W7: Data Science I

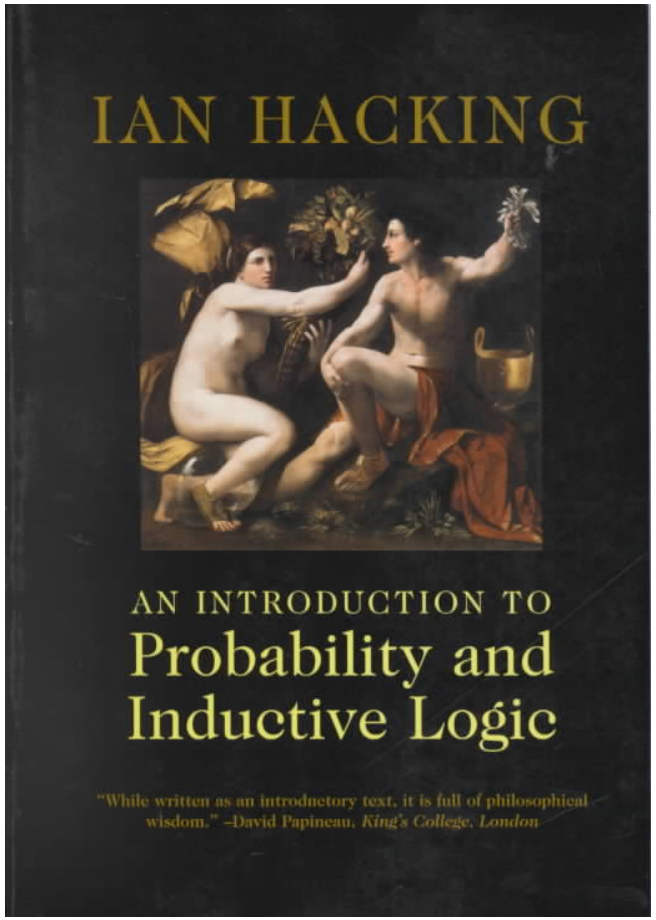
zz@nyu.edu

Segment overview

- Probability and statistics (1 lecture)
- Machine learning (2 lectures)

- Learn by doing, with OOP
- New & useful algorithms:
  - Monte Carlo simulation
  - K-means clustering
  - K-nearest neighbor classification
  - Linear/perceptron classifier

Warming up: probability



Not only about probability and logic

But also about philosophy, science in general

Used in Prof Brad Weslake's class

Odd question #3

- 6/49 (government-run lottery)
- 6 lucky numbers out of 1 to 49, without replacement
- Prizes divided among those who choose the lucky numbers

- Two sequences of lucky numbers:
  - A: 1, 2, 3, 4, 5, 6
  - B: 28, 21, 33, 17, 4, 8
- Q: which one do you prefer

Odd question #4

- A pair of fair dices (with 6 faces)
- Throw of 7: if adds up to 7
- Throw of 6: if adds up to 6

Q:

- Which is more likely?

Probability

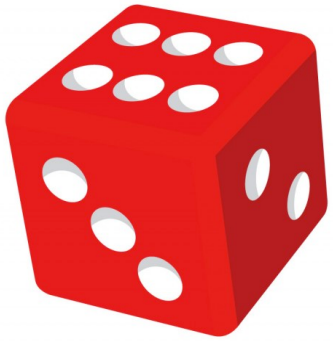
- The measure of the likelihood of an event
  - $P(\boldsymbol{X} = x)$; a real value between 0 and 1

Examples: probability that…

- tomorrow will rain
- tomorrow will rain **or** not rain
- tomorrow will rain **and** not rain
- both today and tomorrow will rain
- tomorrow will rain, given today rains

Joint probability

- See wiki page of joint distribution
- Event A: outcome is an even number
- Event B: outcome is a prime number
- What's the probability that both A and B are true?

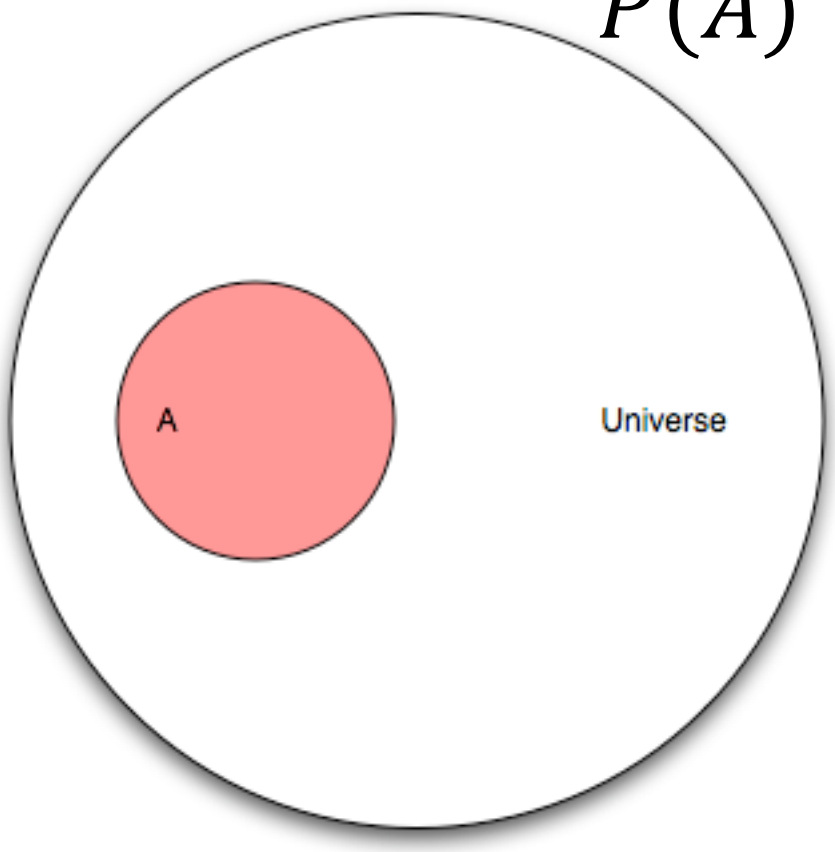


	1	2	3	4	5	6
A	0	1	0	1	0	1
B	0	1	1	0	1	0

$$\mathrm{P}(A=0,B=0)=P\{1\}=\frac{1}{6},\ \mathrm{P}(A=1,B=0)=P\{4,6\}=\frac{2}{6},$$
$$\mathrm{P}(A=0,B=1)=P\{3,5\}=\frac{2}{6},\ \mathrm{P}(A=1,B=1)=P\{2\}=\frac{1}{6}.$$

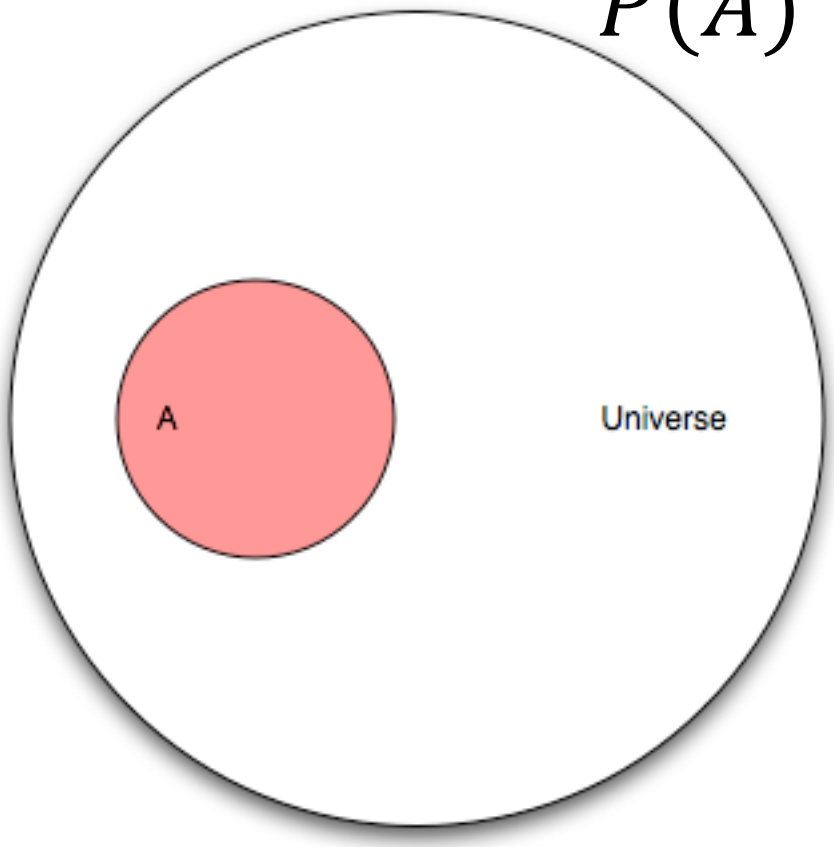
Probability “universe”: individual event

