Problem 6:

$$\frac{d(\theta^t(1-\theta)^h)}{d\theta} = t\theta^{t-1}\cdot(1-\theta)^h - \theta^t\cdot h\cdot(1-\theta)^{h-1}$$

$$= \Big((1-\theta)\cdot t - h\theta\Big)\cdot\theta^{t-1}\cdot(1-\theta)^{h-1}$$

$$\frac{d^2(\theta^t(1-\theta)^h)}{d\theta^2} = \frac{d[((1-\theta)\cdot t-\theta h)\cdot\theta^{t-1}\cdot(1-\theta)^{h-1}]}{d\theta}$$

$$= (-t-h)\cdot\theta^{t-1}\cdot(1-\theta)^{h-1} + \Big((1-\theta)\cdot t-\theta h\Big)\cdot\Big[(t-1)\theta^{t-2}(1-\theta)^{h-1}$$

$$-\theta^{t-1}\cdot(h-1)\cdot(1-\theta)^{h-2}\Big]$$

$$=\Big[\big[(1-\theta)\cdot t-\theta h\big](t-1)\cdot\theta^{t-2}-(t+h)\cdot\theta^{t-1}\Big]\cdot(1-\theta)^{h-1} -$$

$$\Big((1-\theta)t-\theta h\Big)\cdot\theta^{t-1}(h-1)\cdot(1-\theta)^{h-2}$$

$\log\theta^t(1-\theta)^h$:

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

$$= \frac{t}{\theta}+\frac{h}{\theta-1}$$

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

Problem 7: for θ_1 is the local maximum of $\log f(\theta)$

$\log f(\theta_1) > \log f(\theta_2)$

$\log f(\theta_1) > \log f(\theta_3)$ for arbitary θ_2 and θ_3 $\theta_2 < \theta_1$, $\theta_3 > \theta_1$, θ_3 and θ_2 is close to θ_1

log is monotonic function

$\Rightarrow$ $f(\theta_1) > f(\theta_2)$

$f(\theta_1) > f(\theta_3)$

so we can also say θ_n is the local maximum of $f(\theta)$

we can conclude that we can use log to compute the local minimum or maximum θ, the result remains the same as using without log but the computation is smaller.

Problem 8:

Prior: $p(\theta|a,b) = Beta(a,b) \propto \theta^{a-1}(1-\theta)^{b-1}$ $\quad E(\theta|a,b) = \frac{a}{a+b}$

likelihood: $p(X=m|N,\theta) \propto \theta^m(1-\theta)^{N-m}$

$$p(\theta|D) = \frac{p(D|\theta)\cdot p(\theta)}{p(D)} \propto p(D|\theta)\cdot p(\theta) \propto \theta^m\cdot(1-\theta)^{N-m}\cdot\theta^{a-1}(1-\theta)^{b-1}$$

$$= \theta^{m+a-1}\cdot(1-\theta)^{N-m+b-1}$$

$$\propto Beta(m+a, N-m+b)$$

$$E(\theta|D) = \frac{m+a}{m+a+N-m+b}$$

$$= \frac{m}{N+a+b} + \frac{a}{N+a+b}$$

$$MLE: \frac{d \log \theta^m(1-\theta)^{N-m}}{d\theta} = \frac{m}{\theta} - \frac{N-m}{1-\theta} = 0$$

$$m(1-\theta) = (N-m)\cdot\theta$$

$$m - m\theta = (N-m)\theta$$

$$N\theta = m$$

$$\theta_{MLE} = \frac{m}{N}$$

$$E(\theta|D) = (1-\lambda)\cdot\frac{m}{N} + \lambda\cdot\frac{a}{a+b}$$

$$E(\theta|D) = \frac{N}{N+a+b}\cdot\frac{m}{N} + \frac{a+b}{N+a+b}\cdot\frac{a}{a+b}$$

$$\lambda = \frac{a+b}{N+a+b} \quad 1-\lambda = \frac{N}{N+a+b}$$

Problem 4:

