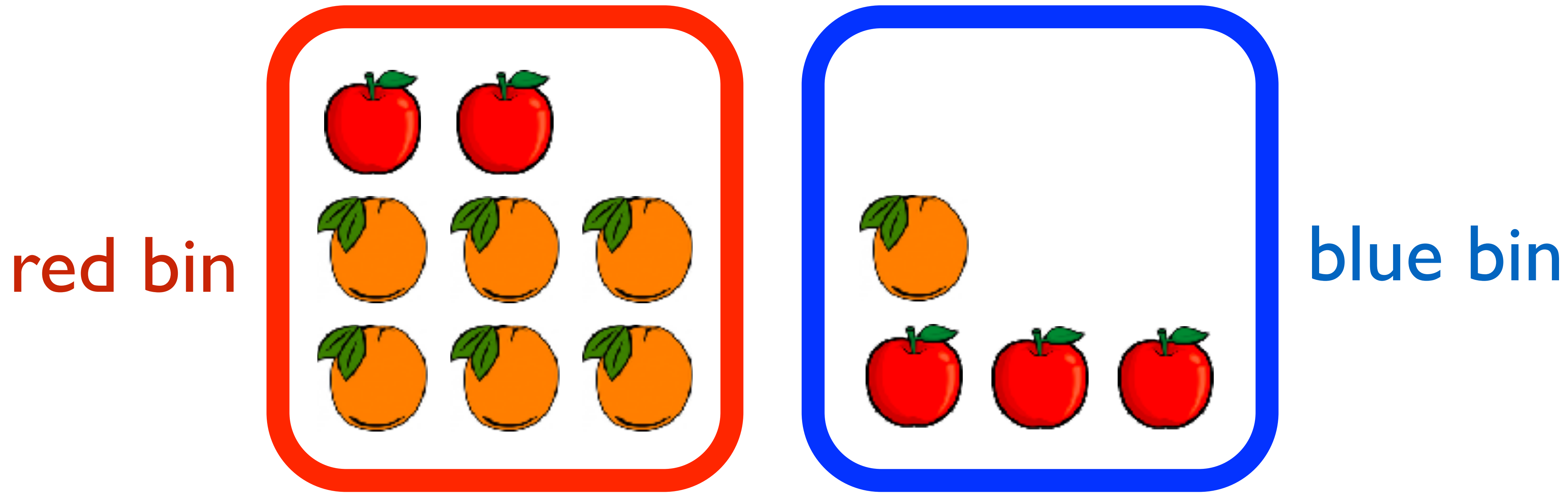
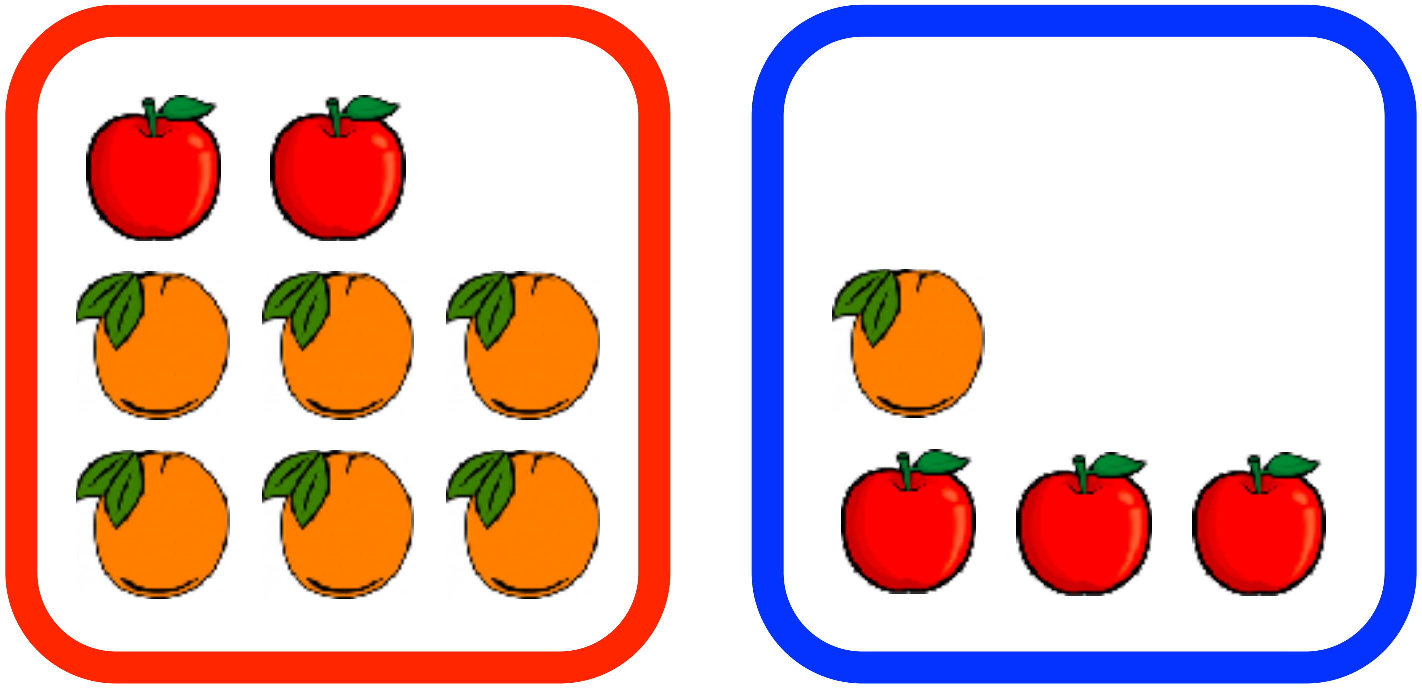
CS492: Probabilistic Programming