$P(A)$

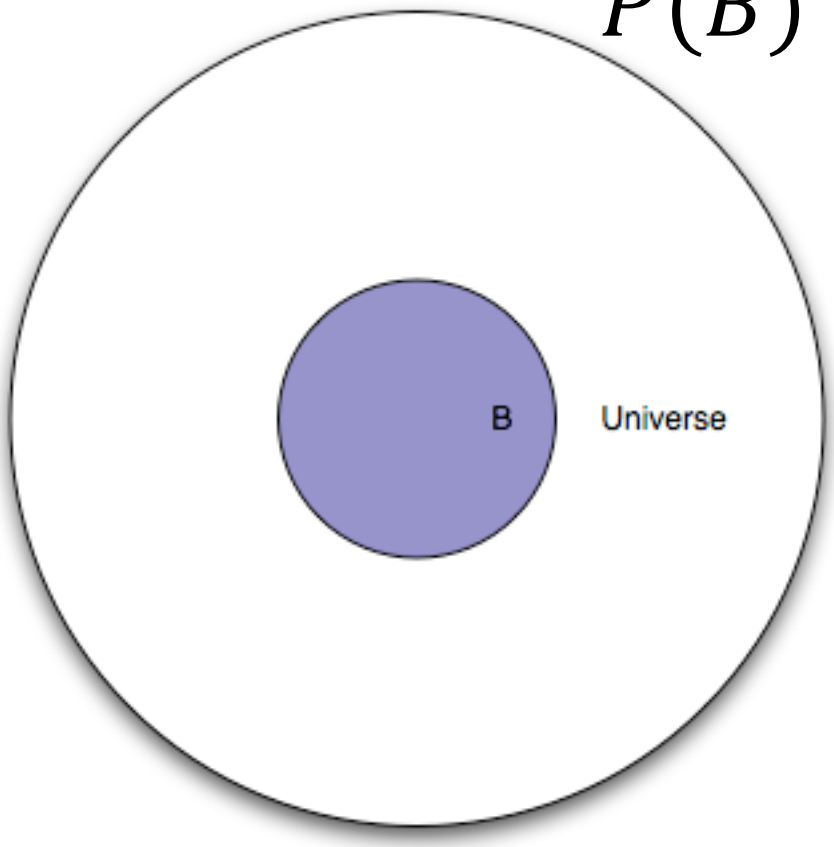


Probability “universe”: individual event

$P(A)$

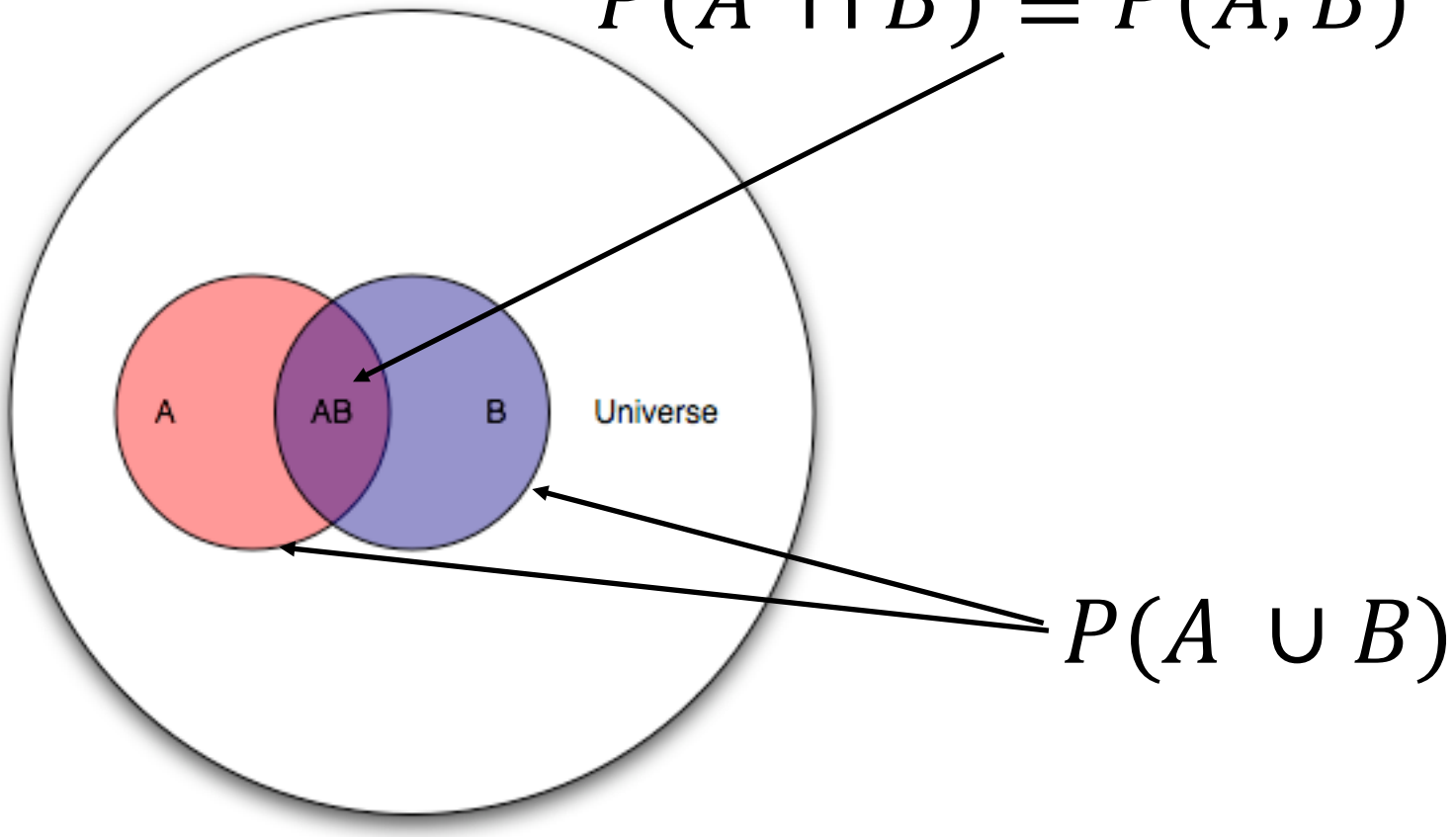


$P(B)$



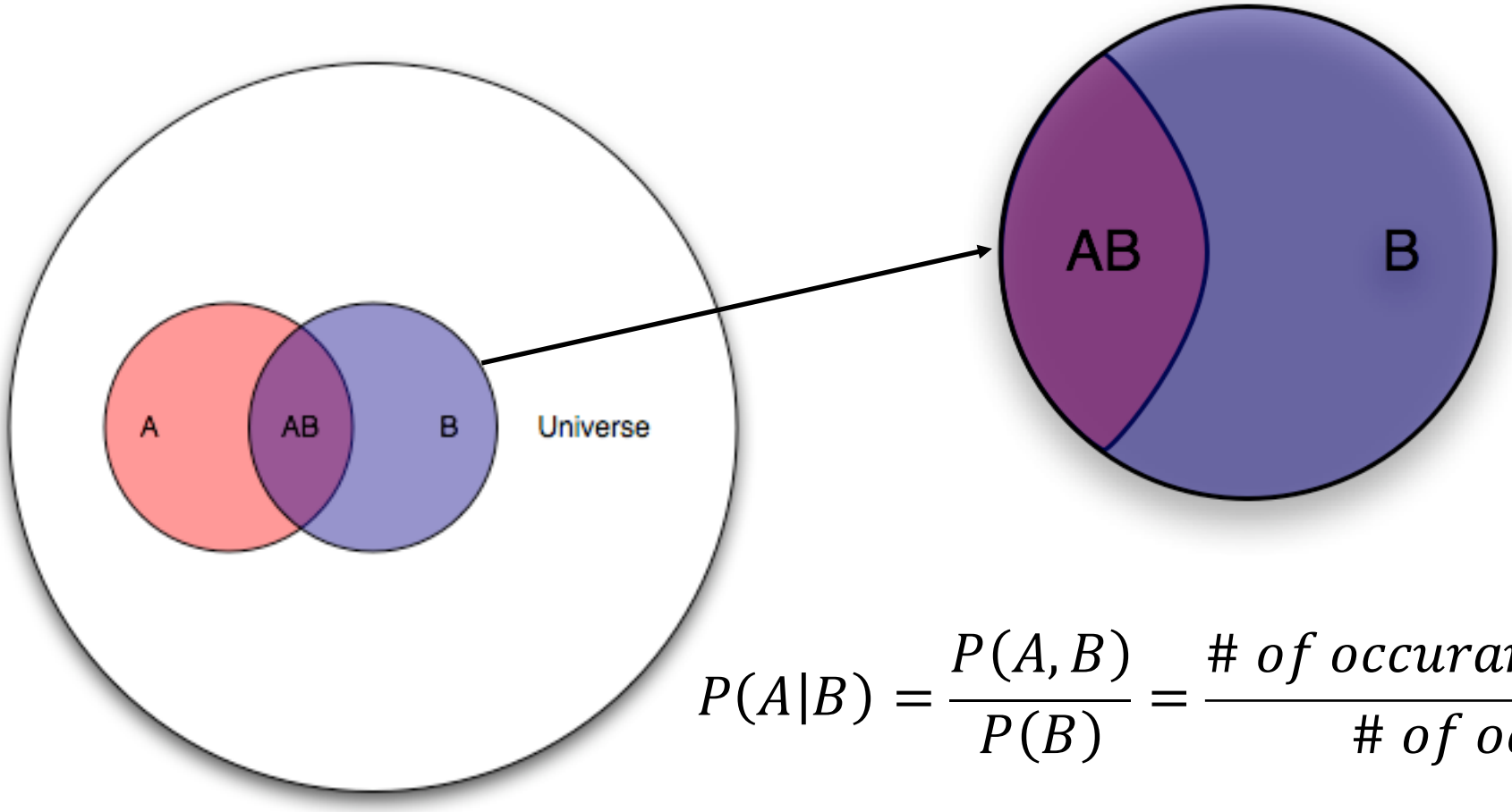
Probability “universe”: joint events

$$P(A \cap B) \equiv P(A, B)$$



$$P(A \cup B)$$

Probability “universe”: conditional probability



$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{\#\ of\ occurance\ of\ both\ A\ and\ B}{\#\ of\ occurance\ of\ B}$$

Conditional probability

- See wiki page of conditional probability
- $P(A|B)$: probability of A, given B has occurred
- $P(A|B) = \frac{P(A,B)}{P(B)} = \frac{\#\ of\ occurance\ of\ both\ A\ and\ B}{\#\ of\ occurance\ of\ B}$

- Example: “Probability that tomorrow will rain, given today rains”
- Example: “Probability that x < 3, given x is an odd number < 10”

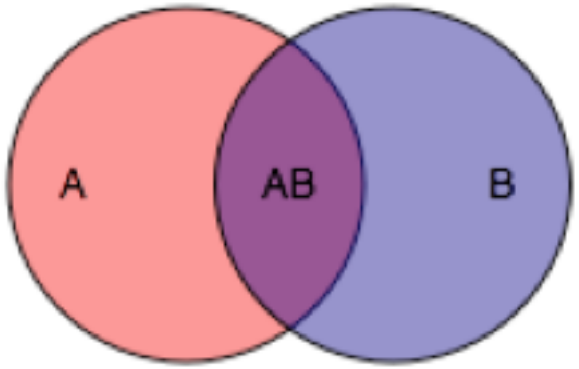
Summary of probability

- See [wiki page of probability]

Event	Probability
A	$P(A) \in [0,1]$
not A	$P(A^c) = 1 - P(A)$
A or B	$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ $P(A \cup B) = P(A) + P(B)$ if A and B are mutually exclusive
A and B	$P(A \cap B) = P(A\|B)P(B) = P(B\|A)P(A)$ $P(A \cap B) = P(A)P(B)$ if A and B are independent
A given B	$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B\|A)P(A)}{P(B)}$

