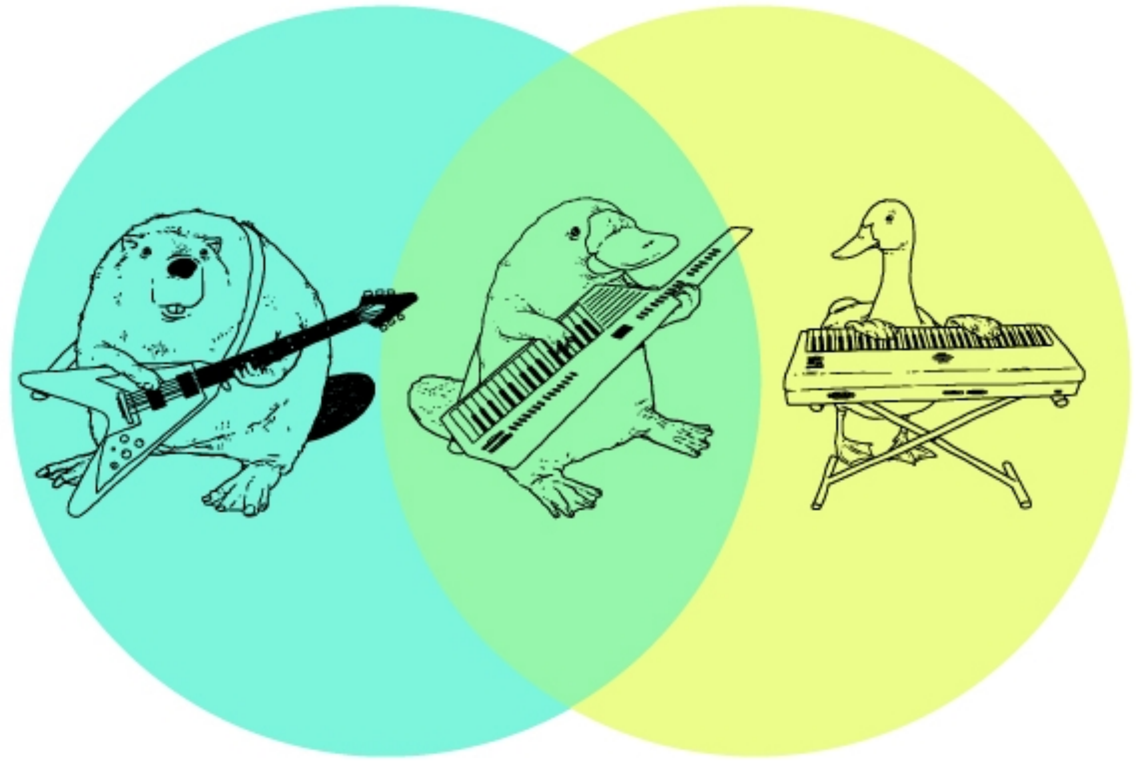
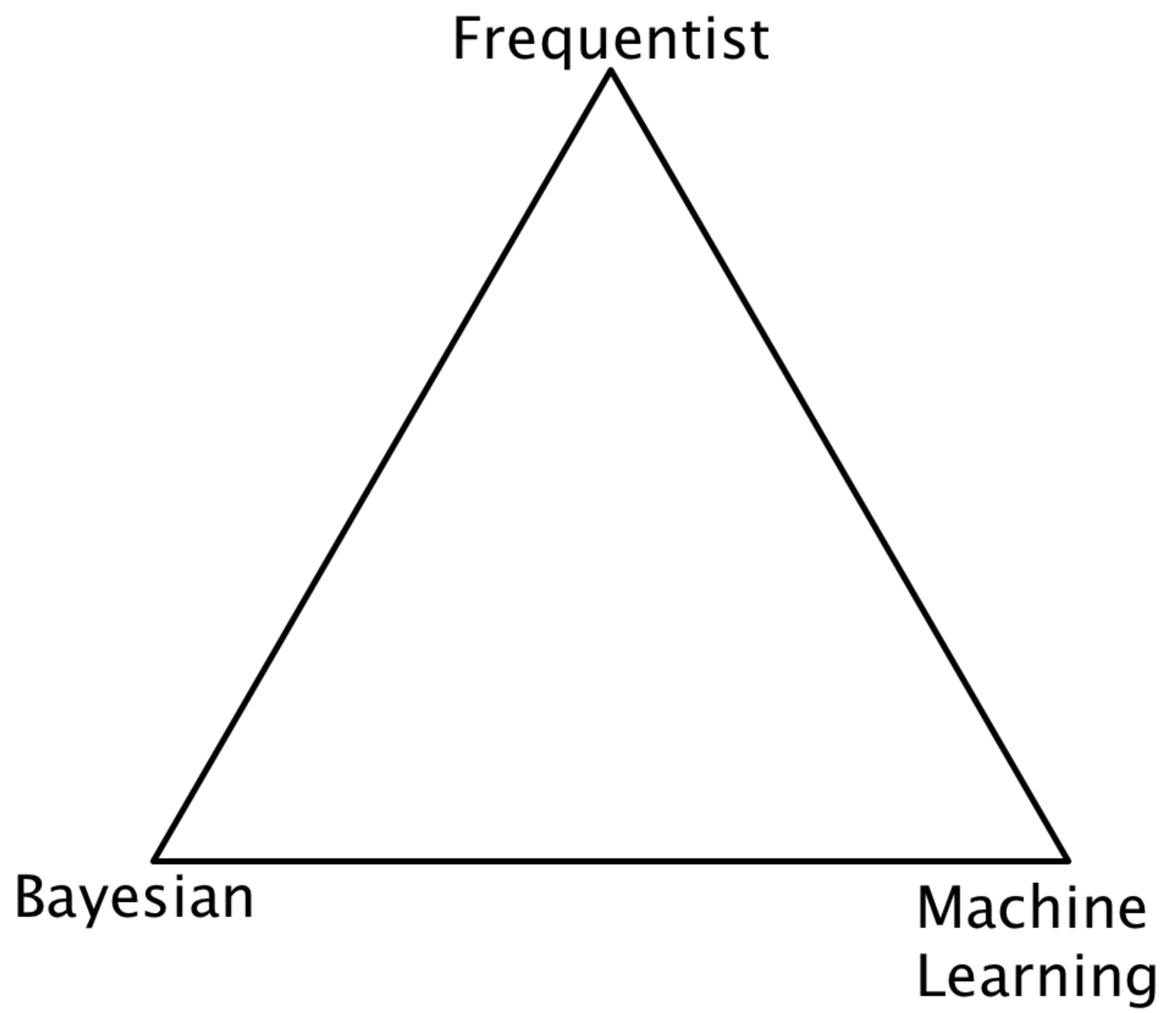
The Duct Tape of Heroes: Bayes Rule



Vincent D. Warmerdam

GoDataDriven + koaning.io + @fishnets88

Different schools of data



Today

Textbook Example of Bayes Rule

Introduce Probabilistic Graphical Models in Python

Show when Bayesians statistics shine

Explain Biased Random Search for OR

Probabilistic Approaches for Heroes of the Storm

Discuss that whole frequentist/bayesian thing

Textbook Bayes Rule

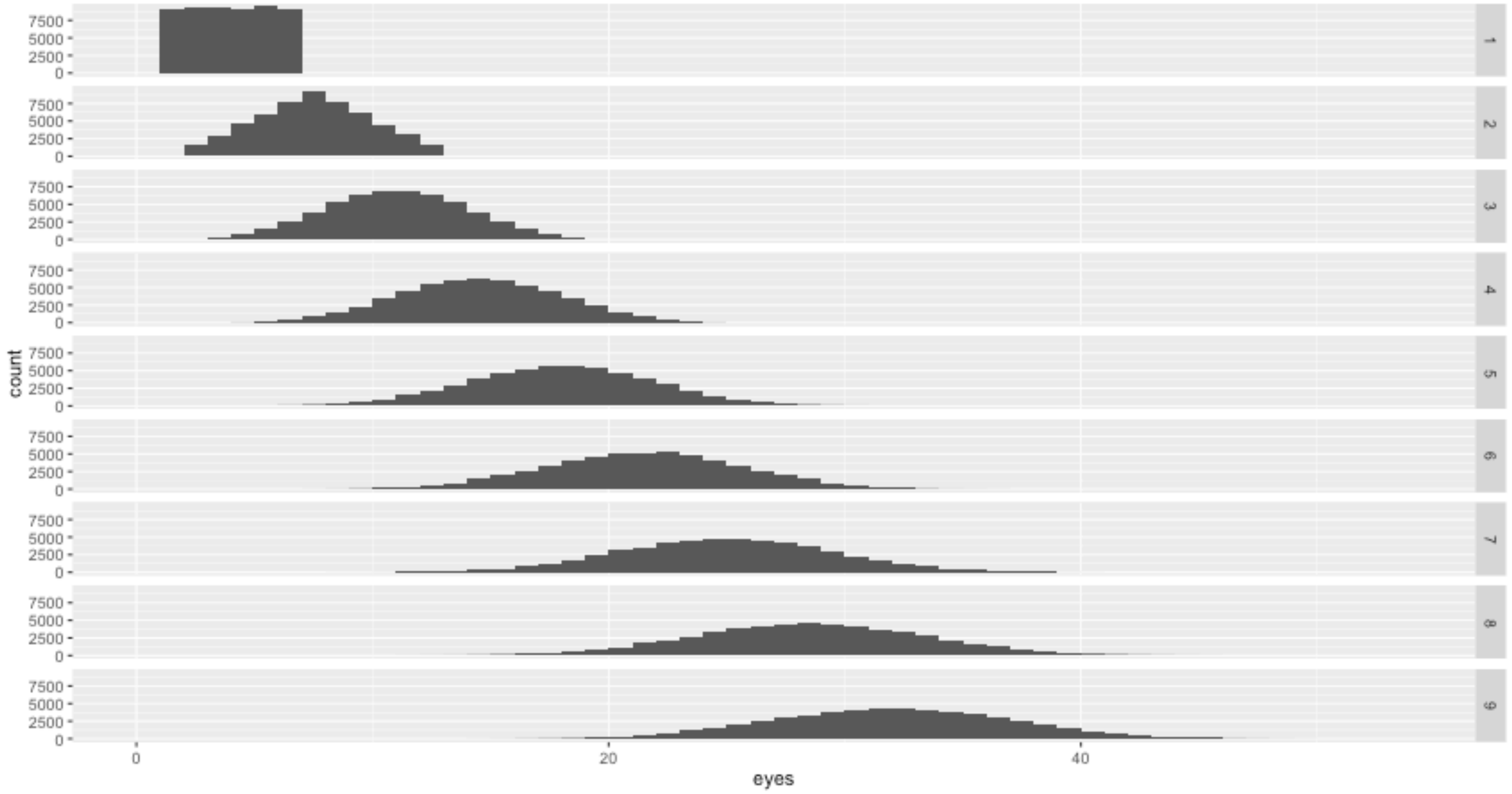
The theory of probabilities is basically just common sense reduced to calculus.

— Pierre-Simon Laplace

Example of common sense

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

P(E|D), given number of dice, probability of sum of eyes



A textbook example

There is an epidemic. A person has a probability $\frac{1}{100}$ to have the disease. The authorities decide to test the population, but the test is not completely reliable: the test generally gives $\frac{1}{110}$ people a positive result but given that you have the disease the probability of getting a positive result is $\frac{65}{100}$.

A textbook answer

Let D denote the event of having the disease, let T denote event of a positive outcome of a test. If we are interested in finding $P(D|T)$ then we can just go and apply Bayes rule:

$$P(D|T) = \frac{P(T|D)P(D)}{P(T)} = \frac{\frac{65}{100} \times \frac{1}{100}}{\frac{1}{110}} \approx \mathbf{0.715}$$

A textbook answer

Let D denote the event of having the disease, let T denote event of a positive outcome of a test. If we are interested in finding $P(D|T)$ then we can just go and apply Bayes rule:

$$P(D|T) = \frac{P(T|D)P(D)}{P(T)} = \frac{\frac{65}{100} \times \frac{1}{100}}{\frac{1}{110}} \approx \mathbf{0.715}$$

What else can we infer?

Inference : we know $P(T|\neg D)$!

$$\begin{aligned} P(T) &= P(T \cap D) + P(T \cap \neg D) \\ &= P(T|D)P(D) + P(T|\neg D)P(\neg D) \end{aligned}$$

Inference : we know $P(T|\neg D)$!

$$\begin{aligned} P(T) &= P(T \cap D) + P(T \cap \neg D) \\ &= P(T|D)P(D) + P(T|\neg D)P(\neg D) \end{aligned}$$

After calculation:

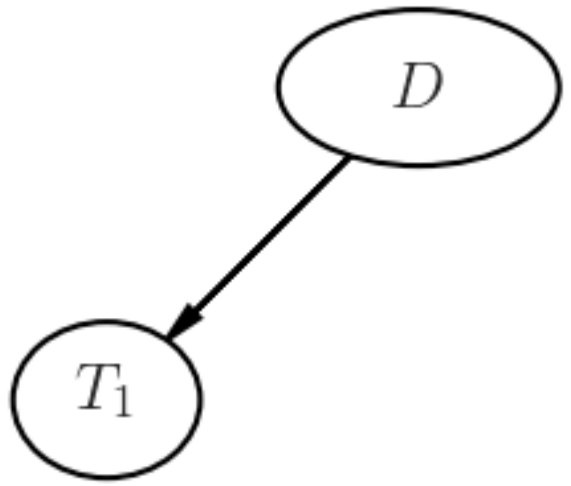
$$P(T|\neg D) = \frac{7}{1980}$$

With this knowledge, we can infer even more!

$$\begin{aligned}
P(TT) &= P(TT|D)P(D) + P(TT|\neg D)P(\neg D) \\
&= \left(\frac{65}{100}\right)^2 \times \frac{1}{100} + \left(\frac{7}{1980}\right)^2 \times \frac{99}{100} \\
&= \frac{839}{198000} \approx 0.004237374
\end{aligned}$$

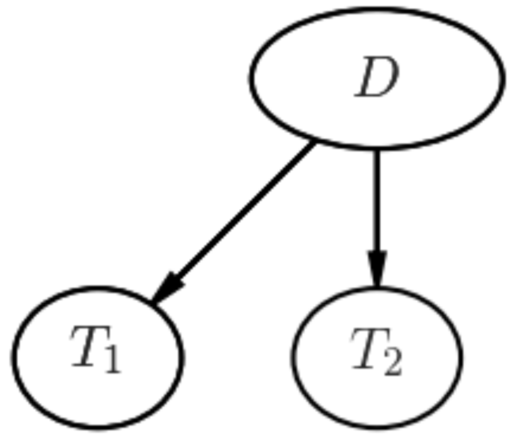
$$\begin{aligned}
P(TTT) &= P(TTT|D)P(D) + P(TTT|\neg D)P(\neg D) \\
&= \left(\frac{65}{100}\right)^3 \times \frac{1}{100} + \left(\frac{7}{1980}\right)^3 \times \frac{99}{100} \\
&= \frac{1076657}{392040000} \approx 0.002746294
\end{aligned}$$

One Test



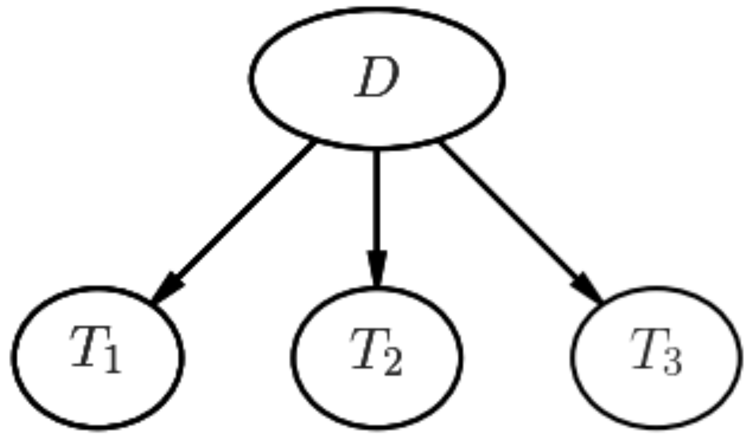
$$P(D|T) = \frac{P(T|D)P(D)}{P(T)} = \frac{\frac{65}{100} \times \frac{1}{100}}{\frac{1}{110}} \approx \mathbf{0.715}$$

Two Tests



$$P(D|TT) = \frac{P(TT|D)P(D)}{P(TT)} = \frac{\left(\frac{\mathbf{65}}{\mathbf{100}}\right)^{\mathbf{2}} \times \frac{\mathbf{1}}{\mathbf{100}}}{\frac{\mathbf{839}}{\mathbf{198000}}} = \frac{\mathbf{16731}}{\mathbf{16780}} \approx 0.9971$$

Three Tests



$$P(D|TTT) = \frac{P(TTT|D)P(D)}{P(TTT)} = \frac{\left(\frac{65}{100}\right)^3 \times \frac{1}{100}}{\frac{1076657}{392040000}} = \frac{21532797}{21533140} \approx 0.9999$$

What just happened?

