In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
import torch
import torch.optim as optim
from torch.optim import Adam
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
```

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.

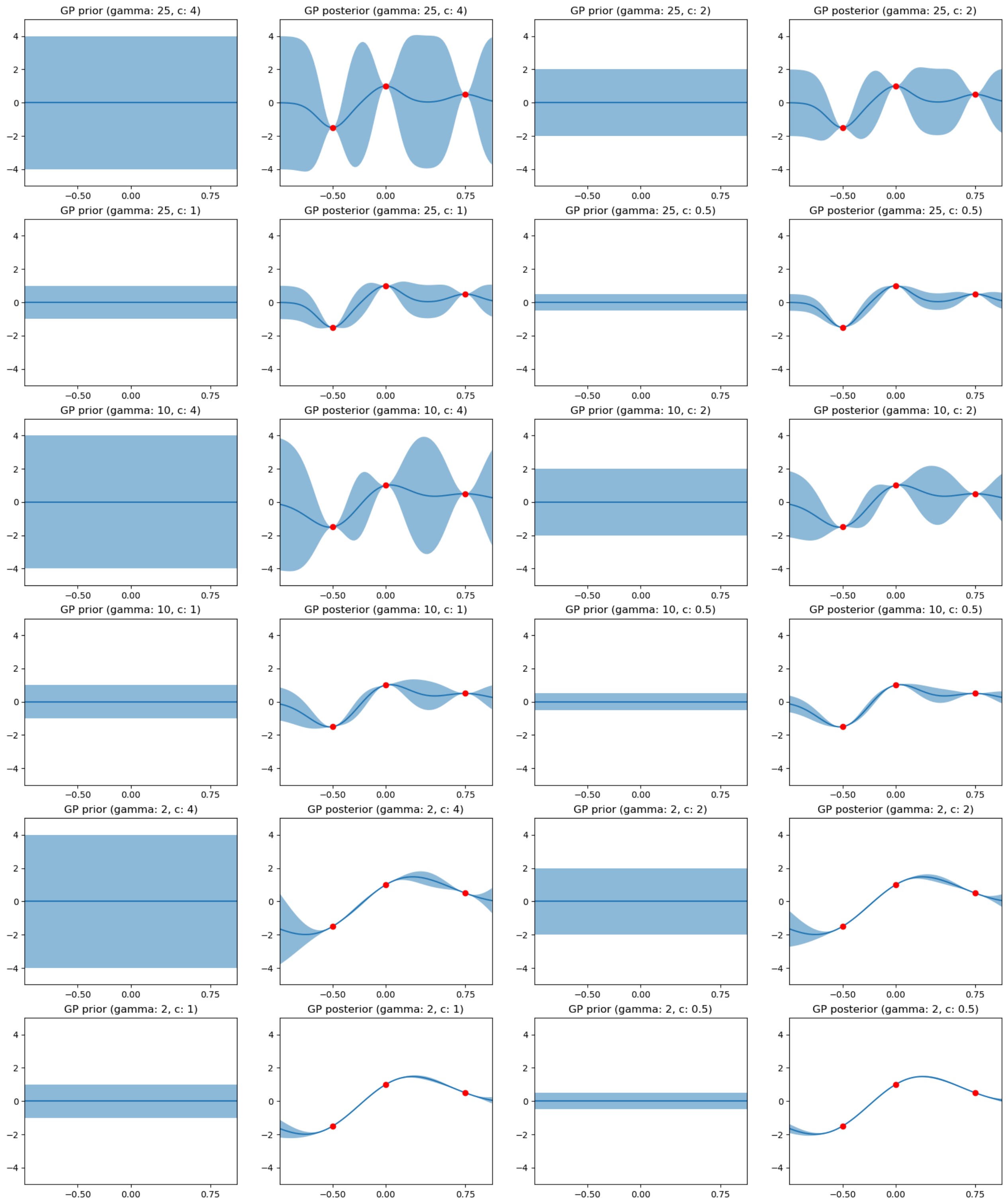
Trabalho de casa 05: Processos Gaussianos para regressão

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.

In [ ]:
```
SEED = 42
np.random.seed(SEED)
s2 = 1e-04 # variância observacional

# Kernel Exponencial Quadrático
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

# Posterior Predictive
def posterior_pred(x, xt, yt, gamma=10.0, c=1.0, sq=s2):
    Kxxt = rbf_kernel(x, xt, gamma, c)
    Kxt = rbf_kernel(xt, xt, gamma, c) + torch.eye(xt.shape[0])*sq
    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

# Plot Gaussian Process
def plot_GP(gamma: list, c: list) -> None:
    n_params = len(gamma)
    fig, axes = plt.subplots(int(n_params/2), 4, figsize=(20, 2 * n_params))
    axes = axes.reshape(-1, 2)
    # Loop para cada parâmetro gamma e c
    for gamma, c, axs in zip(gamma, c, axes):
        x = torch.linspace(-1, 1, 100).reshape(-1, 1) # Criação de um vetor x com 100 pontos entre -1 e 1
        # Cálculo da matriz de covariância K
        K = rbf_kernel(x, x, gamma, c) + torch.eye(x.shape[0])*s2
        # Criação de um vetor de médias mu com zeros
        mu = torch.zeros_like(x)
        # Definição dos pontos de treino
        xtrain = torch.tensor([-0.5, 0.0, 0.75])[:, None]
        ytrain = torch.tensor([-1.5, 1.0, 0.5])[:, None]

        # Plot da média e intervalo de confiança para o prior
        axs[0].plot(x, mu)
        axs[0].set_xticks(xtrain.flatten())
        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([-5, 5])
        axs[0].set_title(f'GP prior (gamma: {gamma}, c: {c})')

        # Cálculo da média e covariância posterior
        post_mu, post_cov = posterior_pred(x, xtrain, ytrain, gamma, c)
        # Plot da média e intervalo de confiança para o posterior
        axs[1].plot(x, post_mu)
        axs[1].set_xticks(xtrain.flatten())
        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([-5, 5])
```

```
        axs[1].set_title(f'GP posterior (gamma: {gamma}, c: {c})')
    plt.show()
```

In [ ]:
```
plot_GP([25, 25, 25, 25, 10, 10, 10, 10, 2, 2, 2, 2], [4, 2, 1, 0.5, 4, 2, 1, 0.5, 4, 2, 1, 0.5])
```



Defino $\gamma = 25, 10, 2$ e $c = 4, 2, 1, 0.5$. Note que:

Na GP Prior, um γ menor resulta em uma função média suave e previsível, enquanto um c menor restringe a variabilidade vertical das funções.

Na GP Posterior, a incerteza do modelo é reduzida perto dos pontos de dados observados, com o efeito de um γ pequeno ainda evidente na suavidade da função média.

- O pequeno valor de γ resultou em uma função suave, tanto na priori quanto na posteriori.
- O pequeno valor de c restringiu a variabilidade das funções sampleadas do GP, mantendo as previsões mais próximas à média e resultando em uma incerteza reduzida (sombreado estreito).

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.

In [ ]:

```
data = np.load(io.BytesIO(base64.b85decode(
```

```
    'P)h>@6aWAK2mk;8Apo$)IktlX005u^000XB6aaK`VQFq(ST1gGc>w?r0H6Z^000000D}Vn000000G-JFQ_}?;!0~YzY=9g==Y&XhJ79<?P+Lz&d|0HoP>&SM'
    '2NI?X(A<!XAf&F9gxYZsB1plJNq0ihIRZpQ_$**23=qcCzMEScGRBv|9&{@)AhLg<_iyi4)x0ld$%ox21(ae|ra*E+%zB5xip_e970F;_<%;vgd{KI?I8*SS'
    '@8X}92%ftnhkr)!{KI3SqaztHT0%1?F#czrlL<L1=1>jXSP9_`ra#1EFBN&dUp#}93U`}z=Uq7W<3m;jT?g?mc%DzX*@x|yc(1GbZBQcI^|rK04V2BBL^Y+2'
    '2>OV&sD>fT|N5`c9=}Oc-;Ux)Kh#5%KK0C}5=2f&UT^G<B|Itrr%Vx`M7!#hh-(FZfcVH+w$FYOzHvTfU7#>wvZ_WXWywJL>hZevBpDQT^trs*DhO5Dx6wZ~'
    '!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)l;5d}FHHXl{WZG7'
    'C;jap_g;LHbT)#a(VMt#9tJ9t)!vXuq6qn={f|sQnfIcF9an-Xxcex}zY}Csq9U0wi6sGLmP$Vj62*^s)Hnt9+sTc3wi*r;gfKt%AR%64&EI#Z8d7FHaQU@N'
    'frB|1=vY_4L7yh-ug^wd53_iglQs<lTeG<iat;hn&RT6P7JRVASl@0mg7!vvm?GPVu0}7Nrg9cDLWkVHE$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*sTO9Z4Fo^1>!x}a9VXWtVJ<Lb(^`50#rcGzT7i~2cO`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(iOYe'
    'O?L<t%BCWTWlE8&iF)a%+|5NB+ODeRWn$XRgdeh1hY^vq1VIv~=?xX@wI;3Ezp&@q^L%_{k)NK9KC^=$Cgjn0DWYT{?HH97lzNoLpwd$L!fatuCYLYdrTo_;'
    'k}^ao+g+5Fl$EmW-u_G`gX-_YpypHmCvWQRlTzoBC(vOzFCY7KAgjDN%<$4duk3PdbIm0DDZgC&UhWh|#(pJ`*DgZi?~DzubqGv7CqC?ppGEf5&gcODRk&;S'
    '&hcASfrB($S$FR@kY=?lb<(Gib8=If@uL-N#|GT}&W>R30q<aG>0{`8IsfaOu5M&p-|t>h-hh_+8izM_D3Rsc_<o7_Dct(U@d$C!3=QvkbQpRTP*GjTUhg)d'
    'GN9TR5<P>VBG&+cUW?hxvb3mHBNCk}<wU1Z@Y)r6?}N5sP(6F>sn|OTt97sDDBr1pNp>0ITsLF<iDvt`*d9pJz8yH&Z-mG%-MjouW=M>Rjk#a<3?38&>eoZf'
    '&{rmIjyOC4H&!gSPBEW=x>BadcI?4#oQHRb_!=0WmIPe2tpQ>3cHzDwb9l<-w4-mrAX?+aI~%#(5O<|y;$+(+NTkSFyD#eS!kM~1#t3?7sN%mRuD3y4aCoux'
    '!8q88>=k7BUl9H67Y=0-4aWMYifd-Cg70`CAwAcC50l9;1I`9a=(4diTV{w;hBh51Yhc3eIlD)qg{lq<otU}+L0rw}WgcdXu?pzZ6g3h{1LFk-8&uF~KWK9;'
    '7+##?dMTqFcmHBoT$auw@4{zb;h1oxmb@O}G>841YXz;dW9Y_OQJi}*gX%it0Qt5SoL+x2*<|m4wUDAs*C#(img;=r7PSeYu7o*DLe$uj_xy&}-~uj@vWxVj'
    'X)G^x&s^R}MAP?#!2`AZxUc-+T$SAjlDi8(c{&awpWPX@*G&f1A*5@R7iNq&xUpH?G6Y-eKBS821<Y?%vKP0%S$B%3Y-__1e4JF@a<~REU%Dt*?Rwnb;9a@&'
    'wHnxLtM(DOAO6lwpHeclu;-FFotH2Ql$kZZv%GoC2`jxrXx2bq`#a}@p7#Qm^6{vaGlWM4>+XkiP57GV!8t@XVOEeqSdj7%=i4Q2d8tkCk*RUy)yNQByr~*V'
    'RM*14P)h>@6aWAK2mk;8Apn6>1aKG%001Bm000UA6aaK(b97%=E^csn0RRvHAP@im000007zzLY00000omcx~j%6R8w`g=49gOr^hM}ny%_{j!$)TvxfsiKK'
    'sKiWZBW<J_t;k_fVx=^dEEYD)XUy5mFx$-O+4<SCqeGF%dtdkG{tMpchwVAs_jO;__xt^vmOS^;u&`H}dCl;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??$c5J5_-`6Hu}_B=etN%={huzcE*R5D^Xd9yjr=QoTI(L|8<{sH'
    'WbKQ8H*lTxZG3S9KU}cq@VGxe0BL}5e3N&G{64)ZUgVI7j9tvj<QZK_HN)_>#7g#XyBUD(L>jrujkC#KBdQw)YDx3Z(2QrZranT`=WZbyz8^oFHEgxsdd)}'
    '&($+4=$`$lJ?oj=uk6)hx75?zeXX{R&yJ4%q}#zdc9~>0Tt~CY?D=&p`Ed8ssgvrs@r$IAql4;bR=d`<juY3nzp-13I`-dIx_m@sE!$hG&fZnai-l8n{<x-='
    'hbw|d9-UiDv-;KXwVda*J0YNdEyvxKbk_*KT87w#@UTsf_W1FPhxKOre|Tt~c6FtPDdsWndpL7^@Z#ZPJbcl7e9S}ZA=}or^YErmKMgx_xrXNH{}k0QYi+xk'
    'UANRQIo);J@Z=gsTf3O`ZViXzn*!Ic%b7<G<@c(g+08#~YFPYt&R<>5ySdR`$W61WS!r%Y`nUW2hlOt9A(wTln`U=chPioh=#=SQdb*i6`VsH2Tije{?Q&*q'
    'HRsI8dFY1&)f_v|G_IOvw^tIY>67By)Ok`hJ?qcTdwx(gcW*NrtmdlyZ~yvz^J>Nx^?vGbMHOGLcK>%y6?<OTIl1$iD&Ev=L9^%QR<YDNMcnu*ZtLKC|M#I)'
    'w1$g$9jiFi9*9Z~8TfXM|Cvhqo3{Q@$!pfB;#O9&$drFpC4ZYezi-~?N+y{yJzB}A6>X#Z+f_2u9OrTc%_(DxDtOU6c5?;u@_S_*T2?{(=XWYNx6TY&1;<`V'
    'd~aB<3I<!Jj%`yxpS#<B``x*6di+g6%V|!3X+t?zVe)I87M3%1STCzo<s31_+*r=QH+J0nU0^v|yqCB!_vUiCtZH2HlreMh-KXw5P{ySxW`N5$#j47CiDev|'
    '_10J4PAp@m0}nr#JFtvbi%g@+SXh}o>b~Y>OuO(&{j(LNG*z0BQ_6_j&4x<Z@7~itADmN4Q?0+kOL;!zt>b>7rOX}p@SD$eETs*je_c#?J$p&YX&3Er&vMaJ'
    '@6WGYw8wImi#01wRXsD>#T+v}kGlBH?{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|O_$#PxD<#7fWCpmOk<7?HopJX$;;U!GZy?K)bRZeo7gOu<d3#ENqmN38L<_EvndxG;GAMi@mM<*C!'
    'q4V(pCs-E~{>!5OoZ#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~5IJHi$MJTnh1vzX!tr_|Vk%udbC5BV(g'
    '3pL2+g1!dn`FuakJoqpTd$Z+X4()jE*Ec@SWBSXt{^!mc^4QVPef%LN+QgO1Mnf-gxg2Q^)<I^QagI62ZfiDQY~6T(Usx=f_0|DiZfzR9pW#DpZt#iR&;1GJ'
    '0Q(qZ#%A0;F1OpdmtFx@*+=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-`+)5%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+zhbC4_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&ClbBxPf?YYK6%{jVh?vX7VV3cryCgB8$!VUa{BV-6y2ouiW7Va=dIE1fo'
    'i8SF9Lxfwngk!`C*GLr3Q6Su-qj&&six;p}Jb@d;8weARK(urO)#4fS7VjWIJcNVdCHRV`5Fy?|ns^LXwEzZ-=de_~2bXvdf#OBPiYJjR-o!29QH&C=;(PHd'
    'n#8;4D;`Focp1mU)9@2-W3qT08RB&`7tbS1ypOM?18_?h&|5lzInoW}N=M);UBLwD4AP`KxSD3`5C%z?uv9t)mvjsFOUDo^UBfQv9B!BHVXSlzsnSJUkWM00'
    'x``y|D2k=4=q#PZH0ds~q{Hx*F5@NXG*YD7sFRLkrF0!TrSou0_u=>eIKBXmPk`ea;P?nQz5-418Ms@%1AXN~@Va~n=F6wxSNRqc$j9J{d=32Mb1+c82b1N4'
    'uvoqbTji5*M!pF*%15Dxd=<jvvoJ%x3oGTruv5MaZuvCaCf|nM@^Kg~Uxx(wJgk-PL#})v&dV3VS3VK_<QowoABhF>l}MA%M4@~quF8j^lYA-Erb<kaZ-wJy'
    'kttsbmwYZ-$oC>pJ{ZI0i{bcWIKCN<kA~x`;rMJgz8j7YhvUn!Rz4lM^6fY$ACI>3^>BPXBINrqUpWA2$^|&CoPd9n8{n@Tfx*fZn4+A4CCVMhP!54hxdcv5'
    'L7;LAhAYP)PPqo(DCc08at}PpLGV#7La=fY#wa&oj&c-Im8+1aoCPO$p`CIVpj-wgry)tX4Ni_jv2q=poCkm9J~%lLPA<eU<wQ8S5l)Uo3*|~YpqvROcOp(X'
    '6swg>;p9}*D!0PPu?SYKg_CnJSGgB!m4o5rVq8#8hOcrnoE(iP<!U4;XT!<eC{_-KlgsfxP)h>@6aWAK2mk;8ApnKqayn=a001Bm000UA6aaK(b97&ME^csn'
    '0RRvHAP@im00000Xb%7Y00000otODL6pYu#QzY9%;*q6HlD$%t@|06(AuY6^WF1>b3ME7#Yg!SNL|L+veK{CoUuK3emccN#EZJIA)Ytnjyyv&`xvq1a`#Se8'
    '_j!Ki;^}jy9BVw+c*_%QZ`s_G*CfavvO6HJLXfv}z3G0_>biyNO``4p@#9v`w`^DWEeER`wyV9uK2=o}!X8Bxf-m9!S-b}K>b6+MXW;Yee5~_^&tXkkhOa<|'
    '1!i+3Yw&nQBV`;e6~vf=L)=m<2Rj~9&#YkvoUntn23KZI#M80=gCl4pe!wf}DYn@@4sS$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$Nf^i})^pak5U<dSRJf^Gn6~qF<01KcoRc{JNA7SsaB(qnG=}Su`#7JIz{zS1Q|(yz^Irq@D9NPF'
    'ey>B^^d~p;Rw_R3ZD-rYv>=7hS}^M910sHA;_vE;G5*F#XJSh(-tLoFE}*0$_2?7jR_iKUN<iNtg$T&;>-XlQmE)$@d&He%lhB2=?^@_a#0z2fFaG;B9}i4L'
    'tTCPbgmPc1P%ym^2|<5VBpsVEeXdqvqBS3b4&O=S`bNS7_MV@1N93UJ+sPSy!7h{tzw}@ylZrAy(-S>@C3xtZ_nv00RH(Ca|DmmtiBg@(@6>vlaQeq4-UACA'
    '$d{z^kj|Nhm7Moyhhj+hftAx3yR#6FXKa5|x1$<2$NM~7RINenD1-L}2T2elr`E1uSA)xAdfa(CYVnD8X%IS)a5`jb`KjdwoXF7?v_77Qt@ZZ)!@6YTUoXoU'
    'WtM|`#Vv~)w2F}+IPxif#2!Qgg=dq_<zYkMo!=F@tr&IRU@K{{6^Yi{8l9=h5ISGl>#~i6qGAU<OHMXpuIHDeLarKg&S_*wMrJ`etuWx&{YG5!xV~wtaWy)}'
    'KH`_tD?<Zwd#B+M5}xO5zbw#RjjeYc?aFVe#pP^6-pGGb(MQEr<3f5Ijz?*BSw^KIIUw}!{I+U*{p1iU<{TMn8uZ_U8GXWZX6(+0fFfWSSG*%hBp|>3KRX-V'
    'lc39hkdRJpfDO8GN8-6+z&WPz$imNDywEOD-Na}FOEvbl(~5MM^Rm7~lxqaGf};ufY(1#Ftv6bZpg?!*?%)T*6rfi5P#t3`U~8_8zL8E1h~maMzdkBh`KFqw'
    '8#TkdaHlhrNCy0YqoKz4>%jSsP2G1PIyjY2$ha)i;2zI~2u)uKD7PQbc@I6H;N~UM7SsiDp(z^<ST%s^Be!c;cp2b+KJV4UKTL2w{hn$}rh)nt(fGU`4JIY%'
    'e}yyK!N+WJ^A3$ps3hrh&d|GoU=^})JE;wR>2`>}Rd0a6L_ekJOEj?Ee6Wanx*iU5aa=AI>V^h3duzovDy-Ad2^FE#K<J`*Vw)BXDk9@fY!U4MjotCKdcV71'
    't}5=H=1&^TYG;nTe>4crk+vo0g(&dY)s&H%MuW&2+fZI%8Z-wc9hP`Xf!Y0}z}F)*$OuSCo#CcKhK*xQOI{C@2B+tf#!8`C%D!Z5Lle0BXGjT;H9}a5W=Em{'
    '9hmHO0k$%Y!0KI@E7+R|$)Wza+d5jHYhH26TCWM7OLL0kakfCfIdLK8N+F165BlUYy5MZ_{HO4O8ek?GA2ALf!6YM?%^{HwYzM)64`OoQtYAyPvQRvz*%{8i'
    '6fXk>?IJtr&}5kVdc5pq799rLYIX(iXJNy$U)%-1L%}JqN6EDP6I9$>*>^U#9yXnJ>rdd!hl9Q9cdKRVA!i8^m2)G2r`L1iQL#$+Xh8PArbB~T+L7kyoF_1^'
    'OsNT8B7u(1j&IV>szFp%H<y@Iij_Ldu?@vJP`7>ffZkXewAXDMtemTXlZ3hfi>VjjB`V;cOlbzDYyb1Tb7e4S(VOBxZ-j{Qn@!9M%|Jc4QSJDpd}vQ4rkPYy'
    ';lCpRiK)bLxYM7q)#P9?s9SOcD;tsEF(HIDD$)Y(@2{J2f2D$($6FP<#wNI?9NBwSsT<m3U86Nrx`6E{FGRdVg*Jowp&e;V$bDUHe*fbj7yx~KV4)l02<zrv'
    'NA$pVVj!`2g#wnFgX}JB?t|<*FXc_N`yqE%JbSm$5XcqpX|cM?0-{o4y-~<8)aaXZx!a6Gfv-zh-jgB7NH)%Lb{mGvI>V8!F)aAL*@W8XGYXX_M{Ke!KSNBK'
    'zO=FH5FC(fYDj%I3Ww8neckZ|0b$Y>E0v$8!2BDl%1~wsq&Mj74*W6$?>-(J+WmYIUSWh-fX@s>w^<16>6nG_&FRD>@;GGd{n@IsaSqgt7jBFQe*s^KpNbD)'
    '2DaK*tbE8A2gUPA2N@Alus1dP4RK-tisNMjx4i!XTF3Rb^^j*E#<riAe|28}QqqrOp|h~8!yL~zH4n``AzC6yi?HF@e)4adIas*2&+~4|JXAPlJ^rpY3+tl7'
    '!Y|A&f@?bU$TEE%m{Or<o!`zvyVUaC_Qlmb<du3ct3`OSXrxL`orgn|D!mo^8K_zmKP$-p6{e+FVO>w=!J?Q`Ql()5w974>(o1I{!H-&6PMw7hcZ}5>{O2IJ'
    '&6Z=LZyth$xzd9K=787R`C5+X6fCUmeIa5r3y--<@}32*>S`L+Ichlr4OTMU<IHIYGv|qVkvR^0n>uMf-Nqo(_r>=ILK7fb)nt{aJ^|N9+-?NtO+aA!O62Y3'
    'afo-%j$L^>1qtT+lgT&=GhW9Ag_g$Qd#n1x?3--J8PFhfDvpEljMCJ_%?t>XsQke%(hstO>3g@|U-diBgBb1I2g*4fkB1dl;FKX)>GQ4^M*N-qXutbl*wHvL'
    'E|CGH_TI$}jr3K0%p>OHZr~2oaR$Q<Fh6m0K~k0q7P4Y4W$x9$<L<*$fAj&?a10ZbU#r7)lVh4sZOdR^n`;KBwWDP<kJ9axT5Ryt%2C5wWC#Cq+M9^@Wa!~~'
    'FYX?ELACum{qi0DCvg1f(N|UI|M0!WjescBuhZ1}N2mtPMQzS)Unb+74a&LObMldv&a6Hj-i?38jE?H^uyBIw)y8}NJ!m$1t9ota0Pa2+2C@&Cxb!p3l+R-b'
    'nMZn$US##)gDWkAJuZEy(YIXYzlMdsLfV2QCI`^1H)l_U-4GhaKULD@8NjG$cK5!wL#V?eyi68lqcqQKbn%Hnthw@ElzK-${+%(4N}cXUNe*3O>7+6ICNsiQ'
    '=s1e$YP1ro^Ph1@|NM5V&Jpx->I)8J4Wn>|9Lc711YhNJWS6)zF`<ax@?g;@+6SwWPfm<u$^iR$Ui>hwYltImnqs5ywi30)s9_x2>A-hfkBxCvYsBp=*14t9'
    'U=!UiiTgMN53p3nQ0;a5@V=L$=qYbB0PjaIbeqX!C20h$mA|f-wGUx9Y0H^=%PjoC5Z{=oJBAnfzwCOd$-)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}}lYnj+87LZ298x2KJrDCDlmXQecNfgL}5'
    'RsK-X(&xo`ZI>~;8(#Xe&0-jR-rOREuk<2s!~!~O>%>y;_#^6!9^@#rQB_as!PpZU?F~G7@R-o<sLJLcJbfZ)v^1N79wAb~a<WY184`N$6h}dUoGXo&$qXE}'
    'B|DWHQjoaTpyv~*6IDOnbMo_Mpq^ZvbM7HJ$_f}C-|~GB<<~C1*y7oR15F1%g!T1!NcUG`ms|^~n-?8zNoV4##I@RDyE-u1MuiyI-HD^qjQh3PRGdxukQy~j'
    '$5%;9j+aHd(bPhY9O=-4Q?qfO&)BW<&Ln1NT`LNGeEa=gGX+ES6eaxk^kCE6q~(BNH=eGNI>))E4rkV%D)U;sYrs^<yIp=vjO6R9l)O2J+<6iY>PxA}E2S>$'
    '$Hzb&zWBK6L;ZO7_ugZ?>pJm~D|>g29Ru}?1LTb&hA`emyY+8xCq|sqWV3fuP`GT1w|9RP#%O3g8-3i3OTif<Mi&?uc&3Gtrre4ZhT)!k<qWi_sv;BCGtiA_'
    'IWAbq#6k~ab%k0M4u+3T`<BzNCeyk2@G%zhI|YefUPr~k{nB&q2>rO<j~bP!(S?gUK3{($(2EnVrd#4h+A+dA*^lxc9dC@9cuSVn;=QXwGXGxdN0BmC!;^px'
    'Jo>hFP}P%;DQg`C$DeeduC<D$kPjKv&4k?h3Mu#?OU#Q;rw@-<{?I<fL&1~MBJ+)5badu%6bmt-V~fF$!xoBlDEm=#g|*a)x|fU|6im}_%fwGpG-yZWZ2ccy'
    'qZ)KN^r6J}B^iV7oVk$|-G*zq7YDKuTe1CY?c)oT9T-cC___TS1?$A7g1h+Ikmxi#$q=c-Zn4_YZ^a+sC+{K7`pia5K1G>VI!40bip>I|Je?TV{?tLMJQ2GS'
    'W%{zZ+fY6E;JIyuwfK-=w6u3MU$1_bX0Z+wJS{!h^gB5g4HRFw1}<g-S8lLCy<`pi`y}_AJbw{-mUE_W{gsL_uh;JUw5tht!`{5Xi|@e3X<xa8NgO_Q5Zcsz'
    'H4GQi$q8F`y++$lzclr3B!akj_?r;xMo>%I8e}5T00h)Gn*8F6(RWL9Uwo|wBfg9GCGQtPjEF)2(W4oznr7{_d0GWy{@!VOjT(TbSwM#rQ3I#%d<s6{l7_aV'
    'Ti=L1>VaR#RoYcG1N3=1zZx`BAo<tbEn;$2aHM+D$&^wJh5WI__M4jE!4g$a<zFhaPU}68&}@YAgtm`C&fV~RzvRuUh20>VvN%*@)&uEHR#c{b6TC8M=12=E'
    'fcx?Ox9mt=plUe6URr7Zm+aFY<<_?YqunPv+`0pj`Qjg4+fV?+v+I6D&h^0hN7m0q)aW2`Yi}{kP@!}xIzroy4qg|VE-9Yw0MVf4waS4M(Es7?F_+W~ku*NS'
    'jBFh+J)W;S5l4mT2^SuM;A%d;oua0d)CK<oP)h*<6ay3h000O8001EXu*W&Jg988npaTE^3jhEB0000000000fB^si004AyVQFq(ST1gGc~DCM0u%!j00008'
```

```
    '0000X06#iWD2D?80H6Z^01E&B00000000000Du9&0{{SYa$#w1UwJNWaCuNm0Rj{Q6aWAK2mk;8Apn6>1aKG%001Bm000UA0000000000004jiga-fsbY*jN'
    'Usx_~aCuNm0Rj{Q6aWAK2mk;8ApnKqayn=a001Bm000UA0000000000004ji*bx8#bY*jNUwJNWaCuNm1qJ{B000C41OVay004X;00000'
)))

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

# Converte os dados para tensores
X_train = torch.tensor(train_X, dtype=torch.float32)
y_train = torch.tensor(train_y, dtype=torch.float32)
X_test = torch.tensor(test_X, dtype=torch.float32)
y_test = torch.tensor(test_y, dtype=torch.float32)
```

In [ ]:
```
# Função para calcular a log-verossimilhança marginal negativa
def GP_log_likelihood(x, y, gamma, c, sigma_noise_sq):
    K = rbf_kernel(x, x, gamma, c) + sigma_noise_sq * torch.eye(x.size(0))
    L = torch.cholesky(K)
    return -0.5 * (y.t() @ torch.cholesky_solve(y, L) + torch.logdet(L) + x.size(0) * torch.log(torch.tensor(2 * torch.pi)))

# Função para calcular o erro quadrático médio
def mse(y_pred, y):
    return torch.mean((y_pred - y) ** 2)
```

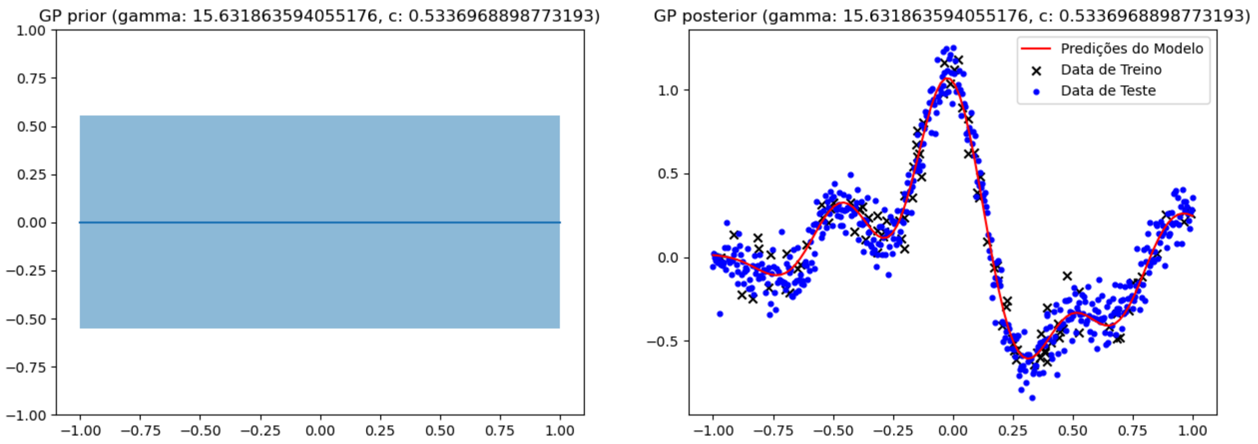
In [ ]:
```
# Inicialização de hiperparâmetros
gamma = torch.tensor([10.0], requires_grad=True)
c = torch.tensor([1.0], requires_grad=True)
sigma_noise_sq = torch.tensor([0.1], requires_grad=True)  # Ruído observacional
num_iterations = 1000 # Número de iterações
```

In [ ]:
```
# Inicializa o otimizador
optimizer = Adam([gamma, c, sigma_noise_sq], lr=0.01)

# Loop de otimização
for i in range(num_iterations):  # num_iterations é o número de etapas de otimização que você deseja executar
    optimizer.zero_grad()
    log_likelihood = -GP_log_likelihood(X_train, y_train, gamma, c, sigma_noise_sq)  # Função para calcular a log-verossimilha
    log_likelihood.backward()
    optimizer.step()

    if (i + 1) % 100 == 0:
        print(f"Iteração {i+1}: Log Likelihood = {log_likelihood.item()}")

# Após a otimização, use os valores otimizados de gamma e c para calcular a média e a covariância do GP posterior e fazer prev
x = torch.linspace(-1, 1, 100)[:, None]
mu = torch.zeros_like(x)
K = rbf_kernel(x, x, gamma, c) + torch.eye(x.shape[0])*sigma_noise_sq
```

```
C:\Users\iaram\AppData\Local\Temp\ipykernel_21624\565260635.py:4: UserWarning: torch.cholesky is deprecated in favor of torc
h.linalg.cholesky and will be removed in a future PyTorch release.
L = torch.cholesky(A)
should be replaced with
L = torch.linalg.cholesky(A)
and
U = torch.cholesky(A, upper=True)
should be replaced with
U = torch.linalg.cholesky(A).mH
This transform will produce equivalent results for all valid (symmetric positive definite) inputs. (Triggered internally at
..\aten\src\ATen\native\BatchLinearAlgebra.cpp:1703.)
  L = torch.cholesky(K)
Iteração 100: Log Likelihood = 35.39397430419922
Iteração 200: Log Likelihood = 35.08833312988281
Iteração 300: Log Likelihood = 34.84060287475586
Iteração 400: Log Likelihood = 34.6528205871582
Iteração 500: Log Likelihood = 34.51133728027344
Iteração 600: Log Likelihood = 34.40361022949219
Iteração 700: Log Likelihood = 34.32072448730469
Iteração 800: Log Likelihood = 34.256385803222656
Iteração 900: Log Likelihood = 34.20603561401367
Iteração 1000: Log Likelihood = 34.166404724121094
```

In [ ]:
```
# Plotting
fig, axs = plt.subplots(1, 2, figsize=(15, 5))  # Create two subplots
# Parâmetros otimizados
gamma_opt, c_opt, sigma_noise_sq_opt = gamma.item(), c.item(), sigma_noise_sq.item()

# GP Prior
axs[0].plot(x.detach().numpy(), mu.detach().numpy())
axs[0].fill_between(x.flatten().detach().numpy(), mu.flatten().detach().numpy()-K.diag().detach().numpy(), mu.flatten().detach
axs[0].set_ylim([-1, 1])
axs[0].set_title(f'GP prior (gamma: {gamma_opt}, c: {c_opt})')

post_mu, post_cov = posterior_pred(x, X_train, y_train, gamma=gamma_opt, c=c_opt, sq=sigma_noise_sq_opt)

# GP Posterior
axs[1].plot(x, post_mu, 'r', label='Predições do Modelo')
axs[1].fill_between(x.flatten(), post_mu.flatten()-post_cov.diag(), post_mu.flatten()+post_cov.diag(), alpha=0.7)
axs[1].scatter(X_train, y_train, color='black', label='Data de Treino', marker='x')
axs[1].scatter(X_test, y_test, color='blue', label='Data de Teste', s=12)
axs[1].set_title(f'GP posterior (gamma: {gamma_opt}, c: {c_opt})')
```

```python
axs[1].legend()
plt.show()
```



In [ ]:
```python
x = torch.linspace(-1, 1, 500)[:, None]
# Calcula a log-verossimilhança e o MSE nos dados de teste
test_log_likelihood = -GP_log_likelihood(X_test, y_test, gamma_opt, c_opt, sigma_noise_sq_opt)
post_mu, post_cov = posterior_pred(x, X_test, y_test, gamma=gamma_opt, c=c_opt, sq=sigma_noise_sq_opt)
```

In [ ]:
```python
mse_ = mse(post_mu, y_test)
print(f"MSE: {mse_}")
print(f"Test Log Likelihood: {test_log_likelihood.item()}")
```

```
MSE: 0.00912289135158062
Test Log Likelihood: 109.20968627929688
```

Exercício de "papel e caneta"

1. Na nota de aula, derivamos a posteriori preditiva $p(y_\star|x_\star, x_1, y_1, \ldots, x_N, y_N)$. Por simplicidade, deduzimos a priori preditiva $p(y_\star, y_1, \ldots, y_N|x_\star, x_1, \ldots, x_N)$ e as condicionamos nas saídas $y_1, \ldots, y_N$ observadas no conjunto de treino. No entanto, também é possível obter o mesmo resultado calculando a posteriori $p(f_\star, f_1, \ldots, f_N|x_\star, x_1, y_1, \ldots, x_N, y_N)$ e, então, calculando o valor esperado de $p(y_\star|x_\star, f_\star)$ sob essa posteriori. Deduza novamente a posteriori preditiva seguindo esse outro procedimento.

(Dica: você pode calcular a conjunta $p(f^*, f_1, \ldots, f_N, y_1, \ldots, y_N|x_*, x_1, \ldots, x_N)$, que também será Gaussiana.)

Resposta:

Para calcular a posteriori preditiva, vamos considerar a distribuição conjunta de $f^*, f_1, \ldots, f_N, y_1, \ldots, y_N$, tal que a média é 0, pois assumimos que o GP é centrado. Sabendo que as observações y são relacionadas com f através de uma relação de ruído, normalmente assumida como normalmente distribuída, temos:

$$y_\star = f_\star + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

Para encontrar a matriz de covariância, iremos calcular a covariância entre as funções latentes e as saídas para os 3 possíveis casos. Seja $i, j \in \{1, \ldots, N\}$, temos:

$$\begin{align}
Cov(f_i, f_j) &= k(x_i, x_j) \tag{1}\\
Cov(f_i, y_j) &= Cov(f_i, f_j + \epsilon_j) = Cov(f_i, f_j) + \underbrace{Cov(f_i, \epsilon_j)}_{0} = k(x_i, x_j) \tag{2}\\
Cov(y_i, y_j) &= Cov(f_i + \epsilon_i, f_j + \epsilon_j) = Cov(f_i, f_j) + \underbrace{Cov(f_i, \epsilon_j)}_{0} + \underbrace{Cov(\epsilon_i, f_j)}_{0} + \underbrace{Cov(\epsilon_i, \epsilon_j)}_{(*)} \tag{3}\\
&= k(x_i, x_j) + (*) \tag{4}\\
(*) &= \begin{cases} \sigma^2, & \text{se } i = j \\ 0, & \text{se } i \neq j \end{cases} \tag{5}
\end{align}$$

Portanto, a distribuição conjunta é dada por:

$$\begin{bmatrix} f^* \\ f_1 \\ \vdots \\ f_N \\ y_1 \\ \vdots \\ y_N \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{k}_{**} & \mathbf{k}_{*X} & \mathbf{k}_{*X} \\ \mathbf{k}_{X*} & \mathbf{k}_{XX} & \mathbf{k}_{XX} \\ \mathbf{k}_{X*} & \mathbf{k}_{XX} & \mathbf{k}_{XX} + \sigma^2\mathbf{I} \end{bmatrix}\right)$$

Onde:

- $\mathbf{k}_{**}$ é a variância do novo ponto x_*
- $\mathbf{k}_{*X}$ é o vetor de covariâncias entre o novo ponto x_* e os pontos de treinamento X: $x_1, \ldots, x_N$
- $\mathbf{k}_{X*}$ é a transposta de $\mathbf{k}_{*X}$
- $\mathbf{k}_{XX}$ é o vetor de covariância entre os pontos de treinamento.

Vamos agrupar a matriz de covariância em 4 blocos da seguinte forma:

$$\Sigma_{1,1} = \begin{bmatrix} \mathbf{k}_{**} & \mathbf{k}_{*X} \\ \mathbf{k}_{X*} & \mathbf{k}_{XX} \end{bmatrix} \tag{6}$$

$$\Sigma_{1,2} = \begin{bmatrix} \mathbf{k}_{*X} \\ \mathbf{k}_{XX} \end{bmatrix} \tag{7}$$

$$\Sigma_{2,1} = \begin{bmatrix} \mathbf{k}_{X*} & \mathbf{k}_{XX} \end{bmatrix} \tag{8}$$

$$\Sigma_{2,2} = \begin{bmatrix} \mathbf{k}_{XX} & \mathbf{k}_{XX} + \sigma^2 \mathbf{I} \end{bmatrix} \tag{9}$$

Utilizaremos a propriedade condicional das distribuições Gaussianas para calcular a posteriori de $\mathbf{f}$ condicional às observações $\mathbf{y}$:

$$\mu_{\mathbf{f}|y} = \mu_1 + \Sigma_{1,2}\Sigma_{2,2}^{-1}(\mathbf{y} - \mu_2) \tag{10}$$

$$= \Sigma_{1,2}\Sigma_{2,2}^{-1}\mathbf{y} \tag{11}$$

$$\Sigma_{\mathbf{f}|y} = \Sigma_{1,1} - \Sigma_{1,2}\Sigma_{2,2}^{-1}\Sigma_{2,1}$$

Ou seja,

$$\begin{bmatrix} f^* \\ f_1 \\ \vdots \\ f_N \end{bmatrix} \mid \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \sim \mathcal{N}(\mu_{\mathbf{f}|y}, \Sigma_{\mathbf{f}|y})$$

Usando a propriedade de gaussiana, temos que a marginal de f^* é:

$$f^*|y_1, \ldots, y_N \sim \mathcal{N}(\mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ \mathbf{y}, \quad \mathbf{k}_{**} - \mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ \mathbf{k}_{X*})$$

Finalmente, a predição de $y^* = f^* + \epsilon$ onde $\epsilon \sim \mathcal{N}(0, \sigma^2)$, resulta em:

$$y^*|x^*, y_1, \ldots, y_N \sim \mathcal{N}(\mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ \mathbf{y}, \quad \mathbf{k}_{**} - \mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ \mathbf{k}_{X*} + \sigma^2)$$

2. Quando trocamos a verossimilhança Gaussiana por uma Bernoulli (i.e., no caso de classificação binária), a posteriori para nosso GP não possui fórmula fechada. Mais especificamente, a verossimilhança para esse modelo é dada por $y|x \sim \text{Ber}(\sigma(f(x)))$ onde σ é a função sigmoide. Em resposta à falta de uma solução analítica, podemos aproximar a posteriori sobre f para qualquer conjunto de pontos de entrada usando as técnicas de inferência aproximada que vimos anteriormente. Discuta como usar a aproximação de laplace nesse caso, incluindo as fórmulas para os termos da Hessiana. Além disse, discuta como usar o resultado desse procedimento para aproximar a posteriori preditiva.

Resposta:

A probabilidade preditiva de que o evento ou a classe binária y^* seja 1, dado o conjunto de dados $\bar{y}$, é dada por:

$$P(y^* = 1|\bar{y}) = \int_\Omega P(y^* = 1, f^*|\bar{y})df^* \tag{12}$$

$$= \int_\Omega P(y^* = 1|f^*)P(f^*|\bar{y})df^* \tag{13}$$

$$= \int_\Omega \sigma(f^*)P(f^*|\bar{y})df^* \quad (*) \tag{14}$$

$$P(f^*|\bar{y}) = \int_\Omega P(f^*, \bar{f}|\bar{y})d\bar{f} \tag{15}$$

$$= \int_\Omega P(f^*|\bar{f})P(\bar{f}|\bar{y})d\bar{f} \tag{16}$$

Onde $P(f^*|\bar{f})$ é a distribuição preditiva de f^* condicional a $\bar{f}$. Pela questão anterior, isso é:

$$f^*|\bar{f} \sim \mathcal{N}(\mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ f, \quad \mathbf{k}_{**} - \mathbf{k}_{*X}\ (\mathbf{k}_{XX} + \sigma^2 I)^{-1}\ \mathbf{k}_{X*})$$

Visto que não existe uma solução analítica para $P(\bar{f}|\bar{y})$ que é a distribuição posteriori de $\bar{f}$ condicional a $\bar{y}$, usaremos a aproximação de Laplace para encontrar uma solução aproximada. Dado que a verossimilhança da classificação binária em GP's é dada pela Bernoulli: $p(y|x) \sim \text{Ber}(\sigma(f(x)))$, onde:

- y são as classes binárias.
- x são as entradas.
- σ é a função sigmoide.

Isso é feito maximizando a soma da log-verossimilhança de Bernoulli com a log-densidade da priori Gaussiana, seguindo os passos a seguir:

1. Calculamos a moda da log-posteriori, que é o ponto de máximo a posteriori (MAP) para a função latente f:

$$\mathbf{m} = \underset{\bar{f}}{\text{argmax}}\ \log p(\bar{y}|\bar{f}) + \log p(\bar{f})$$

A verossimilhança Bernoulli e sua log-verossimilhança são dadas por:

$$
\begin{aligned}
p(\bar{y}|\bar{f}\,) &= \prod_{i=1}^{N} p(y_i|f_i) && (17)\\
&= \prod_{i=1}^{N} (\sigma(f_i))^{y_i}(1-\sigma(f_i))^{1-y_i} && (18)\\
\Rightarrow \log p(\bar{y}|\bar{f}\,) &= \sum_{i=1}^{N} [y_i \log(\sigma(f_i)) + (1-y_i)\log(1-\sigma(f_i))] && (19)
\end{aligned}
$$

Visto que $p(\bar{f}\,) = \mathcal{N}(\bar{f}\,|\mathbf{0}, \mathbf{K})$, temos que a log-densidade da priori é:

$$\log p(\bar{f}\,) = -\frac{1}{2}\bar{f}^T\mathbf{K}^{-1}\bar{f}$$

Juntando as equações acima, temos:

$$\mathbf{m} = \underset{\bar{f}}{\operatorname{argmax}} \sum_{i=1}^{N} [y_i \log(\sigma(f_i)) + (1-y_i)\log(1-\sigma(f_i))] - \frac{1}{2}\bar{f}^T\mathbf{K}^{-1}\bar{f}$$

1. Calculamos a Hessiana da log-posteriori no ponto MAP para obter a matriz de covariância da aproximação Gaussiana. A Hessiana é a matriz de segundas derivadas de uma função, e fornece uma medida da curvatura da função em torno do ponto de interesse. No nosso caso, a Hessiana da log-posteriori é dada pela soma da Hessiana da verossimilhança e da Hessiana da priori.

$$
\begin{aligned}
\mathbf{H} &= -\nabla\nabla \log p(\mathbf{m}|\mathbf{X}, \bar{y})\\
&= -\nabla\nabla \left(\sum_{i=1}^{N} [y_i \log(\sigma(f_i)) + (1-y_i)\log(1-\sigma(f_i))] - \frac{1}{2}\bar{f}^T\mathbf{K}^{-1}\bar{f}\right)
\end{aligned}
$$

1. Com a Hessiana calculada, podemos agora determinar a matriz de covariância da aproximação Gaussiana, que é dada pelo inverso da Hessiana: $\mathbf{H}^{-1}$

2. Com a média $\mathbf{m}$ e $\mathbf{H^{-1}}$ determinados, a distribuição posteriori sobre a função latente f é aproximada como a Gaussiana:

$$p(\bar{f}\,|\bar{y}) \approx \mathcal{N}(\mathbf{m}, \mathbf{H}^{-1})$$

Uma vez que temos uma distribuição para f^*, a probabilidade preditiva de um novo ponto y^* dado um x^* é representada pela integral $(*)$. Para calcular essa integral, podemos usar métodos de integração numérica, como o método de Monte Carlo. Primeiro, amostramos f^* da distribuição $p(f^*|\bar{y})$, e então calculamos a probabilidade preditiva de $y^* = 1$ como a média da função sigmoide aplicada a essas amostras.