Bayesian theory: a worldview

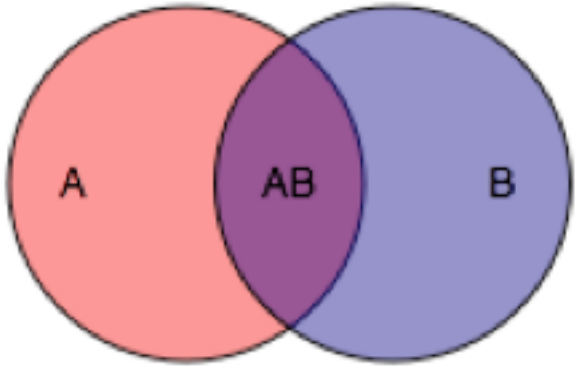
- $P(A,B) = P(A|B)P(B) \equiv P(B|A)|P(A)$
- $P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$



- Evidence shapes belief
- $P(model|event) = \frac{P(event|model)P(model)}{P(event)} \sim P(event|model)P(model)$

Bayesian theory: a worldview

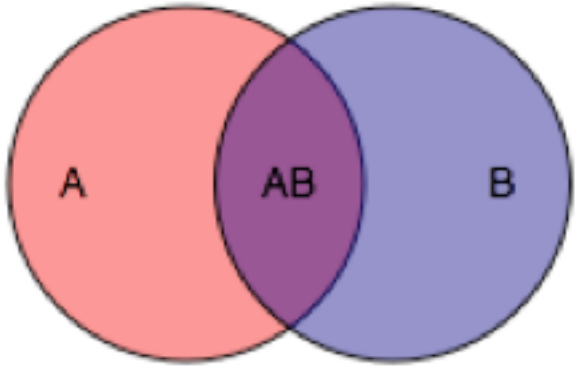
- $P(A,B) = P(A|B)P(B) \equiv P(B|A)|P(A)$
- $P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$



- Evidence shapes belief
- $P(model|event) = \frac{P(event|model)P(model)}{P(event)} \sim P(event|model)P(model)$

Bayesian theory: a worldview

- $P(A,B) = P(A|B)P(B) \equiv P(B|A)|P(A)$
- $P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$



- Evidence shapes belief
- $P(model|event) = \frac{P(event|model)P(model)}{P(event)} \sim P(event|model)P(model)$

Prior w/o seeing data

Posterior belief after see the data

Likelihood of seeing the data under the prior belief

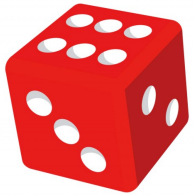
Bayesian theory: a worldview

$$P(model|event) \sim p(event|model)p(model)$$

- Events happen independent of our will
- Belief (or model) of the world can vary dramatically
- Good model(s) should be able to predict!
- .. And allow observations to shape the model

