

Exercício 2

Aprendizado de Máquina

Pedro Henrique Coterli
20 de junho de 2025

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 L^AT_EX externo, o inclua no final da versão pdf do ‘ipynb’— submetendo um único pdf.

1 Cross-validation

- O código abaixo carrega o banco de dados *California housing*. Divida o banco de dados em treino, teste e validação. Use o conjunto de validação para escolher o coeficiente de regularização c para um modelo de regressão linear com penalização L_2 . Use a fórmula analítica para estimar os pesos do modelo de regressão. Plote os MSE no conjunto de trieno e validação em função de c . Comente o resultado. Avalie a performance do modelo ótimo no conjunto de teste e também comente.

```
import numpy as np
from sklearn.datasets import fetch_california_housing

SEED = 44
np.random.seed(SEED)

X, y = fetch_california_housing(return_X_y=True)
```

```
# Seu código aqui
from sklearn.model_selection import train_test_split

X_train_and_val, X_test, y_train_and_val, y_test = train_test_split(X, y,
    test_size = 0.15, random_state = SEED)
X_train, X_val, y_train, y_val = train_test_split(X_train_and_val, y,
    test_size = 3/17, random_state = SEED)
```

```
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)

mses_train = {}
```

```

mses_val = {}

XtX = X_train.T @ X_train
Xty = X_train.T @ y_train
I = np.eye(XtX.shape[0])

c = 0

beta = np.linalg.solve(XtX + c*I, Xty)
y_pred_train = X_train @ beta
current_mse_train = mse(y_train, y_pred_train)
y_pred_val = X_val @ beta
current_mse_val = mse(y_val, y_pred_val)

mses_train[c] = current_mse_train
mses_val[c] = current_mse_val

for i in range(11):
    c = 2**i
    beta = np.linalg.solve(XtX + c*I, Xty)
    y_pred_train = X_train @ beta
    current_mse_train = mse(y_train, y_pred_train)
    y_pred_val = X_val @ beta
    current_mse_val = mse(y_val, y_pred_val)

    mses_train[c] = current_mse_train
    mses_val[c] = current_mse_val

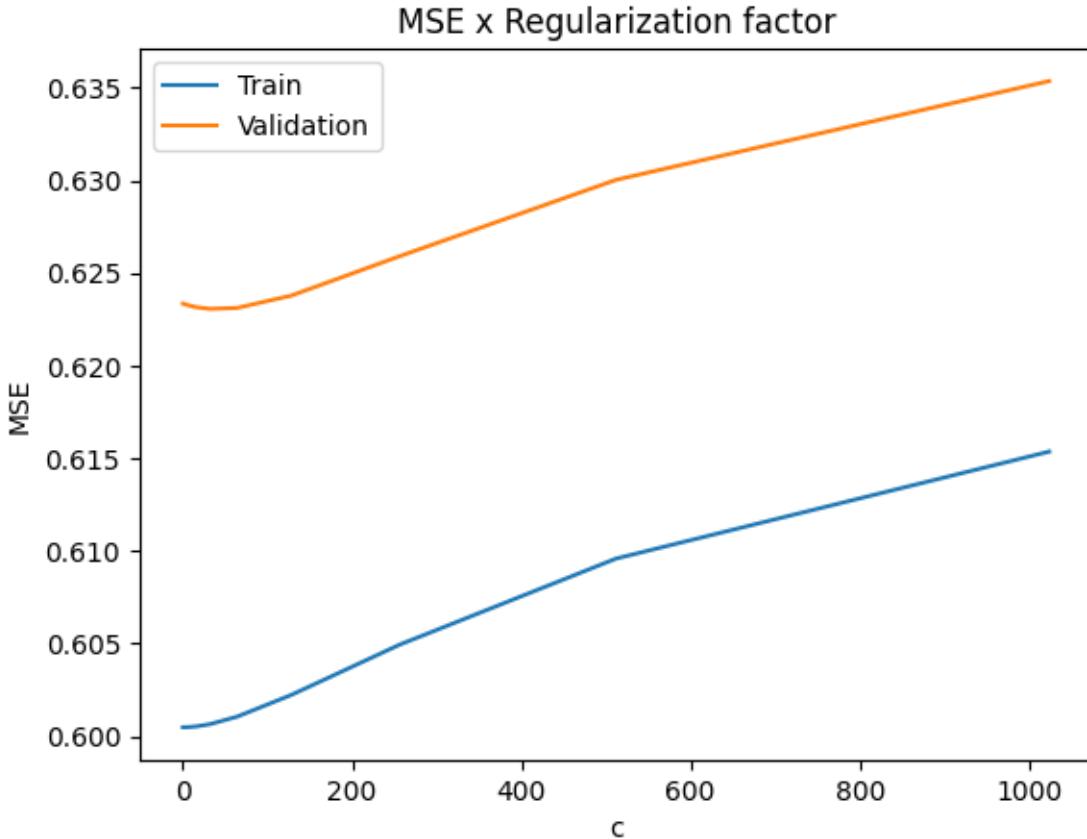
```

```

import matplotlib.pyplot as plt

plt.plot(mses_train.keys(), mses_train.values(), label = "Train")
plt.plot(mses_val.keys(), mses_val.values(), label = "Validation")
plt.title("MSE x Regularization factor")
plt.xlabel("c")
plt.ylabel("MSE")
plt.legend()
plt.show()

```



Resposta 1: Como é possível observar, com valores pequenos para c , o erro no conjunto de treino é o menor encontrado, mas o no conjunto de validação é superior ao ótimo. Isso ocorre justamente porque, sem regularização, o modelo pode se ajustar mais perfeitamente aos dados no conjunto de treino, o que leva a um fraco poder de generalização, resultando em um erro maior para dados de fora da amostra de treino. Por outro lado, aumentando um pouco esse coeficiente, o erro no conjunto de validação começa a diminuir graças à “piora” no ajuste do modelo ao conjunto de treino, permitindo a ele generalizar melhor. No entanto, à medida que esse valor cresce, ambos os erros aumentam em conjunto, já que o modelo acaba não sendo mais capaz de se ajustar nem às características desejáveis presentes no dado.

```
best_c = min(mses_val.items(), key = lambda item: item[1])[0]
print("Melhor valor de c:", best_c)
```

Melhor valor de c: 32

```
XtX = X_train_and_val.T @ X_train_and_val
I = np.eye(XtX.shape[0])
Xty = X_train_and_val.T @ y_train_and_val

final_beta = np.linalg.solve(XtX + best_c * I, Xty)
```

```

y_pred_test = X_test @ final_beta
mse_test = mse(y_test, y_pred_test)

print("MSE no conjunto de teste:", mse_test)

```

MSE no conjunto de teste: 0.6079416453456845

Resposta 2: Observa-se que esse modelo obteve um resultado melhor no conjunto de teste que o obtido no conjunto de validação. A razão disso provavelmente é a utilização de uma quantidade maior de dados para o treinamento, já que esse modelo foi treinado não apenas com os dados de treino, mas com os de validação também.

2. Implemente 5-fold *nested cross-validation* para escolher entre os métodos k -NN e regressão linear com regularização L_2 (similar ao exercício acima). Considere $k \in \{1, 2, 3, 4, 5\}$ e $c \in \{0, 1, 10, 100\}$. Use o mesmo banco de dados do último exercício e comente o resultado. Em média, qual valor de hiperparametro resulta na melhor performance para o método escolhido (use 5-fold cross validation regular para isso)?

Obs.: para simplificar sua vida, use o k -NN para regressão do scikit-learning com distância euclidiana.

Obs. 2: para mais informações sobre o K -fold *nested cross-validation*, recomendamos esses materiais:

- [Algoritmo e breve explicação](#): a autora apresenta uma boa explicação do assunto acompanhada de uma descrição do algoritmo;
- [Ilustrações e explicação acompanhada de código](#): ajuda a visualizar melhor o que é *nested cross-validation*; vale lembrar que seu código, diferente do dos exemplos desse link, não deve utilizar scikit-learn para implementar a *cross-validation*.

```

# Seu código aqui
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler

# Função para separar o conjunto de dados
def split(array, slice_index, total_slices):
    parts = np.array_split(array, total_slices)
    slice = parts[slice_index]
    rest = [parts[i] for i in range(total_slices) if i != slice_index]
    rest = np.concatenate(rest)

    return rest, slice

# Embaralhando os dados
data = np.hstack((X, y.reshape(-1, 1)))
np.random.shuffle(data)

X_shuffled = data[:, :-1]
y_shuffled = data[:, -1]

# Listas com os erros em cada conjunto de teste
errors_test_knn = []

```

```

errors_test_lr = []

scaler = StandardScaler()

# Para cada grupo externo...
for i in range(5):
    # Define os conjuntos de treino + validação e teste
    X_train_and_val, X_test = split(X_shuffled, i, 5)
    y_train_and_val, y_test = split(y_shuffled, i, 5)

    # Dicionários para armazenar os erros em cada conjunto de validação de cada hiperparâmetro
    errors_val_knn = {1: [], 2: [], 3: [], 4: [], 5: []}
    errors_val_lr = {0: [], 1: [], 10: [], 100: []}

    # Para cada grupo interno...
    for j in range(5):
        # Define os conjuntos de treino e validação
        X_train, X_val = split(X_train_and_val, j, 5)
        y_train, y_val = split(y_train_and_val, j, 5)

        # Para cada valor do hiperparâmetro k...
        for k in range(1, 6):
            # Treina o k-NN no conjunto de treino e avalia no de validação
            X_train_scaled = scaler.fit_transform(X_train)
            X_val_scaled = scaler.transform(X_val)
            knn_regressor = KNeighborsRegressor(n_neighbors = k)
            knn_regressor.fit(X_train_scaled, y_train)
            y_pred_val = knn_regressor.predict(X_val_scaled)
            current_mse = mse(y_val, y_pred_val)
            # Salva na lista correspondente
            errors_val_knn[k].append(current_mse)

        # Para cada valor do hiperparâmetro c...
        for c in [0, 1, 10, 100]:
            # Treina a regressão linear no conjunto de treino e avalia no de validação
            beta = np.linalg.solve(X_train.T @ X_train + c * np.eye(X_train.shape[1]),
                                  X_train.T @ y_train)
            y_pred_val = X_val @ beta
            current_mse = mse(y_val, y_pred_val)
            # Salva na lista correspondente
            errors_val_lr[c].append(current_mse)

    # Calcula o erro médio nos conjuntos de validação para cada valor de hiperparâmetro

```

```

averages_val_knn = {i: np.mean(j) for i, j in errors_val_knn.items()}
averages_val_lr = {i: np.mean(j) for i, j in errors_val_lr.items()}

# Pega os valores de hiperparâmetros com menor erro médio no conjunto de
→validação
best_k = min(averages_val_knn.items(), key = lambda item: item[1])[0]
best_c = min(averages_val_lr.items(), key = lambda item: item[1])[0]

# Treina o k-NN no conjunto de treino + validação com esse melhor parâmetro
X_train_and_val_scaled = scaler.fit_transform(X_train_and_val)
X_test_scaled = scaler.transform(X_test)
knn_regressor = KNeighborsRegressor(n_neighbors = best_k)
knn_regressor.fit(X_train_and_val_scaled, y_train_and_val)
# Avalia no conjunto de teste
y_pred_test = knn_regressor.predict(X_test_scaled)
current_mse = mse(y_test, y_pred_test)
# Salva na lista
errors_test_knn.append((best_k, current_mse))

# Treina a regressão no conjunto de treino + validação com esse melhor
→parâmetro
beta = np.linalg.solve(X_train_and_val.T @ X_train_and_val + best_c * np.
→eye(X_train_and_val.shape[1]),
                      X_train_and_val.T @ y_train_and_val)
# Avalia no conjunto de teste
y_pred_test = X_test @ beta
current_mse = mse(y_test, y_pred_test)
# Salva na lista
errors_test_lr.append((best_c, current_mse))

# Calcula o erro médio nos conjuntos de teste
average_test_knn = np.mean([i[1] for i in errors_test_knn])
average_test_lr = np.mean([i[1] for i in errors_test_lr])

print("Erro médio do k-NN: ", average_test_knn)
print("Erro médio da regressão linear:", average_test_lr, "\n")

# Seleciona o melhor modelo
if average_test_knn < average_test_lr:
    best_model = "k-NN"
    errors_test_best_model = errors_test_knn
    best_hyperparam = "k"
else:
    best_model = "Regressão linear"
    errors_test_best_model = errors_test_lr
    best_hyperparam = "c"

```

```

# Seleciona o melhor valor de hiperparâmetro como a moda dos selecionados no
    ↵processo
best_hyperparam_value = np.argmax(np.bincount([i[0] for i in
    ↵errors_test_best_model]))
print("Melhor modelo:      ", best_model)
print(f"Melhor hiperparâmetro {best_hyperparam}:", best_hyperparam_value)

```

Erro médio do k-NN: 0.41201272832403096

Erro médio da regressão linear: 0.6149827524373794

Melhor modelo: k-NN

Melhor hiperparâmetro k: 5

Resposta: Nota-se que o modelo k-NN obteve um resultado superior ao modelo linear. Isso pode ter sido causado por uma não linearidade na relação entre as covariáveis utilizadas, fazendo o modelo de regressão linear não se ajustar adequadamente, e/ou pela existência de fronteiras de decisão mais complexas, situação em que o modelo k-NN geralmente obtém performance superior.

