(6) $$\frac{d}{d\theta}\left(\theta^t (1-\theta)^h\right) = t\,\theta^{t-1}(1-\theta)^h - \theta^t h (1-\theta)^{h-1}$$
$$= \theta^{t-1}(1-\theta)^{h-1}\left[t(1-\theta) - h\theta\right]$$

$$\frac{d^2}{d\theta^2}\left(\theta^{t-1}\cdot(1-\theta)^{h-1}\cdot\left[t\cdot(1-\theta) - h\cdot\theta\right]\right)$$
$$= \left[(t-1)\cdot\theta^{t-2}\cdot(1-\theta)^{h-1} - \theta^{t-1}\cdot(h-1)\cdot(1-\theta)^{h-2}\right]\cdot\left[t(1-\theta) - h\cdot\theta\right] - \left[\theta^{t-1}\cdot(1-\theta)^{h-1}\right]\cdot\left[t+h\right]$$
$$= \underbrace{t(t-1)\cdot\theta^{t-2}\cdot(1-\theta)^h}\ \underbrace{-t(h-1)\cdot\theta^{t-1}\cdot(1-\theta)^{h-1}}\ \underbrace{-h(t-1)\cdot\theta^{t-1}\cdot(1-\theta)^{h-1}}\ \underbrace{-h\,\theta^t\cdot(h-1)\cdot(1-\theta)^{h-2}} - \theta^{t-1}\cdot(1-\theta)^{h-1}\cdot(t+h)$$
$$= \theta^{t-2}\cdot(1-\theta)^h\cdot\left(t(t-1)\right) + \theta^{t-1}\cdot(1-\theta)^{h-1}\cdot\left[\underbrace{-t\cdot(h-1) - h(t-1) + (t+h)}_{\substack{-th+t-th+h+t+h \\ =2[(t+h)-th]}}\right] + \theta^t\cdot(1-\theta)^{h-2}\cdot\left(-h(h-1)\right)$$

$$= \theta^{t-2}\cdot(1-\theta)^h\cdot\left[t(t-1)\right] + \theta^{t-1}\cdot(1-\theta)^{h-1}\cdot\left[2\left((t+h) - th\right)\right] + \theta^t\cdot(1-\theta)^{h-2}\cdot\left(h(1-h)\right)$$

$$d/d\theta\left(\log\left(\theta^t(1-\theta)^h\right)\right) = d/d\theta\left(t\log(\theta) + h\log(1-\theta)\right)$$
$$= \frac{t}{\theta} - \frac{h}{1-\theta}$$

$$d/d\theta\left(t/\theta - h/(1-\theta)\right) = -t/\theta^2 + h/(1-\theta)^2 = \frac{h}{(1-\theta)^2} - \frac{t}{\theta^2}$$

(7) $$\Rightarrow \left(\log(f(x))\right)' = \frac{f'(x)}{f(x)} \Rightarrow \left(\log(f(x))\right)' = 0 \iff f'(x) = 0$$

$$\Rightarrow \left(\log(f(x))\right)'' = \frac{f''(x)f(x) - f(x)'f(x)'}{f(x)^2} = \frac{f''(x)}{f(x)} - \frac{f'(x)^2}{f(x)^2}$$

for f to have a maximum at an arbitrary point x_k we need $\begin{cases} f'(x_k) = 0 \\ f''(x_k) < 0 \end{cases}$

as for $\log(f(x)) = g(x)$, we have $\begin{cases} g'(x) = 0 \Leftrightarrow f'(x) = 0 \\ g''(x) = \frac{f''(x)}{f(x)} - \frac{f'(x)^2}{f(x)^2} \end{cases}$ $\rightarrow$ in regards to x_k where $f'(x_k) = 0$

$$g'(x_k) = f'(x_k) = 0$$

$$g''(x_k) = \frac{f''(x_k)}{f(x)} - \frac{f'(x_k)^2}{f(x)^2} = \frac{f''(x_k)}{f(x)}$$ $\rightarrow > 0$