Via Bayes Rule, we can infer information about $P(D|TTT)$ even though we initially only know about $P(T|D)$.

The trick is that we clearly define the uncertainty that we have and infer the rest via probability theory. In this case the main cheat was realizing that the tests are independant of eachother.

Notice something interesting: the same model can have different data at prediction time and still manage adequate output.

Probabilistic Graphical Models

Probabilistic Graphs!

If we have domain knowledge, we can create a graph like before that defines dependencies on data. Support for Probabilistic modelling is growing, also in python!

Let's create a Probabilistic graph in python with `pomegrenate`. It's a modest library that has likeable documentation.

Probabilistic Graphs!

Suppose you want to know if today's weather is predicted correctly. There are three broadcasters.

- Alice is very accurate; 80% correct
- Bob is terrible: 55%
- Claire is alright: 65%

Probabilistic Graphs!

Suppose you want to know if today's weather is predicted correctly. Each broadcaster tends to present on different days of the week.

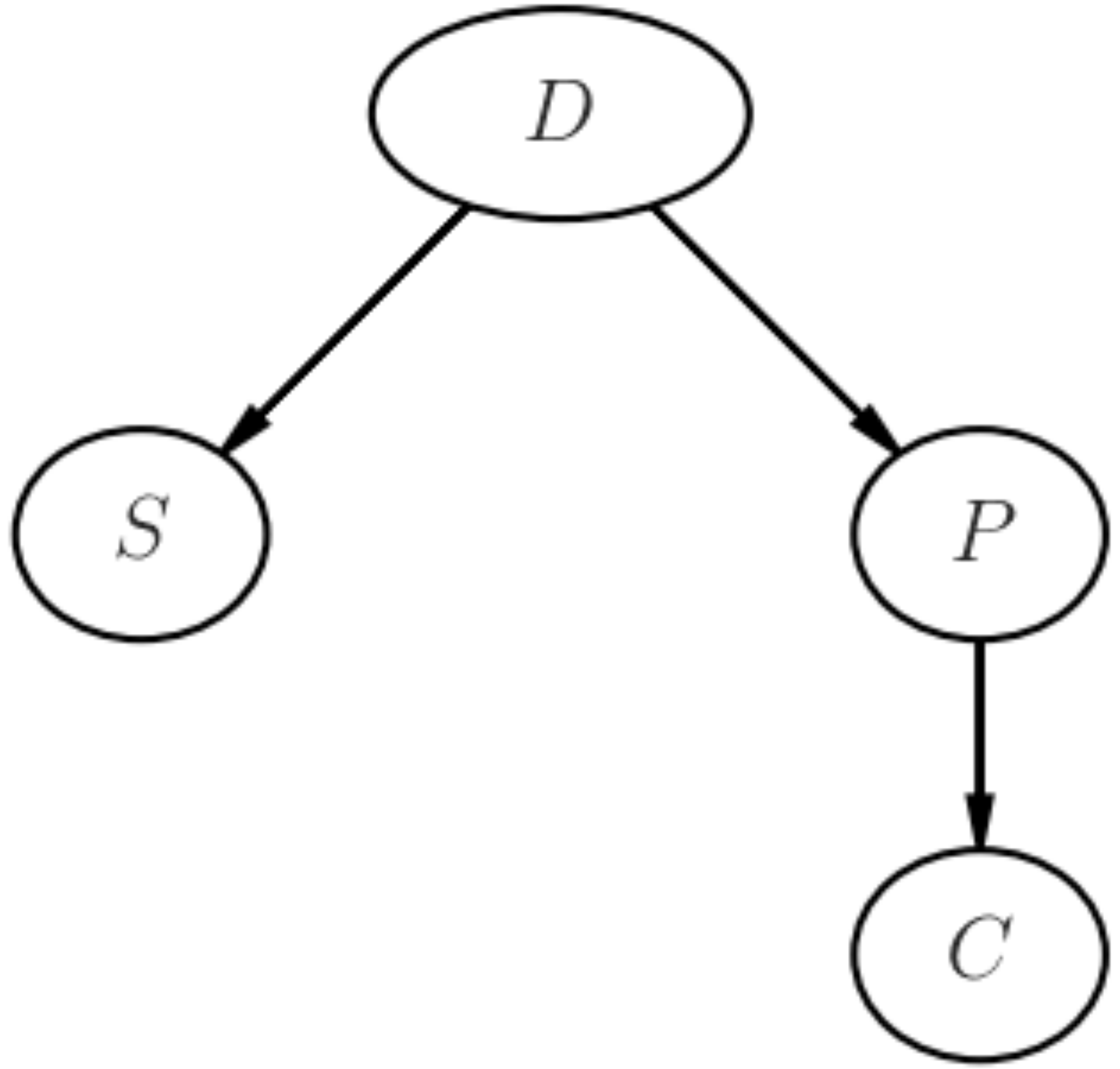
- Alice is more weekday; 70% weekday, 10% weekend
- Bob is more weekend: 10% weekday, 65% weekend
- Claire is both: 20% weekday, 25% weekend

Probabilistic Graphs!

Suppose you want to know if today's weather is predicted correctly. Suppose that during the weekend it is more likely for the sun to shine.

- given weekend: 80% sunny
- given weekday: 60% sunny

Probabilistic Graphs!



Step 1: Define Probabilities

```
day_of_week = pq.DiscreteDistribution(
    {'weekday': 5./7, 'weekend': 2./7}
)

who = pq.ConditionalProbabilityTable([
    ['weekday', 'Alice', 0.9],
    ['weekend', 'Alice', 0.1],
    ['weekday', 'Bob', 0.35],
    ['weekend', 'Bob', 0.65],
    ['weekday', 'Claire', 0.5],
    ['weekend', 'Claire', 0.5]
], [day_of_week])

...
```

Step 2: Build graph

```
s1 = pq.State(day_of_week, name="day_of_week")
s2 = pq.State(who, name="who")
s3 = pq.State(sunny, name="sunny")
s4 = pq.State(correct, name="correct")

network = pq.BayesianNetwork("Weather: day of week")
network.add_states([s1, s2, s3, s4])
network.add_transition(s1, s2)
network.add_transition(s1, s3)
network.add_transition(s2, s4)
network.bake()
```

Step 3: Infer `{'who' : 'Alice'}`

```
day_of_week
    "weekend" :0.054054054054054054,
    "weekday" :0.9459459459459458
who
    "Claire" :0.0,
    "Bob" :0.0,
    "Alice" :1.0
sunny
    "false" :0.21081081081081082,
    "true" :0.7891891891891892
correct
    "false" :0.10000000000000003,
    "true" :0.9
```

Step 3: `{'who': 'Bob'}`

```
day_of_week
    "weekend" :0.7222222222222222,
    "weekday" :0.2777777777777779
who
    "Claire" :0.0,
    "Bob" :1.0,
    "Alice" :0.0
sunny
    "false" :0.3444444444444444,
    "true" :0.6555555555555556
correct
    "false" :0.45,
    "true" :0.55
```

Step 3: `{'who': 'Bob', 'weekend': T}`

```
day_of_week
    "weekend" :1.0,
    "weekday" :0.0
who
    "Claire" :0.0,
    "Bob" :1.0,
    "Alice" :0.0
sunny
    "false" :0.4,
    "true" :0.6
correct
    "false" :0.45,
    "true" :0.55
```

Step 3: `{'correct': True}`

```
day_of_week
    "weekend" :0.23040604343720494,
    "weekday" :0.7695939565627951
who
    "Claire" :0.1841359773371105,
    "Bob" :0.18696883852691226,
    "Alice" :0.6288951841359772
sunny
    "false" :0.24608120868744096,
    "true" :0.7539187913125589
correct
    "false" :0.0,
    "true" :1.0
```

Probabilistic Graphical Models

These types of models are rather powerful and flexible. One of it's most powerful features is that it can logically handle missing data at prediction time.

The example I've given here just handles discrete models, but mathematically you can define any distribution between nodes.

Often coding these problems is the harder part. `pomegrenate` currently only offers discrete distributions, not continous ones. Sampling is an alternative.

When Bayesian Thinking Shines

Learning from Biased Data

Suppose that I have a call-centre.

Learning from Biased Data

Suppose that I have a call-centre.

Suppose we are interested in knowing how long people wait in line.

Learning from Biased Data

Suppose that I have a call-centre.

Suppose we are interested in knowing how long people wait in line.

Suppose we only measure the people in a waiting line who wait for a very long time (ie; that's when they start to complain).

Learning from Biased Data

Suppose that I have a call-centre.

Suppose we are interested in knowing how long people wait in line.

Suppose we only measure the people in a waiting line who wait for a very long time (ie; that's when they start to complain).

What can I then say about the waiting time in general? I only measure a few people and their waiting times and I know that the waiting times are larger than some time m. I know nothing else!

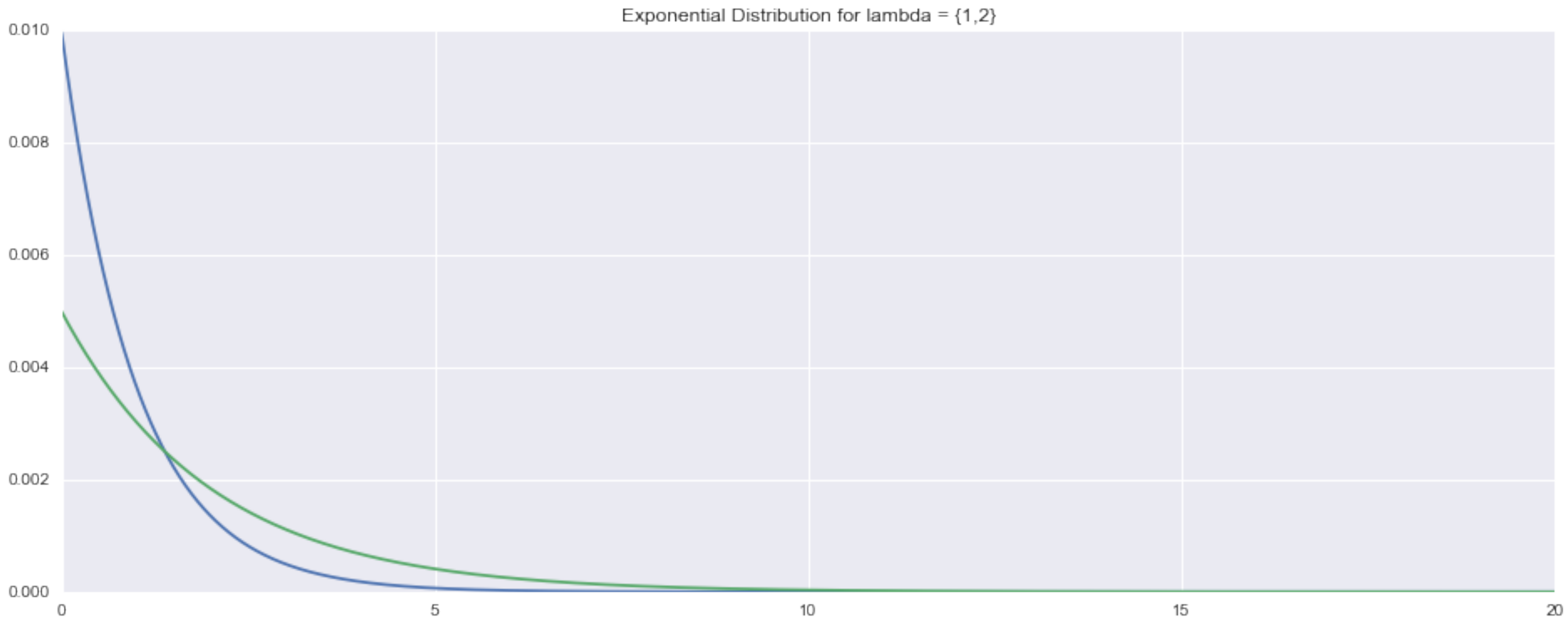
Translation to models

Waiting lines usually suggest exponential distributions.

$$p(x|\lambda) = \frac{1}{\lambda}\, e^{-\lambda x}$$

where x is the waiting time per customer. Higher values of λ correspond to longer waiting times between customers.

Two exponential models



Might feel a bit tricky

How do we find the best value of λ while still keeping in mind that we have a biased dataset?

Might feel a bit tricky

How do we find the best value of λ while still keeping in mind that we have a biased dataset?

BAYES TO THE RESCUE

$$p(\lambda|D_{bias})$$

Might feel a bit tricky

How do we find the best value of λ while still keeping in mind that we have a biased dataset?

BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) = \frac{p(D_{bias}|\lambda)p(\lambda)}{p(D_{bias})}$$

Might feel a bit tricky

How do we find the best value of λ while still keeping in mind that we have a biased dataset?

BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) = \frac{p(D_{bias}|\lambda)p(\lambda)}{p(D_{bias})} \propto p(D_{bias}|\lambda)p(\lambda)$$

Might feel a bit tricky

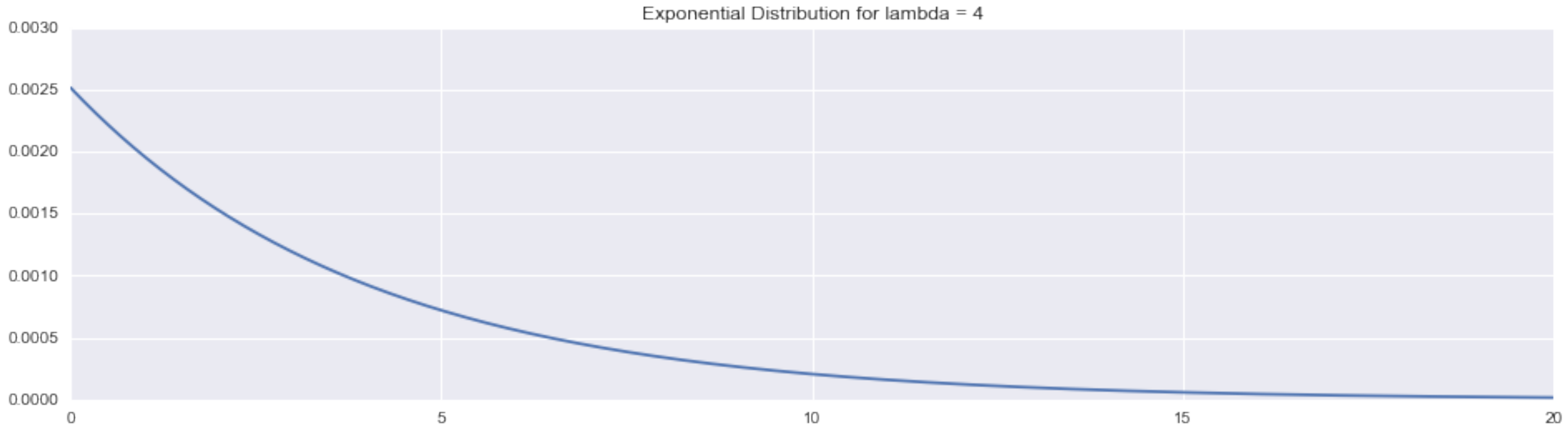
How do we find the best value of λ while still keeping in mind that we have a biased dataset?

BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) = \frac{p(D_{bias}|\lambda)p(\lambda)}{p(D_{bias})} \propto p(D_{bias}|\lambda)p(\lambda) = \Pi_i p(x_i|\lambda)p(\lambda)$$

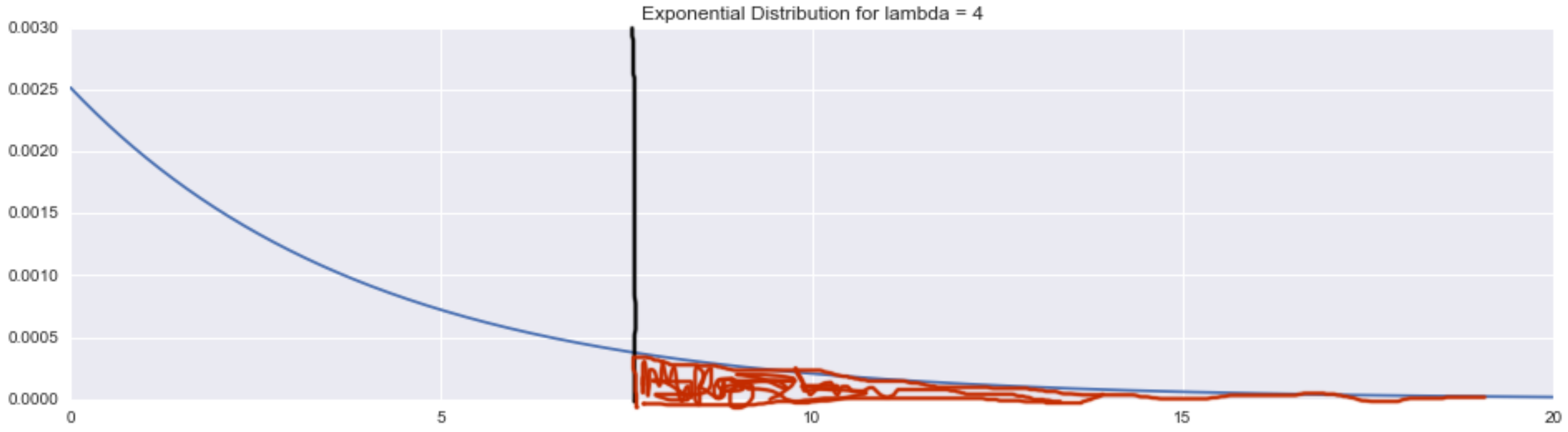
BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) \propto p(\lambda)\Pi_i p(x_i|\lambda)$$



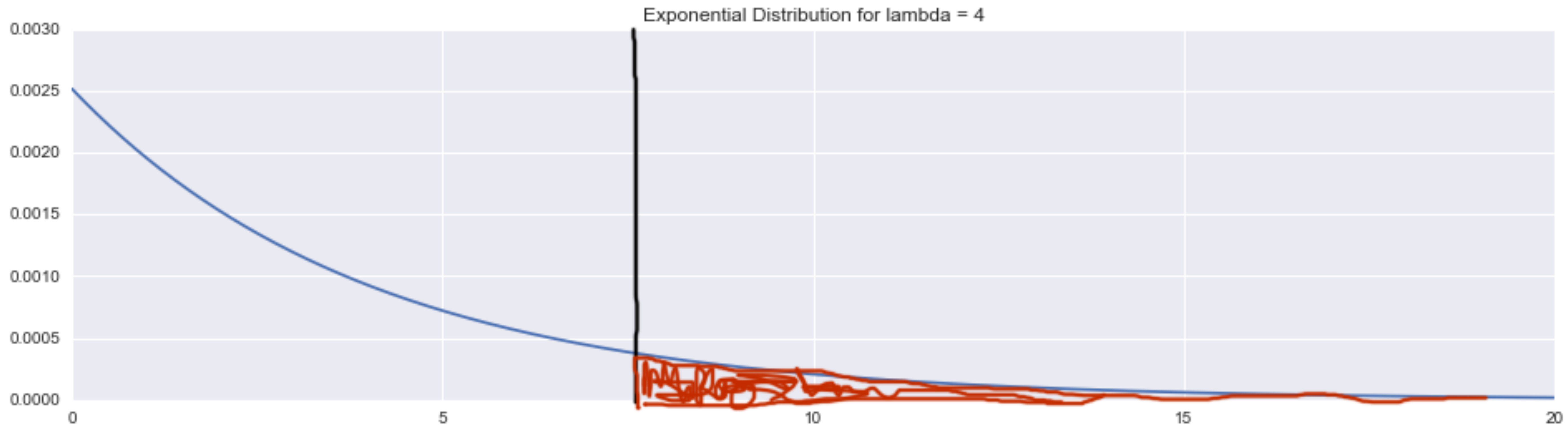
BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) \propto p(\lambda)\Pi_i p(x_i|\lambda)$$



BAYES TO THE RESCUE

$$p(\lambda|D_{bias}) \propto p(\lambda)\Pi_i p(x_i|\lambda)$$



For $m = 7.5$, the red shows the distribution of interest.

BAYES TO THE RESCUE

What the plot shows via maths, what do we know about x_i when $x_i > m$:

$$p(x_i|\lambda) = \Pi_i \frac{\frac{1}{\lambda} e^{-x_i \lambda}}{Z(\lambda)}$$

$$Z(\lambda) = \int_m^\infty \frac{1}{\lambda} e^{-x\lambda} dx$$

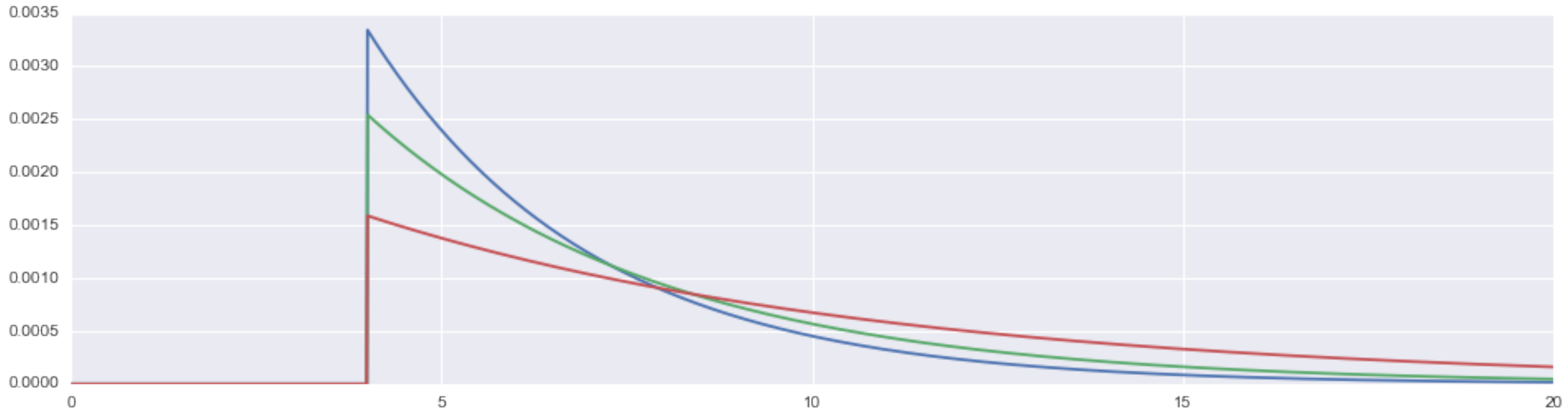
Enough math, time to code!

```
def p_xs_given_lambda(lambd, m):
    xs = np.arange(0, 20, 0.01)
    distr = 1.0/lambd * np.exp(-xs/lambd)
    distr[xs < m] = 0.000001
    return distr/np.sum(distr)

def p_x_given_lambda(x, lambd, m):
    xs = np.arange(0, 20, 0.01)
    distr = p_xs_given_lambda(lambd, m)
    return distr[xs > x][0]
```

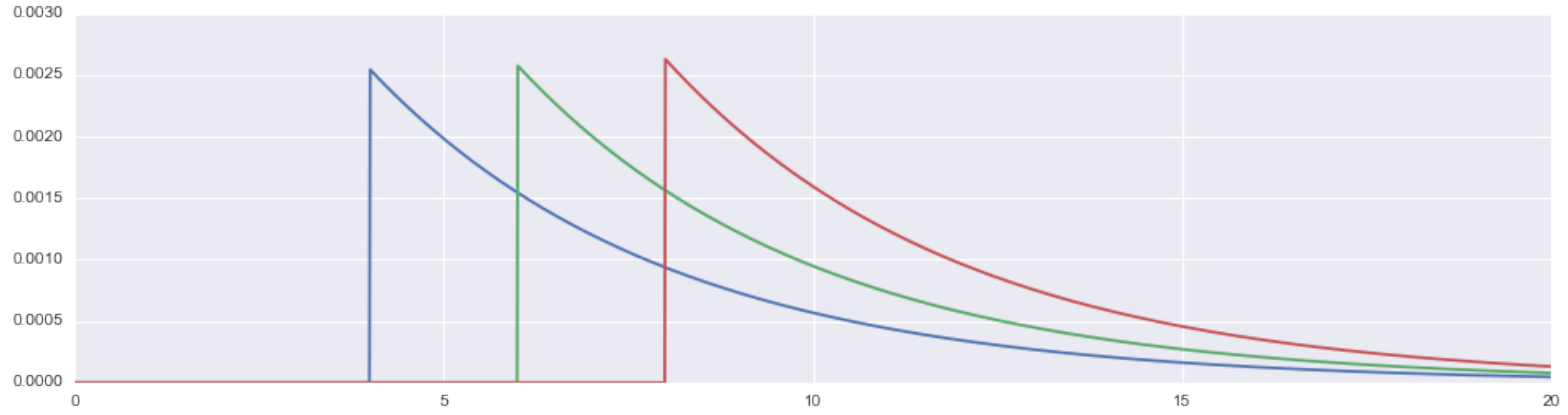
Constant m, differing λ.

```
xs = np.arange(0, 20, 0.01)
plt.plot(xs, p_xs_given_lambda(3, 4))
plt.plot(xs, p_xs_given_lambda(4, 4))
plt.plot(xs, p_xs_given_lambda(7, 4))
```



Differing m, constant λ.

```
xs = np.arange(0, 20, 0.01)
plt.plot(xs, p_xs_given_lambda(4, 4))
plt.plot(xs, p_xs_given_lambda(4, 6))
plt.plot(xs, p_xs_given_lambda(4, 8))
```

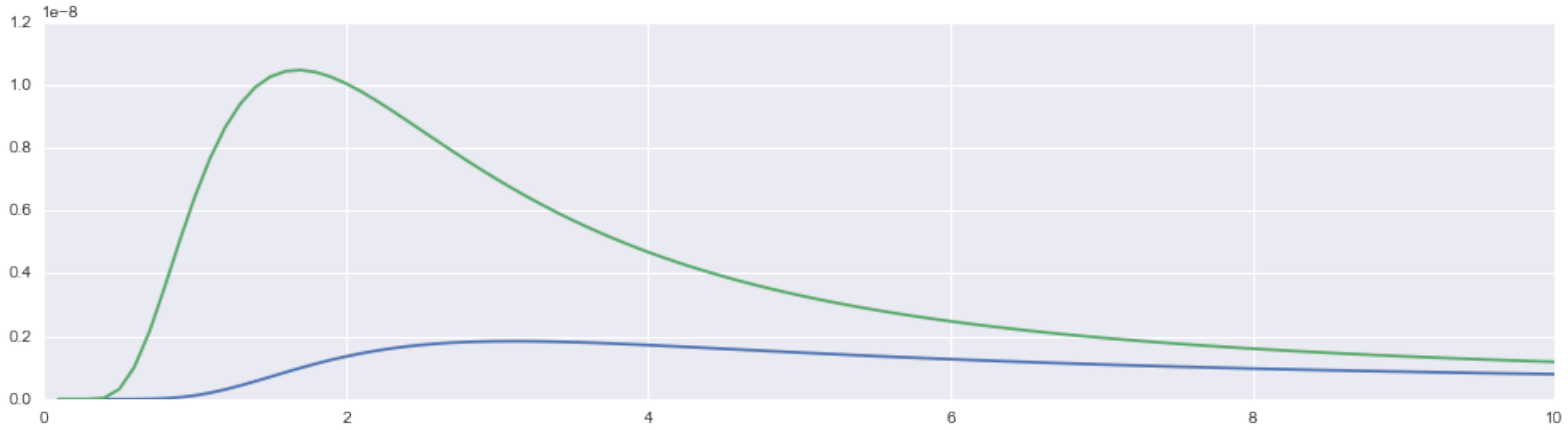


Combining it all

```
def p_d_given_lambda(data, lambd, m):
    res = 1
    for d in data:
        res *= p_x_given_lambda(d, lambd, m)
    return res
```

Different data, λ on x-axis.

```
m = 4
lambdas = np.linspace(0.1, 10, 100)
plt.plot(lambdas, [p_d_given_lambda([7,6,8], l, m) for l in lambdas])
plt.plot(lambdas, [p_d_given_lambda([5,6,6], l, m) for l in lambdas])
```



What have we just done?

