Relatório — Teoria do Aprendizado Estatístico — Criando a própria versão do KNN

Thiago Guilherme Gonçalves[1]

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19081 – 81531-980 – Curitiba – PR – Brasil

thiagoguilherme@ufpr.br

1. Introdução

No contexto deste trabalho, vamos implementar uma versão própria de regressão do algoritmo de aprendizado de máquina K-Nearest Neighbors (KNN) utilizando a *Distância de Manhattan* como função de perda. Vamos nos aprofundar nos detalhes a respeito do KNN e da *Distância de Manhattan* nas subseções a seguir.

1.1. O Método dos K-vizinhos mais próximos (K-Nearest Neighbors)

K-Nearest Neighbors (KNN) é um algoritmo de aprendizado de máquina não paramétrico [Fix and Hodges 1951] que pode ser usado tanto para tarefas de classificação quanto de regressão. No contexto da regressão, que é o que trabalharemos neste relatório, o KNN é frequentemente referido como ”Regressão dos K-Vizinhos Mais Próximos”ou ”KNN-Regressor”. É um algoritmo simples e intuitivo que faz previsões encontrando os K pontos de dados mais próximos de uma entrada fornecida e calculando a média de seus valores alvo (para regressão numérica) ou selecionando a classe majoritária (para classificação) [Cover and Hart 1967]. Abaixo segue uma visão geral de como funciona o KNN-Regressor:

1. **Coleta de Dados:** Inicia-se com um conjunto de dados que inclui tanto as características de entrada quanto os valores alvo. Em tarefas de regressão, os valores alvo são contínuos e representam a saída que deseja-se prever.
2. **Escolhendo o Número de Vizinhos (K):** Deve-se fazer a escolha do número de vizinhos mais próximos, K, que serão usados para fazer previsões. K é um hiperparâmetro que pode ser ajustado com base nas características dos dados. Um K pequeno pode levar a previsões ruidosas, enquanto um K grande pode levar a previsões excessivamente suavizadas.
3. **Métrica de Distância:** O KNN depende de uma métrica de distância para medir a similaridade entre os pontos. Diferentes métricas de distância podem ser usadas dependendo da natureza dos dados.
4. **Previsão:** Quando deseja-se fazer uma previsão para um novo ponto de dados de entrada, o KNN calcula a distância entre este ponto e todos os outros pontos de dados no conjunto de dados. Ele então seleciona os K pontos de dados com as menores distâncias.
5. **Previsão de Regressão:** Para regressão, o valor previsto para o novo ponto de dados é a média dos valores alvo dos K vizinhos mais próximos. O tradicional é uma simples média aritmética.

1.2. Distância de Manhattan

A *Distância de Manhattan* é uma métrica na qual a distância entre dois pontos (no plano cartesiano) é a soma das diferenças absolutas de suas coordenadas cartesianas. De forma simples, é a soma total das diferenças entre as coordenadas x e y.

A *Distância de Manhattan* é também conhecida como *Comprimento de Manhattan*, *Distância Retangular*, *Distância L1* ou *Norma L1*, *Distância L1 de Minkowski*, *Métrica de Táxi* [Thompson 2011], e outras. Suas aplicações são diversas, e são usadas, por exemplo, na regressão linear, para estimar os parâmetros da reta que se ajusta a um conjunto dado de pontos.

A fórmula da *Distância de Manhattan* para o caso n-dimensional [Black 2006] entre duas coordenadas $x = (x_1, x_2, ..., x_n)$ e $x' = (x'_1, x'_2, ..., x'_n)$ pode ser expressa por:

$$d(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} |x_i - x'_i| \tag{1}$$

2. Metodologia

Decidiu-se realizar uma tratativa mais matemática do que no contexto da aprendizagem de máquina em si, para a visualização dos pontos selecionados pelo algoritmo sobre funções bem comportadas, onde tomamos a função a seguir generalizada para n dimensões:

$$y = f(x_1, ..., x_n) = \sin\left(\sqrt{\sum_{i=1}^{n} x_i^2}\right) + \mathcal{E} \qquad \forall x_1 \times ... \times x_n \tag{2}$$

onde $xi \in [-5, 5]$ com N divisões, e $\mathcal{E}$ é um valor somado a cada ponto do tensor de saída dado através da amostra aleatória de uma distribuição normal com $\mu = 0$ e $\sigma = 0.15$.

A implementação do algoritmo foi feita inteiramente na linguagem *Python*, utilizando somente a biblioteca *numpy* [Harris et al. 2020] para ganho de performance computacional, eficiência na manipulação e operação de vetores, trabalhar com funções otimizadas e demais razões.

