Machine Learning: Lista Adicional de Exercícios

Paulo Orenstein

Monitores: Antônio Catão, Otávio Moreira, Melvin Poveda

Verão, 2025

Exercício 1 (Kernels em todo canto)**.** O truque do kernel não é algo particular a SVMs. Nesse exercício, vamos investigar como aplicá-lo a ridge regression.

(a) Supondo uma amostra $\{(x_i, y_i)\}_{i=1}^n$, onde $x_i \in \mathbb{R}^p$ e $y_i \in \mathbb{R}$, ridge regression resolve

$$\hat{\beta}_0, \hat{\beta} = \underset{\tilde{\beta}_0, \tilde{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \tilde{\beta}_0 - \sum_{j=1}^{p} x_{ij}\tilde{\beta}_j \right)^2 + \lambda \sum_{j=1}^{p} \tilde{\beta}_j^2 = \underset{\tilde{\beta}_0, \tilde{\beta}}{\operatorname{argmin}} \|y - \tilde{\beta}_0 \mathbf{1} - X\tilde{\beta}\|_2^2 + \lambda \|\tilde{\beta}\|_2^2.$$

Explique por que, centralizando os dados tais que $y_i \mapsto y_i - \overline{y}$ e $x_{ij} \mapsto x_{ij} - \overline{x}_j$, podemos tomar $\hat{\beta}_0 = 0$. (Daqui em diante, vamos assumir que os dados estão centrados e ignorar $\hat{\beta}_0$.)

(b) Encontre o estimador $\hat{\beta}$ em forma fechada. Argumente que, quando $\lambda \to 0$, ele se torna o estimador de regressão linear, $\hat{\beta}_{\text{lin}} = (X^T X)^{-1} X^T y$.

(c) Mostre que $\hat{\beta}$ pode ser escrito como $\hat{\beta} = \sum_{i=1}^n \hat{\alpha}_i x_i$, onde $\hat{\alpha}_i \in \mathbb{R}$, e identifique o vetor $\hat{\alpha} \in \mathbb{R}^n$ em função de X, y e λ. *Dica:* reescreva $\hat{\beta}$ usando a fórmula de Woodbury para inversão de matrizes: $(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$; use $A = \lambda I$, $B^T = D = X$ e $C = I$. Você deve encontrar $\hat{\beta} = X^T \alpha$ para algum α.

(d) Dado um novo ponto x_*, escreva a fórmula para a previsão $\hat{y}_*$ associada usando a fórmula encontrada no item (c).

(e) A partir do item anterior, explique como é possível kernelizar as previsões de ridge regression. Inclua as fórmulas necessárias para fazer a previsão de um y_* associado a um novo ponto x_* usando um kernel arbitrário $K(\cdot, \cdot)$.

(f) Note que a fórmula para o coeficiente estimado $\hat{w}$ num SVM também tem o formato apontado em (c). Por outro lado, SVM costuma ser computacionalmente mais vantajoso. Explique o porquê comparando o $\hat{\alpha}$ encontrado nos dois casos em termos de esparsidade.

(g) Agora, vamos ver em que medida kernels ajudam uma regressão ridge na prática. Para isso, vamos novamente usar o dataset `bodyfat.csv`. Abra os dados e divida-os em treino e teste de acordo com o seguinte código.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
```

```
from sklearn import preprocessing
from tqdm import tqdm

number_of_folds = 10

data = pd.read_csv("./bodyfat.csv")
X = data.drop(columns=["BodyFat","Density"])
y = data["BodyFat"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=0)
```

(i) Qual é o erro médio quadrático de uma regressão linear nos dados de teste?

(ii) Qual é o erro médio quadrático de uma regressão ridge nos dados de teste? Use o seguinte código para gerar sua resposta:

```
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(...)  # treine seu modelo nos folds de treino
        y_hat = ...  # faça suas previsões no fold de validação
        cv_MSE.append(...)  # salve o MSE encontrado no fold de validação

    ridge_cv_mse.append(np.mean(cv_MSE))

optimal_alpha = ...  # encontre o melhor valor de alpha

ridge_pipeline = make_pipeline(StandardScaler(), Ridge(alpha=optimal_alpha))
ridge_pipeline[1].fit()  # treine seu modelo em todo o conjunto de treino

y_hat_ridge = ... # faça as previsões em todo o conjunto de teste
ridge_test_mse = ... # salve o MSE encontrado no conjunto de teste
print(ridge_test_mse)
```

(iii) Finalmente, vamos usar kernel ridge regression para tentar melhorar esse resultado. Para isso, vamos olhar para os seguintes hiperparâmetros:

