PML Theory Questions:

Q 1: (a) If we consider the product of the likelihood & prior:

$$p(x|c) = \frac{p(c|x)\,p(x)}{p(c)} \propto p(c|x)\,p(x)$$

$$= \left(\frac{N!}{\prod_{\ell}^{K} n_\ell!} \prod_{k=1}^{K} x_k^{n_k}\right) \cdot \left(\frac{\Gamma(\sum \alpha_k)}{\prod \Gamma(\alpha_k)} \prod_k x_k^{\alpha_k - 1}\right)$$

$$\propto \prod_k x_k^{n_k + \alpha_k - 1}$$

which we recognize as the functional form of the dirichlet$(\underline{x}|\underline{\alpha}+\underline{n})$ whence Dirichlet is conjugate prior of the multinomial
(i.e. $\underline{X}|\underline{c} \sim \mathrm{Dir}(-, \underline{\alpha}+\underline{n})$ with $\underline{n} := (n_1, \dots, n_K)$)

(b) WLOG let $i=1$, $j=2$

we aggregate by marginalizing out $j=2$ & summing $x_1 + x_2$:

So let $y := x_1 + x_2$. Then the pdf is given by:

$$~~p(x_1+x~~ \; p(y, x_3, x_4, \dots, x_K) = \int_0^y \frac{\Gamma(\sum \alpha_k)}{\prod \Gamma(\alpha_k)} \, \xi^{\alpha_1 - 1} (y-\xi)^{\alpha_2 - 1} \left(\prod_{j=3}^{K} x_j^{\alpha_j - 1}\right) d\xi$$

$$\propto \underbrace{\left(\int_0^y \xi^{\alpha_1 - 1} (y-\xi)^{\alpha_2 - 1} \, d\xi\right)}_{\overset{!!}{I}} \prod_{k=3}^{K} x_k^{\alpha_k - 1}$$

the above integral is evaluated using the substitution $Q := \frac{\xi}{y}$

$$\Rightarrow I = \int_0^1 y^{\alpha_1 + \alpha_2 - 1} Q^{\alpha_1 - 1} (1-Q)^{\alpha_2 - 1} \, dQ$$

which we recognize as the beta integral $\frac{\Gamma(\alpha_1+\alpha_2)}{\Gamma(\alpha_1)\Gamma(\alpha_2)}$

$$\Rightarrow p(x_1 + x_2, x_3, \dots, x_K) \propto (x_1 + x_2)^{\alpha_1 + \alpha_2 - 1} x_3^{\alpha_3 - 1} \cdots x_K^{\alpha_K - 1}$$

$$\Rightarrow (X_1 + X_2, X_3, \dots, X_K) \sim \mathrm{Dir}(\alpha_1 + \alpha_2, \alpha_3, \dots, \alpha_K)$$

•) The cross entropy loss is given by : (part (c))

$$CE = \underset{\underline{x}}{\text{argmax}} \log p(C|\underline{x}) = \underset{\underline{x}}{\text{argmax}} \log \text{Likelihood}$$

we have

$$\log p(C|\underline{x}) = \log N! + \sum_{k=1}^{K} [\log(x_k)] n_k - \sum_{k=1}^{K} \log n_k$$

in order to find the argmax, one must employ Lagrange multipliers and optimize $\log p(C|x) + \lambda\left(1 - \sum_{k=1}^{K} x_k\right) =: \mathcal{F}(\underline{x},\lambda)$

then $\partial_{x_k} \mathcal{F} = \frac{n_k}{x_k} - \lambda \quad \forall k \in \{1, .., K\}$

optimizing $(\partial_{x_k}\mathcal{F} = 0) \Rightarrow x_k = \frac{n_k}{\lambda} \quad \forall k$

$$\Rightarrow \sum x_k = \frac{\sum n_k}{\lambda} \;\forall k \Rightarrow \lambda = \frac{\sum n_k}{\sum x_k} = \sum n_k = N$$

thus $\underline{x}_{MLE} = \frac{1}{N}(n_1, ..., n_K)$

•) Our posterior is $\mathcal{D}ir(-, \underline{\alpha} + \underline{n})$

so $p(\underline{x}|C) = \frac{\Gamma(\sum \alpha_k + n_k)}{\prod \Gamma(\alpha_k + n_k)} \prod x_k^{\alpha_k + n_k - 1}$

we use the same (Lagrange multiplier) trick to solve the max

$$\log p(\underline{x}|C) = \log\Gamma\left(\sum \alpha_k + n_k\right) - \sum \log\Gamma(\alpha_k + n_k) + \sum (\alpha_k + n_k - 1)\log x_k$$

$$\mathcal{G}(\underline{x},\lambda) := \log p(\underline{x}|C) + \lambda\left(1 - \sum x_k\right)$$