We have shown a way to derive a statistic from biased data using bayes rule. Not only does it allow us to give the most likely λ but it even allows us to talk about the variance of this statistic given the data available.

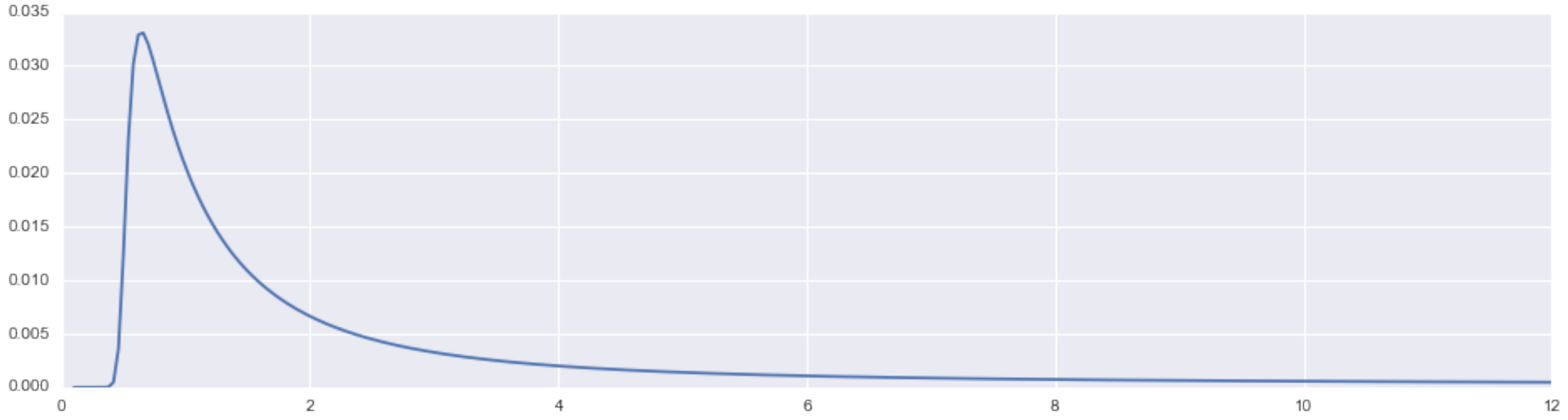
Test

Let's simulate our own data and confirm that this method works.

```
m = 6
true_l = 1
data = np.random.exponential(true_l, N)
data = data[data > m]
```

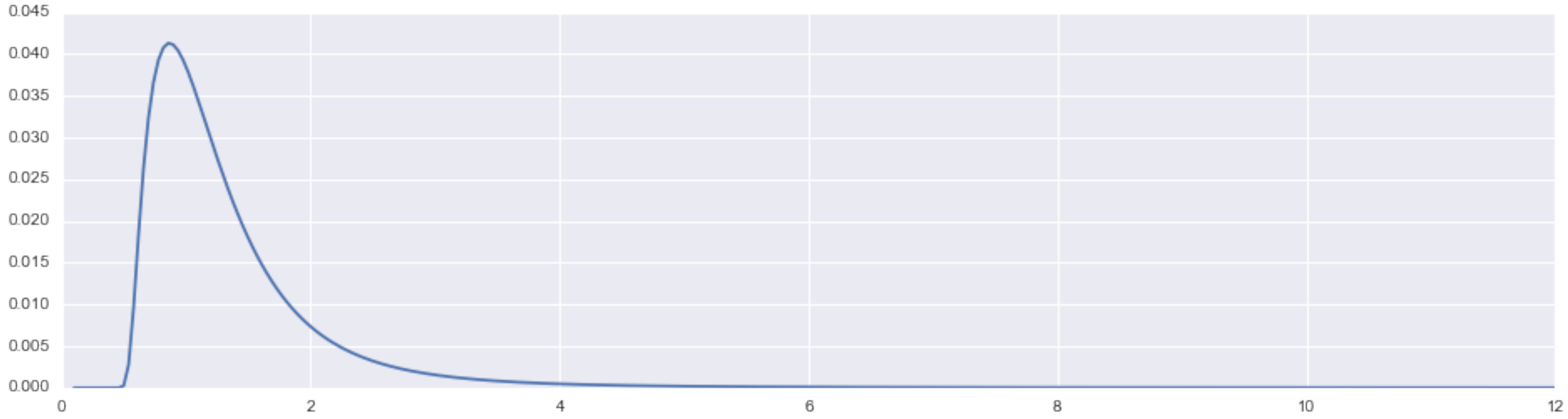
We vary N to to investigate the effect of having many points in the data array.

Different datasize



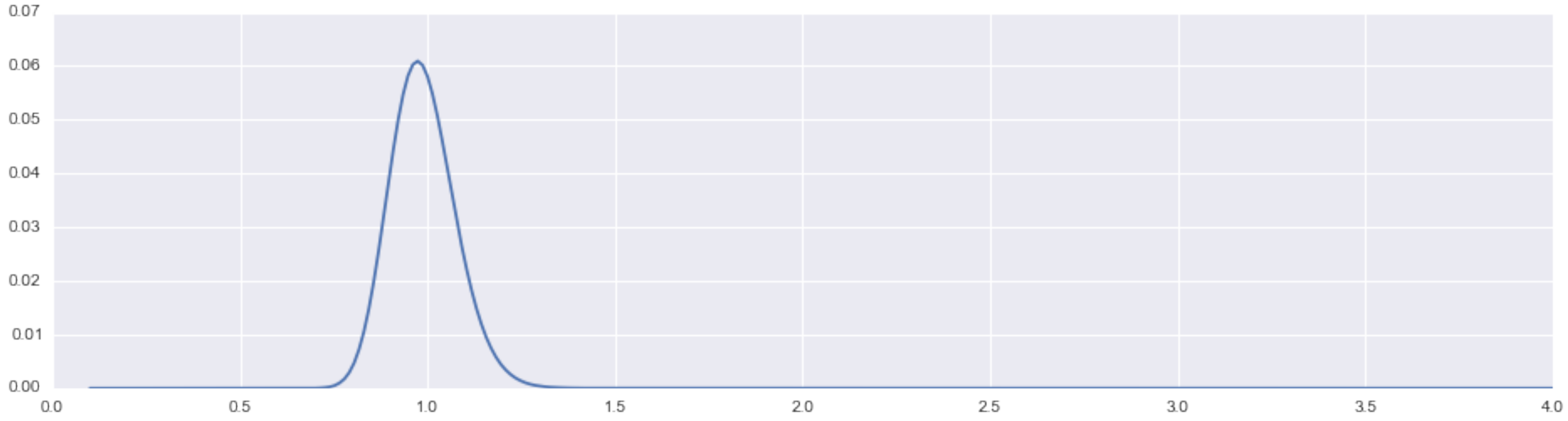
```
array([ 6.09277787,  6.38649219])
```

Different datasize



```
array([ 6.30177249,  8.70951343,  6.04451668,
        6.67389008,  6.76578963, 6.41075569])
```

Different datasize



$$\lambda = 1, m = 2, |D| = 122$$

Inspiration for Optimisation Algorithms

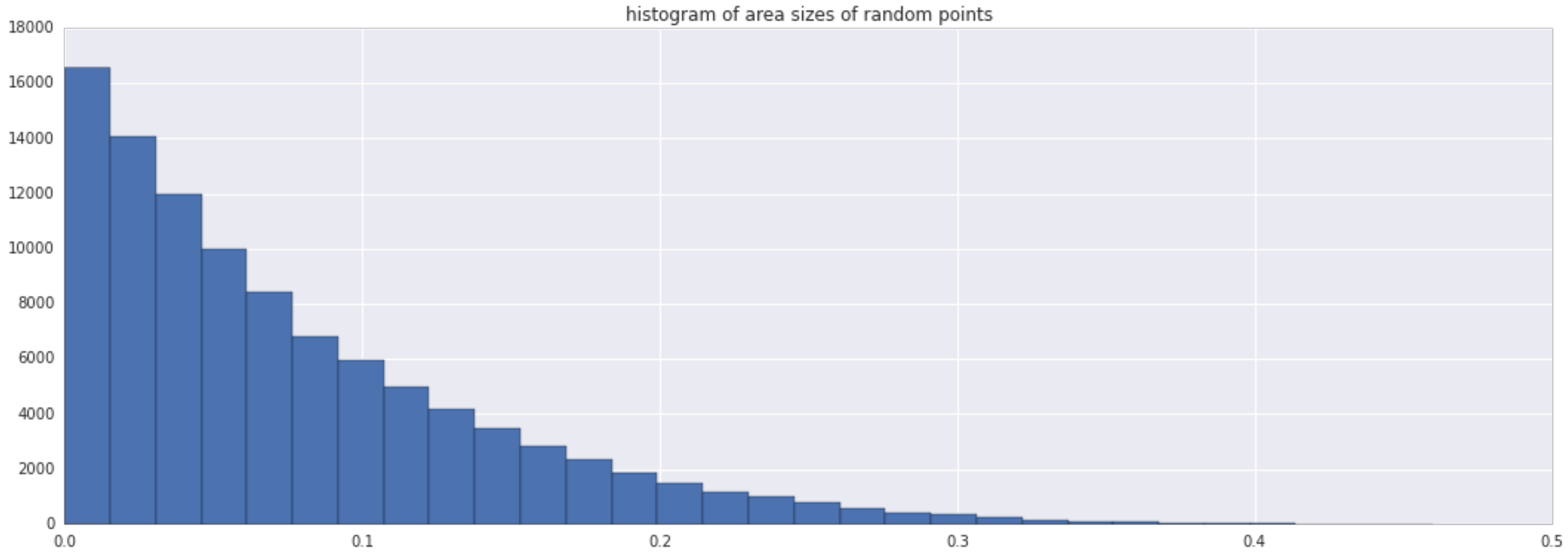
Applications in OR

Sometimes bayes rule can help with a parameter search. Let's take a simple example; finding the largest triangle in a 1x1 square.

This might seem like a silly example because we already know the answer beforehand. This is a good property for now because we want to know how an algorithm performs compared to the best possible solution.

Let's start with random

```
rand_vals = np.random.rand(100000, 6)
_ = plt.hist([shoelace(i) for i in rand_vals], 30)
```



Learning step

Let X be the coordinates of the triangle, let A be the area. The plot we've just shown was the distribution of A. Effectively we've just sampled from this distribution;