Posterior Inference, Basics of Anglican, and Importance Sampling

Hongseok Yang
KAIST


red bin
blue bin

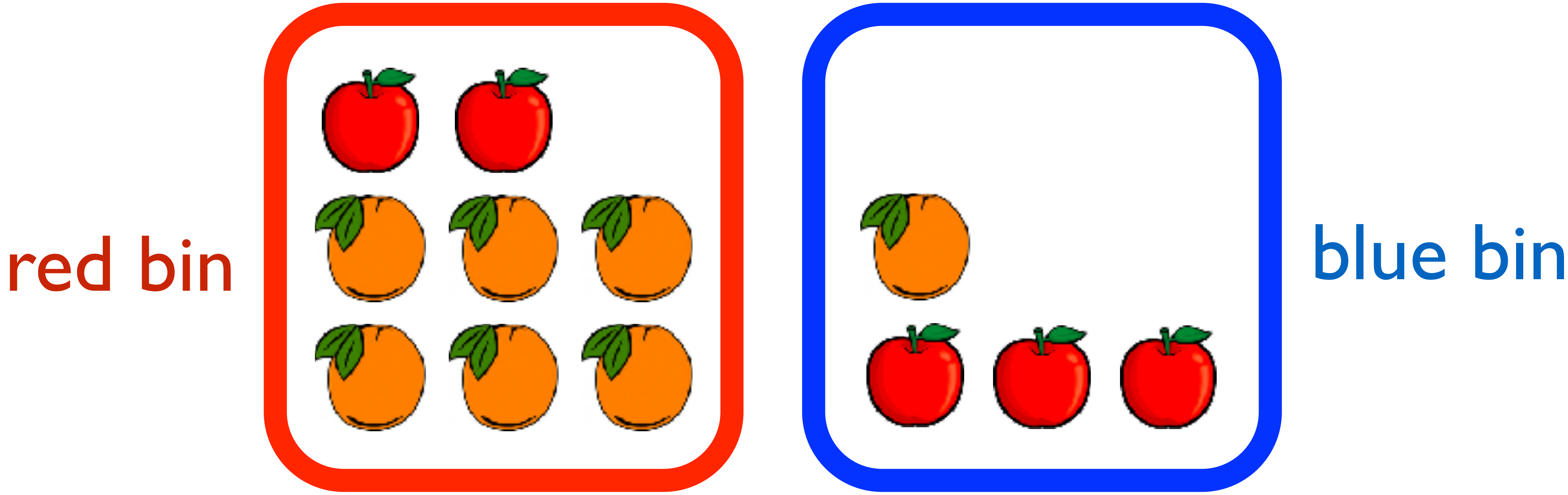
red bin



blue bin

I pick a bin.

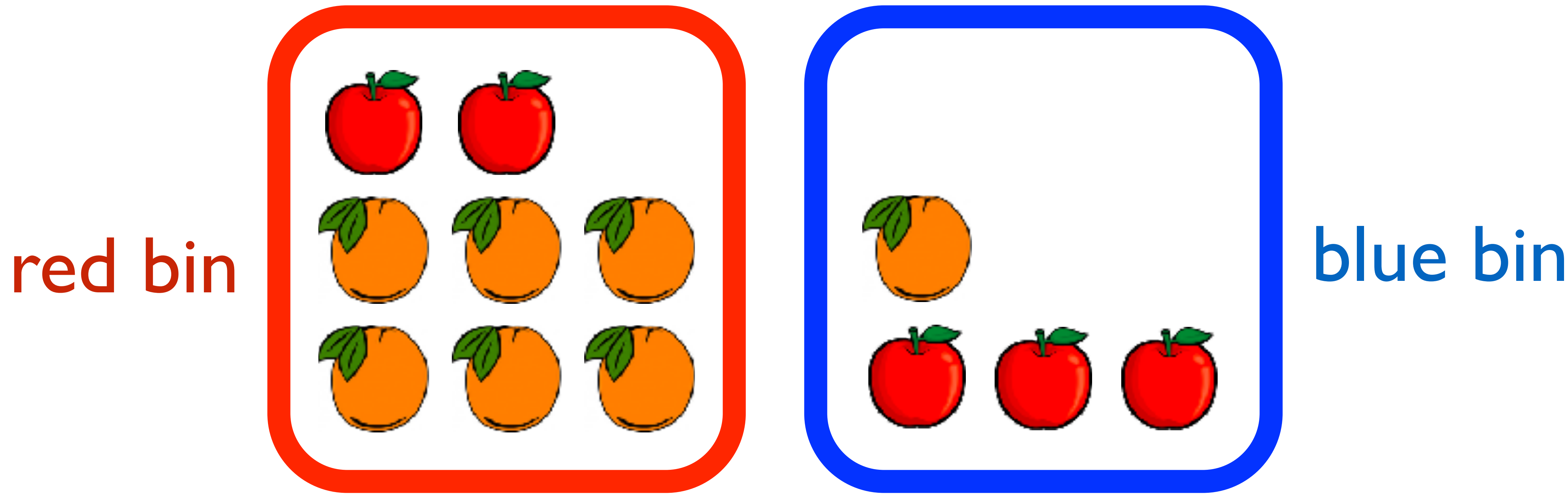
$p(\text{red}) = 1/6 \quad p(\text{blue}) = 5/6$



I pick a bin. Then, I choose a fruit from the bin.