Many good materials around, see for instance "critical thinking"

Coding exercise 1: implement a dice class

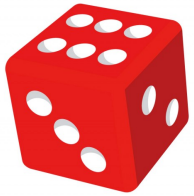


```
10 class Dice:
11     def __init__(self, sides=2):
12         self.n_sides = sides
13         self.bounds = [x/sides for x in range(0, sides)]
14         self.bounds.append(1.0)
15         self.point = None
16         self.lands = 0
17 
18     def set_bounds(self, r):
19         assert len(r) == len(self.bounds)
20         self.bounds = r
21 
22     def get_bounds(self):
23         return self.bounds
24 
25     def roll(self):
26 # replace the following line with you code
27 # it should set self.point as a random var in [0, 1]
28 # return which side the dice lands on
29         return dice_helper.roll(self)
```

- N-sided
- Maintain a list of N-partitioned range
- Roll to land in one of the partition

- Make it biased with set_bounds(self, range)

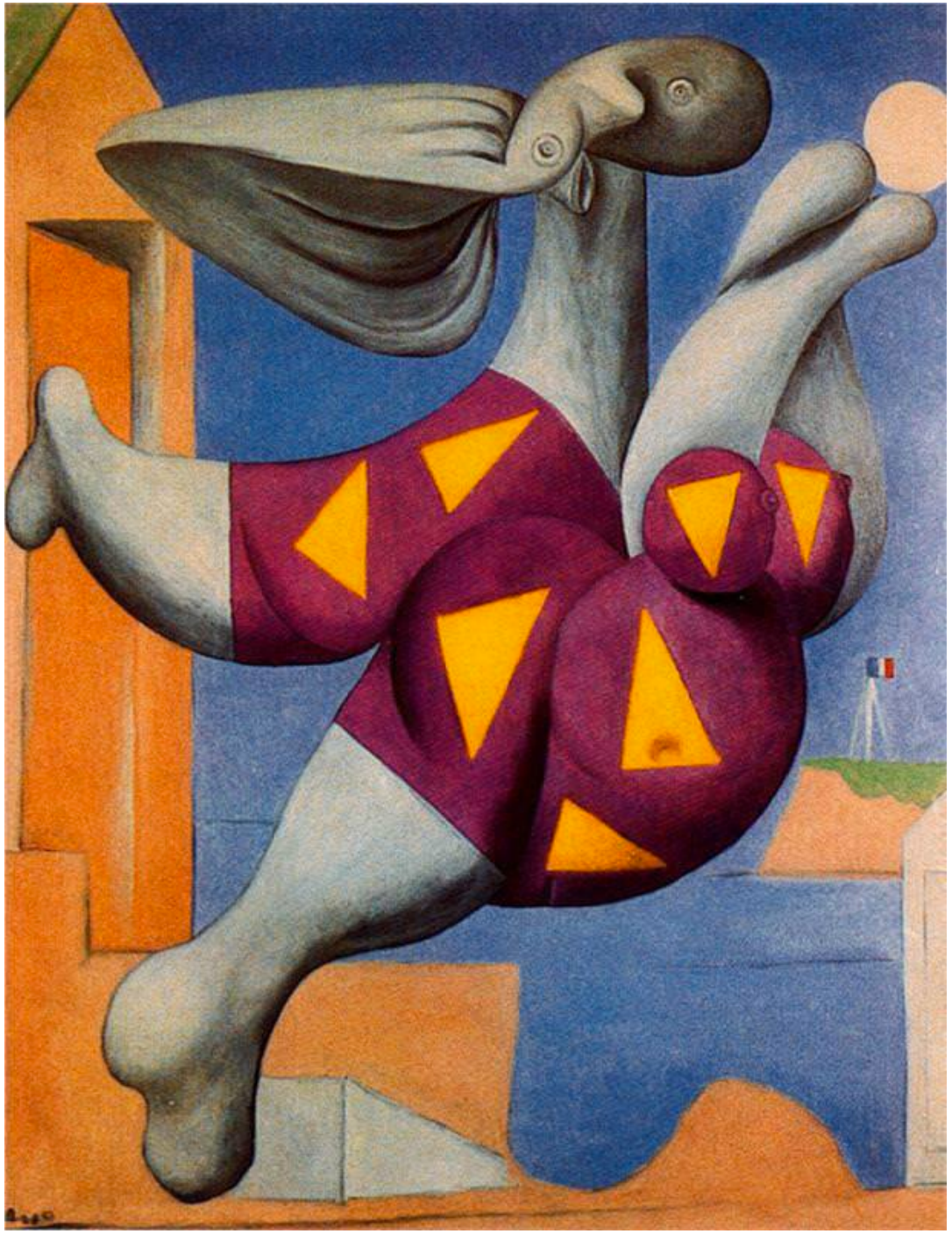
Coding exercise 1: implement a dice class



Make an arbitrarily biased dice

```
31 if __name__ == "__main__":
32     d = Dice()
33     ''' make a biased dice '''
34     d.set_bounds([0.0, 0.3, 1.0])
35     ones = 0
36     num_rolls = 10
37     for i in range(num_rolls):
38         ones += d.roll()
39     print("the dice is:", ones/float(num_rolls))
```


```
In [34]: d2 = Dice()

In [35]: d2.set_bounds([0.0, 0.7, 1.0])

In [36]: r = [d2.roll() for i in range(50)]

In [37]: sum(r)/50
Out[37]: 0.22
```



Task: sizing up your friend

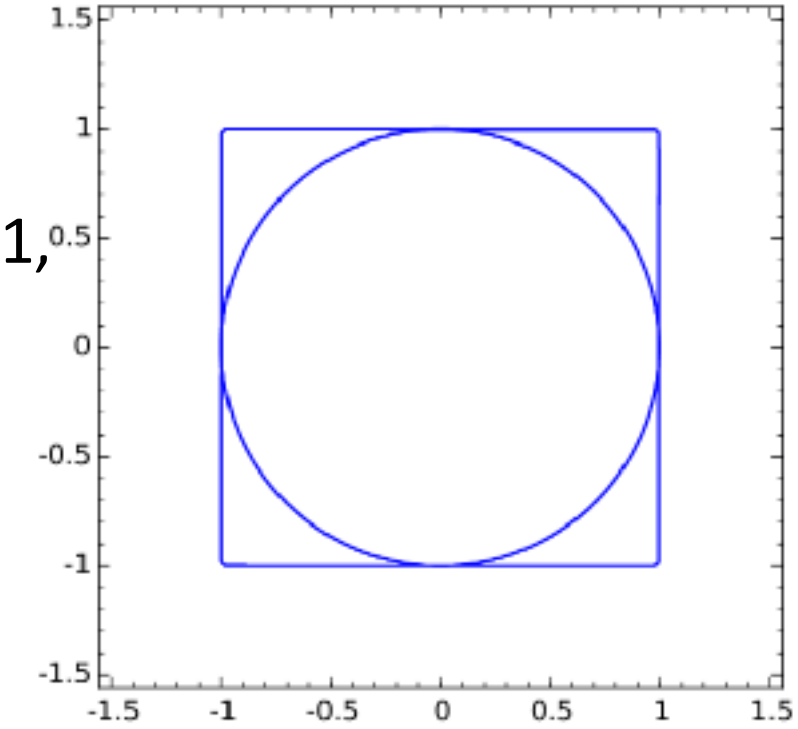
- You have infinite supply of darts

Measuring π

What you have:

- A round dish centered at 0,0 with radius 1
- A square with corners at [-1, -1], [-1, 1], [1, 1] and [1, -1]

- $\frac{S_{circle}}{S_{square}}$ tells something about π
- Can you figure it out?