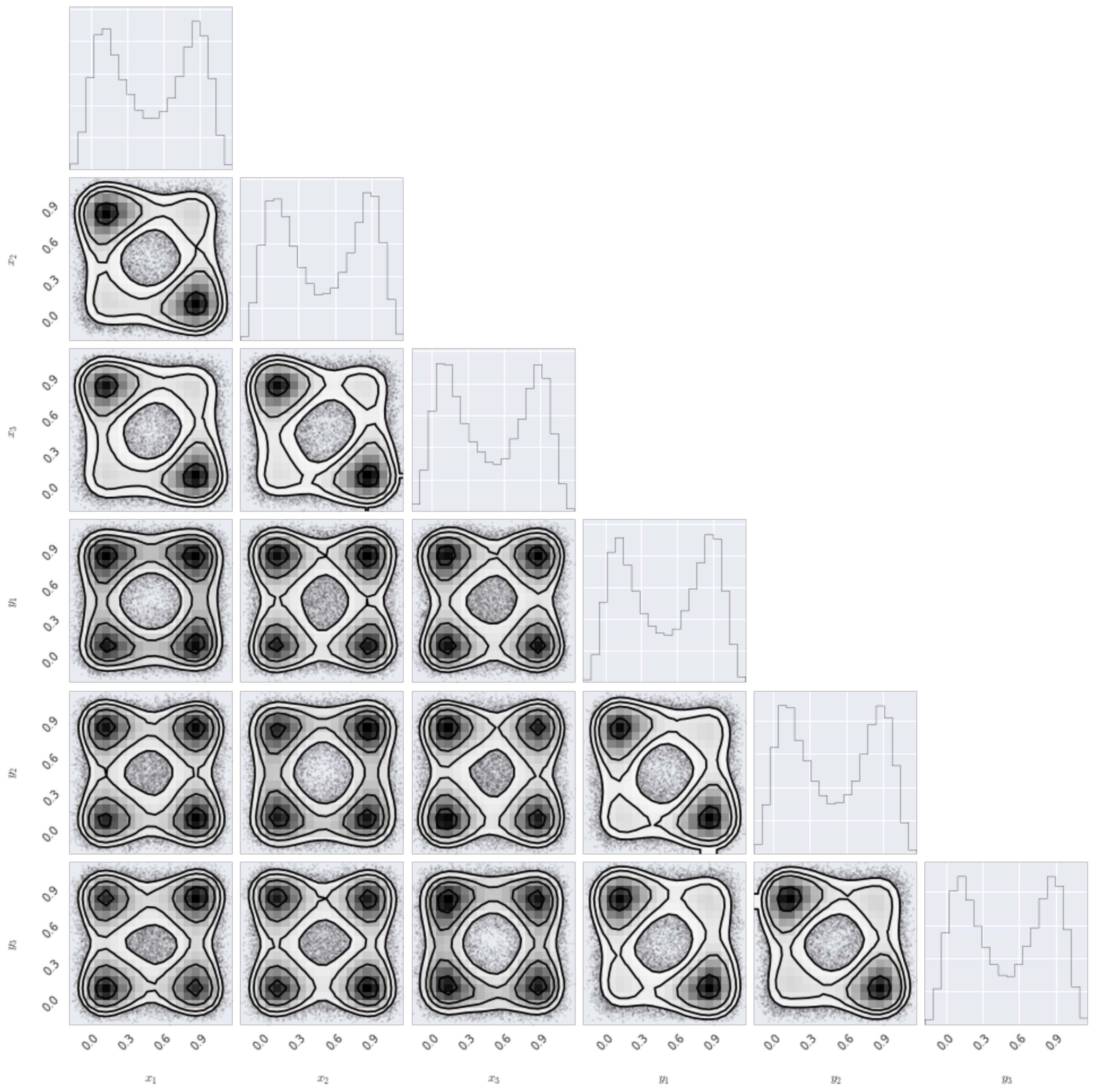
$$p(A) = \int p(A|X)p(X)dx$$

Learning step

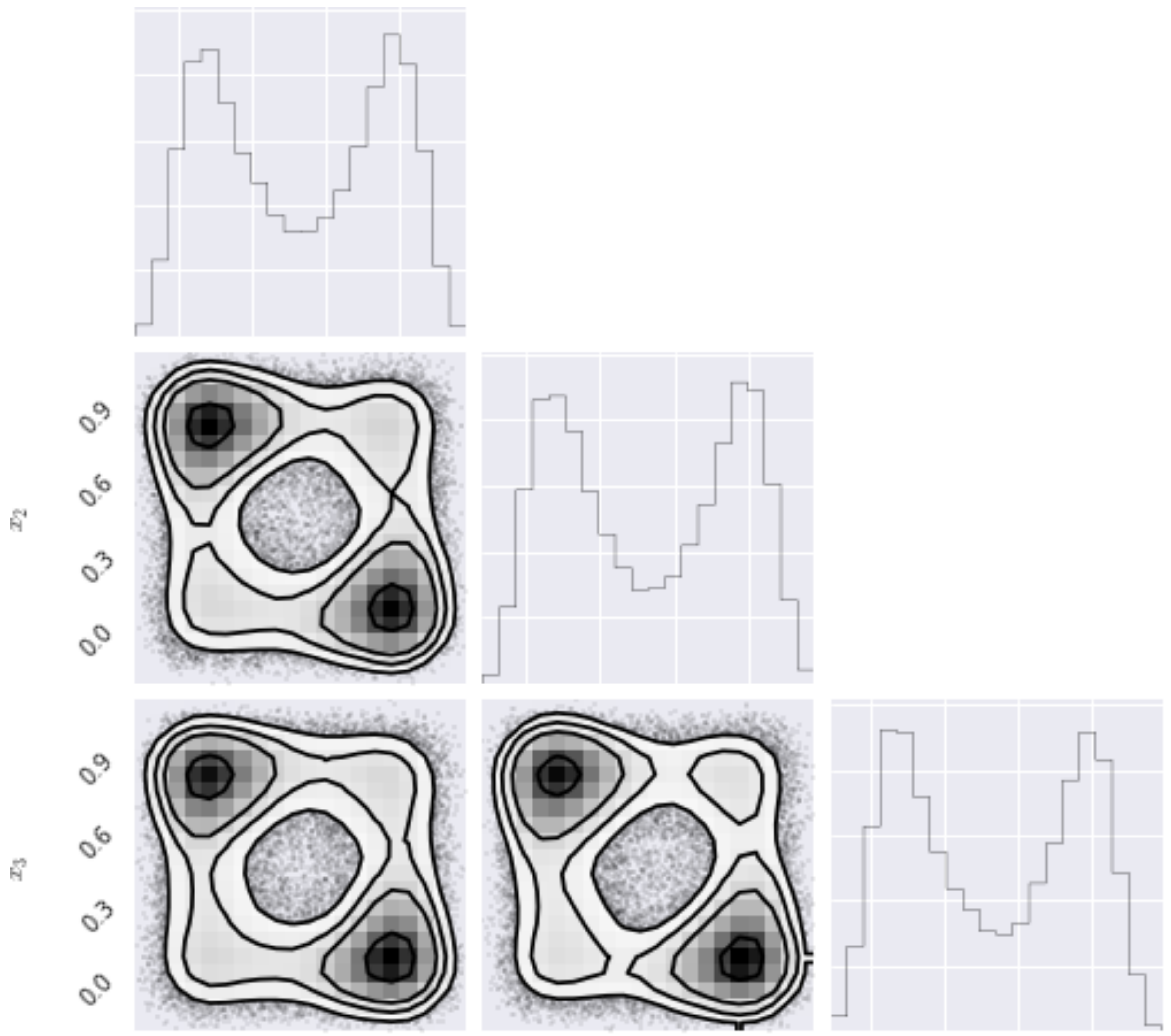
Let's now turn it over itself. Let's try to find this distribution:

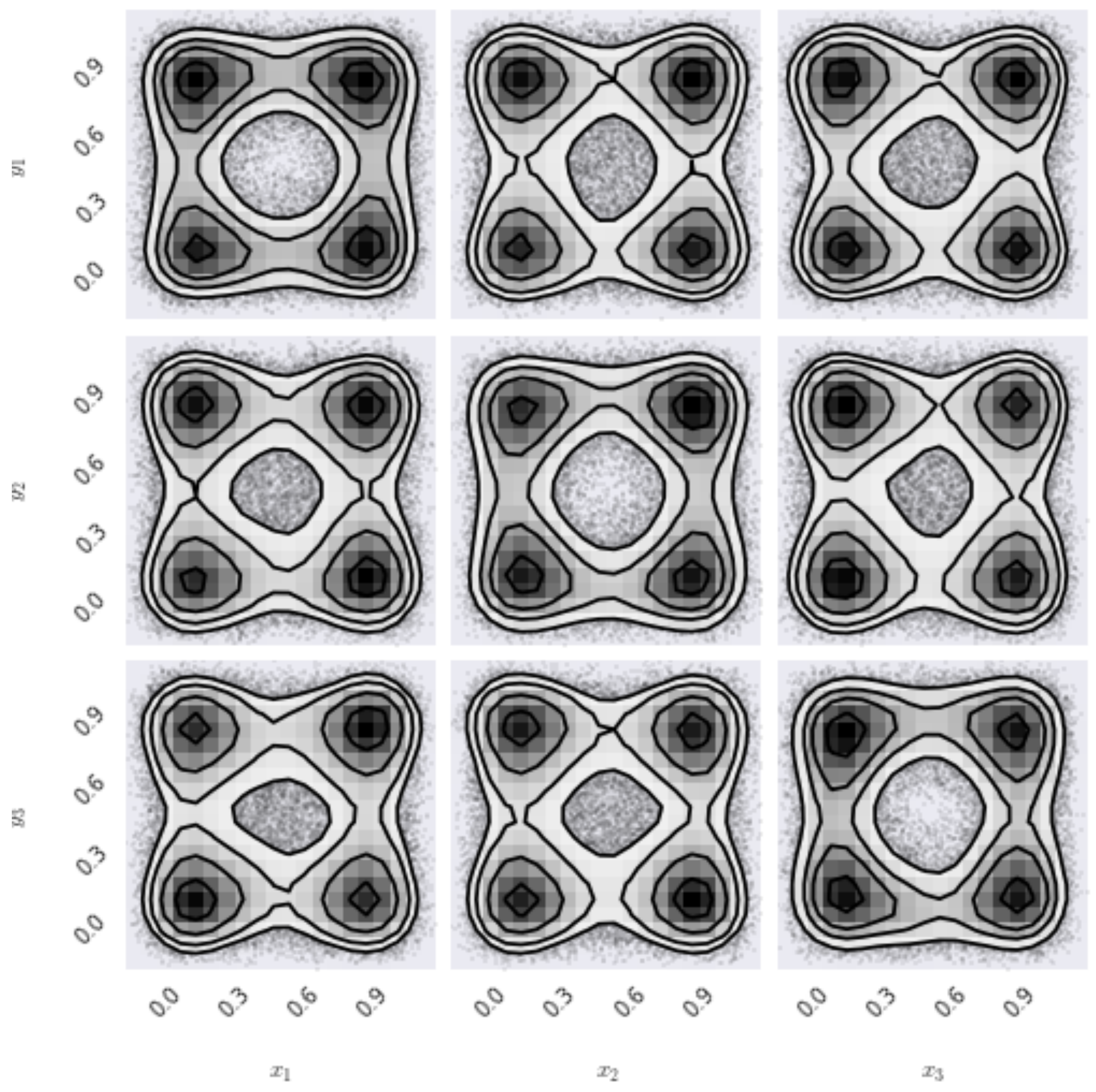
$$p(X|A > m)$$

If we pick m such that we have enough points to learn a distribution from but large enough to be discriminate from uniform then we might be able to update our belief of what good allocations to this problem might look like.



x_1
x_2
x_3
y_1
y_2
y_3
0.0
0.3
0.6
0.9


x_2
x_3
0.0
0.3
0.6
0.9
0.0
0.3
0.6
0.9



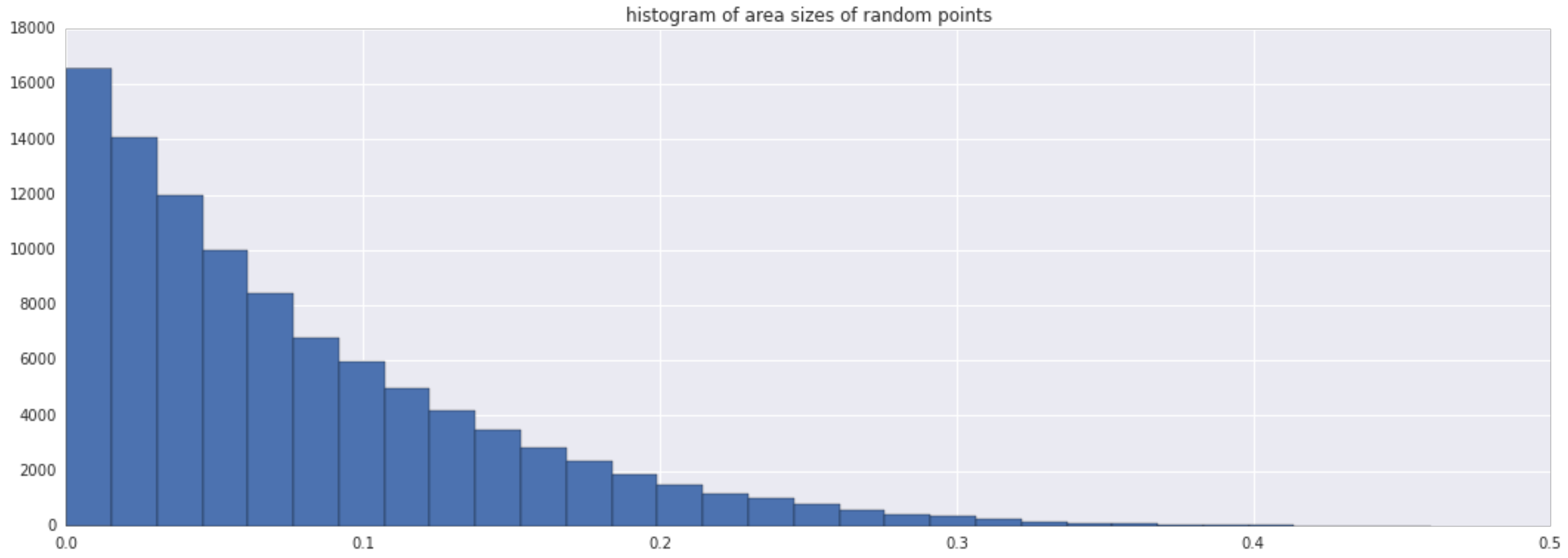
y_1
y_2
y_3
x_1
x_2
x_3
0.0
0.3
0.6
0.9

Learning step

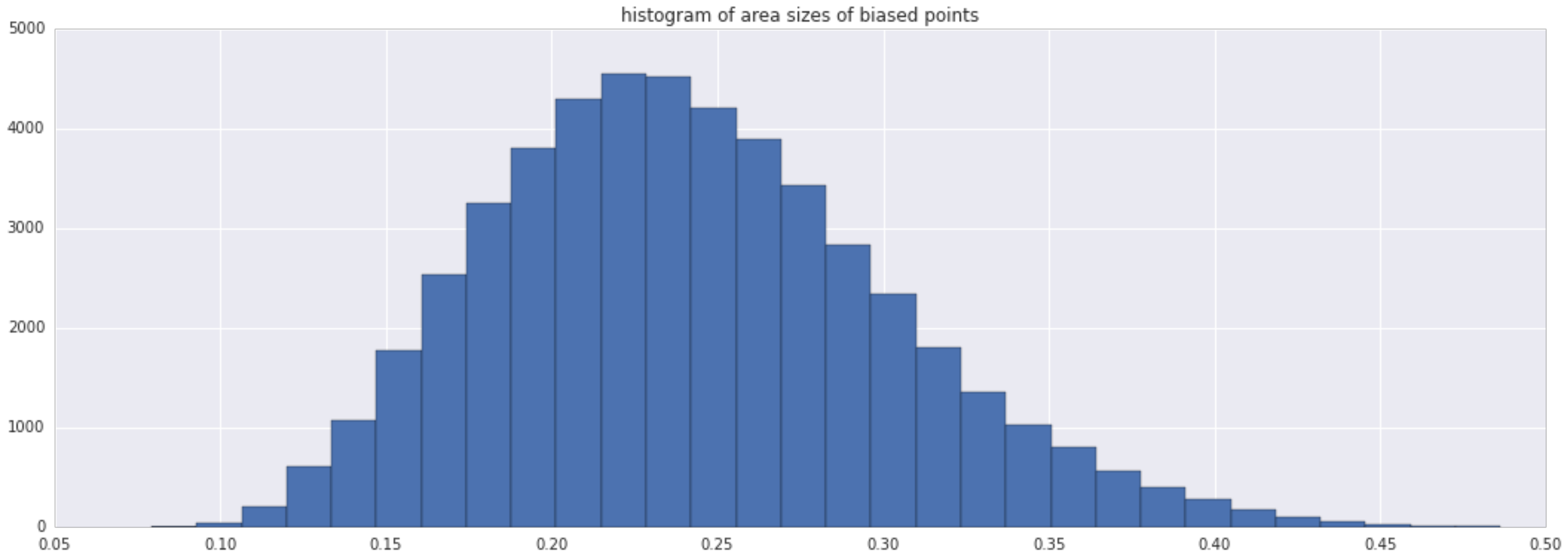
This Probabilistic approach allows us to learn many things from just sampling!

We've just shown $p(X|A > m)$, if we sample new coordinate points from this distribution, what does the new area histogram look like?

Before


histogram of area sizes of random points
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0.0
0.1
0.2
0.3
0.4
0.5

After


histogram of area sizes of biased points
5000
4000
3000
2000
1000
0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50

Next steps

We can repeat the same idea until some form of convergence. Note that m can be chosen in automatic fashions as well, ie. $m = \bar{A}$.

The cool bonus with these algorithms is that we can use inference on our simulated data to learn more about the nature of our optimisation problem.

You may notice a similarity with genetic algorithms here.

Inspiration for Feature Engineering

DARXOTV
HEROES
OF THE STORM™

Can we predict game outcomes?

I only know what each team got for characters. I may assume that some form of balancing already occured beforehand.

Problem: I only have 500.000 games in my dataset.