$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$

$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$



I pick a bin. Then, I choose a fruit from the bin.

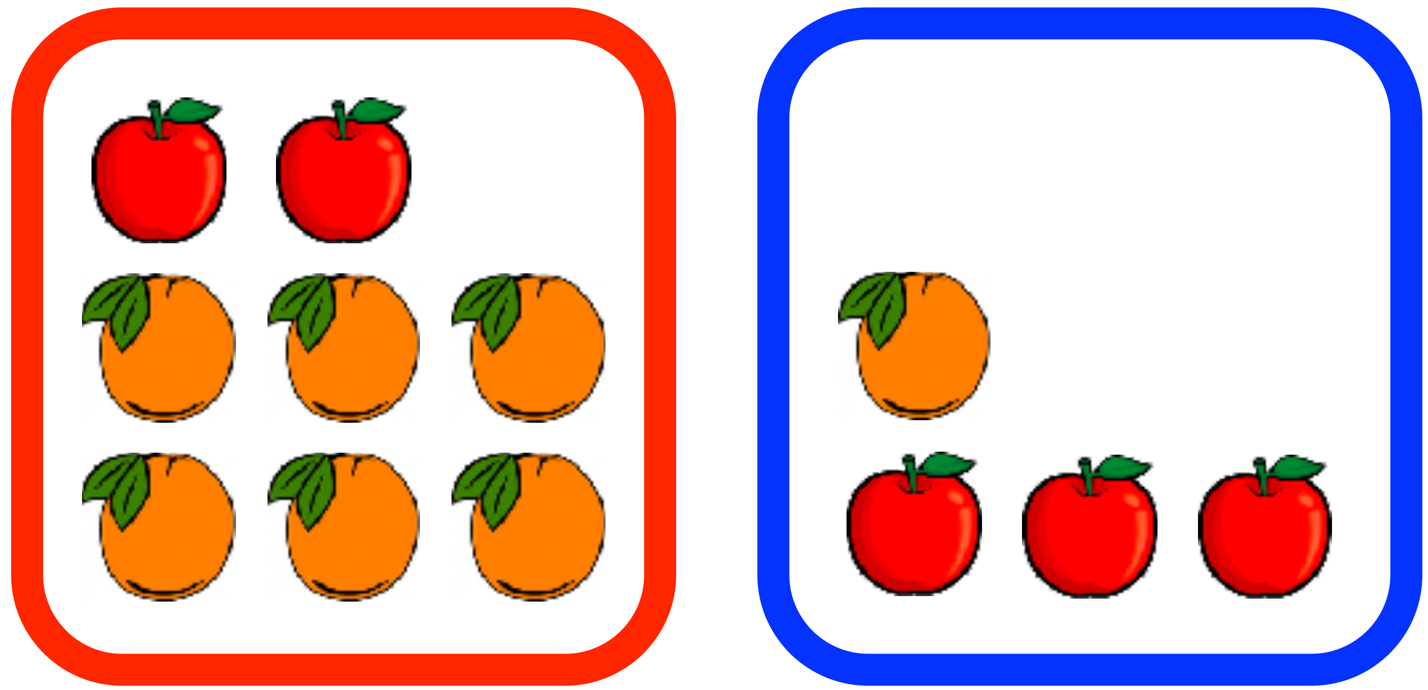
p(red) = 1/6 p(blue) = 5/6

p(apple|red) = 2/8 p(apple|blue) = 3/4

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8

red bin



blue bin

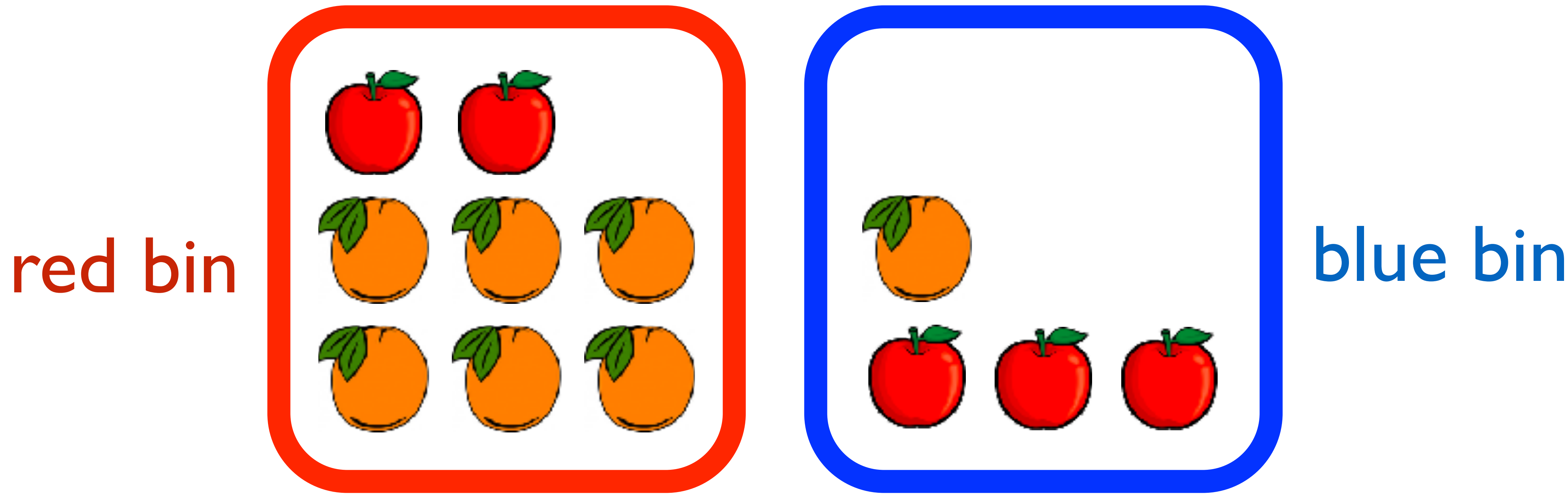
I pick a bin. Then, I choose a fruit from the bin.