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 L^AT_EX externo, o inclua no final da versão pdf do ‘ipynb’ — submetendo um único pdf.

2 Redes Neurais

1. O código abaixo baixa e carrega o banco de dados *CIFAR10*, que contém diversas imagens RGB de **10 classes distintas**. Além disso, os dados já estão separados em um objeto que provê batches de treino, teste e validação.

Também provemos um exemplo de como definir um modelo em *PyTorch* (você é livre para modificá-lo e facilitar sua vida), bem como um exemplo de como carregar batches. **Treine uma CNN para resolver o problema — fazendo uma busca de hiper-parâmetros razoável baseada no erro de validação.**

Eu espero uma busca de hiper-parâmetros razoável (e.g., definindo um grid de hiper-parâmetros). Eu sugiro utilizar early stopping e algum scheduler de learning rate (e.g., ReduceLROnPlateau). Como de costume em deep learning — devido ao tamanho dos bancos ser razoável —, não de faça coisas como cross-validation.

Consulte a documentação do *PyTorch* para entender o que cada função na definição da CNN faz. Reporte os resultados do modelo escolhido no conjunto de teste, bem como suas curvas de aprendizado.

Além disso, mostre como os dados de entrada são transformados ao longo das camadas do modelo ótimo (plota as figuras intermediárias para alguns inputs da sua escolha) e comente.

```

import numpy as np
import matplotlib.pyplot as plt

```

```

from sklearn.datasets import load_breast_cancer
import seaborn as sns
import io
import base64

import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import random_split, DataLoader
import tqdm

```

```

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)

# The batch size
batch_size = 2048

transform_train = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

train_set = torchvision.datasets.CIFAR10(
    root='./data', train=True, download=True,
    transform=transform_train,
)
test_set = torchvision.datasets.CIFAR10(
    root='./data', train=False, download=True,
    transform=transform_test
)

train_loader = DataLoader(
    train_set, batch_size=batch_size, shuffle=True,
    num_workers=2
)
test_loader = DataLoader(
    test_set, batch_size=batch_size, shuffle=True,
    num_workers=2
)

classes = (

```

```

    'plane', 'car', 'bird', 'cat', 'deer',
    'dog', 'frog', 'horse', 'ship', 'truck'
)

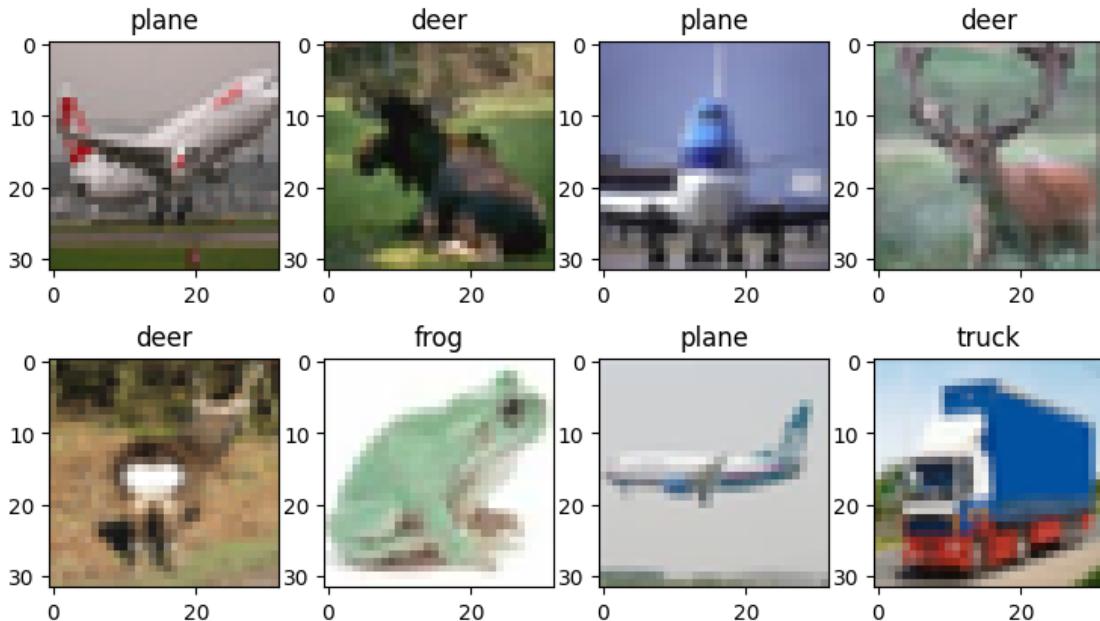
n_img = 8

f, ax = plt.subplots(2, 4, figsize=(9,5))
dataiter = iter(test_loader)
images, labels = next(dataiter)

for i in range(n_img):
    npimg = images[i]/2 + 0.5
    npimg = npimg.numpy()
    ax[i%2,i//2].imshow(np.transpose(npimg, (1, 2, 0)))
    ax[i%2,i//2].set_title(classes[labels[i]])

```

cpu



```

from torch.utils.data import Subset
import json
import os

# Separando o conjunto de treino anterior em treino e validação
train_indices, val_indices = train_test_split(range(len(train_set)), test_size = 0.2, random_state = SEED)

```

```

X_train = Subset(train_set, train_indices)
X_val = Subset(train_set, val_indices)

# Novos loaders dos conjuntos de treino e validação
train_loader = DataLoader(X_train, batch_size = batch_size, shuffle = True,↳
↳num_workers = 2)
val_loader = DataLoader(X_val, batch_size = batch_size, shuffle = True,↳
↳num_workers = 2)

# Classe da rede neural convolucional
class CNN(nn.Module):
    # Recebe o número de camadas convolucionais e o número de neurônios nas↳
↳camadas lineares desejados
    def __init__(self, n_convs, n_neur_linear):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.fc1 = nn.LazyLinear(n_neur_linear[0])
        self.fc2 = nn.Linear(n_neur_linear[0], n_neur_linear[1])
        self.fc3 = nn.Linear(n_neur_linear[1], 10)
        self.n_convs = n_convs

    def forward(self, x, return_intermediate = False):
        # Saídas intermediárias das camadas de convolução
        intermediate_outputs = {}

        x = F.relu(self.conv1(x))
        if return_intermediate:
            intermediate_outputs["conv1"] = x.clone()
        x = self.pool(x)
        if return_intermediate:
            intermediate_outputs["pool1"] = x.clone()

        # Se quiser mais de uma camada convolucional, usa a segunda camada
        if self.n_convs > 1:
            x = F.relu(self.conv2(x))
            if return_intermediate:
                intermediate_outputs["conv2"] = x.clone()
            x = self.pool(x)
            if return_intermediate:
                intermediate_outputs["pool2"] = x.clone()

        # Se quiser mais de duas camadas convolucionais, usa a terceira↳
↳camada
        if self.n_convs > 2:

```

```

        x = F.relu(self.conv3(x))
        if return_intermediate:
            intermediate_outputs["conv3"] = x.clone()

        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        if return_intermediate:
            return x, intermediate_outputs
        return x

criterion = nn.CrossEntropyLoss()
num_epochs = 50

# Número de épocas de tolerância para o early stopping
patience_early_stopping = 5
# Variação mínima exigida no erro da validação
delta = 0.001

# HIPERPARÂMETROS QUE SERÃO TESTADOS
# Taxa de aprendizado
lrs = [0.01, 0.001, 0.0001]
# Número de camadas convolucionais
ns_convs = [1, 2, 3]
# Número de neurônios nas camadas lineares
ns_neur_linear = [(240, 168), (120, 84), (60, 42)]

errors_lr = {}

# Se já não tiver o arquivo com os dados...
if not os.path.exists("errors.json"):
    # Para cada taxa de aprendizado...
    for lr in lrs:
        print(f"Iniciando com lr {lr}")
        errors_n_conv = {}
        # Para cada número de camadas convolucionais...
        for n_convs in ns_convs:
            print(f"- Iniciando com {n_convs} camadas de convolução")
            errors_n_neur_linear = {}
            # Para cada configuração de neurônios nas camadas lineares
            for n_neur_linear in ns_neur_linear:
                print(f"-- Iniciando com {n_neur_linear} neurônios")

                # Instancia a CNN correspondente
                cnn = CNN(n_convs, n_neur_linear).to(device)

```

```

# Instancia o otimizador correspondente
optimizer_cnn = torch.optim.Adam(cnn.parameters(), lr=lr)

# Scheduler para diminuir a taxa de aprendizado
scheduler = torch.optim.lr_scheduler.
↪ReduceLROnPlateau(optimizer_cnn,
                     mode = "min",
                     factor = 0.1,
                     patience = 5)

# Melhor erro de validação na rede atual (para o early stopping)
current_best_val_error = None
# Contador do early stopping (há quantas épocas está estagnado)
counter_early_stopping = 0

for epoch in range(num_epochs): # loop over the dataset
↪multiple times
    # Ativa o modo de treinamento
    cnn.train()
    # Erro total da época atual no treino
    running_loss = 0.0

    # Para cada batch de dados...
    for i, data in enumerate(train_loader):
        # Avança o modelo e avalia no conjunto de treino
        inputs, labels = data[0].to(device), data[1].to(device)
        optimizer_cnn.zero_grad()
        outputs = cnn(inputs)
        error = criterion(outputs, labels)
        error.backward()
        optimizer_cnn.step()
        running_loss += error.item()

    # Erro médio no treino
    average_train_error = running_loss / len(train_loader)

    # Ativa o modo de avaliação
    cnn.eval()
    # Erro total da época atual na validação
    error_val = 0

    # Avalia o desempenho no conjunto de validação
    with torch.no_grad():
        for data in val_loader:
            inputs, labels = data[0].to(device), data[1].
↪to(device)
            outputs = cnn(inputs)

```

```

        error = criterion(outputs, labels)
        error_val += error.item()

    # Erro médio na validação
    average_val_error = error_val / len(val_loader)

    print(f"--- Época [{epoch+1}/{num_epochs}], "
          f"Erro no treino: {average_train_error:.4f}, "
          f"Erro na validação: {average_val_error:.4f}")

    scheduler.step(average_val_error)

    if current_best_val_error is None:
        current_best_val_error = average_val_error
    # Se o erro atual não for tão melhor que o melhor até agora..
    ↪ .
    elif average_val_error > current_best_val_error - delta:
        # Aumenta o contador para o early stopping
        counter_early_stopping += 1
        # Se o contador superar a tolerância, encerra o
    ↪ treinamento
    if counter_early_stopping >= patience_early_stopping:
        break
    # Se for melhor, salva como o melhor e reseta o contador
    else:
        current_best_val_error = average_val_error
        counter_early_stopping = 0

    errors_n_neur_linear[n_neur_linear[0]] = current_best_val_error
    print(f"-- Finalizado com {n_neur_linear} neurônios")

    errors_n_conv[n_convs] = errors_n_neur_linear
    print(f"- Finalizado com {n_convs} camadas de convolução")

    errors_lr[lr] = errors_n_conv
    print(f"Finalizado com lr {lr}")

    with open("errors.json", "w", encoding = "utf-8") as f:
        json.dump(errors_lr, f, ensure_ascii = False, indent = 4)

```

```
# TREINANDO A REDE SELECIONADA NO CONJUNTO TREINO + VALIDAÇÃO
```

```

# Pegando os melhores hiperparâmetros
best_lr = None
best_n_convs = None
best_n_neur_linear = None

```

```

best_val_error = np.inf

with open("errors.json", "r", encoding = "utf-8") as f:
    errors_val = json.load(f)

for lr, errors_n_conv in errors_val.items():
    for n_conv, errors_n_neur_linear in errors_n_conv.items():
        for n_neur_linear, error in errors_n_neur_linear.items():
            if error < best_val_error:
                best_lr = float(lr)
                best_n_convs = int(n_conv)
                best_n_neur_linear = n_neur_linear
                best_val_error = error

for n_neur_linear in ns_neur_linear:
    if n_neur_linear[0] == int(best_n_neur_linear):
        best_n_neur_linear = n_neur_linear
        break

print("Melhor taxa de aprendizado:", best_lr)
print("Melhor número de camadas convolucionais:", best_n_convs)
print("Melhor número de neurônios:", best_n_neur_linear)

# Loader do conjunto treino + validação
train_and_val_loader = DataLoader(
    train_set, batch_size=batch_size, shuffle=True,
    num_workers=2
)

# Instancia a CNN com os melhores hiperparâmetros
best_cnn = CNN(best_n_convs, best_n_neur_linear).to(device)
# Instancia o otimizador com o melhor hiperparâmetro
optimizer_cnn = torch.optim.Adam(best_cnn.parameters(), lr=best_lr)

if not os.path.exists("model.pt"):
    # Histórico dos erros no treino + validação
    train_and_val_errors = []
    for epoch in range(50):
        # Ativa o modo de treinamento
        best_cnn.train()
        # Erro total da época atual no conjunto treino + validação
        running_loss = 0.0

        # Para cada batch de dados...
        for i, data in enumerate(train_and_val_loader):
            # Avança o modelo e avalia no conjunto de treino + validação
            inputs, labels = data[0].to(device), data[1].to(device)