$$\lambda_{MAP} = \underset{\lambda}{\arg\max}\, P(\lambda \mid x, a, b)$$

$$= \underset{\lambda}{\arg\max}\, \log\left(P(x \mid \lambda) \cdot P(\lambda \mid a, b)\right)$$

$$= \underset{\lambda}{\arg\max}\, \log(\lambda^x e^{-\lambda}) - \log x! + \log\frac{b^a}{\Gamma(a)} + \log \lambda^{a-1} + \log e^{-b\lambda}$$

$$= \underset{\lambda}{\arg\max}\left(x \cdot \log \lambda - \lambda + \log\left(\frac{b^a}{\Gamma(a)}\right) + (a-1) \cdot \log \lambda - b\lambda - \log x!\right)$$

$$\frac{d\left(x \cdot \log \lambda - \lambda + \log\frac{b^a}{\Gamma(a)} + (a-1) \cdot \log \lambda - b\lambda - \log(x!)\right)}{d\lambda} = 0$$

$$\frac{x}{\lambda} - 1 + (a-1) \cdot \frac{1}{\lambda} - b = 0$$

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

$$\lambda_{MAP} = \frac{a-1+x}{b+1}$$

Programming Task: Probabilistic Inference

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

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

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).

Simulating data

The following function simulates flipping a biased coin.

In [2]:
```
# 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_p
```

In [3]:
```
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]
```

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 $a \cdot b$.

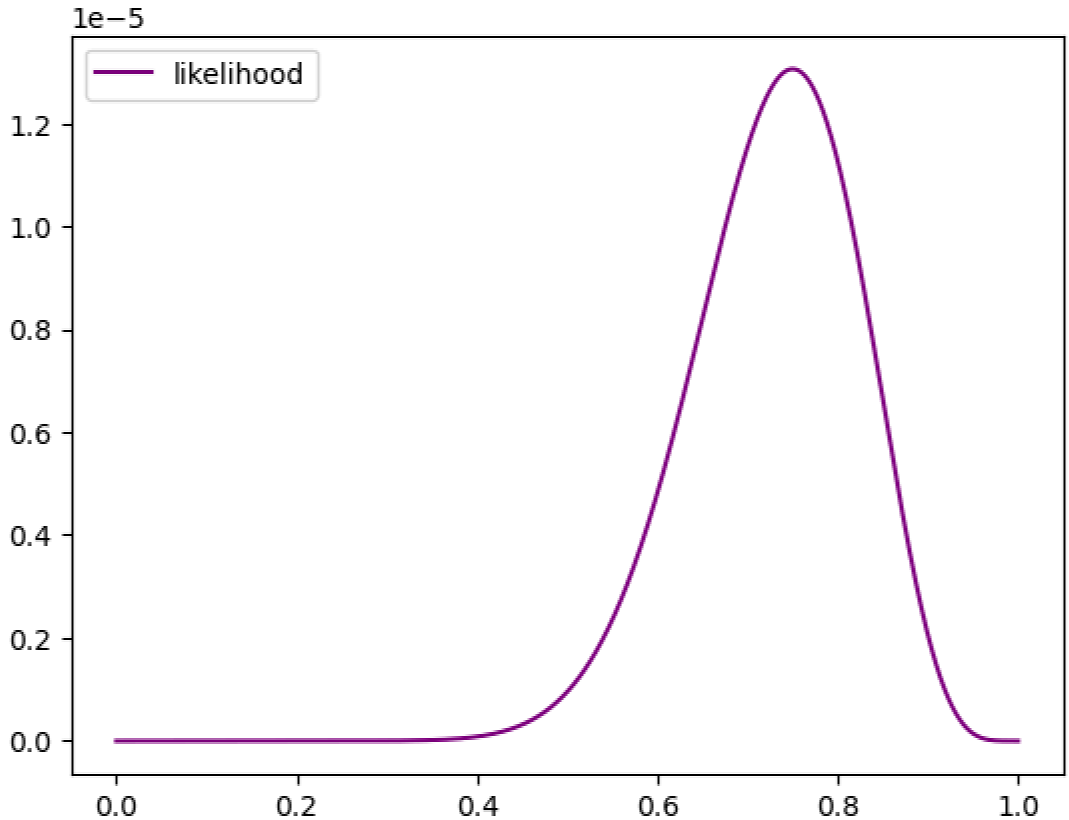
For this reason, we usually compute **log-probabilities** instead of **probabilities**. Virtually all machine learning libraries are dealing with log-probabilities instead of probabilities (e.g. Tensorflow-probability or Pyro).