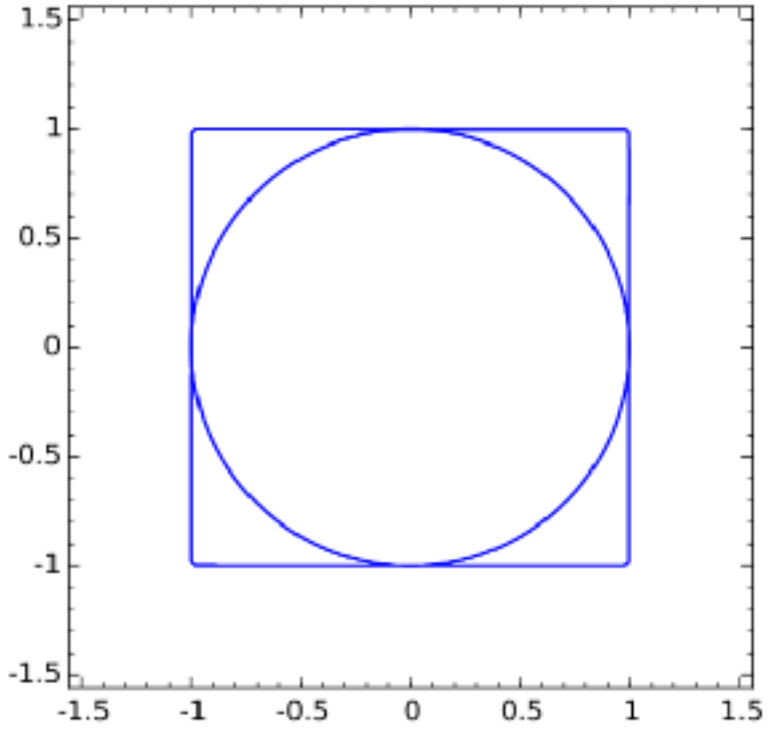


Measuring π

$$S_{square} = (2r)^2 = 4r^2$$

$$S_{circle} = \pi r^2$$

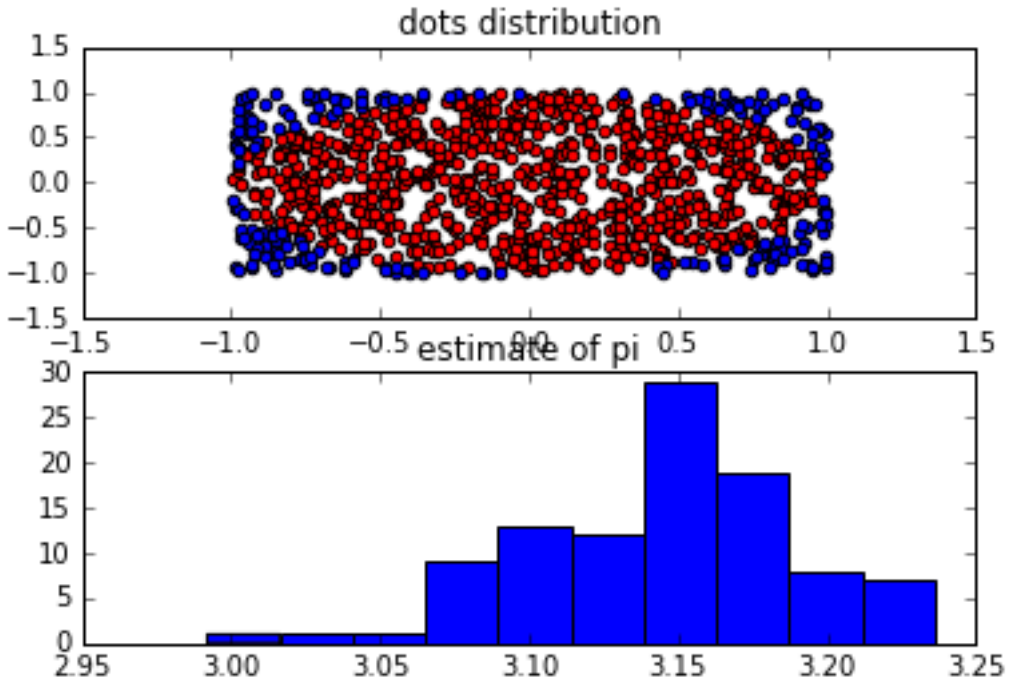
$$\pi = 4 \times (S_{circle} / S_{square})$$



Estimate $\frac{S_{circle}}{S_{square}}$ as number of times random dots land within the circle

Coding exercise 2: estimate π

```python
# compute the Eucliear distance of two vectors
def comp_dist(a, b):
    assert len(a) == len(b)
    d = 0
    for i in range(len(a)):
        d += (a[i] - b[i]) ** 2
    return math.sqrt(d)

def estimate_pi(num_points):
# replace the following line with your code
# use math.uniform(-1, 1) twice to get cordinate of a random dot
# record how many times dots are within unit circle
# should return:
# - an estimate of pi
# - the x-axis cordinates list for those within the unit circle
# - the y-axis cordinates list for those within the unit circle
# - the x-axis cordinates list for those outside the unit circle
# - the y-axis cordinates list for those outside the unit circle
    return montecalo_helper.estimate_pi(num_points)

results = []
# total number of random dots per trial
num_points_per_estimate = 1000
# perform multiple trails
num_trials = 100
for i in range(num_trials):
    est_pi, in_x, in_y, out_x, out_y = \
        estimate_pi(num_points_per_estimate)
    results.append(est_pi)

print("estimated pi: ", sum(results)/num_trials)
```

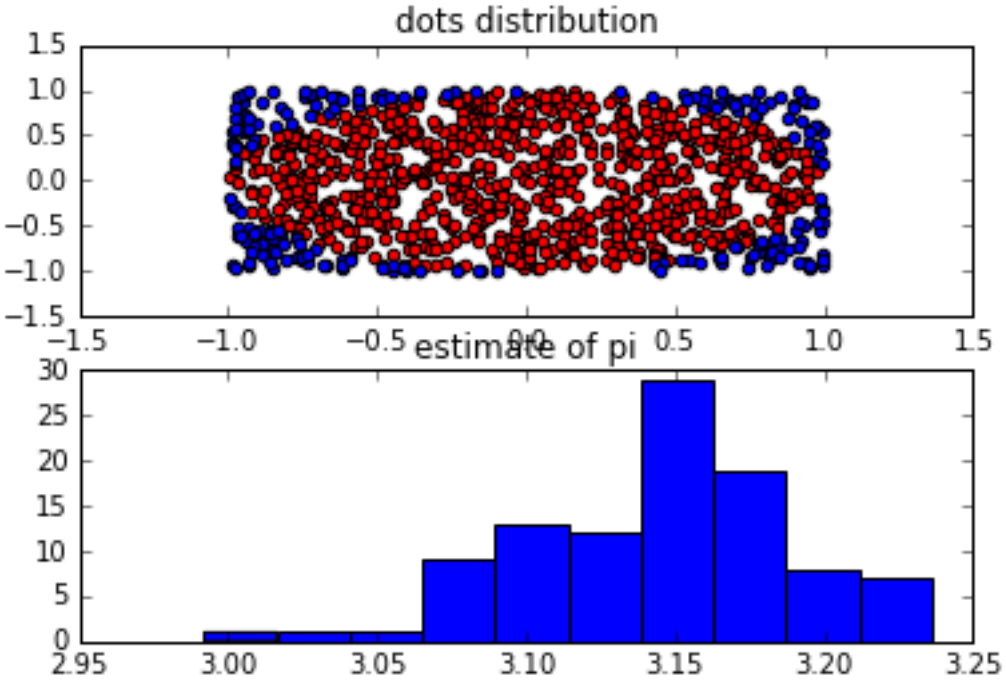


Coding exercise 2: estimate π

Two useful visualization techniques

- Scatter plot and histogram

```
45 # visualize the dots in the last trial
46 plt.subplot(2, 1, 1)
47 plt.title('dots distribution')
48 plt.scatter(in_x, in_y, c='r')
49 plt.scatter(out_x, out_y, c='b')
50
51 # visualize the estimations of all trials
52 plt.subplot(2, 1, 2)
53 plt.title('estimate of pi')
54 plt.hist(results)
```



How can you tell if a coin is *fair*?

