Probabilistic & Approximate Computing

Sasa Misailovic

UIUC

Exact Inference

Naïve approach: Compute $P(x_1, x_2, \dots, x_n)$

Better approach:

Take advantage of (conditional) independencies

- Whenever we can expose conditional independence, e.g., $\mathrm{P}(x_1, \mathrm{x}_2|\mathrm{x}_3) = \mathrm{P}(\mathrm{x}_1|\mathrm{x}_3) \cdot \mathrm{P}(\mathrm{x}_2|\mathrm{x}_3)$ the computation is more efficient

Compute distributions from parents to children

Complexity of Exact Inference

Number of variables: $\boldsymbol{n}$

Naïve enumeration: complexity is $O(2^n)$

Variable Elimination: if the maximum number of parents of the nodes is $k \in \{1, \dots, n\}$, then the complexity is $n \cdot O(2^k)$.

For many models this is a good improvement, but

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
```

(

X	Y
True	--

, 0.7), (

X	Y
False	--

, 0.3)

```
Y := not X;
```

(

X	Y
True	False

, 0.7), (

X	Y
False	True

, 0.3)

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
```

(

X	Y
True	--

, 0.7), (

X	Y
False	--

, 0.3)

```
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
condition ( X == True );
return Y;
```

(

X	Y
True	True

, 0.49/0.7), (

X	Y
True	False

, 0.21/0.7),

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
condition ( X == Y );
return Y;
```

(

X	Y
True	True

, 0.49/0.58), (

X	Y
False	False

, 0.09/0.58)

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
factor ( X ? 0 : -1 );
```

$e^0 = 1$

$e^{-1} = 0.37$

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.077)(

X	Y
False	False

, 0.033)

Example: Bernoulli Program

factor => condition

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
factor ( X ? 0 : -∞ );
```

$e^0 = 1$

$e^{-\infty} = 0$

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0), (

X	Y
False	False

, 0)

Example: Bernoulli Program

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
factor ( X ? 0 : -1 );
```

$e^0 = 1$

$e^{-1} = 0.37$

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.077)(

X	Y
False	False

, 0.033)

Likelihood or Log-likelihood?

(

X	Y
--	--

, 0.0)

```
X := Bernoulli(0.7);
```

(

X	Y
True	--

, -0.36),(

X	Y
True	--

, -1.20)

Likelihood or Log-likelihood?

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
```

(

X	Y
True	--

, -0.36),(

X	Y
True	--

, -1.20)

```
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, -0.72),(

X	Y
True	False

, -1.56), (

X	Y
False	True

, -1.56),(

X	Y
False	False

, -2.4)

Likelihood or Log-likelihood?

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, -0.72),(

X	Y
True	False

, -1.56), (

X	Y
False	True

, -1.56),(

X	Y
False	False

, -2.4)

```
factor ( X ? 0 : -1 );
```

(

X	Y
True	True

, -0.72),(

X	Y
True	False

, -1.56),(

X	Y
False	True

, -2.56),(

X	Y
False	False

, -3.4)

Likelihood or Log-likelihood?

(

X	Y
--	--

, 1.0)

```
X := Bernoulli(0.7);
Y := Bernoulli(0.7);
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.21), (

X	Y
False	False

, 0.09)

