

Probabilistic Programming

Marius Popescu

popescunmarius@gmail.com

2019 - 2020

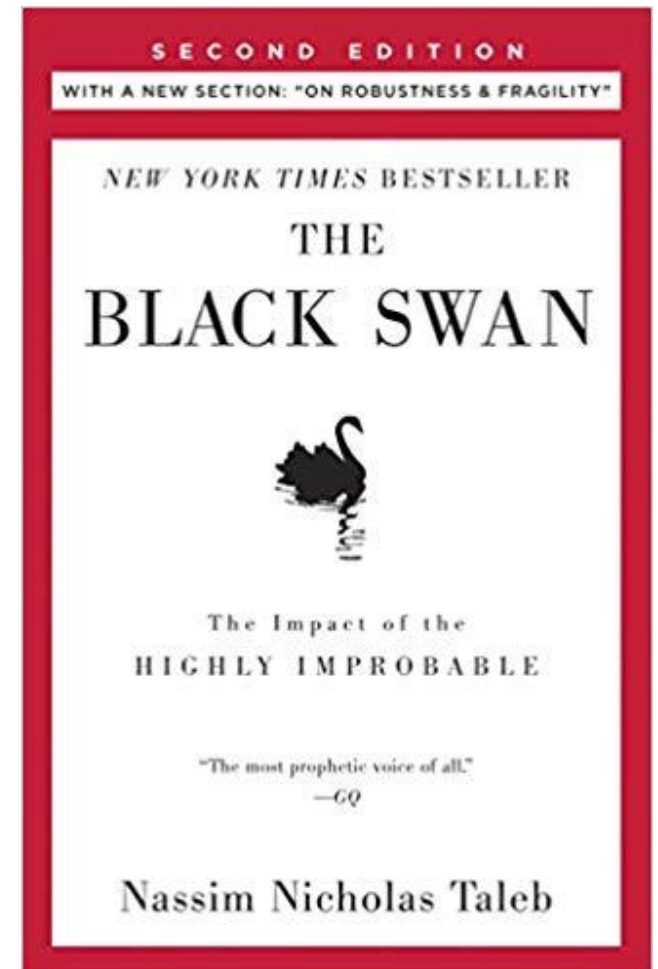
Loss Functions & Machine Learning

Bayesian Approach

Loss Functions

A meteorologist is predicting the probability of a possible hurricane striking his city. He estimates, with 95% confidence, that the probability of it not striking is between 99% - 100%. He is very happy with his precision and advises the city that a major evacuation is unnecessary. Unfortunately, the hurricane does strike and the city is flooded

Using a measure that emphasizes estimation accuracy, while an appealing and objective thing to do, misses the point of why you are even performing the statistical inference in the first place: results of inference. Furthermore, we'd like a method that stresses the importance of payoffs of decisions, not the accuracy of estimation alone. Nassim Taleb distills this quite succinctly: "I would rather be vaguely right than very precisely wrong."



Loss Functions

By shifting our focus from trying to be incredibly precise about parameter estimation to focusing on the outcomes of our parameter estimation, we can customize our estimates to be optimized for our application. This requires us to use loss functions that reflect our goals and outcomes.

A *loss function* is a function of the true parameter, and an estimate of that parameter:

$$L(\theta, \hat{\theta}) = f(\theta, \hat{\theta}), \quad f: \Theta \times \Theta \rightarrow \mathbb{R}^+$$

The important point of loss functions is that it measures how *bad* our current estimate is: the larger the loss, the worse the estimate is according to the loss function.

Loss Functions

Classic:

- squared-error loss: $L(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$
- absolute-loss: $L(\theta, \hat{\theta}) = |\theta - \hat{\theta}|$
- zero-one loss: $L(\theta, \hat{\theta}) = 1_{\theta \neq \hat{\theta}}$

Custom:

- $L(\theta, \hat{\theta}) = \frac{|\theta - \hat{\theta}|}{\theta(1-\theta)}$, $\theta, \hat{\theta} \in [0, 1]$ emphasizes an estimate closer to 0 or 1 since if the true value θ is near 0 or 1, the loss will be very large unless $\hat{\theta}$ is similarly close to 0 or 1
- $L(\theta, \hat{\theta}) = 1 - e^{-(\theta - \hat{\theta})^2}$ similar to the zero-one loss above, but not quite as penalizing to estimates that are close to the true

Loss Functions

Complicated non-linear loss functions can be programmed.

Weather forecasters have an interesting loss function for their predictions.

People notice one type of mistake — the failure to predict rain — more than other, false alarms. If it rains when it isn't supposed to, they curse the weatherman for ruining their picnic, whereas an unexpectedly sunny day is taken as a serendipitous bonus.

[The Weather Channel's bias] is limited to slightly exaggerating the probability of rain when it is unlikely to occur — saying there is a 20 percent chance when they know it is really a 5 or 10 percent chance — covering their butts in the case of an unexpected sprinkle.

new york times bestseller
noise and the noise
the signal and the noise
and the noise and the noise
the noise and the noise
why so many noise
predictions fail—
but some don't
and the noise and the noise
nate silver the noise

"Could turn out to be one of the more momentous books of the decade." —*The New York Times Book Review*



Bayesian Point Estimate

In Bayesian inference, we have a mindset that the unknown parameters are really random variables with prior and posterior distributions. Concerning the posterior distribution, a value drawn from it is a possible realization of what the true parameter could be. Given that realization, we can compute a loss associated with an estimate. As we have a whole distribution of what the unknown parameter could be (the posterior), we should be more interested in computing the *expected loss* given an estimate. This expected loss is a better estimate of the true loss than comparing the given loss from only a single sample from the posterior.

Bayesian Point Estimate

The systems and machinery present in the modern world are not built to accept posterior distributions as input. It is also rude to hand someone over a distribution when all they asked for was an estimate. In the course of an individual's day, when faced with uncertainty we still act by distilling our uncertainty down to a single action. Similarly, we need to distill our posterior distribution down to a single value (or vector in the multivariate case). If the value is chosen intelligently, we can avoid the flaw of frequentist methodologies that mask the uncertainty and provide a more informative result. The value chosen, if from a Bayesian posterior, is a *Bayesian point estimate*.

Bayesian Point Estimate

Suppose $P(\theta|X)$ is the posterior distribution of θ after observing data X , then the following function is understandable as the *expected loss* of choosing estimate $\hat{\theta}$ to estimate θ :

$$l(\hat{\theta}) = E_{\theta}[L(\theta, \hat{\theta})]$$

This is also known as the risk of estimate $\hat{\theta}$

How to approximate expected values? Given N samples θ_i , $i = 1, \dots, N$ from the posterior distribution, and a loss function L , we can approximate the expected loss of using estimate $\hat{\theta}$ by the Law of Large Numbers:

$$\frac{1}{N} \sum_{i=1}^N L(\theta_i, \hat{\theta}) \approx E_{\theta}[L(\theta, \hat{\theta})] = l(\hat{\theta})$$

Notice that measuring your loss via an expected value uses more information from the distribution than the MAP estimate which, if you recall, will only find the maximum value of the distribution and ignore the shape of the distribution. Ignoring information can over-expose yourself to tail risks, like the unlikely hurricane, and leaves your estimate ignorant of how ignorant you really are about the parameter.

Example: Financial Prediction

Suppose the future return of a stock price is very small, say 0.01 (or 1%). We have a model that predicts the stock's future price, and our profit and loss is directly tied to us acting on the prediction. How should we measure the loss associated with the model's predictions, and subsequent future predictions? A squared-error loss is agnostic to the signage and would penalize a prediction of -0.01 equally as bad a prediction of 0.03:

$$(0.01 - (-0.01))^2 = (0.01 - 0.03)^2 = 0.004$$

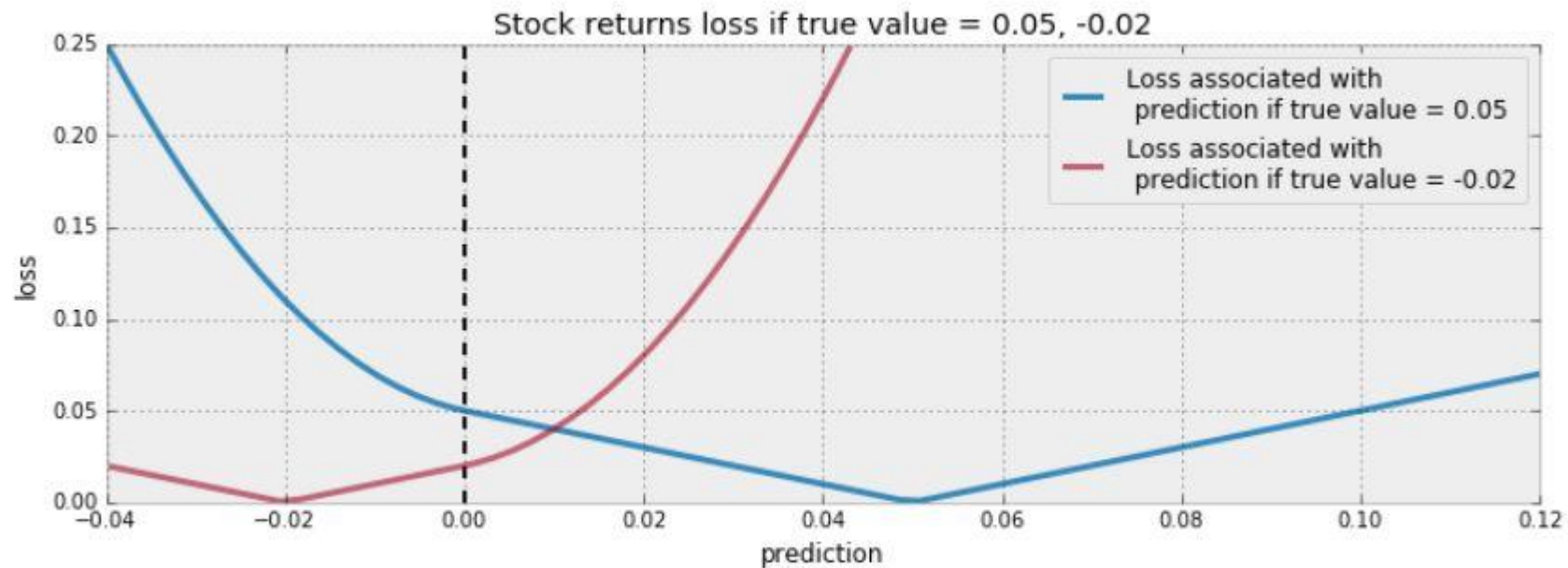
If you had made a bet based on your model's prediction, you would have earned money with a prediction of 0.03, and lost money with a prediction of -0.01, yet our loss did not capture this. We need a better loss that takes into account the sign of the prediction and true value.

