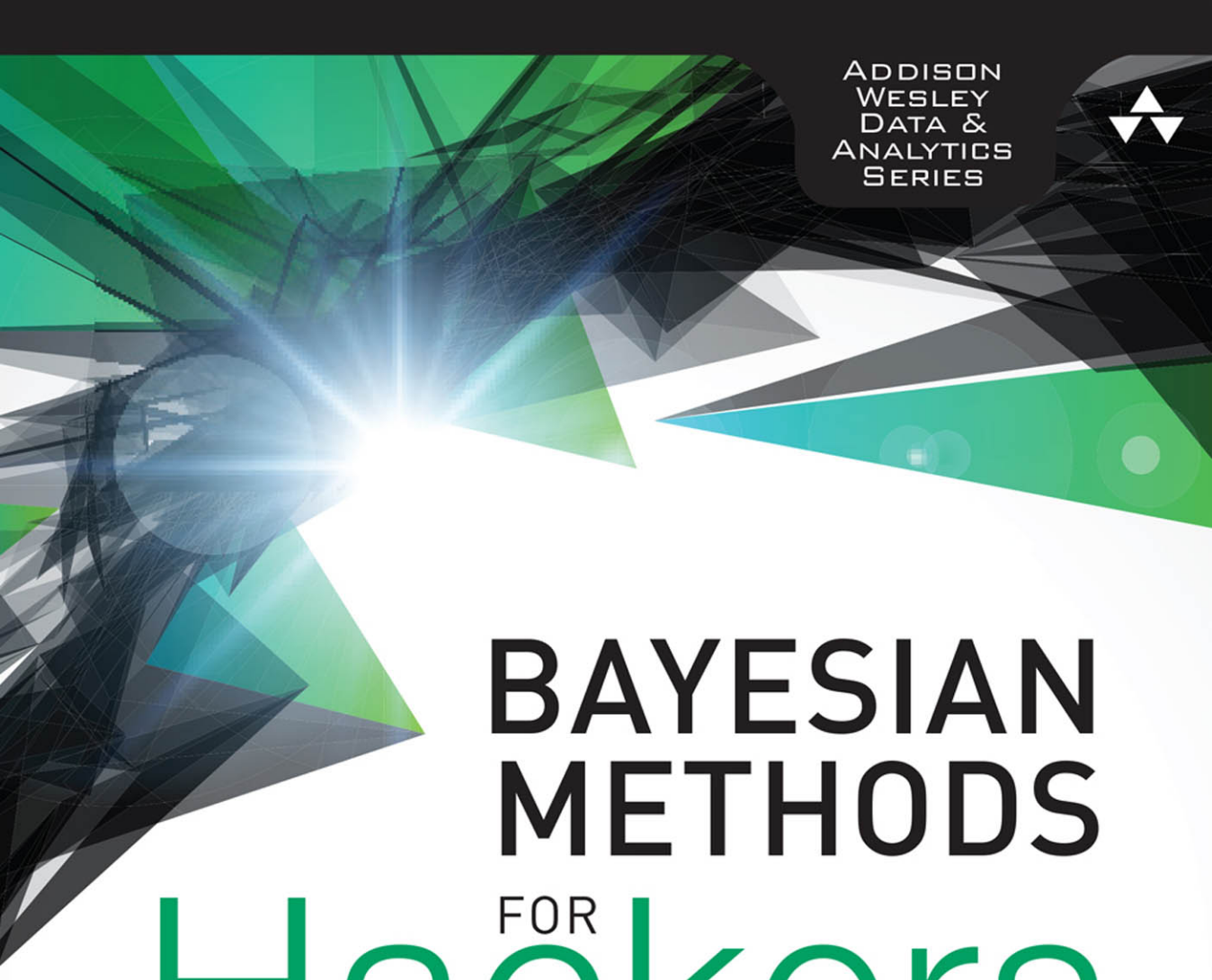

BAYESIAN METHODS FOR Hackers

Probabilistic Programming and Bayesian Inference

CAMERON DAVIDSON-PILON

Bayesian Methods for Hackers

```
x = np.linspace(-4, 4, 100)

plt.plot(x, logistic(x, 1), label=r"$\beta = 1$", ls="--", lw=1)
plt.plot(x, logistic(x, 3), label=r"$\beta = 3$", ls="--", lw=1)
plt.plot(x, logistic(x, -5), label=r"$\beta = -5$", ls="--", lw=1)

plt.plot(x, logistic(x, 1, 1), label=r"$\beta = 1, \alpha = 1$",
         color="#348ABD")
plt.plot(x, logistic(x, 3, -2), label=r"$\beta = 3, \alpha = -2$",
         color="#A60628")
plt.plot(x, logistic(x, -5, 7), label=r"$\beta = -5, \alpha = 7$",
         color="#7A68A6")

plt.title("Logistic function for different $\beta$ and\
           $\alpha$ values")
plt.xlabel("$x$")
plt.ylabel("Logistic function at $x$")
plt.legend(loc="lower left");
```

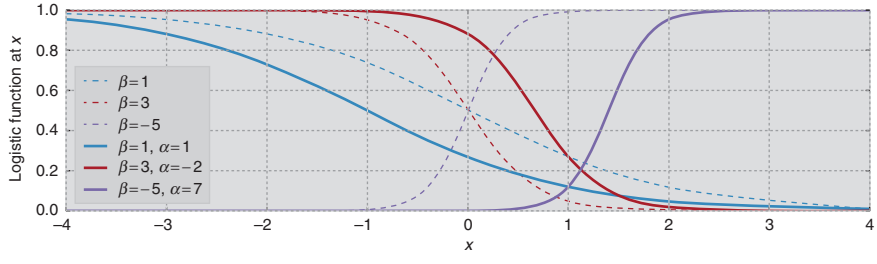

Figure 2.2.12: Logistic function for different β and α values

Adding a constant term α amounts to shifting the curve left or right (hence the name *bias*).

Let's start modeling this in PyMC. The β, α parameters have no reason to be positive, bounded, or relatively large, so they are best modeled by a *Normal random variable*, introduced next.

2.2.11 The Normal Distribution

A Normal random variable, denoted $X \sim N(\mu, 1/\tau)$, has a distribution with two parameters: the mean, μ, and the *precision*, τ. Those familiar with the Normal distribution already have probably seen σ^2 instead of τ^{-1}. They are in fact reciprocals of each other. The change was motivated by simpler mathematical analysis. Just remember: The smaller the τ, the wider the distribution (i.e., we are more uncertain); the larger the τ, the tighter the distribution (i.e., we are more certain). Regardless, τ is always positive.

The probability density function of a $N(\mu, 1/\tau)$ random variable is:

$$f(x|\mu, \tau) = \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2}(x-\mu)^2\right)$$

We plot some different density functions of the Normal distribution in Figure 2.2.13.

```
import scipy.stats as stats

nor = stats.norm
x = np.linspace(-8, 7, 150)
mu = (-2, 0, 3)
tau = (.7, 1, 2.8)
colors = ["#348ABD", "#A60628", "#7A68A6"]
parameters = zip(mu, tau, colors)

for _mu, _tau, _color in parameters:
    plt.plot(x, nor.pdf(x, _mu, scale=1./_tau),
             label="$\mu = %d,\;\\tau = %.1f$" % (_mu, _tau),
             color=_color)
    plt.fill_between(x, nor.pdf(x, _mu, scale=1./_tau), color=_color,
                     alpha=.33)

plt.legend(loc="upper right")
plt.xlabel("$x$")
plt.ylabel("Density function at $x$")
plt.title("Probability distribution of three different Normal random \
          variables");
```

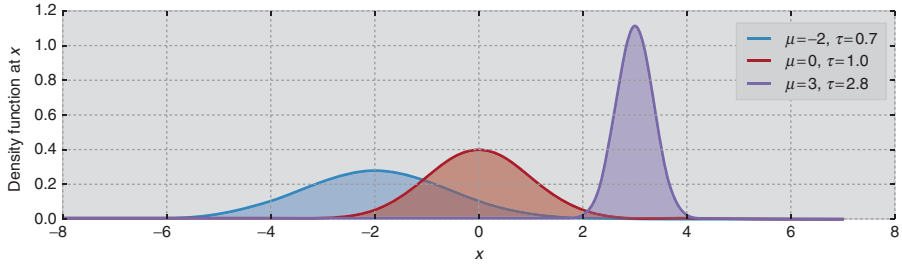

Figure 2.2.13: Probability distribution of three different Normal random variables

A Normal random variable can take on any real number, but the variable is very likely to be relatively close to μ. In fact, the expected value of a Normal is equal to its μ parameter:

$$E[X|\mu, \tau] = \mu$$

and its variance is equal to the inverse of τ:

$$Var(X|\mu, \tau) = \frac{1}{\tau}$$

Let's continue our modeling of the *Challenger* space craft.

```
import pymc as pm

temperature = challenger_data[:, 0]
D = challenger_data[:, 1]  # defect or not?

# Notice the "value" here. We will explain it later.
beta = pm.Normal("beta", 0, 0.001, value=0)
alpha = pm.Normal("alpha", 0, 0.001, value=0)


@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta*t + alpha))
```

We have our probabilities, but how do we connect them to our observed data? We can use a *Bernoulli* random variable, introduced in Section 2.2.3. Thus, our model can look like:

$$\text{Defect Incident, } D_i \sim \text{Ber}(\, p(t_i)\,), \quad i = 1 \ldots N$$

where $p(t)$ is our logistic function (and strictly between 0 and 1) and t_i is the temperatures we have observations about. Notice that in this code we had to set the values of `beta` and `alpha` to 0. The reason for this is that if `beta` and `alpha` are very large, they make `p` equal to 1 or 0. Unfortunately, `pm.Bernoulli` does not like probabilities of exactly 0 or 1, though they are mathematically well-defined probabilities. So by setting the coefficient values to `0`, we set the variable `p` to be a reasonable starting value. This has no effect on our results, nor does it mean we are including any additional information in our prior. It is simply a computational caveat in PyMC.

```
p.value
```

```
[Output]:
array([ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,
        0.5])
```

```
# Connect the probabilities in "p" with our observations through a
# Bernoulli random variable.
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)

model = pm.Model([observed, beta, alpha])
```

(*Continues*)

(*Continued*)

```
# mysterious code to be explained in Chapter 3
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)
```

```
[Output]:

[-----------------100%-----------------] 120000 of 120000 complete
    in 15.3 sec
```

We have trained our model on the observed data; now we can sample values from the posterior. Let's look at the posterior distributions for α and β, illustrated in Figure 2.2.14.

```
alpha_samples = mcmc.trace('alpha')[:, None]  # best to make them 1D
beta_samples = mcmc.trace('beta')[:, None]

figsize(12.5, 6)

# histogram of the samples
plt.subplot(211)
plt.title(r"Posterior distributions of the model parameters \
          $\alpha, \beta$")
plt.hist(beta_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\beta$", color="#7A68A6", normed=True)
plt.legend()

plt.subplot(212)
plt.hist(alpha_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=r"posterior of $\alpha$", color="#A60628", normed=True)
plt.xlabel("Value of parameter")
plt.xlabel("Density")
plt.legend();
```

All samples of β are greater than 0. If instead the posterior was centered around 0, we might suspect that $\beta = 0$, implying that temperature has no effect on the probability of defect. Similarly, all α posterior values are negative and far away from 0, implying that it is correct to believe that α is significantly less than 0. Regarding the spread of the data, we are very uncertain about what the true parameters might be (though considering the small sample size and the large overlap of defects and non-defects, this behavior is perhaps expected).

Next, let's look at the *expected probability* for a specific value of the temperature. That is, we average over all samples from the posterior to get a likely value for $p(t_i)$.

```
t = np.linspace(temperature.min() - 5, temperature.max()+5, 50)[:, None]
p_t = logistic(t.T, beta_samples, alpha_samples)

mean_prob_t = p_t.mean(axis=0)
```

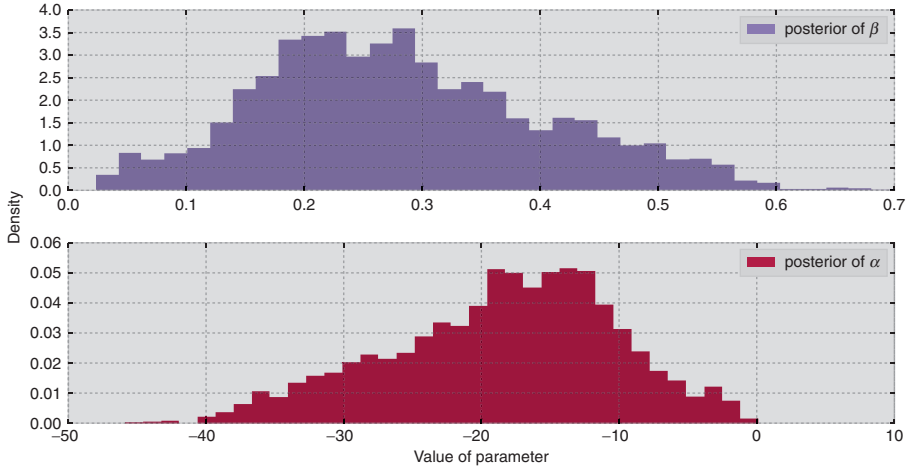

Figure 2.2.14: Posterior distributions of the model parameters α and β

```
figsize(12.5, 4)

plt.plot(t, mean_prob_t, lw=3, label="average posterior \nprobability \
         of defect")
plt.plot(t, p_t[0, :], ls="--", label="realization from posterior")
plt.plot(t, p_t[-2, :], ls="--", label="realization from posterior")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.title("Posterior expected value of the probability of defect, \
          including two realizations")
plt.legend(loc="lower left")
plt.ylim(-0.1, 1.1)
plt.xlim(t.min(), t.max())
plt.ylabel("Probability")
plt.xlabel("Temperature");
```

In Figure 2.2.15, we also plot two possible realizations of what the actual underlying system might be. Both are equally as likely as any other draw. The blue line is what occurs when we average all the 20,000 possible dotted lines together.

An interesting question to ask is, For what temperatures are we most uncertain about the defect probability? In Figure 2.2.16, we plot the expected value line *and* the associated 95% credible intervals (CI) for each temperature.

```
from scipy.stats.mstats import mquantiles

# vectorized bottom and top 2.5% quantiles for "credible interval"
qs = mquantiles(p_t, [0.025, 0.975], axis=0)
plt.fill_between(t[:, 0], *qs, alpha=0.7,
                 color="#7A68A6")
```

(*Continues*)

(*Continued*)

```
plt.plot(t[:, 0], qs[0], label="95% CI", color="#7A68A6", alpha=0.7)

plt.plot(t, mean_prob_t, lw=1, ls="--", color="k",
         label="average posterior \nprobability of defect")

plt.xlim(t.min(), t.max())
plt.ylim(-0.02, 1.02)
plt.legend(loc="lower left")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.xlabel("Temperature, $t$")

plt.ylabel("Probability estimate")
plt.title("Posterior probability of estimates, given temperature $t$");
```

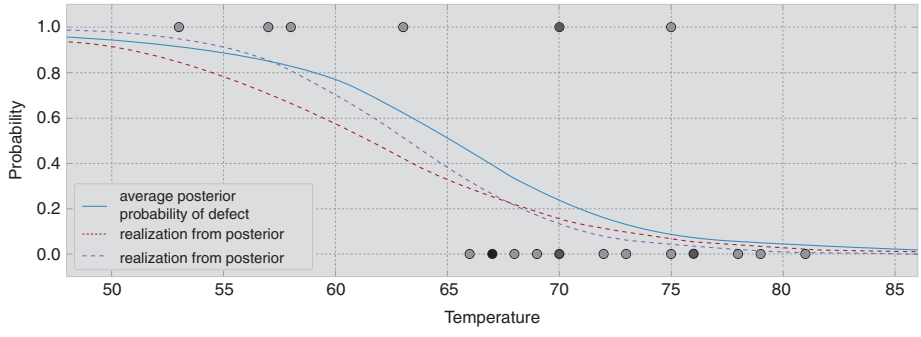

Figure 2.2.15: Posterior expected value of the probability of defect, including two realizations

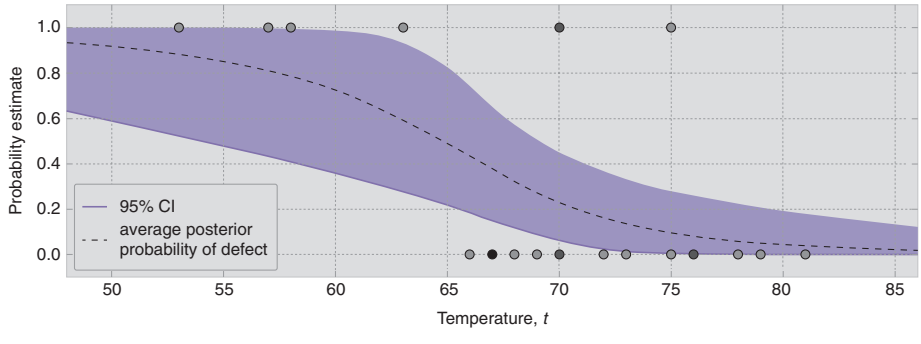

Figure 2.2.16: Posterior probability of estimates, given temperature *t*

The **95% credible interval**, or 95% CI, painted in purple, represents the interval, for each temperature, that contains 95% of the distribution. For example, at 65 degrees, we can be 95% sure that the probability of defect lies between 0.25 and 0.75. This is different from the frequentist *confidence interval*, which does not have the same interpretation.