Introduction to probabilistic programming (Introducing our new friend Edward)

Adam Richards

galvanize

01.04.2018

What are we going to cover?

- Probabilistic programming intro
- Box's Loop (Model $\rightarrow$ infer $\rightarrow$ criticize)
- Bayesian inference and some related tools
- Examples in both PyMC3 and Edward (model)
- Discuss the tools used for inference and criticism
- Switchpoint and multilevel modeling examples

Probabilistic programming

A probabilistic programming language makes it easy to:

1. Write out complex probability models
2. And subsequently solve these models automatically.

Generally this is accomplished by:

1. Random variables are handled as a primitive
2. Inference is handled behind the scenes
3. Memory and processor management is abstracted away

The pros and the cons

Why you might want to use probabilistic programming

1. Customization - We can create models that have built-in hypothesis tests
2. Propagation of uncertainty - There is a degree of belief associated prediction and estimation
3. Intuition - The models are essentially 'white-box' which provides insight into our data

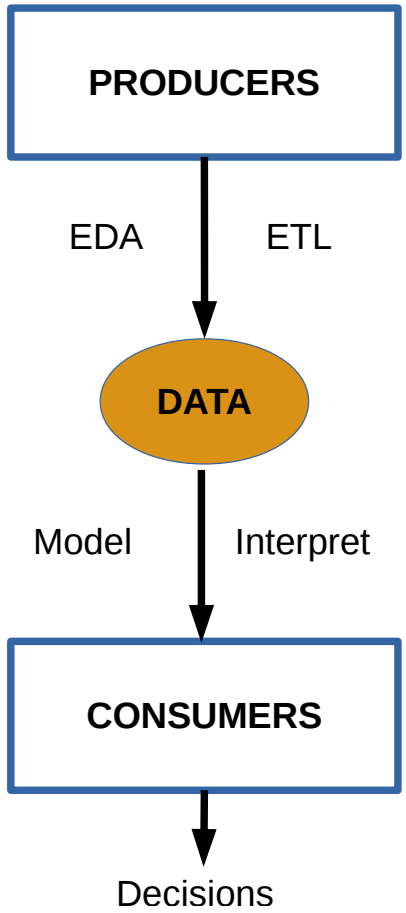
Why you might NOT want use out probabilistic programming

1. Deep dive - Many of the online examples will assume a fairly deep understanding of statistics
2. Overhead - Computational overhead might make it difficult to be production ready
3. Sometimes simple is enough - The ability to customize models in almost a plug-n-play manner has to come with some cost.

Doing data science

The taxonomy of what a data scientist does

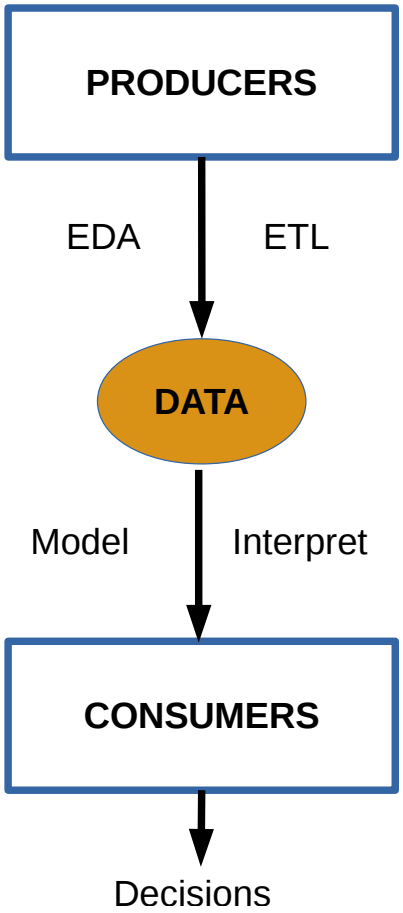
1. Obtaining data
2. Scrubbing data
3. Exploring data
4. Modeling data
5. iNterpreting data



OSEMN is pronounced as Awesome.

See Hilary Mason and Chris Wiggins blog post
http://www.dataists.com/2010/09/a-taxonomy-of-data-science

probabilistic programming and data science



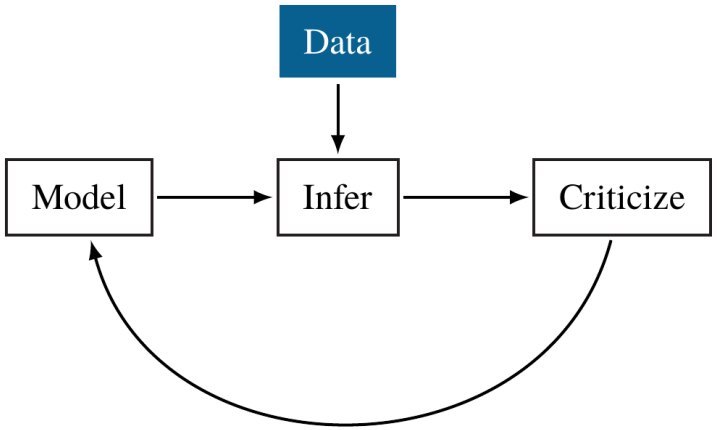


Figure 3: Box's loop.

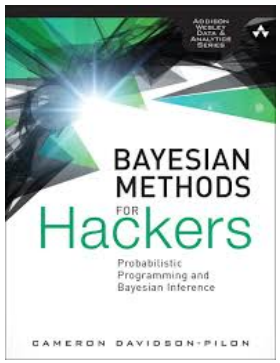
(Tran et al., 2016; Blei, 2014)

Bayesian Inference

Degree of belief

You are a skilled programmer, but bugs still slip into your code. After a particularly difficult implementation of an algorithm, you decide to test your code on a trivial example. It passes. You test the code on a harder problem. It passes once again. And it passes the next, *even more difficult*, test too! You are starting to believe that there may be no bugs in this code...

Bayesian methods for hackers



Some terminology

Probabilistic model

A joint distribution $p(X, \theta)$ of data X and latent variables θ.

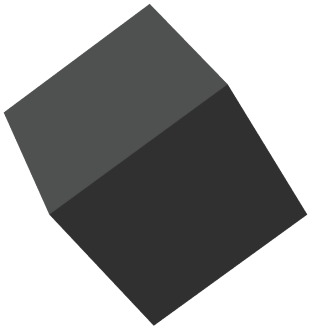
$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{1}$$

- Prior - $P(\theta)$ - one's beliefs about a quantity before presented with evidence
- Posterior - $P(\theta|x)$ - probability of the parameters given the evidence
- Likelihood - $P(x|\theta)$ - probability of the evidence given the parameters
- Normalizing constant - $P(x)$

- $P(\theta)$: This big, complex code likely has a bug in it.
- $P(\theta|X)$: The code passed all X tests; there still might be a bug, but it is less likely now.

Related libraries and frameworks

Libraries with high level of abstraction...



Edward

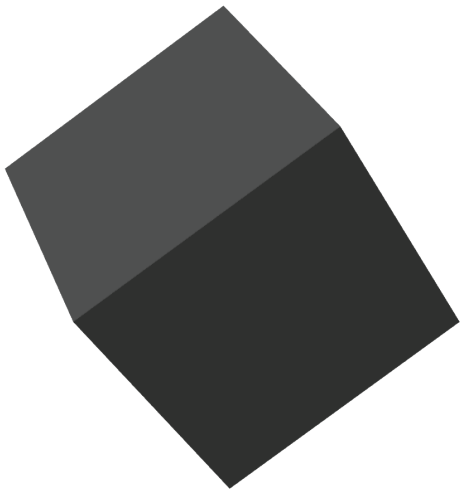
Keras

Computational framework libraries...

TensorFlow

theano

The three main probabilistic programming libraries



Edward

Stan

PyMC3

```
import pymc3 as pm
```

- Developed by John Salvatier, Thomas Wiecki, and Christopher Fonnesbeck (Salvatier et al., 2016)
- Comes with loads of good examples
- API is is not backwards compartible with models specified in PyMC2
- Can still be run in Python2.7+.

Basic workflow

1. Define hyperpriors
2. Open a model context
3. Perform inference

Edward

Edward is named after the statistician George Edward Pelham Box. Its design is based on his philosophy of statistics

First gather data from some real-world phenomena.
Then cycle through Box's loop

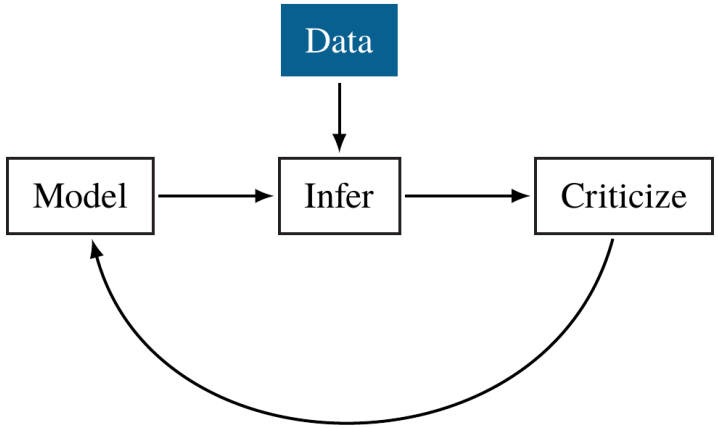


Figure 3: Box's loop.

(Tran et al., 2016)

Generalized procedure for probabilistic programming

Model,Inference,Criticism

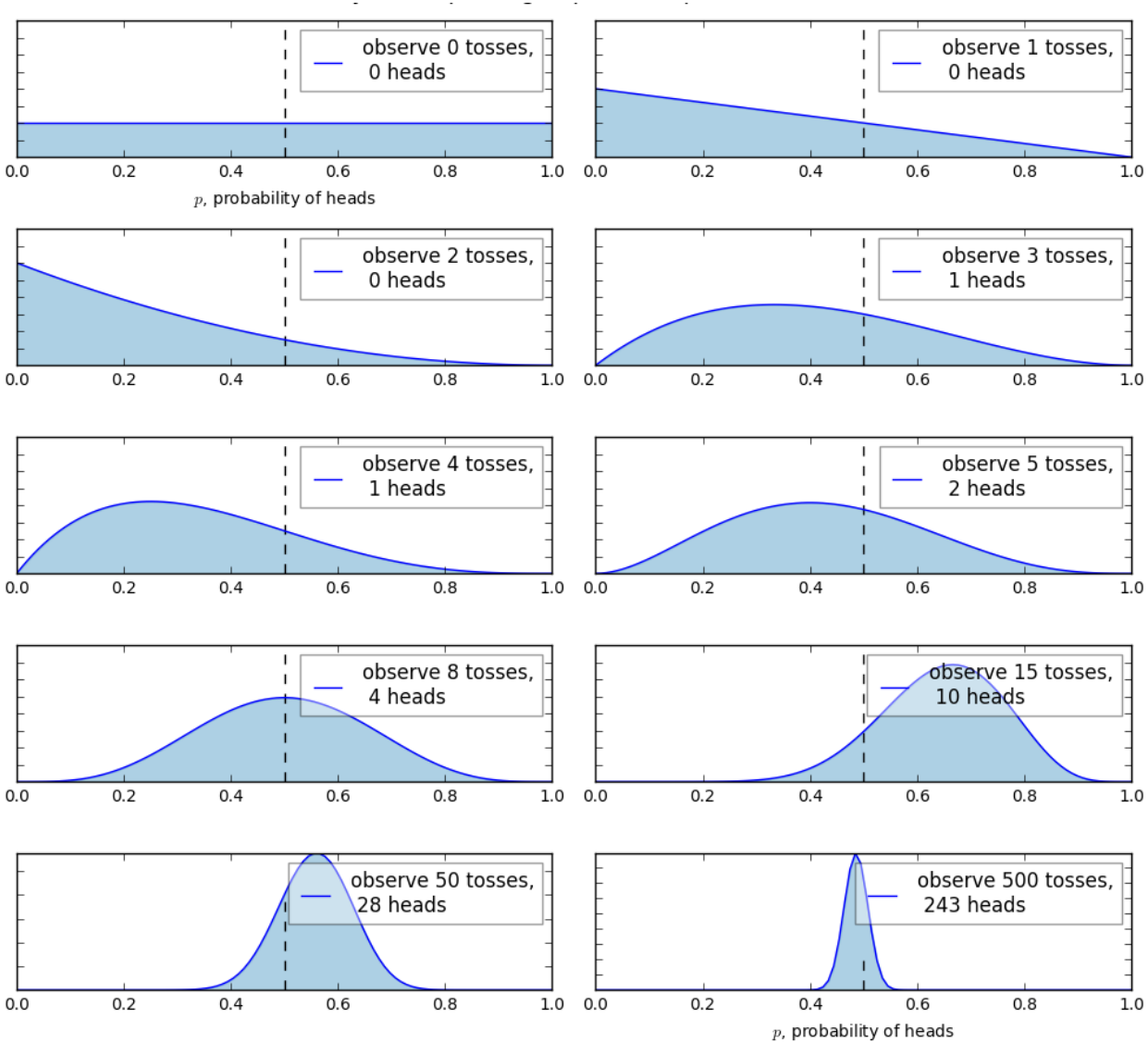
1. Build a probabilistic model of the phenomena
2. Reason about the phenomena given model and data
3. Criticize the model, revise and repeat

A child flips a coin ten times...

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

1. She is interested in the probability that the coin lands on heads
2. Given she only see heads and tails she reason that a binomial distribution might be appropriate
3. She finally analyzes whether her model captures the real-world phenomenon of coin flips (she may revise the model and repeat)

(Tran et al., 2016)


observe 0 tosses,
0 heads
observe 1 tosses,
0 heads
observe 2 tosses,
0 heads
observe 3 tosses,
1 heads
observe 4 tosses,
1 heads
observe 5 tosses,
2 heads
observe 8 tosses,
4 heads
observe 15 tosses,
10 heads
observe 50 tosses,
28 heads
observe 500 tosses,
243 heads
0.0
0.2
0.4
0.6
0.8
1.0
p, probability of heads

Coin flip in PyMC3

```
import pymc3 as pm

n,h,alpha,beta,niter = 100,61,2,2,1000

# context management
with pm.Model() as model:
    p = pm.Beta('p', alpha=alpha, beta=beta)
    y = pm.Binomial('y', n=n, p=p, observed=h)

    start = pm.find_MAP()
    step = pm.Metropolis()
    trace = pm.sample(niter, step, start)
```

Data → Model context → Priors → Likelihood → Sampler → Inference

To the notebooks!

Bayesian Linear Regression

For a set of N data points $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_n, y_n)\}$, the model can be specified as follows:

$$p(\mathbf{w}) = Normal(\mathbf{w}|\mathbf{0}, \sigma_w^2 \mathbf{I}) \tag{2}$$

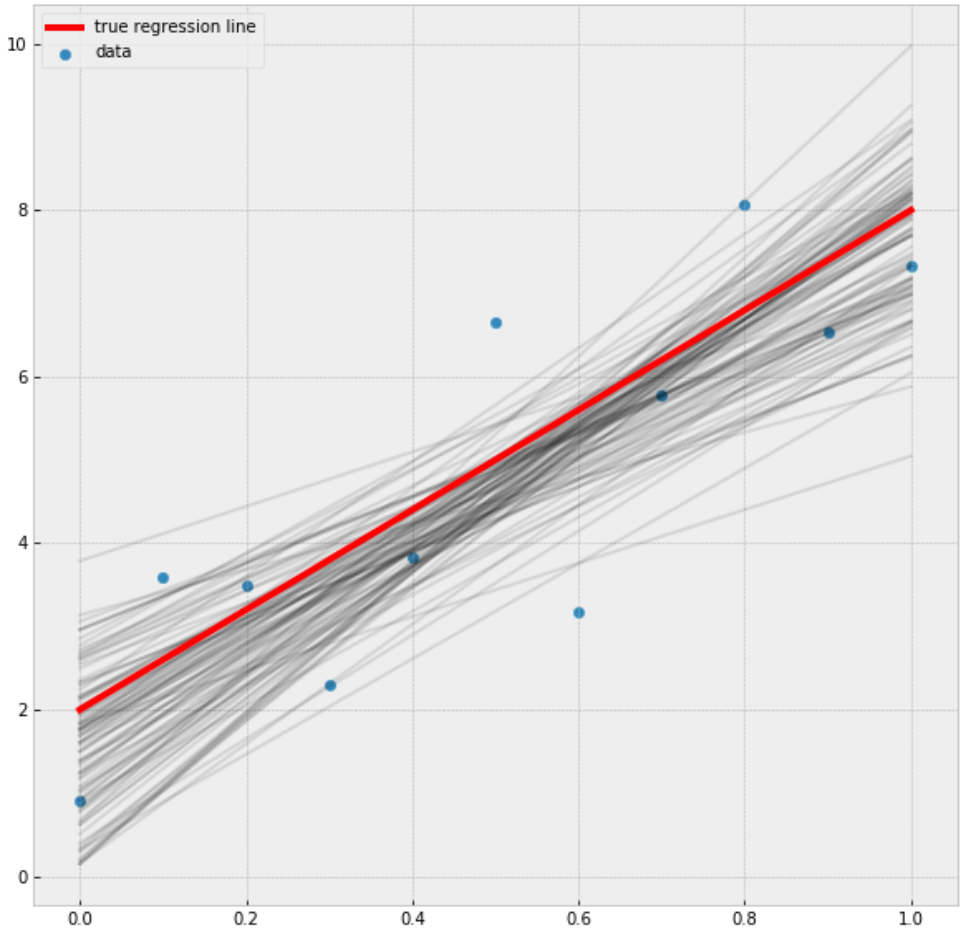
$$p(b) = Normal(b|0, \sigma_b^2) \tag{3}$$

$$p(\mathbf{y}|\mathbf{w}, b, \mathbf{X}) = \prod_{n=1}^{N} Normal(y_n|\mathbf{x}_n^T \mathbf{w} + b, \sigma_y^2) \tag{4}$$

- The latent variables are the linear model's weights **w** and intercept **b**
- We assume the prior and likelihood variances are known: σ_w^2, σ_b^2 and σ_y^2
- The mean of the likelihood is given by a linear transformation of the inputs in $\mathbf{x}_n$

(Murphy, 2012)

Bayes linear regression in PyMC3



Markov chain Monte Carlo (MCMC)

MCMC

- It is a family of algorithms for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult.
- The sequence can then be used to approximate the distribution
- It allows for inference on complex models

A particularly useful class of MCMC, known as Hamliltonian Monte Carlo, requires gradient information which is often not readily available so PyMC3 uses Theano to get around this problem. Something that has recently made this whole field a lot more interesting is the No-U-turn sampler (NUTS) because there are self-tuning strategies (Hoffman and Gelman, 2014).

One of the really nice things about probabilistic programming is that you do not have to know how inference is performed, but it can be useful.

- MCMC for Dummies
- More on Hamliltonian MCMC (Not for dummies)
- How to animate MCMC (for everyone)

There are a lot of tools for inference available

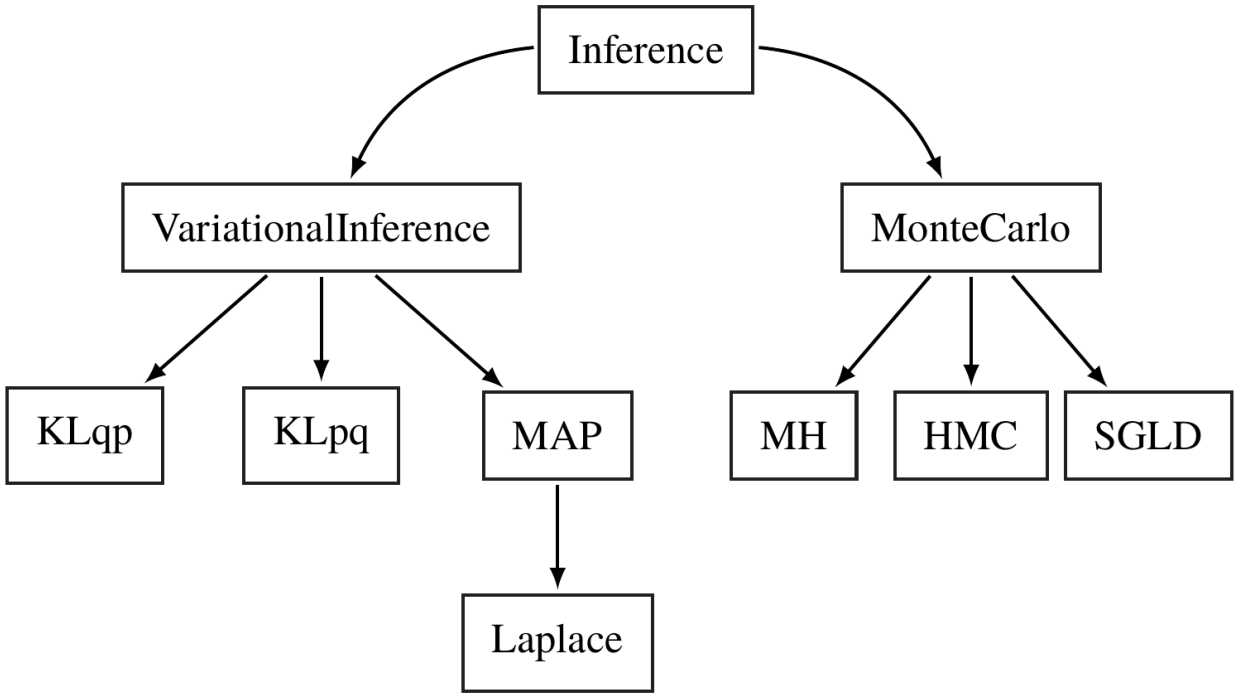


Figure 9: Dependency graph of inference methods. Nodes are classes in Edward and arrows represent class inheritance.

(Tran et al., 2016)

Bayes Factor

It is the Bayesian way to compare models. This is done by computing the marginal likelihood of each model

$$BF = \frac{p(y|M_0)}{p(y|M_1)} \tag{5}$$

Can be used to replace the p-value

Posterior predictive checks

Analyze the degree to which data generated from the model deviate from data generated from the true distribution. Can by used for hypothesis testing, model comparison, model selection, model averaging and to validate test data.

Links

- Bayes Factor in PyMC3
- Posterior predictive checks in PyMC3
- replacing p-values
- Bayesian estimation supersedes the t-test

We can never validate whether a model is true.

In practice, all models are wrong -George Box.

However, we can try to uncover where the model goes wrong. Model criticism helps justify the model as an approximation or point to good directions for revising the model.

Posterior predictive checks (PPCs) analyze the degree to which data generated from the model deviate from data generated from the true distribution. They can be used either numerically to quantify this degree, or graphically to visualize this degree.

RMSE and MSE

Model evaluation has become so much easier...

```
print("Mean squared error on test data:")
print(ed.evaluate('MSE', data={X: X_test, y_post: y_test}))
```

```
print("Mean absolute error on test data:")
print(ed.evaluate('MAE', data={X: X_test, y_post: y_test}))
```

Note: RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

Switchpoint analysis

- Probabilistic Programming and Bayesian Methods for Hackers
  Text Messages
- PyMC3 Documentation
  Coal Mining Disasters

Switchpoint analysis applications

- Did the change actually happen?
- Did change X on January 1 affect the number of sales we have seen in our company?
- Did imposing a new safety policy last summer reduce the number of accidents?
- After those staffing decisions how longs did it take before revenue was affected?
- Based on my time-series forecasting model when will a change actually occur?
- ...

If *p*-values are your tool of choice you could try Bayes Factors to have numeric values to make decisions from.

Multilevel modeling

Multilevel model. Multilevel models (also known as hierarchical linear models, nested data models, mixed models, random coefficient, random-effects models, random parameter models, or split-plot designs) are statistical models of parameters that vary at more than one level

- PyMC3 - Radon example GLM style
- Edward - Instructor evaluation ratings

https://en.wikipedia.org/wiki/Multilevel_model

A note on deep learning...

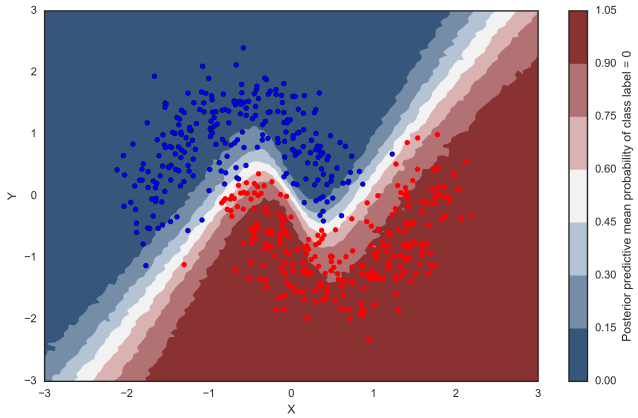
1. Neural Networks compute point estimates
2. They tend to be overly confident about class prediction
3. They are prone to overfitting
4. They have many parameters that may require tuning

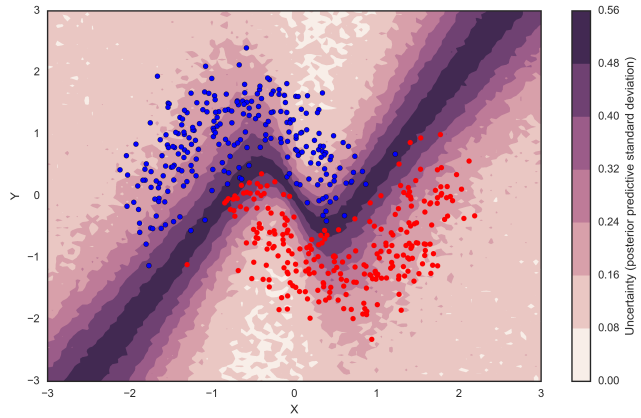
How confident am I in my prediction—can I quantify the uncertainty?

Bayesian neural network: Is a neural network with a prior distribution on the weights

https://www.youtube.com/watch?v=I09QVNrUS3Q (Tran et al., 2017)

Simple Neural Network using PyMC3





Another take on probabilistic programming

Another way of thinking about this: unlike a traditional program, which only runs in the forward directions, a probabilistic program is run in both the forward and backward direction. It runs forward to compute the consequences of the assumptions it contains about the world (i.e., the model space it represents), but it also runs backward from the data to constrain the possible explanations. In practice, many probabilistic programming systems will cleverly interleave these forward and backward operations to efficiently home in on the best explanations.

Why Probabilistic Programming matters? (Beau Cronin)

What did we cover again?

- ✓ Probabilistic programming intro
- ✓ Box's Loop (Model $\rightarrow$ infer $\rightarrow$ criticize)
- ✓ Bayesian inference and some related tools
- ✓ Examples in both PyMC3 and Edward (model)
- ✓ Discuss the tools used for inference and criticism
- ✓ Switchpoint and multilevel modeling examples

Where to go from here

Examples, examples, examples...

- Edward examples
- PyMC3 examples
- PyMC3 Getting started guide
- Bayesian methods for hackers
- Blog by Thomas Wiecki
- Doing Bayesian Data Analysis by John Kruschke
- Resource by Mark Dregan
- This is a nice intro to Bayesian thinking done on kdnuggets (using PyMC3)

There is also PyStan (Stan paper)

Galvanize

Upcoming events

- Essential Mathematical Foundations for Data Science (January 08-10)
- Intro to Tableau for Data Science (January 11th)
- Statistics short-course (January 22-24)
- Intro to Python - Part time Course (February-March)
- Data Science Immersive (February-May)

References I

Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable models. *Annual Review of Statistics and Its Application*.

Hoffman, M. D. and Gelman, A. (2014). The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55.

Tran, D., Hoffman, M. D., Saurous, R. A., Brevdo, E., Murphy, K., and Blei, D. M. (2017). Deep probabilistic programming. In *International Conference on Learning Representations*.

Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., and Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.