$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$

$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8



I pick a bin. Then, I choose a fruit from the bin.

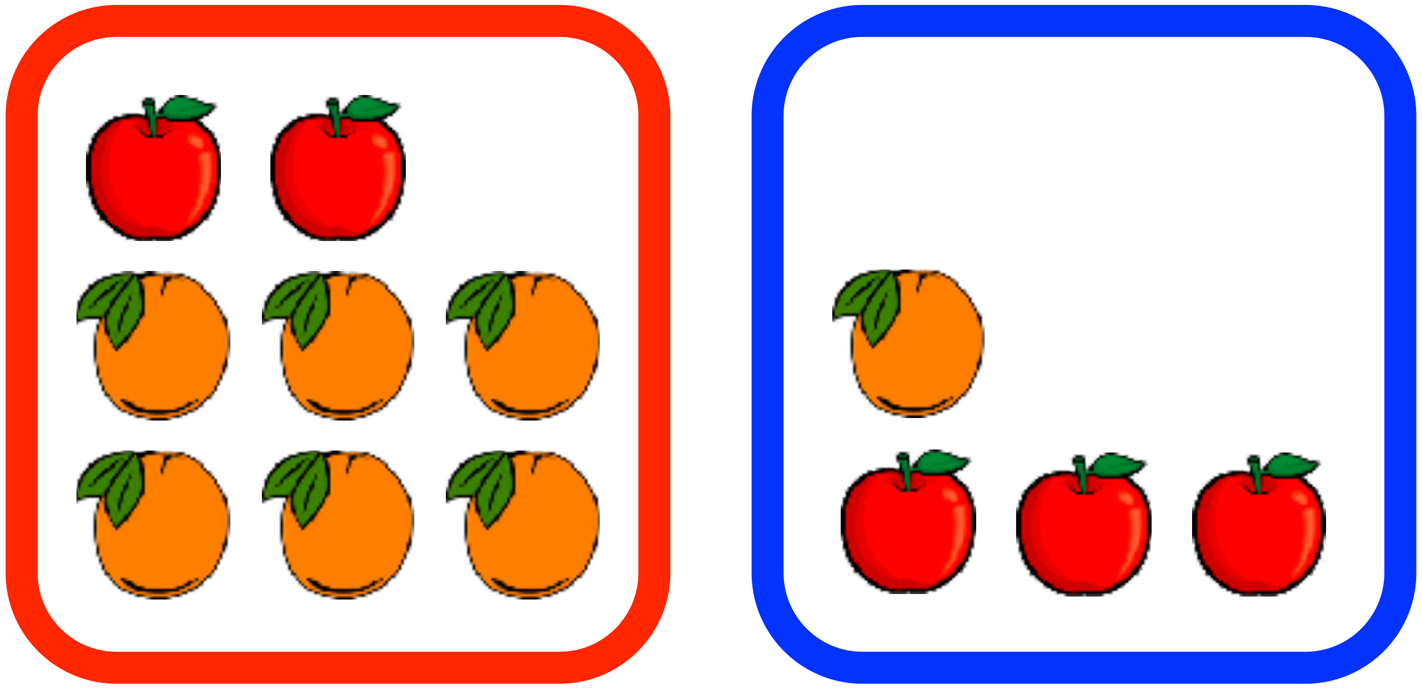
$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$

$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8

red bin



blue bin

I pick a bin. Then, I choose a fruit from the bin.

