

# bayesian

August 29, 2021

```
In [2]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st
%matplotlib inline
sns.set(style='ticks', palette='Set2')
```

```
In [11]: #Example in Assignment of Week 4
```

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import math

# Set a random seed for repeated computation
np.random.seed(123)

# Global constants - these are what we need to change
# This is my initial belief about the mean of the average IQ score on campus
prior_sigma = 10 # my uncertainty about the mean
prior_mean = 100 # my initial belief about the mean
sigma_observations = 3 # uncertainty in my observations

# New observations - We are going to be updating this list to see how observing
# different data changes our beliefs about the average IQ score
new_data = [110,110,110,125,125]

# Compute some statistics on the new data
n = len(new_data)

# Some hairy math to avoid doing integrals (Wikipedia has the math!)
if n != 0:
    posterior_mean = (( 1 / prior_sigma ** 2) + n / sigma_observations ** 2)**(-1) * ((
    posterior_sd = (1 / prior_sigma ** 2 + n / sigma_observations ** 2)**(-1)
else:
    posterior_mean = prior_mean
```

```

posterior_sd = prior_sigma

# Plot the distribution of the prior and the posterior
x = np.linspace(50, 150, 100)
plt.plot(x, mlab.normpdf(x, prior_mean, prior_sigma), color = "blue")
plt.plot(x, mlab.normpdf(x, posterior_mean, posterior_sd), color = "red", linestyle =
plt.title("Red = Posterior, Blue = Prior")

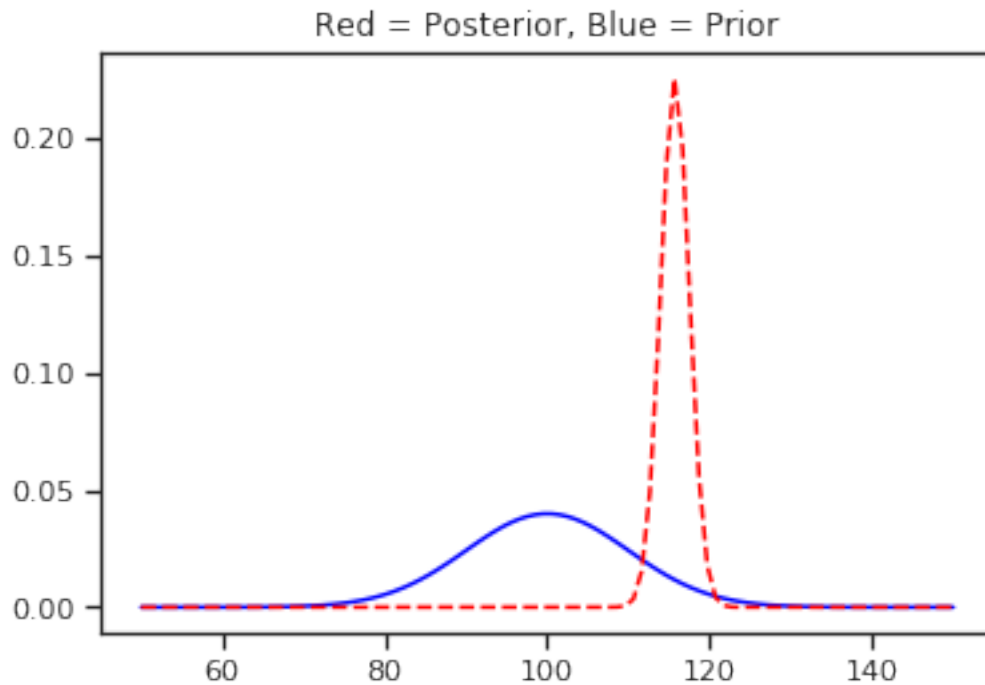
# To view the graph, comment-out print(posterior_mean)
# To view the numerical output, comment-out plt.show()

plt.show()
print(posterior_mean)

print(posterior_sd)

```

/opt/conda/lib/python3.6/site-packages/ipykernel\_launcher.py:33: MatplotlibDeprecationWarning:  
/opt/conda/lib/python3.6/site-packages/ipykernel\_launcher.py:34: MatplotlibDeprecationWarning:



115.71709233791748  
1.768172888015717

# 1 Bayesian in Python

In this tutorial, we are going to go over basic bayesian analysis in python.

## 1.1 Review

**Prior  $p(H)$ :** Our prior reflects what we know about the value of some parameter before seeing data. This could refer to previous trials and distributions.

**Likelihood  $p(D|H)$ :** what is the plausibility that our data is observed, given our prior?

**Posterior  $p(H|D)$ :** This is result of the Bayesian analysis and reflects all that we know about a problem (given our data and model).

**Evidence  $p(D)$ :** Evidence is the probability of observing the data averaged over all the possible values the parameters can take. Also known as the normalizing factor. The normalising constant makes sure that the resulting posterior distribution is a true probability distribution by ensuring that the sum of the distribution is equal to 1.

Because  $p(D)$  is considered a normalizing constant we can say:  $p(H|D) \propto p(D|H) * p(H)$

## 1.2 Coin - Flipping Problem

Let's think of these terms in the context of a coin-flipping experiment.

On a standard coin, we have two sides, heads or tails. Both of which are equally likely to show after a coin flip, or a 50% probability.

In the case of a coin-flipping trials, we may want to consider this probability our prior.

Let's go ahead and create our prior distribution:

```
In [3]: coin_flips_prior = np.random.binomial(n = 1, p = 0.5, size = 1000)
        coin_flips_prior[:5]
```

```
Out[3]: array([0, 0, 0, 1, 1])
```

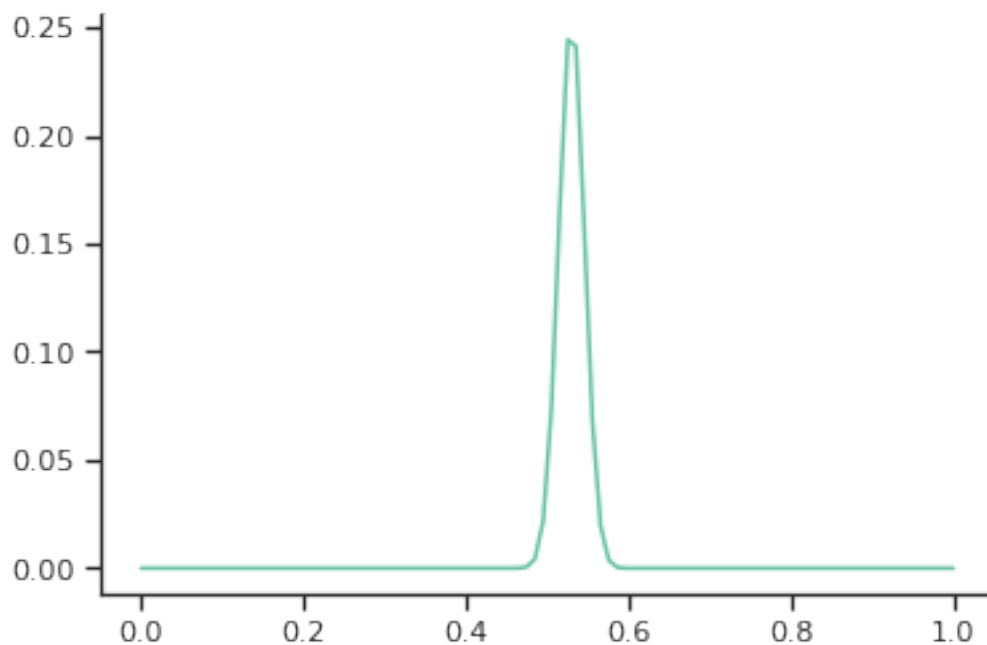
```
In [4]: params = np.linspace(0,1,100)
        params
```

```
Out[4]: array([0.          , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
               0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
               0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
               0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
               0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
               0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
               0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
               0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
               0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
               0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
               0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
               0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
               0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
               0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
               0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
               0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
               0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
               0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
               0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
               0.95959596, 0.96969697, 0.97979798, 0.98989899, 0.99999999])
```

```
0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.      ])
```

```
In [5]: p_prior = np.array([np.product(st.bernoulli.pmf(coin_flips_prior, p)) for p in params])
```

```
In [6]: p_prior = p_prior/np.sum(p_prior)
plt.plot(params, p_prior)
sns.despine()
```

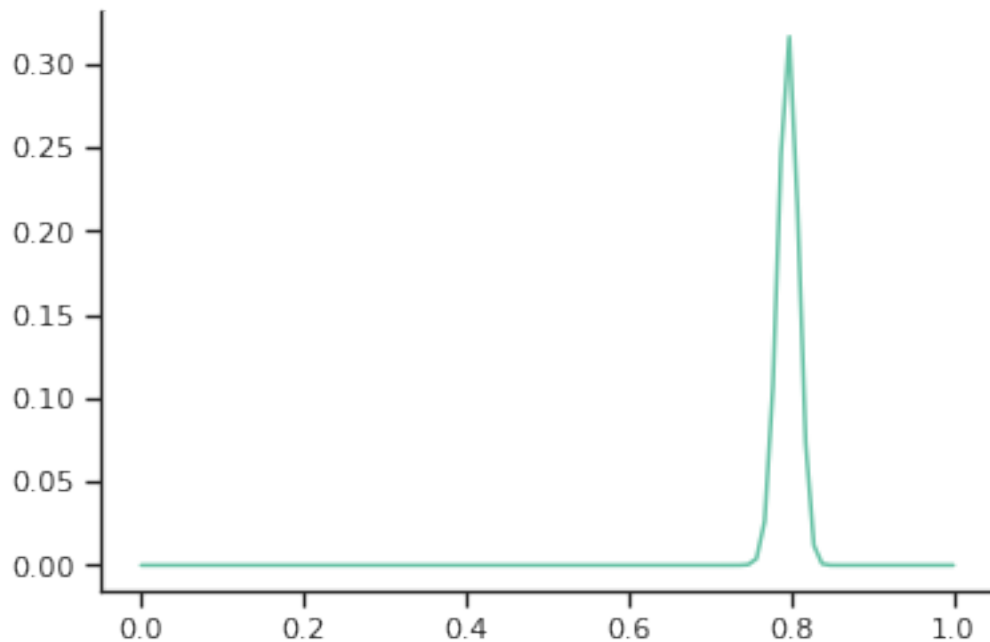


As you can see, our prior distribution peaks at 0.5 which is what our probability for our fair coin is.

Now, let's introduce some observations from trials with an unfair coin. Let's say the probability is now weight 80-20, where the probability a head is shown is 0.8.

Let's create this sampling distribution:

```
In [7]: coin_flips_observed = np.random.binomial(n=1, p=0.8, size = 1000)
p_observed = np.array([np.product(st.bernoulli.pmf(coin_flips_observed, p)) for p in params])
p_observed = p_observed/np.sum(p_observed)
plt.plot(params, p_observed)
sns.despine()
```

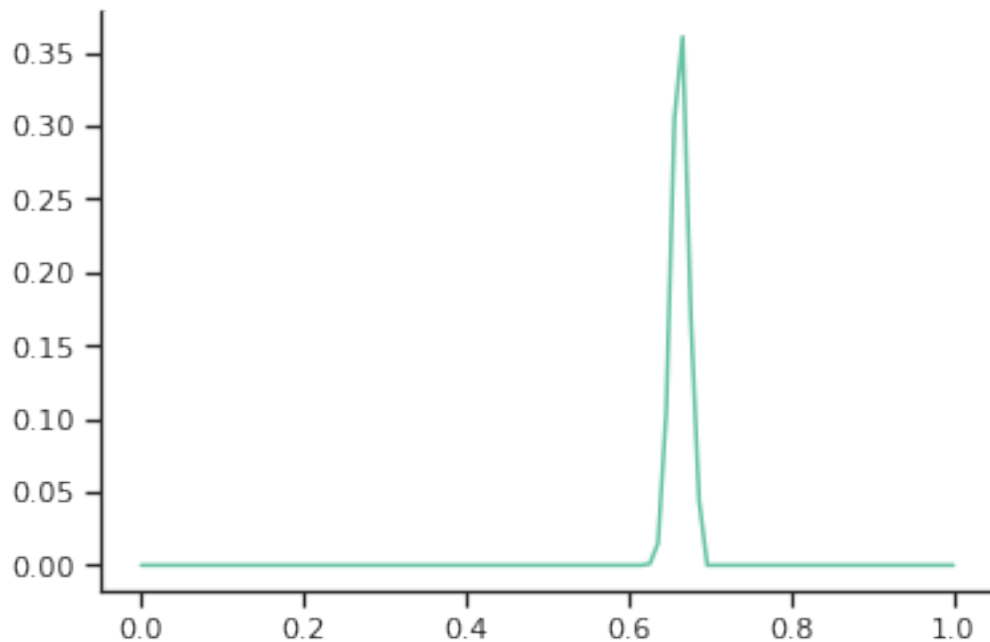


The peak for our sampling distribution is around 0.8.

While our observations from our sampling distribution indicate a probability around 0.8, because our prior is 0.5, we have to assess the likelihood that these values could be observed and find our posterior distribution.

Remember,  $p(H|D) \propto p(D|H) * p(H)$  OR *Posterior*  $\propto$  *Likelihood* \* *Prior*

```
In [8]: p_posterior = [p_prior[i] * p_observed[i] for i in range(len(p_prior))]
        p_posterior = p_posterior/np.sum(p_posterior)
        plt.plot(params, p_posterior)
        sns.despine()
```



### 1.3 University of Michigan Student IQs

We'll do another example where we have some prior belief about the IQ of University of Michigan students.

For our prior distribution, we'll have a normal distribution with a mean IQ of 100 and a standard deviation of 10.

```
In [0]: prior_distribution = np.random.normal(100, 10, 1000)
plt.hist(prior_distribution)
sns.despine()
```

Now, let's say we are collecting some observations of student IQs which takes the shape of a normal distribution with mean 115 and standard deviation of 7.5 and want to construct our posterior distribution.

In order to do this, we update our prior by calculating the mean and variance after each observation.

The equations for our updated prior mean and variance are:

$$\text{Updated Prior Mean} = \frac{\sigma_{\text{observed}}^2 \mu + \sigma_{\text{prior}}^2 x}{\sigma_{\text{observed}}^2 + \sigma_{\text{prior}}^2}$$

$$\text{Updated Prior Variance} = \frac{\sigma_{\text{observed}}^2 \sigma_{\text{prior}}^2}{\sigma_{\text{observed}}^2 + \sigma_{\text{prior}}^2}$$

```

In [0]: np.random.seed(5)
        observed_distribution = np.random.normal(115, 10, 1000)
        mu = [100] * 1000
        sigma = [10] * 1000

        mu[0] = (10**2*observed_distribution[0] + (10**2)*100)/(10**2+10**2)
        sigma[0] = (10**2*10**2)/(10**2+10**2)

        for i in range(1000):
            if i == 999:
                break
            mu[i + 1] = (sigma[i]*observed_distribution[i+1] + (10**2)*mu[i] )/(sigma[i]+10**2)
            sigma[i + 1] = (sigma[i]*10**2)/(sigma[i]+10**2)

        posterior_distributions = [[]] * 20

        for i in range(20):
            posterior_distributions[i] = np.random.normal(mu[i], sigma[i], 1000)

        plt.hist(prior_distribution)
        plt.hist(observed_distribution, alpha = 0.75)
        plt.hist(posterior_distributions[14], alpha = 0.5)
        sns.despine()

```