Lista_3-Raphael_Levy

April 4, 2023

[1]:
```
# !pip install torch
# !pip install typing-extensions --upgrade
# !pip install Cython
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
import torch
import torch.nn.functional as F
from torch.autograd.functional import hessian
from torch.distributions.multivariate_normal import MultivariateNormal
import seaborn as sns
```

Instruções gerais: Sua submissão deve conter: 1. Um "ipynb" com seu código e as soluções dos problemas 2. Uma versão pdf do ipynb

Caso você opte por resolver as questões de "papel e caneta" em um editor de LaTeX externo, o inclua no final da versão pdf do 'ipynb'— submetendo um único pdf.

1 Trabalho de casa 03: Regressão logística e inferência Bayesiana aproximada

O pedaço de código abaixo carrega o banco de dados 'breast cancer' e adiciona uma coluna de bias. Além disso, ele o particiona em treino e teste.

1. Implemente a estimativa de máximo a posteriori para um modelo de regressão logística com priori $\mathcal{N}(0, cI)$ com $c = 100$ usando esse banco de dados;
2. Implemente a aproximação de Laplace para o mesmo modelo;
3. Implemente uma aproximação variacional usando uma Gaussiana diagonal e o truque da reparametrização;
4. Calcule a accuracy no teste para todas as opções acima — no caso das 2 últimas, a prob predita é $\int_\theta p(y|x,\theta)q(\theta)$;
5. Para cada uma das 3 técnicas, plote um gráfico com a distribuição das entropias para as predições corretas e erradas (separadamente), use a função kdeplot da biblioteca seaborn.
6. Comente os resultados, incluindo uma comparação dos gráficos das entropias.

Explique sua implementação também!

Para facilitar sua vida: use PyTorch, Adam para otimizar (é uma variação SGD) com lr=0.001, use o banco de treino inteiro ao invés de minibatchces, use binary_cross_entropy_with_logits para implementar a -log verossimilhança, use torch.autograd.functional para calcular a Hessiana. Você pode usar as bibliotecas importadas na primeira célula a vontade. Verifique a documentação de binary_cross_entropy_with_logits para garantir que a sua priori está implementada corretamente, preservando as proporções devidas. Use 10000 amostras das aproximações para calcular suas predições.

[3]:
```
data = load_breast_cancer()
N = len(data.data)
Ntrain = int(np.ceil(N*0.6))
perm = np.random.permutation(len(data.data))
X = torch.tensor(data.data).float()
X = torch.cat((X, torch.ones((X.shape[0], 1))), axis=1)
y = torch.tensor(data.target).float()

Xtrain, ytrain = X[perm[:Ntrain]], y[perm[:Ntrain]]
Xtest, ytest = X[perm[Ntrain:]], y[perm[Ntrain:]]
```

Questão 1

[4]:
```
from torch.optim import Adam
from torch.nn.functional import binary_cross_entropy_with_logits
from sklearn.metrics import accuracy_score

n, d = Xtrain.shape
c = 100
learning_rate = 1e-3
epochs = 100000

# p(\theta) = N(0, 100), desv pad = 10
theta = torch.tensor(torch.zeros(d), requires_grad=True)

print('Theta_0: ', theta)
optimizer = torch.optim.Adam([theta], learning_rate, maximize=True)

# Mínimos Quadrados Lineares (Linear Least Squares)
lls = []

for epoch in range(epochs):
    optimizer.zero_grad()
    logits = Xtrain @ theta
    # Calculo da Sigmoide
    softmax = torch.sigmoid(logits)
    # log p(\theta | y) = \sum_i y_i log(\sigma(\theta^T x_i)) + (1 - y_i) log(1␣
 ↪- \sigma(\theta^T x_i) ~ Bernoulli(D | \theta)
    log_likelihood = -F.binary_cross_entropy_with_logits(logits, ytrain)
    # Log Verossimilhança da Dist. de Bernoulli
```