A New Loss that is Better for Financial Applications

Let's design a new loss that is better for financial applications:

```
def stock_loss(true_return, yhat, alpha=100.):  
    if true_return * yhat < 0:  
        # opposite signs, not good  
        return alpha * yhat ** 2 - np.sign(true_return) * yhat \  
            + abs(true_return)  
    else:  
        return abs(true_return - yhat)
```

A New Loss that is Better for Financial Applications



Note the change in the shape of the loss as the prediction crosses zero. This loss reflects that the user really does not want to guess the wrong sign, especially be wrong and a large magnitude.

Why would the user care about the magnitude? Why is the loss not 0 for predicting the correct sign? Surely, if the return is 0.01 and we bet millions we will still be (very) happy.

The Data

Our dataset is artificial, as most financial data is not even close to linear.

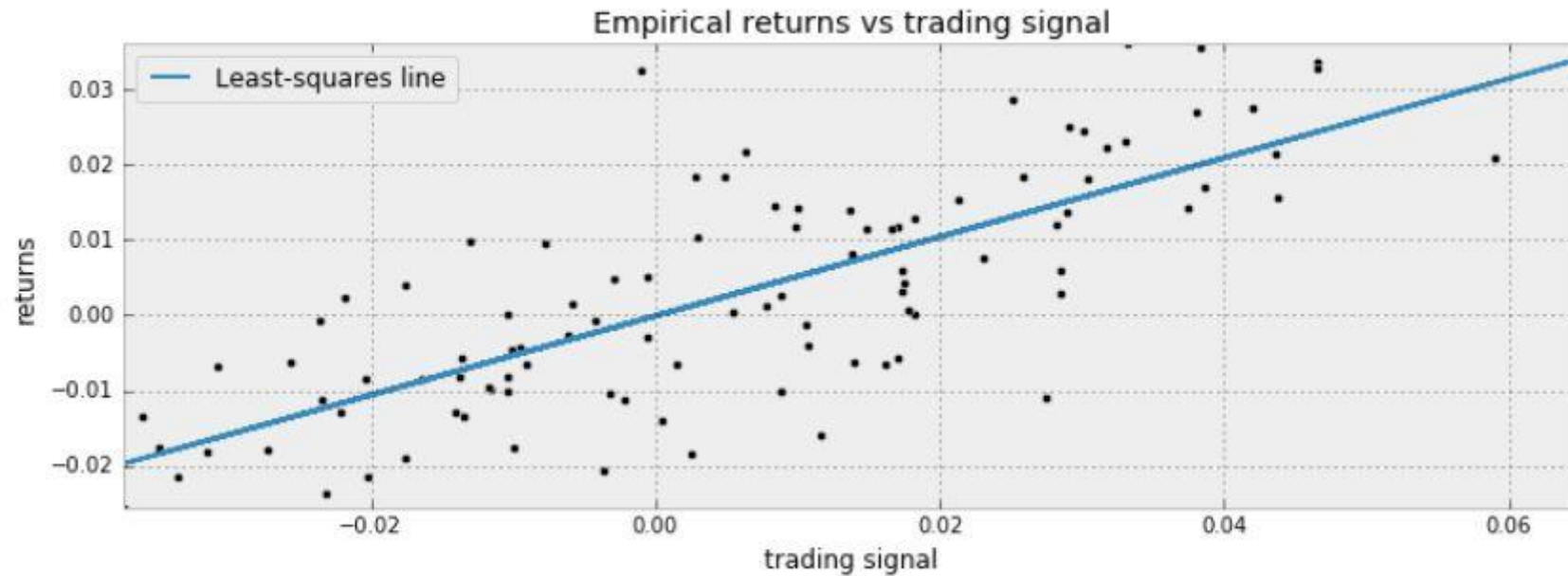
```
N = 100
```

```
X = 0.025 * np.random.randn(N)
```

```
Y = 0.5 * X + 0.01 * np.random.randn(N)
```

```
ls_coef_ = np.cov(X, Y)[0, 1] / np.var(X)
```

```
ls_intercept = Y.mean() - ls_coef_ * X.mean()
```



Bayesian Regression on this dataset

```
std = pm.Uniform("std", 0, 100, trace=False)
```

```
@pm.deterministic
```

```
def prec(U=std):
```

```
    return 1.0 / (U) ** 2
```

```
beta = pm.Normal("beta", 0, 0.0001)
```

```
alpha = pm.Normal("alpha", 0, 0.0001)
```

```
@pm.deterministic
```

```
def mean(X=X, alpha=alpha, beta=beta):
```

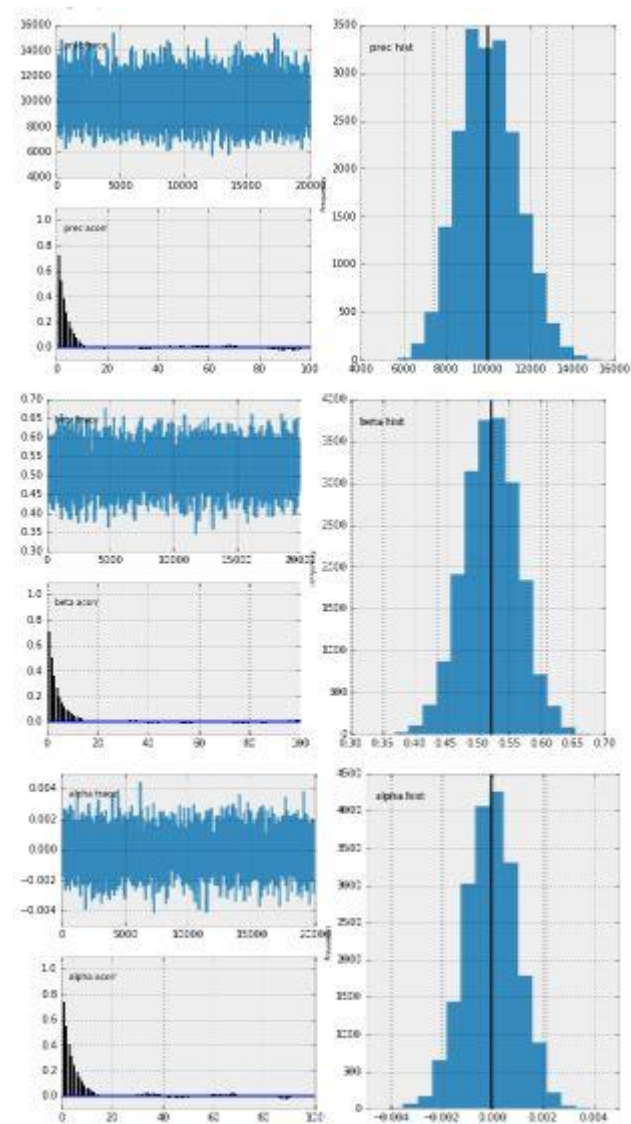
```
    return alpha + beta * X
```

```
obs = pm.Normal("obs", mean, prec, value=Y, observed=True)
```

```
mcmc = pm.MCMC([obs, beta, alpha, std, prec])
```

```
mcmc.sample(100000, 80000)
```

```
#pm.Matplotlib.plot(mcmc)
```



Best Prediction with Respect to our Loss

For a specific trading signal, call it x , the distribution of possible returns has the form:

$$R_i(x) = \alpha_i + \beta_i x + \varepsilon$$

where $\varepsilon \sim N\left(0, \frac{1}{\tau}\right)$ and i indexes our posterior samples.

We wish to find the solution to:

$$\operatorname{argmin}_r E_{R(x)}[L(R(x), r)]$$

according to the loss given above. This r is our Bayes prediction for trading signal x .