```
factor ( X ? 0 : -1 );
```

(

X	Y
True	True

, 0.49), (

X	Y
True	False

, 0.21), (

X	Y
False	True

, 0.077)(

X	Y
False	False

, 0.033)

Continuous Models

The distributions in the program are continuous

We are computing the log-likelihood of the trace

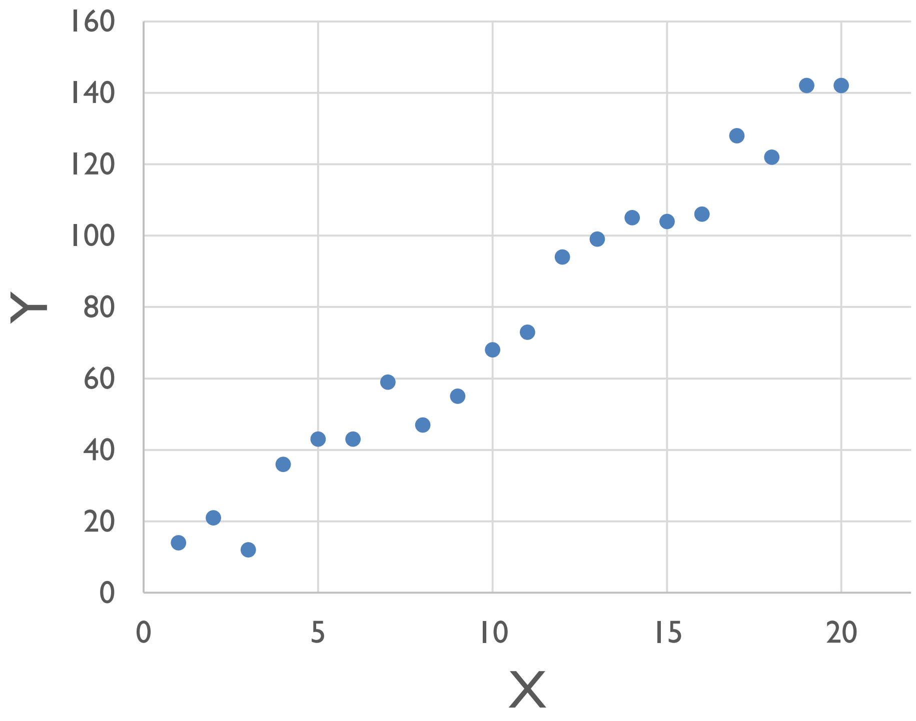
Doing ‘hard’ observations is ineffective: the probability of each observation is 0.

Instead, use ‘factor’: the ‘soft’ version of observe:

- It adds the value of the sample to the log-likelihood of the program

Continuous Models: Linear Regression

Given a set of points, find a linear relationship that most accurately describes this set



$$Y = \boldsymbol{w} \cdot X + \boldsymbol{b}$$

Slope

Intercept

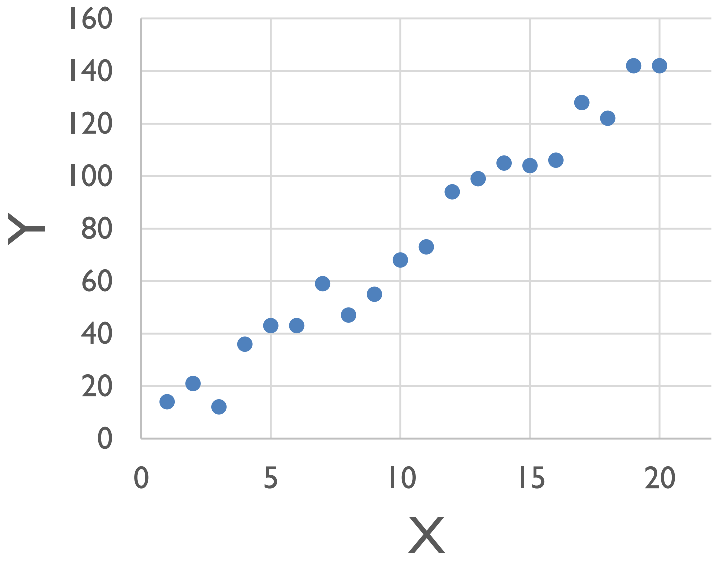
Linear Regression

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, .... ];
```