```

```

        optimizer_cnn.zero_grad()
        outputs = best_cnn(inputs)
        error = criterion(outputs, labels)
        error.backward()
        optimizer_cnn.step()
        running_loss += error.item()

# Erro médio no treino
average_train_and_val_error = running_loss / len(train_and_val_loader)
train_and_val_errors.append(average_train_and_val_error)

print(f"--- Época [{epoch+1}/{num_epochs}], "
      f"Erro no treino + validação: {average_train_and_val_error:.4f}")

# Salvando o modelo
torch.save(best_cnn.state_dict(), "model.pt")
with open("train_and_val_errors.json", "w", encoding = "utf-8") as f:
    json.dump(train_and_val_errors, f, ensure_ascii = False, indent = 4)

```

Melhor taxa de aprendizado: 0.001
 Melhor número de camadas convolucionais: 2
 Melhor número de neurônios: (240, 168)

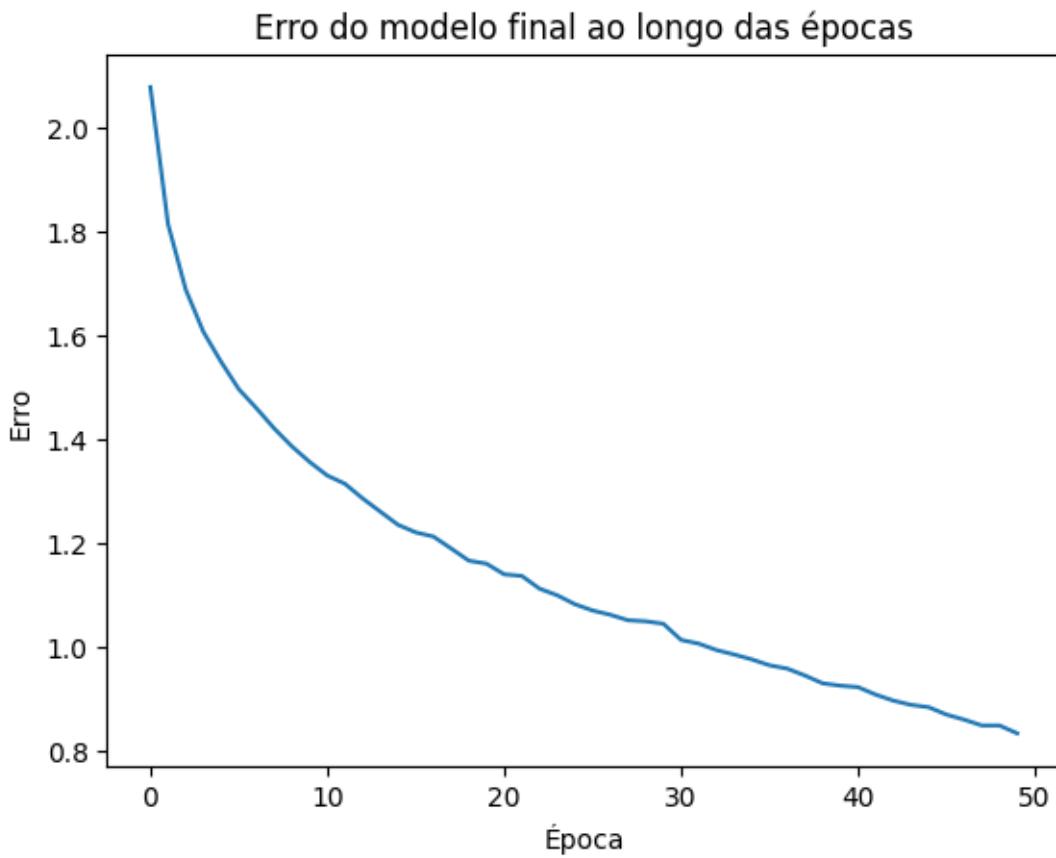
```

# Carregando o modelo
best_cnn.load_state_dict(torch.load("model.pt"))

# Carregando os dados de treinamento
with open("train_and_val_errors.json", "r", encoding = "utf-8") as f:
    train_and_val_errors = json.load(f)

plt.plot(train_and_val_errors)
plt.title("Erro do modelo final ao longo das épocas")
plt.xlabel("Época")
plt.ylabel("Erro")
plt.show()

```



```

# Ativando o modo de avaliação
best_cnn.eval()
# Número de previsões corretas
correct_test = 0
# Número total de previsões
total_test = 0
# Erro total no conjunto de teste
error_test = 0

with torch.no_grad():
    # Para cada batch do conjunto de teste...
    for data in test_loader:
        # Calcula e salva o erro
        images, labels = data[0].to(device), data[1].to(device)
        outputs = best_cnn(images)
        current_error = criterion(outputs, labels)
        error_test += current_error

    # Calcula as previsões

```

```

    _, predicted = torch.max(outputs.data, 1)
    total_test += labels.size(0)
    correct_test += (predicted == labels).sum().item()

# Erro médio no teste
average_error_test = error_test / len(test_loader)
# Acurácia no teste
accuracy_test = correct_test / total_test

print(f"Erro no conjunto de teste: {average_error_test:.4f}")
print(f"Acurácia no conjunto de teste: {accuracy_test}")

```

Erro no conjunto de teste: 1.0589
Acurácia no conjunto de teste: 0.6346

```

# Função para desenhar as saídas intermediárias da rede
def plot_intermediate_images(original_image, intermediate_images, image_label, ↵
    image_index = 0):
    # Ajusta a imagem para exibição
    np_original_image = original_image.cpu().numpy() / 2 + 0.5
    np_original_image = np.transpose(np_original_image, (1, 2, 0))

    plt.figure()
    plt.imshow(np_original_image)
    plt.title(f"Imagen original ({classes[image_label]})")
    plt.axis("off")
    plt.show()

    for layer_name, activation_tensor in intermediate_images.items():
        feature_maps = activation_tensor[image_index].cpu().numpy()

        # Determina o formato do grid de plots
        num_features_to_plot = min(feature_maps.shape[0], 16)
        grid_cols = 4
        grid_rows = (num_features_to_plot + grid_cols - 1) // grid_cols

        fig_conv, axes_conv = plt.subplots(grid_rows, grid_cols, figsize = ↵
            (grid_cols * 2.5, grid_rows * 2.5))
        fig_conv.suptitle(f"{layer_name}")

        # Plota cada imagem
        for i in range(num_features_to_plot):
            row = i // grid_cols
            col = i % grid_cols
            ax = axes_conv[row, col] if grid_rows > 1 else axes_conv[col]

            fm = feature_maps[i]

```

```

fm_normalized = (fm - fm.min()) / (fm.max() - fm.min() + 1e-8)
ax.imshow(fm_normalized, cmap = "gray")
ax.axis("off")
ax.set_title(f"Canal {i}")

for i in range(num_features_to_plot, grid_rows * grid_cols):
    row = i // grid_cols
    col = i % grid_cols
    ax = axes_conv[row, col] if grid_rows > 1 else axes_conv[col]
    ax.axis('off')

plt.tight_layout(rect = [0, 0.03, 1, 0.95])
plt.show()

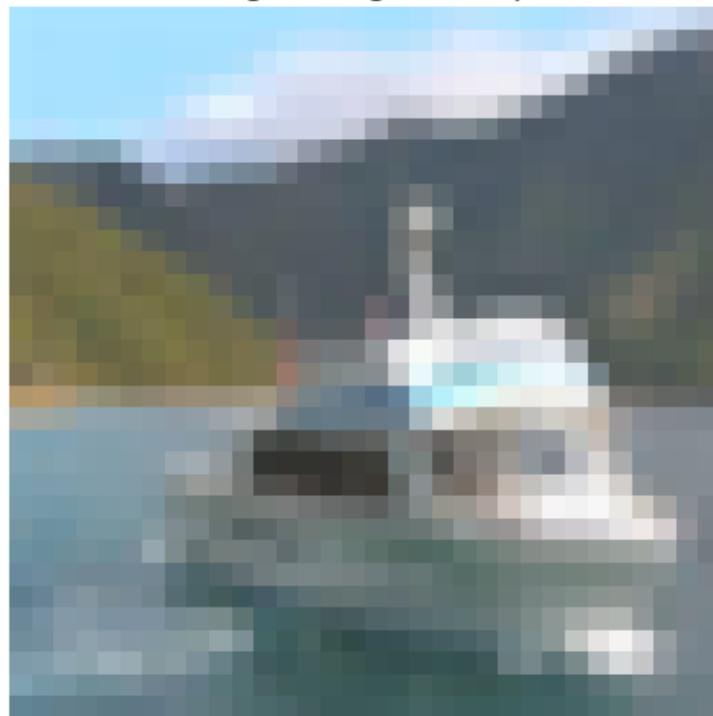
# Pegando algumas imagens do teste
dataiter = iter(test_loader)
images, labels = next(dataiter)

num_images_to_plot = 2
selected_images = images[:num_images_to_plot].to(device)
selected_labels = labels[:num_images_to_plot]

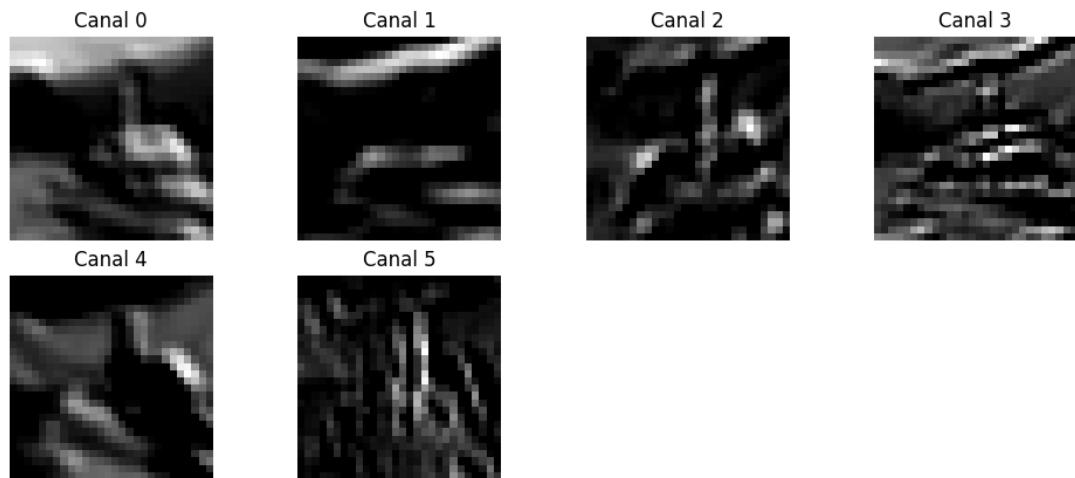
# Calculando suas fases intermediárias e plotando
with torch.no_grad():
    for i in range(num_images_to_plot):
        _, single_image_intermediate_activations = best_cnn(selected_images[i:i+1], return_intermediate = True)
        plot_intermediate_images(selected_images[i], single_image_intermediate_activations, selected_labels[i])

```

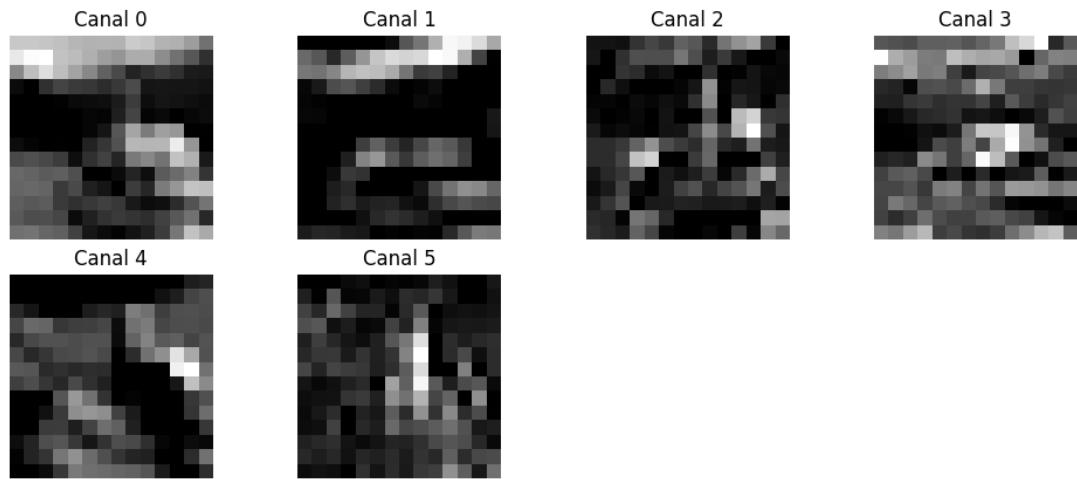
Imagen original (ship)