$\partial_{x_k}\mathcal{G} = \frac{\alpha_k + n_k - 1}{x_k} - \lambda$, Setting equal to zero gives

$x_k = \frac{\alpha_k + n_k - 1}{\lambda}$. By the same logic as above; $\longrightarrow$

$\lambda = \sum_i \alpha_k + \sum_i n_k - K$ (summing k expressions)

where the max is given by

$$\hat{x}_k = \frac{\alpha_k + n_k - 1}{\sum_i \alpha_h + n_h - K} \quad \forall k \in \{1, \dots, K\}$$

The two forms are very similar but the posterior takes into account the prior information (parametrized by $\underline{\alpha}$)

(d) (i) we have $MLE(\underline{x} \mid C_1) = (0,0,0,0,1)$

$MLE(\underline{x} \mid C_2) = \frac{1}{410}(1,0,12,43,354)$

(ii) $p(\underline{x} \mid C_1, \underline{\alpha}) = D(\underline{x} \mid (1,1,1,1,4))$

$p(\underline{x} \mid C_2, \underline{\alpha}) = D\left(\underline{x} \mid \frac{2}{415}, \frac{1}{415}, \frac{13}{415}, \frac{44}{415}, \frac{355}{415}\right)$

~~and~~ we have $p(C_{N+1,i} = (0,0,0,0,1) \mid \underline{x}, C, \underline{\alpha}) = x_5 \quad \forall i \in \{1,2\}$

(likelihood of single event is just probability of that event)

$$\Rightarrow p((0,0,0,0,1) \mid C, \alpha, \underline{x}) = \int_{[0,1]^5} p(C_{N+1} \mid \underline{x})\, p(\underline{x} \mid C, \alpha, \underline{x})\, d\underline{x}$$

$$= \int_{[0,1]^5} x_5 \, D(\underline{x} \mid \underline{\alpha} + \underline{n})\, d\underline{x} = \mathbb{E}_{x \sim D(\underline{n} + \underline{\alpha})}(x_5)$$

= if we apply the aggregation property to indices 1-4 of $\underline{x} = (x_1, \dots, x_5)$

we get $(x_1 + x_2 + x_3 + x_4, x_5) \sim Dir(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4, \alpha_5) = Beta\left(\sum_1^4 \alpha_j, \alpha_5\right)$

our probability then reduces to $\frac{\alpha_5}{\sum_1^5 \alpha_j}$ (Beta integral has ~~easy~~ known solution)
(and for 4-stars, $\frac{\alpha_4}{\sum^5 \alpha_j}$)

Thus for C_1, we have:

$$p(C_{N+1} = 5\bigstar s \mid \text{data, prior}) = \frac{4}{8} = \frac{1}{2}$$

$$p(C_{N+1} \in \{4\bigstar s, 5\bigstar s\} \mid \text{info}) = \frac{1}{2} + \frac{1}{8} = \frac{5}{8}$$

and for C_2:

$$p(C_{N+1} = 5\bigstar s \mid \text{info}) = \frac{355}{415}$$

$$p(C_N \in \{4\bigstar s, 5\bigstar s \mid \text{info}) = \frac{355}{415} + \frac{44}{415} = \frac{399}{415}$$

So C_2 seems a better choice with a reasonable prior!

Ex02

April 30, 2023

1 Probabilistic Machine Learning

University of Tübingen, Summer Term 2023 © 2023 P. Hennig

1.1 Exercise Sheet No. 2 — Laplace Approximations

Submission by: * Sam, Laing : 6283670 * Albert Catalan Tatjer : 6443478

[4]:
```
from io import StringIO

import pandas as pd
import requests

import jax
from jax import numpy as jnp
from jax.scipy import optimize

from matplotlib import pyplot as plt
import matplotlib.tri as tri
from matplotlib import ticker
import numpy as np

from tueplots import bundles
from tueplots.constants.color import rgb


plt.rcParams.update(bundles.beamer_moml())
plt.rcParams.update({"figure.dpi": 200})
```

2 Exercise 2.2 (Coding Exercise)

In this exercise we are going to practice the Laplace approximation, as well as `jax`. You can use the functionality from `jax` whereever you want to. Your tasks are the following:

Task 1. Implement the Beta distribution:

$$p_z(z) = Beta(z; a, b)$$

You can do it yourself, or use `jax.scipy.stats.beta.pdf`.