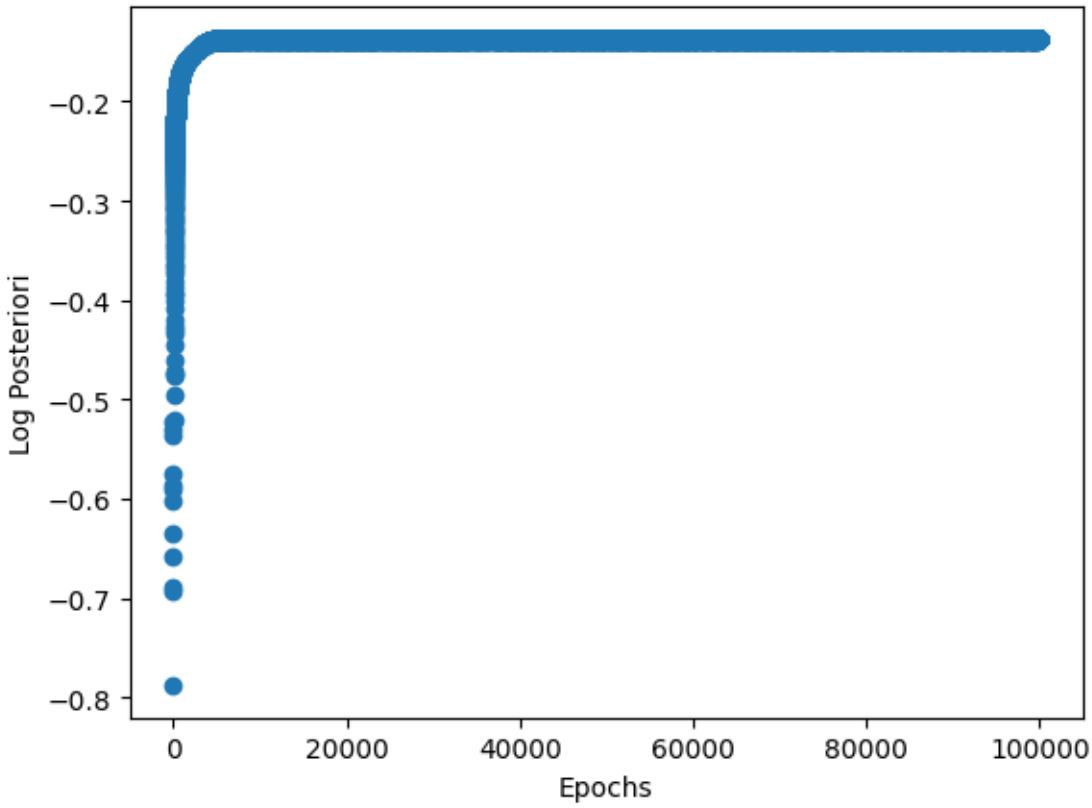
```
    # Log Verossimilhança = torch.sum(ytrain * torch.log(softmax) + (1 - ytrain)␣
 ↪* torch.log(1 - softmax))
    if(torch.sum(torch.isnan(log_likelihood)) > 0):
        print('Epoch: ', epoch)
        print('Log_Verossimilhança: ', log_likelihood)
        print('softmax: ', softmax)
        print('y_train: ', ytrain)
        print('Theta: ', theta)
        break

    # log p(\theta) = -\frac{1}{2} \theta^T \Sigma^{-1} \theta ~ N(0, \Sigma I)
    log_priori = -torch.sum((theta ** 2) / (2*c))

    log_posteriori = (log_likelihood + log_priori)

    # Posteriori = torch.exp(log_posteriori)
    lls.append(log_posteriori.item())
    log_posteriori.backward()

    optimizer.step()
    if torch.norm(theta.grad) < 1e-6:
        break
    theta.grad.zero_()

# Máximo a posteriori de theta
print('Theta final: ',theta)

# Calcula a estimativa MAP de theta

y_pred_1 = torch.sigmoid(Xtest @ theta)
print('Y predito:', y_pred_1)
```

```
<ipython-input-4-4c09842d8e7e>:11: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  theta = torch.tensor(torch.zeros(d), requires_grad=True)
```

```
Theta_0:  tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0.], requires_grad=True)
Theta final:  tensor([ 0.9993,  0.1203,  0.2047, -0.0141, -0.0428, -0.1797,
-0.2492, -0.1188,
        -0.0875, -0.0114,  0.0211,  0.3462,  0.1342, -0.1062, -0.0031, -0.0299,
        -0.0328, -0.0141, -0.0234, -0.0020,  0.9694, -0.2394, -0.2105, -0.0128,
        -0.0769, -0.5337, -0.6588, -0.2376, -0.2656, -0.0504,  0.1744],
       requires_grad=True)
Y predito: tensor([9.8924e-01, 2.3312e-02, 8.1948e-01, 9.8127e-01, 2.1714e-08,
```

```
8.9470e-01,
        7.3746e-01, 1.6697e-07, 1.3641e-01, 5.4525e-01, 7.9158e-01, 2.9189e-01,
        8.3167e-01, 9.9621e-01, 9.4186e-02, 3.3923e-05, 2.7359e-25, 5.6467e-04,
        9.9887e-01, 9.7470e-01, 3.6759e-01, 9.9457e-01, 7.3906e-02, 9.6800e-01,
        9.9452e-01, 9.9612e-01, 7.9576e-01, 9.9346e-01, 9.9597e-01, 9.9213e-01,
        5.7782e-01, 9.8141e-01, 9.9270e-01, 9.8528e-01, 9.9780e-01, 9.7535e-01,
        2.6797e-03, 9.8934e-01, 9.9875e-01, 9.5200e-01, 1.6395e-01, 1.3228e-01,
        9.4300e-01, 1.9714e-04, 4.6169e-07, 7.3732e-04, 2.1077e-04, 2.4637e-03,
        1.7741e-18, 9.1368e-01, 8.7981e-01, 5.7688e-14, 9.6097e-01, 8.6425e-01,
        9.8775e-01, 8.6637e-01, 9.9176e-01, 9.7601e-01, 1.6597e-02, 9.9016e-01,
        3.1692e-16, 7.0902e-01, 4.9475e-03, 9.8885e-01, 8.0072e-01, 1.0351e-22,
        9.5062e-01, 8.7988e-01, 9.9131e-01, 9.9266e-01, 1.0684e-02, 9.7888e-01,
        9.9042e-01, 9.8815e-01, 9.1060e-01, 1.2964e-08, 1.8424e-07, 9.5083e-01,
        9.9735e-01, 4.7842e-06, 1.0997e-13, 3.3977e-01, 4.2100e-01, 9.8505e-01,
        9.9851e-01, 9.8836e-01, 4.5448e-08, 9.9575e-01, 8.5038e-01, 9.9796e-01,
        3.4615e-30, 9.7160e-01, 9.4713e-01, 1.5534e-07, 5.3911e-06, 9.8092e-01,
        5.9980e-02, 2.1509e-07, 7.8114e-02, 8.1816e-01, 7.3831e-01, 7.0373e-01,
        9.9770e-01, 9.9488e-01, 9.1648e-01, 2.3524e-11, 9.8767e-01, 9.9556e-01,
        9.9232e-01, 3.3471e-11, 8.5139e-06, 2.0351e-02, 9.9443e-01, 9.6200e-01,
        9.9712e-01, 9.8214e-01, 9.9208e-01, 8.3950e-01, 1.5962e-03, 9.7428e-01,
        8.1351e-02, 1.6277e-13, 9.6445e-01, 8.0373e-01, 8.5408e-01, 9.9831e-01,
        9.8773e-01, 2.1725e-18, 9.8129e-01, 9.4176e-01, 9.9635e-01, 2.5267e-04,
        9.2964e-01, 6.9549e-02, 9.8023e-01, 9.9562e-01, 8.1675e-01, 3.8224e-02,
        4.0062e-04, 9.8513e-01, 9.5618e-01, 6.1594e-04, 8.8347e-01, 9.8723e-01,
        9.9679e-01, 8.7633e-01, 9.8996e-01, 9.9881e-01, 2.0206e-03, 9.9428e-01,
        5.1621e-04, 1.1491e-08, 9.2903e-01, 9.8820e-01, 6.3126e-01, 9.7953e-01,
        9.6473e-01, 3.5088e-09, 9.1237e-01, 1.0424e-07, 1.3822e-04, 8.4179e-02,
        1.1920e-08, 4.2484e-11, 9.9911e-01, 9.9572e-01, 9.7322e-01, 8.7501e-09,
        1.7000e-01, 9.9602e-01, 9.8662e-01, 9.2443e-01, 9.8107e-05, 3.5028e-02,
        9.9182e-01, 9.9966e-01, 9.8178e-01, 4.8063e-01, 5.9171e-03, 9.9788e-01,
        9.9844e-01, 9.9149e-01, 9.9508e-01, 7.0799e-07, 9.9614e-01, 9.8354e-01,
        9.8966e-01, 9.9818e-01, 9.1607e-01, 8.6240e-01, 1.1246e-04, 2.0753e-08,
        9.8147e-01, 9.7561e-01, 1.9921e-01, 8.8679e-01, 2.7330e-09, 1.8856e-07,
        9.4874e-01, 9.2147e-01, 9.9692e-01, 9.9900e-01, 4.2710e-03, 9.9877e-01,
        9.7287e-01, 4.9297e-09, 9.8736e-01, 9.7872e-01, 8.2969e-08, 9.9116e-01,
        5.3566e-03, 2.2062e-01, 9.7880e-01, 1.8826e-02, 9.8655e-01, 4.9014e-01,
        3.7537e-06, 2.9363e-09, 9.7375e-01, 9.5321e-01, 2.7697e-01, 9.4291e-01,
        9.9467e-01, 1.8261e-14, 9.9027e-01, 9.9745e-01, 7.3477e-01],
       grad_fn=<SigmoidBackward0>)
```