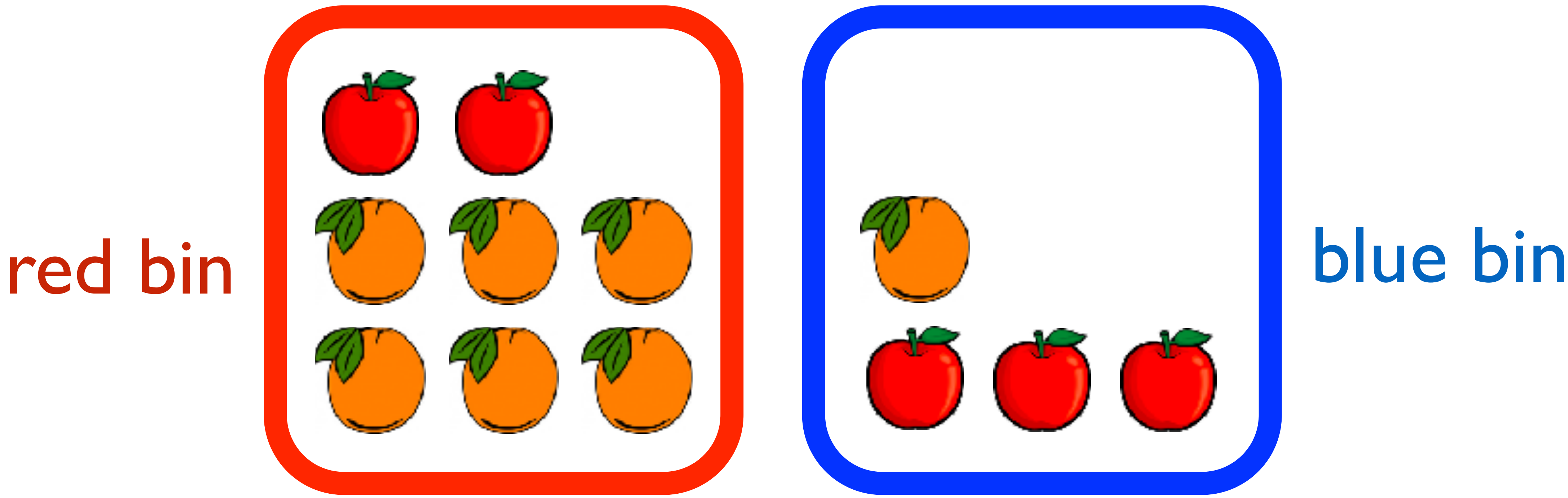
p(red) = 1/6 p(blue) = 5/6

p(apple|red) = 2/8 p(apple|blue) = 3/4

[Q p(orange|red) = 3/4 p(orange|blue) = 1/4

that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8



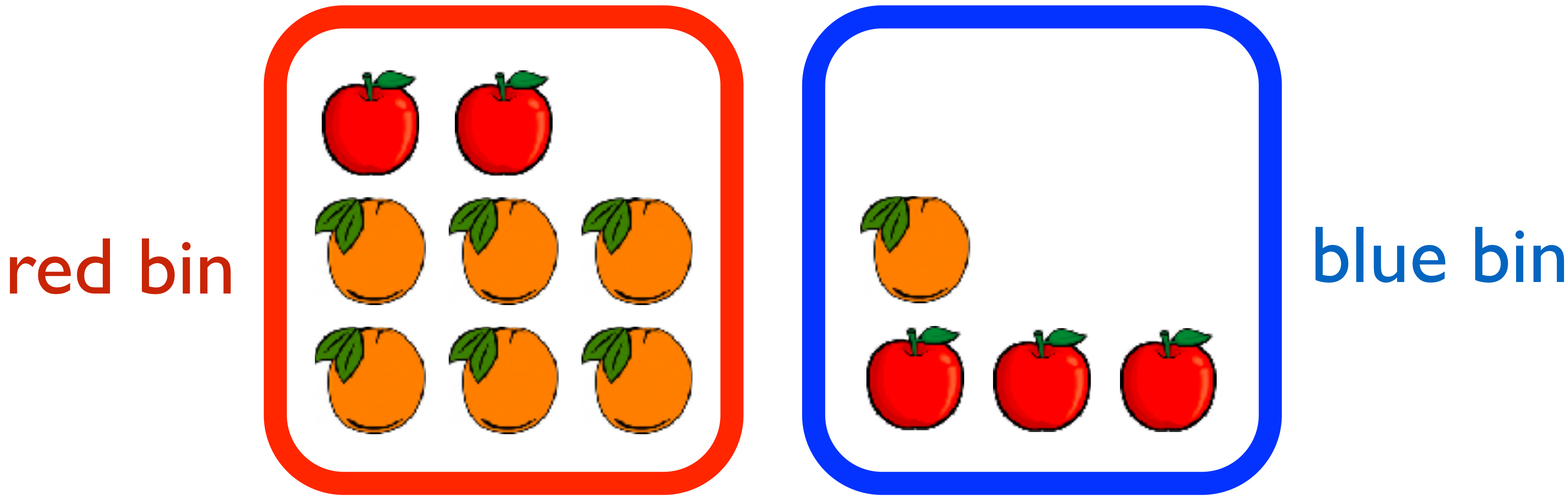
I pick a bin. Then, I choose a fruit from the bin.

$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$

$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8



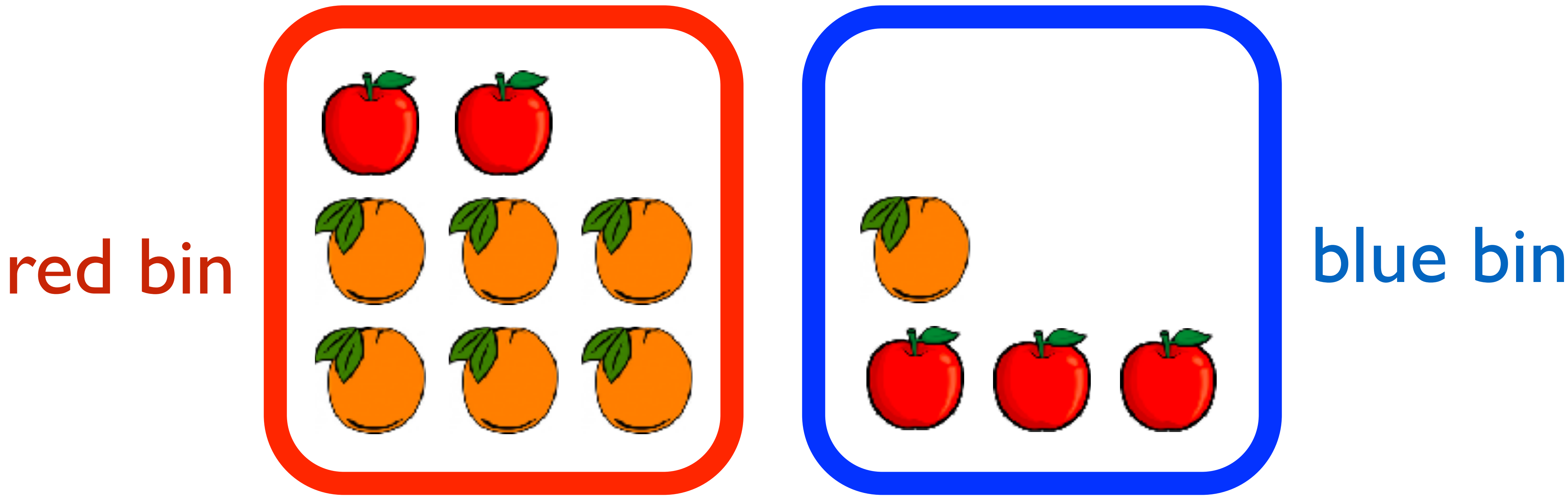
I pick a bin. Then, I choose a fruit from the bin.

$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$

$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6 2) 1/4 3) 5/8



I pick a bin. Then, I choose a fruit from the bin.

p(red) = 1/6 p(blue) = 5/6

p(apple|red) = 2/8 p(apple|blue) = 3/4

[Q] If I pick an orange, what is the probability that I picked the blue bin?

1) 5/6

2) 1/4

3) 5/8

Learning outcome

- Can describe prior, likelihood, posterior, Bayes' rule.
- Can solve the puzzle using Bayes' rule
- Can express/solve the puzzle in Anglican.
- Can explain importance sampling.

We will use discrete probabilities mostly.

Review of discrete probability, and posterior inference

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

$$p(x=0,y=0) = 1/24 \qquad p(x=0,y=1) = 3/24$$
$$p(x=1,y=0) = 5/24 \qquad p(x=1,y=1) = 15/20$$
$$p(x=0) = 4/24 \qquad p(x=1) = 20/24$$

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

p(x=0,y=0) = 1/24	p(x=0,y=1) = 3/24
p(x=1,y=0) = 5/24	p(x=1,y=1) = 15/20
p(x=0) = 4/24	p(x=1) = 20/24

[Requirement 1] $\sum_{v,w}$ p(x=v, y=w) = 1.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

p(x=0,y=0) = 1/24	p(x=0,y=1) = 3/24
p(x=1,y=0) = 5/24	p(x=1,y=1) = 15/20
p(x=0) = 4/24	p(x=1) = 20/24

[Requirement 1] $\sum_{v,w}$ p(x=v, y=w) = 1.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

p(x=0,y=0) = 1/24 p(x=0,y=1) = 3/24

p(x=1,y=0) = 5/24 p(x=1,y=1) = 15/20

p(x=0) = 4/24 p(x=1) = 20/24

[Requirement 1] $\sum_{v,w}$ p(x=v, y=w) = 1.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

$$p(x=0,y=0) = 1/24 \qquad p(x=0,y=1) = 3/24$$
$$p(x=1,y=0) = 5/24 \qquad p(x=1,y=1) = 15/20$$
$$p(x=0) = 4/24 \qquad p(x=1) = 20/24$$

[Requirement 1] $\sum_{v,w} p(x=v, y=w) = 1$.

[Requirement 2] $p(x=v) = \sum_{w} p(x=v, y=w)$.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

$p(x=0,y=0) = 1/24$	$p(x=0,y=1) = 3/24$
$p(x=1,y=0) = 5/24$	$p(x=1,y=1) = 15/20$
$p(x=0) = 4/24$	$p(x=1) = 20/24$

[Requirement 1] $\sum_{v,w} p(x=v, y=w) = 1$.

[Requirement 2] $p(x=v) = \sum_{w} p(x=v, y=w)$.

- Consider random variables x,y,z,… having values in countable sets, such as {true,false} and $\mathbb{N}$.
- Probability p assigns numbers between 0 and 1 for all possible value assignments of some variables.

$$p(x=0,y=0) = 1/24 \qquad p(x=0,y=1) = 3/24$$
$$p(x=1,y=0) = 5/24 \qquad p(x=1,y=1) = 15/20$$
$$p(x=0) = 4/24 \qquad p(x=1) = 20/24$$

[Requirement 1] $\sum_{v,w} p(x=v, y=w) = 1$.

[Requirement 2] $p(x=v) = \sum_{w} p(x=v, y=w)$.

[Q] Compute p(y=0) and p(y=1).

- Consider random variables x,y,~~z,…~~ having values in countable sets, such as {true,false} and $\mathbb{N}$

- Probability p assigns numbers betw all possible value assignments of so

Enough.
Determines
p(x=v), p(y=w).

p(x=0,y=0) = 1/24 p(x=0,y=1) = 3/24
p(x=1,y=0) = 5/24 p(x=1,y=1) = 15/20

p(x=0) = 4/24 p(x=1) = 20/24

[Requirement 1] $\sum_{v,w}$ p(x=v, y=w) = 1.

[Requirement 2] p(x=v) = $\sum_{w}$ p(x=v, y=w).

[Q] Compute p(y=0) and p(y=1).

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

[Lemma 1] $\sum_v p(x{=}v \mid y{=}w) = 1$.

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

[Lemma 1] $\sum_v p(x{=}v \mid y{=}w) = 1$.

[Lemma 2] (Bayes' rule)

$$p(x{=}v \mid y{=}w) = \frac{p(y{=}w \mid x{=}v) \times p(x{=}v)}{p(y{=}w)}$$

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

[Lemma 1] $\sum_v p(x{=}v \mid y{=}w) = 1$.

[Lemma 2] (Bayes' rule)

$$p(x \quad \mid y \quad) = \frac{p(y \quad \mid x \quad) \times p(x \quad)}{p(y \quad)}$$

In sloppy but simpler popular notation.

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

[Lemma 1] $\sum_v p(x{=}v \mid y{=}w) = 1$.

[Lemma 2] (Bayes' rule)

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

In sloppy but simpler popular notation.

Conditional probability

$$p(x{=}v \mid y{=}w) =_{\text{def}} \frac{p(x{=}v, y{=}w)}{p(y{=}w)}$$

Says the prob. of x=v conditioned on y=w.

[Lemma 1] $\sum_v p(x{=}v \mid y{=}w) = 1$.

[Lemma 2] (Bayes' rule)

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

In sloppy but simpler popular notation.

[Q] Prove both lemmas.

Bayes' rule

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?

Bayes' rule

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.

y — observation

Bayes' rule

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.

y — observation
x — target latent variable

Bayes' rule

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.

y — observation
x — target latent variable

Bayes' rule

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, $p(x)$ & $p(y|x)$ specified (not $p(x,y)$).

y — observation
x — target latent variable

Bayes' rule

prior distribution

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, $p(x)$ & $p(y|x)$ specified (not $p(x,y)$).

y — observation
x — target latent variable

Bayes' rule

likelihood

prior distribution

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, p(x) & p(y|x) specified (not p(x,y)).

y — observation
x — target latent variable

Bayes' rule

likelihood

prior distribution

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

posterior distribution

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, p(x) & p(y|x) specified (not p(x,y)).

Bayes' rule

y — observation
x — target latent variable

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

likelihood

prior distribution

marginal likelihood

posterior distribution

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, p(x) & p(y|x) specified (not p(x,y)).

y — observation
x — target latent variable

Bayes' rule

likelihood

prior distribution

$$p(x \mid y) = \frac{p(y \mid x) \times p(x)}{p(y)}$$

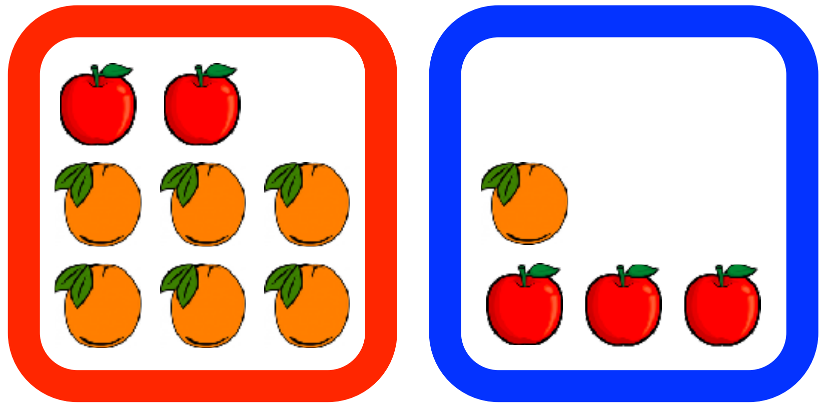
marginal likelihood

posterior distribution

posterior $\propto$ likelihood $\times$ prior

- Trivial fact. But super famous. Why?
- Says how to combine prior knowledge with observed data consistently.
- Typically, p(x) & p(y|x) specified (not p(x,y)).

Puzzle again



I pick a bin. Then, I choose a fruit from the bin.

$$p(\text{red}) = 1/6 \qquad p(\text{blue}) = 5/6$$
$$p(\text{apple}|\text{red}) = 2/8 \qquad p(\text{apple}|\text{blue}) = 3/4$$

[Q] If I pick an orange, what is the probability that I picked the blue bin?

Posterior inference

- Computation of $p(x|y)$ given i) $p(y|x)$ and $p(x)$ and ii) an observed value w of y.
- Bayes' rule and Req 2 give an algorithm:

$$p(x \mid y{=}w) = \frac{p(y{=}w \mid x) \times p(x)}{p(y{=}w)}$$

$$= \frac{p(y{=}w \mid x) \times p(x)}{\sum_v p(x{=}v, y{=}w)}$$

Posterior inference

- Computation of $p(x|y)$ given i) $p(y|x)$ and $p(x)$ and ii) an observed value w of y.
- Bayes' rule and Req 2 give an algorithm:

$$p(x \mid y{=}w) = \frac{p(y{=}w \mid x) \times p(x)}{p(y{=}w)}$$

$$= \frac{p(y{=}w \mid x) \times p(x)}{\sum_v p(x{=}v, y{=}w)}$$

Big sum for realistic models. Inefficient.

Approximate posterior inference

- Approximates posterior $p(x|y)$ using a set of samples or a simpler distribution.
- Commonly used in practice.
- Anglican implements many such algorithms.

Conditioning and posterior inference in Anglican

Conditioning in Anglican

In Anglican, we condition a model by observed random variables using the observe construct:

`(observe` *distribution-object observed-value*`)`

Examples:

```
(observe (flip p) true)
```

```
(observe
  (categorical
    {:blue p, :red q, :green r})
  :blue)
```

[Q] Write an Anglican query for our puzzle using categorical distribution.

```
(defquery puzl [fruit]
  (let [bin
                                                        ]

    bin))
```

[Q] Write an Anglican query for our puzzle using categorical distribution.

```
(defquery puzl [fruit]
  (let [bin
            . . . . .
                                    ]

    .    .    .    .    .

    bin))
```

```
(observe
  (categorical
    {:blue p, :red q, :green r})
  :blue)
```

[Q] Write an Anglican query for our puzzle using categorical distribution.

```
(defquery puzl [fruit]
  (let [bin (sample (categorical
                          {:red (/ 1 6),
                           :blue (/ 5 6)}))]
    (if (= bin :red)



      ●     ●     ●     ●     ●


    bin))
```

```
(observe
  (categorical
    {:blue p, :red q, :green r})
  :blue)
```

[Q] Write an Anglican query for our puzzle using categorical distribution.

```
(defquery puzl [fruit]
  (let [bin (sample (categorical
                      {:red (/ 1 6),
                       :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
```

Anglican function.
Performs inference.
Returns a lazy infinite sequence of Clojure maps.

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(println (first x))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(println (first x))
```

```
{:log-weight -1.3862943611198906,
:result :blue, :predicts []}
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))
(println (count y))
(println (first (rest y)))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))
(println (count y))
(println (first (rest y)))
```

```
10000
{:log-weight -1.3862943611198906,
:result :blue, :predicts []}
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))

(defn f [m] (exp (:log-weight m)))
(defn g [m]
  (if (= (:result m) :blue) (f m) 0.0))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))

(defn f [m] (exp (:log-weight m)))
(defn g [m]
  (if (= (:result m) :blue) (f m) 0.0))

(/ (reduce + (map g y))
   (reduce + (map f y)))
```

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))

(defn f [m] (exp (:log-weight m)))
(defn g [m]
  (if (= (:result m) :blue) (f m) 0.0))

(/ (reduce + (map g y))
   (reduce + (map f y)))
```

[Q] Does anyone see what goes on here?

We perform approximate posterior inference using the importance-sampling algo. of Anglican.

```
(def x (doquery :importance puzl [:orange]))
(def y (take 10000 x))

(defn f [m] (exp (:log-weight m)))
(defn g [m]
  (if (= (:result m) :blue) (f m) 0.0))

(/ (reduce + (map g y))
   (reduce + (map f y)))
```

[Q] Does anyone see what goes on here?

[A] Portion of (weighted) blue samples among all (weighted) samples.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

posterior

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior $p(x)$.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior $p(x)$.
2. Compute weight $w_i = p(y|x_i)$ for each i.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior $p(x)$.
2. Compute weight $w_i = p(y|x_i)$ for each i.
3. Return weighted avg. $(\sum_i w_i \times f(x_i)) / \sum_j w_j$.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior $p(x)$.
2. Compute weight $w_i = p(y|x_i)$ for each i.
3. Return weighted avg. $(\sum_i w_i \times f(x_i)) / \sum_j w_j$.

[puzl] f(x)=0 if (:result x) is :red. If :blue, f(x)=1.

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior $p(x)$.
2. Compute weight $w_i = p(y|x_i)$ for each i.
3. Return weighted avg. $(\sum_i w_i \times f(x_i)) / \sum_j w_j$.

[puzl] $f(x)=0$ if (:result x) is :red. If :blue, $f(x)=1$.

[Q1] Why is this a sensible algorithm?

Likelihood weighted importance sampling

[Goal] Estimate $\mathbb{E}_{p(x|y)}[f(x)]$ for a given f.

1. Sample $x_1, \ldots, x_N$ from prior p(x).
2. Compute weight $w_i = p(y|x_i)$ for each i.
3. Return weighted avg. $(\sum_i w_i \times f(x_i)) / \sum_j w_j$.

[puzl] f(x)=0 if (:result x) is :red. If :blue, f(x)=1.

[Q1] Why is this a sensible algorithm?

[Q2] How to implement 1 & 2 for Anglican queries?

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

1. Use a global variable w initialised to 1.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

1. Use a global variable w initialised to 1.
2. Run Q as usual except for `sample` & `observe`.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

1. Use a global variable w initialised to 1.
2. Run Q as usual except for `sample` & `observe`.
   a. (`sample` *dist*): draw a sample from *dist*.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

1. Use a global variable w initialised to 1.
2. Run Q as usual except for `sample` & `observe`.
   a. (`sample` *dist*): draw a sample from *dist*.
   b. (`observe` *dist v*): update $w := w \times p_{dist}(v)$.

[Input] N and a Anglican query Q.

[Output] Weighted samples $(w_1,s_1),\ldots,(w_N,s_N)$.

Run Q as follows for N times.

1. Use a global variable w initialised to 1.
2. Run Q as usual except for `sample` & `observe`.
   a. (`sample` *dist*): draw a sample from *dist*.
   b. (`observe` *dist v*): update $w := w \times p_{dist}(v)$.
3. Return w and the result s of Q.

fruit = :orange

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0

:blue

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0

bin = :blue

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0

bin = :blue

```clojure
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0

bin = :blue

fruit = :orange

w = 1.0

bin = :blue

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0 × 0.25

bin = :blue

```
(defquery puzl [fruit]
  (let [bin (sample
              (categorical
                {:red (/ 1 6),
                 :blue (/ 5 6)}))]
    (if (= bin :red)
      (observe (categorical
                 {:apple (/ 2 8),
                  :orange (/ 6 8)})
               fruit)
      (observe (categorical
                 {:apple (/ 3 4),
                  :orange (/ 1 4)})
               fruit))
    bin))
```

fruit = :orange

w = 1.0×0.25

bin = :blue

Thus, returns (0.25, :blue)

Likelihood weighted importance sampling

- Simple.
- Regarded as a semi-official semantics for Anglican and other probabilistic PLs.
- OK, but inefficient. Can you guess why?

Summary

- Learnt posterior inference using Bayes' rule in the context of discrete probabilities.
- In Anglican, we can condition using observe and perform posterior inference.
- Discussed the likelihood weighted importance sampling algorithm.