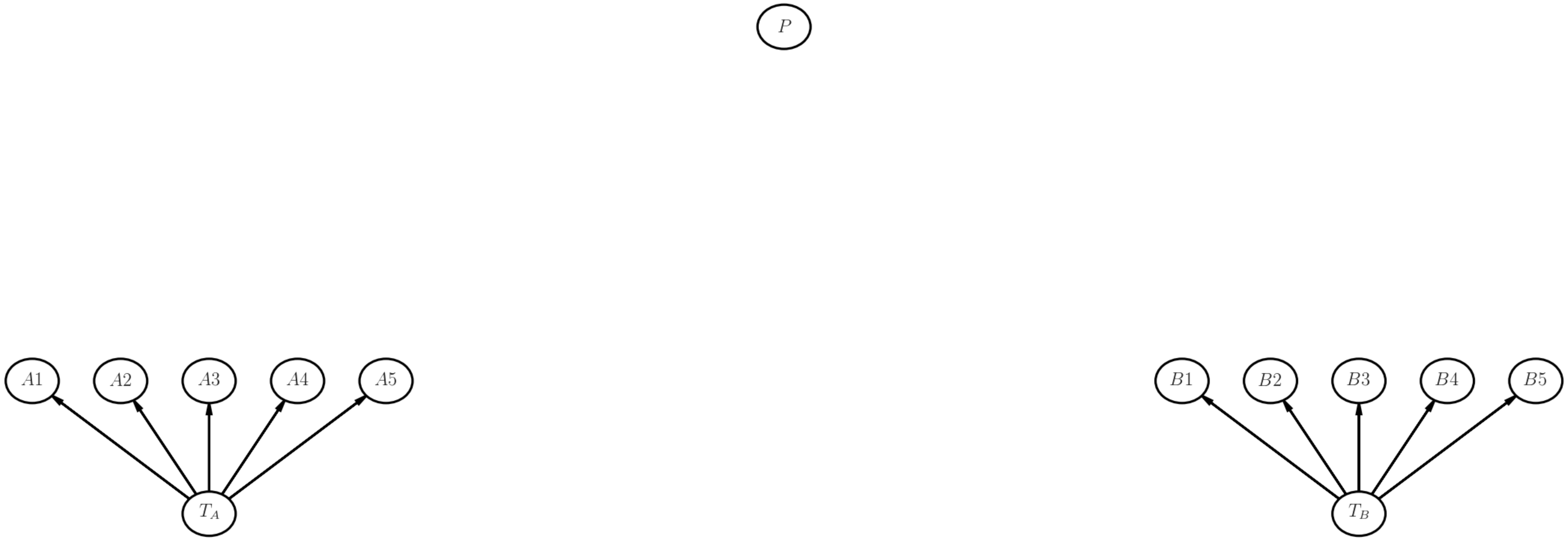
$$C \propto n^2(n-1)^2(n-2)^2(n-3)^2(n-4)^2 \approx O(n^{10}).$$

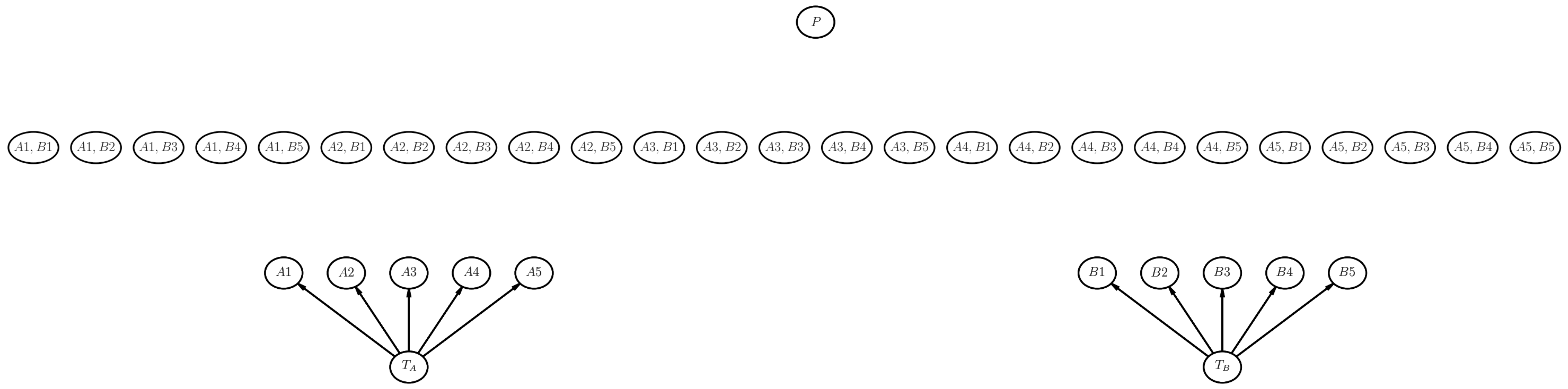
If we only have 6 heroes, search space consists of 518,400 possible allocations. The dataset dates when $n = 36$, though currently $n = 48$. Problem!

Can we predict game outcomes?

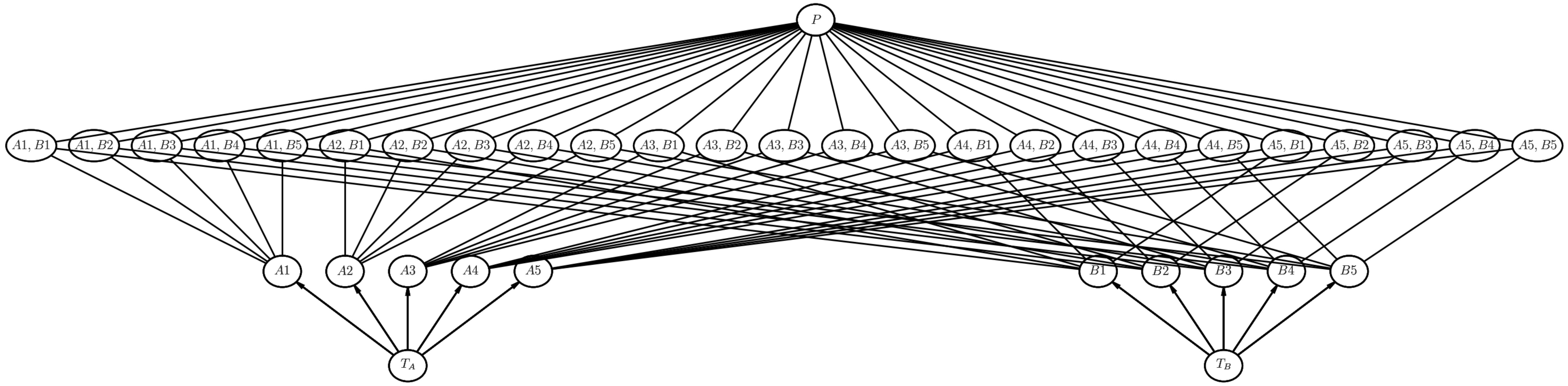
A lot of machine learning models are expected to fail here. We have 72 columns out of which every row will only have 10 items filled in. This dataset will be sparse even for random forests/deep neural nets.

Instead of trying hard to create features that might preform well, let's apply a little bit of domain knowledge to help us out Probabilistically.


P
$A1$
$A2$
$A3$
$A4$
$A5$
T_A
$B1$
$B2$
$B3$
$B4$
$B5$
T_B

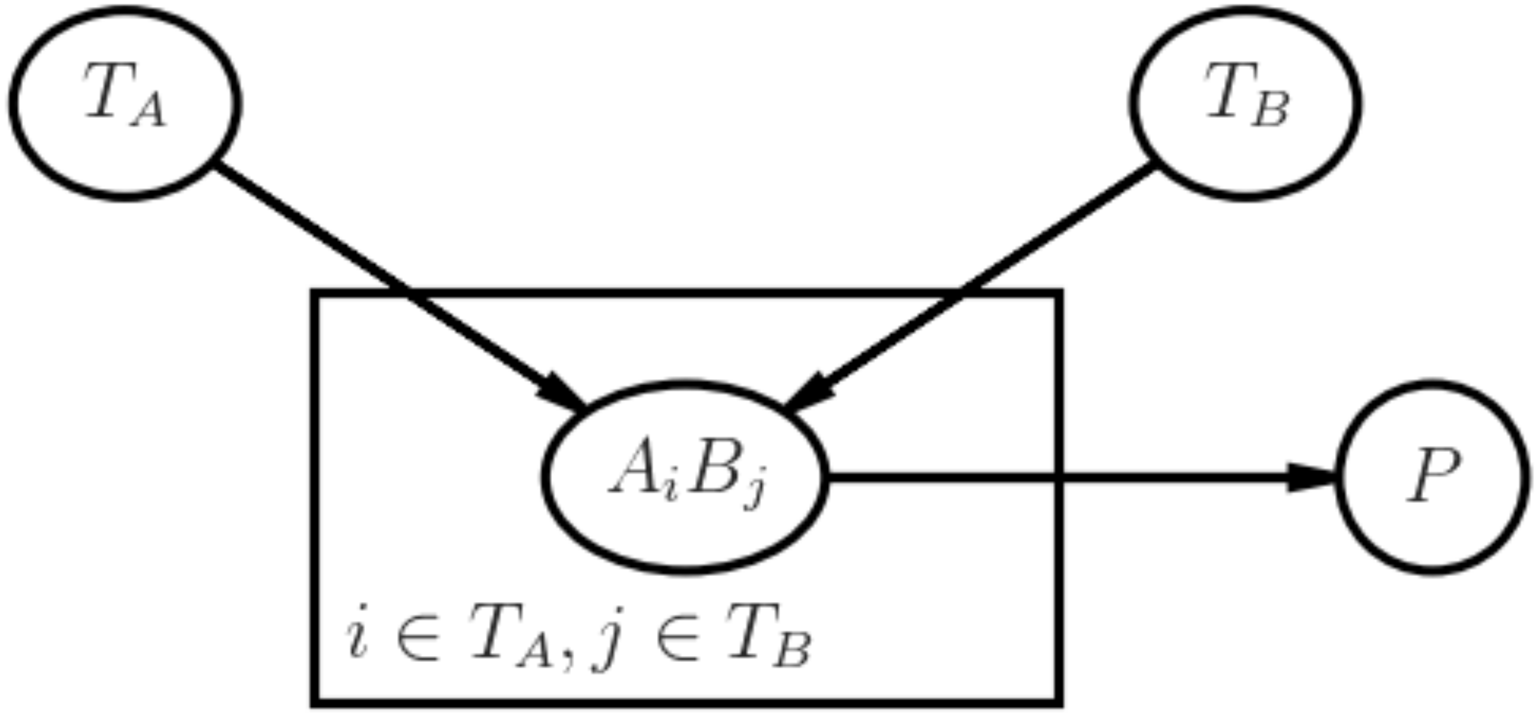


P
$A1, B1$
$A1, B2$
$A1, B3$
$A1, B4$
$A1, B5$
$A2, B1$
$A2, B2$
$A2, B3$
$A2, B4$
$A2, B5$
$A3, B1$
$A3, B2$
$A3, B3$
$A3, B4$
$A3, B5$
$A4, B1$
$A4, B2$
$A4, B3$
$A4, B4$
$A4, B5$
$A5, B1$
$A5, B2$
$A5, B3$
$A5, B4$
$A5, B5$
$A1$
$A2$
$A3$
$A4$
$A5$
T_A
$B1$
$B2$
$B3$
$B4$
$B5$
T_B

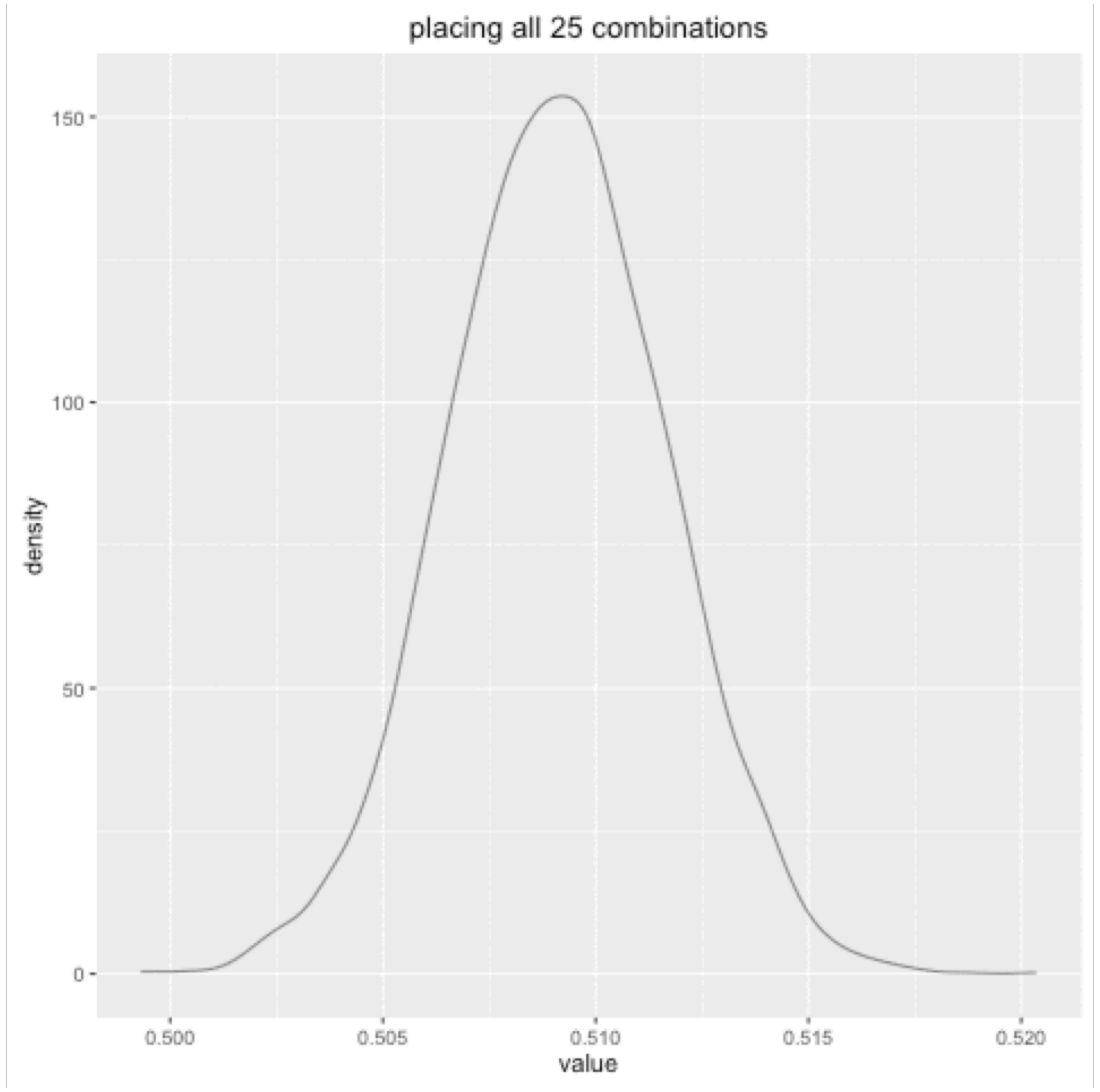


P
$A1, B1$
$A1, B2$
$A1, B3$
$A1, B4$
$A1, B5$
$A2, B1$
$A2, B2$
$A2, B3$
$A2, B4$
$A2, B5$
$A3, B1$
$A3, B2$
$A3, B3$
$A3, B4$
$A3, B5$
$A4, B1$
$A4, B2$
$A4, B3$
$A4, B4$
$A4, B5$
$A5, B1$
$A5, B2$
$A5, B3$
$A5, B4$
$A5, B5$
$A1$
$A2$
$A3$
$A4$
$A5$
$B1$
$B2$
$B3$
$B4$
$B5$
T_A
T_B