Rudimentary statistics: mean of a random variable

- A fair coin: $P(rollout\ is\ head) = P(rollout\ is\ tail) = 0.5$
- A single flip is only 1 (for head) or 0 (for tail)
- So we throw it many times; this is called a *block of trials*

- Question: what is a reasonable length of a block?

Code 3: measure a coin

One block trial of coin throw

```
11 class Block_trial:
12     def __init__(self, sides=2, block_size=10):
13         self.dice = dice_student.Dice(sides)
14         self.block_size = block_size
15         self.outcomes = {}
16         for i in range(sides):
17             self.outcomes[i] = 0
18
19     def run(self):
20 # replace the following line with your code
21 # roll the dice block_size number of times, recode in
22 # the outcome dictionary
23         coin_helper.run(self)
24
25     def get_run_stats(self):
26 # report the statistics in a list
27 # the i-th entry of the list is frequency/percentage
28 # the coin lands on the i-th side
29         return coin_helper.get_run_stats(self)
```

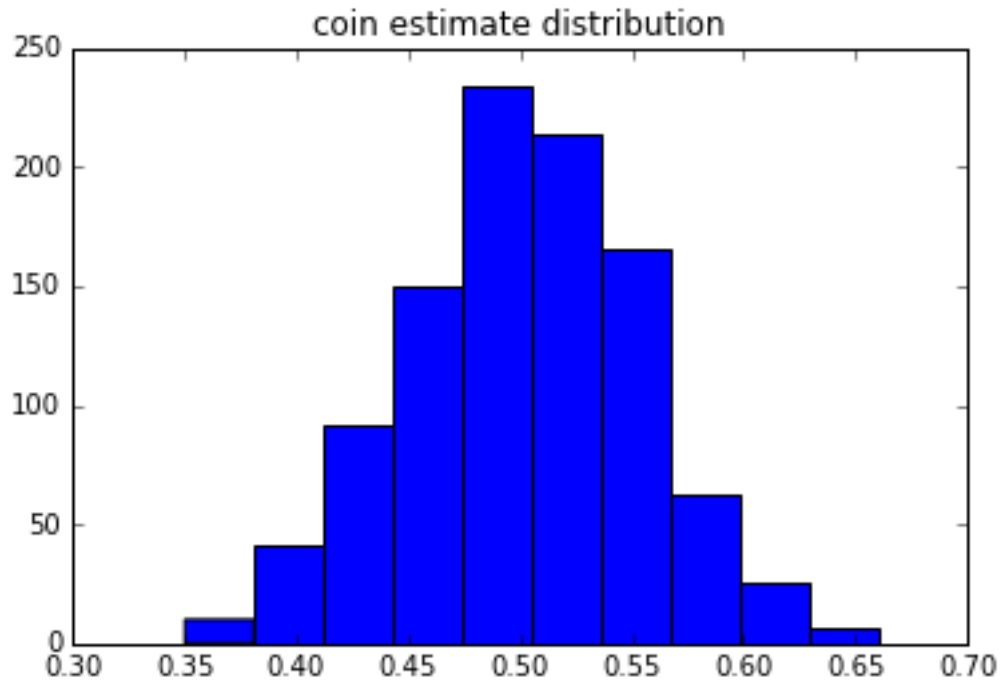
```
31 if __name__ == "__main__":
32     a_trial = Block_trial()
33     a_trial.run()
34     print(a_trial.get_run_stats())
```

```
In [45]: t = Block_trial(2, 100)

In [46]: t.run()

In [47]: t.get_run_stats()
Out[47]: [0.43, 0.57]
```

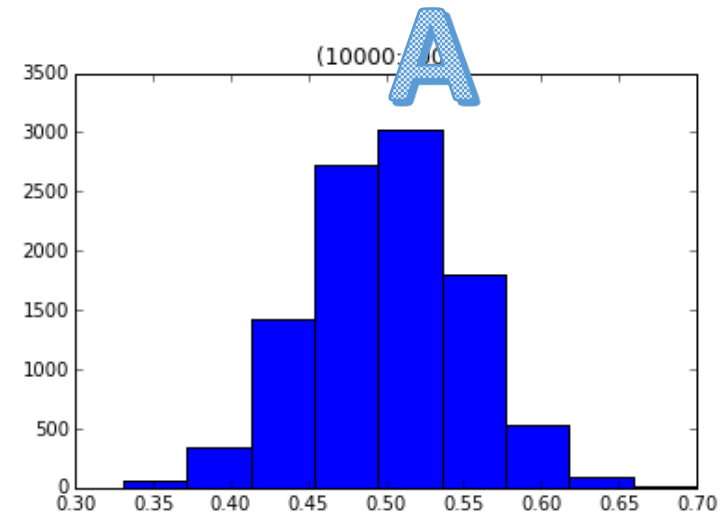
Code 3: measure a coin

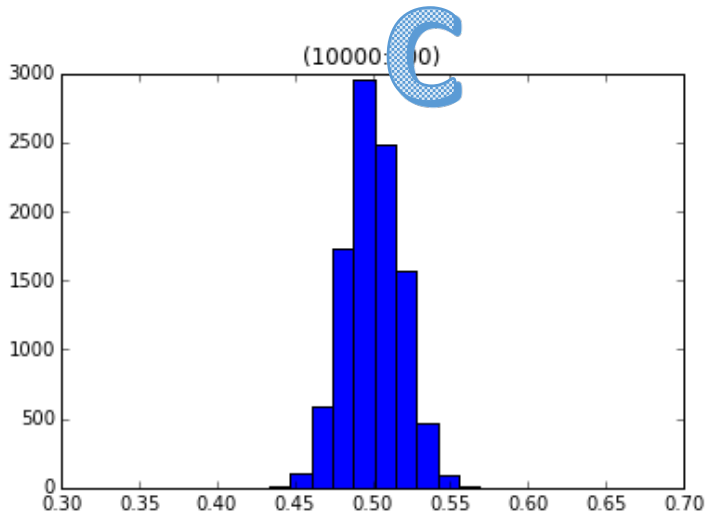
- Make many “block trials”
- What’s the effect of changing block size and number of block trials?

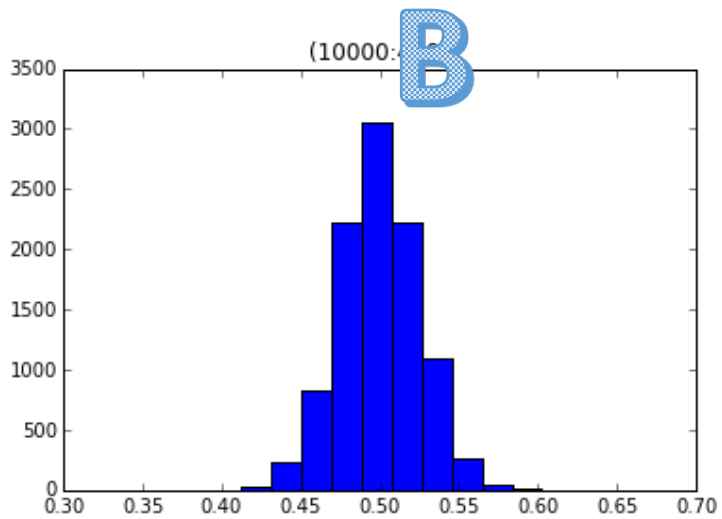
```
36    block_size = 100
37    total_blocks = 1000
38    result = []
39    for i in range(total_blocks):
40        one_trial = Block_trial(block_size=block_size)
41        one_trial.run()
42        result.append(one_trial.get_run_stats()[0])
43
44    plt.hist(result)
45    plt.title('coin estimate distribution')
46    plt.show()
```

