Sampling for General Inference

Chris Piech

CS109, Stanford University
CS109 Contest
Null Hypothesis Test

<table>
<thead>
<tr>
<th>Nepal Happiness</th>
<th>Bhutan Happiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.44</td>
<td>2.15</td>
</tr>
<tr>
<td>3.36</td>
<td>3.01</td>
</tr>
<tr>
<td>5.87</td>
<td>2.02</td>
</tr>
<tr>
<td>2.31</td>
<td>1.43</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3.70</td>
<td>1.83</td>
</tr>
</tbody>
</table>

$\mu_1 = 3.1$  
$\mu_2 = 2.4$

Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points ($p = 0.008$).
Something brand new...
Review
Collect one (or more) numbers from each person
The underlying distribution ($F$) is represented by the sample histogram, which shows the probability distribution for IID Samples $= \{X_1, X_2, \ldots, X_n\}$.
The underlying distribution \((F)\) 

IID Samples = \([20, 38, \ldots, 38]\) 

Sample histogram normalized
Sample Statistics

**Sample Mean**

\[ \bar{X} = \sum_{i=1}^{n} \frac{X_i}{n} \]

**Sample Variance**

\[ S^2 = \sum_{i=1}^{n} \frac{(X_i - \bar{X})^2}{n-1} \]

**Var of Sample Mean**

\[ \text{Var}(\bar{X}) = \frac{S^2}{n} \]

Oh my that can be thought of as a random variable.
Sample Mean

Average Happiness

\[ \text{Std}(\bar{X}) \]

Variance of Happiness

\[ \text{Std}(S^2) ? \]

Claim: The average happiness of Bhutan is 83 ± 2
You can estimate the PMF of the underlying distribution, using your sample.
Bootstrap Algorithm (sample):
1. Repeat 10,000 times:
   a. Choose \texttt{len(sample)} elems from \texttt{sample}, with replacement
   b. Recalculate the stat on the resample
2. You now have a distribution of your stat
Bootstrap for p values
### Null Hypothesis Test

<table>
<thead>
<tr>
<th>Population 1</th>
<th>Population 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.44</td>
<td>2.15</td>
</tr>
<tr>
<td>3.36</td>
<td>3.01</td>
</tr>
<tr>
<td>5.87</td>
<td>2.02</td>
</tr>
<tr>
<td>2.31</td>
<td>1.43</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3.70</td>
<td>1.83</td>
</tr>
</tbody>
</table>

\[ \mu_1 = 3.1 \]

\[ \mu_2 = 2.4 \]
End Review
Null Hypothesis Test

<table>
<thead>
<tr>
<th>Vietnam Happiness</th>
<th>Bhutan Happiness</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.44</td>
<td>2.15</td>
</tr>
<tr>
<td>3.36</td>
<td>3.01</td>
</tr>
<tr>
<td>5.87</td>
<td>2.02</td>
</tr>
<tr>
<td>2.31</td>
<td>1.43</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>3.70</td>
<td>1.83</td>
</tr>
</tbody>
</table>

$\mu_1 = 3.1$

$\mu_2 = 2.4$

Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points.
Null hypothesis: even if there is no pattern (i.e., the two samples are identically distributed) your results might have arisen by chance.
Universal Sample

- Bhutan PMF
- Nepal PMF
- Universal PMF

Piech, CS106A, Stanford University
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample

        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = \mid \mu_{\text{Nepal}} - \mu_{\text{Bhutan}} \mid
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000
def pvalueBootstrap(bhutanSample, nepalSample):
    N = size of the bhutanSample
    M = size of the nepalSample

    uniSamples = combine bhutanSamples and nepalSamples
    count = 0

    repeat 10,000 times:
        bhutanResample = draw N resamples from the uniSamples
        nepalResample = draw M resamples from the uniSamples
        muBhutan = sample mean of the bhutanResample
        muNepal = sample mean of the nepalResample
        meanDiff = |muNepal - muBhutan|
        if meanDiff > observedDifference:
            count += 1

    pValue = count / 10,000

Bootstrap for P Values
With replacement!
def drawWithReplace(samples):
    # Estimate the PMF using the samples
    # Draw K new samples from the PMF
def drawWithReplace(samples):
    # Estimate the PMF using the samples
    # Draw K new samples from the PMF
    return np.random.choice(samples, K)
Bootstrap

got assumptions?

Lets try it!

