introduction_to_bayes

November 13, 2020

1 *"When the facts change, I change my mind. What do you do, sir?"*

1.1 –John Maynard Keynes

2 Bayes' Theorem

$$P\left(\boldsymbol{\theta}|\mathbf{d}\right) = \frac{P\left(\mathbf{d}|\boldsymbol{\theta}\right)P\left(\boldsymbol{\theta}\right)}{P\left(\mathbf{d}\right)}$$

$\boldsymbol{\theta}$ are parameters, d are the data

| Means "conditional on''

2.1 This is really just rearranging the law of conditional probabilities

2.2 $P\left(\boldsymbol{\theta}|\mathbf{d}\right)P\left(\mathbf{d}\right) = P\left(\mathbf{d}|\boldsymbol{\theta}\right)P\left(\boldsymbol{\theta}\right)$

2.2.1 *Um, what?*

2.3 Let's check out a Venn diagram and explore conditional probabilities

By Gnathan87 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=15991401

2.4 What is the probability of A if we know we are in B_1? The equation for this is $P\left(A|B_1\right)$

2.5 Easy to see this is 100% or $P\left(A|B_1\right) = 1.0$

2.6 As a general rule, we can state

$$P\left(A|B_1\right) = \frac{P\left(A \cap B_1\right)}{P\left(B_1\right)}$$

2.7 or, equivalently

$$P\left(A \cap B_1\right) = P\left(A|B_1\right)P\left(B_1\right)$$

2.8 So what about $P\left(A|B_2\right)$?

2.9 Let's check out a Venn diagram and explore conditional probabilities

By Gnathan87 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=15991401

$$P\left(A|B_2\right) = \frac{P\left(A \cap B_2\right)}{P\left(B_2\right)} = \frac{0.12}{0.12 + 0.04} = 0.75$$

2.9.1 So now we can derive Bayes' theorem because joint probabilities are symmetrical. Switching notation to

$$\boldsymbol{\theta} \text{ and } \mathbf{d}$$

$$P\left(\boldsymbol{\theta} \cap \mathbf{d}\right) = P\left(\mathbf{d} \cap \boldsymbol{\theta}\right)$$

$$P\left(\boldsymbol{\theta}|\mathbf{d}\right) P\left(\mathbf{d}\right) = P\left(\mathbf{d}|\boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}\right)$$

2.9.2 With the tiniest little algebra, we get Bayes' theorem – #boom#!

$$P\left(\boldsymbol{\theta}|\mathbf{d}\right) = \frac{P\left(\mathbf{d}|\boldsymbol{\theta}\right) P\left(\boldsymbol{\theta}\right)}{P\left(\mathbf{d}\right)}$$

3 So, what does this really mean?:

3.1 Where are my keys??????

3.2 Let's play with another concrete example, one hinging on life, death, trust, and promises kept!

3.2.1 You have a plant at home, and you're going to go away for a week.

3.2.2 If it gets watered, its probability of dying is 15%. If it doesn't get watered, it is 80% likely to die.

3.2.3 You ask your partner to water it for you and you are 90% certain they will do it.

3.2.4 We can express this all in terms of probabilities and conditional probabilities.

3.3 First a couple definitions:

3.3.1 θ_w: partner waters the plant

3.3.2 θ_{nw}: partner forgets to water the plant

3.3.3 d_a: plant is alive when we return

3.3.4 d_d: plant is dead when we return

3.3.5 $\mathbf{d} = [d_a, d_d]$: a vector of all possible outcomes

3.3.6 $\boldsymbol{\theta} = [\theta_w, \theta_{nw}]$: a vector of all possible outcomes

3.4 Cool, so let's express what we know in probability equations

$$P\left(d_d|\theta_w\right) = 0.15$$

$$P\left(d_d|\theta_{nw}\right) = 0.8$$

$$P\left(\theta_w\right) = 0.9$$

$$P\left(\theta_{nw}\right) = 0.1$$

3.5 And we can assign these as python variables to get our maths groove on

[1]:
```
PDd_thw=0.15
PDd_thnw = 0.8
Prior_thw = 0.9
Prior_thnw = 0.1
```

3.6 Now we can ask questions like, "what is the probability the plant is dead"

3.6.1 To calculate, we add up all the conditional probablities like this:

$$P\left(d_d\right) = P\left(d_d \cap \theta_w\right) + P\left(d_d \cap \theta_{nw}\right)$$

$$P\left(d_d\right) = P\left(d_d|\theta_w\right) P\left(\theta_w\right) + P\left(d_d|\theta_{nw}\right) P\left(\theta_{nw}\right)$$

[2]:
```python
PDd = PDd_thw*Prior_thw + PDd_thnw*Prior_thnw
print ('Probability Plant is dead = {0:.3f}'.format(PDd))
```

```
Probability Plant is dead = 0.215
```

3.7 Since we only have two discrete outcomes, the probability of the plant being alive is simply

$$P(d_a) = 1 - P(d_d)$$

[3]:
```python
PDa = 1-PDd
print ('Probability Plant is alive = {0:.3f}'.format(PDa))
```

```
Probability Plant is alive = 0.785
```

4 Great! So we can incorporate all the possible arrangements of events to determine likely outcomes. But....what we are *really* interested in is what we learn with partial information. This is where household harmony can be made or broken!

4.1 Example 1: We come home and see that the plant is dead (crumbs!). Who to blame? What is the probability that our partner forgot to water it?

4.2 Mathematically, this is;

$$P(\theta_{nw}|d_d)$$

4.3 We can use Bayes' theorem to evaluate this new information (e.g. we have observed that the plant is dead)

$$P(\theta_{nw}|d_d) = \frac{P(d_d|\theta_{nw})\,P(\theta_{nw})}{P(d_d)}$$

[4]:
```python
PthnwDd = PDd_thnw * Prior_thnw/ PDd
print ("Probability that partner failed to water the plant")
print("having seen it's dead is {0:.3f}".format(PthnwDd))
```

```
Probability that partner failed to water the plant
having seen it's dead is 0.372
```

4.4 Alternatively, we can see the converse: How likely did our partner water the plant given that it's alive?

$$P(\theta_w|d_a) = \frac{P(d_a|\theta_w)\,P(\theta_w)}{P(d_a)}$$

[5]:
```python
PthwDa = (1-PDd_thw) * Prior_thw/ PDa
print ("Probability that partner did water the plant")
print ("having seen it's alive is {0:.3f}".format(PthwDa))
```

```
Probability that partner did water the plant
having seen it's alive is 0.975
```

4.5 How likely did our partner forget given that we see it's alive?

$$P\left(\theta_{nw}|d_a\right) = \frac{P\left(d_a|\theta_{nw}\right) P\left(\theta_{nw}\right)}{P\left(d_a\right)}$$

[6]:
```
PthnwDa = (1-PDd_thnw) * Prior_thnw/ PDa
print ("Probability that partner forgot to water the plant")
print("having seen it's alive is {0:.3f}".format(PthnwDa))
```

```
Probability that partner forgot to water the plant
having seen it's alive is 0.025
```

5 Right then, but we are in the world of continuous variables, not simple discrete probabilities

5.1 This means that we end up with probability density functions rather than discrete probabilities and the denominator on the RHS gets tricky to evaluate (the total probability). Luckily, we are mostly conncerned with finding the parameters that maximize the probability and less concerned with the probability itself.

5.2 This is a learning framework, where what we know at the end is a function of what we started with and what we *learned* through a new experiment (model) or new information

$$\underbrace{P(\boldsymbol{\theta}|\mathbf{d})}_{\substack{\text{posterior}\\\text{pdf}}} \propto \underbrace{\mathcal{L}(\boldsymbol{\theta}|\mathbf{d})}_{\substack{\text{likelihood}\\\text{function}}} \underbrace{P(\boldsymbol{\theta})}_{\substack{\text{prior}\\\text{pdf}}}$$