[5]:
```
# plot the loss against epochs using scatter plot
plt.scatter(range(len(lls)), lls)
plt.xlabel('Epochs')
plt.ylabel('Log Posteriori')
plt.show()
```



Questão 2

[6]:
```
# Implementa a aproximação de Laplace para a distribuicao posteriori de theta
# \mu: arg max da distribuicao posteriori p(\theta | D)
mu = theta.detach().numpy()
# \Sigma: inversa da matriz hessiana da distribuicao log posteriori
hessiana = hessian(lambda theta: -F.binary_cross_entropy_with_logits(Xtrain @␣
 ↪theta, ytrain) - torch.sum((theta ** 2) / (2*c)), theta).detach().numpy()
sigma = np.linalg.inv(hessiana)

# Gera N amostras da distribuicao posteriori
theta_samples = np.random.multivariate_normal(mu, sigma, 10000)
# Calcula a verossimilanca de cada exemplo
likelihoods = torch.sigmoid(Xtest @ torch.tensor(theta_samples).float().T)
# Calcula a média das verossimilhancas
y_pred_2 = torch.mean(likelihoods, axis=1)

print('Y predito:', y_pred_2)
```

```
Y predito: tensor([0.6351, 0.4274, 0.5576, 0.6187, 0.3741, 0.5817, 0.5372,
0.3788, 0.4668,
        0.4973, 0.5369, 0.4754, 0.5668, 0.6397, 0.4305, 0.3811, 0.3977, 0.4116,
```

```
        0.6256, 0.6340, 0.4885, 0.6390, 0.4628, 0.6268, 0.6296, 0.6418, 0.5369,
        0.6386, 0.6360, 0.6418, 0.5075, 0.6304, 0.6311, 0.6269, 0.6588, 0.6138,
        0.4172, 0.6484, 0.6449, 0.5964, 0.4713, 0.4584, 0.5612, 0.4071, 0.3792,
        0.4020, 0.3896, 0.4002, 0.3780, 0.5917, 0.5565, 0.3950, 0.6296, 0.5619,
        0.6371, 0.5993, 0.6470, 0.6294, 0.4160, 0.6462, 0.3609, 0.5329, 0.4122,
        0.6484, 0.5601, 0.3837, 0.6052, 0.5406, 0.6318, 0.6547, 0.4230, 0.6158,
        0.6447, 0.5871, 0.6271, 0.3884, 0.3729, 0.6243, 0.6487, 0.3830, 0.3768,
        0.4737, 0.4954, 0.6560, 0.6336, 0.6352, 0.3778, 0.6436, 0.5619, 0.6296,
        0.4239, 0.5639, 0.5500, 0.3791, 0.3917, 0.6141, 0.4523, 0.3700, 0.4608,
        0.5458, 0.5243, 0.5384, 0.6231, 0.6341, 0.6294, 0.3951, 0.6221, 0.6314,
        0.6504, 0.3934, 0.3771, 0.4081, 0.6404, 0.6319, 0.6276, 0.6115, 0.6563,
        0.5643, 0.3934, 0.6007, 0.4195, 0.3832, 0.6041, 0.5448, 0.5455, 0.6452,
        0.6311, 0.3715, 0.5706, 0.6142, 0.6576, 0.3998, 0.5889, 0.4455, 0.6441,
        0.6341, 0.5638, 0.4448, 0.4055, 0.6472, 0.6263, 0.4253, 0.5968, 0.6383,
        0.6134, 0.5903, 0.6465, 0.6448, 0.4398, 0.6315, 0.3738, 0.3646, 0.5805,
        0.6405, 0.5190, 0.6362, 0.5794, 0.3746, 0.5866, 0.3879, 0.4009, 0.4308,
        0.3747, 0.3622, 0.6388, 0.6270, 0.6396, 0.3894, 0.4599, 0.6471, 0.6399,
        0.5699, 0.4191, 0.4286, 0.6323, 0.6502, 0.5975, 0.4936, 0.4215, 0.6176,
        0.6430, 0.6302, 0.6556, 0.3852, 0.6527, 0.5946, 0.6222, 0.6238, 0.5623,
        0.5490, 0.4002, 0.3728, 0.6437, 0.6259, 0.4598, 0.5512, 0.3740, 0.3763,
        0.6195, 0.5855, 0.6411, 0.6367, 0.4112, 0.6407, 0.6269, 0.3730, 0.6309,
        0.6081, 0.3710, 0.6331, 0.4104, 0.4822, 0.5843, 0.4195, 0.6264, 0.4906,
        0.3882, 0.3612, 0.6068, 0.5866, 0.4769, 0.6065, 0.6467, 0.3604, 0.6327,
        0.6278, 0.5316])

<ipython-input-6-1bb2d74963c8>:9: RuntimeWarning: covariance is not positive-
semidefinite.
  theta_samples = np.random.multivariate_normal(mu, sigma, 10000)
```