Best Prediction with Respect to our Loss

```
def stock_loss(price, pred, coef=500):  
    """vectorized for numpy"""  
    sol = np.zeros_like(price)  
    ix = price * pred < 0  
    sol[ix] = coef * pred ** 2 - np.sign(price[ix]) * pred + abs(price[ix])  
    sol[~ix] = abs(price[~ix] - pred)  
    return sol
```


Best Prediction with Respect to our Loss

```
from scipy.optimize import fmin

tau_samples = mcmc.trace("prec")[:]
alpha_samples = mcmc.trace("alpha")[:]
beta_samples = mcmc.trace("beta")[:]

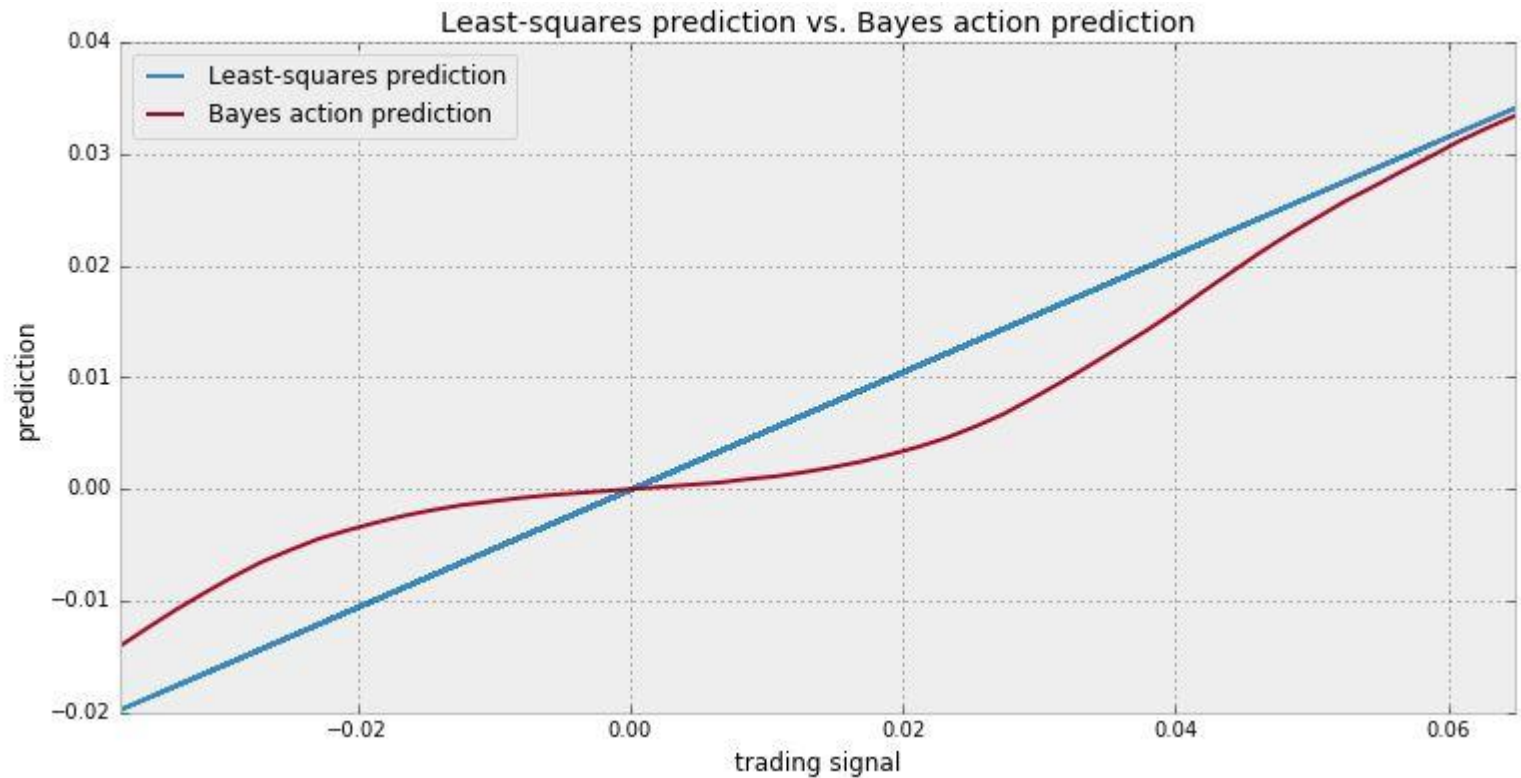
N = tau_samples.shape[0]

noise = 1. / np.sqrt(tau_samples) * np.random.randn(N)

possible_outcomes = lambda signal: alpha_samples + beta_samples * signal + noise

opt_predictions = np.zeros(50)
trading_signals = np.linspace(X.min(), X.max(), 50)
for i, _signal in enumerate(trading_signals):
    _possible_outcomes = possible_outcomes(_signal)
    tomin = lambda pred: stock_loss(_possible_outcomes, pred).mean()
    opt_predictions[i] = fmin(tomin, 0, disp=False)
```

Our Prediction



Interesting, when the signal is near 0, and many of the possible returns are possibly both positive and negative, our best (with respect to our loss) move is to predict close to 0; that is, take on no position. Only when we are very confident do we enter into a position. As the signal becomes more and more extreme, and we feel more and more confident about the positiveness / negativeness of returns, our position converges with that of the least-squares line.

Two Machine Learning Success Stories

One Place: Kaggle

One Man: Tim Salimans



Kaggle posted a blog

2 days ago



Data Notes: Impact of Game of Thrones on US Baby Names

Welcome to Kaggle Data Notes! Are Game of Thrones fans naming their babies Sansa and Tyrion? Enjoy these new, intriguing and overlooked datasets and kernels

1.?? Impact of Game of Thrones on US Baby Names (Link) 2. ?? Skin Lesion Analyzer + Tensorflow.js Web App (Link) 3. ?? Where do People Learn ML / DS? (Link) 4. ?? What Makes a Kagglers Valuable? (Link) 5. ?? What Software do Kagglers Use to



Marius Popescu

Joined 6 years ago

⚙️ Competitions Master

- ☐ 1/5 gold medals
- ☐ Solo gold medal

⚙️ Kernels Contributor

- ☐ 0/5 bronze medals

⚙️ Discussion Contributor

- ☐ 0/50 bronze medals



Amplitude Inc. is hiring

Senior Data Scientist - PRODUCT ANALYTICS

📍 San Francisco, CA, USA



Tim Salimans

Netherlands

Joined 8 years ago · last seen 7 months ago

[in](#)

Followers 50



Competitions
Grandmaster

[Home](#) [Competitions \(18\)](#) [Discussion \(34\)](#) [Followers \(50\)](#)

[Contact User](#)

[Follow User](#)

Competitions Grandmaster



Current Rank

675

of 93,434

Highest Rank

2



8



2



1

[Observing Dark Worlds](#)

🥇 · 6 years ago · Top 1%

1st

of 353

[Deloitte/FIDE Chess Ratin...](#)

1st

Kernels Contributor



Unranked



0



0



0

No kernel results

Discussion Contributor



Unranked



1



1



5

[2nd place solution](#)

🥈 · 4 years ago

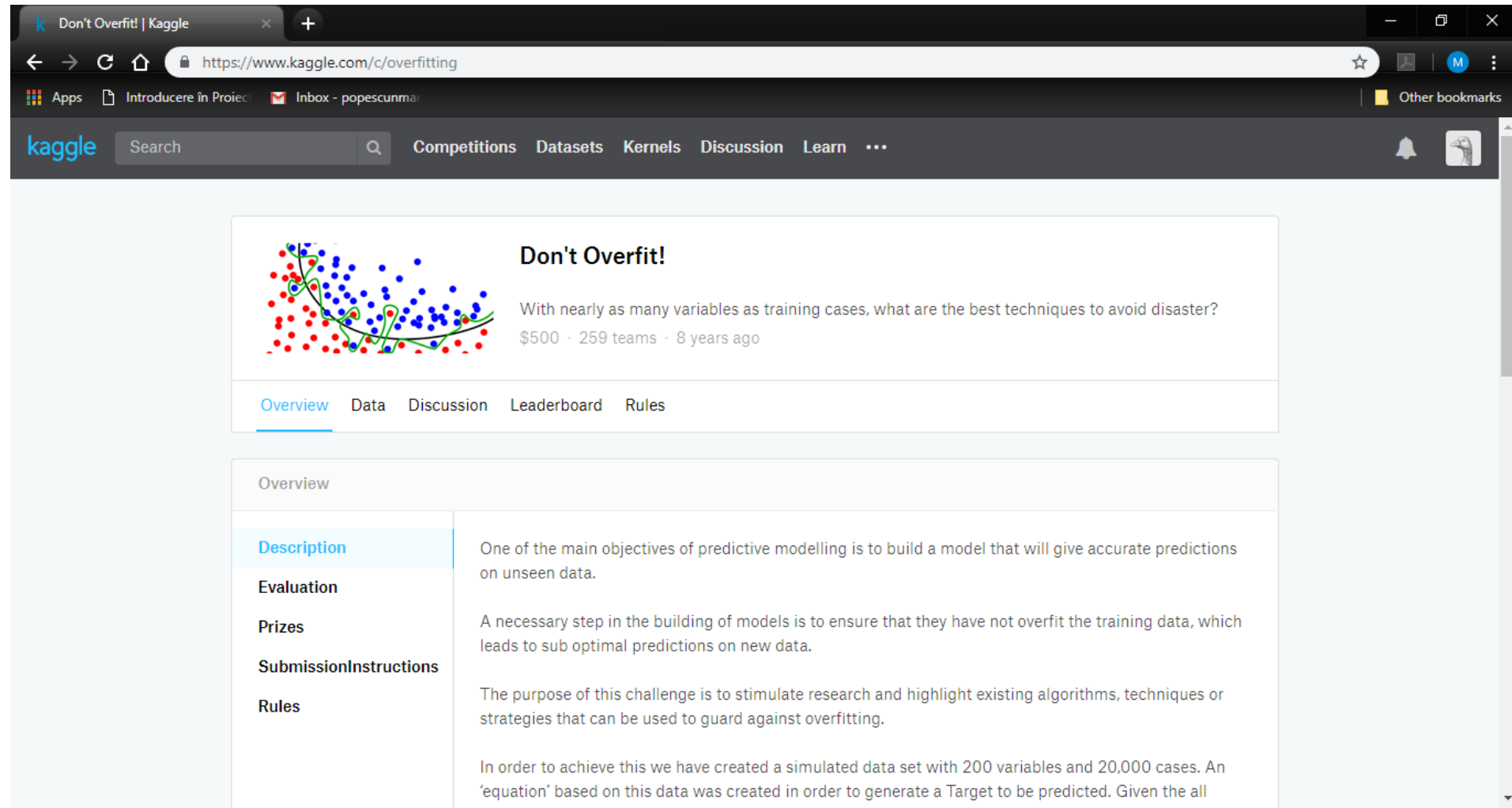
18

votes

[Solutions](#)

10

Between February 28, 2011 and May 15, 2011, Kaggle hosted a prediction competition entitled “Don’t Overfit!”