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

In [4]:
```
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.
    """
    ### YOUR CODE HERE ###
    num_heads, num_tails = np.bincount(samples)
    y = num_heads * np.log(1 - theta) + num_tails * np.log(theta)
    return y
```

In [5]:
```
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()
```

Out[5]:
```
<matplotlib.legend.Legend at 0x1dde6925730>
```



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.

In [6]:
```
# 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
```

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

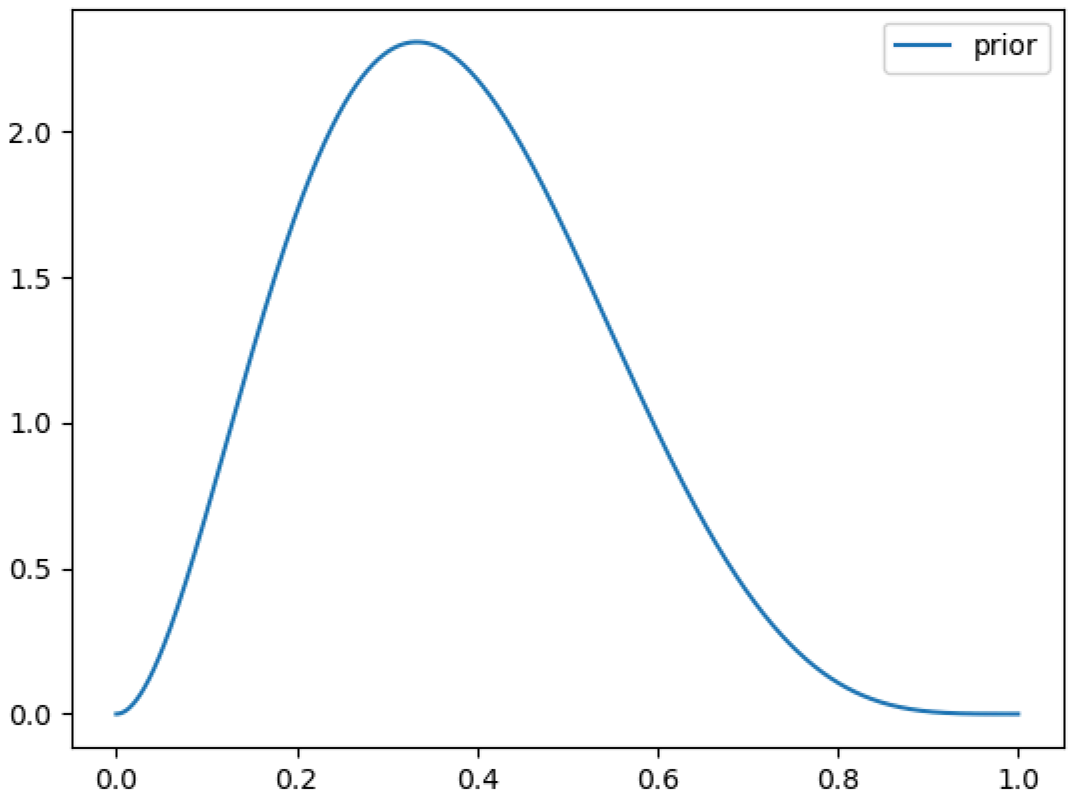
The function `loggamma` from the `scipy.special` package might be useful here. (It's already imported - see the first cell)

In [7]:
```
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.

    """
    ### YOUR CODE HERE ###
    y = loggamma(a + b) - loggamma(a) - loggamma(b) + (a - 1) * np.log(theta) + (b
    return y
```

In [8]:
```
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()
```

Out[8]: <matplotlib.legend.Legend at 0x1dde6a33670>



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

In [9]:
```
int_prior = 1.0 * np.mean(prior)
print(f'Integral = {int_prior:.4}')
```

```
Integral = 0.999
```