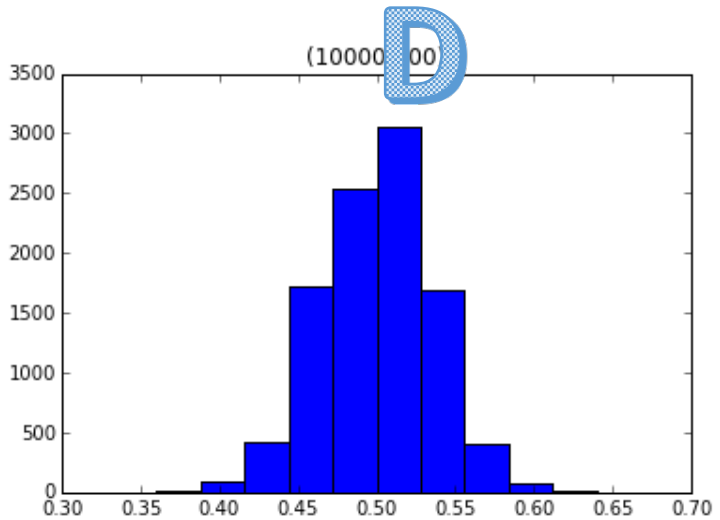


The effect of block size per trial

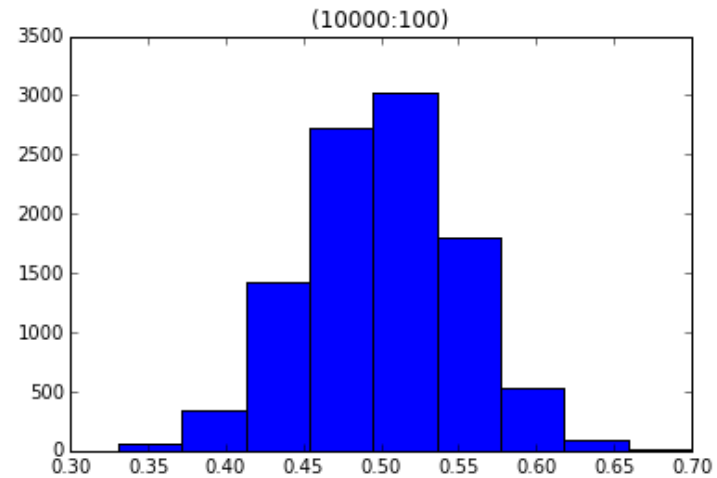


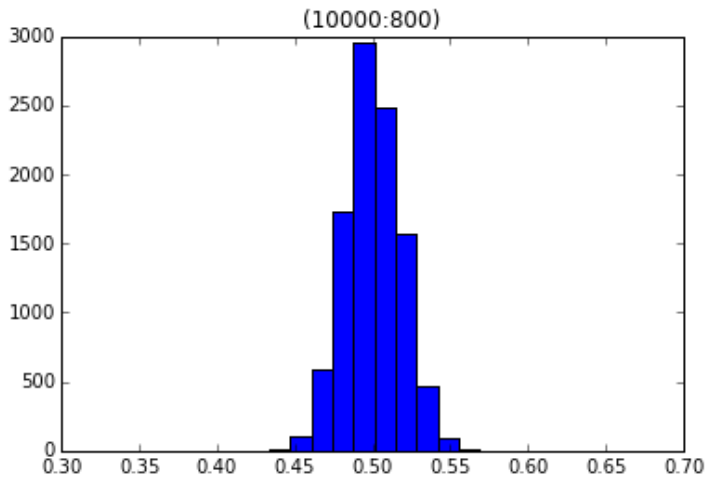


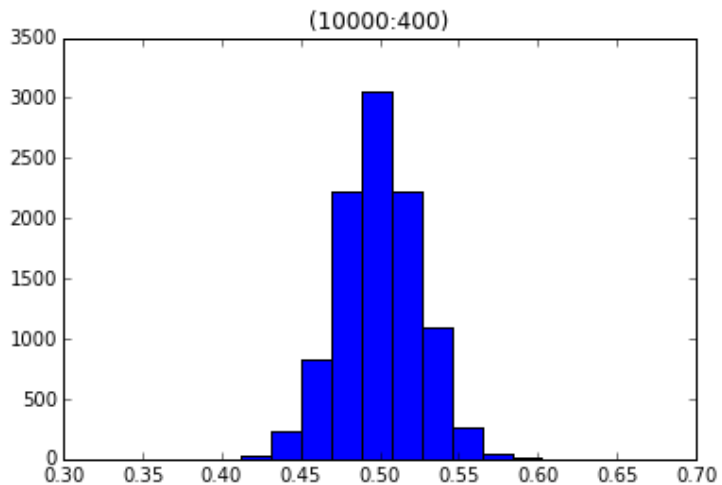


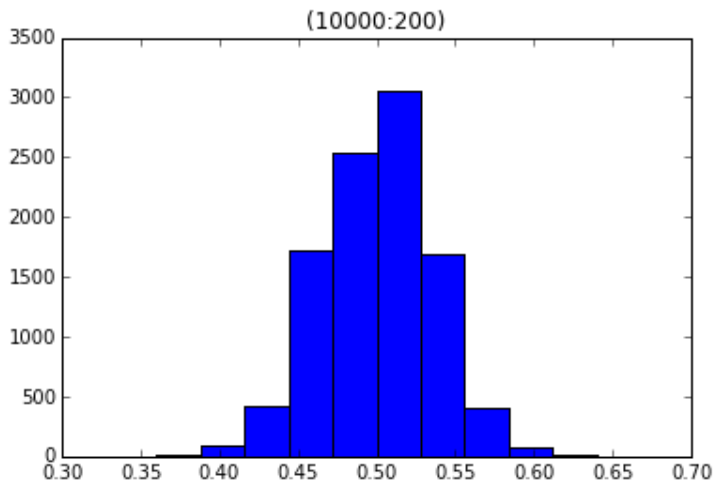


The effect of block size per trial





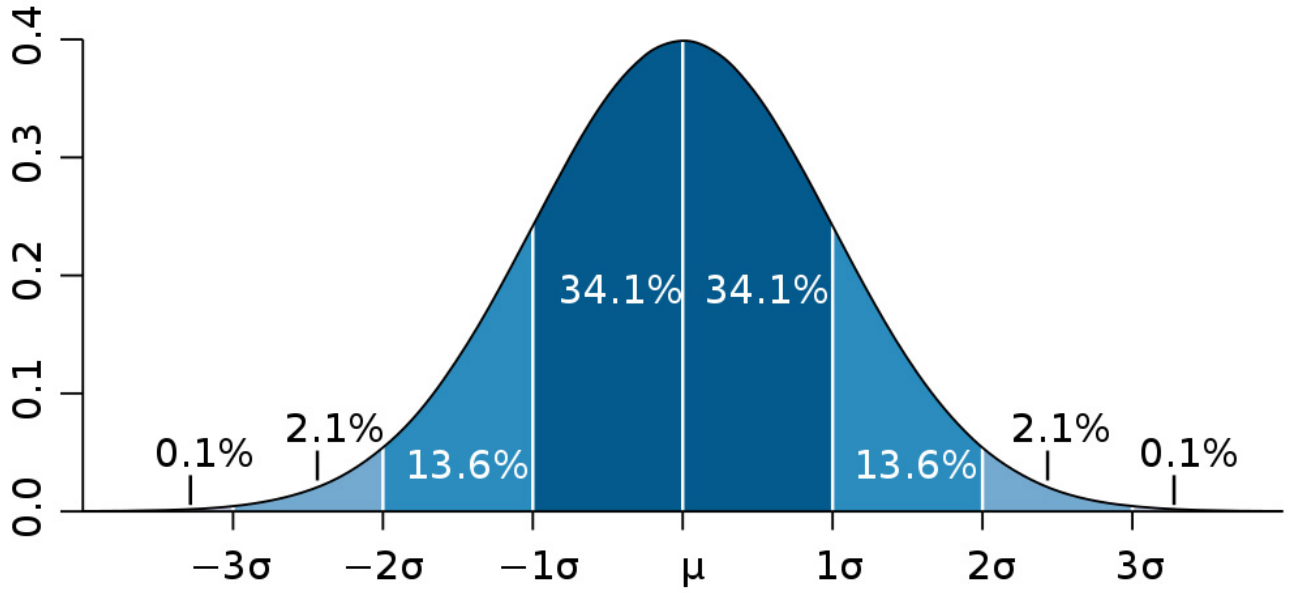




The coin estimation makes a *distribution*

Normal distribution: $N(\mu, \sigma^2)$

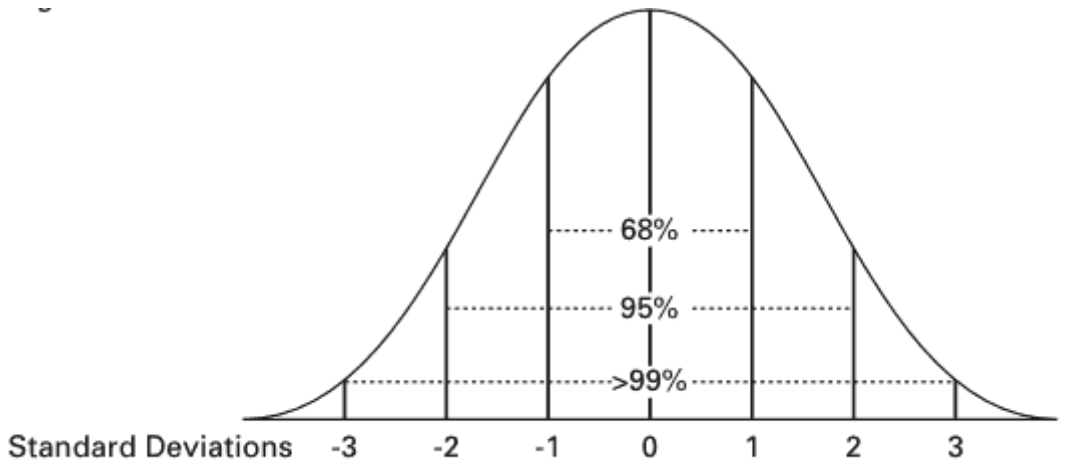
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Areas within	Percentage
1 σ	68%
2 σ	95%
3 σ	99%

Understand the *spread*: standard deviation

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$



Understand the *spread*: standard deviation

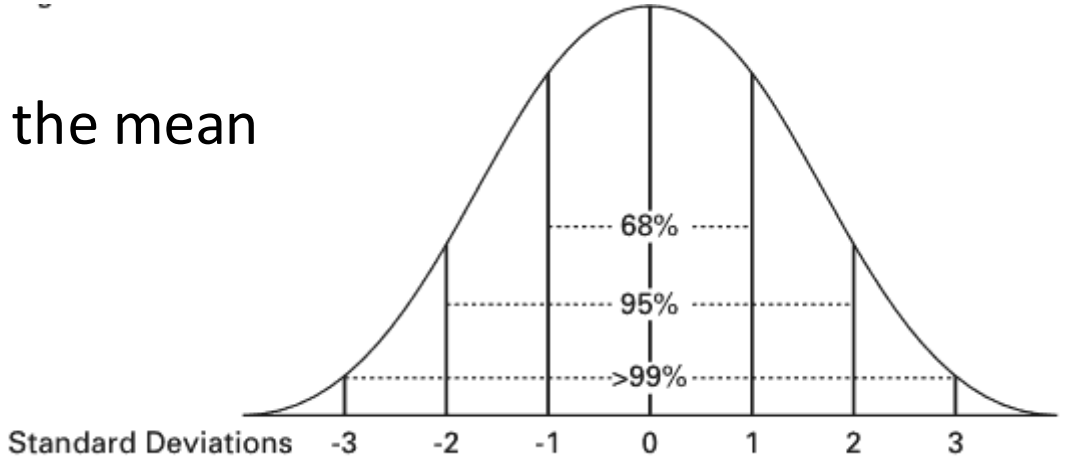
Bring back to the same unit

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Average distance to the mean

The spread (std) $\sim \frac{1}{\sqrt{n}}$

That explains why you want to throw many times!



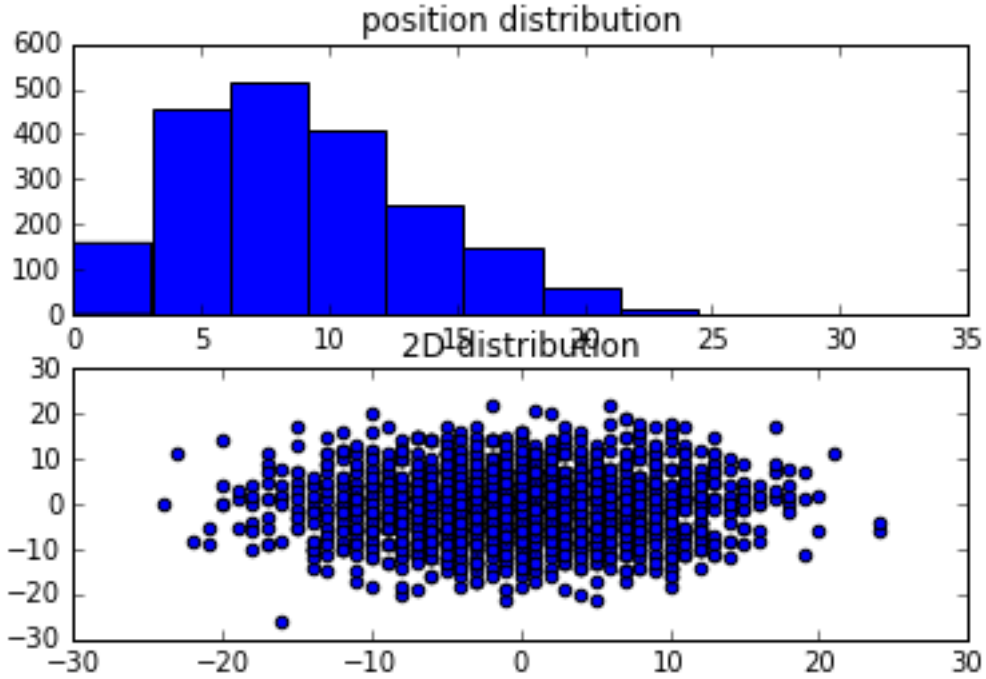
Advanced reading

- An Introduction of probability and inductive logic (Ian Hacking)
- The MIT ICS book (chapter 12)

```
class Walker:
    def __init__(self, directions=4, num_steps=10):
        self.dice = dice_student.Dice(directions)
        self.num_steps = num_steps
        self.x_pos = 0
        self.y_pos = 0

    def set_ranges(self, r):
        self.dice.set_ranges(r)

    def run(self):
# replace the following line with you code
# roll the dice, according to the outcome:
# - go left one step (x -= 1)
# - go right one step (x += 1)
# - go north one step (y += 1)
# - go south one step (y -= 1)
        random_walk_helper.run(self)

    def get_position(self):
        return self.x_pos, self.y_pos

    def get_dist(self):

        def comp_dist(a, b):
            assert len(a) == len(b)
            d = 0
            for i in range(len(a)):
                d += (a[i] - b[i]) ** 2
            return math.sqrt(d)

        return comp_dist([0, 0], [self.x_pos, self.y_pos])
```

Coding exercise 4: random walk

- Simulate with a 4-sided dice
- Mean distance ~ $\sqrt{n}$



Q&A