Introduction to probabilistic programming (with PyMC3)

A. Richards

02.08.2017

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.

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

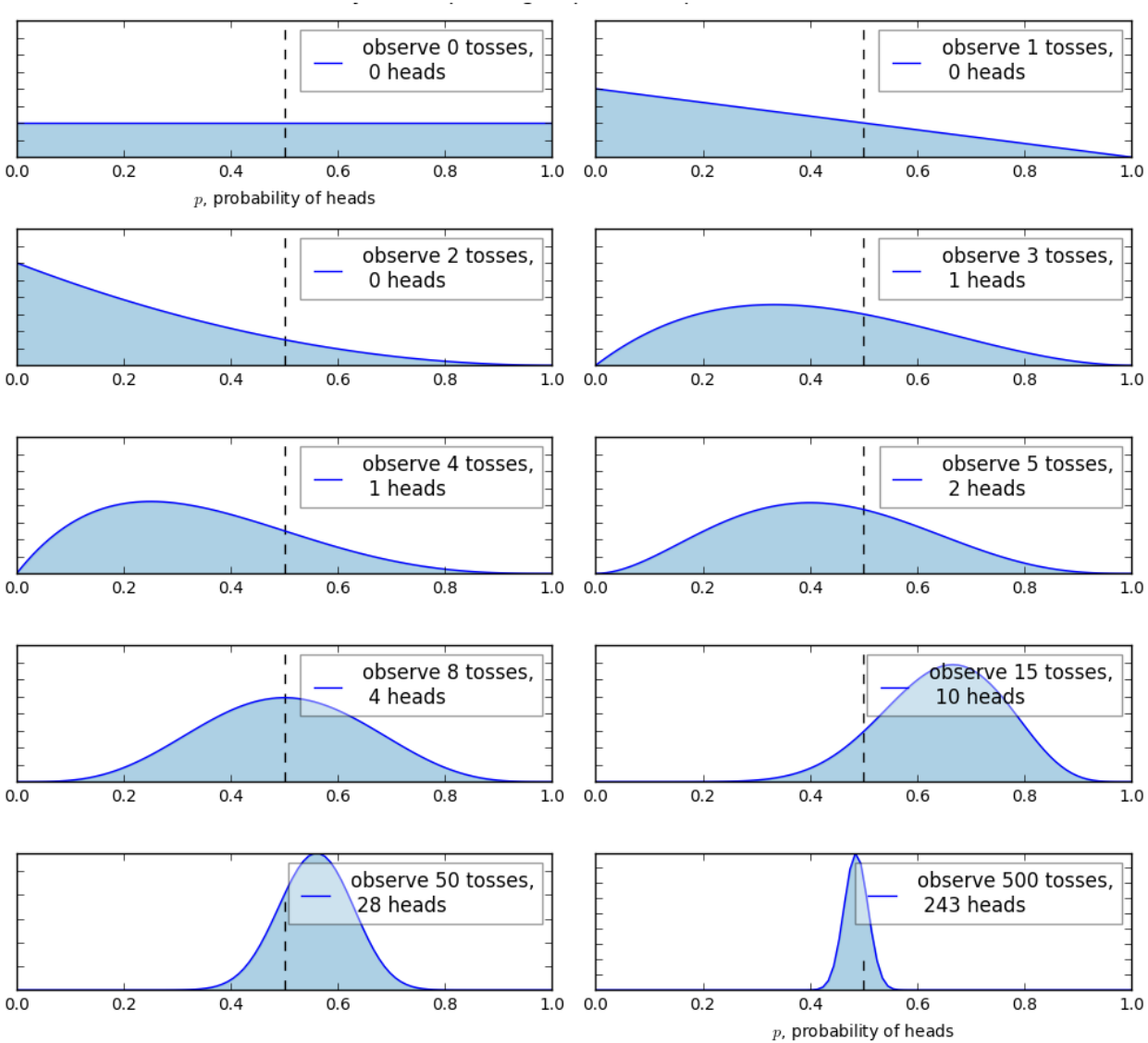
This is a nice intro to Bayesian thinking done on kdnuggets (using PyMC3)

Some terminology

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

- 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.


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

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

Markov chain Monte Carlo (MCMC)

MCMC

- It is an 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)

PyMC3 is an improvement over PyMC2

- Intuitive model specification syntax e.g.

$$x \sim N(0,1) \text{ becomes } x = \text{Normal}(0,1)$$

- Powerful sampling algorithms such as the No U-Turn Sampler
- Variational inference: ADVI for fast approximate posterior estimation as well as mini-batch ADVI for large data sets.
- Relies on Theano which provides:
  - Numpy broadcasting and advanced indexing
  - Linear algebra operators
  - Computation optimization and dynamic C compilation
  - Simple extensibility
- Transparent support for missing value imputation

Getting started

We will be using probabilistic programming with PyMC3 to perform automatic Bayesian inference on user-defined probabilistic models

These are some of the best getting started resources out there

- PyMC3 repo
- Getting started guide
- Bayesian methods for hackers

Now that we have an intuition for Bayesian inference and a general idea of what to expect with PyMC3 lets dive in.

Bayes Factor

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

$$BF = \frac{p(y|M_0)}{p(y|M_1)} \tag{2}$$

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

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!

Where are we...

- ✓ Overview
- ✓ Coin-flip example
- ✓ Estimating the mean and standard deviation of a Normal
- ✓ Switchpoint analysis of text messages

There are more examples..

But what about linear regression–the classical example??

The linear regression example is well explained in the docs, but I did want to point out that we now have access to GLM style formulations in PyMC3.

```
import pymc3 as pm

...

data = dict(x=x, y=y)

with pm.Model() as model:
    pm.glm.glm('y ~ x', data)
    step = pm.NUTS()
    trace = pm.sample(2000, step, progressbar=True)
```

Recommenders

What will a user *buy*, *click*, *like*...

	A	B	C	D	E	F	G	...
Frodo	1	?	2	1	1	4	1	...
Sam	?	?	1	3	?	3	1	...
Merry	?	?	1	1	?	1	1	...
Gimli	1	1	?	1	?	1	?	...
Legolas	1	1	?	1	?	1	?	...

- Ebay and Amazon who recommends items for purchase
- Movies, dating services, social network feeds, recipes, jokes, hikes, ...

There are numerous types of recommender systems

- Popularity - Most viewed, most well-liked, not customized to users
- User-User - Recommend items liked by users with similar results
- Item-Item - Recommend items similar to what the current user rated favorably
- Matrix-Factorization - Estimate a users underlying preferences

---

Two of the most commonly implemented varieties are:

- collaborative filtering - Imagine that user 1 and user 2 both like the same 4 science fiction novels we can then infer that if user 1 also likes another similar novel then user 2 will have a good chance of also liking it.
- Content-based recommender - These systems also make use of extra features associated with the user

Probabilistic matrix factorization (PMF)

- Probabilistic approach to the collaborative filtering problem that takes a Bayesian perspective (Salakhutdinov and Mnih, 2008).
- The ratings R are modeled as draws from a Gaussian distribution
- We use precision α, a fixed parameter, that reflects the uncertainty of the estimations; the normal distribution is commonly reparameterized in terms of precision, which is the inverse of the variance.
- small precision parameters help control the growth of our latent parameters

The following implementation is modified from the example in the PyMC3 documentation. Back to the notebook!

Conclusions

- Our results demonstrate that the mean of means method is our best baseline on our prediction task.
- We are able to obtain a significant decrease in RMSE using the PMF MAP estimate obtained via Powell optimization.
- We illustrated one way to monitor convergence of an MCMC sampler with a high-dimensionality sampling space
- The traceplots using this method seem to indicate that our sampler converged to the posterior.
- Results using this posterior showed that attempting to improve the MAP estimation using MCMC sampling actually overfit the training data and increased test RMSE. This was likely caused by the constraining of the posterior via fixed precision parameters α, αU and αV.

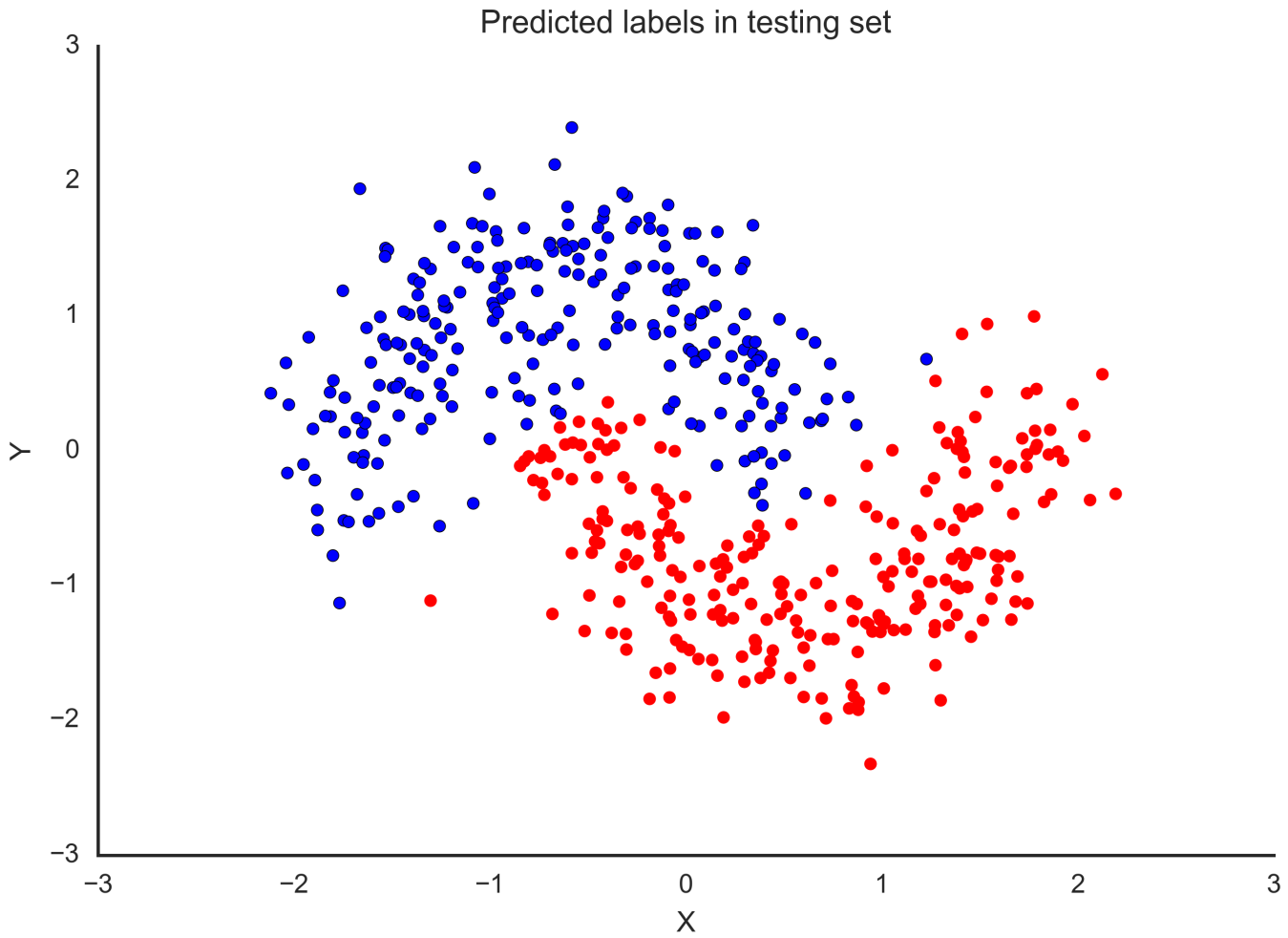
this gist is a working version of a fully Bayesian implementation.

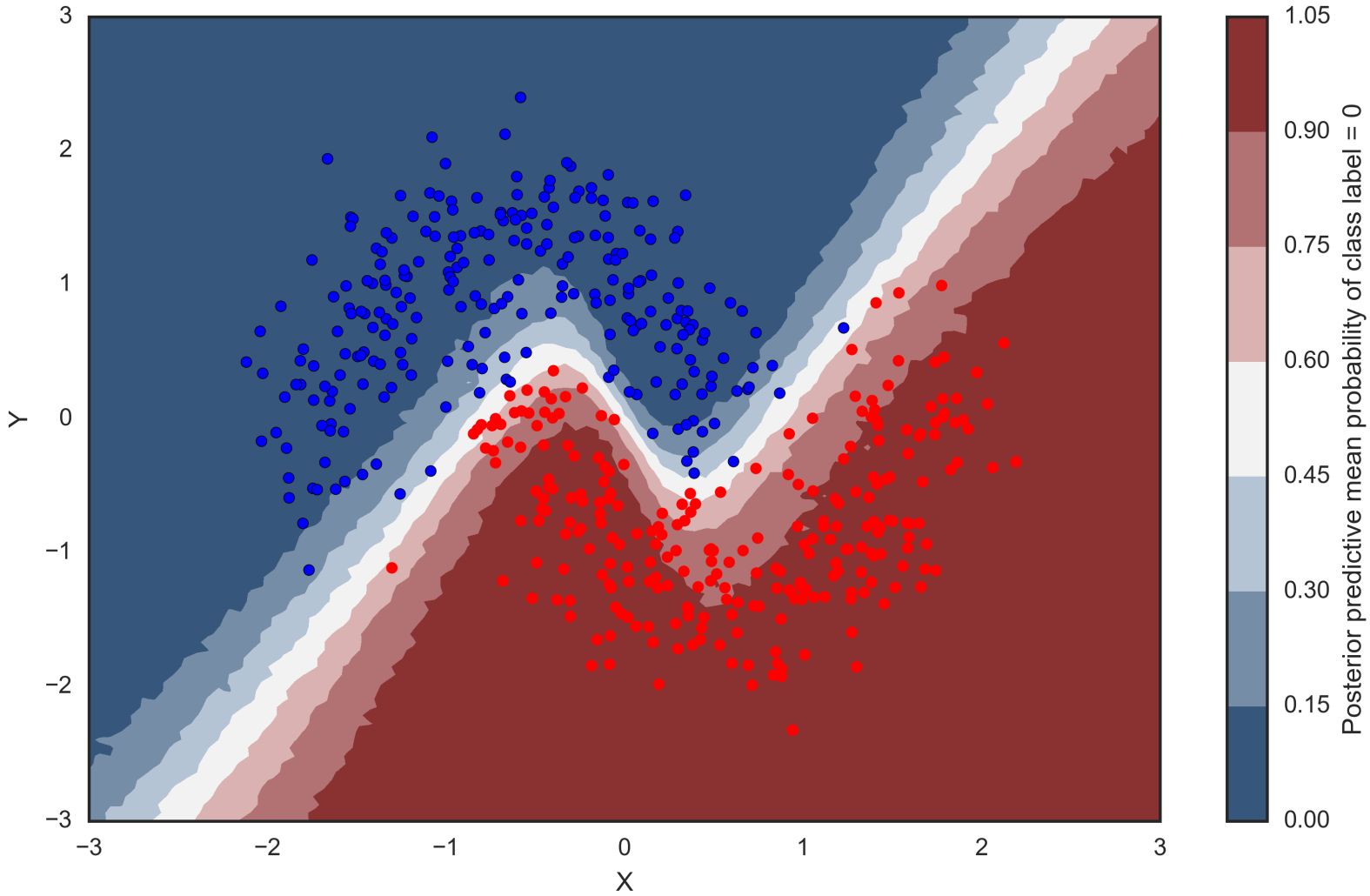
Neural Nets

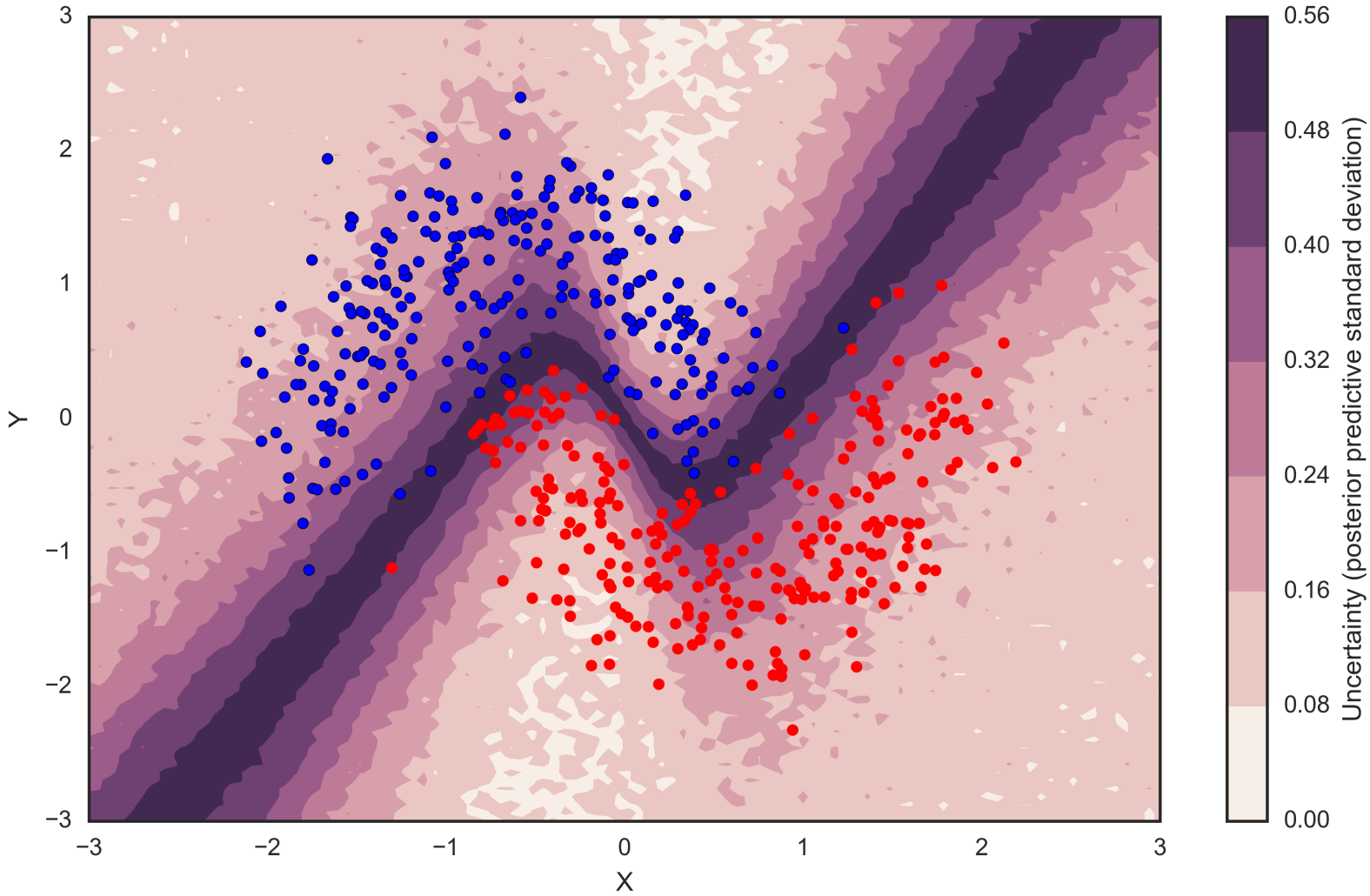
Thomas Wiecki

Has a great blog post talking about exactly how to do this

The notebook that is provided in this repo has not been significantly modified from its original form. Although, check the original repository for the latest version.


Predicted labels in testing set
3
2
1
0
−1
−2
−3
Y
−3
−2
−1
0
1
2
3
X


3
2
1
0
−1
−2
−3
Y
−3
−2
−1
0
1
2
3
X
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00
Posterior predictive mean probability of class label = 0


3
2
1
0
−1
−2
−3
Y
−3
−2
−1
0
1
2
3
X
0.56
0.48
0.40
0.32
0.24
0.16
0.08
0.00
Uncertainty (posterior predictive standard deviation)

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?

- ✓ Overview
- ✓ Coin-flip example
- ✓ Estimating the mean and standard deviation of a Normal
- ✓ Switchpoint analysis of text messages
- ✓ Probabilistic matrix factorization
- ✓ Probabilistic neural networks

Where to go from here

Examples, examples, examples...

- PyMC3 repo
- Getting started guide
- Bayesian methods for hackers
- Blog by Thomas Wiecki
- Doing Bayesian Data Analysis by John Kruschke
- Resource by Mark Dregan

There is also PyStan (Stan paper) And Edward.

References I

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.

Salakhutdinov, R. and Mnih, A. (2008). Probabilistic matrix factorization. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA.

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