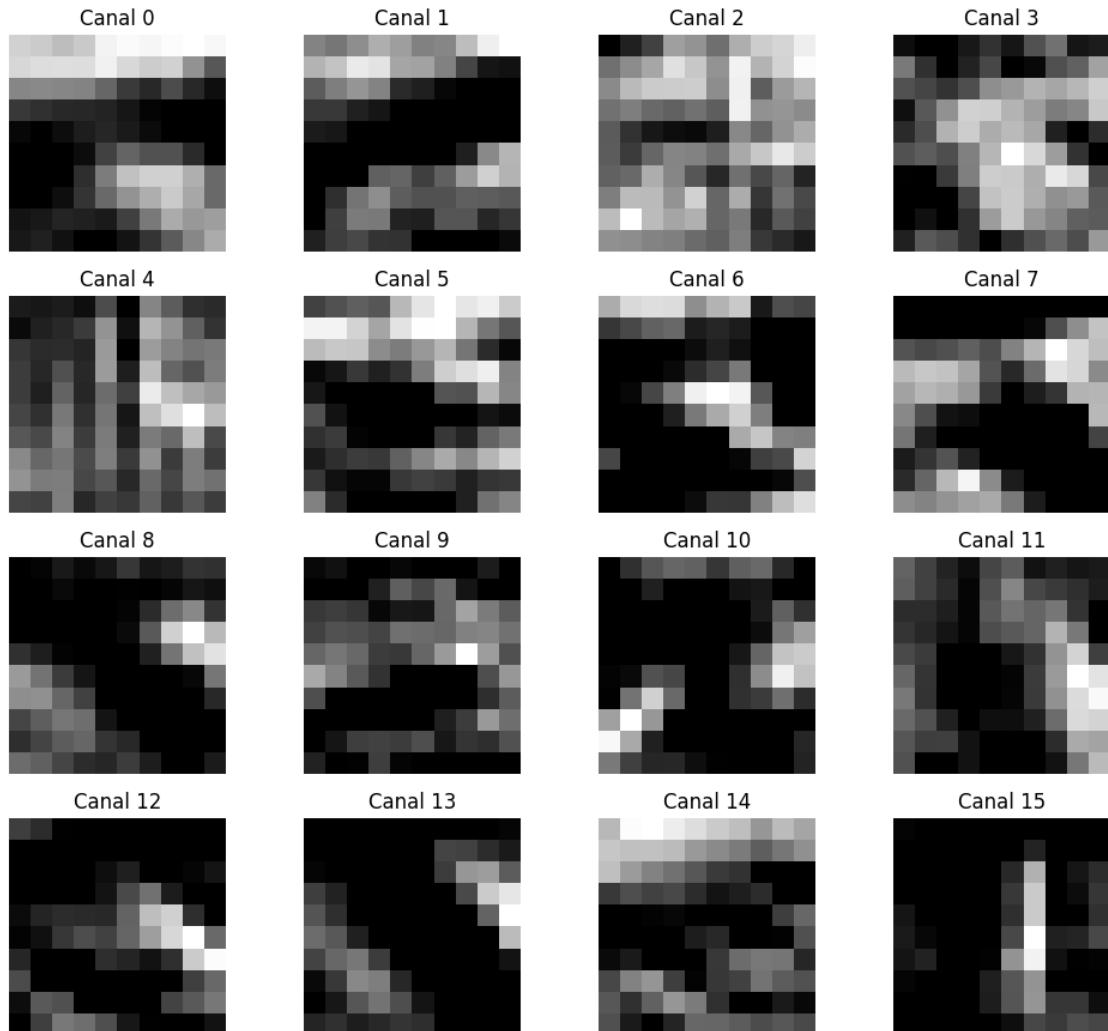
conv1



pool1



conv2



pool2

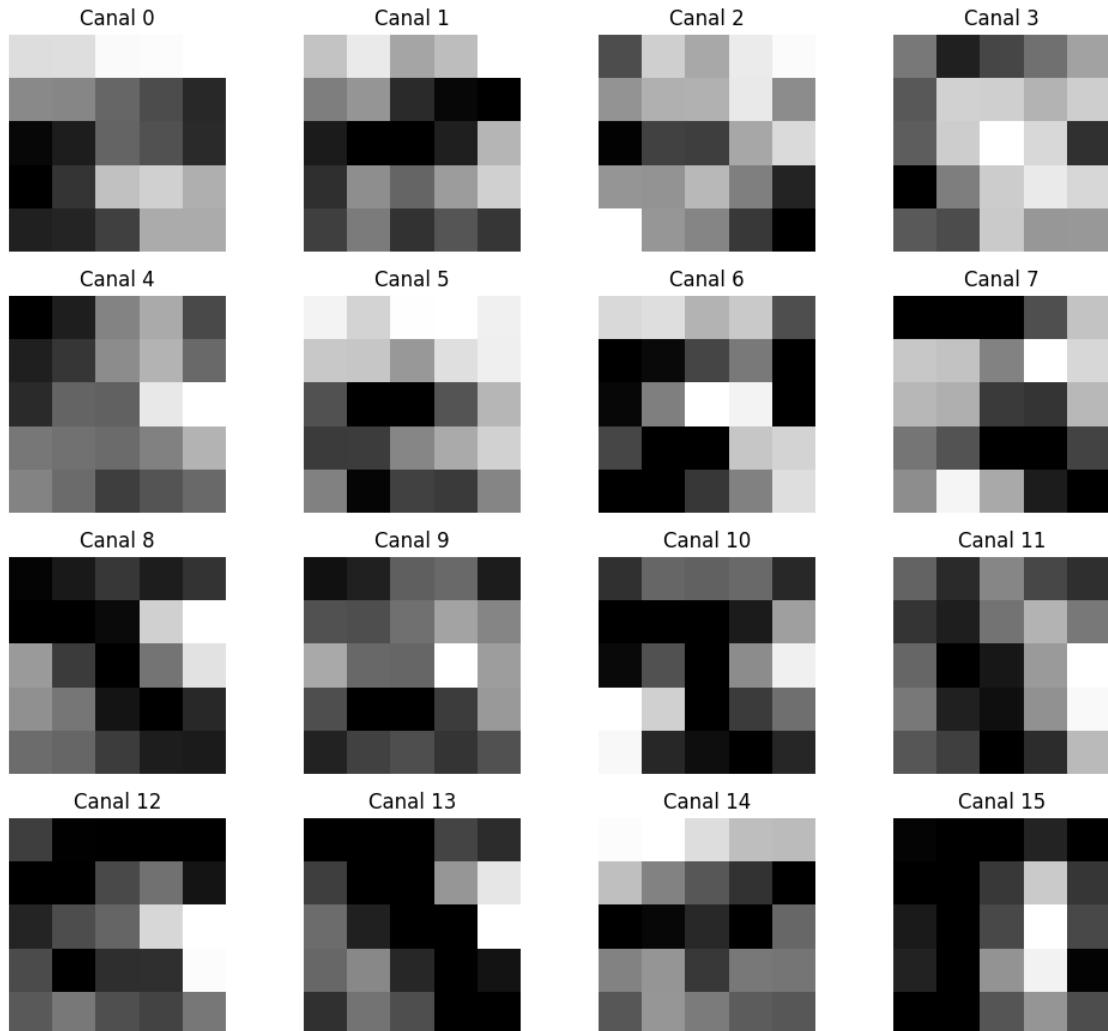
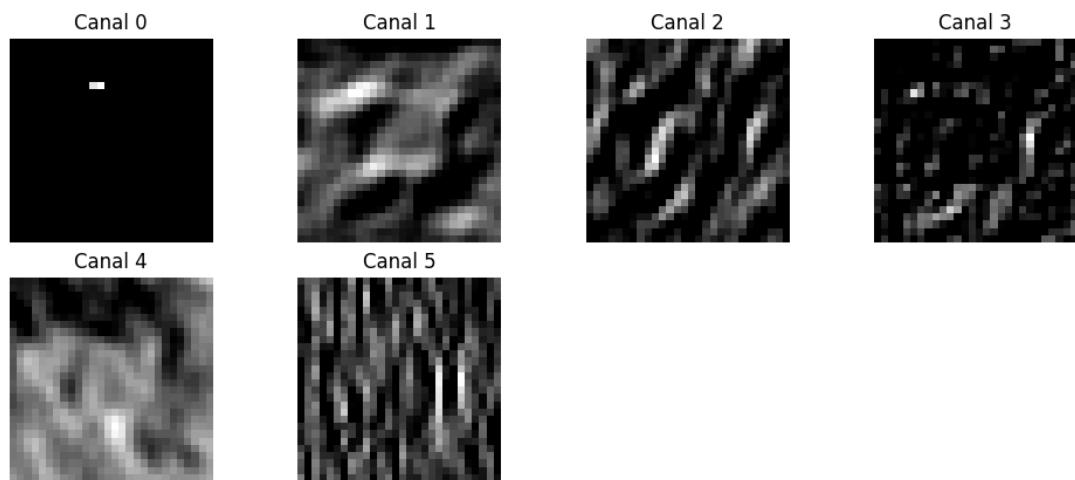


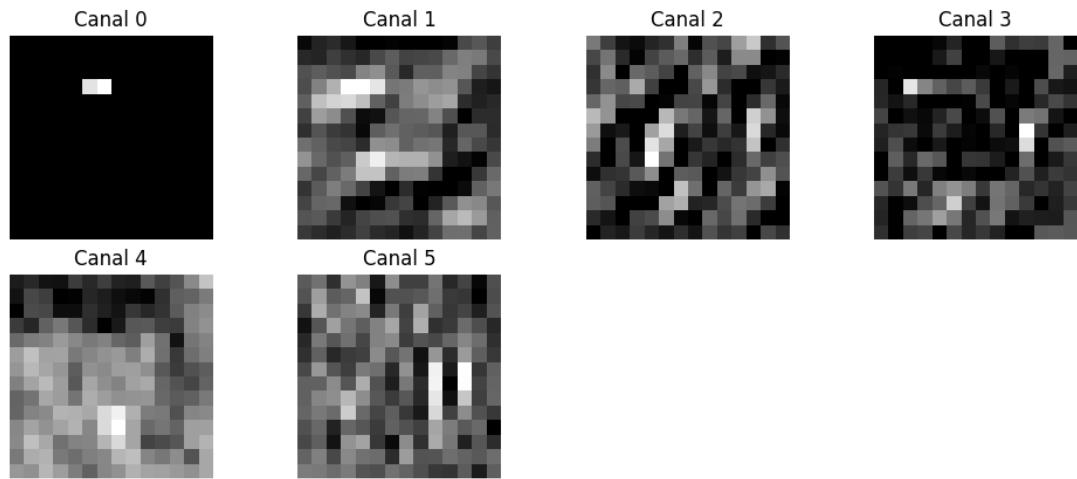
Imagen original (deer)



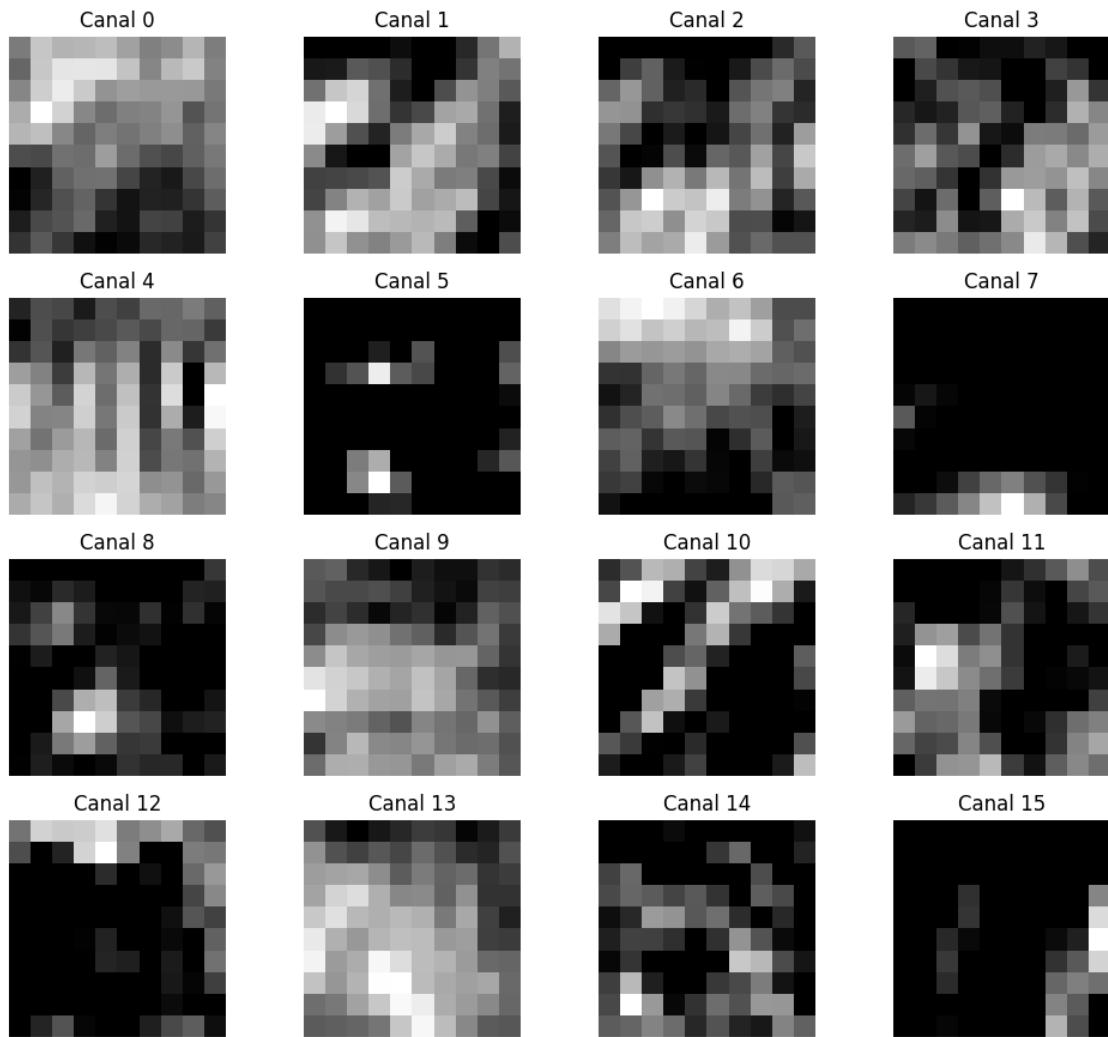
conv1

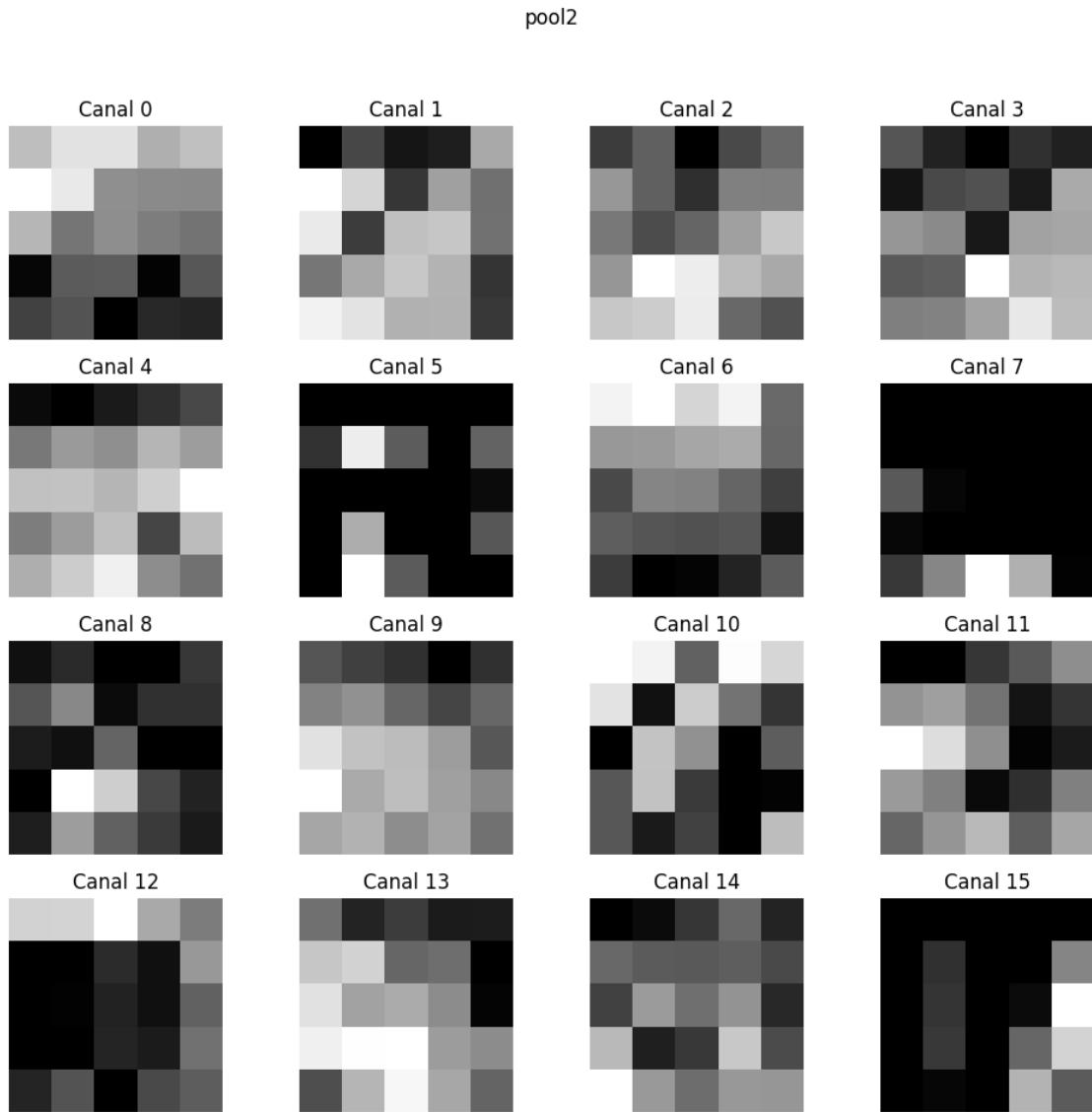


pool1



conv2





Resposta: Observando as imagens, nota-se que as camadas de convolução tentam procurar por algum padrão na imagem recebida, identificando formatos e padrões de luminosidade. Com a implementação de múltiplos canais nessas camadas, o modelo é capaz de atribuir o processamento de cada um desses padrões a um canal, permitindo-lhe combinar os resultados posteriormente para gerar um retorno adequado. No entanto, vale ressaltar que esses padrões tornam-se cada vez mais sutis à medida que se avança nas camadas dessa rede. Enquanto isso, as camadas de pooling realizam uma redução de dimensionalidade nessas imagens, reduzindo seus números de pixels ao formar uma nova imagem removendo linhas e colunas de pixels da imagem original. Isso permite à rede reduzir seu esforço computacional para computar e ajustar esses parâmetros e resultados.

3 Gaussian Processes

```
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
import io
import base64
```

1. Durante a aula, discutimos como construir uma priori GP e o formato da posteriori preditiva para problemas de regressão com verossimilhança Gaussiana (com média definida pelo GP). O código abaixo cria um GP com kernel exponencial quadrático, mostra a priori preditiva e a posteriori preditiva. Experimente com o código e comente a influência de ambos os parâmetros do kernel exponencial quadrático, tanto na priori preditiva quanto na posteriori preditiva. Nos gráficos gerados, os pontos vermelhos são observações, as curvas sólidas azuis são a médias das preditivas e o sombreado denota +- um desvio padrão.

```
SEED = 42
np.random.seed(SEED)

s2 = 1e-04 # variância observacional

def rbf_kernel(x1, x2, gamma=10.0, c=1.0):
    assert(gamma>0)
    assert(c>0)
    return (-gamma*(torch.cdist(x1, x2)**2)).exp()*c

x = torch.linspace(-1, 1, 100)[:, None]

K = rbf_kernel(x, x) + torch.eye(x.shape[0])*s2
mu = torch.zeros_like(x)

fig, axs = plt.subplots(1, 2, figsize=(9, 4))

axs[0].plot(x, mu)
axs[0].fill_between(x.flatten(), mu.flatten()-K.diag(), mu.flatten()+K.diag(), alpha=0.5)
axs[0].set_xlim([-1, 1])
axs[0].set_ylim([-3, 3])
axs[0].set_title('GP prior')

xtrain = torch.tensor([-0.5, 0.0, 0.75])[:, None]
```

```

ytrain = torch.tensor([-1.5, 1.0, 0.5])[:, None]

def posterior_pred(x, xt, yt, gamma=10.0, c=1.0):
    Kxxt = rbf_kernel(x, xt, gamma, c)
    Kxt = rbf_kernel(xt, xt, gamma, c) + torch.eye(xt.shape[0])*s2
    Kinv = torch.linalg.inv(Kxt)
    Kxx = rbf_kernel(x, x, gamma, c)

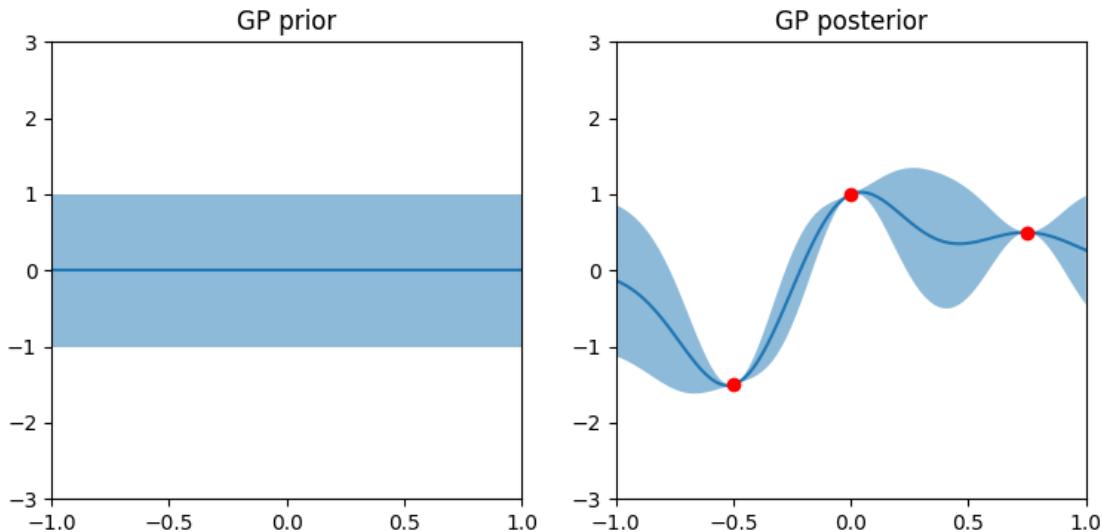
    mu = Kxxt @ Kinv @ yt
    cov = Kxx - Kxxt @ Kinv @ Kxxt.T
    return mu, cov

post_mu, post_cov = posterior_pred(x, xtrain, ytrain)
axs[1].plot(x, post_mu)
axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.
                     flatten()+post_cov.diag(), alpha=0.5)
axs[1].scatter(xtrain, ytrain, color='red', zorder=5)

axs[1].set_xlim([-1, 1])
axs[1].set_ylim([-3, 3])
axs[1].set_title('GP posterior')

```

[130]: Text(0.5, 1.0, 'GP posterior')



Resposta: Após alguns testes, foram observadas as seguintes características:

- Quanto maior o valor do hiperparâmetro c (conhecido como amplitude), mais as funções amostradas da priori tenderão a se afastar da média, enquanto que, quanto menor ele for, mais contidas essas funções serão, ficando mais próximas da média. Com relação à posteriori,

nota-se que, quando maior o valor desse parâmetro, maior a incerteza do modelo em regiões sem dados, já que isso permite a ele explorar um espaço maior de funções. Ao mesmo tempo, se ele for muito pequeno, o modelo pode acabar subestimando essa incerteza, tendo dificuldade em representar a função geradora dos dados.

- Com relação ao hiperparâmetro γ (conhecido como o inverso do lengthscale), ele parece não influenciar tanto na priori, pelo menos nesse caso. No entanto, ele impacta fortemente a posteriori: quanto maior seu valor, mais “detalhada” é a forma da posteriori, apresentando flutuações rápidas. Nessa situação, a incerteza aumenta rapidamente ao se afastar dos pontos de dados, além de assumir o valor da priori nesses intervalos. Por outro lado, quanto menor seu valor, mais suave é essa curva, interpolando os pontos de forma mais linear. Além disso, a incerteza aumenta lentamente à medida que se afasta das regiões com dados.

2. Durante a aula, discutimos como escolher os hiper-parametros do nosso GP. Estime os parâmetros ótimos para os dados carregados abaixo (acredite, é isso que o código faz). Reporte a evidência obtida e faça um plot similar ao acima. Para o dado de teste, reporte a i) log verossimilhança e ii) o MSE com relação à média. Em caso de dúvidas, recorra a nota de aula e o link adicionado no eclass.

```

data = np.load(io.BytesIO(base64.b85decode(
    'P)h>@6aWAK2mk;8Ap0$)Ikt1X005u^000XB6aaK`VQFq(ST1gGc>w?
    →r0H6Z^000000D}Vn000000G-JFQ_{?;!0~YzY=9g==Y&XhJ79<?P+Lz&d|0HoP>&SM
    '2NI?X(A<!
    →XAf&F9gxYZsB1plJNq0ihIRZpQ_**23=qcCzMEScGRBv|9&{@)AhLg<_iyi4)x0ld$%ox21(ae|ra*E+%zB5xip_e970
    →_<%;vgd{KI?I8*SS
    '@8X}92%ftnhkr)!{KI3SqaztHT0%1?
    →F#czrlL<L1=1>jXSP9_`ra#1EFBN&dUp#}93U`}z=Uq7W<3m;jT?g?
    →mc%DzX*x|yc(1GbZBQcI^|rK04V2BBL^Y+2'
    '2>OV&sD>fT|N5`c9=}>0c-;Ux)Kh#5%KKOC}5=2f&UT^G<B|Itrr%Vx`M7!
    →#hh-(FZfcVH+w$FY0zHvTfU7#>wvZ_WXWywJL>hZevBpDQT^trs*Dh05Dx6wZ^'
    '!v-$V<-Dtd`E%k=Xs3rE#nf`py+{YH_eyu}Zl1;=a)XkTsql7Za04@Q26fwFD3`OfxLm?
    →bo9G(@bL+s2Gth#yia`8xS&mna<{tU`x8IS%3~Ifuzle~nF1$0q
    '+KbUaeQpyT1dh$Edi<>I#kEak-MsA!kV)ShvPHNAZV}_Rs^}u9td17*yGFS8ad~gY_ar)B^!
    →hCNgc^wsrl)g6fk9cRPwVpLp>u+}><ki)1;5d}FHHX1{WZG7'
    'C;jap_g;LHbT)#a(VMt#9tJ9t) !
    →vXuq6qn={f|sQnfIcF9an-Xxcex}zy}Csq9U0wi6sGLmP$Vj62*^s)Hnt9+sTc3wi*r;
    →gfKt%AR%64&EI#Z8d7FHaQU@N
    'frB|1=vY_4L7yh-ug^wd53_iglQs<1TeG<iat;hn&RT6P7JRVAS1@0mg7!vvm?
    →GPVu0}7Nrg9cDLWkVHE$fAnxT!kAuz*F65;ZZ+B$)Esds=y8P*zU@S*wFK'
    'w<vb^QyY$cXQ9a=2f&wBV0_}&4N|+@`oqK=);
    →7=Hp|j1XuDMqk<e`D^gt6@A*VOpYsq*sT09Z4Fo^1>
    →x}a9VXWtVJ<Lb(``50#rcGzT7i^2c0`tsJoQ2aQa'
    'N(u`b=2&rY#1fFUM&Q*1;RC6!5@Yw1yXmo7=qaT7y4vhOHYq#!
    →dE*%FTX3j) #0ZX}3~rwGKGbt4aW^j#Fnz!28|wixCgxm8jGG<AU+R5ARn<n6GToWiXp{Ia'
    'P)h>@6aWAK2mk;8Apk!*Q7DH4005u^000XB6aaK`VQFq(c`j~nc>w?
    →r0H6Z^000000EYtr000000G*KSGaCjRfKy`8t*7DEnx=P)x;5ltqPkws_{k~h(i0Ye
    '0?L<t%BCWTW1E8&iF)a%+|5NB+0DeRwn$XRgdeh1hY^vq1VIv^=?xX@wI;
    →3Ezp&@q^L%_{k)NK9KC=$Cgjn0DWYT{HH971zNoLpwd$L!fatuCYLYdrTo_;
```

□

→ 'k}^ao+g+5F1\$EmW-u_G`gX-_YpypHmCvWQR1TzoBC(v0zFCY7KAgjDN%<\$4duk3PdbImODDZgC&UhWh|#(pJ`*DgZi?
 → ~DzubqGv7CqC?ppGEf5&gc0DRk&;S'
 '&hcASfrB(\$\$FR@kY=?
 → l b<(Gib8=If@uL-N#|GT}&W>R30q<aG>0{`8Isfa0u5M&p-|t>h-hh_+8izM_D3Rsc_<o7_Dct(U@d\$C!
 → 3=QvkbQpRTP*GjTUhg)d'
 'GN9TR5<P>VBG&+cUW?hxvb3mHBNck}<wU1Z@Y)r6?
 → }N5sP(6F>sn|0Tt97sDDBr1pNp>0ITsLF<iDvt`*d9pJz8yH&Z-mG%-MjouW=M>Rjk#a<3?
 → 38&>eoZf'
 '&{rmIjy0C4H&!gSPBEW=x>BadcI?4#oQHRb_=!0WmIPe2tpQ>3cHzDwb91<-w4-mrAX?
 → +aI~%#(50<|y;\$+(+NTkSFyD#eS!kM~1#t3?7sN%mRuD3y4aCoux'
 '!8q88>=k7BU19H67Y=0-4aWMYifd-Cg70`CAwAcC5019;1I`9a=(4diTV{w;
 → hBh51Yhc3eI1D)qg{lq<otU}+L0rw)WgcdXu?pzZ6g3h{1LFk-8&uF^KWK9;'
 '7+##?dMTqFcmHBoT\$auw@4{zb;h1oxmb@0}G>841YXz;
 → dW9Y_0QJi}*gX%it0Qt5SoL+x2*<|m4wUDAs*C#(img;=r7PSeYu7o*DLe\$uj_xy&}-~uj@vWxVj'
 'X)G~x&s^R}MAP?#!2`AZxUc-+T\$SAjlDi8(c{&awWPX@*G&f1A*5@R7iNq&xUpH?
 → G6Y-eKBS821<Y?%vKP0%\$B%3Y-__1e4JF@a<~REU%Dt*?Rwnb;9a@&'
 'wHnxLtM(D0AO61wpHeclu;
 → -FFotH2Q1\$kZZv%GoC2`jxrXx2bq`#a}@p7#Qm^6{vaGlWM4>+XkiP57GV!
 → 8t@XVOEeqSdj7%=i4Q2d8tkCk*RUy)yNQByr~*V'
 'RM*14P)h>@6aWAK2mk;
 → 8Apn6>1aKG%001Bm000UA6aaK(b97%=E^csnORRvHAP@im000007zzLY00000omcx~j%6R8w`g=49g0r^hM}ny%_{j!
 → \$)TxvxfstiKK'
 'sKiWZBW<J_t;k_fVx=^dEEYD)XUy5mFx\$-0+4<SCqeGF%dtdkG{tMpchwVAs_j0;_xt^vm0S^;
 → u&`H}dC1;e6%Y|QeL_sYV}1dBqe21#{Q{z<#=H|Ve#&c8'
 'V<IBW_XEdIo*rp^KYh~p=t%2xw?~440{tH99_TmQ@Be(Y^0Hq|EV*^>(jI3UnSRgb-Cx??
 → \$c5J5_-`6Hu}_B=etN%={huzcE*R5D^Xd9yjr=QoTI(L|8<{sH'

□

→ 'WbKQ8H*1TxZG3S9KU}cq@VYGxe0BL}5e3N&G{64)ZUgVI7j9tvj<QZK_HN)_>#7g#XyBUD(L>jrujkC#KBdQw)YDx3Z(
 '&(\$+4=\$`\$1J?oj=uk6)hx75?zeXX{R&yJ4%q}#zdc9~>0Tt~CY?D=&p`Ed8ssgvrs@r\$IAql4;
 → bR=d`<juY3nzp-13I`-dIx_m@sE!\$hG&fZnai-18n{<x=-'
 'hbw|d9-UiDv-;
 → KXwVda*j0YNdEyvxKbk_*KT87w#@UTsf_W1FPhxK0re|Tt~c6FtPDdsWndpL7^@Z#ZPJbc17e9S}ZA=}or^YErMkMgx_x
 'UANRQIO);J@Z=gsTf30`ZViXzn*!Ic%b7<G<@c(g+08#~YFPYt&R<>5ySdR`\$W61WS!
 → r%Y`nUW2h10t9A(wTln`U=chPioh==SQdb*i6`VsH2Tije{?Q&*q'
 'HRsI8dFY1&)f_v|G_I0vw^tIY>67By)Ok`hJ?qcTdwx(gcW*NrtmdlyZ~yvz^J>Nx~?
 → vGbMHOGlcK>%y6?<OTI11\$iD&Ev=L9~%QR<YDNMcnu*ZtLKC|M#I')
 'w1\$g\$9jiFi9*9Z~8TfXM|Cvhqo3{Q@\$!pfB;#09\$drFpC4ZYezi-~?
 → N+y{yJzB}A6>X#Z+f_2u90rTc%_(DxDt0U6c5?;u@_S_*T2?{ (=XWYNx6TY&1;<`V'
 'd~aB<3I<!Jj%`yxpS#<B``x*6di+g6%V|!3X+t?
 → zVe)I87M3%1STCzo<s31_+*r=QH+J0nU0~v|yqCB!
 → _vUiCtZH2H1reMh-KXw5P{ySxW`N5\$#j47CiDev|'
 '_10J4PAp@m0}nr#JFtvbi%g@+SXh}o>b~Y>0u0(&{j(LNG*z0BQ_6_j&4x<Z@7~itADmN4Q?
 → 0+kOL;!zt>b>7r0X}p@SD\$eETs*je_c#?J\$p&YX&3Er&vMaJ'
 '@6WGYw8wImi#01wRXsD>#T+v}kG1BH?{8YIbg|*%3C}ERJj*rTottj;
 → lrLN8#^keXCE(Lqq^ZVzhB2+}fo7MUTc27Qd4?r}?MCr)3(ft?PP4x@K>PA>r}>kG'

'Vr%_{cT}!1%<Hlc(6G@vA8Jic>V`ePYxp#&zA2VU>zOPZ|0_#\$PxD<#7fWCpm0k<7?HopJX\$;;U!
 →GZy?K)bRZeo7g0u<d3#ENqmN38L<_EvndxG;GAMi@mM<*C!'
 'q4V(pCs-E~{>!50oZ#xu8;)0I7t^Mv_+on78z|;
 →B0gTZGewjt=tiho@zQILIGUN4UAwy03Hx=@tY0tDmrke5v6>=<Fnx*M;
 →^2>eYzZ~Z-3w2g&IJ1o*wBzjg'
 □
 →'fPFBQ*+4B|a^yI}e0*k(8NdQwTWK>JH*WQ|nCuv%3T)<LdkbBCIviu3ecYp*8(^5|D1)D}nTMWnuUaiV%2gspjz~5IJH
 →tr_|Vk%fdbC5BV(g'
 '3pL2+g1!dn`FuakJoqpTd\$Z+X4()jE*Ec@SWBSXt{^!
 →mc^4QVPef%LN+Qg01Mnf-gxg2Q^)<I^QagI62ZfiDQY~6T(Usx=f_0|DiZfzR9pW#DpZt#iR&;1GJ'
 '0Q(qZ#%AO;
 →F10pdmtFx@*+=f>M7u0|m~XZ{d=Eno8Nlx5>Kp5=\$=pqEtI4<S-NoqXX7YBi_\$M=IISe&4|3nT`4LxS;
 →q>rJM;GK*\$aNm+mE{jz*Ke&T~FL|s<'
 '\$Y%NjMmVw=VgrZtemuT-+)%GIG=J52hMg>yyP;1D~L69ApbwCfi0Dg2|xUV)4ulTe;
 →=QRu(k3F!49@W-~`x?7G\$acXnU6\$*NpBFWE%Ak^2n&T>Op8PmZ-J'
 'k;Z(R=r%Aq=Ii_4*s-2IhIZ0_<_4Si*D=hLYsHUjbejhMz(qzbKUl*c1N&*;
 →GUbK|AKkfc70+T@ZX8+zhdC4_Tps58`_Rt%y=U+nMu4p@=NQ=dAz>T06do0'
 '?})SBh108z{U2WX9tpY*Ub+vVS_fWQ7uk9a67*d7X&*??
 →zTnb65vqM-wa\$TNIV2uqPGsxcSfn~2TXjk`>I6U44bQ8NNKjporaGfkb%&4ofKc@X(drXY)HfVf'
 'AJI&GMUeW84eC2WH3ultTp(I=0&mR?R%?zBqPar8<_wcGckt32V!7rLL7G\$S)7&C1bBxPf?
 →YYK6%{jVh?vX7VV3cryCgB8\$!VUa{BV-6y2ouiW7Va=dIE1fo'
 'i8SF9Lxfwngk!`C*GLr3Q6Su-qj&&six;p}Jb@d;8weARK(ur0)#4fS7VjWIJcNVdCHRV`5Fy?
 →|ns~LxwEzZ-=de_~2bXvdf#OBPiYjJR-o!29QH&C=;(PHd'
 'n#8;4D;`Focp1mU)9@2-W3qT08RB&`7tbS1yp0M?18_?h&|51zInoW}N=M);
 →UBLwD4AP`KxSD3`5C%z?uv9t)mvjsFOUDO^UBfQv9B!BHVXS1zsnSJUkWM00'
 'x` `y|D2k=4=q#PZH0ds~q{Hx*F5@NXG*YD7sFRLkrF0!TrSou0_u=>eIKBXmPk`ea;P?
 →nQz5-418Ms@%1AXN~@Va~n=F6wxSNRqc\$j9J{d=32Mb1+c82b1N4'
 'uvoqbTji5*M!
 →pF*%15Dxd=<jvv0J%x3oGTruv5MaZuvCaCf|nM@^Kg~Uxx(wJgk-PL#})v&dV3VS3VK_<QowoABhF>1}MA%M4@~quF8j~
 'kttsbmwYZ-\$oC>pJ{ZIOi{bcWIKCN<kA~x`;rMJgz8j7YhvUn!
 →Rz41M^6fY\$ACI>3>BPXBINrqUpWA2\$~|&CoPd9n8{n@Tfx*fZn4+A4CCVMhP!54hxdcv5'
 'L7;
 →LAhAYP)PPqo(DCc08at}PpLGV#7La=fY#wa&o j&c-Im8+1aoCPO\$p`CIVpj-wgry)tx4Ni_jv2q=poCkm9J~%1LPA<eU<
 '6swg>;p9}*D!OPPU?SYKg_CnJSGgB!m4o5rVq8#8h0crnoE(iP<!U4;XT!
 →<eC{_-KlgsfxP)h>@6aWAK2mk;8ApnKqayn=a001Bm000UA6aaK(b97&ME^csn'
 'ORRvHAP@im00000Xb%7Y00000ot0DL6pYu#QzY9%;*q6H1D\$%t@|06(AuY6~WF1>b3ME7#Yg!
 →SNL|L+veK{CoUuK3emccN#EZJIA)Ytnjyyv`xvq1a`#Se8'
 '_j!Ki;
 →^}jy9BVw+c*_%QZ`s_G*Cfavv06HJLXfv}z3G0_>biyNO`^4p@#9v`w`^DWEeER`wyV9uK2=o}!
 →X8Bxf-m9!S-b}K>b6+MXW;Yee5~_^&tXkkh0a<|'
 '1!i+3Yw&nQBV`;e6~vf=L)=m<2Rj~9&#YkvoUntn23KZI#M80=gC14pe!wf}DYn@@4sS\$qq>|!
 →N(JAL25j~DX3>bcRgQS@W5iVsJE6#c7ADL31>(>P9i#}UO'
 □
 →'31)(yfi%JTPY5n_r9Y>+B%r3iJCEY*7Z@8V&\$w\$+ieJAG_>vT(QEM_i i9HZXzkiAwJ9eVv@3jH1|3a?
 →HkpZ;ryU>gs@Jo!_42hs#xUdhoKS%KKU=>I;;#Y6'

'YrzU4wz+WS^6Qrzv~bwA!
 ↳ 9#Z2ND1~fTK-I4euY(EsRxt\$Nf^i})^pak5U< dSRJf^Gn6~qF<01KcoRc{JNA7SsaB(qnG=}Su`#7JIz{zS1Q| (yz^Irc
 'ey>B^^d~p;Rw_R3ZD-rYv>=7hs}^M910sHA;_vE;G5*F#XJSh(-tLoFE}*0\$_2?
 ↳ 7jR_iKUN< iNtg\$T&;>-X1QmE)\$@d&He%lhB2=?^@_a#0z2fFaG;B9}i4L'
 'tTCPbgmPc1P%ym^2|<5VBpsVEeXdqvqBS3b4&0=S`bNS7_MV@1N93UJ+sPSy!
 ↳ 7h{tzw}@ylZrAy(-S>@C3xtZ_nv00RH(Ca|DmmtiBg@(@6>vlaQeq4-UACA'
 □
 ↳ '\$d{z^kj|Nhm7Moyhhj+hftAx3yR#6FXKa5|x1\$<2\$NM~7RINenD1-L}2T2elr`E1uSA)xAdfa(CYVnD8X%IS)a5`jb`K
 ↳ v_77Qt@ZZ)!@6YTUoXoU'
 'WtM|`#Vv~)w2F}+IPxif#2!Qgg=dq_<zYkMo!
 ↳ =F@tr&IRU@K{|{6^Yi{819=h5ISGl>#~i6qGAU<OHMXpuIHDeLarKg&S_*wMrJ`etuWx&{YG5!
 ↳ xV~wtaWy)}'
 'KH`_tD?<Zwd#B+M5}x05zbw#RjjjeYc?aFVe#pP^6-pGGb(MQEr<3f5Ijz?*BSw^KIIUw}!
 ↳ {I+U*{p1iU<{TMn8uZ_U8GXWZX6(+0fFfWSSG*%hBp|>3KRX-V'
 '1c39hkdRJpfD08GN8-6+z&WPz\$imNDywEOD-Na}FOEvbl(~5MM^Rm7~lxqaGf};
 ↳ ufY(1#Ftv6bZpg?!*?%)T*6rfi5P#t3`U~8_8zL8E1h~maMzdkBh`KFqw'
 '8#TkdaHlhrNCy0YqoKz4>%jSsP2G1PIyjY2\$ha)i;2zI~2u)uKD7PQbc@I6H;
 ↳ N~UM7SsiDp(z^<ST%`Be!c;cp2b+KJV4UKTL2w{hn\$}rh)nt(fGU`4JIY%'
 'e}yyk!N+WJ^A3\$ps3hrh&d|GoU=^})JE;wr>2`>}Rd0a6L_ekJOEj?
 ↳ Ee6Wanx*iU5aa=AI>V~h3duzovDy-Ad2^FE#K<J~*Vw)BXdk9@fY!U4MjotCKdcV71'
 't}5=H=1&~TYG;nTe>4crk+vo0g(&dY)s&H%MuW&2+fZI%8Z-wc9hP`Xf!
 ↳ YO}z}F)*\$OuSCo#CcKhK*xQOI{C@2B+tf#!8`C%D!Z5LleOBXGjT;H9}a5W=Em{'
 '9hmH00k\$%Y!
 ↳ OKI@E7+R|\$)Wza+d5jHYhH26TCWM70LL0kakfCfIdLK8N+F165B1UYy5MZ_{H0408ek?GA2ALf!6YM?
 ↳ %~{HwYzM}64`0oQtYAyPvQRvz*%{8i'
 '6fXk>?IJtr&)5kVdc5pq799rLYIX(iXJNy\$U)%-1L%}JqN6EDP6I9\$>>^U#9yXnJ>rdd!
 ↳ h19Q9cdKRVA!i8~m2)G2r`L1iQL#\$+Xh8PArbB~T+L7kyoF_1`'
 'OsNT8B7u(1j&IV>szFp%H<y@Iij_Ldu?@vJP`7>ffZkXewAXDMtemTX1Z3hfi>VjjB`V;
 ↳ c01bzDYyb1Tb7e4S(VOBxZ-j{Qn@!9M%|Jc4QSJDpd}vQ4rkPYy'
 ' ;lCpRiK)bLxYM7q)#P9?s9S0cD;tsEF(HIDD\$)Y(@2{J2f2D\$(\$6FP<#wNI?
 ↳ 9NBwSsT<m3U86Nrx`6E{FGRdVg*Jowp&e;V\$bDUHe*fbj7yx~KV4)102<zrv'
 'NA\$pVVj!`2g#wnFgX}JB?t|<*FXc_N`yqE%JbSm\$5Xcqpx|cM?0-{o4y-~<8)aaXZx!
 ↳ a6Gfv-zh-jgB7NH)%Lb{mGvI>V8!F)aAL*@W8XGYXX_M{Ke!KSNBK'
 'z0=FH5FC(fYDj%I3Ww8necKZ|0b\$Y>E0v\$8!2BD1%1~wsq&Mj74*w0\$?
 ↳ >-(J+WmYIUSWh-fX@s>w^<16>6nG_&FRD>@;GGd{n@IsaSqgt7jBFQe*s^KpNbD)
 '2DaK*tbE8A2gUPA2N@Alus1dP4RK-tisNMjx4i!XTF3Rb~~j*E#<riAe|28}Qqqr0p|h~8!
 ↳ yL~zH4n`AzC6yi?HF@e)4adIas*2&+~4|JXAP1J~rpY3+t17'
 '!Y|A&f@?bU\$TEE%m{Or<o!`zvyVuA_C_Qlmb<du3ct3`0SXrxL`orgn|D!
 ↳ mo^8K_zmKP\$-p6{e+FV0>w!=J?Q`Q1()5w974>(o1I{!H-&6PMw7hcZ}5>{02IJ'
 □
 ↳ '&6Z=LZyth\$xzd9K=787R`C5+X6fCUmeIa5r3y--<@}32*>S`L+Ichlr40TMU<IHIYGV|qVkvR~On>uMf-Nqo(_r>=ILK
 ↳ Nt0+aA!062Y3'
 'afo-%j\$L^>1qtT+lgT&=GhW9Ag_g\$Qd#n1x?3--J8PFhfDvpEljMCJ_%?
 ↳ t>XsQke%(hst0>3g@|U-diBgBb1I2g*4fkB1d1;FKX)>GQ4^M*N-qXutbl*wHvL'

'E|CGH_TI\$}jr3K0%p>0Hzr~2oaR\$Q<Fh6m0K~k0q7P4Y4W\$x9\$<L<*\$fAj&?
 ↳a10ZbU#r7)1Vh4sZ0dR^n` ;KBwWDP<kJ9axT5Ryt%2C5wWC#Cq+M9^@Wa!~-'
 'FYX?ELACum{qiODCvg1f(N|UI|M0!WjescBuhZ1}N2mtPMQzS)Unb+74a&LObMldv&a6Hj-i?
 ↳38jE?H^uyBIw)y8}NJ!m\$1t9ota0Pa2+2C@&Cxb!p31+R-b'
 'nMZn\$US##)gDWkAJuZEy(YIXYz1MdsLfV2QCI`^1H)1_U-4GhaKULD@8NjG\$cK5!wL#V?
 ↳eyi68lqcqQKbn%Hnthw@ElzK-\${\$+%(4N)cXUNe*30>7+6ICNsIq'
 '=s1e\$YP1ro^Ph1@|NM5V&Jpx->I)8J4Wn>|9Lc711YhNJWS6)zF`
 ↳@+6SwWPfm<u\$^ir\$Ui>hwY1tImnqs5ywi30)s9_x2>A-hfkBxCvYsBp=*l4t9'
 'U=!UiTgMN53p3nQ0;a5@V=L\$=qYbBOPjaIbeqX!
 ↳C20h\$mA|f-wGUx9YOH=%PjoC5Z={oJBAnfzwC0d\$-)gvt!gJvvG7W%>Q@Ix7G~M|i!N;+!;
 ↳y=@`%T\${'
 'XnuWXQ&uYrgB433XS(&\$_sofjuLL^2=8|#~KGTH-&R^~LWCrofq6fF*sX-)Fa3>Yz_Tr)4!
 ↳hV+^htbSOd;gy6Y})1Ynj+87LZ298x2KJrDCD1mXQecNfgL}5'
 'RsK-X(&xo`ZI>^;8(#Xe&0-jR-r0REuk<2s!~!~0>%>y;_#^6!9^@#rQB_as!PpZU?
 ↳F~G7@R-o<sLJLcJbfZ)v^1N79wAb^a<WY184^N\$6h}dUoGXo&\$qXE}'
 'B|DWHQjoaTpyv^*6ID0nbMo_Mpq^ZvbM7HJ\$_f}C-|^~GB<<~C1*y7oR15F1%g!T1!
 ↳NcUG`ms|~~n-?8zNoV4##I@RDyE-u1MuiyI-HD^qjQh3PRGdxukQy^j'
 '\$5%;9j+aHd(bPhY90=-4Q?
 ↳qf0&)BW<&Ln1NT`LNGeEa=gGX+ES6eaxk^kCE6q~(BNH=eGNI>))E4rkV%D)U;
 ↳sYrs^<yIp=vj06R91)02J+<6iY>Px A}E2S>\$'
 '\$Hzb&zWBK6L;Z07_ugZ?>pJm^D|>g29Ru}?
 ↳1LTb&hA`emyY+8xCq|sqWV3fuP`GT1w|9RP%#03g8-3i30Tif<Mi&?uc&3Gtrre4ZhT)!k<qWi_sv;
 ↳BCGtiA_'
 'IWAjq#6k~ab%kOM4u+3T`
 ↳gUK3{(\$2EnVrd#4h+A+dA*^1xc9dc@9cuSVn;=QXwGXGxdNOBmC!;^px'
 'Jo>hFP}P%;DQg`C\$DeeduC<D\$kPjKv&4k?h3Mu#?OU#Q;rw@-<{?
 ↳I<fL&1~MBJ+)5badu%6bmt-V^fF\$!xoB1DEm=#g|*a)x|fU|6im}_%fwGpG-yZWZ2ccy'
 'qZ)KN^r6J}B^iV7oVk\$|-G*zq7YDKuTe1CY?c)oT9T-cC__TS1?
 ↳\$A7g1h+Ikmx i#\$q=c-Zn4_YZ^a+sC+{K7`pia5K1G>VI!40bip>I|Je?TV{?tLMJQ2GS'
 'W%{zZ+fY6E;JIyuwfK-=w6u3MU\$1_bX0Z+wJS{!
 ↳h^gB5g4HRfw1)<g-S81LCy<`pi`y}_AJbw{-mUE_W{gsL_uh;JUw5t ht!
 ↳`{5Xi|@e3X<xa8Ng0_Q5Zcsz'
 'H4GQi\$q8F`y++\$1zclr3B!akj_?r;
 ↳xMo>%I8e}5T00h)Gn*8F6(RWL9Uwo|wBfg9GCGQtPjEF)2(W4oznr7{_d0GWy{@!
 ↳VOjT(TbSwM#rQ3I#%d<s6{17_aV'
 'Ti=L>VaR#RoYcG1N3=1zZx`BAo<tbEn;\$2aHM+D\$&^wJh5WI__M4jE!
 ↳4g\$a<zFhaPU}68&}{@YAgtm`C&fV`RzvRuUh20>VvN%*@)&uEHR#c{b6TC8M=12=E'
 'fcx?0x9mt=p1Ue6URr7Zm+aFY<<_?YqunPv+`0pj`Qjg4+fV?
 ↳v+I6D&h^0hN7m0q)aW2`Yi}{kP@!}xIzroy4qg|VE-9Yw0MVf4waS4M(Es7?F_-+W~ku*NS'
 'jBFh+J)W;S514mT2^SuM;A%d;
 ↳oua0d)CK<oP)h*<6ay3h00008001EXu*W&Jg988npaTE^3jhEB0000000000fB^si004AyVQFq(ST1gGc^DCM0u%!
 ↳j00008'
 '0000X06#iWD2D?80H6Z~01E&B00000000000Du9&0{{SYa\$#w1UwJNWaCuNmORj{Q6aWAK2mk;
 ↳8Apn6>1aKG%001Bm000UA00000000004jiga-fsbY*jN'

```

'Usx_~aCuNmORj{Q6aWAK2mk;
˓→8ApnKqayn=a001Bm000UA0000000000000004ji*bx8#bY*jNUwJNWaCuNm1qJ{B000C410Vay004X;
˓→00000'
)))

train_X, train_y = data['train_X'], data['train_y']
test_X, test_y = data['test_X'], data['test_y']