O algoritmo KNN foi implementado de forma generalizada para n-dimensões de entrada, onde produziu-se algumas visualizações para obter uma validação visual do comportamento do algoritmo e enriquecer o desenvolvimento o trabalho. As visualizações foram geradas utilizando a biblioteca *plotly*, também em *Python*, com as saídas do algoritmo. O código que contém a implementação realizada segue no Listing 1 logo a seguir.

```
import numpy as np
import plotly.graph_objects as go

def create_meshgrid(*xi):
    meshgrid_x = np.meshgrid(*xi)
    y = np.sin(np.sqrt(np.sum(x_**2 for x_ in meshgrid_x))) + \
        np.random.normal(loc=0, scale=0.15, size=N)

    return meshgrid_x, y

def knn(*xi, x_new, K):
    meshgrid_x, y = create_meshgrid(*xi)
    meshgrid_x_new = np.meshgrid(*x_new)

    # Calculate the Manhtattan Distance for the whole input grid \
        to the 'x_new' point
    D = np.sum(np.abs( np.array(meshgrid_x) - \
        np.array(meshgrid_x_new) ), axis=0)

    # Flatten the distance grid and get the indexes of the K \
        points closest to the 'x_new' point
    closest_indexes = np.argsort(D.flatten())[:K]

    # Collect the K values for xi,...,xn and y values
    xi_closest = [xi.flatten()[closest_indexes] for xi in \
        meshgrid_x]
    y_closest = y.flatten()[closest_indexes]

    # Take the average value from the K 'y' values
    y_fit = np.array(np.mean(y_closest))

    # Some return variables here are for plotting purposes
    return xi_closest, y_closest, y_fit, meshgrid_x, y, \
        meshgrid_x_new

# Setting K (number of neighbors) for the KNN method
K = 5
# Number of splits in the input data
N = 100
# x1,x2,...,xn variables
xi = [np.linspace(-5, 5, N) for _ in range(2)]

# New data point --> x_new = [x1,x2,...,xn]
x_new = np.array([2,-2,]) #

xi_closest, y_closest, y_fit, meshgrid_x, y, meshgrid_x_new = \
    knn(*xi, x_new=x_new, K=K)
```

Listing 1. Implementação em Python do algoritmo KNN

3. Análise de Resultados e Comentários Finais

Essa versão do algoritmo KNN foi programada de modo a calcular a Distância de Manhattan D entre o ponto novo $x\_new = [x'_1, x'_2, ..., x'_n]$ e todos os pontos do tensor de coordenadas formado pelas entradas $x_1, x_2, ..., x_n$ simultaneamente em memória, o que pode ser problemático para um alto número de observações (N, no contexto do nosso algoritmo) ou dimensões x_i, pois embora o cálculo seja simples, possui um custo de N^n para o armazenamento do tensor em memória. Um possível próximo passo seria a otimização desse algoritmo, com ênfase no cálculo de D por segmentos/blocos do tensor de coordenadas, evitando sobrecarga na memória.

Foram geradas algumas visualizações, particulares para o caso $y = f(x_1, x_2)$ para o ponto $x\_new = [2, -2]$, com os mesmos exatos parâmetros do Listing 1, em que podemos observar os K pontos em azul escolhidos pelos algoritmo, e o ponto previsto em vermelho.

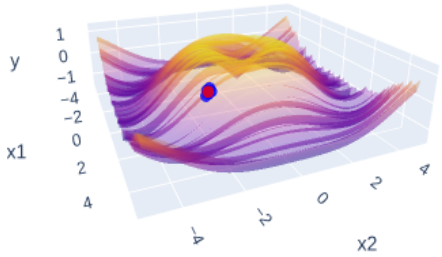

Figura 1. Visualização de y=f(x1,x2) com os K=5 pontos mais próximos (em azul) selecionados pelo algoritmo, junto com a previsão do ponto novo (em vermelho).

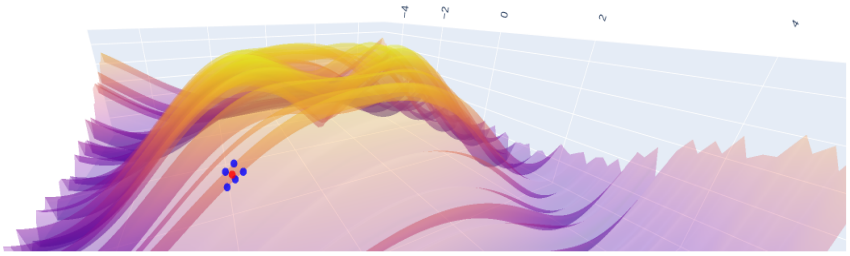

Figura 2. Visualização da Figura 1 ampliada. Percebe-se coerência espacial entre os pontos escolhidos e o ponto previsto.

Referências

Black, P. E. (2006). Manhattan distance. *Dictionary of algorithms and data structures*, 18:2012.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Fix, E. and Hodges, J. L. (1951). Discriminatory analysis, nonparametric discrimination.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Thompson, K. P. (2011). The nature of length, area, and volume in taxicab geometry. *International Electronic Journal of Geometry*, 4(2):193–207.