Piech, CS106A, Stanford University
Claim: The difference in happiness between Nepal and Bhutan is 0.7 happiness points (p = 0.008).
Randomized Algorithms
Bootstrapping
Thompson Sampling
Monte Carlo Integration

Generate $N$ values $(X_1, X_2, \ldots, X_N)$ uniformly sampled over a range $(a, b)$. We can approximate the integral of a function $h$ over $(a, b)$ as:

$$\int_a^b h(x) \, dx \approx \frac{b - a}{N} \sum_{i=1}^{N} h(X_i)$$

A “Monte Carlo” algorithm uses randomization but might not get the right answer.
A Rose by Any Other Name

Monte Carlo, Monaco

Las Vegas, Nevada
Something brand new...
General “Inference”
General “Inference”
General “Inference”

Flu

One inference question:

\[ P(F = 1 | N = 1, T = 1) \]

Nausea

Fever

Cold

Cancer

Undergrad

Chest Pain

Sore Throat

Tired
Another inference question:

\[ P(Co = 1, Fl = 1| S = 1, Fe = 0) \]
How Many Things Can You Condition On?

Possible conditions

\[ 3^N \]

\( N \) things:
N is large...
Simple WebMd

Flu

Undergrad

Fever

Tired
Naively specifying a joint is often impossible...
Describe the joint using causality!

$$P(Fl = a, Fe = b, U = c, T = d)$$?
Describe the joint using causality!

\[ P(Fl = 1) = 0.1 \]

\[ P(U = 1) = 0.8 \]

\[ P(Fl = 1) = 0.1 \]

\[ P(U = 1) = 0.8 \]

\[ P(Fl = 1) = 0.9 \]

\[ P(Fl = 1) = 0.1 \]

\[ P(Fl = 0) = 0.05 \]

\[ P(Fl = 0) = 0.1 \]

\[ P(Fl = 0) = 0.9 \]

\[ P(Fl = 0) = 0.8 \]

\[ P(Fl = 0) = 1.0 \]

\[ P(Fl = a, Fe = b, U = c, T = d) ? \]
If you know causality,

Make a network of causality for you random vars.

To define a joint you simply need give:

\[ P(\text{values} \mid \text{parents}) \]

for each random variable.

Prob can be a conditional probability table or an equation!
Probabilistic Model

- $P(Fl = 1) = 0.1$
- $P(Fev = 1|Flu = 1) = 0.9$
- $P(Fev = 1|Flu = 0) = 0.05$

- $P(U = 1) = 0.8$
- $P(T = 1|Flu = 0, U = 0) = 0.1$
- $P(T = 1|Flu = 0, U = 1) = 0.8$
- $P(T = 1|Flu = 1, U = 0) = 0.9$
- $P(T = 1|Flu = 1, U = 1) = 1.0$
Alg #0: Straight Math

Too many possible inference questions one could ask…
Alg #1: Joint Sampling

```python
N_SAMPLES = 100000

# Program: Joint Sample
# _______________________
# we can answer any probability question
# with multivariate samples from the joint,
# where conditioned variables match

def main():
    obs = getObservation()
    print 'Observation = ', obs

    samples = sampleATon()
    prob = probFluGivenObs(samples, obs)
    print 'Pr(Flu) = ', prob
```
# Method: Sample A Ton
# -----------------------------------
# chose N_SAMPLES with likelihood proportional to the joint distribution

def sampleATon():
samples = []
    for i in range(N_SAMPLES):
        sample = makeSample()
        samples.append(sample)
return samples
Recall: Probabilistic Model

Fever

Flu

Undergrad

P(U = 1) = 0.8

P(Fl = 1) = 0.1

P(Fev = 1|Flu = 1) = 0.9

P(Fev = 1|Flu = 0) = 0.05

P(T = 1|Flu = 0, U = 0) = 0.1

P(T = 1|Flu = 0, U = 1) = 0.8

P(T = 1|Flu = 1, U = 0) = 0.9

P(T = 1|Flu = 1, U = 1) = 1.0
# Method: Make Sample
# chose a single sample from the joint distribution based on the medical "Probabilistic Graphical Model"

def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flu
    if flu == 1: fev = bern(0.9)
    else: fev = bern(0.05)

    # choose tired based on (undergraduate and flu)
    if und == 1 and flu == 1: tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else: tir = bern(0.1)

    # a sample from the joint has an assignment to *all* random variables
    return [flu, und, fev, tir]
# Method: Make Sample
#
# chose a single sample from the joint distribut.
# based on the medical "Probabilistic Graphical
#

def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)
    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else: fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1: tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else: tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
# Method: Make Sample
#
# chose a single sample from the joint distibut.
# based on the medical "Probabilistic Graphical! 

def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else: fev = bern(0.05)

    # choose tired based on (undergrade and flu)
    if und == 1 and flu == 1: tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else: tir = bern(0.1)

    # a sample from the joint has an
    # assignment to *all* random variables
    return [flu, und, fev, tir]
# Method: Make Sample