```
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]
```

Adapte o código do item anterior, mantendo os mesmos folds, para escolher o conjunto de hiperparâmetros com menor erro de validação cruzada. Quais são os hiperparâmetros escolhidos?

(iv) Encontre o erro de teste do modelo de kernel ridge com o hiperparâmetro encontrado no item anterior. Compare a performance de regressão linear, ridge e kernel ridge.

Exercício 2 (Kernel radial)**.** Um dos kernels mais importantes usados em SVM é o *radial basis function kernel* (RBF). Ele é definido como:

$$K(x, x') = e^{-\gamma\|x-x'\|_2^2}, \gamma > 0.$$

Neste exercício, estamos interessados em mostrar que essa função define um kernel e entender melhor seu comportamento.

(a) Um kernel é uma função simétrica $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, onde $\mathcal{X} = \mathbb{R}^p$, tal que existe um feature map $\Phi : \mathcal{X} \to \mathbb{R}^k$ (onde $k \in \mathbb{N} \cup \{\infty\}$) com:

$$K(x, x') = \Phi(x)^\intercal \Phi(x'). \tag{1}$$

Equivalentemente, todo kernel quando aplicado a um conjunto finito de pontos pode ser representado como uma matriz simétrica positiva semi-definida, isto é, para quaisquer pontos $x_1, \ldots, x_n \in \mathcal{X}$ e $c_1, \ldots, c_n \in \mathbb{R}$, a matriz $n \times n$ de entradas $K(x_i, x_j)$ satisfaz:

$$\sum_{j=1}^{n}\sum_{i=1}^{n} c_i K(x_i, x_j) c_j \geq 0. \tag{2}$$

Vamos provar que o RBF é um kernel em etapas. Se K_1 e K_2 são kernels, prove que também são kernels:

(i) $K(x, x') = cK_1(x, x') + d,\ c > 0,\ d \geq 0.$
(ii) $K(x, x') = K_1(x, x') + K_2(x, x').$
(iii) $K(x, x') = K_1(x, x') \times K_2(x, x').$
(iv) $K(x, x') = \lim_{l\to\infty} K_l(x, x')$, onde K_l é um kernel para todo $l \in \mathbb{N}$.
(v) $K(x, x') = e^{K_1(x,x')}.$
(vi) $K(x, x') = f(x)K_1(x, x')f(x')$ para $f : \mathcal{X} \to \mathbb{R}$.
(vii) $K(x, x') = e^{-\gamma\|x-x'\|_2^2}, \gamma > 0.$

(b) Considere um SVM sem intercepto, isto é, com $\beta_0 = 0$. O que acontece com esse SVM usando o kernel RBF quando $\gamma \to \infty$? Isto é, qual é a forma de:

$$f(x) = \lim_{\gamma\to\infty} \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i \tilde{y}_i e^{-\gamma\|x_i-x\|_2^2}\right)?$$

Relacione a forma obtida acima com o classificador kNN.

(c) Vamos agora verificar computacionalmente o item (b).

(i) Rode os seguintes comandos para carregar os pacotes necessários, definir algumas funções de utilidade e gerar um conjunto de dados artificial.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier as KNN

def apply_svm_no_intercept(svm, X):
    dec_func = svm.decision_function(X) - svm.intercept_
```

```
    return np.sign(dec_func)

# Função auxiliar para plotar região de decisão
def plot_decision_boundary(clf, X, y, h=0.01, pad=0.05):
    x_min, x_max = X[:, 0].min() - pad, X[:, 0].max() + pad
    y_min, y_max = X[:, 1].min() - pad, X[:, 1].max() + pad
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    plt.scatter(X[y==-1,0], X[y==-1,1], s=70, c="blue", label=-1)
    plt.scatter(X[y==1,0], X[y==1,1], s=70, c="orange", label=1)
    if 0 in y:
        plt.scatter(X[y==0,0], X[y==0,1], s=70, c="blue", label=-1)

    if type(clf) == SVC:
        Z = apply_svm_no_intercept(clf,np.c_[xx.ravel(), yy.ravel()])
        sv = clf.support_vectors_
        plt.scatter(sv[:,0], sv[:,1], c='w', marker='*', s=21, linewidths=1)
        print('Numero de vetores de suporte: ', clf.support_.size)
    else:
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.10)
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()

n = 100  # Número de pontos por classe
d = 2    # Dimensão
k = 2    # Número de classes

X = np.zeros((n*k,d))
y = np.zeros(n*k)

r = np.linspace(0.0,1,n) # Raio

np.random.seed(42)

for j in range(k):
    ind = range(n*j,n*(j+1))
    t = np.linspace(j*4,(j+1)*4,n) + np.random.randn(n)*0.4 # theta
    X[ind] = np.c_[r*np.sin(t), r*np.cos(t)]
    y[ind] = j
```

(ii) Treine um kNN com $k = 1$ em todo o conjunto de dados e faça o gráfico da região de decisão usando:

```
plot_decision_boundary(knn, X, y)
```

(iii) Ao invés do kNN, considere rodar um SVM, definido por:

```
svm = SVC(C=1, kernel='rbf', gamma=g, random_state = 42)
```

Repita o procedimento para os valores g $\in \{1,\ 5,\ 100\}$. Compare os gráficos das regiões de decisão associadas a cada valor de g e sua relação com o gráfico obtido no item acima para o kNN.