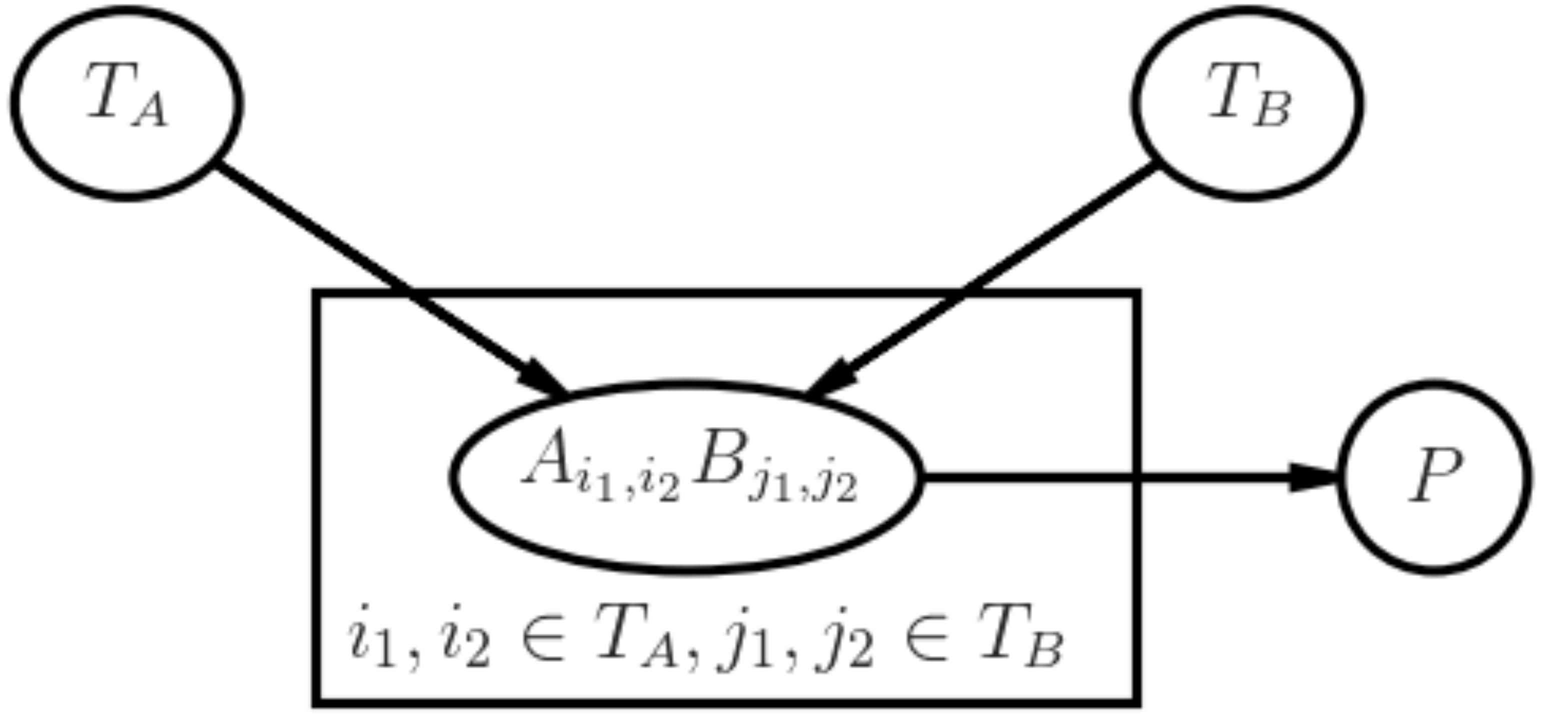
Clean model representation



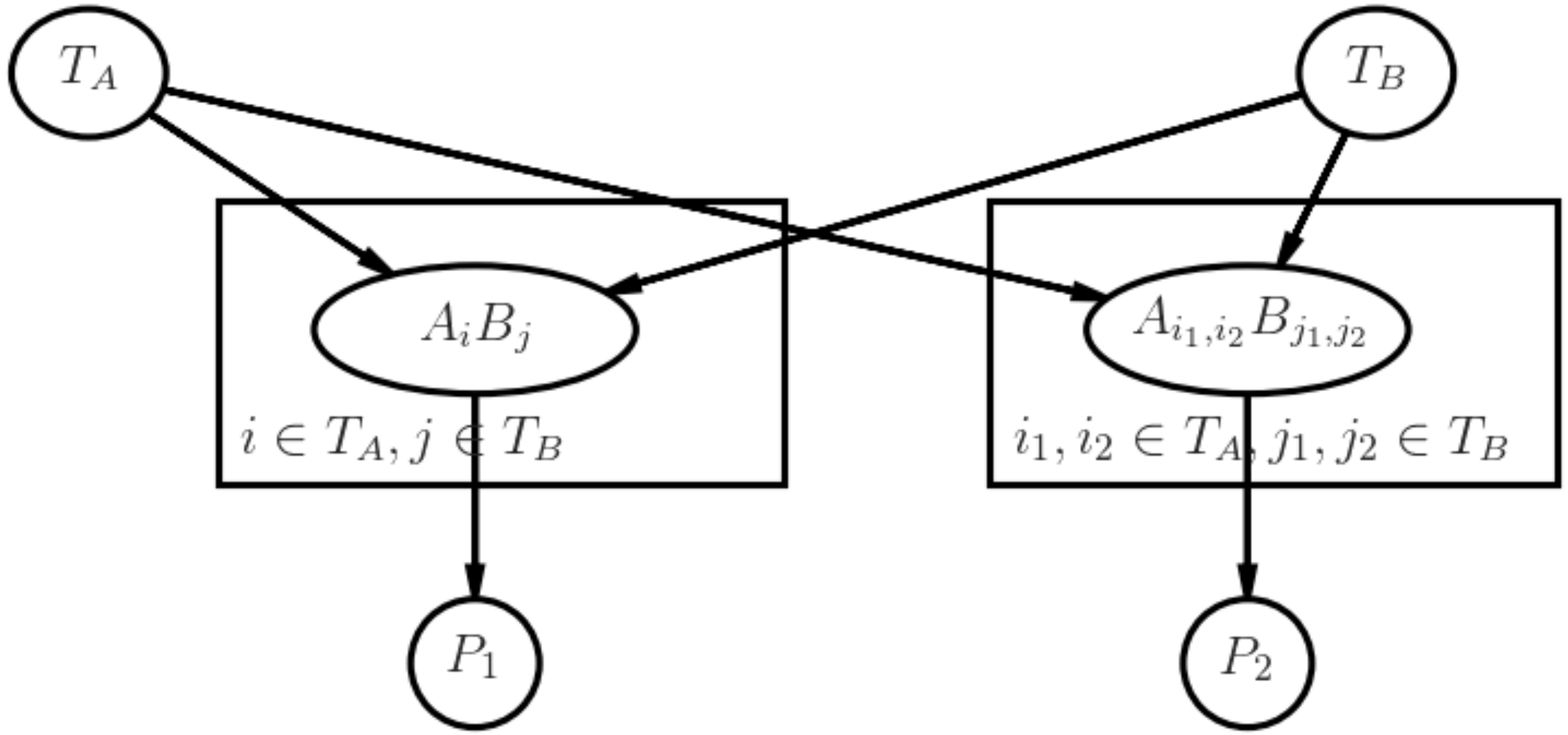
Combining Priors



Alternative model



Ensemble, Bayesian Convolution?


T_A
T_B
$A_i B_j$
$i \in T_A, j \in T_B$
$A_{i_1,i_2} B_{j_1,j_2}$
$i_1, i_2 \in T_A, j_1, j_2 \in T_B$
P_1
P_2

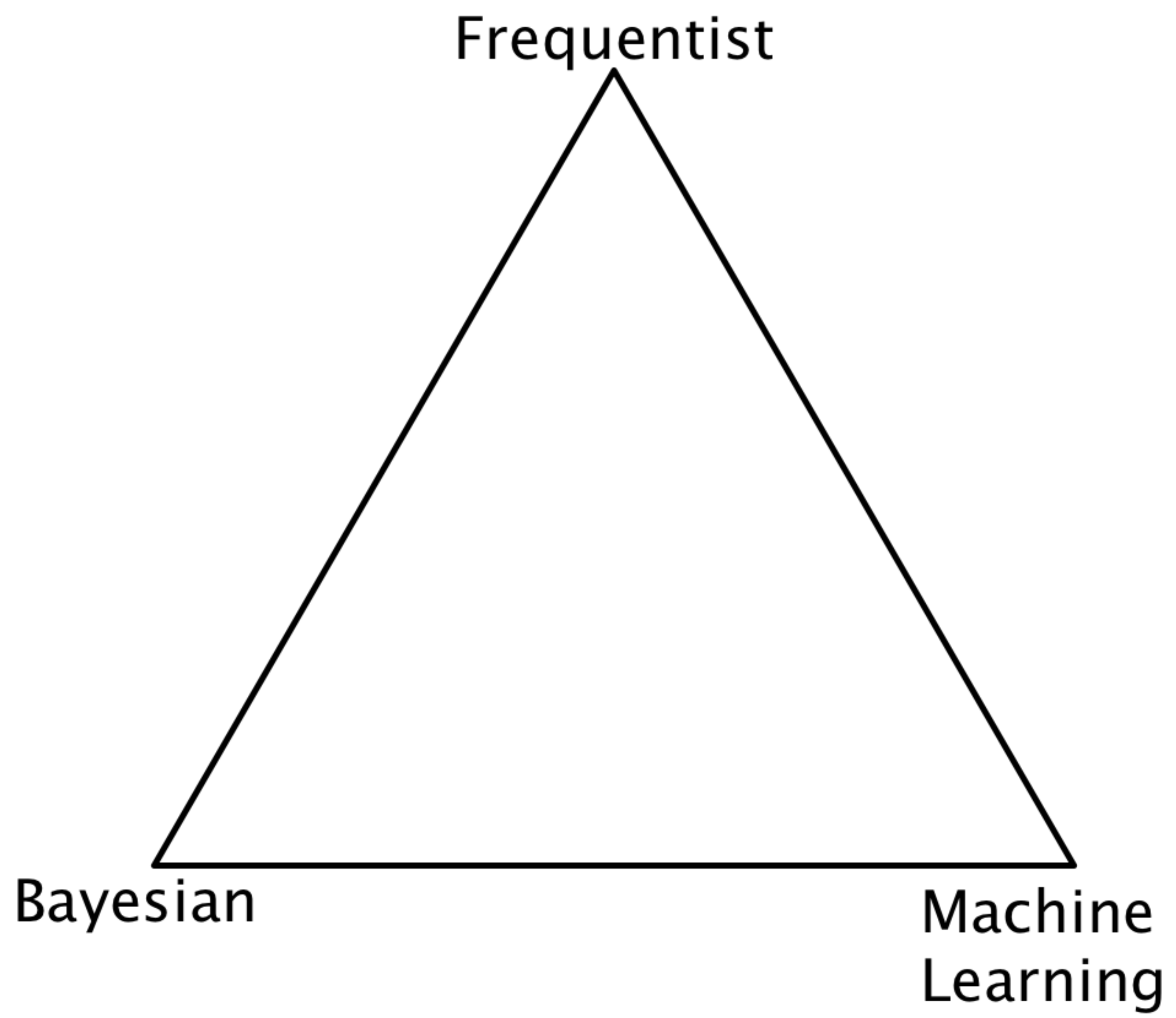
Conclusion by Comparing Schools

Different schools of data

Professionals as well as students often suggest to me that this way of Probabilistic thinking comes across as very new, if not different.

- no confidence intervals
- no p-values
- no talk about statistical significance
- no hypothesis tests

Different schools of data



Where is your uncertainty?

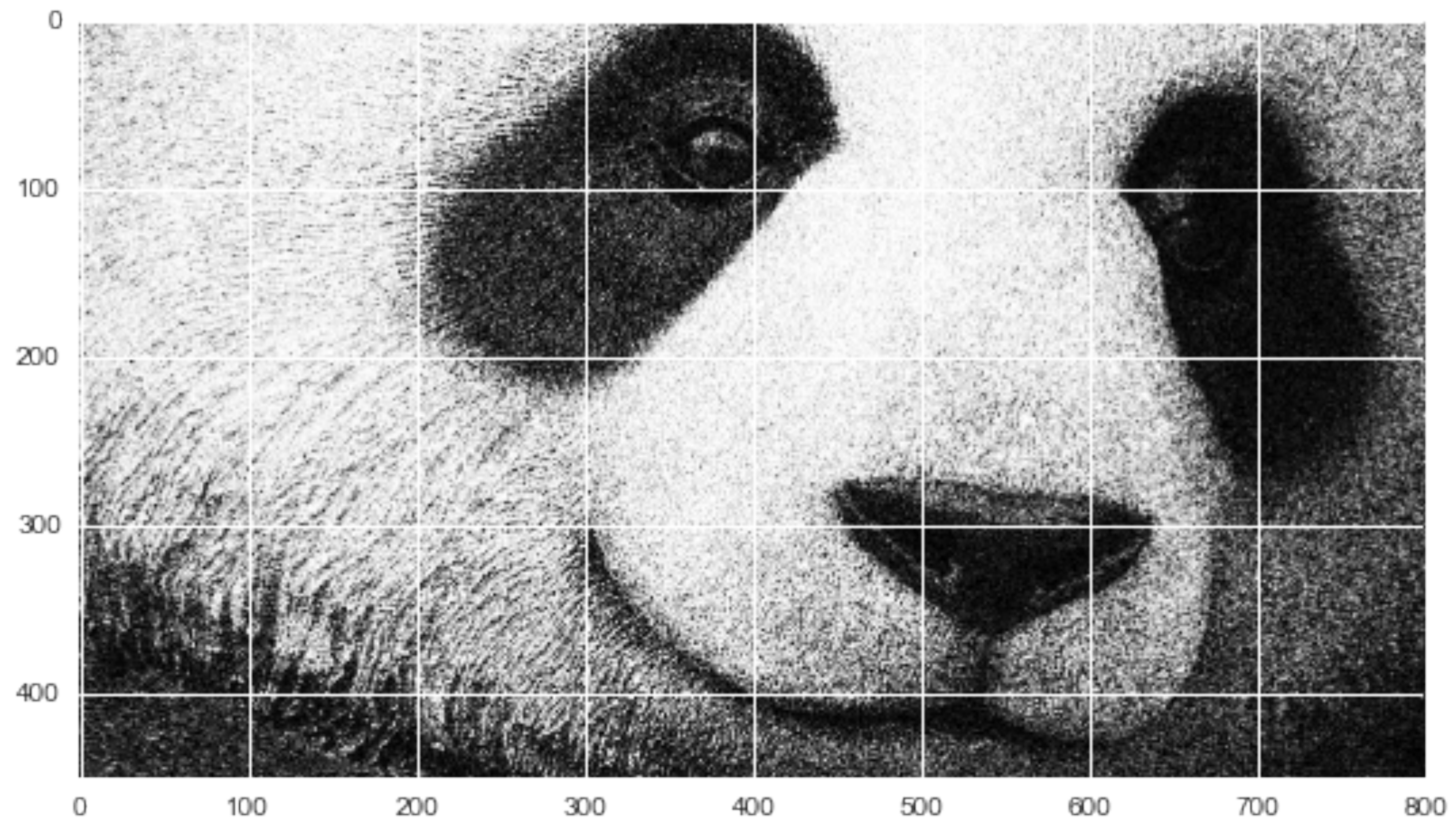
All models are wrong. Some models are useful.