$\Rightarrow g''(x)$ has the same sign as $f''(x)$ in f's critical points

$\Rightarrow$ when f has a local maximum we have

$f'(x) = 0 \wedge f''(x) < 0 \Leftrightarrow g'(x) = 0 \wedge g''(x) < 0$

So any local maximum in f is also a local maximum of g

$\Rightarrow$ concluding, we have that the computation of the log-likelihood is trivial compared to the standard likelihood, and the logarithm preserves the monotony of the original function. Implying that we can compute the log-likelihood when trying to solve likelihood optimization problems.

8

- find posterior distribution $\longrightarrow p(\theta|D) = \frac{p(D|\theta)\,p(\theta)}{p(D)}$
- prior is $Beta(a,b) = p(\theta)$

- so we know that $p(D|\theta)$ is modeled as a binomial distribution with param. θ
- since the random variable originates from a Bernoulli experiment we can assume that the variable is i.i.d

$\Rightarrow$ likelihood $= P(D|\theta) = \theta^m (1-\theta)^{N-m}$ $\rightarrow$ leading constant has no impact in maximizing likelihood

$$\Rightarrow p(\theta|D) = \left[\theta^m (1-\theta)^{N-m} \cdot \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1} (1-\theta)^{b-1} \right] / P(D)$$

$$\propto \theta^{m+a-1} (1-\theta)^{N-m-b-1}$$ $\longrightarrow$ similar to Beta distribution

$$\sim Beta(m+a, N-m-b) = \underbrace{\frac{\Gamma(N+a-b)}{\Gamma(m+a)\Gamma(N-m-b)}}_{1/P(D)} \theta^{m+a-1} (1-\theta)^{N-m-b-1}$$

$\Rightarrow$ so now we know that $P(\theta | D) = \text{Beta}(m+a, N-m-b)$

$\rightarrow$ and $E[\theta | D]$ is known for a Beta distribution $\longrightarrow \frac{\alpha}{\alpha+\beta} \longrightarrow \frac{m+a}{N+a-b}$

$\rightarrow$ the prior distribution was also a Beta dist. with $E[\theta] = \frac{a}{a+b}$

$\rightarrow$ we need to show that $\frac{m+a}{N+a-b} = \lambda\left(\frac{a}{a+b}\right) + (1-\lambda)\,\theta_{MLE}$

$$\rightarrow \theta_{MLE} = \arg\max_\theta p(D|\theta) = \arg\max_\theta \theta^m (1-\theta)^{N-m} = \arg\max_\theta \log\left(\theta^m (1-\theta)^{N-m}\right)$$

$$= \arg\max_\theta m \log\theta + (N-m)\log(1-\theta)$$

$$\rightarrow \left(m\log(\theta) + (N-m)\log(1-\theta)\right)' = 0 \Leftrightarrow$$

$$\Leftrightarrow \frac{m}{\theta} - \frac{(N-m)}{1-\theta} = 0 \Leftrightarrow \frac{1-\theta}{\theta} = \frac{N-m}{m} \Leftrightarrow \frac{1}{\theta} = \frac{N-m-m}{m} \Leftrightarrow \theta = \frac{m}{N-2m}$$

$$\Leftrightarrow \frac{m+a}{N+a-b} = \lambda\left(\frac{a}{a+b}\right) + (1-\lambda)\left(\frac{m}{N-2m}\right) = \lambda\left(\frac{a}{a+b}\right) + \frac{m}{N-2m} - \frac{\lambda m}{N+2m} \Leftrightarrow$$

$$\Leftrightarrow \frac{m+a}{N+a-b} - \frac{m}{N-2m} = \lambda\left(\frac{a}{a+b} - \frac{m}{N-2m}\right) \Leftrightarrow \lambda = \frac{\frac{m+a}{N+a-b} - \frac{m}{N-2m}}{\frac{a}{a+b} - \frac{m}{N-2m}}$$