Datasets

```
w ~ Normal(6 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```

$$Y = \boldsymbol{w}.X + \boldsymbol{b}$$



Linear Regression Model

Linear Regression

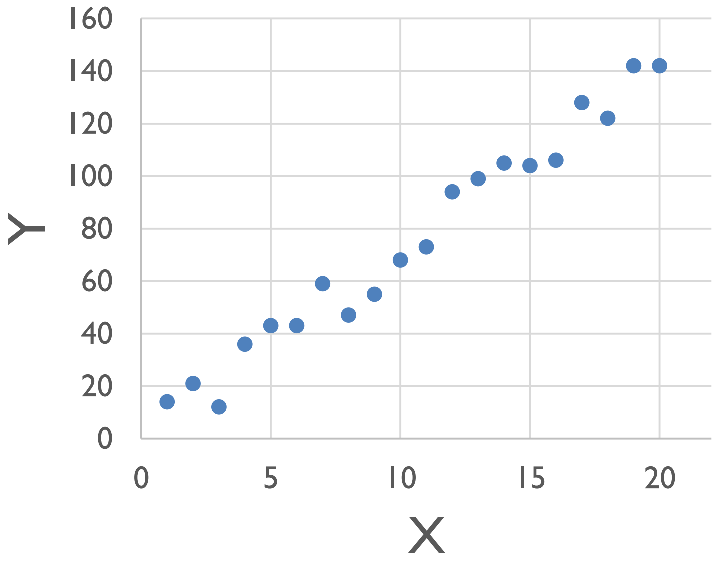
```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, .... ];
```

Datasets

```
w ~ Normal(6 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```

Priors

$$Y = \boldsymbol{w}.X + \boldsymbol{b}$$



Linear Regression Model

Linear Regression

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, .... ];
```

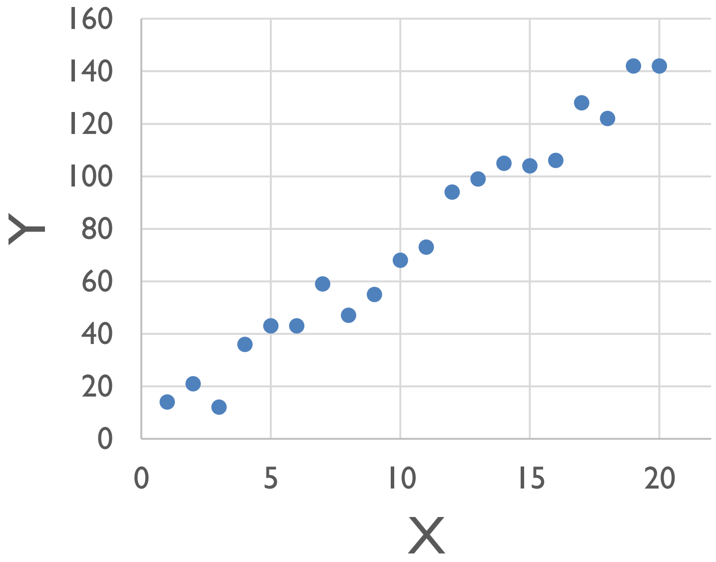
Datasets

```
w ~ Normal(6 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```

Priors

Conditioning on Data

$Y = \boldsymbol{w}.X + \boldsymbol{b}$



Linear Regression Model

Linear Regression

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, .... ];
```

Datasets

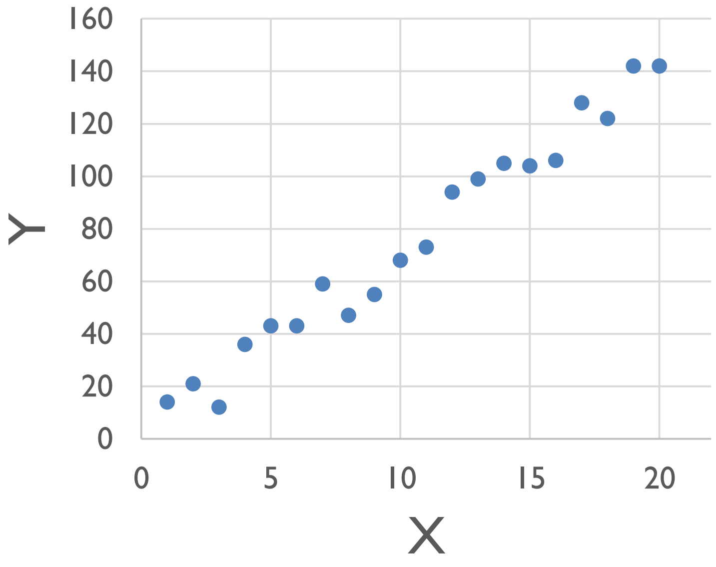
```
w ~ Normal(6 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```