The screenshot shows the Kaggle website interface for the "Don't Overfit!" competition. The browser address bar displays the URL <https://www.kaggle.com/c/overfitting>. The page header includes the Kaggle logo, a search bar, and navigation links for Competitions, Datasets, Kernels, Discussion, and Learn. The competition title "Don't Overfit!" is prominently displayed, accompanied by a scatter plot visualization with red and blue data points and green lines representing model fits. Below the title, the competition details are listed: "With nearly as many variables as training cases, what are the best techniques to avoid disaster?", "\$500 · 259 teams · 8 years ago". A navigation bar below the title contains links for Overview, Data, Discussion, Leaderboard, and Rules. The "Overview" section is expanded, showing a table of contents with links to Description, Evaluation, Prizes, SubmissionInstructions, and Rules. The "Description" section is currently selected, providing a detailed explanation of the competition's purpose and data.

Don't Overfit!

With nearly as many variables as training cases, what are the best techniques to avoid disaster?
\$500 · 259 teams · 8 years ago

[Overview](#) [Data](#) [Discussion](#) [Leaderboard](#) [Rules](#)

Overview

Description	One of the main objectives of predictive modelling is to build a model that will give accurate predictions on unseen data.
Evaluation	
Prizes	A necessary step in the building of models is to ensure that they have not overfit the training data, which leads to sub optimal predictions on new data.
SubmissionInstructions	
Rules	The purpose of this challenge is to stimulate research and highlight existing algorithms, techniques or strategies that can be used to guard against overfitting.

In order to achieve this we have created a simulated data set with 200 variables and 20,000 cases. An 'equation' based on this data was created in order to generate a Target to be predicted. Given the all

Don't Overfit!

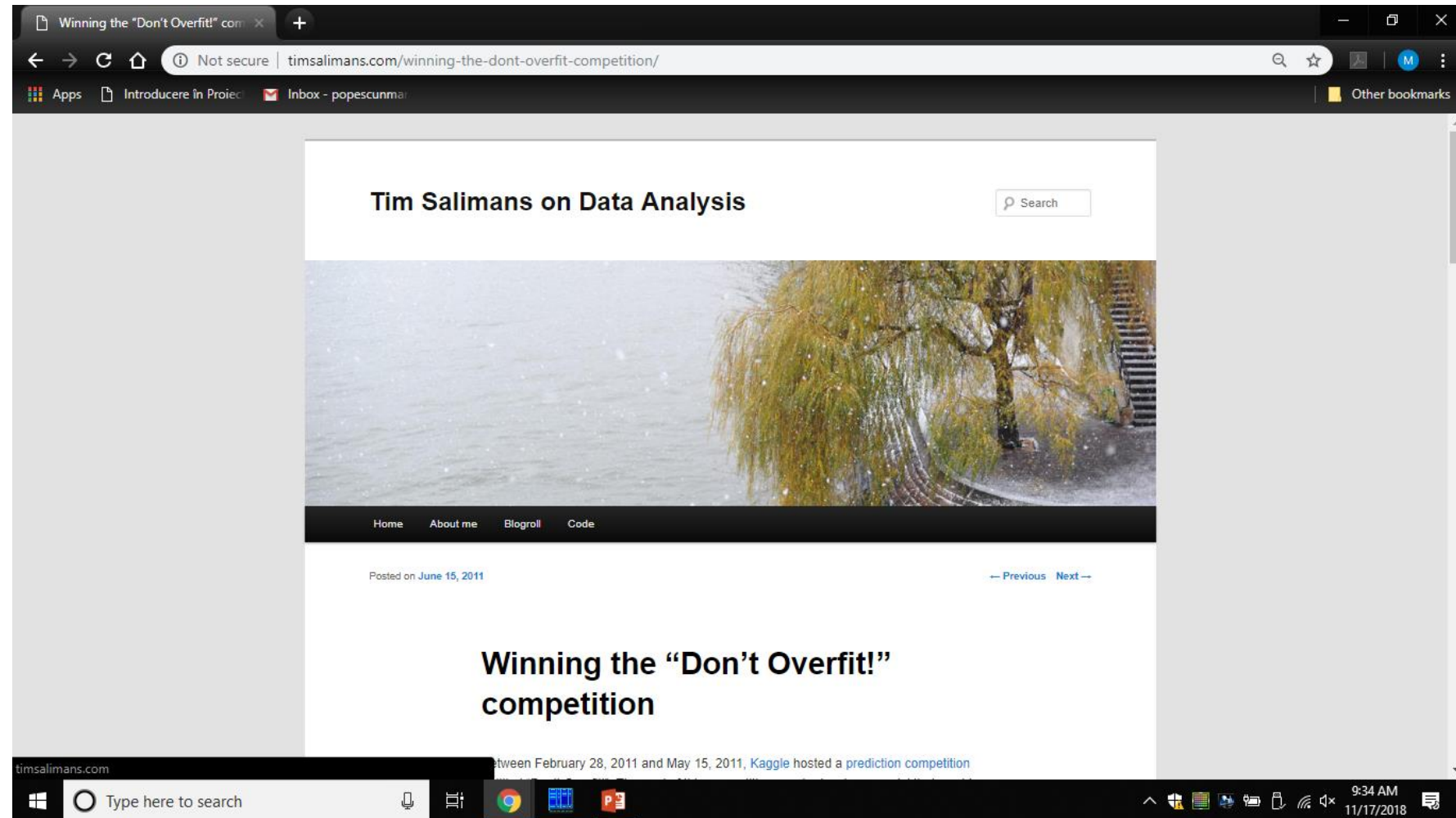
The goal of this competition was to develop a model that would predict well in a setting where you have little data and many explanatory variables.

“The purpose of this challenge is to stimulate research and highlight existing algorithms, techniques or strategies that can be used to guard against overfitting.

In order to achieve this we have created a simulated data set with 200 variables and 20,000 cases. An ‘equation’ based on this data was created in order to generate a Target to be predicted. Given the all 20,000 cases, the problem is very easy to solve – but you only get given the Target value of 250 cases – the task is to build a model that gives the best predictions on the remaining 19,750 cases.”

Winning the “Don’t Overfit!” Competition

“The question “how do we prevent overfitting?” has been asked again and again, but in my opinion the answer has been known since Laplace: **use Bayesian analysis with a sensible prior.**”



Winning the “Don’t Overfit!” Competition

“

I started this contest with very little time left, but fortunately the other participants had already left me lots of clues in the forum. In particular, a quick read revealed the following:

- The “equation” used to generate the data seemed to be linear
- The coefficient of the explanatory variables all seemed to be of the same sign
- According to Phil [Brierley the competition organizer] the “equation” did not have any noise in it

”

Winning the “Don’t Overfit!” Competition

“

Based on these clues and some experimentation, I guessed that the data was generated as follows:

1. Sample the 200 explanatory variables ‘X’ uniformly on [0,1]
2. With probability 0.5 select each different X variable for use in the “equation”
3. For each included variable uniformly sample a coefficient A
4. Define $Y = A_1 * X_1 + A_2 * X_2$ etc
5. Define $Z = Y - \text{mean}(Y)$
6. Set $T_i = 1$ if $Z_i < 0$ and set $T_i = 0$ otherwise
7. Round all X variables to 3 decimal places

”

Don't Overfit! In PyMC

```
d = 200
n = 20000 # number of data points
x_data = np.random.random((n,d))
A = np.random.random(d)
isel = np.random.binomial(1, 0.5, d)
A = A * isel
y_data = np.dot(x_data, A)
y_data = y_data - y_data.mean()
y_data = 0.5 * (np.sign(y_data) + 1)

it = np.random.choice(20000, 250)

ix_training = it
ix_testing = np.ones(20000)
ix_testing[it] = 0
ix_testing = ix_testing == 1

training_data = x_data[ix_training]
testing_data = x_data[ix_testing]
training_labels = y_data[ix_training]
testing_labels = y_data[ix_testing]
```

