising all the features of the program.

In general, this is an intractable problem.

- For small program fragments, formal verification using logic is useful
- It also leads to insights into program design.

Consider the following algorithm for multiplication:

Input x [Integer ≥ 0]
y [Integer]
Algorithm Mult
$prod \leftarrow 0$
$u \leftarrow 0$
repeat $u = x$
$prod \leftarrow prod + y$
$u \leftarrow u + 1$
end repeat

How do we prove this is correct?

- Idea (due to Floyd and Hoare): annotate program with assertions that are true of the line of code immediately following them.
- An assertion just before a loop is true each time the loop is entered. This is a *loop invariant*.
- An assertion at the end of a program is true after running the program.

Input x [Integer ≥ 0]

y [Integer]

Algorithm Mult

$\Pi \leftarrow 0$

$u \leftarrow 0$

$\{prod = uy\}$ [Loop invariant]

repeat $u = x$

$prod \leftarrow prod + y$

$u \leftarrow u + 1$

end repeat

$\{prod = uy \wedge u = x\}$

Thus, we must show $prod = uy$ is true each time we enter the loop.

- Proof is by induction (big surprise)

It follows that $prod = uy \wedge u = x$ holds after exiting the program, since we exit after trying the loop (so $prod = uy$) and discovering $u = x$. It follows that $prod = xy$ at termination.

But how do we know the program terminates?

- We prove (by induction!) that after the kth iteration of the loop, $u = k$.
- Since $x \geq 0$, eventually $u = x$, and we terminate the loop (and program)

We won't be covering Boolean algebra (it's done in CS 314), although you should read Section 7.5!