```

```

# Função para calcular a evidência
def evidence(x, y, gamma, c, s2):
    aux = rbf_kernel(x, x, gamma, c) + s2 * torch.eye(x.size(0))
    evidence = -(1/2) * (torch.logdet(aux) + y.T @ torch.linalg.inv(aux) @ y)

    return evidence

# Convertendo os dados para tensores
x_tensor = torch.from_numpy(train_X).double()
y_tensor = torch.from_numpy(train_y).double()

# Ranges para sortear os hiperparâmetros iniciais
gamma_range = (0.1, 10)
c_range = (0.1, 10)
s2_range = (0.01, 1)

# Melhor configuração
best_evidence = -np.inf
best_hyperparams = None
all_evidences = []

# Multi-start
for start in range(10):
    # Sorteia os pontos iniciais
    start_gamma = np.random.uniform(gamma_range[0], gamma_range[1])
    start_c = np.random.uniform(c_range[0], c_range[1])
    start_s2 = np.random.uniform(s2_range[0], s2_range[1])

    # Hiperparâmetros
    gamma = torch.tensor(start_gamma, requires_grad = True)
    c = torch.tensor(start_c, requires_grad = True)
    s2 = torch.tensor(start_s2, requires_grad = True)

    # Optimizador Adam
    optimizer = torch.optim.Adam([gamma, c, s2], lr = 0.01, maximize = True)

    # Histórico das evidências
    evidences = []

```

```

# Otimizando os hiperparâmetros
num_iterations = 1000
for i in range(num_iterations):
    optimizer.zero_grad()
    current_evidence = evidence(x_tensor, y_tensor, gamma, c, s2)
    evidences.append(current_evidence.item())
    current_evidence.backward()
    optimizer.step()

# Salva a melhor configuração
if current_evidence.item() > best_evidence:
    best_evidence = current_evidence.item()
    best_hyperparams = [gamma.item(), c.item(), s2.item()]

all_evidences.append(evidences)
print(f"Start {start+1}: evidência", current_evidence.item())

```

```

Start 1: evidência 140.3199902609281
Start 2: evidência 147.12181975738866
Start 3: evidência 139.8820399022978
Start 4: evidência 104.38601074827974
Start 5: evidência 150.0243419628421
Start 6: evidência 142.34380747452536
Start 7: evidência 143.51203512116354
Start 8: evidência 141.9360791477854
Start 9: evidência 143.4785276631286
Start 10: evidência 145.4444816673761

```

```

print("Melhor gamma:", best_hyperparams[0])
print("Melhor c:", best_hyperparams[1])
print("Melhor s2:", best_hyperparams[2])
print("Evidência obtida:", current_evidence.item())

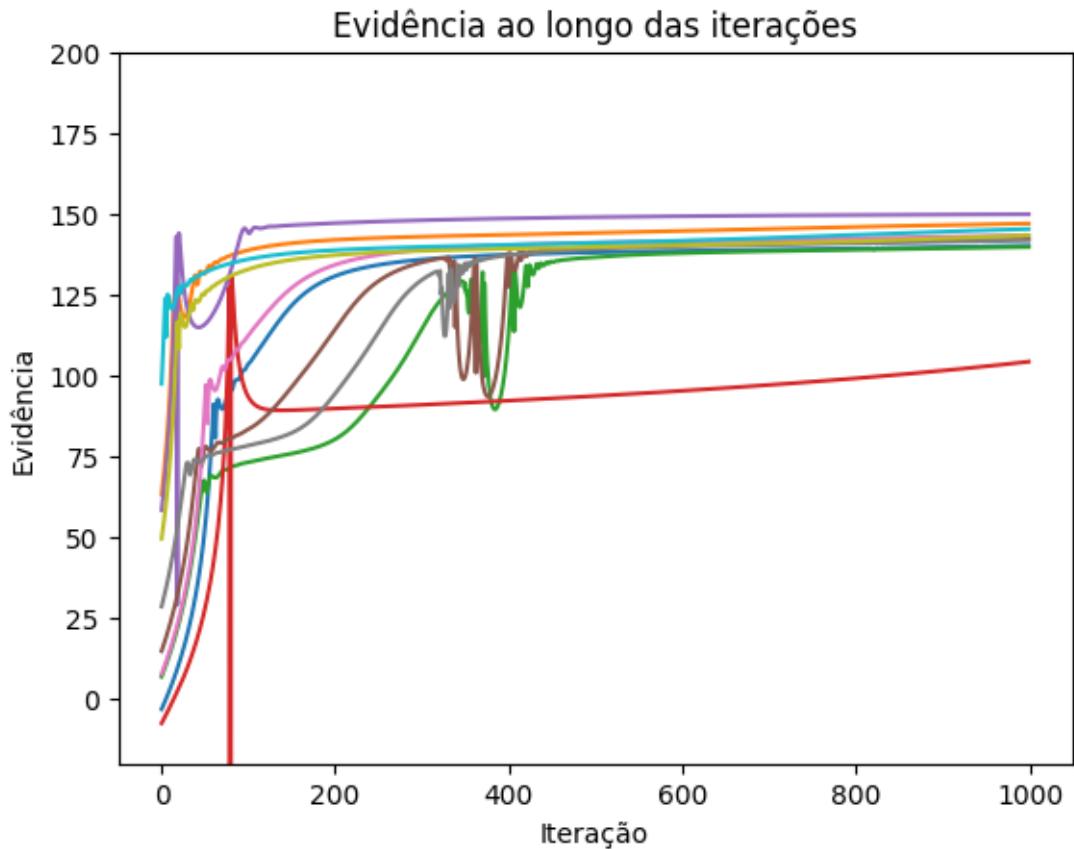
for evidences in all_evidences:
    plt.plot(evidences)
plt.title("Evidência ao longo das iterações")
plt.xlabel("Iteração")
plt.ylabel("Evidência")
plt.ylim((-20, 200))
plt.show()

```

```

Melhor gamma: 15.218957901000977
Melhor c: 0.2800978720188141
Melhor s2: 0.010290687903761864
Evidência obtida: 145.4444816673761

```



```
# Pegando os melhores hiperparâmetros
best_gamma, best_c, best_s2 = best_hyperparams

x_plot = torch.linspace(-1, 1, 100)[:, None].double()

fig, axs = plt.subplots(1, 2, figsize=(9, 4))

# Priori
prior_mu = torch.zeros_like(x_plot)
prior_cov = rbf_kernel(x_plot, x_plot, best_gamma, best_c) + torch.eye(x_plot.
    .shape[0])*best_s2
prior_std = torch.sqrt(torch.diag(prior_cov))

axs[0].plot(x_plot.flatten().numpy(), prior_mu.flatten().numpy())
axs[0].fill_between(x_plot.flatten().numpy(),
    (prior_mu.flatten()-prior_std).numpy(),
    (prior_mu.flatten()+prior_std).numpy(), alpha=0.5)
axs[0].set_xlim([-1, 1])
axs[0].set_ylim([-1.5, 1.5])
```

```

axs[0].set_title('GP prior')

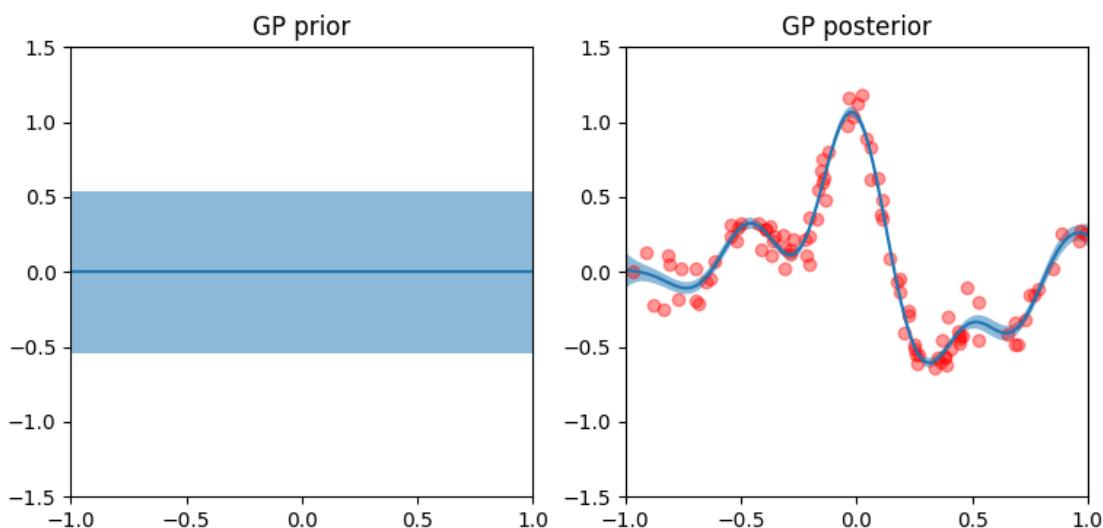
# Posteriori
post_mu, post_cov = posterior_pred(x_plot, x_tensor, y_tensor, best_gamma,
                                    best_c)
post_std = torch.sqrt(torch.diag(post_cov))

axs[1].plot(x_plot.flatten().numpy(), post_mu.flatten().detach().numpy())
axs[1].fill_between(x_plot.flatten().numpy(),
                    (post_mu.flatten()-post_std).detach().numpy(),
                    (post_mu.flatten()+post_std).detach().numpy(), alpha=0.5)
axs[1].scatter(x_tensor.numpy(), y_tensor.numpy(), color='red', zorder=1, alpha=
               0.4)

axs[1].set_xlim([-1, 1])
axs[1].set_ylim([-1.5, 1.5])
axs[1].set_title('GP posterior')

plt.show()

```



```

# Convertendo para tensores
test_X_tensor = torch.from_numpy(test_X).double()
test_y_tensor = torch.from_numpy(test_y).double()

# Predições
mu_pred, cov_pred = posterior_pred(test_X_tensor, x_tensor, y_tensor,
                                    best_gamma, best_c)

```

```

# Calculando a log-verossimilhança
aux1 = (test_y_tensor - mu_pred).T @ torch.linalg.inv(cov_pred + best_s2 * torch.
    ↪eye(cov_pred.size(0))) @ (test_y_tensor - mu_pred)
aux2 = torch.logdet(cov_pred + best_s2 * torch.eye(cov_pred.size(0)))
aux3 = test_X_tensor.size(0) * torch.log(torch.tensor(2 * np.pi, dtype = torch.
    ↪double, device=device))

log_lik_test = -(1/2) * (aux1 + aux2 + aux3)

# Calculando o MSE
mse_test = mse(test_y_tensor.numpy(), mu_pred.detach().numpy())

print("Log-verossimilhança no teste:", log_lik_test.item())
print("MSE no teste:", mse_test)

```

Log-verossimilhança no teste: 445.0601848882908

MSE no teste: 0.009956151544597449