Don't Overfit! In PyMC

```
to_include = pm.Bernoulli("to_include", 0.5, size=200)
```

```
coef = pm.Uniform("coefs", 0, 1, size=200)
```

```
@pm.deterministic
```

```
def Z(coef=coef, to_include=to_include, data=training_data):
```

```
    ym = np.dot(to_include * training_data, coef)
```

```
    return ym - ym.mean()
```

```
@pm.deterministic
```

```
def T(z=Z):
```

```
    return 0.45 * (np.sign(z) + 1.1)
```

```
obs = pm.Bernoulli("obs", T, value=training_labels, observed=True)
```

```
model = pm.Model([to_include, coef, Z, T, obs])
```

```
map_ = pm.MAP(model)
```

```
map_.fit()
```

```
mcmc = pm.MCMC(model)
```

```
mcmc.sample(1000000, 990000)
```

The Results

```
0.976
0.980734767025
0.972
0.843077027826
0.758189458762
```

```
t_trace = mcmc.trace("T")[:]
include_trace = mcmc.trace("to_include")[:]
coef_trace = mcmc.trace("coefs")[:]

print((np.round(t_trace.mean(axis=0)) == training_labels).mean())

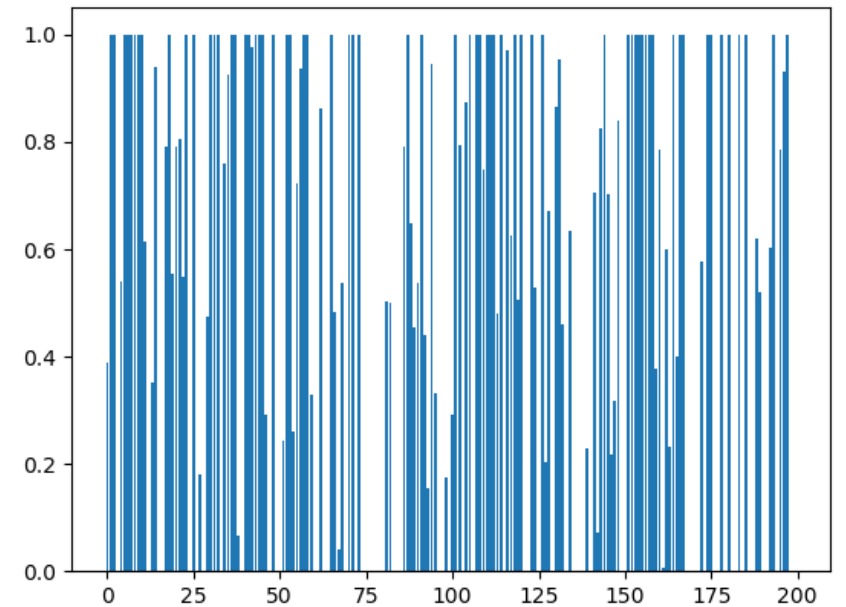
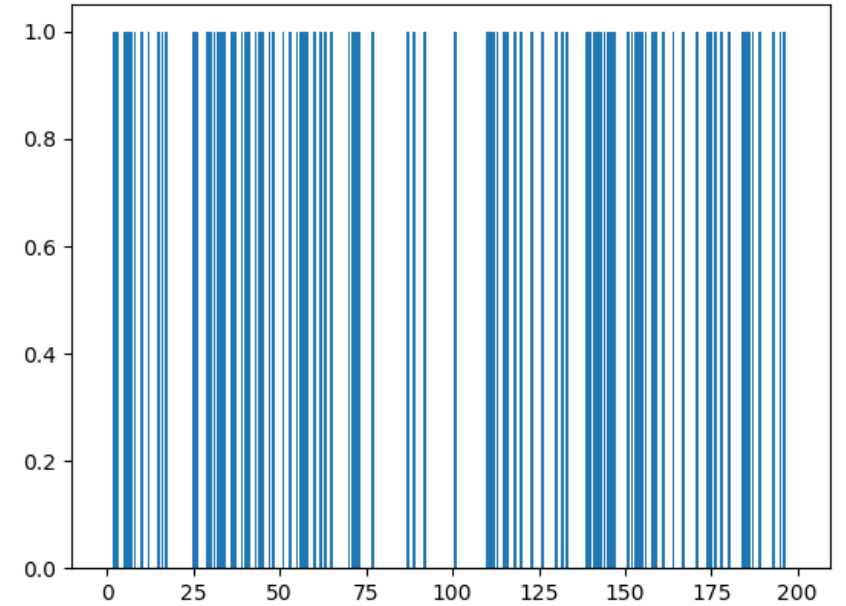
include = include_trace.mean(axis=0)
coef = coef_trace.mean(axis=0)

yh = np.dot(include * training_data, coef)
yh = yh - yh.mean()
print(metrics.roc_auc_score(training_labels, yh))
yh = 0.5 * (np.sign(yh) + 1)
print((yh == training_labels).mean())

yt = np.dot(include * testing_data, coef)
yt = yt - yt.mean()
print(metrics.roc_auc_score(testing_labels, yt))
yt = 0.5 * (np.sign(yt) + 1)
print((yt == testing_labels).mean())
```

The Results

```
plt.bar(range(200), isel)  
plt.show()  
  
plt.bar(range(200), include)  
plt.show()
```