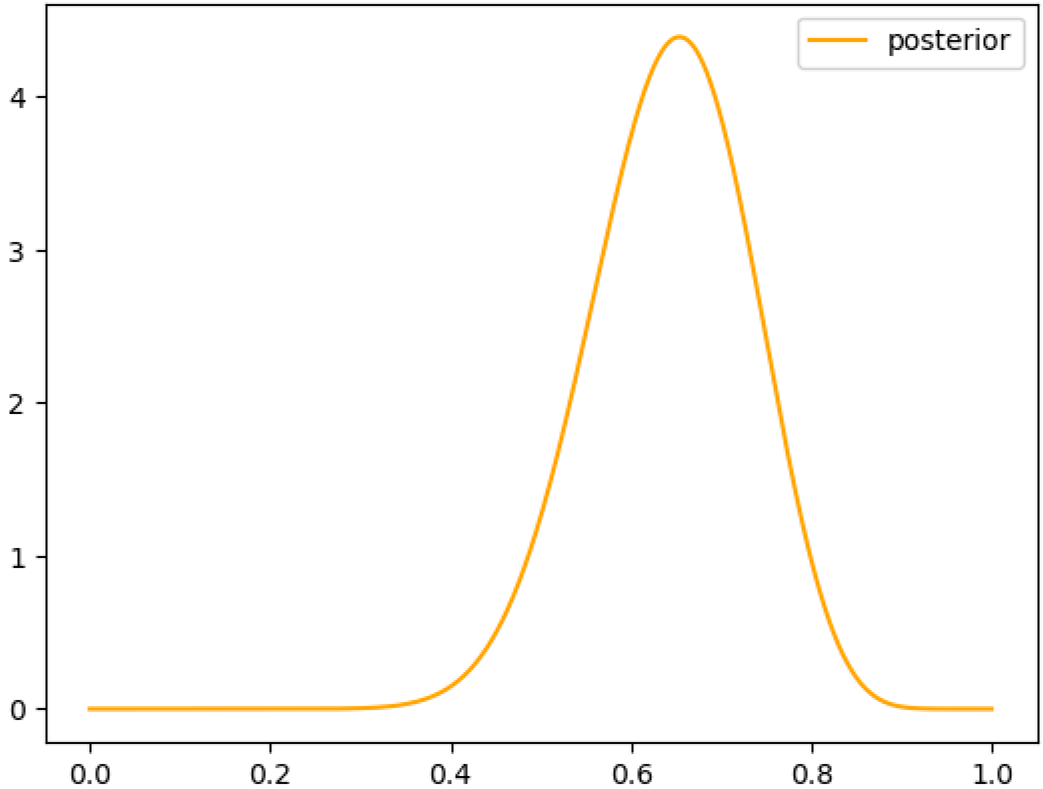
Task 3: Compute $\log p(\theta \mid \mathcal{D}, a, b)$ for different values of θ

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

In [10]:
```
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.
    """
    ### YOUR CODE HERE ###
    num_heads, num_tails = np.bincount(samples)
    y = loggamma(a + num_tails + b + num_heads) - loggamma(a + num_tails) - loggamm
    return y
```

In [11]:
```
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()
```

Out[11]:
```
<matplotlib.legend.Legend at 0x1dde409f130>
```



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

In [12]:
```
int_posterior = 1.0 * np.mean(posterior)
print(f'Integral = {int_posterior:.4}')
```

```
Integral = 0.999
```

Task 4: Compute θ_{MLE}

In [13]:
```
import numpy as np
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.
    """
    ### YOUR CODE HERE ###
    num_heads, num_tails = np.bincount(samples)
    return num_tails / (num_heads + num_tails)
```

In [14]:
```
theta_mle = compute_theta_mle(samples)
```

```
print(f'theta_mle = {theta_mle:.3f}')
```

```
theta_mle = 0.750
```

Task 5: Compute θ_{MAP}

In [15]:
```
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.
    """
    ### YOUR CODE HERE ###
    num_heads, num_tails = np.bincount(samples)
    return (num_tails + a - 1) / (num_heads + num_tails + a + b - 2)
```

