Probabilistic Machine Learning

Submission by:

- Batuhan, Oezcoemlekci, Matrikelnummer: 6300476
- Aakarsh, Nair, Matrikelnummer: 6546577

Theory Question: In the lecture, we encountered the Beta distribution

$$p(x \mid a, b) = \mathcal{B}(x; a, b) := \frac{1}{B(a, b)} x^{a-1}(1-x)^{b-1}$$

where the normalization constant $B(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ is the Beta function, using the (Euler) Gamma function. The Beta is conjugate to the binomial distribution

$$p(n, m \mid x) = \binom{n+m}{n} x^n (1-x)^m$$

That is, the posterior on x arising from the Beta prior and the binomial likelihood is $\mathcal{B}(x; a+n, b+m)$. In this question, we will study the generalization to the multinomial case: Consider a data set $C = \left[c_1, c_2, \ldots, c_N\right]$ of discrete labels $c_i \in \{1, 2, 3, \ldots, K\}$. It is convenient to use the notation $n_k = \left|\left\{c_i \in C \mid c_i = k\right\}\right|$ for the number of labels observed in class k. We assume these data are drawn iid. from the multinomial distribution $$

$$p(C \mid \boldsymbol{x}) = \prod_{i=1}^{N} x_{c_i} = \frac{N!}{\prod_{\ell=1}^{K} n_\ell !} \prod_{k=1}^{K} x_k^{n_k}$$

.

(a) Show that the Dirichlet distribution (with parameter vector $\boldsymbol{\alpha} \in \mathrm{R}_+^K$)

$$\mathcal{D}(\boldsymbol{x} \mid \boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma\left(\alpha_k\right)} \prod_k x_k^{\alpha_k - 1}$$

is the conjugate prior for the multinomial (1). What is the associated posterior?

Answer:

We first note that the observed data can be summarized with our sufficient statistics $\Phi(C) = n = [n_1, \ldots, n_k]^T$. Our model is summarized in the vector of class probabilities $x = [x_1, x_2, \ldots, x_k]^T$.

Let $B(\alpha)$ be the generalization of of the beta distribution in the general multi-dimensional case such that

$$B(\alpha) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^{K} \alpha_k)}$$

The *Dirichlet* distribution can then be rewritten a:

$$D([x_1, x_2, \ldots, x_k]; \alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} x_k^{\alpha_k - 1}$$

There $B(\alpha)$ served as a normalizer for the product term and it is assumed $\sum x_k = 1$.

Now consider the likelihood function which uses the class observations $n = [n_1, \ldots, n_k]$ to determine the likelihood of $x = [x_1, \ldots, x_k]$. As our data generating function is assumed to be a multinomial-distribution, our observation $\Phi(C) = [n_1 \ldots, n_k]$ change the likelihood of our model x as:

$$L(x) = P(\Phi(C) \,|\, x) \propto \prod_{k=1}^{K} x_k^{n_k}$$

Now we can determine the posterior up to a proportionality constant assuming a dirichlet prior with exponents $\alpha = [\alpha_1, \ldots, \alpha_k]$

$P(x \,|\, \Phi(C), \alpha) \propto \prod_{k=1}^{K} x_k^{n_k} \prod_{k=1}^{K} x_k^{\alpha_k - 1}$

Thus combining like exponent gives us the posterior in the same functional form of the Dirichlet distribution in un-normalized form as

$$P(x \mid \Phi(C), \alpha) \propto \prod_{k=1}^{K} x_k^{n_k + \alpha_k - 1}$$

Let $\alpha' = n_k + \alpha_k$ then the above can be normalized with the corresponding beta function $B(\alpha')$ and is thus given by

$$P(x \mid \Phi(C), \alpha) = D(x \mid \alpha + \Phi(c)) = D(x \mid \alpha + n)$$

where $n = [n_1, \ldots, n_k]$.

In short

We need to show that the posterior distribution of $\boldsymbol{x}$ after observations are also a Dirichlet distribution.

The prior distribution of $\boldsymbol{x}$ is given by

$$\mathcal{D}(\boldsymbol{x} \mid \boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma\left(\alpha_k\right)} \prod_k x_k^{\alpha_k - 1}.$$

The posterior distribution of $\boldsymbol{x}$ is given by

$$\begin{aligned}
p(\boldsymbol{x} \mid C, \boldsymbol{\alpha}) &\propto p(C \mid \boldsymbol{x}) \mathcal{D}(\boldsymbol{x} \mid \boldsymbol{\alpha}) \\
&= \frac{N!}{\prod_{\ell=1}^{K} n_\ell !} \prod_{k=1}^{K} x_k^{n_k} \frac{\Gamma\left(\sum_k \alpha_k\right)}{\prod_k \Gamma\left(\alpha_k\right)} \prod_k x_k^{\alpha_k - 1} \\
&\propto \prod_{k=1}^{K} x_k^{n_k + \alpha_k - 1},
\end{aligned}$$

which is the kernel of a Dirichlet distribution with parameter vector $\boldsymbol{\alpha}' = \boldsymbol{\alpha} + \boldsymbol{n}$, where $\boldsymbol{n} = \left[n_1, n_2, \ldots, n_K\right]$ is the vector of counts of the labels in the dataset C. Therefore, the Dirichlet distribution is the conjugate prior for the multinomial distribution.

The associated posterior is

$$\mathcal{D}(\boldsymbol{x} \mid \boldsymbol{\alpha}') = \frac{\Gamma\left(\sum_k (\alpha_k + n_k)\right)}{\prod_k \Gamma\left(\alpha_k + n_k\right)} \prod_k x_k^{\alpha_k + n_k - 1}.$$

(b) Show that the Dirichlet distribution has the aggregation property: If $p(\boldsymbol{x}) = \mathcal{D}(\boldsymbol{x}; \boldsymbol{\alpha})$, then

$$\begin{aligned}
&p\left(\left[x_1, x_2, \ldots, x_i + x_j, \ldots x_{j-1}, x_{j+1}, \ldots, x_K\right]\right) \\
&\qquad = \mathcal{D}\left(\cdot ; \left[\alpha_1, \alpha_2, \ldots, \alpha_i + \alpha_j, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_K\right]\right)
\end{aligned}$$

Answer

To prove the aggregation property, we need to show that if we have a Dirichlet distribution $\mathcal{D}(x; \boldsymbol{\alpha})$ and we aggregate two of its components, say x_i and x_j to form a new component $x_i + x_j$, then the resulting distribution is again a Dirichlet distribution with parameter vector $[\alpha_1, \alpha_2, \ldots, \alpha_i + \alpha_j, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_K]$.

If $p(x) = \mathcal{D}(x; \boldsymbol{\alpha})$. Then we have:

$$\begin{aligned}
&\mathrm{p}\left([x_1, x_2, \ldots, x_i + x_j, \ldots x_{j-1}, \ldots x_k]\right) \\
&= \frac{\Gamma(\sum_{l=1}^{K} \alpha_l)}{\prod_{l=1}^{K} \Gamma(\alpha_l)} \prod_{l=1}^{K} x_l^{\alpha_l - 1} \\
&= \frac{\Gamma(\alpha_i + \alpha_j)}{\Gamma(\alpha_i)\Gamma(\alpha_j)} (x_i + x_j)^{\alpha_i + \alpha_j - 1} \frac{\Gamma(\sum_{l \neq i,j} \alpha_l)}{\prod_{l \neq i,j} \Gamma(\alpha_l)} \prod_{l \neq i,j} x_l^{\alpha_l - 1} \\
&= \frac{\Gamma(\alpha_i + \alpha_j)}{\Gamma(\alpha_i + \alpha_j)} \frac{\Gamma(\sum_{l=1}^{K} \alpha_l)}{\prod_{l=1}^{K} \Gamma(\alpha_l)} \prod_{l=1}^{K} x_l^{\alpha_l - 1} \\
&= \mathcal{D}(x; [\alpha_1, \alpha_2, \ldots, \alpha_i + \alpha_j, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_K]) \\
&= \mathcal{D}\left([x_1, x_2, \ldots, x_i + x_j, \ldots x_{j-1}, \ldots x_k]; [\alpha_1, \alpha_2, \ldots, \alpha_i + \alpha_j, \ldots, \alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_K]\right)
\end{aligned}$$

Thus, we have shown that the Dirichlet distribution has the aggregation property.

(c) Show that the logarithm of the likelihood (1), as a function of $\boldsymbol{x}$, is the crossentropy loss[1] for $\boldsymbol{x}$. What is the maximum-likelihood estimate for $\boldsymbol{x}$? How does it relate to the maximum [2] of the posterior?

Answer:

Consider the likelihood function assuming that our data is drawn from a multinomial distribution with parameters $x_1, \dots x_k$. The observed dataset C is converted into observed class counts $\Phi(C) = n = [n_1, \dots, n_k]$. Then the likelihood of the model based on observed class counts is given by:

Then the log-likelihood becomes,

$$\log(L(x)) = \log(\prod_{i=1}^{N} x_{c_i}) = \log\left(\frac{N!}{\prod_{l=1}^{K} n_l!}\prod_{k=1}^{K} x_k^{n_k}\right)$$

Simplifying the log likelihood into a model independent constant δ and sum of model terms we get:

$$\log(L(x)) = \sum_{k=1}^{K} n_k \log(x_k) + \delta$$

Dividing this loss by N before optimizing does not effect the optimization per-se but allows us to see the relationship to *cross-entropy*. It is also natural to express the optimization as a minimization problem we consider the term to minimize the the (Cross Entropy) loss as :

$$-\log(L(x)) = -\sum_{k=1}^{K} \frac{n_k}{N} \log(x_k) + \delta$$

The below loss function must be minimized subject to the constraint that $\sum_k x_k = 1$. That is the model probabilities must sum to 1. We introduce the constraint into our function with the lagrangian parameter λ as follows:

$$-\log(L(x)) = -\sum_{k=1}^{K} n_k \log(x_k) + \delta + \lambda(1 - \sum_{k=1} x_k)$$

Now we consider the gradient with respect to x in order minimize negative log-likelihood or maximize the loss, the δ parameter is independent of any model parameters and thus goes to zero.

Consider a i-th term in the gradient $\frac{\partial}{\partial x_i}$ it only preserves the terms in the summation where $i = k$.

$$\frac{\partial L(x, \lambda)}{\partial x_i} = -\frac{n_i}{x_i} - \lambda = 0$$

Thus we have $x_i = -\frac{n_i}{\lambda}$, as we knew that $\sum_k x_i = 1$ we have.

$$\frac{-n_1}{\lambda} + \dots + \frac{-n_k}{\lambda} = x_1 + \dots x_k = 1$$

Thus $\lambda = -\sum_k n_k = -N$.

The MLE estimate for $x_i = \frac{n_i}{N}$ is thus, or the fractional occurrence of class in the training data.

Next we attempt to compute the MAP estimate,

The maximum of the posterior distribution for x can be found using Bayes' rule:

$$p(\boldsymbol{x} \mid C) = \frac{p(C \mid \boldsymbol{x})p(\boldsymbol{x})}{p(C)}$$

$$\log p(\boldsymbol{x} \mid C) = \log p(C \mid \boldsymbol{x}) + \log p(\boldsymbol{x}) - \log p(C) = \sum_{k=1}^{K} n_k \log x_k + \log p(\boldsymbol{x}) - \log p(C)$$

where $p(\boldsymbol{x} \mid C)$ is the posterior distribution of the parameters $\boldsymbol{x}$ given the observed data C, and $p(\boldsymbol{x})$ is the prior distribution of $\boldsymbol{x}$.

The first term on the right-hand side of the equation is the log-likelihood of the observed data given the parameters $\boldsymbol{x}$, which is the same as the log-likelihood we derived earlier for the maximum likelihood estimate. The second term is the log-prior of the parameters, which encodes any prior knowledge or assumptions about the distribution of $\boldsymbol{x}$. The third term is a normalization constant that ensures that the posterior is a valid probability distribution.

The MAP estimate can be obtained by maximizing the log-posterior in the case of a Dirichlet prior:

$$\log p(\boldsymbol{x} \mid C) = \sum_{k=1}^{K} (n_k + \alpha_k - 1)\log x_k - N\sum_{k=1}^{K} \log x_k + \text{const.}$$

α_k is the hyperparameter of the Dirichlet prior distribution for x. Taking the derivative of the log-posterior with respect to x_k, setting it to zero, and solving for x_k, we get:

$$x_k = \frac{n_k + \alpha_k - 1}{\sum_{\ell=1}^{K}(n_\ell + \alpha_\ell - 1)}$$

which is the MAP estimate of x_k in terms of the α_k s. Note that this reduces to the maximum likelihood estimate when all the hyperparameters α_k are equal to 1, which corresponds to a uniform prior distribution. So the MAP estimate is given as $x_i = \frac{n_i+\alpha_i-1}{\sum_{k=1}(n_k+\alpha_k-1)}$, boils down to uniform priors $\alpha_k = 1$ for example we get back to the MLE estimate.

(d) Consider the following situation: You are on an e-commerce marketplace, which uses ratings on the common "five stars" scale. You have the choice between two vendors there. The first vendor has only three ratings so far, but they are all five-star ratings i.e. their rating "vector" is $C_1 = [0, 0, 0, 0, 3]$. The second vendor has many more ratings, their rating vector is $C_2 = [1, 0, 12, 43, 354]$. Let's assume (questionably) that these ratings are drawn iid. from two multinomial distributions with parameter $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, one for each vendor. Use the results above to answer the following questions:

i. What are the maximum likelihood estimates for $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$?

Answer

The MLE estimate for $x_i = \frac{n_i}{N}$ is thus, or the fractional occurrence of class in the data.

x_1 = [0/3,0/3,0/3,0/3,3/3]=[0,0,0,0,1]

x_2 = [1/410,0/410,12/410,43/410,354/410]=[0.0024,0,0.029,0.105,0.86]

ii. Under a uniform Dirichlet prior ($\boldsymbol{\alpha} = [1, 1, 1, 1, 1]$) what is the probability for your rating to be five stars? What is the probability for your rating to be four or five stars? (Assume you pick either one of the vendors, and your future rating will be distributed just as the existing ones. To answer this question, you need to compute integrals of the form $\int x_i \mathcal{D}(\boldsymbol{x} \mid \boldsymbol{\alpha}) d\boldsymbol{x}$. Think about how to do this.)

Answer

Under a uniform Dirichlet prior, the posterior distribution for the parameter $\boldsymbol{x}$ of the multinomial distribution is also a Dirichlet distribution, with parameter $\boldsymbol{\alpha} + \boldsymbol{n}$, where $\boldsymbol{n}$ is the vector of counts of each category in the observed data. Specifically, we have:

$$p(\boldsymbol{x} \mid \boldsymbol{n}, \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha} + \boldsymbol{n})} \prod_{k=1}^{K} x_k^{n_k + \alpha_k - 1}$$

where $B(\boldsymbol{\alpha} + \boldsymbol{n})$ is the multinomial beta function coefficient with $B(\boldsymbol{\alpha}) = \frac{\Pi_{i=1}^{k} \Gamma\left(\alpha_k\right)}{\Gamma\left(\sum_k \alpha_k\right)}$

For the first vendor, we have $\boldsymbol{n}_1 = [0, 0, 0, 0, 3]$, so the posterior distribution for $\boldsymbol{x}_1$ is:

$$\begin{aligned} p(\boldsymbol{x}_1 \mid \boldsymbol{n}_1, \boldsymbol{\alpha}) &= \frac{1}{B([1, 1, 1, 1, 4])} \prod_{k=1}^{5} x_{1,k}^{n_{1,k} + \alpha_k - 1} \\ &= \frac{1}{B([1, 1, 1, 1, 4])} x_{1,5}^{4} \end{aligned}$$

where we used the fact that $n_{1,k} = 0$ for $k \neq 5$, and $\alpha_k = 1$ for all k. Note that the posterior distribution is proportional to $x_{1,5}^{4}$, so the probability that a new rating for the first vendor is five stars is:

$$\begin{aligned} \int_0^1 x_{1,5} p(\boldsymbol{x}_1 \mid \boldsymbol{n}_1, \boldsymbol{\alpha}) dx_{1,5} &= \frac{1}{B([1, 1, 1, 1, 4])} \int_0^1 x_{1,5}^{5} dx_{1,5} \\ &= \frac{1}{B([1, 1, 1, 1, 4])} \frac{1}{6} \end{aligned}$$

To compute the probability that a new rating for the first vendor is four or five stars, we need to integrate over the probabilities of both events:

To compute the probability that a new rating for the first vendor is four or five stars, we need to compute the integral $\int_{x_4+x_5} p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha}) d\boldsymbol{x}_1$, where $p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha})$ is the Dirichlet distribution with parameter $\boldsymbol{\alpha} = [1, 1, 1, 1, 1]$.

Using the formula for the Dirichlet distribution, we have:

$$p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{5} x_k^{\alpha_k - 1} = \frac{1}{B(1, 1, 1, 1, 1)} \prod_{k=1}^{5} x_k^{1-1} = \frac{1}{B(1, 1, 1, 1, 1)}$$

where $B(\cdot)$ is the Beta function.

Therefore, we have:

$$\int_{x_4+x_5} p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha}) d\boldsymbol{x}_1 = \int_0^1 \int_0^{1-x_4} p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha}) dx_4 dx_5 = \int_0^1 \int_0^{1-x_4} \frac{1}{B(1, 1, 1, 1, 1)} dx_4 dx_5$$

Integrating this expression gives:

$$\int_{x_4+x_5} p(\boldsymbol{x}_1 \mid \boldsymbol{\alpha}) d\boldsymbol{x}_1 = \frac{1}{B(1,1,1,1,1)} \int_0^1 \int_0^{1-x_4} dx_4 dx_5 = \frac{1}{B(1,1,1,1,1)} \int_0^1 (1-x_4) dx_4 = \frac{1}{B(1,1,1,1,1)} \cdot \frac{1}{2}$$

where we used the fact that the integral of $(1 - x_4)$ over $[0, 1]$ is equal to $1/2$. Therefore, the probability for a new rating for the first vendor to be four or five stars is $\frac{1}{2B(1,1,1,1,1)}$.

Laplace Approximations

In [ ]:
```
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
import warnings

warnings.filterwarnings( # ignore warnings for this notebook only
    "ignore")


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

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` wherever 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`.

In [ ]:
```
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
    p_Z = jax.scipy.stats.beta.pdf(z, a, b)
    return p_Z #/ jnp.sum(p_Z)
```

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

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

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

Theorem (Transformation Law, general) Let $X = \left(X_1, \ldots, X_d\right)$ have a joint density p_X. Let $g: \mathrm{R}^d \rightarrow \mathrm{R}^d$ be continously differentiable and injective, with non-vanishing Jacobian J_g. Then $X = g(Z)$ has density

$$p_X(x) = \begin{cases} p_Z\left(g^{-1}(x)\right) \cdot \left|J_{g^{-1}}(x)\right| & \text{if } x \text{ is in the range of } g \\ 0 & \text{otherwise.} \end{cases}$$

In [ ]:
```
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
    logistic_x = jax.scipy.special.expit(x)
    p_Z = p_z(logistic_x, a, b)
    p_X = p_Z * jax.scipy.special.expit(x) * (1 - jax.scipy.special.expit(x))
    return p_X #/ jnp.sum(p_X)
```

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

In general Laplace Approximation is:

$$p(x) \approx f(\hat{x})\exp\left[-\frac{H}{2}(x-\hat{x})^2\right]$$

where H is the Hessian (matrix or scalar depending on the dimensions) and $\hat{x}$ is the mode. Approximate distribution, $q(x)$ by making use of the standard normalisation of a Gaussian so that $q(x)$ is:

$$q(x) = \left(\frac{H}{2\pi}\right)^{1/2}\exp\left[-\frac{H}{2}(x-\hat{x})^2\right]$$

Similar formulation to the above can also be seen at Lecture 03 Slide 33-35.

In [ ]:
```
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.

    Returns:
      A function with the same argument as the beta distribution.
    """
    # TODO

    #Laplace approximation for the beta distribution
    z_hat = (a - 1) / (a + b - 2) # mode of the beta distribution
    psi = - (a-1)/z_hat**2 - (b-1)/(1-z_hat)**2  # Hessian at the mode
    p_Z = lambda z: jnp.exp(jnp.log(p_z(z_hat, a, b))+psi/2*(z-z_hat)**2) \
        #/ jnp.sum(jnp.exp(p_z(z_hat, a, b)+psi/2*(z-z_hat)**2)) #normalization

    return p_Z

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
    p_Z = laplace_z(a, b)
    p_X = lambda x: p_Z(jax.scipy.special.expit(x)) * jax.scipy.special.expit(x) * (1 - jax.scipy.special.expit
        #/ jnp.sum(p_Z(jax.scipy.special.expit(x)) * jax.scipy.special.expit(x) * (1 - jax.scipy.special.expit(
    return p_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.

The approximation is undefined for a + b = 2 as $\hat{z} = \frac{a-1}{a+b-2}$

In [ ]:
```
# TODO: Plot
z = jnp.linspace(0, 1, 1000)
x = jnp.linspace(-5, 5, 1000)
a = 2
b = 3
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

P_z = p_z(z, a, b)
P_x = p_x(x, a, b)
P_z_laplace = laplace_z(a, b)(z)
P_x_laplace = laplace_x(a, b)(x)
```

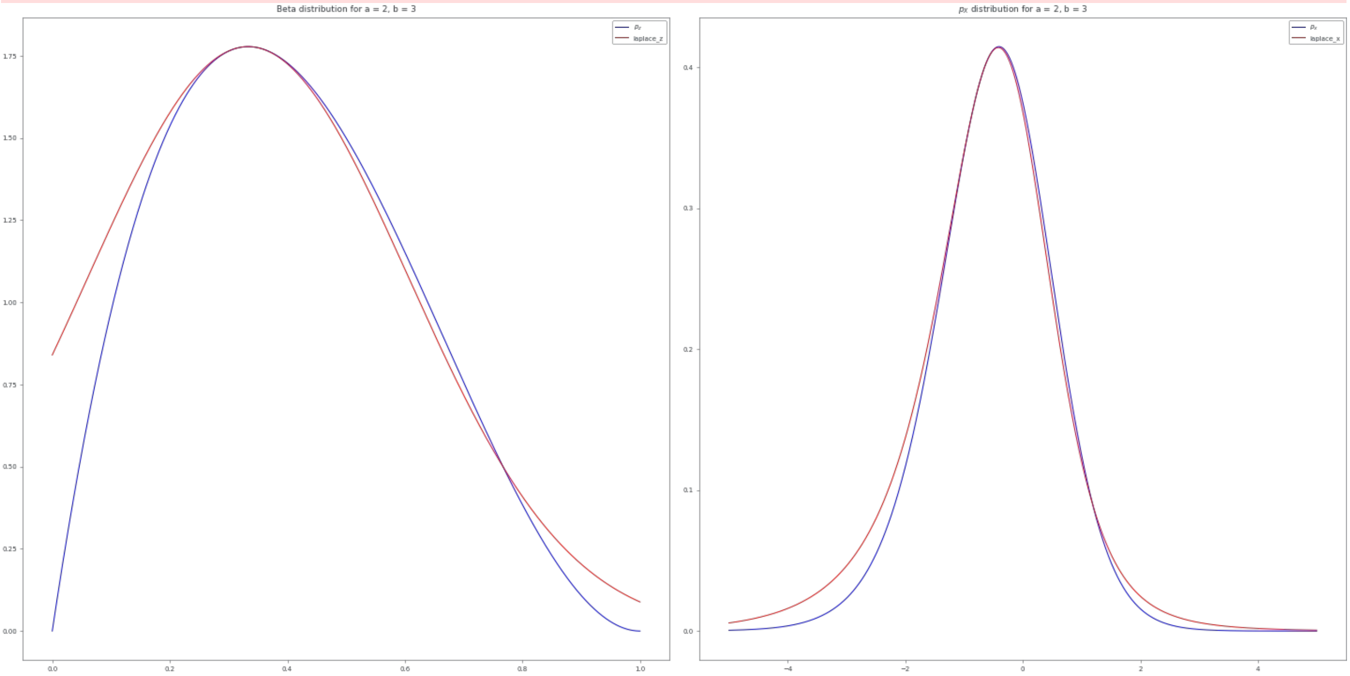
```
ax1.plot(z, P_z, label="$p_z$", color="blue")
ax1.plot(z, P_z_laplace, label="laplace_z", color="red")
ax1.set_title("Beta distribution for a = 2, b = 3")
ax1.legend()

ax2.plot(x, P_x, label="$p_x$", color="blue")
ax2.plot(x, P_x_laplace, label="laplace_x", color="red")
ax2.set_title("$p_X$ distribution for a = 2, b = 3")
ax2.legend()

# TODO: Plot
a = 5
b = 5
z =jnp.linspace(0, 1, 1000)
x = jnp.linspace(-5, 5, 1000)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
ax1.plot(z, P_z, label="$p_z$", color="blue")
ax1.plot(z, P_z_laplace, label="laplace_z", color="red")
ax1.set_title("Beta distribution for a = 5, b = 5")

ax1.legend()

ax2.plot(x, P_x, label="$p_x$", color="blue")
ax2.plot(x, P_x_laplace, label="laplace_x", color="red")
ax2.set_title("$p_X$ distribution for a = 5, b = 5")
ax2.legend()
```

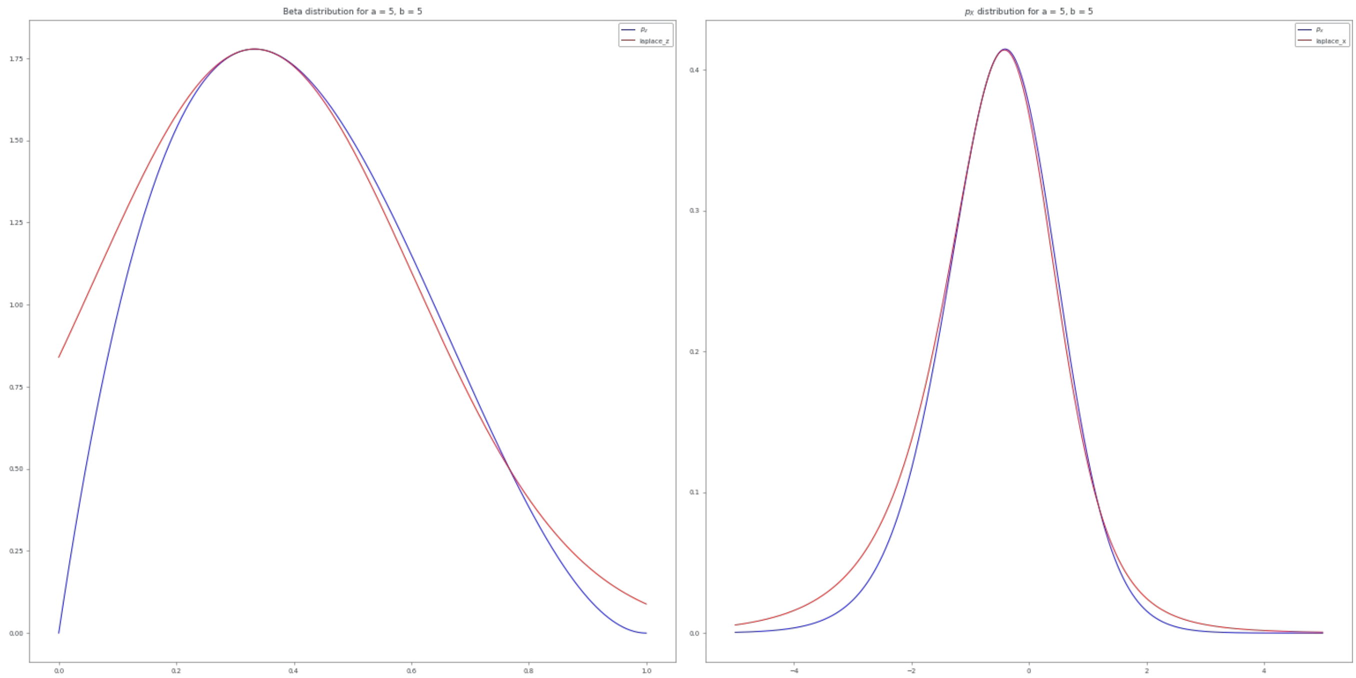
```
WARNING:jax._src.lib.xla_bridge:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun fo
r more info.)
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7f03cc032520>
```

```
WARNING:matplotlib.font_manager:findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans-serif' not found because none of the following f
amilies were found: Roboto Condensed
WARNING:matplotlib.font_manager:findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans-serif' not found because none of the following f
amilies were found: Roboto Condensed
WARNING:matplotlib.font_manager:findfont: Font family ['cursive'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'cursive' not found because none of the following fami
lies were found: Apple Chancery, Textile, Zapf Chancery, Sand, Script MT, Felipa, Comic Neue, Comic Sans MS, cu
rsive
WARNING:matplotlib.font_manager:findfont: Font family ['sans'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans' not found because none of the following familie
s were found: Roboto Condensed
WARNING:matplotlib.font_manager:findfont: Font family ['sans'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans' not found because none of the following familie
s were found: Roboto Condensed
WARNING:matplotlib.font_manager:findfont: Font family ['sans'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans' not found because none of the following familie
s were found: Roboto Condensed
```





Task 5. Implement the Dirichlet distribution

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

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

The mode of the Dirichlet distribution is given by:

$$\mathbf{x}_{\text{mode}} = \left(\frac{\alpha_1 - 1}{\sum_{j=1}^{K} (\alpha_j - 1)}, \frac{\alpha_2 - 1}{\sum_{j=1}^{K} (\alpha_j - 1)}, \ldots, \frac{\alpha_K - 1}{\sum_{j=1}^{K} (\alpha_j - 1)} \right).$$

.

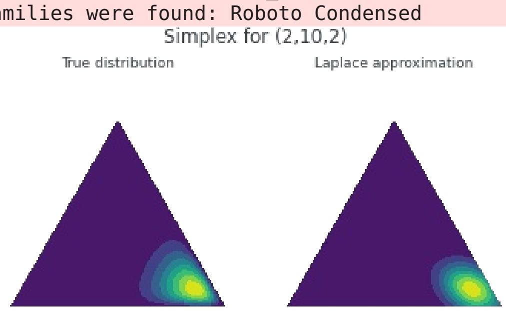
The Hessian of the log-likelihood at $\mathbf{x}_{mode}$ is a negative definite matrix with entries $-\frac{\alpha_i - 1}{x_{mode}^2}$ on the diagonal.

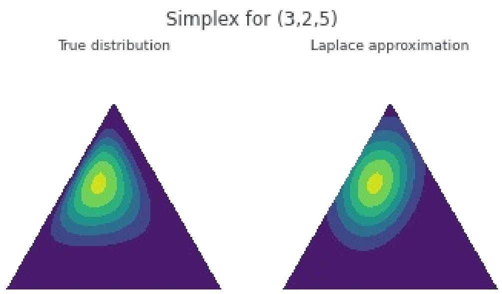
In [ ]:
```python
from scipy.special import digamma

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 jax.scipy.stats.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.
    """
    #Laplace approximation for the Dirichlet distribution using the mean and covariance of the multivariate nor
    mode = (alpha - 1) / jnp.sum(alpha - 1)
    hessian = jnp.diag( (alpha - 1) / mode**2)
    return lambda y: jnp.exp(jnp.log(p_y(mode, alpha)) - 0.5 * (y - mode) @ hessian @ (y - mode))
```

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.

```python
def simplex_contour_plot(fun1, fun2, description="Simplex for (2,10,2)"):
    """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=5)

    # 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))
    plt.suptitle(description)
    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')
    axes[0].set_title("True distribution")
    axes[1].set_title("Laplace approximation")
    plt.show()


def func_p_y(alpha):
    return lambda y: p_y(y, alpha)


# TODO: Plot
alpha = jnp.array([2, 10, 2])
simplex_contour_plot(func_p_y(alpha=alpha), laplace_y(alpha), description="Simplex for (2,10,2)")

alpha = jnp.array([3, 2, 5])
simplex_contour_plot(func_p_y(alpha=alpha), laplace_y(alpha), description="Simplex for (3,2,5)")
```

```
WARNING:matplotlib.font_manager:findfont: Font family ['sans-serif'] not found. Falling back to DejaVu Sans.
WARNING:matplotlib.font_manager:findfont: Generic family 'sans-serif' not found because none of the following f
amilies were found: Roboto Condensed
```





Note that throughout exercise 2.2, probability distributions are not normalized to sum to 1. If desired, this can be easily done by uncommenting the necessary lines coded.