$\Rightarrow$ This can be solved for λ and theoretically (assuming no mistakes) should result in a value in $[0,1]$

(9) $\lambda_{MAP} = \arg\max_\lambda p(\lambda | x)$

$$p(\lambda|x) \propto p(x|\lambda)\,p(\lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \cdot \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}$$

$$\propto \lambda^x e^{-\lambda} \cdot \lambda^{a-1} e^{-b\lambda} = \lambda^{x+a-1} e^{-\lambda(b+1)}$$

$\rightarrow$ we are looking for $\arg\max_\lambda \left(\lambda^{x+a-1} e^{-\lambda(b+1)}\right)$

⇒ taking the logarithm (preserving monotony)

$$\log\left(\lambda^{x+a-1} e^{-\lambda(b+1)}\right) = (x+a-1)\log(\lambda) - \lambda(b+1)\underbrace{\log(e)}_{1}$$

$$\left(\log(p(\lambda|x))\right)' = \frac{x+a-1}{\lambda} - (b+1)$$

$$= 0 \Leftrightarrow \frac{x+a-1}{b+1} = \lambda$$

$$\left(\log(p(\lambda|x))\right)'' = \frac{-(x+a-1)}{\lambda^2}$$

$$\to -\frac{(x+a-1)}{\frac{(x+a-1)^2}{(b+1)^2}} = -\frac{\overbrace{(b+1)^2}^{>0}}{\underbrace{x+a-1}_{>0}} < 0 \;//$$

$$\Rightarrow \text{so } \lambda_{MAP} = \arg\max_{\lambda}\left(p(\lambda|x)\right) = \frac{x+a-1}{b+1}$$

notebook

November 10, 2021

1 Programming Task: Probabilistic Inference

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from scipy.special import loggamma
%matplotlib inline
```

1.1 Your task

This notebook contains code implementing the methods discussed in `Lecture 3: Probabilistic Inference`. Some functions in this notebook are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring which specifies the input and and expected output. Write your code in a way that adheres to it. You may only use plain python and anything that we imported for you above such as `numpy` functions (i.e. no scikit-learn classifiers).

1.2 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook (`Kernel -> Restart & Run All`) 2. Export/download the notebook as PDF (`File -> Download as -> PDF via LaTeX (.pdf)`) 3. Concatenate your solutions for other tasks with the output of Step 2. On Linux you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` **Version 5.5 or later** by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 Simulating data

The following function simulates flipping a biased coin.

[ ]:
```
# This function is given, nothing to do here.
def simulate_data(num_samples, tails_proba):
    """Simulate a sequence of i.i.d. coin flips.
```

```
    Tails are denoted as 1 and heads are denoted as 0.

    Parameters
    ----------
    num_samples : int
        Number of samples to generate.
    tails_proba : float in range (0, 1)
        Probability of observing tails.

    Returns
    -------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    """
    return np.random.choice([0, 1], size=(num_samples), p=[1 - tails_proba,␣
↪tails_proba])
```

[ ]:
```
np.random.seed(123)  # for reproducibility
num_samples = 20
tails_proba = 0.7
samples = simulate_data(num_samples, tails_proba)
print(samples)
```

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

2 Important: Numerical stability

When dealing with probabilities, we often encounter extremely small numbers. Because of limited floating point precision, directly manipulating such small numbers can lead to serious numerical issues, such as overflows and underflows. Therefore, we usually work in the **log-space**.

For example, if we want to multiply two tiny numbers a and b, we should compute $\exp(\log(a) + \log(b))$ instead of naively multiplying $a \cdot b$.

For this reason, we usually compute **log-probabilities** instead of **probabilities**. Virtually all machine learning libraries are dealing with log-probabilities instead of probabilities (e.g. Tensorflow-probability or Pyro).

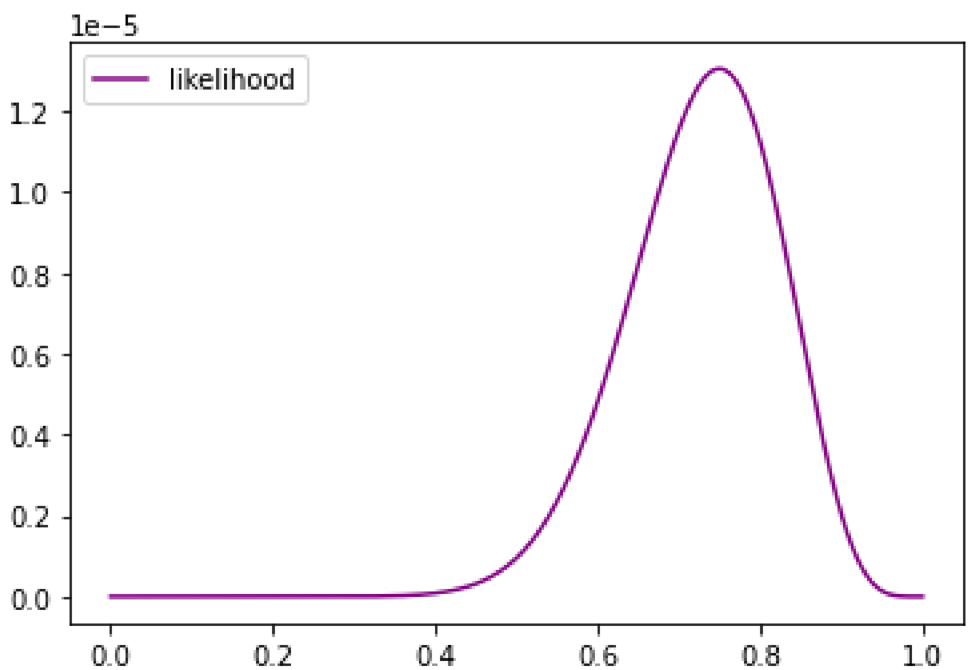
2.1 Task 1: Compute $\log p(\mathcal{D} \mid \theta)$ for different values of θ

[ ]:
```
def compute_log_likelihood(theta, samples):
    """Compute log p(D | theta) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-likelihood.
    samples : array, shape (num_samples)
```

```
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.

    Returns
    -------
    log_likelihood : array, shape (num_points)
        Values of log-likelihood for each value in theta.
    """

    tails = np.count_nonzero(samples == 1)
    heads = np.count_nonzero(samples == 0)

    return np.log(theta**tails*(1-theta)**heads)
```

[ ]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)
log_likelihood = compute_log_likelihood(x, samples)
likelihood = np.exp(log_likelihood)
plt.plot(x, likelihood, label='likelihood', c='purple')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7f121a63e2b0>



Note that the likelihood function doesn't define a probability distribution over θ — the integral $\int_0^1 p(\mathcal{D} \mid \theta) d\theta$ is not equal to one.

To show this, we approximate $\int_0^1 p(\mathcal{D} \mid \theta) d\theta$ numerically using the rectangle rule.

[ ]:
```
# 1.0 is the length of the interval over which we are integrating p(D | theta)
int_likelihood = 1.0 * np.mean(likelihood)
print(f'Integral = {int_likelihood:.4}')
```

```
Integral = 3.068e-06
```

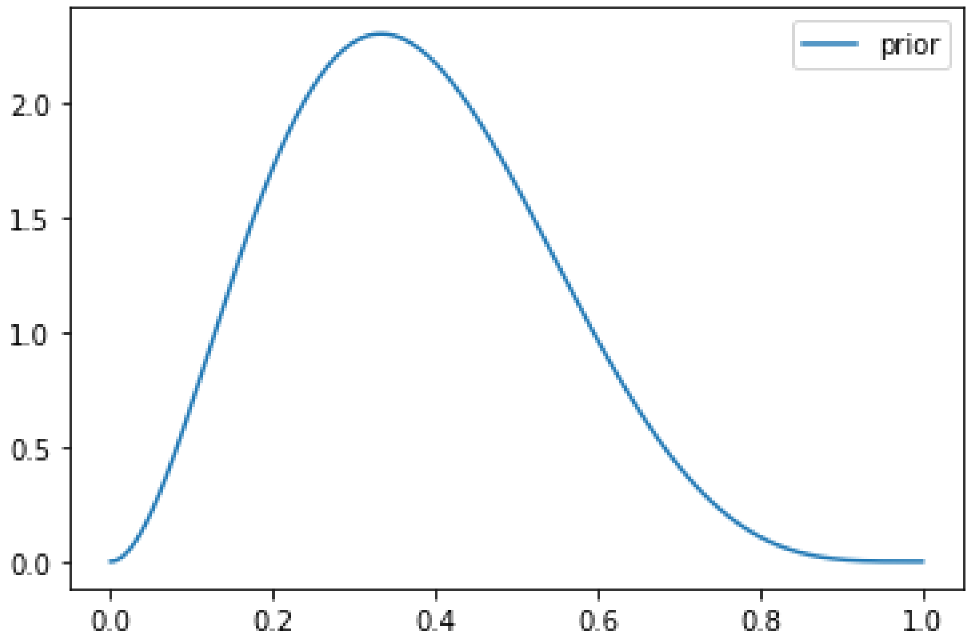
2.2 Task 2: Compute $\log p(\theta \mid a, b)$ for different values of θ

The function `loggamma` from the `scipy.special` package might be useful here. (It's already imported - see the first cell)

[ ]:
```
def compute_log_prior(theta, a, b):
    """Compute log p(theta | a, b) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-prior.
    a, b: float
        Parameters of the prior Beta distribution.

    Returns
    -------
    log_prior : array, shape (num_points)
        Values of log-prior for each value in theta.

    """

    return loggamma(a+b)-loggamma(a)-loggamma(b)+np.
 ↪log(theta**(a-1)*(1-theta)**(b-1))
```

[ ]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)
a, b = 3, 5

# Plot the prior distribution
log_prior = compute_log_prior(x, a, b)
prior = np.exp(log_prior)
plt.plot(x, prior, label='prior')
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f1218243d30>
```



Unlike the likelihood, the prior defines a probability distribution over θ and integrates to 1.

[ ]:
```
int_prior = 1.0 * np.mean(prior)
print(f'Integral = {int_prior:.4}')
```

```
Integral = 0.999
```

2.3 Task 3: Compute $\log p(\theta \mid \mathcal{D}, a, b)$ for different values of θ

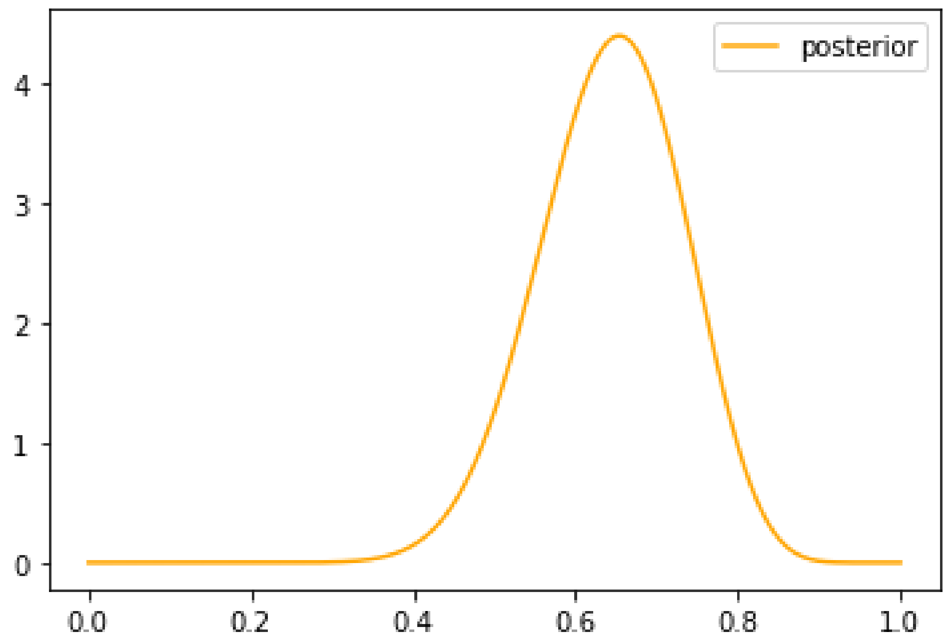
The function `loggamma` from the `scipy.special` package might be useful here.

[ ]:
```
def compute_log_posterior(theta, samples, a, b):
    """Compute log p(theta | D, a, b) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-prior.
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    a, b: float
        Parameters of the prior Beta distribution.

    Returns
    -------
    log_posterior : array, shape (num_points)
        Values of log-posterior for each value in theta.
```

```
    """

    tails = np.count_nonzero(samples == 1)
    heads = np.count_nonzero(samples == 0)

    return loggamma(tails+a+heads+b)-loggamma(tails+a)-loggamma(heads+b)+np.
 ↪log(theta**(tails+a-1)*(1-theta)**(heads+b-1))
```

[ ]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)

log_posterior = compute_log_posterior(x, samples, a, b)
posterior = np.exp(log_posterior)
plt.plot(x, posterior, label='posterior', c='orange')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x7f11e7fbe2b0>



Like the prior, the posterior defines a probability distribution over θ and integrates to 1.

[ ]:
```
int_posterior = 1.0 * np.mean(posterior)
print(f'Integral = {int_posterior:.4}')
```

```
Integral = 0.999
```

2.4 Task 4: Compute θ_{MLE}

[ ]:
```python
def compute_theta_mle(samples):
    """Compute theta_MLE for the given data.

    Parameters
    ----------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.

    Returns
    -------
    theta_mle : float
        Maximum likelihood estimate of theta.
    """

    tails = abs(np.count_nonzero(samples == 1))
    heads = abs(np.count_nonzero(samples == 0))

    return tails/(tails + heads)
```

[ ]:
```python
theta_mle = compute_theta_mle(samples)
print(f'theta_mle = {theta_mle:.3f}')
```

```
theta_mle = 0.750
```

2.5 Task 5: Compute θ_{MAP}

[ ]:
```python
def compute_theta_map(samples, a, b):
    """Compute theta_MAP for the given data.

    Parameters
    ----------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    a, b: float
        Parameters of the prior Beta distribution.

    Returns
    -------
    theta_mle : float
        Maximum a posteriori estimate of theta.
    """

    tails = abs(np.count_nonzero(samples == 1))
    heads = abs(np.count_nonzero(samples == 0))

    return (tails+a-1)/(heads+tails+a+b-2)
```

[ ]:
```python
theta_map = compute_theta_map(samples, a, b)
print(f'theta_map = {theta_map:.3f}')
```

```
theta_map = 0.654
```

3 Putting everything together

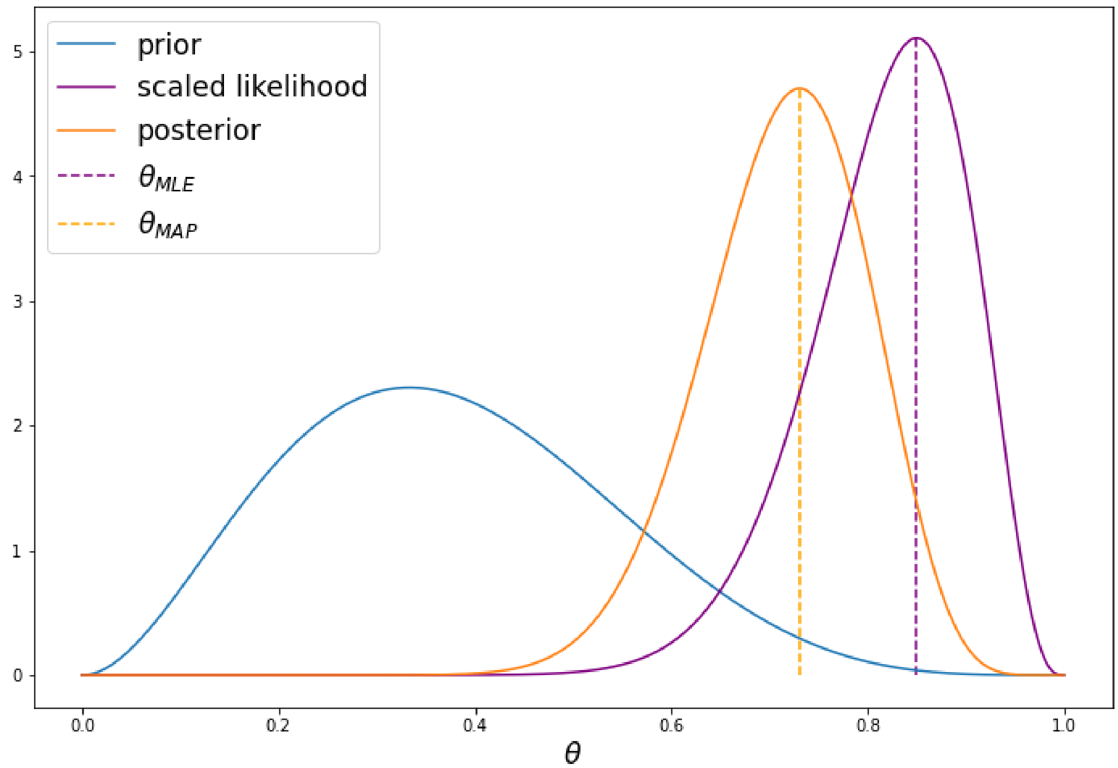
Now you can play around with the values of `a`, `b`, `num_samples` and `tails_proba` to see how the results are changing.

[ ]:
```python
num_samples = 20
tails_proba = 0.7
samples = simulate_data(num_samples, tails_proba)
a, b = 3, 5
print(samples)
```

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

[ ]:
```python
plt.figure(figsize=[12, 8])
x = np.linspace(1e-5, 1-1e-5, 1000)

# Plot the prior distribution
log_prior = compute_log_prior(x, a, b)
prior = np.exp(log_prior)
plt.plot(x, prior, label='prior')

# Plot the likelihood
log_likelihood = compute_log_likelihood(x, samples)
likelihood = np.exp(log_likelihood)
int_likelihood = np.mean(likelihood)
# We rescale the likelihood - otherwise it would be impossible to see in the␣
 ↪plot
rescaled_likelihood = likelihood / int_likelihood
plt.plot(x, rescaled_likelihood, label='scaled likelihood', color='purple')

# Plot the posterior distribution
log_posterior = compute_log_posterior(x, samples, a, b)
posterior = np.exp(log_posterior)
plt.plot(x, posterior, label='posterior')

# Visualize theta_mle
theta_mle = compute_theta_mle(samples)
ymax = np.exp(compute_log_likelihood(np.array([theta_mle]), samples)) /␣
 ↪int_likelihood
plt.vlines(x=theta_mle, ymin=0.00, ymax=ymax, linestyle='dashed',␣
 ↪color='purple', label=r'$\theta_{MLE}$')
```

```
# Visualize theta_map
theta_map = compute_theta_map(samples, a, b)
ymax = np.exp(compute_log_posterior(np.array([theta_map]), samples, a, b))
plt.vlines(x=theta_map, ymin=0.00, ymax=ymax, linestyle='dashed',␣
↪color='orange', label=r'$\theta_{MAP}$')

plt.xlabel(r'$\theta$', fontsize='xx-large')
plt.legend(fontsize='xx-large')
plt.show()
```




```
[ ]:
```