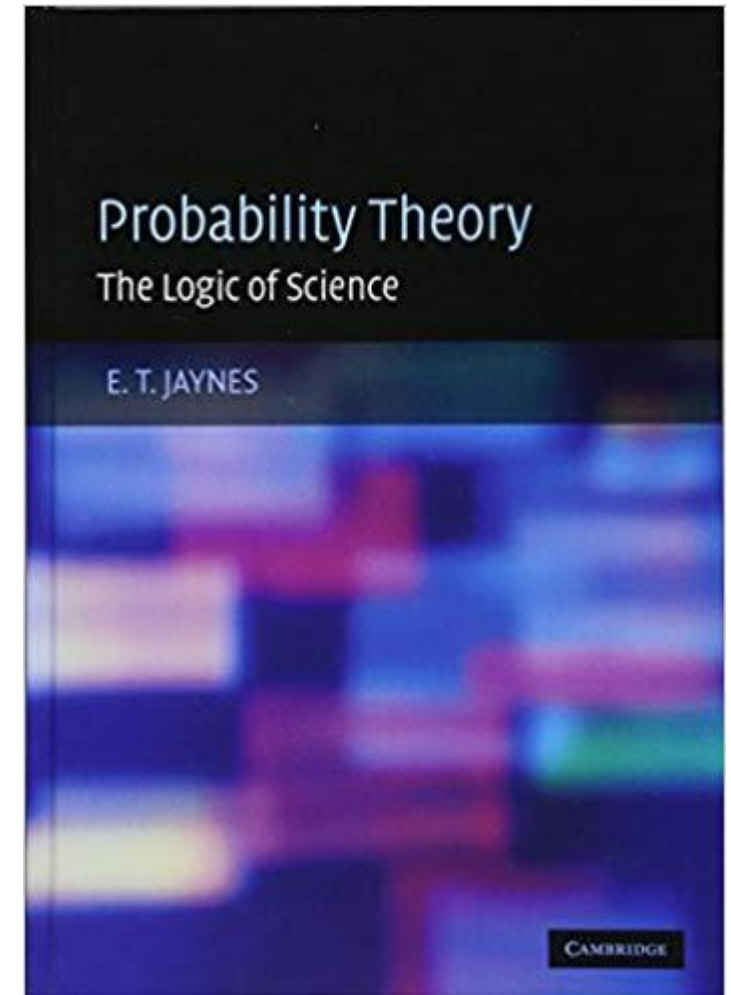


Winning the “Don’t Overfit!” Competition

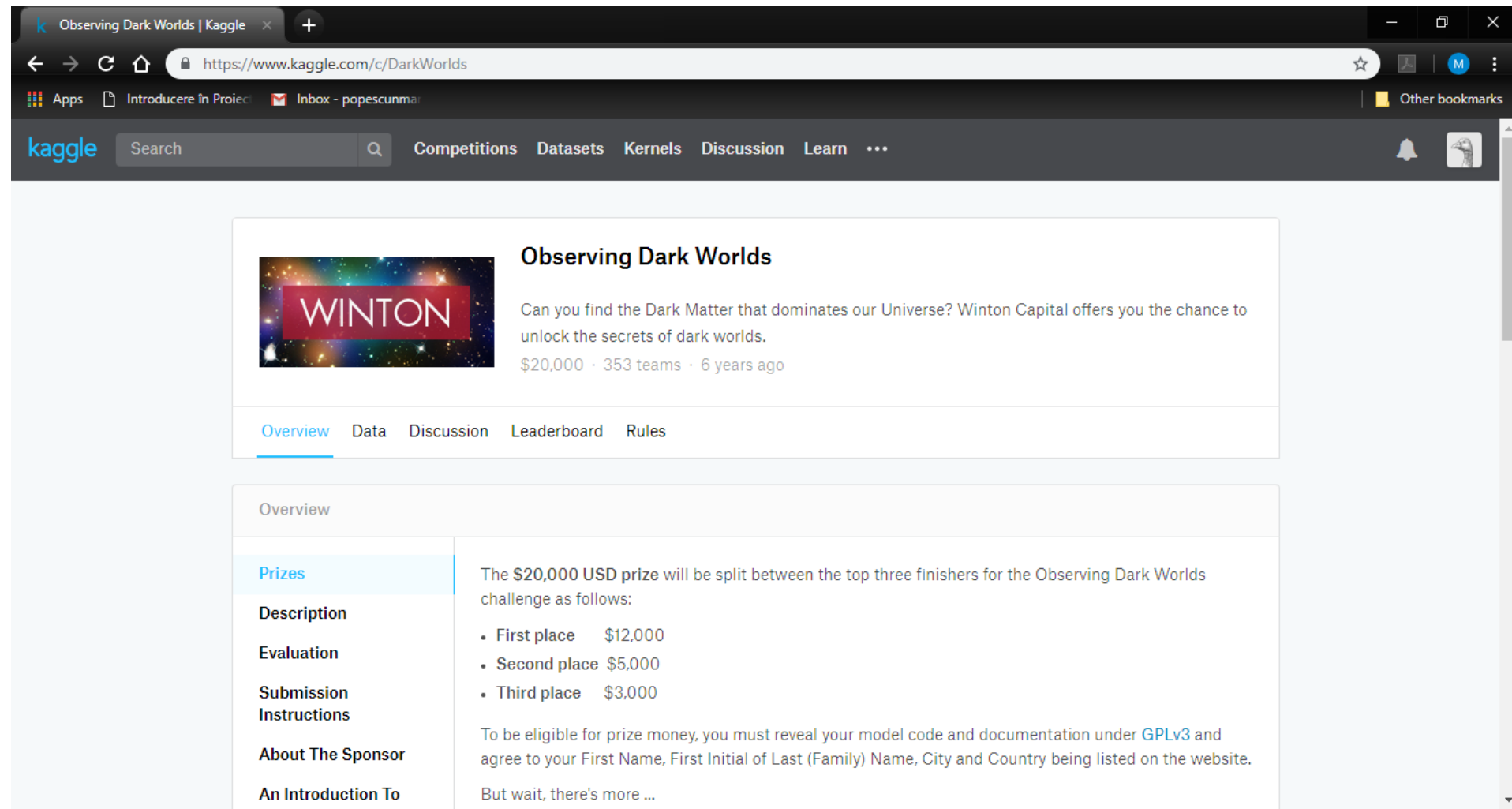
“

I had fun with this competition, and I would like to thank Phil Brierley for organizing it. If this post has coincidentally managed to convert any of you to the Bayesian religion 😊, I strongly recommend reading Jaynes’s “Probability Theory: The Logic of Science. (that’s how I first learned Bayesian analysis)

”



Between Oct. 12, 2012 and Dec. 17, 2012, Kaggle hosted a prediction competition entitled “Observing Dark Worlds”



The screenshot shows the Kaggle website interface for the 'Observing Dark Worlds' competition. The browser address bar shows the URL <https://www.kaggle.com/c/DarkWorlds>. The Kaggle logo and navigation links (Competitions, Datasets, Kernels, Discussion, Learn) are visible in the header. The competition title 'Observing Dark Worlds' is prominently displayed, along with a description: 'Can you find the Dark Matter that dominates our Universe? Winton Capital offers you the chance to unlock the secrets of dark worlds.' Below the title, the prize amount '\$20,000', the number of teams '353 teams', and the time '6 years ago' are listed. A tabbed interface shows 'Overview' as the selected tab. On the left side of the overview, there is a sidebar with links to 'Prizes', 'Description', 'Evaluation', 'Submission Instructions', 'About The Sponsor', and 'An Introduction To'. The main content area under 'Overview' details the prize structure: 'The \$20,000 USD prize will be split between the top three finishers for the Observing Dark Worlds challenge as follows:' followed by a list: 'First place \$12,000', 'Second place \$5,000', and 'Third place \$3,000'. It also mentions eligibility requirements: 'To be eligible for prize money, you must reveal your model code and documentation under [GPLv3](#) and agree to your First Name, First Initial of Last (Family) Name, City and Country being listed on the website.' The text 'But wait, there's more ...' is partially visible at the bottom.

Observing Dark Worlds | Kaggle

<https://www.kaggle.com/c/DarkWorlds>

kaggle Search Competitions Datasets Kernels Discussion Learn

Observing Dark Worlds

Can you find the Dark Matter that dominates our Universe? Winton Capital offers you the chance to unlock the secrets of dark worlds.

\$20,000 · 353 teams · 6 years ago

[Overview](#) Data Discussion Leaderboard Rules

Overview

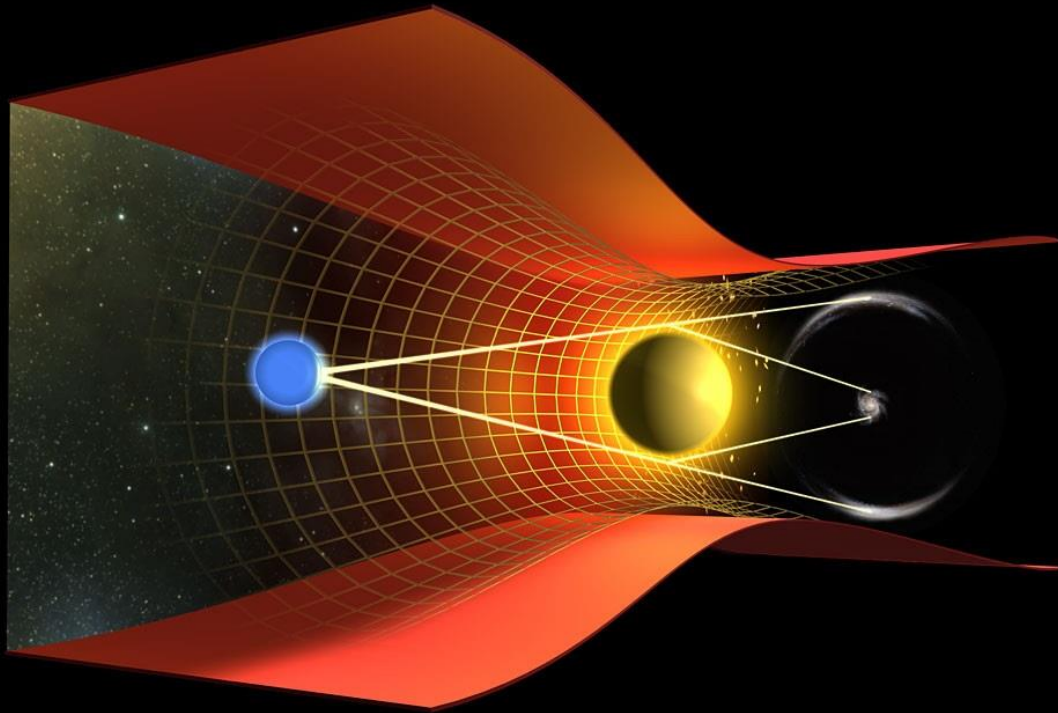
Prizes	The \$20,000 USD prize will be split between the top three finishers for the Observing Dark Worlds challenge as follows:
Description	<ul style="list-style-type: none">• First place \$12,000• Second place \$5,000• Third place \$3,000
Evaluation	
Submission Instructions	To be eligible for prize money, you must reveal your model code and documentation under GPLv3 and agree to your First Name, First Initial of Last (Family) Name, City and Country being listed on the website.
About The Sponsor	
An Introduction To	But wait, there's more ...

Observing Dark Worlds

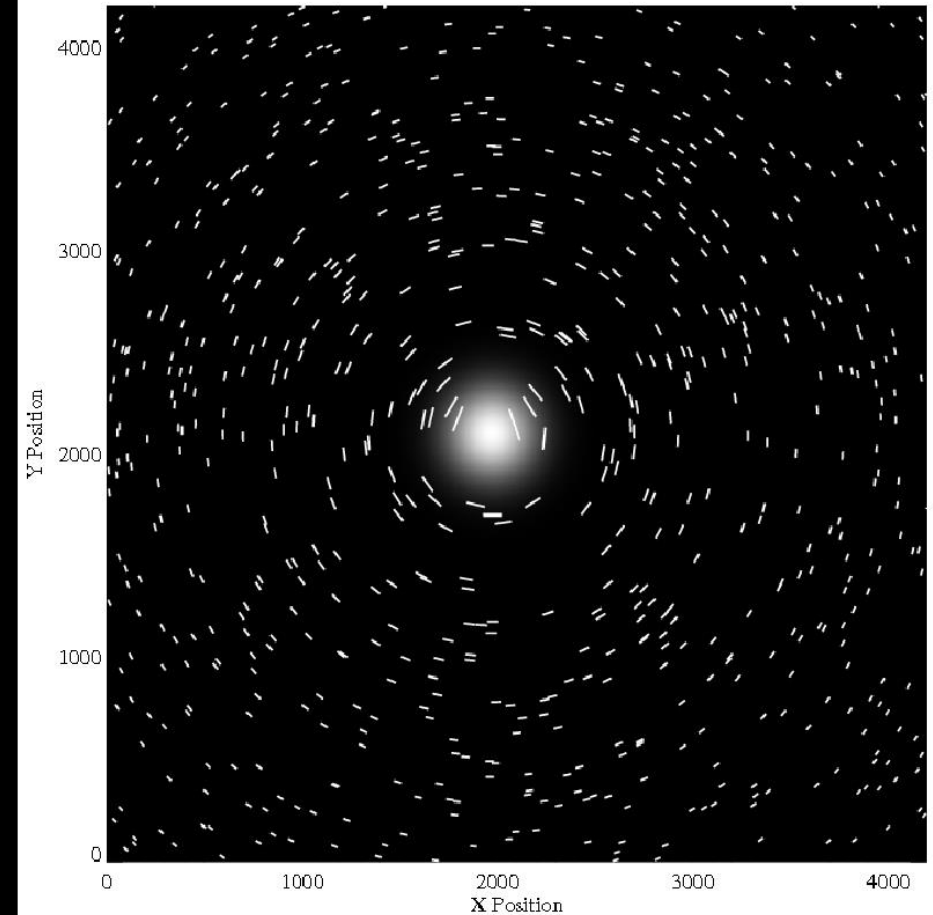
The goal of this competition was to detect clouds of dark matter floating around the universe through their effect on the light emitted by background galaxies.

“There is more to the Universe than meets the eye. Out in the cosmos exists a form of matter that outnumbers the stuff we can see by almost 7 to 1, and we don't know what it is. What we do know is that it does not emit or absorb light, so we call it Dark Matter. Such a vast amount of aggregated matter does not go unnoticed. In fact we observe that this stuff aggregates and forms massive structures called Dark Matter Halos. Although dark, it warps and bends spacetime such that any light from a background galaxy which passes close to the Dark Matter will have its path altered and changed. This bending causes the galaxy to appear as an ellipse in the sky.”

Use this “bending of light” to estimate where in the sky this dark matter is located.



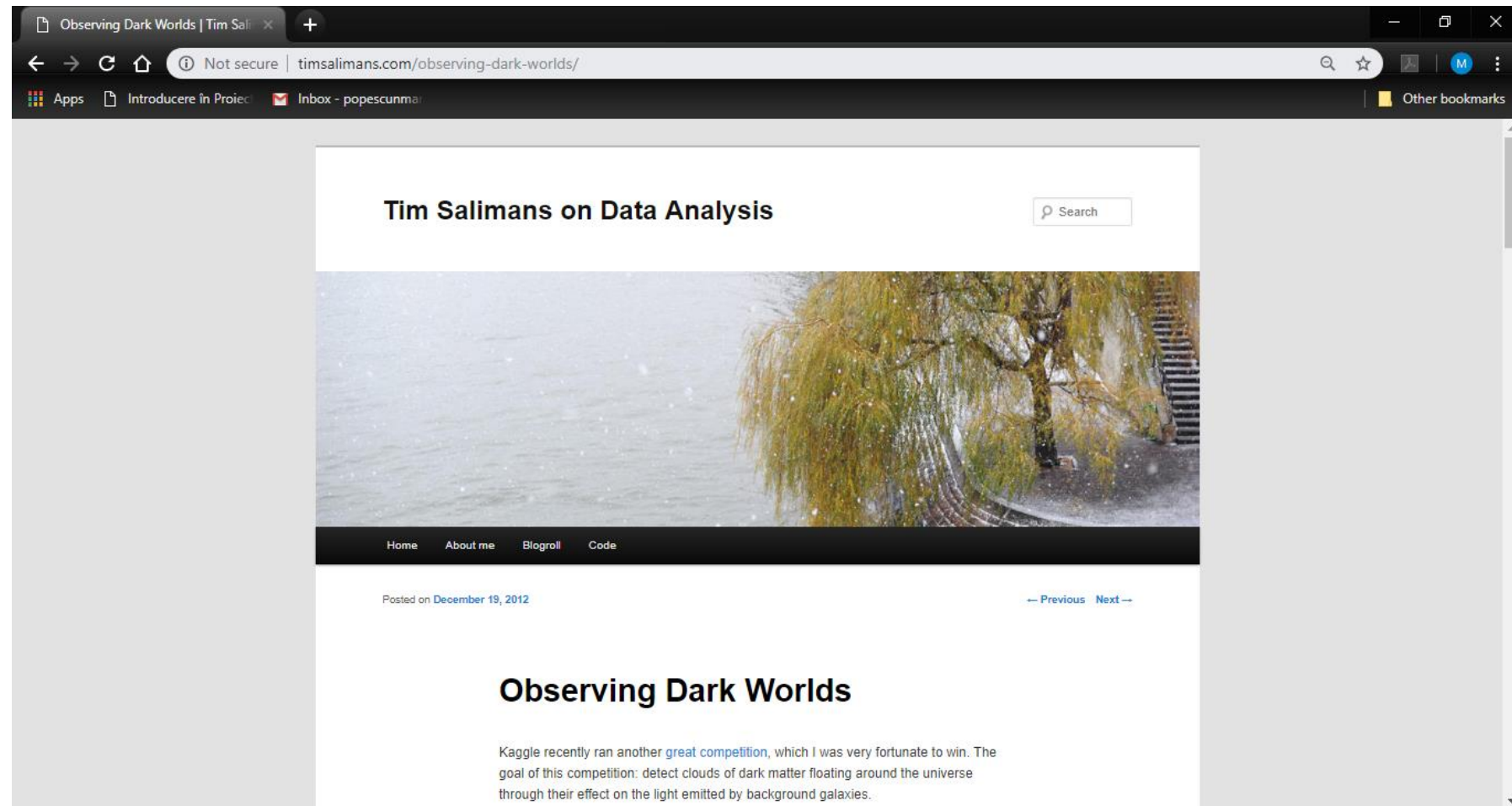
Credit: NASA, ESA, and Johan Richard (Caltech, USA)



Winning the “Observing Dark Worlds” Competition

“Although the description makes this sound like a physics problem, it is really one of statistics: given the noisy data (the elliptical galaxies) recover the model and parameters (position and mass of the dark matter) that generated them.

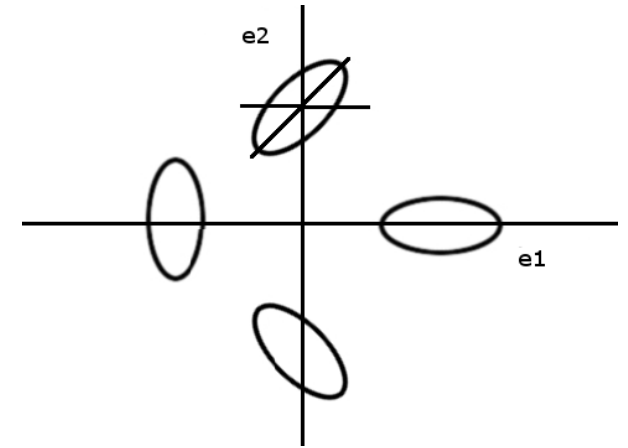
Bayesian analysis provided the winning recipe for solving this problem.”



The Data

The dataset is actually 300 separate files, each representing a sky. In each file, or sky, are between 300 and 720 galaxies. Each galaxy has an x and y position associated with it, ranging from 0 to 4200, and measures of ellipticity: e_1 and e_2 . e_1 describes how elongated the galaxy is in the x and y direction, while e_2 describes how elongated the galaxy is along a 45-degree angle. But for our purposes it does not matter besides for visualization purposes.

Each sky has one, two or three dark matter halos in it.



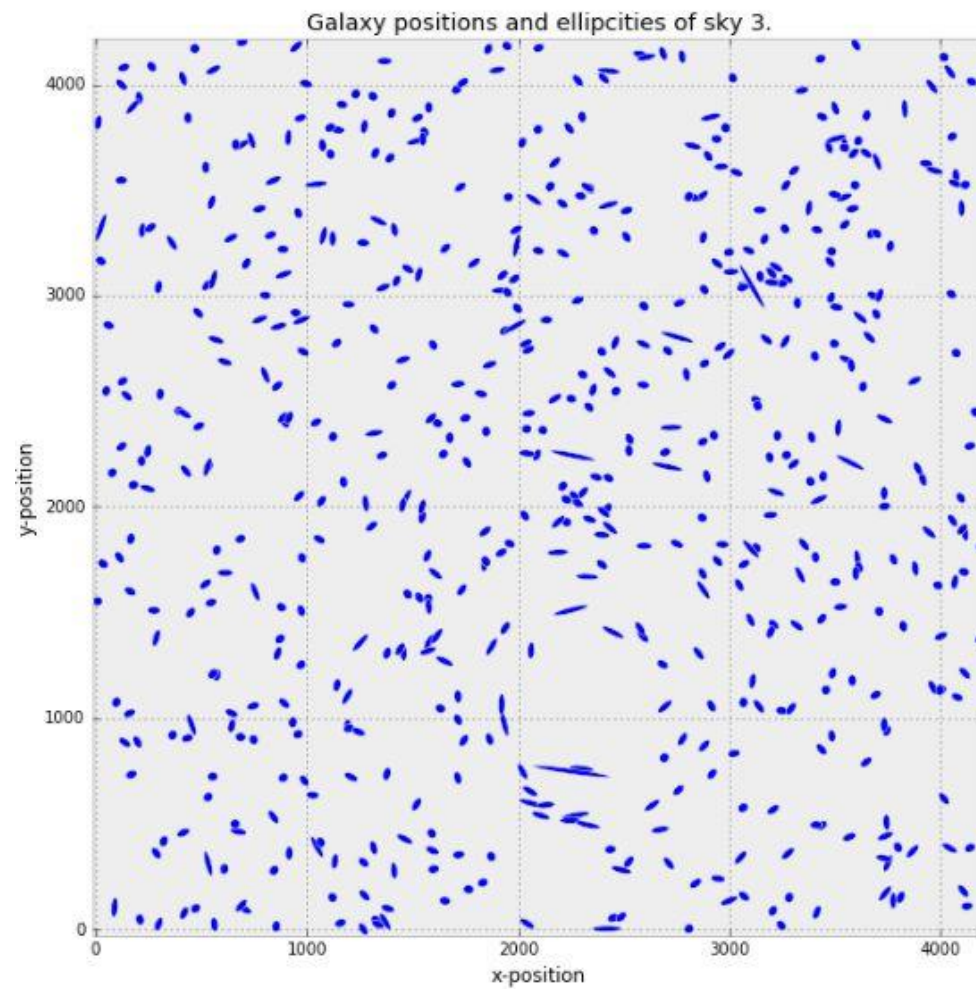
Training_Sky3.csv

```
GalaxyID,x,y,e1,e2
Galaxy1,162.69,1600.06,0.114664,-0.190326
Galaxy2,2272.28,540.04,0.623555,0.214979
Galaxy3,3553.64,2697.71,0.283527,-0.301870
```

Training_halos.csv

```
SkyId,numberHalos,x_ref,y_ref,halo_x1,halo_y1,halo_x2,halo_y2,halo_x3,halo_y3
Sky1,1,1086.80,1114.61,1086.80,1114.61,0.00,0.00,0.00,0.00
Sky2,1,3477.71,1907.33,3477.71,1907.33,0.00,0.00,0.00,0.00
Sky3,1,2315.78,1081.95,2315.78,1081.95,0.00,0.00,0.00,0.00
```


The Data



The Loss Function

The loss function in this problem is very complicated. For the very determined, the loss function is contained in the file `DarkWorldsMetric.py` (provided by the organizers). Though I suggest not reading it all, suffice to say the loss function is about 160 lines of code — not something that can be written down in a single mathematical line. The loss function attempts to measure the accuracy of prediction, in a Euclidean distance sense, such that no shift-bias is present.

The Model

“

1. Construct a prior distribution for the halo positions $p(x)$, i.e. formulate our expectations about the halo positions before looking at the data
2. Construct a probabilistic model for the data (observed ellipticities of the galaxies) given the positions of the dark matter halos: $p(e|x)$
3. Use Bayes' rule to get the posterior distribution of the halo positions: $p(x|e) = \frac{p(e|x)p(x)}{p(e)}$, i.e. use the data to guess where the dark matter halos might be
4. Minimize the expected loss with respect to the posterior distribution over the predictions for the halo positions: $\hat{x} = \operatorname{argmin}_{\text{prediction}} E_{p(x|e)}[L(\text{prediction}, x)]$, i.e. tune our predictions to be as good as possible for the given error metric.

”

The Model

“

For step 1. I simply assumed that the dark matter halos were distributed uniformly at random across the sky. Step 2 is more complicated. After reading through the tutorials and forum posts it became clear that the following model should be reasonable:

$$p(e_i|x) = N\left(\sum_{j=\text{all halos}} d_{i,j} m_j f(r_{i,j}), \sigma^2\right)$$

where $d_{i,j}$ is the *tangential direction*, i.e. the direction in which halo j bends the light of galaxy i , m_j is the mass of halo j , and $f(r_{i,j})$ is a decreasing function in the Euclidean distance $r_{i,j}$ between galaxy i and halo j .

After looking at the data I fixed the variance of the Gaussian distribution σ^2 at 0.05.

”

The Model

“

Like most competitors I also noticed that all skies seemed to have a single large halo, and that the other halos were much smaller. For the large halo I assigned the halo mass m a log-uniform distribution between 40 and 180, and I set $f(r_{i,j}) = \frac{1}{\max(r_{i,j}, 240)}$. For the small halos I fixed the mass at 20, and I used $f(r_{i,j}) = \frac{1}{\max(r_{i,j}, 70)}$.

The resulting model is likely to be overly simplistic but it seems to capture most of the signal that is present in the data. In addition, keeping the model simple protected me against overfitting the data.

”

Observing Dark Worlds in PyMC

```
def euclidean_distance(x, y):  
    return np.sqrt(((x - y) ** 2).sum(axis=1))
```

```
def f_distance(gxy_pos, halo_pos, c):  
    # foo_position should be a 2-d numpy array  
    return np.maximum(euclidean_distance(gxy_pos, halo_pos), c)[: , None]
```

```
def tangential_distance(glxy_position, halo_position):  
    # foo_position should be a 2-d numpy array  
    delta = glxy_position - halo_position  
    t = (2 * np.arctan(delta[:, 1] / delta[:, 0]))[: , None]  
    return np.concatenate([-np.cos(t), -np.sin(t)], axis=1)
```

Observing Dark Worlds in PyMC

```
# set the size of the halo's mass
mass_large = pm.Uniform("mass_large", 40, 180, trace=False)

# set the initial prior position of the halos, it's a 2-d Uniform dist.
halo_position = pm.Uniform("halo_position", 0, 4200, size=(1, 2))

@pm.deterministic
def mean(mass=mass_large, h_pos=halo_position, glx_pos=data[:, :2]):
    return mass / f_distance(glx_pos, h_pos, 240) * \
        tangential_distance(glx_pos, h_pos)

ellpty = pm.Normal("ellipcity", mean, 1. / 0.05, observed=True,
                    value=data[:, 2:])

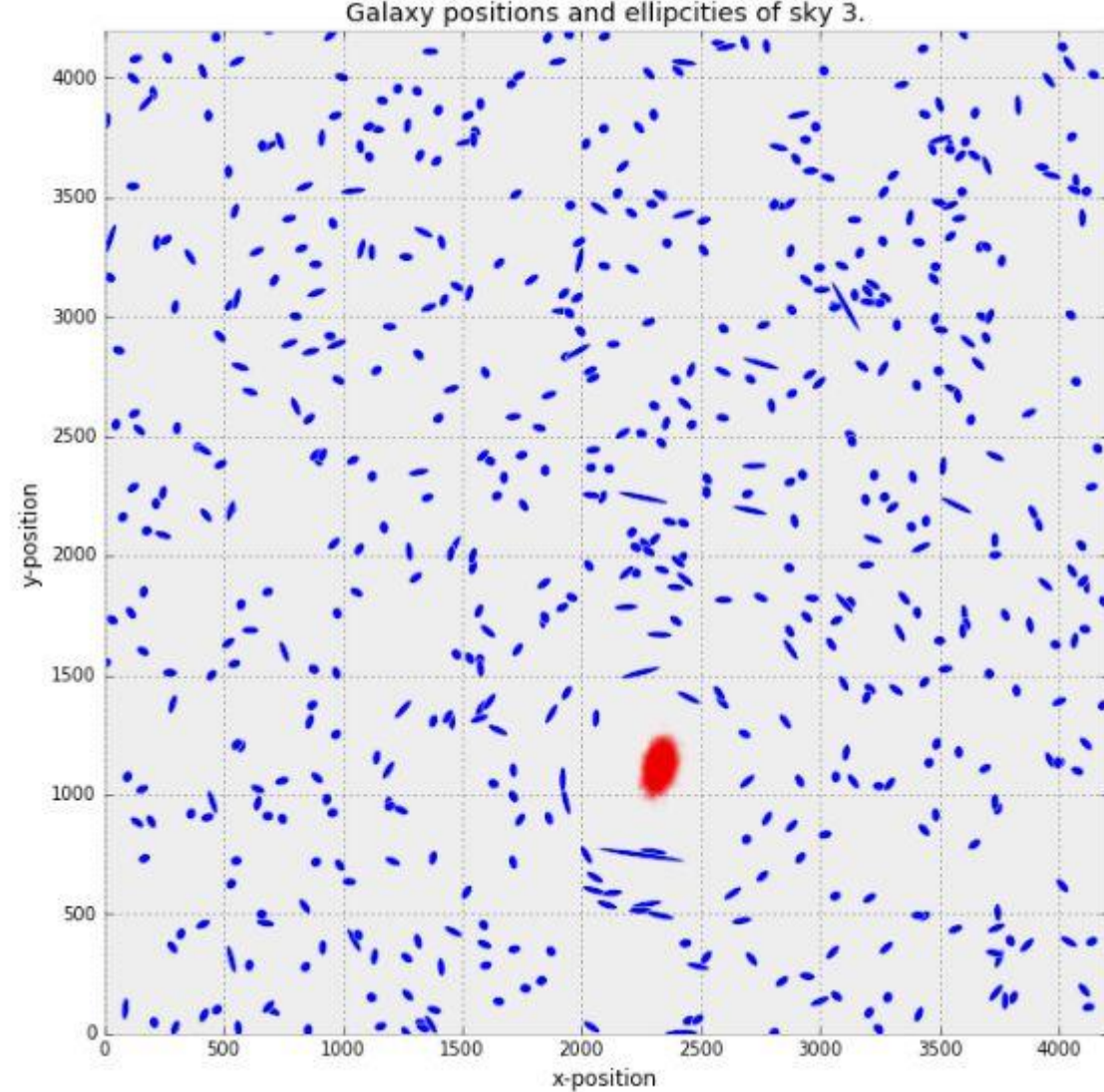
mcmc = pm.MCMC([ellpty, mean, halo_position, mass_large])
map_ = pm.MAP([ellpty, mean, halo_position, mass_large])
map_.fit()

mcmc.sample(200000, 140000, 3)
```

A Heatmap of the Posterior Distribution

Training_halos.csv

```
SkyId,numberHalos,x_ref,y_ref,halo_x1,halo_y1,halo_x2,halo_y2,halo_x3,halo_y3
Sky1,1,1086.80,1114.61,1086.80,1114.61,0.00,0.00,0.00,0.00
Sky2,1,3477.71,1907.33,3477.71,1907.33,0.00,0.00,0.00,0.00
Sky3,1,2315.78,1081.95,2315.78,1081.95,0.00,0.00,0.00,0.00
```



A Heatmap of the Posterior Distribution