# chose a single sample from the joint distrubution
# based on the medical "Probabilistic Graphical"

def makeSample():
    # prior on causal factors
    flu = bern(0.1)
    und = bern(0.8)

    # choose fever based on flue
    if flu == 1: fev = bern(0.9)
    else: fev = bern(0.05)

    # choose tired based on (undergrade and flue)
    if und == 1 and flu == 1:  tir = bern(1.0)
    elif und == 1 and flu == 0: tir = bern(0.8)
    elif und == 0 and flu == 1: tir = bern(0.9)
    else: tir = bern(0.1)

    # a sample from the joint has an assignment to *all* random variables
    return [flu, und, fev, tir]
Alg #1: Joint Sampling

```
N_SAMPLES = 100000

# Program: Joint Sampling
# _______________
# we can answer any prob
# with multivariate sampl
# where conditioned var

def main():
    obs = getObservation
    print 'Observation

    samples = sampleAtObs
    prob = probFluGiven
    print 'Pr(Flu) = ',
```

```
[0, 1, 0, 1]
[1, 1, 1, 1]
[0, 1, 0, 1]
[0, 1, 0, 0]
[0, 1, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
```
Alg #1: Joint Sampling

N_SAMPLES = 100000

# Program: Joint Sample
# _______________________
# we can answer any probability question
# with multivariate samples from the joint,
# where conditioned variables match

def main():
    obs = getObservation()
    print 'Observation = ', obs

    samples = sampleATon()
    prob = probFluGivenObs(samples, obs)
    print 'Pr(Flu) = ', prob
# Method: Probability of Flu Given Observation

# Calculate the probability of flu given many samples from the joint distribution and a set of observations to condition on.

def probFluGivenObs(samples, obs):
    # reject all samples which don't align with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
# Method: Probability of Flu Given Observation
#
# Calculate the probability of flu given many samples from the joint distribution and a set of observations to condition on.
def probFluGivenObs(samples, obs):
    # reject all samples which don't align with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
# Method: Probability of Flu Given Observation

# Calculate the probability of flu given many
# samples from the joint distribution and a set
# of observations to condition on.

def probFluGivenObs(samples, obs):
    # reject all samples which don't align
    # with condition
    keepSamples = []
    for sample in samples:
        if checkObsMatch(sample, obs):
            keepSamples.append(sample)

    # from remaining, simply count...
    fluCount = 0
    for sample in keepSamples:
        [flu, und, fev, tir] = sample
        if flu == 1:
            fluCount += 1

    # counting can be so sweet...
    return float(fluCount) / len(keepSamples)
Alg #1: Joint Sampling

```python
N_SAMPLES = 100000

def main():
    obs = getObservation()
    print('Observation:
    samples = sampleAtObservation(obs)
    prob = probFluGivenTests(samples)
    print('Pr(Flu) =', prob)
```

```
[0, 1, 0, 1]
[0, 1, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 0, 0, 0]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[1, 1, 0, 1]
```

Pr(Flu) = 0.141503173687
If you can sample enough from the joint distribution, you can answer any probability question.

Each one of these is one joint sample:

```
[0, 1, 1, 0]
[1, 0, 1, 1]
[0, 1, 0, 1]
[0, 1, 0, 0]
[0, 1, 0, 0]
[0, 1, 1, 0]
[1, 1, 1, 1]
[0, 1, 0, 0]
[0, 0, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 1, 0, 1]
[0, 0, 1, 0]
[0, 1, 0, 1]
[0, 0, 0, 0]
[0, 0, 0, 0]
```

Observation = [None, None, None, None, None]

\[
\Pr(\text{Flu} \mid \text{Obs}) = 0.10164
\]
Alg #1: Joint Sampling

With enough samples:
• Probability estimates will be correct
• Conditional probability estimates will be correct
• Expectation estimations will be correct
What’s the matter with joint sampling?
Probabilistic Model

$P(Fl = 1) = 0.1$

$P(Fev = 1|Flu = 1) = 0.9$
$P(Fev = 1|Flu = 0) = 0.05$

$P(U = 1) = 0.8$

$P(T = 1|Flu = 0, U = 0) = 0.1$
$P(T = 1|Flu = 0, U = 1) = 0.8$
$P(T = 1|Flu = 1, U = 0) = 0.9$
$P(T = 1|Flu = 1, U = 1) = 1.0$
Probabilistic Model

\[ P(Fl = 1) = 0.1 \]

\[ P(U = 1) = 0.8 \]

\[ P(T = 1|Flu = 0, U = 0) = 0.1 \]
\[ P(T = 1|Flu = 0, U = 1) = 0.8 \]
\[ P(T = 1|Flu = 1, U = 0) = 0.9 \]
\[ P(T = 1|Flu = 1, U = 1) = 1.0 \]

Fever | Flu
---|---
\[ F_{e v}|Flu = 0 \sim N(100.0, 1.81) \]
\[ F_{e v}|Flu = 1 \sim N(98.25, 0.73) \]
The Magic School Bus™
Markov Chain

MCMC

Monte Carlo
MCMC is a way to sample with conditioned variables fixed.