[5]:
```
from jax.scipy.stats import beta

def p_z(z, a, b):
    """Beta distribution p_z(z).

    Args:
      z: Float, Argument of the beta distribution.
      a: Float, Parameter of the beta distribution.
      b: Float, Parameter of the beta distribution.

    Returns:
      Value of the probability density function at z.
    """
    # TODO
    return beta.pdf(z,a,b)
```

[6]:
```
# some useful imports for clearer expressions
from jax.scipy.special import logit
from jax.scipy.special import expit
from jax.numpy import exp
```

Task 2. What is the distribution $p_x(x)$ of x if

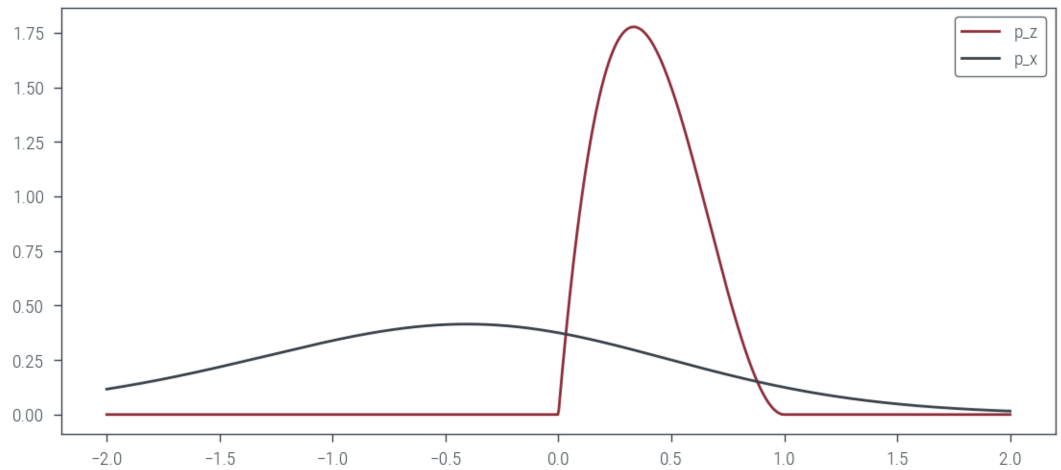
$$z = logisitc(x) \qquad \text{with} \qquad logistic(x) = 1/(1 + exp(-x)?$$

Implement it using the transformation rules from the lecture. `jax.jacrev` might be helpful for calculating Jacobians.

[30]:
```
def p_x(x, a, b):
    """Probability density function for x with z=logistic(x).

    Args:
      z: Float, Argument of p_x.
      a: Float, Parameter of the beta distribution of z.
      b: Float, Parameter of the beta distribution of z.

    Returns:
      Value of the probability density function p_x(x) at x.
    """
    # TODO
    return p_z(expit(x),a,b) * (expit(x)*(1-expit(x)) )
```

Checking out the plot:

[36]:
```
x = np.linspace(-2,2,1000)
z = p_z(x,2,3)
plt.plot(x,z, label = "p_z")

X = p_x(x,2,3)
plt.plot(x,X, label = "p_x")
plt.legend()
plt.show()
```



Task 3. Compute the Laplace approximations for both, $p_z(z)$ and $p_x(x)$.

From slide 33 of lecture 3, we have the Laplace approximation of $p_z(z)$
We have to compute $p_x(x)$ by hand using the expression from above