Questão 3

[7]:
```
epochs = 1000
mu = torch.randn(d, requires_grad=True)
param_estimator = torch.randn(1, requires_grad=True)
learning_rate = 1e-3

optimizer = torch.optim.Adam([mu, param_estimator], learning_rate) # as␣
 ↪variaveis que serão otimizadas são mu e o estimador

losses = []
n = 1000 # número de amostras para calcular a média da verossimilhança (10000␣
 ↪levou muito tempo executando)
priori_var = 100 # variância da priori

for epoch in range(epochs):
    optimizer.zero_grad()
    estimator_quad = param_estimator**2
```

```
    # \theta^t = epsilon^t * sigma + mu
    # Truque da reparametrizacao
    thetas = torch.randn(n, d) * estimator_quad + mu # amostras de theta

    loss = 0

    for t in range(n):
        theta_t = thetas[t, :]
        logits = Xtrain @ theta_t  # logits = X * theta^t

        # a cross-entropy binária é o -log da verossimilhança de Bernoulli
        neg_log_likelihood = F.binary_cross_entropy_with_logits(logits, ytrain)
 ↪# neg_log_likelihood = -log(p(y|X, theta^t))
        log_priori = -0.5 * (theta_t @theta_t )/priori_var # log_priori =
 ↪log(p(theta^t))
        log_q = -d * estimator_quad.log() - 0.5 * ((theta_t  - mu)@(theta_t -
 ↪mu))/estimator_quad # log_q = log(q(theta^t))

        loss += (neg_log_likelihood - log_priori + log_q)/n # loss = E_{theta^t}
 ↪[ -log(p(y|X, theta^t)) + log(p(theta^t)) - log(q(theta^t)) ]
        losses.append(loss.detach())

    loss.backward()
    optimizer.step()

y_pred_3 = torch.zeros_like(ytest)

# Monte Carlo
for t in range(n):
    thetas = torch.randn(d) * param_estimator**2 + mu
    logits = Xtest @ thetas
    y_pred_3 += torch.sigmoid(logits)

# Calcula a média de y_pred
y_pred_3 = y_pred_3/n

print('Y predito:', y_pred_3)
```

```
Y predito: tensor([0.6872, 0.4011, 0.4382, 0.5567, 0.0874, 0.5992, 0.3478,
0.1367, 0.2432,
        0.2284, 0.7040, 0.3690, 0.6759, 0.6469, 0.2545, 0.1393, 0.0114, 0.3133,
        0.6395, 0.5194, 0.4226, 0.7209, 0.2653, 0.6403, 0.9158, 0.9254, 0.5599,
        0.7295, 0.7589, 0.6895, 0.5973, 0.7114, 0.5737, 0.4761, 0.7401, 0.5504,
        0.1326, 0.5750, 0.6905, 0.5842, 0.4094, 0.3826, 0.9039, 0.1326, 0.1879,
        0.1661, 0.1769, 0.2038, 0.0326, 0.4170, 0.4713, 0.0270, 0.4339, 0.5419,
        0.5361, 0.4274, 0.6036, 0.6515, 0.2160, 0.5793, 0.0926, 0.5134, 0.2089,
        0.8218, 0.4436, 0.0260, 0.4526, 0.6310, 0.5828, 0.6278, 0.1798, 0.8026,
```

```
0.5898, 0.4364, 0.5081, 0.0689, 0.1262, 0.5775, 0.6440, 0.0895, 0.0593,
0.2504, 0.4470, 0.7118, 0.7647, 0.6402, 0.2017, 0.8275, 0.3399, 0.9055,
0.4937, 0.3667, 0.6365, 0.1387, 0.0933, 0.5258, 0.1303, 0.1031, 0.4589,
0.4133, 0.6528, 0.4944, 0.9422, 0.6606, 0.4654, 0.0401, 0.9562, 0.6189,
0.6558, 0.0410, 0.1422, 0.2184, 0.6420, 0.5767, 0.9542, 0.6794, 0.7344,
0.3392, 0.1752, 0.4316, 0.2829, 0.0668, 0.4884, 0.5700, 0.2993, 0.8681,
0.7248, 0.0435, 0.7532, 0.5123, 0.8526, 0.1189, 0.5657, 0.3335, 0.6491,
0.7780, 0.4423, 0.2481, 0.0932, 0.5730, 0.5740, 0.4255, 0.3719, 0.8146,
0.7035, 0.4676, 0.8693, 0.9364, 0.2285, 0.8202, 0.2319, 0.2499, 0.7271,
0.6697, 0.5482, 0.6774, 0.5908, 0.1663, 0.6470, 0.0690, 0.1150, 0.3574,
0.0701, 0.0885, 0.8739, 0.7626, 0.5130, 0.1519, 0.2254, 0.7177, 0.5733,
0.8930, 0.1169, 0.2586, 0.4997, 0.8576, 0.5228, 0.3458, 0.3467, 0.9083,
0.8098, 0.5583, 0.6674, 0.0861, 0.8444, 0.9778, 0.5115, 0.8592, 0.4889,
0.3399, 0.1021, 0.1316, 0.6375, 0.6792, 0.2476, 0.5869, 0.0704, 0.1735,
0.7180, 0.4454, 0.7783, 0.6736, 0.1626, 0.7449, 0.5449, 0.1876, 0.8409,
0.4379, 0.0896, 0.6573, 0.1983, 0.6342, 0.9025, 0.1949, 0.5825, 0.3343,
0.1135, 0.1248, 0.8470, 0.6052, 0.2928, 0.5431, 0.9932, 0.0891, 0.5686,
0.5312, 0.3454], grad_fn=<DivBackward0>)
```

Questão 4

[8]:
```
# Calcula a acurácia da questao 1
accuracy_1 = torch.sum((y_pred_1 > 0.5) == ytest)/len(ytest)
print('Acurácia_Q1: ', accuracy_1)

# Calcula a acurácia da questao 2
accuracy_2 = torch.sum((y_pred_2 > 0.5) == ytest)/len(ytest)
print('Acurácia_Q2: ', accuracy_2)

# Calcula a acurácia da questao 3
accuracy_3 = torch.sum((y_pred_3 > 0.5) == ytest)/len(ytest)
print('Acurácia_Q3: ', accuracy_3)
```

```
Acurácia_Q1:  tensor(0.9515)
Acurácia_Q2:  tensor(0.9559)
Acurácia_Q3:  tensor(0.8546)
```