— *Science*

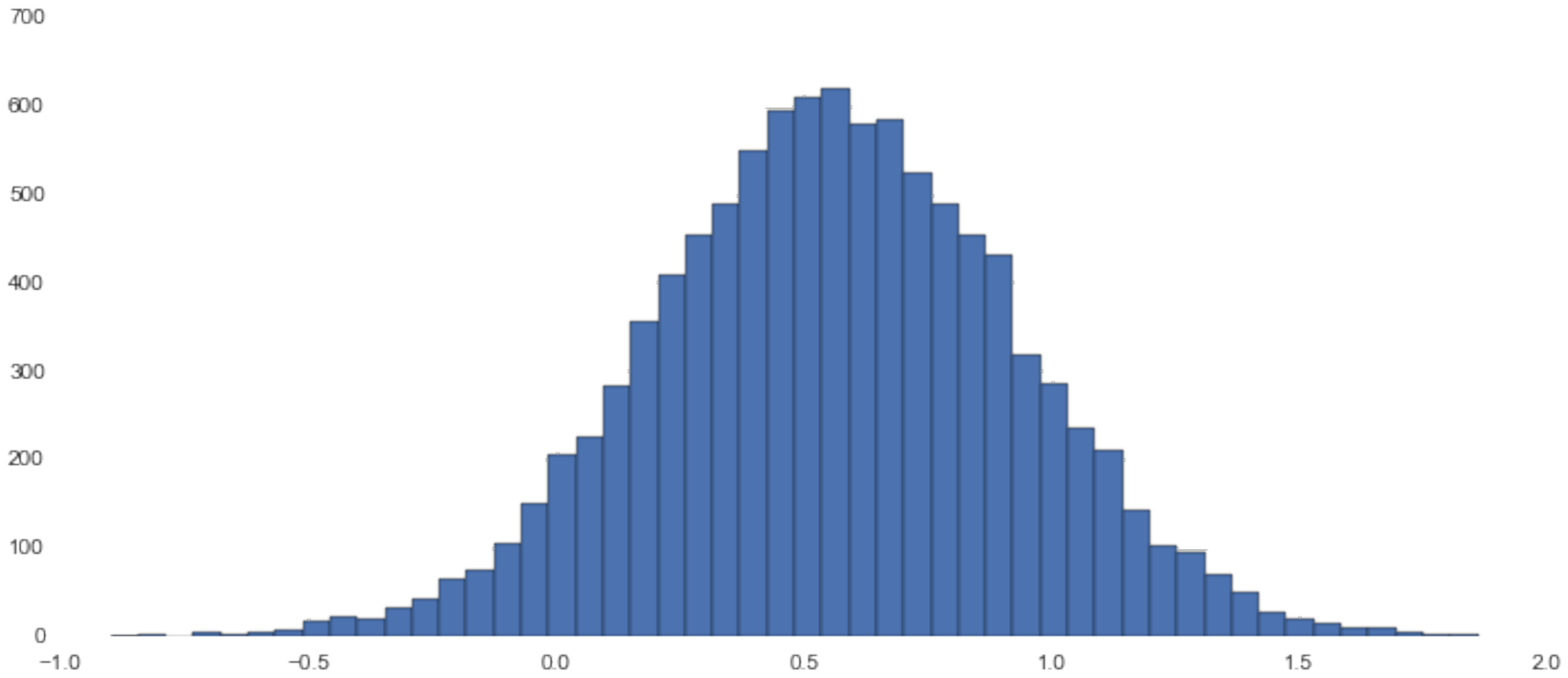
- Are you unsure about the data and are you trying to find the best single estimator to summerise this?
- Are you unsure about the model and do you accept the data to be the only truth available?
- Are you willing to ignore all uncertainties when your black box model performs very well against a lot of test sets?

How would you handle a panda?

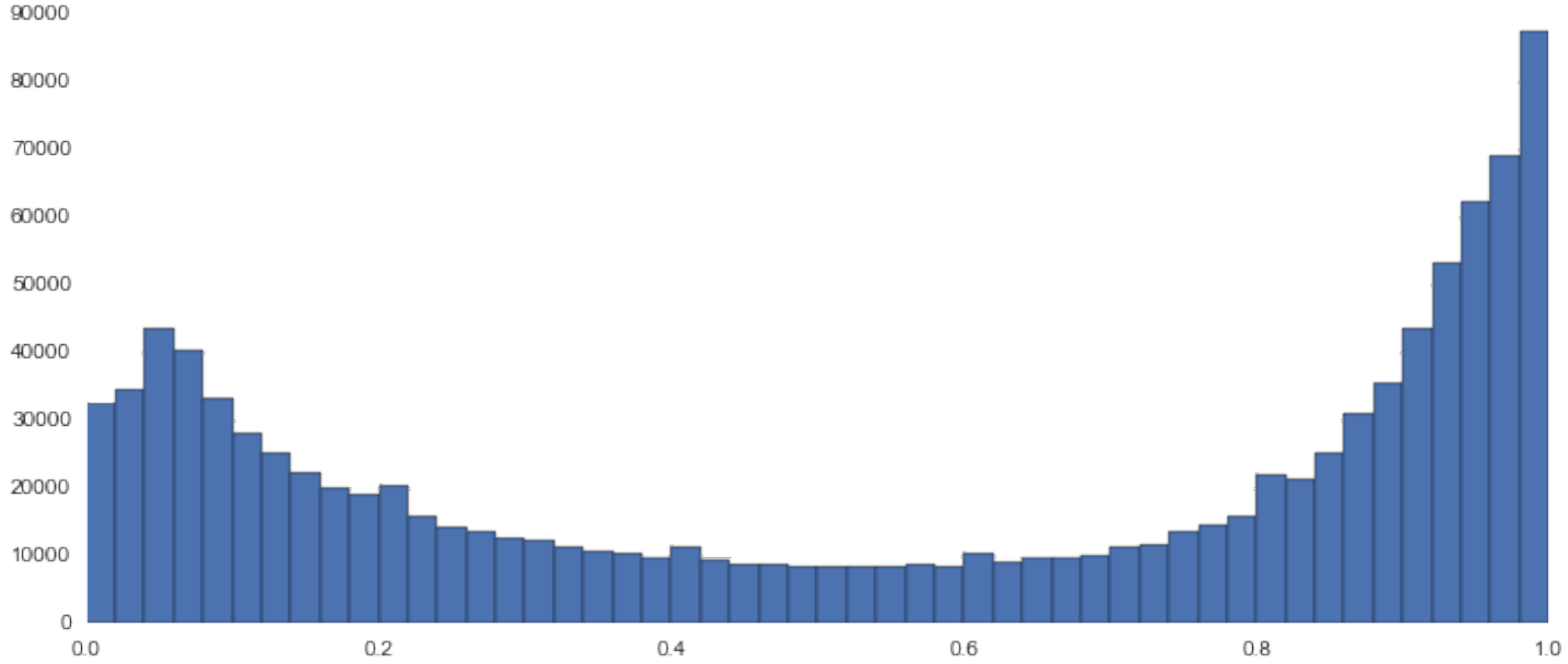
```
import matplotlib.image as mpimg
panda = mpimg.imread('panda-face.png')
```



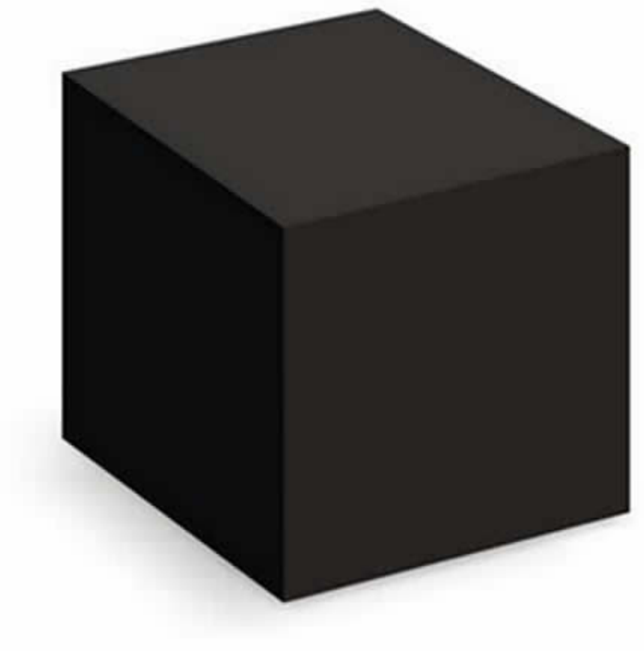
Some say the average panda is grey



Some describe the uncertainty of color



Others have this black box ...



... with awesome test set performance.

Never let your school get in the way of your education.

— *Mark Twain*

That's all folks

Thanks for listening!

Intermezzo

Those graphs were pretty, weren't they?

Also, did you think I did all those calculations by hand?

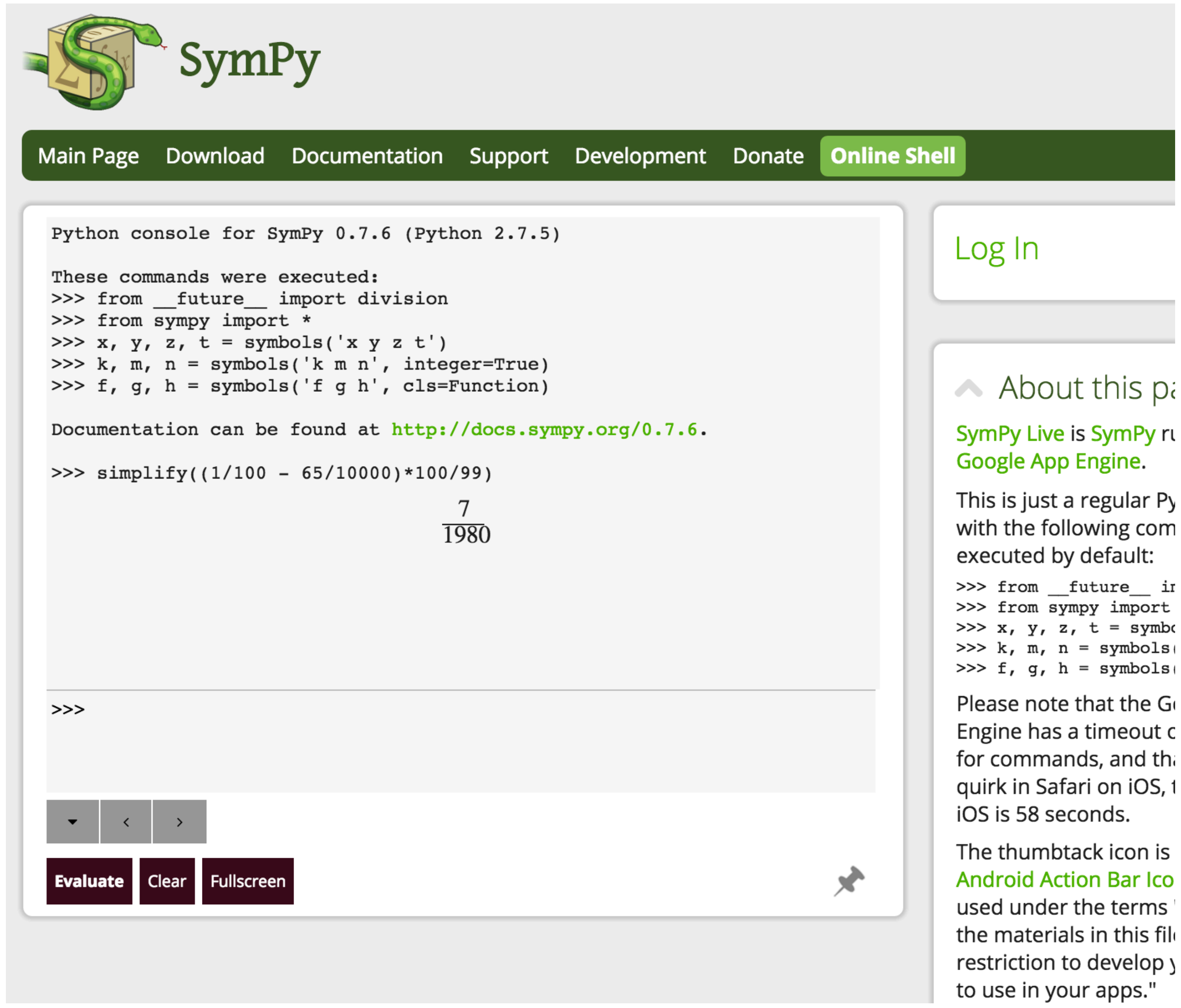
Graphs were made with **`daft`**

```python
from matplotlib import rc
import daft

rc("font", family="serif", size=12)
rc("text", usetex=True)

pgm = daft.PGM([5, 2.7], origin=[1.15, 0.65])
pgm.add_node(daft.Node("D", r"$D$", 3, 3, aspect=1.8))
pgm.add_node(daft.Node("T_1", r"$T_1$", 2, 2, aspect=1.2))
pgm.add_edge("D", "T_1")
pgm.render()
pgm.figure.savefig("illness1.png", dpi=150)
```

Simplifying calculations with **SymPy**!



```
Python console for SymPy 0.7.6 (Python 2.7.5)

These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)

Documentation can be found at http://docs.sympy.org/0.7.6.

>>> simplify((1/100 - 65/10000)*100/99)
```

$$\frac{7}{1980}$$

```
>>>
```

About this pa

SymPy Live is SymPy ru
Google App Engine.

This is just a regular Py
with the following com
executed by default:

```
>>> from __future__ ir
>>> from sympy import
>>> x, y, z, t = symbo
>>> k, m, n = symbols(
>>> f, g, h = symbols(
```

Please note that the G
Engine has a timeout c
for commands, and th
quirk in Safari on iOS, t
iOS is 58 seconds.

The thumbtack icon is
Android Action Bar Ico
used under the terms
the materials in this fil
restriction to develop y
to use in your apps."

Appendix: links to resources

- my blog will have slides at the bottom
- blog post on model selection