[11]:
```
import scipy.optimize as so
```

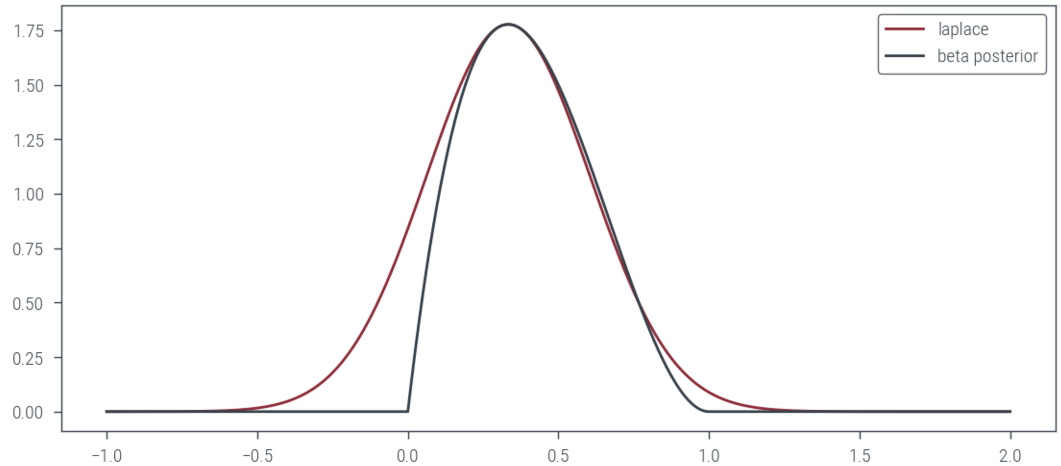
[105]:
```
# functions related to Z
def z_hat(a, b):
    return (a - 1) / (a + b - 2)


def psi_z(a, b):
    return -(1 / (a - 1) + 1 / (b - 1)) * (a + b - 2) ** 2


def laplace_z(a, b):
    """Laplace approximation for the beta distribution.

    Args:
      a: Float, Parameter of the beta distribution.
      b: Float, Parameter of the beta distribution.

```

```python
    Returns:
      A function with the same argument as the beta distribution.
    """

    # TODO
    def l_z(z):
        return p_z(z_hat(a, b), a, b) * exp(0.5 * psi_z(a, b) * (z - z_hat(a, b)) ** 2)

    return l_z


# functions related to X
def x_hat(a, b):
    return logit(a / (a + b))


def psi_x(a, b):
    t = x_hat(a, b)
    return -(a + b) * expit(t) * (1 - expit(t))


def laplace_x(a, b):
    """Laplace approximation for p_x with z=logistic(x).

    Args:
      a: Float, Parameter of the beta distribution.
      b: Float, Parameter of the beta distribution.

    Returns:
      A function with the same argument as p_x.
    """

    # TODO
    def l_x(x):
        return p_x(x_hat(a, b), a, b) * exp(0.5 * psi_x(a, b) * (x - x_hat(a, b)) ** 2)

    return l_x
```

Task 4. Make a plot for $p_z(z)$ and it's Laplace approximation for the parameter combinations $a = 2, b = 3$ and $a = 5, b = 5$. Are there parameter combinations, where the Laplace approximation is undefined? Make the same plot for x, too.

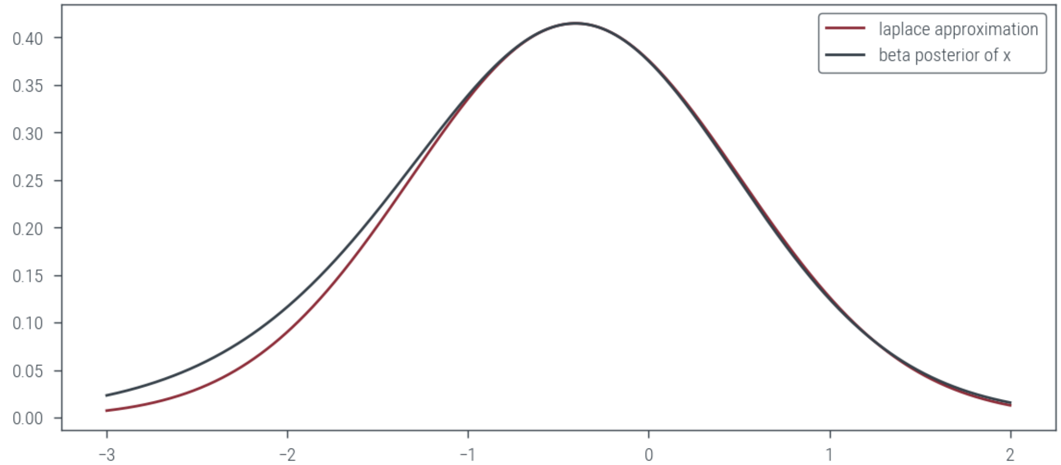
Plot of p_z with laplace approximation

```
[107]: z = np.linspace(-1,2, 1000)
lz = laplace_z(2,3)(z)
pz = p_z(z,2,3)
plt.plot(z,lz, label = "laplace")
plt.plot(z,pz, label = "beta posterior")
plt.legend()
plt.show()
```



Plot of p_x with laplace approximation

```
[108]: x = np.linspace(-3, 2, 1000)
lx = laplace_x(2, 3)(x)
px = p_x(x, 2, 3)
plt.plot(x, lx, label="laplace approximation")
plt.plot(x, px, label="beta posterior of x")
plt.legend()
plt.show()
```



Task 5. Implement the Dirichlet distribution

$$p_y(y) = Dirichlet(y; \alpha)$$

(alternative:`jax.scipy.stats.dirichlet.pdf`) and it's Laplace approximation.

[20]:
```python
from jax.scipy.stats import dirichlet
```

[96]:
```python
def p_y(y, alpha):
    """Dirichlet distribution p_y(y).

    Args:
      y: ArrayLike, Argument of the Dirichlet distribution.
      alpha: ArrayLike, Parameter of the Dirichlet distribution.

    Returns:
      Value of the probability density function at z.
    """
    # TODO
    return dirichlet.pdf(y, alpha)


def laplace_y(alpha):
    """Laplace approximation for the Dirichlet distribution p_y.

    Args:
      alpha: ArrayLike, Parameter of the Dirichlet distribution.

    Returns:
      A function with the same argument as p_y.
    """
    # TODO
    y_hat = alpha / alpha.sum()
    log_differential = -(alpha - 1) / (y_hat**2)

    def l_y(y):
        return p_y(y_hat, alpha) * exp(
            0.5 * (log_differential * (y - y_hat) ** 2).sum()
        )

    return l_y
```

Task 6. For $\alpha = (2, 10, 2)$ and $\alpha = (3, 2, 5)$, plot $p_y(y)$ and it's Laplace approximation next to each other. The function `simplex_contour_plot` implemented below can help with contour plots over the simplex. You can adapt it in any way you like.

[93]:
```
# TODO: Plot
def simplex_contour_plot(fun1, fun2):
    """Make contour plots for two functions, each defined over the probability␣
 ↪simplex
       represented by a triangualar surface.

    Args:
      fun1: function, defined over the probability simplex in three dimensions.
      fun2: function, defined over the probability simplex in three dimensions.

    Based on: https://blog.bogatron.net/blog/2014/02/02/
 ↪visualizing-dirichlet-distributions/
    """

    # Define the triangle
    corners = np.array([[0, 0], [1, 0], [0.5, 0.75**0.5]])
    area = 0.5 * 1 * 0.75**0.5
    triangle = tri.Triangulation(corners[:, 0], corners[:, 1])
    refiner = tri.UniformTriRefiner(triangle)
    trimesh = refiner.refine_triangulation(subdiv=8)

    # For each corner of the triangle, the pair of other corners
    pairs = [corners[np.roll(range(3), -i)[1:]] for i in range(3)]

    # The area of the triangle formed by point xy and another pair or points
    tri_area = lambda xy, pair: 0.5 * np.linalg.norm(np.cross(*(pair - xy)))

    # Convert cartesian to barycentric coordinates
    def xy2bc(xy, tol=1e-6):
        coords = np.array([tri_area(xy, p) for p in pairs]) / area
        return np.clip(coords, tol, 1.0 - tol)

    values1 = [fun1(xy2bc(xy)).item() for xy in zip(trimesh.x, trimesh.y)]
    values2 = [fun2(xy2bc(xy)).item() for xy in zip(trimesh.x, trimesh.y)]

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(5, 3))
    axes[0].tricontourf(trimesh, values1)
    axes[1].tricontourf(trimesh, values2)
    axes[0].axis("equal")
    axes[1].axis("equal")
    axes[0].axis("off")
    axes[1].axis("off")
    plt.show()
```

[94]:
```
alpha = jax.numpy.array([3, 2, 5])
simplex_contour_plot(lambda y: p_y(y, alpha=alpha), laplace_y(alpha))
```

[95]:
```
alpha = jax.numpy.array([2, 10, 2])
simplex_contour_plot(lambda y: p_y(y, alpha=alpha), laplace_y(alpha))
```

2.0.1 How to submit your work:

Export your answer into a pdf (for example using jupyter's `Save and Export Notebook as` feature in the `File` menu). Make sure to include all outputs, in particular plots. Also include your answer to the theory question, either by adding it as LaTeX code directly in the notebook, or by adding it as an extra page (e.g. a scan) to the pdf. Submit the exercise on Ilias, in the associated folder. **Do not forget to add your name(s) and matricel number(s) above!)**