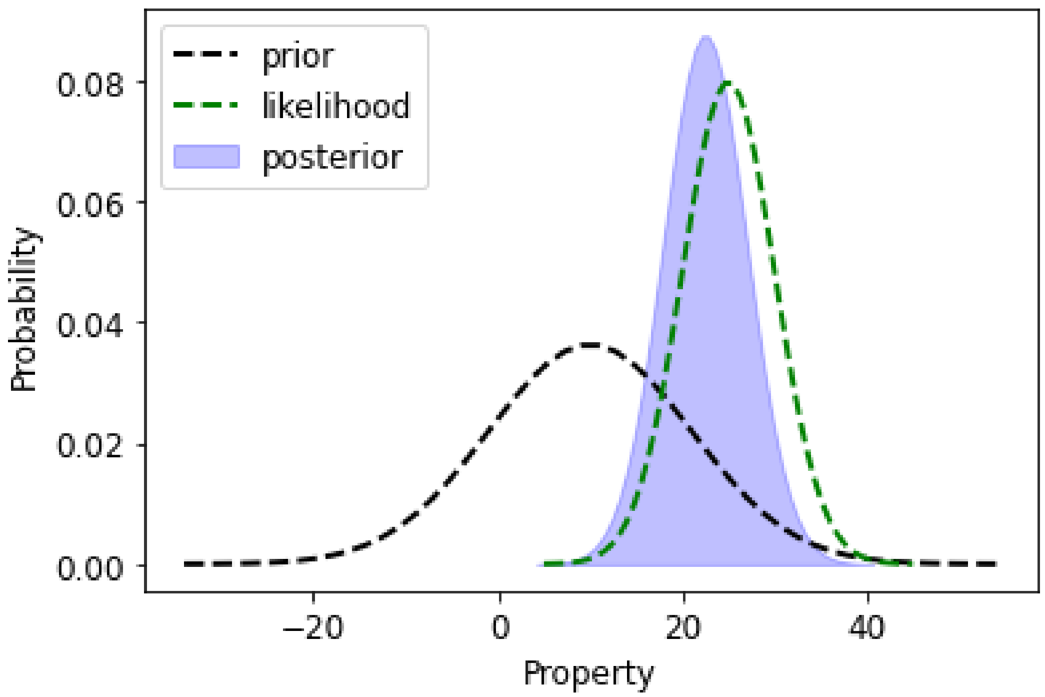
$$\underbrace{P(\boldsymbol{\theta}|\mathbf{d})}_{\substack{\text{what we}\\\text{know now}}} \propto \underbrace{\mathcal{L}(\boldsymbol{\theta}|\mathbf{d})}_{\substack{\text{what we}\\\text{learned}}} \underbrace{P(\boldsymbol{\theta})}_{\substack{\text{what we}\\\text{knew}}}$$

5.3 Let's look at an interactive example of how distributions behave

[7]:
```
import bayes_helper as bh
from ipywidgets import interact
```

[8]:
```
bh.plot_posterior(prior_mean=10, prior_std=11, likeli_mean = 25, likeli_std=5)
```

```
findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
```



[9]:
```
interact(bh.plot_posterior,
         prior_mean=(10, 100., .5), likeli_mean=(10,200, 1),
         prior_std=(.1, 50, 1), likeli_std=(.1, 20, .1));
```

```
interactive(children=(FloatSlider(value=55.0, description='prior_mean', min=10.
↪0, step=0.5), FloatSlider(value…
```

[9]:
```
<function bayes_helper.plot_posterior(prior_mean, prior_std, likeli_mean,
likeli_std, legend=True, savefigure=False)>
```

6 Here's a "mandatory coin-flipping example"

Borrowed from Bayesian Methods for Hackers. The full Github repository is available at http://github.com/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers.

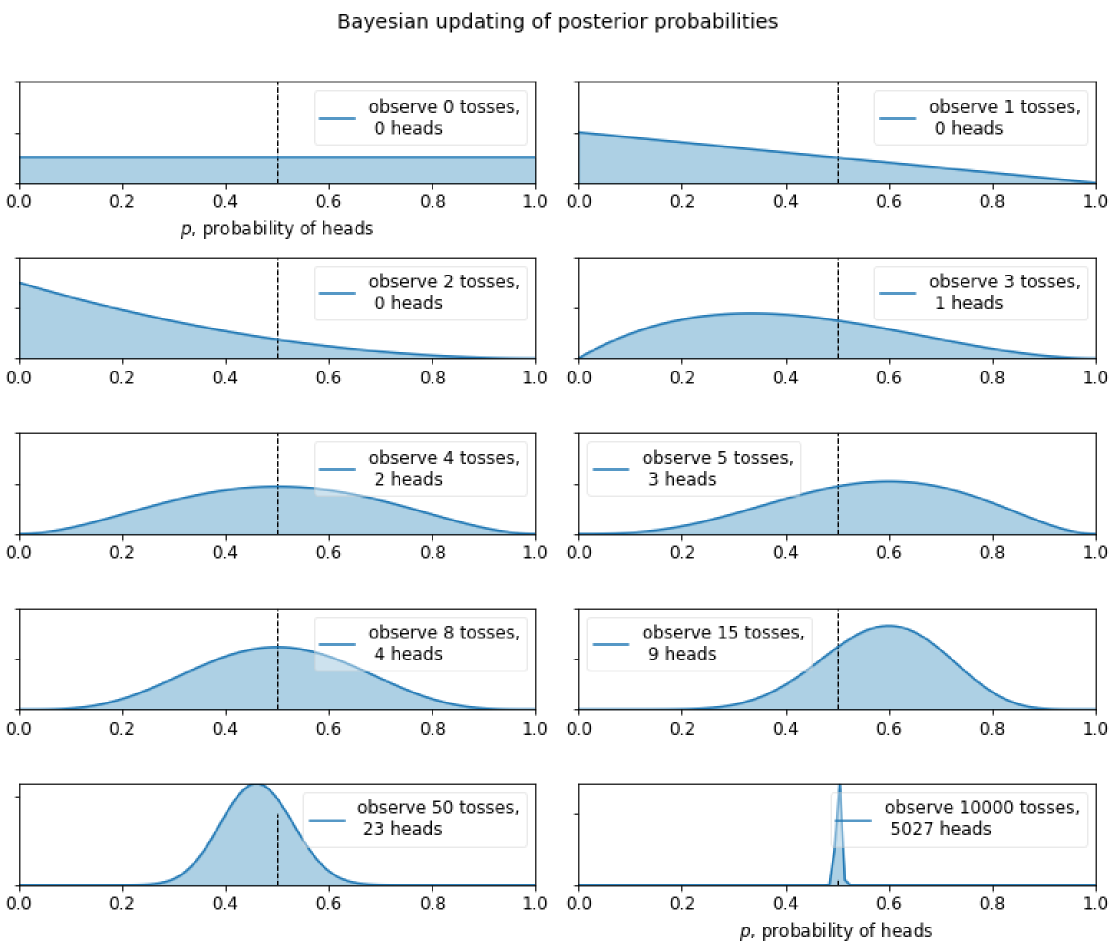
6.0.1 We can start with an "ignorance" prior - equal probabilities of all outcomes (both, in the case—heads and tails). By flipping a coin we can observer outcomes, constantly updating and learning from each experiment.

[10]:
```
max_trials = 10000


# The code below can be passed over, as it is currently not important, plus it
# uses advanced topics we have not covered yet.
```

```python
%matplotlib inline
from IPython.core.pylabtools import figsize
import numpy as np
from matplotlib import pyplot as plt
figsize(11, 9)

import scipy.stats as stats

dist = stats.beta
n_trials = [0, 1, 2, 3, 4, 5, 8, 15, 50, max_trials]
data = stats.bernoulli.rvs(0.5, size=n_trials[-1])
x = np.linspace(0, 1, 100)

# For the already prepared, I'm using Binomial's conj. prior.
for k, N in enumerate(n_trials):
    sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
    plt.xlabel("$p$, probability of heads") \
        if k in [0, len(n_trials) - 1] else None
    plt.setp(sx.get_yticklabels(), visible=False)
    heads = data[:N].sum()
    y = dist.pdf(x, 1 + heads, 1 + N - heads)
    plt.plot(x, y, label="observe %d tosses,\n %d heads" % (N, heads))
    plt.fill_between(x, 0, y, color="#348ABD", alpha=0.4)
    plt.vlines(0.5, 0, 4, color="k", linestyles="--", lw=1)

    leg = plt.legend()
    leg.get_frame().set_alpha(0.4)
    plt.autoscale(tight=True)


plt.suptitle("Bayesian updating of posterior probabilities",
             y=1.02,
             fontsize=14)

plt.tight_layout()
```

```
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
```

```
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
<ipython-input-1-0b50ece5e486>:21: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  sx = plt.subplot(len(n_trials) / 2, 2, k + 1)
findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
```



The posterior probabilities are represented by the curves, and our uncertainty is proportional to the width of the curve. As the plot above shows, as we start to observe data our posterior probabilities start to shift and move around. Eventually, as we observe more and more data (coin-flips), our probabilities will tighten closer and closer around the true value of $p = 0.5$ (marked by a dashed line).

Notice that the plots are not always *peaked* at 0.5. There is no reason it should be: recall we assumed we did not have a prior opinion of what p is. In fact, if we observe quite extreme data, say 8 flips and only 1 observed heads, our distribution would look very biased *away* from lumping around 0.5 (with no prior opinion, how confident would you feel betting on a fair coin after observing 8 tails and 1 head). As more data accumulates, we would see more and more probability being assigned at $p = 0.5$, though never all of it.

```
[ ]:
```

```
[ ]:
```