Each one of these is one joint sample:

[Flu, Undergrad, Fever, Tired]
Alg #2: MCMC

All Samples = []

Initial Sample = [0, 0, 101.0, 1]
Alg #2: MCMC

All Samples = []

$S^{(0)} = [0, 0, 101.0, 1]$
Alg #2: MCMC

All Samples = \([S^{(0)}]\)

\[ S^{(0)} = [0, 0, 101.0, 1] \]

From \(S_t\) make \(S_{t+1}\)
**Alg #2: MCMC**

All Samples = \([S^{(0)}]\)

\[
S^{(1)} = [0, 0, 101.0, 1]
\]

\[
P(Flu = 1|\text{All others})
= P(Flu = 1|\text{Und} = 0, \text{Fev} = 98.3, \text{Tir} = 1)
= 0.21
\]

\[
Flu_1 = \text{Sample}\left[P(Flu = 1|\text{All others})\right]
\]
**Alg #2: MCMC**

All Samples = \([S^{(0)}]\]

\[S^{(1)} = [1, 0, 101.0, 1]\]

\[P(Flu = 1|All\ others) = P(Flu = 1|Und = 0, Fev = 98.3, Tir = 1) = 0.21\]

\[Flu_1 = \text{Sample}\left[ P(Flu = 1|All\ others) \right]\]
Alg #2: MCMC

All Samples = $[S^{(0)}]$

$S^{(1)} = [1, 0, 101.0, 1]$

$$P(Und = 1|\text{All others})$$

$$= P(Und = 1|Flu = 1, Fev = 98.3, Tir = 1)$$

$$= 0.91$$

$Und_1 = \text{Sample}\left[P(Und = 1|\text{All others})\right]$
Alg #2: MCMC

All Samples = \([S^{(0)}]\)

\[
S^{(1)} = [1, 1, 101.0, 1]
\]

\[
P(Und = 1|\text{All others}) = P(Und = 1|\text{Flu} = 1, Fev = 98.3, Tir = 1) = 0.91
\]

\[
Und_1 = \text{Sample}\left[ P(Und = 1|\text{All others}) \right]
\]
Let's say you are conditioning on fever being 101.0... then don't change that value.
Alg #2: MCMC

All Samples = \([S^{(0)}]\)

\[S^{(1)} = [1, 1, 101.0, 1]\]
Alg #2: MCMC

All Samples = $[S^{(0)}]$ 

$S^{(1)} = [1, 1, 101.0, 1]$
All Samples = $[S^{(0)}]$ 

$S^{(1)} = [1, 1, 101.0, 1]$
Alg #2: MCMC

All Samples = \([S^{(0)}, S^{(1)}]\)

\(S^{(1)} = [1, 1, 101.0, 1]\)
Alg #2: MCMC

All Samples = \([S^{(0)}, S^{(1)}]\)

\[
S^{(2)} = [0, 1, 101.0, 1]
\]
Alg #2: MCMC

All Samples = \( [S^{(0)}, S^{(1)}, S^{(2)}] \)

Repeat at least 10,000 times
Alg #2: MCMC

All Samples = [S^{(0)}, S^{(1)}, S^{(2)}, ..., S^{(10000)}]

Repeat at least 10,000 times
MCMC is a way to sample with conditioned variables fixed.

Each one of these is one joint sample:

[Flu, Undergrad, Fever, Tired]
For each random variable you must specify a way to sample from

$$P(X_j^{(t+1)}|X_1^{(t+1)}, \ldots, X_{j-1}^{(t+1)}, X_{j+1}^{(t)}, \ldots X_n^{(t)})$$

Current random variable

All the other values in the sample
Probabilistic Model

\[ P(Fl = 1) = 0.1 \]

\[ P(U = 1) = 0.8 \]

\[ Fev | Flu = 0 \sim N(100.0, 1.81) \]
\[ Fev | Flu = 1 \sim N(98.25, 0.73) \]

\[ P(T = 1 | Flu = 0, U = 0) = 0.1 \]
\[ P(T = 1 | Flu = 0, U = 1) = 0.8 \]
\[ P(T = 1 | Flu = 1, U = 0) = 0.9 \]
\[ P(T = 1 | Flu = 1, U = 1) = 1.0 \]
Alg #1: Joint Sampling

With enough samples:
• Probability estimates will be correct
• Conditional probability estimates will be correct
• Expectation estimations will be correct