Priors

Conditioning on Data

Queries

$$Y = \boldsymbol{w}.X + \boldsymbol{b}$$



Linear Regression Model

Linear Regression

```
x : [1.0, 2.0, ... ];
y : [7.01, 14.2, .... ];
```

Datasets

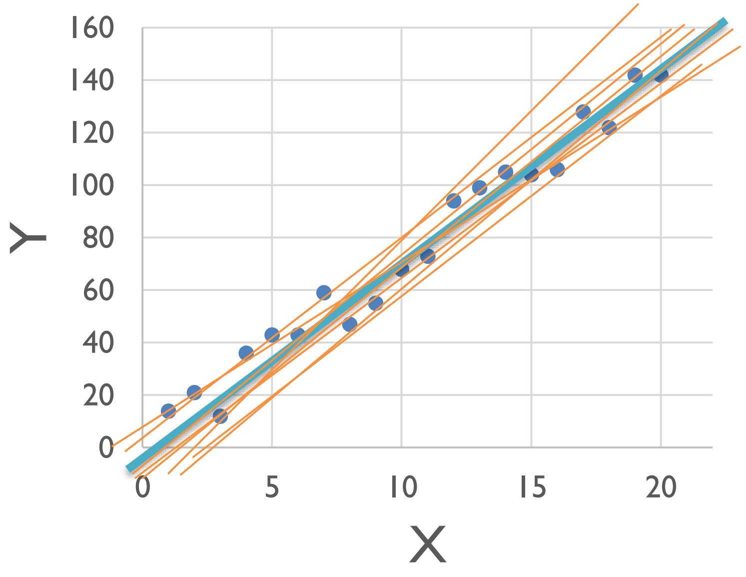
```
w ~ Normal(6 , 10);
b ~ Normal(1 , 5);
observe(y==Normal(w*x + b, 1.0));
posterior w;
posterior b;
```

Priors

Conditioning on Data

Queries

$$Y = \boldsymbol{w}.X + \boldsymbol{b}$$



Linear Regression Model

Continuous Models: Linear Regression

```
var xs = [0, 1, 2, 3];  var ys = [0, 1, 4, 6];

var model = function() {
  var slope = gaussian(0, 2);
  var intercept = gaussian(0, 2);
  var sigma = 1; // for more interesting result, change to gamma(1, 1);

  var f = function(x) { return slope * x + intercept; };

  map2(
    function(x, y) {  observe(Gaussian({mu: f(x), sigma: sigma}), y);  },
 // function(x, y) {  factor(Gaussian({mu: f(x), sigma: sigma}).score(y));  },
    xs, ys);

  return [slope,intercept];
}
viz.marginals({method: 'MCMC', samples: 10000}, model));
```

Continuous Models: TrueSkill

TrueSkill:

- Measure player skills in various sports

Each player has an unknown parameter skill that cannot be directly measured (i.e., it is hidden)

What we can observe is how the in-game performance of the player (which depends on the skill) compares to the performance of the other player

TrueSkill Model

Player skill: initially, we assume all players have similar (randomly assigned) skills, centered around some average:

$$Skill \sim Gaussian(100,10)$$

Player performance: it is based on the skill, but can be either higher or lower, depending on the moment of inspiration:

$$Perf \sim Gaussian(Skill, 15)$$

Tournament scores: Each player plays against each other, we can observe that a player with better performance won

TrueSkill Example: 3 Players

```
var trueSkill = function(){

  var skillA = gaussian(100, 10);
  var skillB = gaussian(100, 10);
  var skillC = gaussian(100, 10);

  var perfA1 = gaussian(skillA, 15), perfB1 = gaussian(skillB, 15);
  condition (perfA1 > perfB1);

  var perfB2 = gaussian(skillB, 15), perfC2 = gaussian(skillC, 15);
  condition (perfB2 > perfC2);

  var perfA3 = gaussian(skillA, 15), perfC3 = gaussian(skillC, 15);
  condition (perfA3 > perfC3);

  return skillA;
}
```

```
var res = Infer({method: 'MCMC', samples:
                    50000}, trueSkill)
print("Expected value: "+expectation(res));
viz.auto(res);
```