Questão 5

[9]:
```
preds = [y_pred_1.detach().numpy(), y_pred_2.detach().numpy(), y_pred_3.detach().
 ↪numpy()]

# plot the entropy for each y_pred
# split y in correct and incorrect
y_correct_1 = (y_pred_1 > 0.5) == ytest
print('Classificação Questão 1:\n', y_correct_1)

print(' ')
```

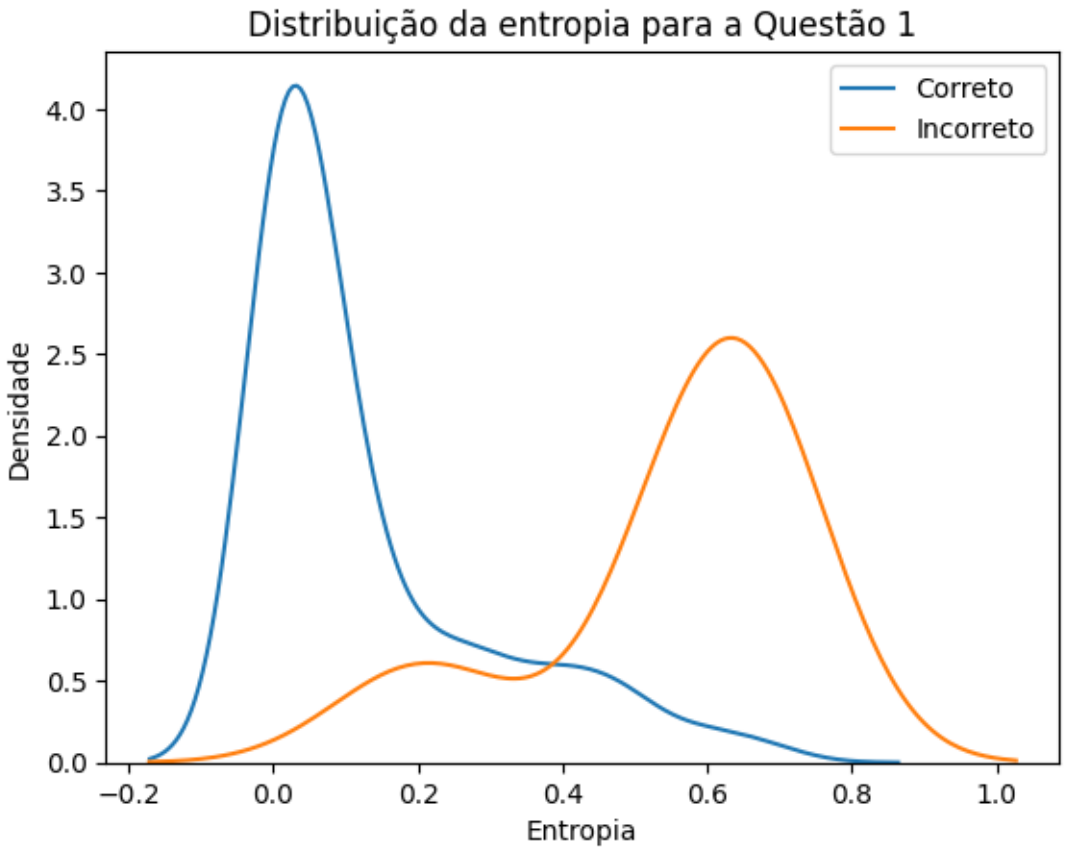
```
y_correct_2 = (y_pred_2 > 0.5) == ytest
print('Classificação Questão 2:\n', y_correct_2)

print(' ')

y_correct_3 = (y_pred_3 > 0.5) == ytest
print('Classificação Questão 3:\n', y_correct_3)
```

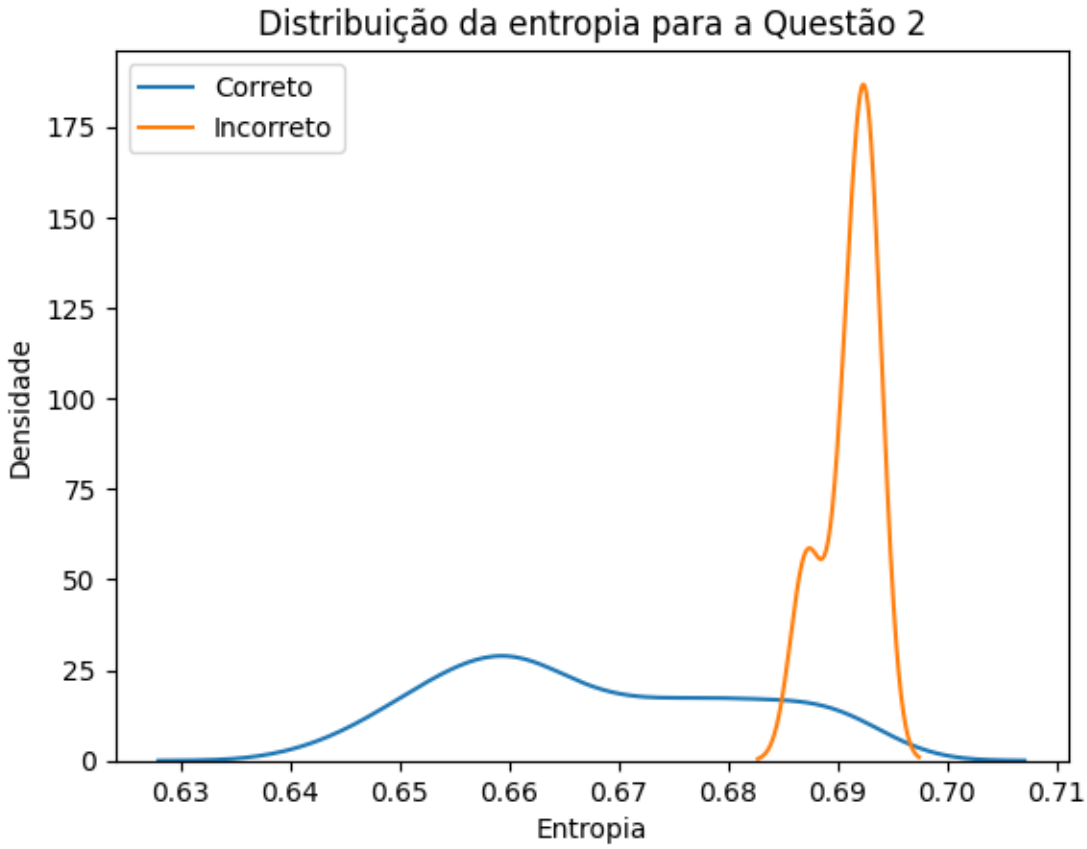
```
Classificação Questão 1:
 tensor([ True,  True,  True,  True,  True,  True, False,  True,  True, False,
         True, False,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True, False,  True,  True,  True, False,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True, False,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True, False,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True, False,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True, False])

Classificação Questão 2:
 tensor([ True,  True,  True,  True,  True,  True, False,  True,  True,  True,
         True, False,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True, False,  True,  True,  True, False,  True,  True,
```

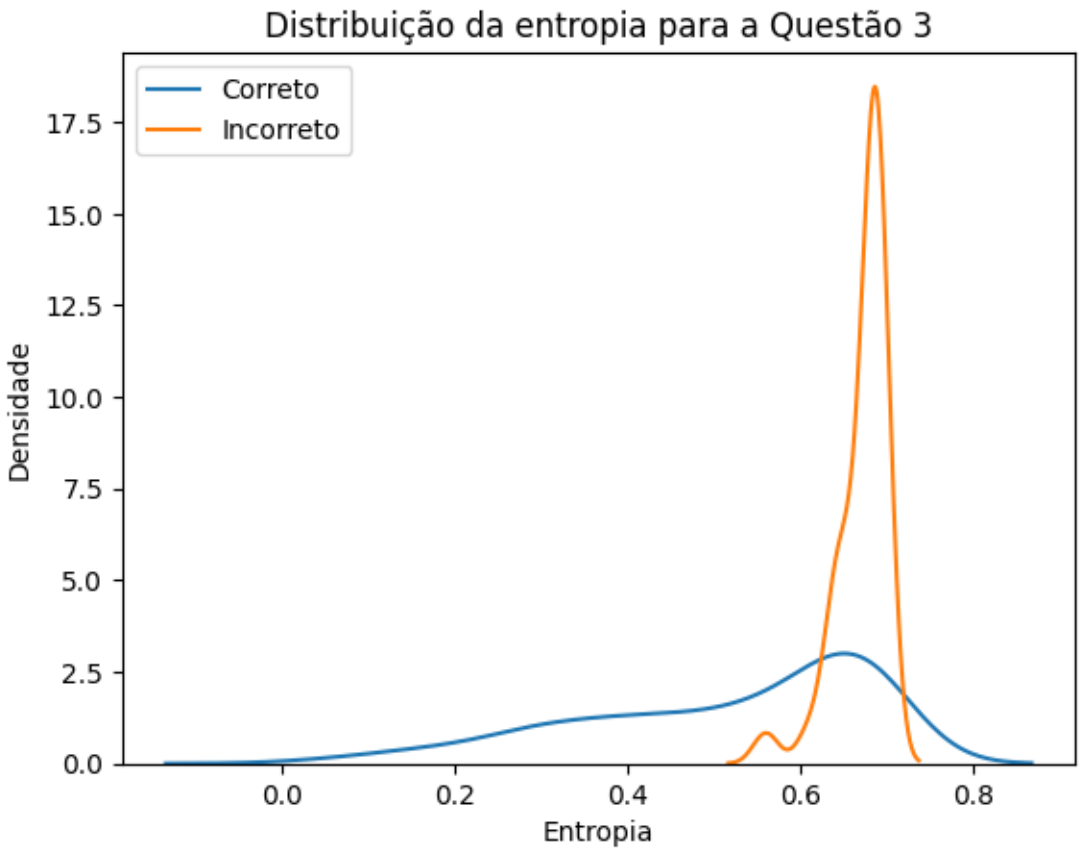
```
        True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True, False,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True, False])

Classificação Questão 3:
 tensor([ True,  True, False,  True,  True,  True,  True,  True,  True,  True,
         True, False,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True,  True,  True,  True,  True,  True,  True,  True,
        False,  True,  True, False,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True, False,
        False,  True, False,  True,  True, False,  True,  True,  True,  True,
         True,  True,  True,  True, False,  True, False,  True,  True,  True,
         True,  True,  True, False,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True, False,  True,
         True, False,  True,  True,  True,  True,  True,  True,  True, False,
         True, False,  True,  True, False,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True, False,  True, False,
         True,  True, False,  True, False,  True,  True,  True,  True,  True,
         True,  True,  True, False,  True,  True, False, False,  True,  True,
         True,  True, False,  True,  True, False,  True,  True,  True,  True,
         True,  True,  True,  True, False,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True, False,  True,  True, False,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True, False, False,
         True,  True,  True,  True,  True,  True,  True,  True,  True, False,
         True,  True,  True,  True,  True,  True,  True, False,  True,  True,
         True,  True,  True,  True,  True,  True,  True,  True,  True,  True,
         True,  True,  True,  True,  True,  True,  True])
```

[10]:
```python
predicoes = [y_pred_1, y_pred_2, y_pred_3]
y_test = ytest

for i in range(3):
    y_pred = predicoes[i]

    # Indices de predicoes corretas e incorretas
    index_correto = (y_pred > 0.5) == y_test
    index_incorreto = (y_pred > 0.5) != y_test

    # Valores de predicoes corretas e incorretas
    y_pred_correto = y_pred[index_correto]
```

```
    y_pred_incorreto = y_pred[index_incorreto]

    def log_nan_handler(x):
        # calcula o logaritmo de um tensor de entrada no domínio real,␣
↪garantindo que não haja valores NaN ou infinitos
        return torch.log(torch.clamp(x, min=torch.finfo(x.dtype).tiny))

    # Calcula a entropia de predicoes corretas e incorretas
    entropia_correta = -y_pred_correto * log_nan_handler(y_pred_correto) - (1 -␣
↪y_pred_correto) * log_nan_handler(1 - y_pred_correto)
    entropia_incorreta = -y_pred_incorreto * log_nan_handler(y_pred_incorreto) -␣
↪(1 - y_pred_incorreto) * log_nan_handler(1 - y_pred_incorreto)

    sns.kdeplot(entropia_correta.detach(), label='Correto')
    sns.kdeplot(entropia_incorreta.detach(), label='Incorreto')
    plt.title(f'Distribuição da entropia para a Questão {i+1}')
    plt.xlabel('Entropia')
    plt.ylabel('Densidade')

    # Legend the correct and incorrect predictions
    plt.legend()
    plt.show()
```




Distribuição da entropia para a Questão 2
Correto
Incorreto
Densidade
175
150
125
100
75
50
25
0
0.63
0.64
0.65
0.66
0.67
0.68
0.69
0.70
0.71
Entropia



Questão 6 Para a questão 1, é possível ver que a implementação da estimativa de máximo a priori foi bem adequada, dada sua acurácia de aproximadamente 95%, ou seja, a escolha da priori foi bem executada. A aproximação de Laplace da questão 2 teve uma acurácia ainda melhor que a implementação direta da priori, ainda que a diferença tenha sido pequena. Isso pode ser dado a não-necessidade de calcular a integral de Bayes. Pode-se notar também que a acurácia da questão 3 foi a menor dentre as implementações. Isso pode ser devido a distribuição variacional utilizada nesse caso.

Dado que a entropia mede o grau de incerteza inerente aos possíveis resultados para uma variável, temos que a incerteza é bem maior para os modelos de aproximação da regressão, já que o valor da moda das entropias é bem maior para as classificações incorretas do que as corretas, com exceção da estimativa de máximo a posteriori não aproximada.

2 Exercícios de "papel e caneta"

1. Derive a fórmula para a divergência KL entre duas distribuições Gaussianas univariadas, i.e., $D_{\mathrm{KL}}\left(\mathcal{N}(\mu_1, \sigma_1^2) \| \mathcal{N}(\mu_2, \sigma_2^2)\right)$;
2. Suponha que P é a família das distribuições categórica com suporte em $\{1, \ldots, L\}$. Qual $p \in P$ possui maior entropia?

3. Use a desigualdade de Jensen para mostrar que a divergência KL é não-negativa.
4. Derive a aproximação de Laplace para a distribuição Beta(α, β). Mostre uma fórmula para valores genéricos $\alpha, \beta > 1$ e a instancie para $\alpha = \beta = 2$.
5. Derive a posteriori para o modelo Bayesiano com verossimilhança Categórica e priori Dirichlet, i.e.:

$$y_1, \ldots, y_N \sim Cat(\theta)$$
$$\theta \sim Dirichlet(\alpha)$$

onde θ e α são vetores L-dimensionais.

2.0.1 Soluções:

1.

Temos a seguinte fórmula para a divergência KL:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\Big[\log \frac{P(x)}{Q(x)}\Big] = \sum P(x) \log \frac{P(x)}{Q(x)} = \int P(x) \log \frac{P(x)}{Q(x)} dx =$$
$$= -\int P(x) \log Q(x) dx + \int P(x) \log P(x) dx$$

Para derivar a sua fórmula entre duas distribuições Gaussianas univariadas, podemos fazer:

$$\int P(x) \log P(x) dx = -\frac{1}{2}(1 + \log 2\pi\sigma_1^2)$$
$$-\int P(x) \log Q(x) dx = -\int P(x) \log \frac{1}{(2\pi\sigma_2^2)(1/2)} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}} dx$$

Agora, podemos separar a equação de $-\int P(x) \log Q(x) dx$ da seguinte maneira:

$$\frac{1}{2}\log(2\pi\sigma_2^2) - \int P(x) \log e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}} dx = \frac{1}{2}\log(2\pi\sigma_2^2) - \int P(x)\Big(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\Big) dx =$$
$$= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\int P(x)x^2 dx - \int P(x) 2x\mu_2 dx + \int P(x)\mu_2^2 dx}{2\sigma_2^2} = \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + \mu_1^2 - 2\mu_1\mu_2 + \mu_2^2}{2\sigma_2^2} =$$
$$= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}$$

Juntando as equações:

$$D_{KL}(P||Q) = -\int P(x)\log Q(x)dx + \int P(x)\log P(x)dx = \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log 2\pi\sigma_1^2) =$$
$$= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

KL divergence between two univariate Gaussians (https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians)

KL Divergence between 2 Gaussian Distributions (https://mr-easy.github.io/2020-04-16-kl-divergence-between-2-gaussian-distributions/)

2.

Temos a seguinte fórmula para a entropia:

$$H(X) = -\sum_{i=1}^{n} P(x_i)\log P(x_i)$$

Nesse caso, a entropia da distribuição categórica é dada por $H(p_1, \ldots, p_L) = -\sum_{i=1}^{L} p_i \log p_i$, dado que $\sum_{i=1}^{L} p_i = 1$

Sendo P uma distribuição uniforme discreta, a distribuição de máxima entropia é dada por $p_i = \frac{1}{L}$, $i = 1, \ldots, L$.

Com isso, calculamos a entropia:

$$H(p) = -\sum_{i=1}^{L} p_i \log(p_i) = -\sum_{i=1}^{L} \frac{1}{L}\log(\frac{1}{L}) = L\frac{1}{L}\log(L) = \log(L)$$

Maximum entropy probability distribution (https://en.wikipedia.org/wiki/Maximum_entropy_probability_dis

Maximum Entropy Distributions (https://bjlkeng.github.io/posts/maximum-entropy-distributions/)

Entropy of a uniform distribution (https://math.stackexchange.com/questions/1156404/entropy-of-a-uniform-distribution)

3.

Primeiramente, temos que a desigualdade de Jensen é definida como:

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$$

Temos também que a divergência KL nos diz o quão bem uma distribuição Q aproxima a distribuição P calculando a entropia cruzada menos a entropia:

$$D_{KL}(P||Q) = H(P, Q) - H(P),$$

onde

$$H(P,Q) = \mathbb{E}_{x\sim P}[-\log Q(x)]$$
$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$$

Assim:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\Big[\log \frac{P(x)}{Q(x)}\Big] = \sum_x P(x)\Big[\log \frac{P(x)}{Q(x)}\Big]$$

Agora, para mostrar que a divergência é não-negativa, podemos mostrar que $-D_{KL}(P||Q) \leq 0$, e portanto $D_{KL}(P||Q) \geq 0$:

$$-D_{KL}(P||Q) = -\sum_x P(x)\Big[\log \frac{P(x)}{Q(x)}\Big] = \sum_x P(x)\Big[\log \frac{Q(x)}{P(x)}\Big] \leq \sum_x P(x)\Big(\frac{Q(x)}{P(x)} - 1\Big) = \sum_x Q(x) - \sum_x P(x) = 1 - 1$$

Tendo isso, concluímos que a divergência KL é não-negativa utilizando a desigualdade de Jensen.

Jensen's inequality (https://en.wikipedia.org/wiki/Jensen's_inequality)

KL Divergence Demystified (https://kikaben.com/kl-divergence-demystified/#:~:text=3.-,The%20KL%20divergence%20is%20non%2Dnegative,%E2%81%A1%20P%20Q%20%3D%20log%20%E2%81%A

Why KL divergence is non-negative? (https://stats.stackexchange.com/questions/335197/why-kl-divergence-is-non-negative)

4. Queremos encontrar a aproximação $q(\theta)$ para a distribuição Beta, com a seguinte fórmula para q:

$$q(\theta) = \mathcal{N}\left(\theta|\mu = m, \Sigma = H^{-1}\right),$$

onde $m = \arg\max_{\theta\in\Theta} p(\theta|\mathcal{D})$ e $H = \nabla^2_\theta - \log p(\theta|\mathcal{D})$.

Dado que estamos usando a distribuição Beta, podemos usar de forma direta que a média é igual a moda m, que por sua vez é igual a $\dfrac{\alpha - 1}{\alpha + \beta - 2}$.

Agora, temos $q(x) \propto x^{(\alpha-1)}(1-x)^{(\beta-1)}$, e $\Sigma = H^{-1} = -\dfrac{1}{\dfrac{d^2 \log q(x)}{dx^2}}$

Assim:

$$\log q(x) = \log(x^{(\alpha-1)}(1-x)^{(\beta-1)}) = \log(x)^{\alpha-1} + \log(x-1)^{\beta-1} = (\alpha-1)\log(x) + (\beta-1)\log(x-1) \Rightarrow$$
$$\Rightarrow \frac{d\log q(x)}{dx} = (\alpha-1)\frac{d\log(x)}{dx} + (\beta-1)\frac{d\log(x-1)}{\partial x} = \frac{(\alpha-1)}{x} + \frac{(\beta-1)}{x-1} \Rightarrow$$
$$\Rightarrow \frac{d^2\log q(x)}{dx^2} = (\alpha-1)\frac{d(1/x)}{dx} + (\beta-1)\frac{d(1/(x-1))}{dx} = -\frac{(\alpha-1)}{x^2} - \frac{(\beta-1)}{(x-1)^2} \Rightarrow$$
$$\Rightarrow \Sigma = H^{-1} = -\frac{1}{\dfrac{(\alpha-1)}{x^2} - \dfrac{(\beta-1)}{(x-1)^2}} = \frac{x^2(x-1)^2}{(\alpha-1)(x-1)^2 + (\beta-1)x^2}$$

Com isso, devemos ter que $\alpha \neq 1$ e $\beta \neq 1$, e também que a função não terá máximo caso $0 < \alpha, \beta < 1$ por não ser convexa nesse intervalo. Sendo assim, temos uma fórmula para a distribuição Beta(α, β) para $\alpha, \beta > 1$ genéricos.

No caso $\alpha = \beta = 2$:

$$m = \frac{\alpha - 1}{\alpha + \beta - 2} = \frac{2-1}{2+2-2} = \frac{1}{2}$$
$$\Sigma = \frac{x^2(x-1)^2}{(2-1)(x-1)^2 + (2-1)x^2} = \frac{x^2(x-1)^2}{(x-1)^2 + x^2}$$

Usando $x = m$:

$$\Sigma = \frac{(1/2)^2((1/2)-1)^2}{((1/2)-1)^2 + (1/2)^2} = \frac{(1/4)(1/4)}{(1/4)+(1/4)} = \frac{1/16}{1/2} = \frac{1}{8}$$

Assim:

$$Beta(2,2) = \mathcal{N}\left(\theta | \frac{1}{2}, \frac{1}{8}\right)$$

Laplace's approximation (https://en.wikipedia.org/wiki/Laplace%27s_approximation)

Beta distribution (https://en.wikipedia.org/wiki/Beta_distribution)

The beta density, Bayes, Laplace, and P´olya (http://www.cs.hunter.cuny.edu/~saad/courses/bayes/notes/not

5. Primeiramente, temos a distribuição de Dirichlet (Dir$(\alpha_1, \ldots, \alpha_k)$):

$$f(x_1, \ldots, x_k; \alpha_1, \ldots, \alpha_k) = \frac{\Gamma(\Sigma_{i=1}^k \alpha_i)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k x_i^{\alpha_i - 1}$$

Agora, a distribuição Categórica:

$$f(x = i|p) = p_i$$
$$f(x|p) = \prod_{i=1}^k p_i^{[x=i]}$$

Temos a fórmula da posteriori:

$$f(\theta|D) = \frac{f(\theta, D)}{f(D)} \propto f(\theta, D) = f(\theta|\alpha) \prod_{y_i \in D} f(y_i|\theta),$$

onde D é o nosso dataset e $y_i \in D$ são amostras retiradas iid da distribuição $f(y; \theta)$.

Agora, usando o enunciado:

$$y_1, \ldots, y_N \sim Cat(\theta)$$
$$\theta \sim Dirichlet(\alpha)$$

$$f(\theta|D) \propto f(\theta, D) = f(\theta|\alpha) \prod_{y_i \in D} f(y_i|\theta)$$
$$\propto \prod_{j=1}^{N} \theta_j^{\alpha_j - 1} \prod_{y_i \in D} \prod_{j=1}^{N} \theta_j^{\mathbb{1}\{y_i = j\}} = \prod_{j=1}^{N} \theta_j^{\alpha_j - 1 + \sum_{y_i \in D} \mathbb{1}\{y_i = j\}}$$

Com isso, chegamos que a densidade da posteriori será a de uma distribuição de Dirichlet com $\alpha'_j = \alpha_j + \sum_{y_i \in D} \mathbb{1}\{y_i = j\}$, ou seja, $f(\theta|D) = Dir(\alpha')$

Dirichlet distribution (https://en.wikipedia.org/wiki/Dirichlet_distribution)

Categorical distribution (https://en.wikipedia.org/wiki/Categorical_distribution)

The Dirichlet-Multinomial and Dirichlet-Categorical models for Bayesian inference (https://stephentu.github.io/writeups/dirichlet-conjugate-prior.pdf)

[ ]: