Lista Extra de Machine Learning

Gabriel Dias Vilela

IMPA, Verão 2025

Sumário

1 Exercicio 1a

Seja

$$\tilde{\beta}' = \begin{bmatrix} \tilde{\beta}_0 \\ \tilde{\beta}_1 \\ \vdots \\ \tilde{\beta}_p \end{bmatrix} \quad \text{e} \quad X' = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & & & & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}.$$

Assim, a função perda pode ser reescrita como:

$$\hat{\beta}_0, \hat{\beta} = \arg\min_{\tilde{\beta}_0, \tilde{\beta}} \| y - X'\tilde{\beta}' \|_2^2 + \lambda \|\tilde{\beta}\|_2^2.$$

Centralizando os dados:

$$X' \ \rightarrow \ X' - \bar{X}', \quad y \ \rightarrow \ y - \bar{y}.$$

Obtemos,

$$\hat{\beta}_0, \hat{\beta} = \arg\min_{\tilde{\beta}_0, \tilde{\beta}} ||y - \bar{y} - (X' - \bar{X}')\tilde{\beta}'||_2^2 + \lambda ||\tilde{\beta}||_2^2$$

Onde

$$X' - \bar{X}' = \begin{bmatrix} 0 & (x_{11} - \bar{x}_1) & (x_{12} - \bar{x}_2) & \dots & (x_{1p} - \bar{x}_p) \\ 0 & (x_{21} - \bar{x}_1) & (x_{22} - \bar{x}_2) & \dots & (x_{2p} - \bar{x}_p) \\ \vdots & & & & \vdots \\ 0 & (x_{n1} - \bar{x}_1) & (x_{n2} - \bar{x}_2) & \dots & (x_{np} - \bar{x}_p) \end{bmatrix}.$$

Nota que a expressão acima não depende de $\hat{\beta}_0$ devido as entradas nulas na primeira coluna de $X' - \bar{X}'$ e, portanto, podemos arbitrar seu valor.

De maneira intuitiva percebemos que, após a centralização, as *features* e o output se relacionam não mais pela sua posição no espaço, mas sim por uma posição relativa a média. Assim, é esperado que $\hat{\beta}_0$ não gere impacto no modelo, visto que ele não afeta a posição relativa, afinal ele é anulado na operação $y_i - \bar{y}$.

2 Exercicio 1b

Tome a expressão para dados centrados:

$$\hat{\beta} = \arg\min_{\tilde{\beta}} \sum_{i=1}^{n} \Big(y_i - \sum_{j=1}^{p} x_{ij} \tilde{\beta}_j \Big)^2 \; + \; \lambda \sum_{j=1}^{p} (\tilde{\beta}_j)^2. = \arg\min_{\tilde{\beta}} E(\tilde{\beta})$$

Mostraremos ao fim da solução que essa função é convexa, por hora assuma isso como verdade. Derivando a expressão em relação a $\tilde{\beta}$ e analisando uma componente:

$$\frac{\partial E(\tilde{\beta})}{\partial \tilde{\beta}_j} = \sum_{i=1}^{n} [2 \Big(y_i - x_{ij} \tilde{\beta}_j \Big) . (-x_{ij})] \; + \; 2\lambda \, \tilde{\beta}_j = 0.$$

Desenvolvendo a expressão acima, obtemos:

$$\sum_{i=1}^{n} \Big(- x_{ij} \, y_i + \; x_{ij}^2 \, \tilde{\beta}_j \Big) + \; \lambda \, \tilde{\beta}_j = 0.$$

Estendendo para a forma matricial e substituindo $\tilde{\beta}$ por $\hat{\beta}$:

$$- X^\top y \; + \; X^\top X \, \hat{\beta} \; + \; \lambda \, \hat{\beta} = 0.$$

Logo,

$$\hat{\beta} = (\lambda \, I + X^\top X)^{-1} X^\top y.$$

Tomando $\lambda \to 0$, obtemos $\hat{\beta} = (X^\top X)^{-1} X^\top y$, ou seja, o estimador da regressão linear, afinal a perda, fazendo $\lambda = 0$, é exatamente a soma dos quadrados dos resíduos usada como perda na regressão linear.

Partindo da forma matricial para a derivada de E e derivando novamente em relação a beta obetemos como resultado a matriz $X^T X + \lambda I$, que é positiva semidefinida. Logo, E é convexo.

3 Exercicio 1c

Como mostrado:

$$\hat{\beta} = (\lambda I + X^T X)^{-1} X^T y.$$

Pela fórmula de Woodbury (supondo $\lambda \neq 0$):

$$\hat{\beta} = (\lambda I + X^T X)^{-1} X^T y = X^T \left[\frac{1}{\lambda} y \; - \; \frac{1}{\lambda^2} \Big(I + \tfrac{1}{\lambda} X X^T \Big)^{-1} X X^T y \right].$$

Assim, podemos escrever:

$$\hat{\beta} = X^T \alpha \quad \text{onde} \quad \alpha = \frac{1}{\lambda} y \; - \; \frac{1}{\lambda^2} \Big(I + \tfrac{1}{\lambda} X X^T \Big)^{-1} X X^T y.$$

4 Exercicio 1d

$$\hat{y}_* = x_*^T\,\hat{\beta} = x_*^T\,X^T\,\alpha = x_*^T\,X^T\Big[\tfrac{1}{\lambda}\,y - \tfrac{1}{\lambda^2}\left(I + \tfrac{1}{\lambda}\,X\,X^T\right)^{-1} X\,X^T\,y\Big].$$

5 Exercicio 1e

Escrevendo X^T como $[\,x_1,\ x_2,\ \ldots,\ x_n\,]_{p\times n}$, notamos que a predição y^* depende, em termos de *features*, apenas do produto escalar entre x_* e cada amostra de treino x_i, $i = 1, 2, \ldots, n$ (do termo $x_*^\top X^T$) e também do produto escalar entre as amostras de treino tomadas 2 a 2, $x_i^\top x_j$, $i, j \in \{1, 2, \ldots, n\}$, (do termo $X\,X^T$). Assim, definindo uma nova operação para substituir o produto escalar tradicional — o *Kernel* $K(x_i, x_j)$ —, podemos conceder mais flexibilidade ao modelo sem alterar a dimensionalidade das *features*. Reescrevendo a expressão do item (d) em termos do produto escalar tradicional e substituindo-o pelo *Kernel*, obtemos:

$$y^* = \begin{bmatrix}\langle x_*^T,\, x_1\rangle & \langle x_*^T,\, x_2\rangle & \ldots & \langle x_*^T,\, x_n\rangle\end{bmatrix}_{1\times n} \left[\tfrac{1}{\lambda}\, y - \tfrac{1}{\lambda^2}\left(I + \tfrac{1}{\lambda}\, M\right)^{-1} M\, y\right],$$

onde

$$M = X\,X^T = \begin{bmatrix}\langle x_1, x_1\rangle & \langle x_1, x_2\rangle & \ldots & \langle x_1, x_n\rangle \\ \langle x_2, x_1\rangle & \langle x_2, x_2\rangle & \ldots & \langle x_2, x_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1\rangle & \langle x_n, x_2\rangle & \ldots & \langle x_n, x_n\rangle\end{bmatrix}_{n\times n}.$$

Substituindo a operação pelo *Kernel*:

$$y^* = \begin{bmatrix}K(x_*^T,\, x_1) & K(x_*^T,\, x_2) & \ldots & K(x_*^T,\, x_n)\end{bmatrix}_{1\times n} \left[\tfrac{1}{\lambda}\, y - \tfrac{1}{\lambda^2}\left(I + \tfrac{1}{\lambda}\, M_K\right)^{-1} M_K\, y\right],$$

onde

$$M_K = \begin{bmatrix}K(x_1, x_1) & K(x_1, x_2) & \ldots & K(x_1, x_n) \\ K(x_2, x_1) & K(x_2, x_2) & \ldots & K(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_n, x_1) & K(x_n, x_2) & \ldots & K(x_n, x_n)\end{bmatrix}_{n\times n}.$$

6 Exercicio 1f

É esperado que SVMs sejam mais vantajosos devido a maneira como os coeficientes são calculados. Enquanto em Ridge Regression utilizamos todos dados amostrais para estimar os coeficientes e realizar previsões, em SVMs utilizamos apenas um conjunto de vetores específicos chamados vetores de suporte que se encontram próximos a fronteira de decisão e a definem .

7 Exercicio 1g

(i) Ao executar o código abaixo:

```
# (i)
# Linear Regression:

linear_regression = LinearRegression()
fitted_linear_regression = linear_regression.fit(X_train, y_train)
train_mean_squared_error = np.mean((fitted_linear_regression.predict(X_train) -
    y_train) ** 2)
test_mean_squared_error = np.mean((fitted_linear_regression.predict(X_test) -
    y_test) ** 2)

print(
    "Erro de treino do modelo de regress o linear: ",
    train_mean_squared_error,
    "\n",
    "Erro de teste do modelo de regress o linear: ",
    test_mean_squared_error,
)
```

Obtemos o seguinte output:

Erro de treino do modelo de regressão linear: 18.090133641655054
Erro de teste do modelo de regressão linear: 15.910489104854424

(ii) Ao executar o código abaixo:

```
# (ii)
# Ridge Regression:
alphas = 10 ** np.linspace(3, -2, 100)

ridge_cv_mse = []
for alpha in tqdm(alphas):
    cv_MSE = []
    folds = KFold(n_splits=number_of_folds, shuffle=True, random_state=2023).split
    (X_train, y_train)
    for train_idx, val_idx in folds:
        ridge_pipeline = make_pipeline(StandardScaler(), Ridge(alpha=alpha))
        ridge_pipeline[1].fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
        y_hat = ridge_pipeline[1].predict(X_train.iloc[val_idx])
        cv_MSE.append(np.mean(y_hat - y_train.iloc[val_idx]) ** 2)

    ridge_cv_mse.append(np.mean(cv_MSE))

optimal_alpha = alphas[np.argmin(ridge_cv_mse)]

ridge_pipeline = make_pipeline(StandardScaler(), Ridge(alpha=optimal_alpha))
ridge_pipeline[1].fit(X_train, y_train)

y_hat_ridge = ridge_pipeline[1].predict(X_test)
ridge_test_mse = np.mean((y_hat_ridge - y_test) ** 2)
print(ridge_test_mse)
```

Obtemos o seguinte output: 15.910432432652605

(iii e iv) Ao executar o código abaixo:

```
# (iii)

kernels = ["linear", "polynomial", "rbf", "laplacian"]
gammas = [10**-3]
alphas = 10 ** np.linspace(3, -2, 100)
hyperparams = [(kernel, gamma, alpha) for kernel in kernels for gamma in gammas
    for alpha in alphas]

ridge_cv_mse = []
for hyperparam in tqdm(hyperparams):
    cv_MSE = []
    folds = KFold(n_splits=number_of_folds, shuffle=True, random_state=2023).split
    (X_train, y_train)
    for train_idx, val_idx in folds:
        ridge_pipeline = make_pipeline(StandardScaler(), KernelRidge(alpha=
    hyperparam[2], kernel=hyperparam[0], gamma=hyperparam[1]))
        ridge_pipeline[1].fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
        y_hat = ridge_pipeline[1].predict(X_train.iloc[val_idx])
        cv_MSE.append(np.mean(y_hat - y_train.iloc[val_idx]) ** 2)

    ridge_cv_mse.append(np.mean(cv_MSE))

optimal_hyperparam = hyperparams[np.argmin(ridge_cv_mse)]

ridge_pipeline = make_pipeline(StandardScaler(), KernelRidge(alpha=
    optimal_hyperparam[2], kernel=optimal_hyperparam[0], gamma=optimal_hyperparam
    [1]))
ridge_pipeline[1].fit(X_train, y_train)

y_hat_ridge = ridge_pipeline[1].predict(X_test)
ridge_test_mse = np.mean((y_hat_ridge - y_test) ** 2)
print(ridge_test_mse)
print(optimal_hyperparam)
```

Obtemos o seguinte output:
Melhores Hiperparâmetros:
Kernel: Linear
Gamma: 0.001
Alpha: 1000
Erro de treino obtido com estes hiperparâmetros: 18.459461584351413
Comparando os resultados, notamos que a ridge regression tradicional do item (ii) apresentou o melhor resultado, seguido pela regressão linear e pela ridge regression com kernel linear. Isso indica um comportamento linear do output em relação as features. Esse comportamento pode ser constatado ao plotarmos um gráfico de cada feature e o output correspondente. Além disso, o valor de α no caso (ii) é 0.01, o menor falo possível dentre o array de alphas testados. Isso indica que o melhor modelo encontrado é muito próximo da regressão linear.

8 Exercicio 2a

(i) Seja K_1 um kernel em $\mathcal{X} \times \mathcal{X}$. Queremos mostrar que a função

$$K(x, x') \;=\; c\,K_1(x, x') \;+\; d, \quad \text{com } c > 0 \text{ e } d > 0,$$

também é um kernel.

Por hipótese, como K_1 é kernel, existe um feature map

$$\phi_1 : \mathcal{X} \to \mathbb{R}^m \quad \text{tal que} \quad K_1(x, x') \;=\; \phi_1(x)^\top \phi_1(x').$$

Definamos então

$$\phi(x) \;=\; \begin{bmatrix} \sqrt{c}\,\phi_1(x) \\ \sqrt{d} \end{bmatrix}.$$

Note que, para quaisquer $x, x' \in \mathcal{X}$,

$$\phi(x)^\top \phi(x') \;=\; (\sqrt{c}\,\phi_1(x))^\top (\sqrt{c}\,\phi_1(x')) \;+\; \sqrt{d}\,\sqrt{d} \;=\; c\,\phi_1(x)^\top \phi_1(x') \;+\; d \;=\; c\,K_1(x, x') \;+\; d.$$

Portanto,

$$K(x, x') \;=\; \phi(x)^\top \phi(x'),$$

o que mostra que K é de fato um kernel.

(ii) Como K_1 e K_2 são kernels, existe ϕ_1 e ϕ_2 tais que:

$$K_1(x, x') = \phi_1(x)^T \phi_1(x') \quad \text{e} \quad K_2(x, x') = \phi_2(x)^T \phi_2(x')$$

Definindo ϕ como:

$$\begin{bmatrix} \phi_1(x) \\ \phi_2(x) \end{bmatrix}.$$

Obtemos:

$$\phi(x)^T \phi(x') = \phi_1(x)^T \phi_1(x') + \phi_2(x)^T \phi_2(x') = K_1(x, x') + K_2(x, x') = K(x, x')$$

Logo, K é um kernel.

(iii) Para mostrar que o produto de dois kernels K_1 e K_2 também é kernel, seja

$$K_1(x, x') = \phi_1(x)^\top \phi_1(x') \quad \text{e} \quad K_2(x, x') = \phi_2(x)^\top \phi_2(x'),$$

Definindo

$$\phi(x) \;=\; \phi_1(x) \;\times\; \phi_2(x),$$

temos

$$K(x, x') \;=\; K_1(x, x') \times K_2(x, x') \;=\; (\phi_1(x)^\top \phi_1(x')) \times (\phi_2(x)^\top \phi_2(x')) \;=\; (\phi_1(x) \times \phi_2(x))^\top (\phi_1(x') \times \phi_2(x')) \;=$$

$$= \phi(x)^\top \phi(x'),$$

mostrando que K é também um kernel.

(iv): Basta mostrar que, para qualquer conjunto finito $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, a matriz

$$\Big[K_\ell(x_i, x_j)\Big]_{i,j=1}^n$$

é positiva semidefinida (PSD) para todo ℓ, e que o limite de matrizes PSD continua sendo PSD. Seja

$$M_\ell := \big[K_\ell(x_i, x_j)\big]_{i,j=1}^n.$$

Como K_ℓ são kernels, cada M_ℓ é PSD. Supondo que $K_\ell(x, x') \to K(x, x')$ pontualmente, então

$$M := \lim_{\ell\to\infty} M_\ell = \big[\lim_{\ell\to\infty} K_\ell(x_i, x_j)\big]_{i,j=1}^n = \big[K(x_i, x_j)\big]_{i,j=1}^n.$$

A estabilidade do cone de matrizes PSD sob limites garante que M é PSD. Logo, o novo K define uma matriz PSD para qualquer escolha de pontos, satisfazendo a condição de kernel.

(v): Suponhamos que K_1 seja um kernel. Observe a expansão em série:

$$e^{K_1(x,x')} \;=\; \sum_{n=0}^{\infty} \frac{1}{n!} (K_1(x, x'))^n.$$

Para mostrar que cada termo $(K_1(x, x'))^n$ é kernel, aplicamos o item (iii): se $K_1(x, x')$ é kernel, então $K_1(x, x') \times K_1(x, x') = K_1(x, x')^2$ é kernel, e assim por indução obtemos $K_1(x, x')^n$ é kernel para todo n. Agora, pela propriedade (i) ou (ii), a soma finita de kernels é também kernel. Logo, cada soma parcial

$$K^{(m)}(x, x') \;=\; \sum_{n=0}^{m} \frac{1}{n!} \left(K_1(x, x')\right)^n$$

é kernel. Por fim, o item (iv) (limite de kernels) garante que

$$K(x, x') \;=\; \lim_{m\to\infty} K^{(m)}(x, x') \;=\; \lim_{m\to\infty} \sum_{n=0}^{m} \frac{1}{n!} \left(K_1(x, x')\right)^n$$

também é kernel. Conclui-se, portanto, que $K(x, x') = e^{K_1(x,x')}$ é kernel.

(vi): Suponha que K_1 seja um kernel e $f : \mathcal{X} \to \mathbb{R}$ seja uma função qualquer. Para um conjunto finito de pontos $\{x_1, \ldots, x_n\}$, a matriz de kernel correspondente a K é

$$M := \big[\, K(x_i, x_j)\big]_{i,j=1}^n = \big[\, f(x_i)\, K_1(x_i, x_j)\, f(x_j)\big]_{i,j=1}^n.$$

Note que

$$M = D\, M_1\, D,$$

onde $M_1 = [\, K_1(x_i, x_j)\,]_{i,j=1}^n$ (que é PSD, pois K_1 é kernel) e $D = \mathrm{diag}(f(x_1), \ldots, f(x_n))$. Como a multiplicação $D\, M_1\, D$ preserva positividade para matrizes diagonais D (e M_1 PSD), concluímos que M também é PSD. Logo $K(x, x')$ satisfaz a definição de kernel.

(vii): Note que

$$e^{-\gamma\|x-x'\|^2} \;=\; e^{-\gamma(\|x\|^2 - 2\,x^\top x' + \|x'\|^2)} \;=\; e^{-\gamma\|x\|^2}\; e^{\,2\gamma\, x^\top x'}\; e^{-\gamma\|x'\|^2}.$$

Aqui:

$$K_1(x, x') = 2\,\gamma\, x^\top x'$$

é kernel por (i) pois $x^T x'$ é kernel, assim $e^{K_1(x,x')}$ também é kernel (por (v)). Desse modo, ao multiplicar por $f(x)$ e $f(x')$ (onde $f(x) = e^{-\gamma\|x\|^2}$), ainda obtemos um kernel (por (vi)). Portanto,

$$K(x, x') = f(x)\,(e^{K_1(x,x')})\,f(x')$$

é kernel, concluindo-se que $e^{-\gamma\|x-x'\|^2}$ é um kernel válido.

9 Exercicio 2b

Seja $x_k \neq x$ o ponto mais próximos de x. Separando o somatório em k e dividindo todos os termos por $e^{-\gamma||x_k - x||_2^2}$:

$$f(x) = \lim_{\gamma \to \infty} \text{sign}\Big(\sum_{i \neq k} \alpha_i \, \tilde{y}_i \, e^{\gamma(\|x_k - x\|_2^2 - \|x_i - x\|_2^2)} \; + \; \alpha_k \, \tilde{y_k} \; \Big)$$

$$= \text{sign} \lim_{\gamma \to \infty} \Big(\sum_{i \neq k} \alpha_i \, \tilde{y}_i \, e^{\gamma(\|x_k - x\|_2^2 - \|x_i - x\|_2^2)} \; + \; \alpha_k \, \tilde{y_k} \; \Big)$$

Como $x_k \neq x$ é o ponto mais próximo de x, $\|x_k - x\|_2^2 - \|x_i - x\|_2^2$ é menor que 0, logo $\gamma \to \infty$ implica que $\sum_{i \neq k} \alpha_i \, y_i \, e^{\gamma(\|x_k - x\|^2 - \|x_i - x\|^2)} \; \to \; 0$. E portanto,

$$f(x) \; \to \; \text{sign}(\alpha_k \, \tilde{y_k}).$$

Caso $x_k = x$, aplicando o limite diretamente na expressão inicial, obtemos $f(x) \to \text{sign}(\alpha_k \, \tilde{y_k})$.

Além disso, caso hajam múltiplos pontos "mais próximos", todos equidistantes, basta realizar as mesmas operações acima dividindo todos os termos por essa distância miníma. Com isso $f(x)$ irá convergir para o *sign* da soma de $\alpha_k \tilde{y_k}$ para as amostras equidistantes.

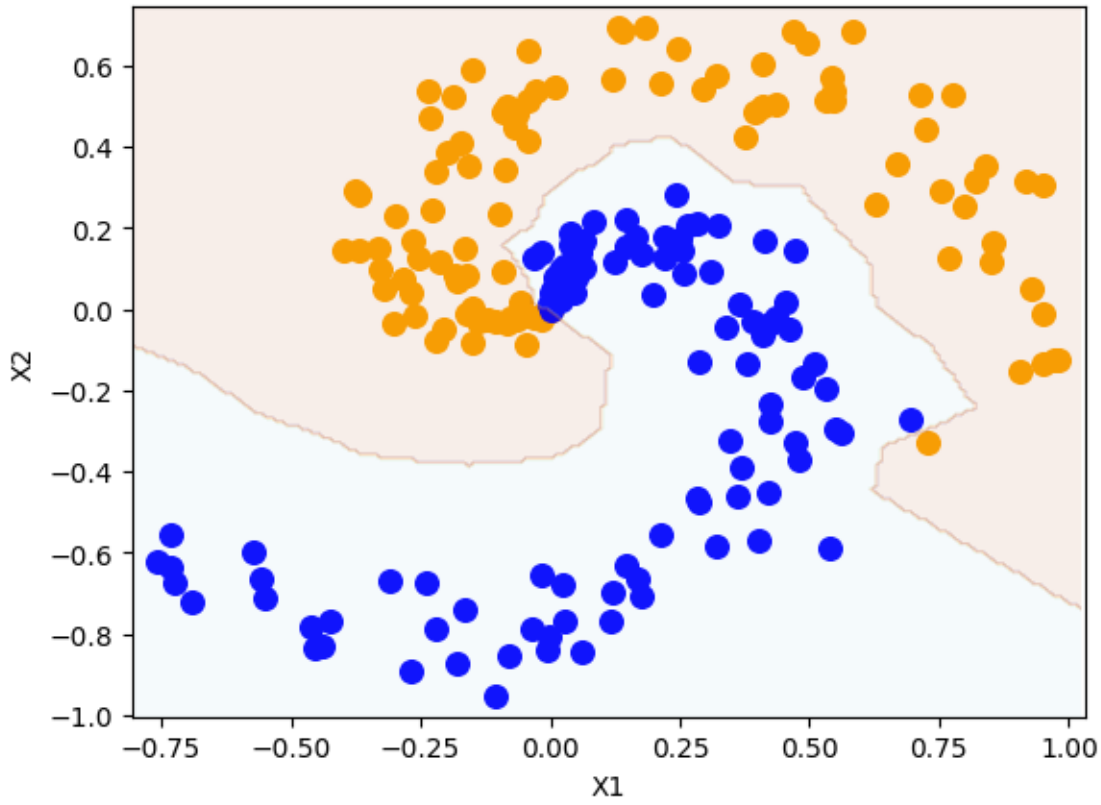
Todos os casos explorados acima geram uma mesma conclusão: o modelo fará suas previsões baseado apenas no vizinho mais próximo (ou nos vizinhos mais próximos, desde que sejam equidistantes).

Assim, como modelo leva em conta apenas o ponto mais próximo para classificar x, ele se comporta de maneira similar ao kNN no caso $k = 1$.

10 Exercicio 2c

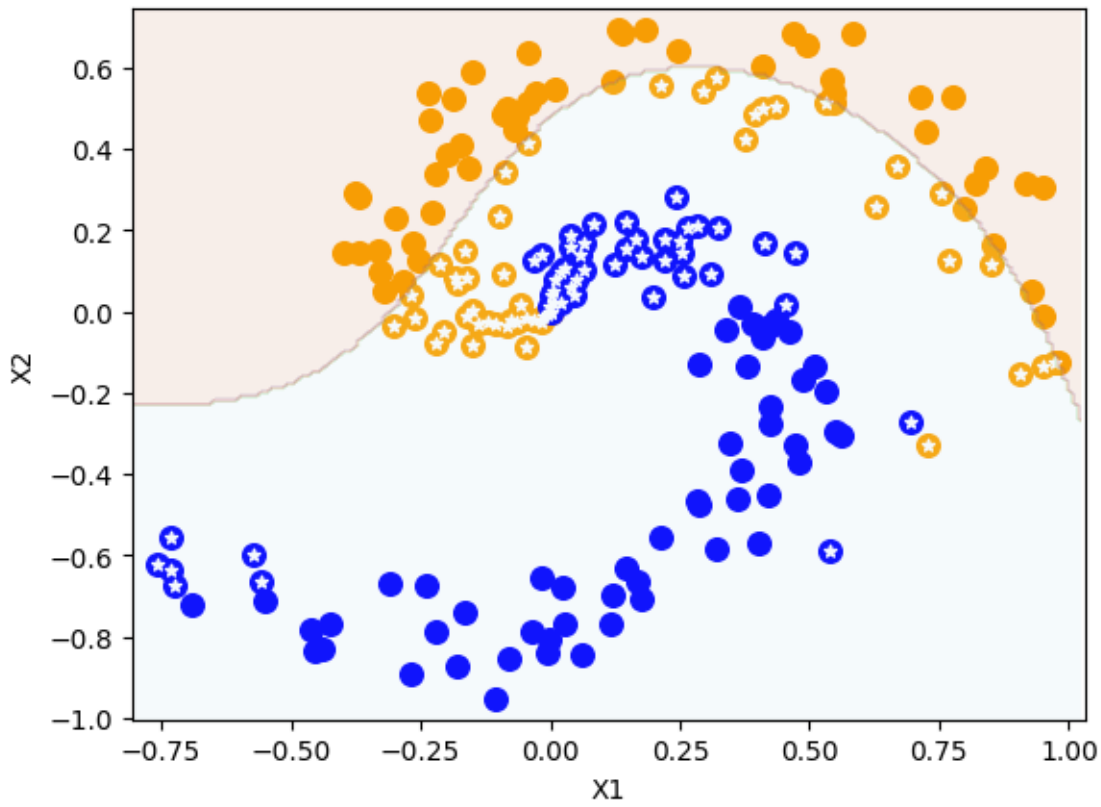
(ii) Ao executar o código abaixo obtemos a imagem anexada:

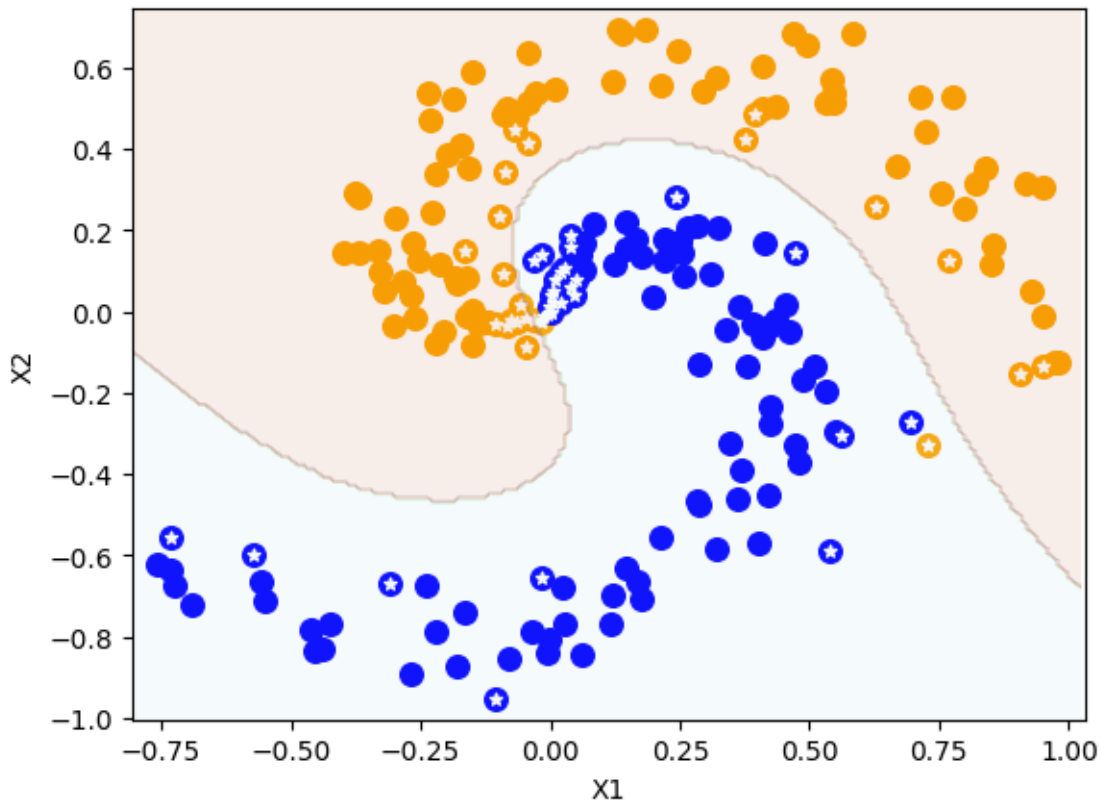
```
knn = KNN(n_neighbors=1)
knn.fit(X, y)
plot_decision_boundary(knn, X, y)
```

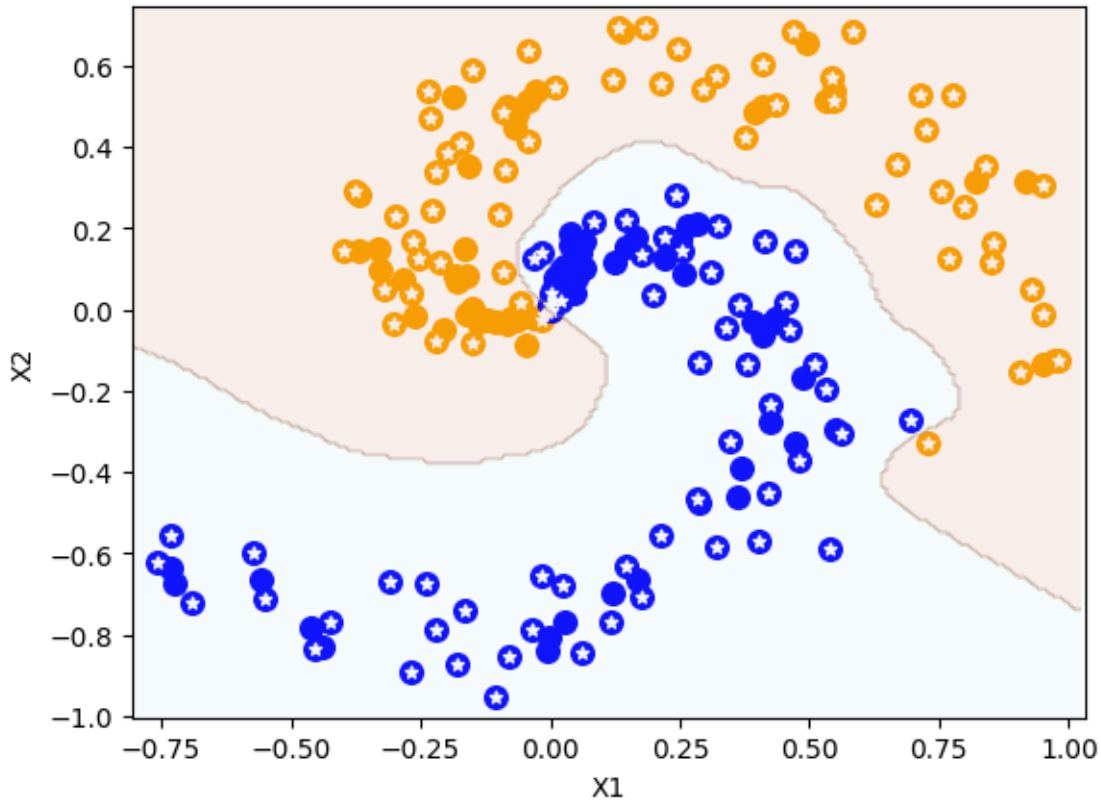


(iii) Ao executar o código abaixo, alternando entre os inputs comentados, obtemos os seguintes gráficos na mesma sequência apresentada no código:

```
g_list = [1, 5, 100]

g = g_list[0]
#g = g_list[1]
#g = g_list[2]
svm = SVC(C=1, kernel="rbf", gamma=g, random_state=42)

svm.fit(X, y)

plot_decision_boundary(svm, X, y)
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
X2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X1


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
X2
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X1



Como podemos notar, a relação entre o γ do SVM e o k do kNN indica que essas variáveis, em relação aos seus respectivos modelos, são inversamente proporcionais. Enquanto o aumento do gamma no SVM garante maior flexibilidade ao modelo introduzindo viés, o aumento do k no kNN reduz essa flexibilidade diminuindo o viés. Ao avaliarmos o kNN com $k = 1$ (ou seja, o kNN mais flexível possível de se obter), verificamos que sua fronteira de decisão é extremamente próxima a do SVM com $\gamma = 100$. Tais constatações também confirmam o resultado encontrado no item anterior, onde fazendo $\gamma- > \infty$, geramos um modelo extremamente flexível que só prevê a classe de um ponto a partir da amostra de treino mais próxima a esse ponto. Além disso, quando fizemos $\gamma- > 0$, geramos um modelo pouco flexível que basicamente classificava o ponto de acordo com a classificação da maioria das amostras.