In [16]:
```
theta_map = compute_theta_map(samples, a, b)
print(f'theta_map = {theta_map:.3f}')
```

```
theta_map = 0.654
```

Putting everything together

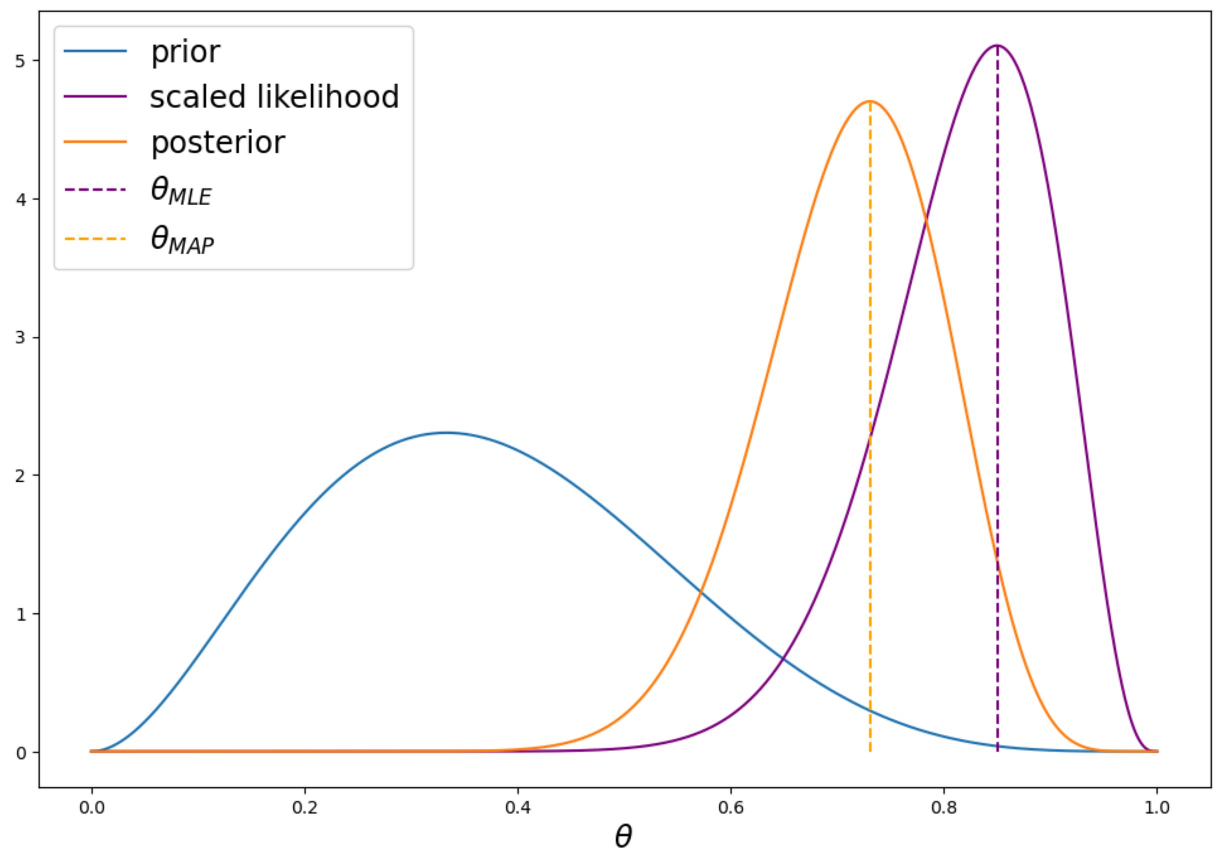
Now you can play around with the values of `a`, `b`, `num_samples` and `tails_proba` to see how the results are changing.

In [17]:
```
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]
```

In [18]:
```
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)
```

```python
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_likelih
plt.vlines(x=theta_mle, ymin=0.00, ymax=ymax, linestyle='dashed', color='purple', l


# 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', l

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



In [ ]: