Machine Learning Homework 03

Author: Jesus Arturo Sol Navarro

1 Problem 6

Compute the first and second derivative of this likelihood w.r.t. θ . Then compute first and second derivative of the log-likelihood $\log \theta^t (1-\theta)^h$.

$f(\theta) = \theta^t (1-\theta)^h$

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

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

$\frac{dlog(f)}{d\theta} = \frac{d}{d\theta}(tlog(\theta) + hlog(1-\theta)) = \frac{t}{\theta} - \frac{h}{1-\theta}$

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

2 Problem 7

Show that for any differentiable, positive function $f(\theta)$ every local maximum of $logf(\theta)$ is also a local maximum of $f(\theta)$. Considering this and the previous exercise, what is your conclusion?

Taking in consideration that $f(\theta) > 0$ for all θ, then $h(\theta) = log(f(\theta))$ is a well-defined, differentiable function.

Computing the first derivative and setting it to zero.

$\frac{dh(\theta)}{d\theta} = \frac{dlogf(\theta)}{d\theta} = \frac{f^{'}(\theta)}{f(\theta)} \stackrel{!}{=} 0$

It can be identified that $h^{'}(\theta) = 0$ if and only if $f^{'}(\theta) = 0$. Since the first derivative of both functions share the same zero points, $h(\theta) = log(f(\theta))$ has local maxima at the same points as $f(\theta)$.

Therefore, maximizing $log(f(\theta))$ is equivalent to maximizing $f(\theta)$, this simplifies greatly the process of finding MLE or MAP.

3 Problem 8

For the random variables:

$$\Theta \sim Beta(a,b) = \Theta^{a-1}(1-\Theta)^{b-1}$$
$$X \sim Binom(N,\Theta) = \Theta^m (1-\Theta)^{N-m}$$

Identification of the posterior distribution:

$$p(\Theta|X) = \frac{p(X|\Theta)\cdot p(\Theta)}{p(x)} \propto p(X|\Theta)\cdot p(\Theta) \propto \Theta^m(1-\Theta)^{N-m}\cdot\Theta^{a-1}(1-\Theta)^{b-1} \propto \Theta^{m+a-1}(1-\Theta)^{N-m+b-1}$$

because of the equation $N - m = l$ we get:

$$\theta^{m+a-1}(1-\Theta)^{l+b-1}$$

which is a Beta distribution for $\Theta|X \sim Beta(m+a, l+b)$ The posterior mean of Θ is:

$$\mathbf{E}[\Theta|D] = \frac{m+a}{m+a+l+b} = \frac{m}{m+a+l+b} + \frac{a}{m+a+l+b} = \frac{m}{m+l}\cdot\frac{m+l}{m+a+l+b} + \frac{a}{a+b}\cdot\frac{a+b}{m+a+l+b}$$

$\frac{m+l}{m+l+a+b} = 1-\lambda$ and $\frac{a+b}{m+a+l+b} = \lambda$
So $\frac{m}{m+l}$ is the maximum likelihood estimate and $\frac{a}{a+b}$ is the prior mean value of Θ

4 Problem 9

$$\lambda_{MAP} = \arg\max_{\lambda} p(\lambda|x,a,b) = \arg\max_{\lambda} \log p(\lambda|x,a,b) = \arg\max_{\lambda}(\log p(x|\lambda) + \log p(\lambda|a,b)) =$$

$$= \arg\max_{\lambda}(\log(\frac{\lambda^x exp(-\lambda)}{x!}) + log(\frac{b^a}{\Gamma(a)}\lambda^{a-1}exp(-b\lambda)) =$$

$$= \arg\max_{\lambda}(x\log(\lambda) - \lambda + \log(\frac{1}{x}) + \log(\frac{b^a}{\Gamma(a)}) + (a-1)\log(\lambda) - b\lambda) =$$

$$= \arg\max_{\lambda}((x+a-1)\log(\lambda) - (1+b)\lambda)$$

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

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

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

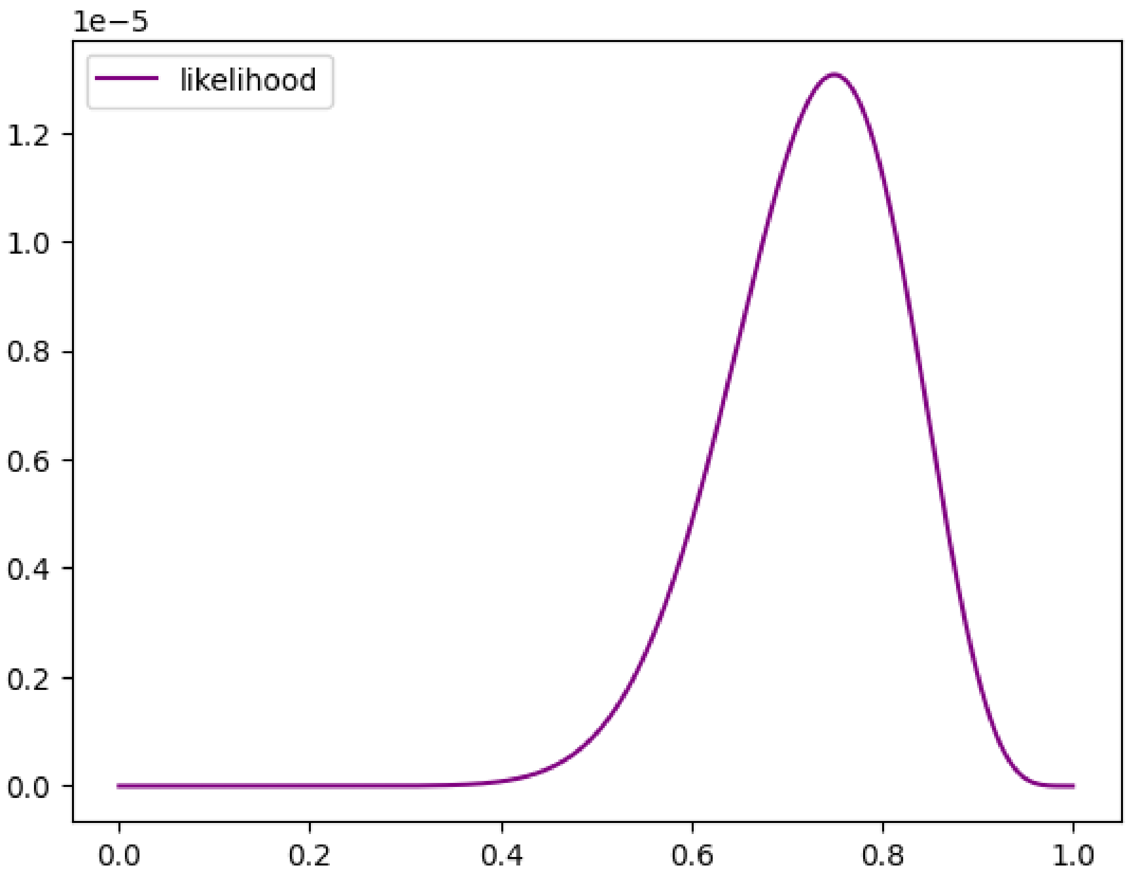
5 Problem 10

5.1 Simulating data

```
    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)
```

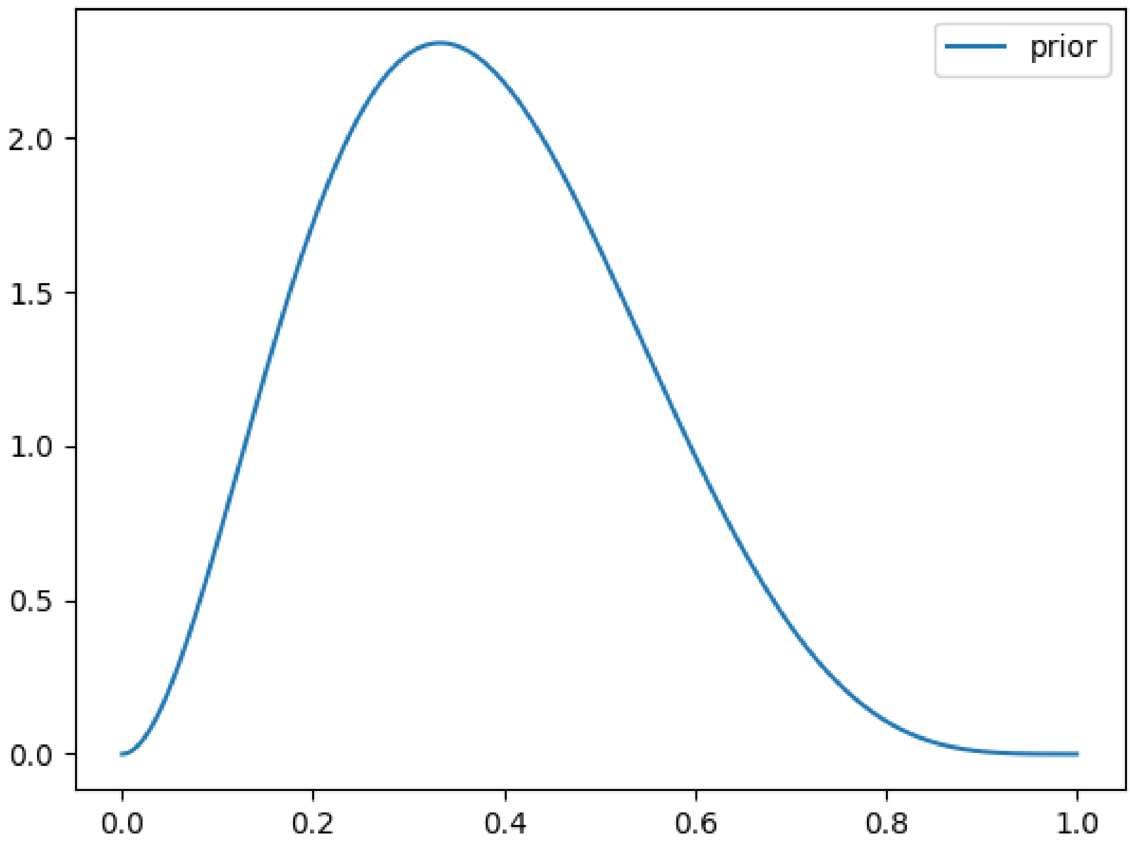
5.2 Compute $\log p(\mathcal{D} \mid \theta)$ for different values of θ

```
    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.
    """

    number_of_tails = np.count_nonzero(samples==1)
    number_of_heads = len(samples) - number_of_tails
    likelihood = np.power(theta,number_of_tails) * (np.power((1-theta),number_of_heads))

    log_likelihood1 = number_of_tails*(np.log(theta))
    log_likelihood2 = number_of_heads*(np.log(1-theta))
    log_likelihood = log_likelihood1+log_likelihood2

    return log_likelihood

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()

int_likelihood = 1.0 * np.mean(likelihood)
print(f'Integral = {int_likelihood:.4}')
```



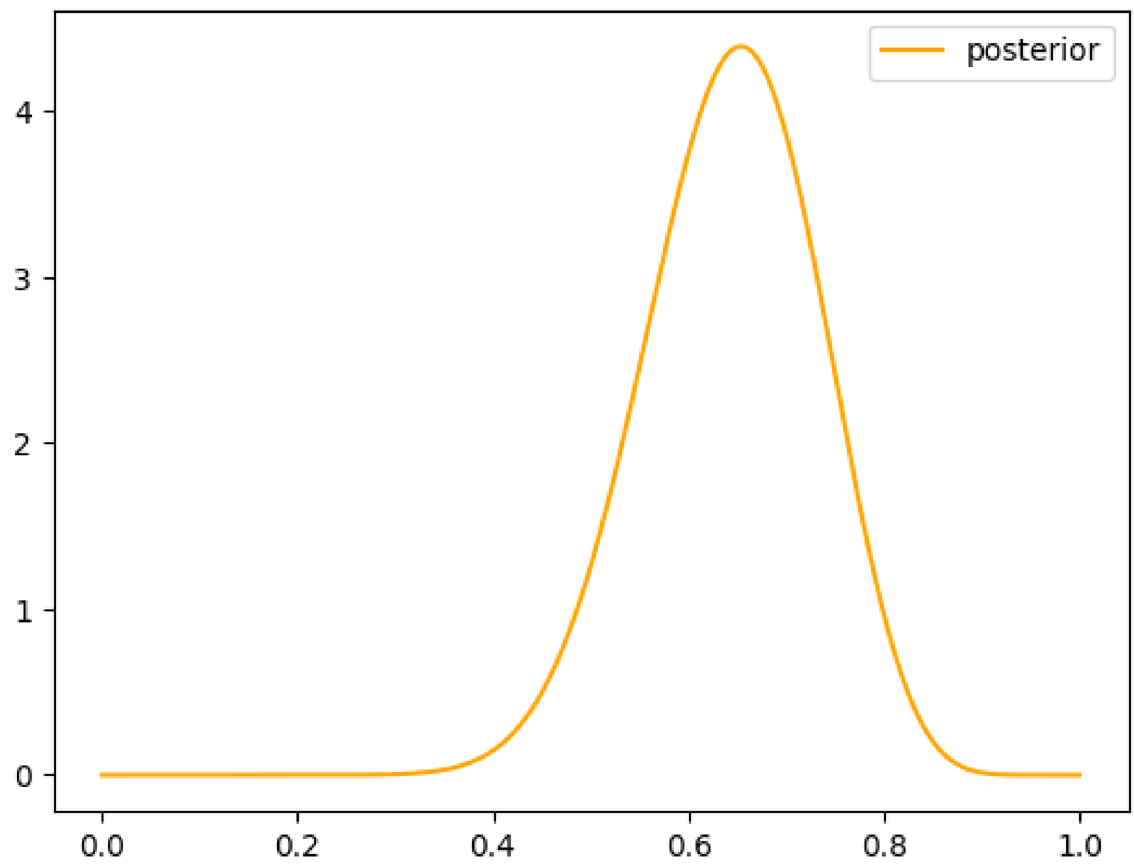
5.3 Compute $\log p(\theta \mid a, b)$ for different values of θ

```
    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.

    """
    m= (a-1)*np.log(theta)
    n= (b-1)*np.log(1-theta)
    o= loggamma(a)+loggamma(b)-loggamma(a+b)
    logbeta = m+n-o

    return logbeta

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()

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



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

```
    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.
        """
        number_of_tails = np.count_nonzero(samples==1)
        number_of_heads = len(samples) - number_of_tails
        log_posterior=compute_log_prior(theta, a+number_of_tails, b+number_of_heads)

        return log_posterior

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()

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



5.5 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.
    """
    number_of_tails = np.count_nonzero(samples==1)
    number_of_heads = len(samples) - number_of_tails

    theta_mle= number_of_tails / ((number_of_tails+number_of_heads))
    return theta_mle

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

theta_mle = 0.750

5.6 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.
    """
    number_of_tails = np.count_nonzero(samples==1)
    number_of_heads = len(samples) - number_of_tails

    theta_map = (number_of_tails+a+1)/(number_of_tails-2+number_of_heads+a+b)
    return theta_map

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

theta_map = 0.731

5.7 Putting everything together

```
num_samples = 20
tails_proba = 0.7
samples = simulate_data(num_samples, tails_proba)
a, b = 5, 5
print(samples)

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()
```