# Aplicação do Algoritmo k-NN

Pablo Luiz Leon

*Santo André – SP, UFABC*

Abstract

This work aims to show the implementation of the kNN algorithm (k Nearest Neighbors), to predict a value, based on a set of values (Database).

The objective of the work is to implement the kNN algorithm in two ways, sequentially and in parallel. To show the benefits of parallel processing. artifacts.

*Keywords:* knn, spark, python, big data, parallel programming

## 1. Algoritmo K-NN

O algoritmo k-NN é um algoritmo de classificação o K vizinhos mais próximos (do inglês: K nearest neighboors – KNN). O KNN foi proposto por Fukunaga e Narendra em 1975 [1]. É um dos classificadores mais simples de ser implementado, de fácil compreensão e ainda hoje pode obter bons resultados dependendo de sua aplicação.

A ideia principal do KNN é determinar o rótulo de classificação de uma amostra baseado nas amostras vizinhas advindas de um conjunto de treinamento indicado.

Para tornar o entendimento mais fácil vamos aplicar um exemplo para explicar o seu funcionamento do algoritmo como o da Figura 1, na qual temos um problema de classificação com dois rótulos de classe e com k = 7. No exemplo, são aferidas as distâncias de uma nova amostra, representada por uma estrela, às demais amostras de treinamento, representadas pelas bolinhas azuis e amarelas. A variável k representa a quantidade de vizinhos mais próximos que serão utilizados para averiguar de qual classe a nova amostra pertence. Com isso, das sete amostras de treinamento mais próximas da nova amostra, 4 são do rótulo A e 3 do rótulo B. Portanto, como existem mais vizinhos do rótulo A, a nova amostra receberá o mesmo rótulo deles, ou seja, A.

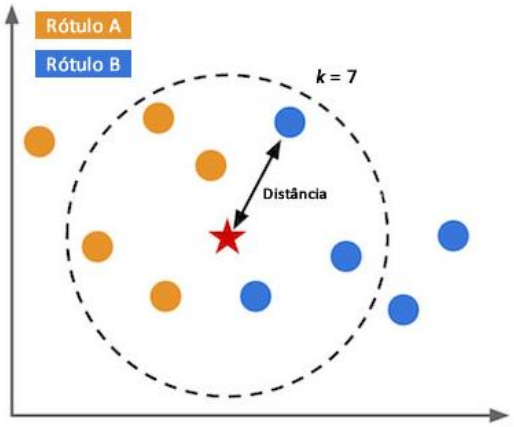


Figura 1: Explicação gráfica do Algoritmo k-NN

Dois pontos chaves que devem ser determinados para aplicação do KNN são: a métrica de distância e o valor de k. Para métrica de distância a mais utilizada é a distância Euclidiana, descrita por:

$$D = \sqrt{(p_1 - q_1)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad (1)$$

onde $P = (p_1, \cdots, p_n)$ e $Q = (q_1, \cdots, q_n)$ são dois pontos $n$-dimensionais. No exemplo que citei da figura, essa distância seria calculada entre as bolinhas (azuis e laranjas) e a estrela (a nova entrada). Como o exemplo usa dois parâmetros, cada ponto tem dois valores $x_1$ e $x_2$. Portanto para esse cenário do exemplo seria $(x_1\text{-}y_1)^2 + (x_2\text{-}y_2)^2$, caso o seu problema tenha mais de dois parâmetros a abordagem é a mesma para o cálculo euclidiano das distâncias.

Em relação ao valor k, não existe um valor único para a constante, a mesma varia de acordo com a base de dados. É recomendável sempre utilizar valores ímpares/primos, mas o valor ótimo varia de base para base, onde o sugerido para esse algoritmo seria calcular o valor de k baseado no logaritmo na base dois do total de amostras da base de dados em questão.

Em alto nível a grande vantagem do método do kNN é a sua abordagem simples para ser compreendida e implementada, todavia, efetuar a distâncias dos pontos gerar um custo computacional para calcular as distâncias de forma bem relevante no tempo

total da execução do algoritmo, caso tenha uma quantidade grande de amostras de treino essa atividade custosa pode gerar um tempo relativamente grande para a execução do algoritmo. Além disso, o método é sensível à escolha do k (quantidade de vizinhos observados).

```
inicialização:
    Preparar conjunto de dados de entrada e saída
    Informar o valor de k;
para cada nova amostra faça
    Calcular distância para todas as amostras
    Determinar o conjunto das k's distâncias mais próximas
    O rótulo com mais representantes no conjunto dos k's
    vizinhos será o escolhido
fim para
retornar: conjunto de rótulos de classificação
```

Figura 2: Passos do Algoritmo k-NN

## 2. Aplicação do para o trabalho proposto

Para um conjunto de dados onde temos os dados de cotação de ações da bolsa americana em um arquivo com o seguinte layout (x, y) onde:

x → data do valor da ação;

y → valor da ação.

Crie uma massa de dados onde dado um valor N determinado pelo usuário seja construído a massa de dados com o seguinte layout (l, v) onde:

l → lista de valores do ativo, onde total da lista é igual ao N especificado;

v →próximo valor da lista, na sequência de valores de ativo.

O Algoritmo foi dividido em algumas partes, funções que fazem parte especifica do contexto todo do problema.

A Função criarDados(RDD_Base, tamanho_list), que efetua o preparo dos dados. A ideia é ter um arquivo que indica os valores de cada ação. Essa função recebe o arquivo e transforma os dados para que seja criado o RDD com os dados formatados para o cálculo do k-NN.

A Função DistanciaEuclediana(RDDDados, DadosBase), que efetua o cálculo euclidiano dos valores indicado, onde recebe um RDD com os dados de treino, um valor de base para predição.

A Função kNN(k, RDDTreino, Dados), que efetua o cálculo do k-NN, aonde passando a base de dados, o valor de k e a base de treino ele efetua o cálculo e retorna à predição do valor em questão para aquela sequência de dados.

Para esse trabalho temos a problemática de predizer o valor do ativo dado a lista de dados com os valores do ativo indicado em uma lista. Portanto, para efetuar essa predição iremos pegar o k mais próximo do valor da lista de valores de ação em questão analisado, com isso o parâmetro k da implementação fica fixo em k=1.

Durante a execução do Algoritmo foram feitos testes para medir o tempo de execução, levando em consideração 1: modelo sequencial e 2: modelo paralelizado.

Para o modelo 1: sequencial, a carga no Data set de dados (RDD) inicialmente é feita de forma normal sem usar paralelismo, conforme o código abaixo:

```
arquivo = os.path.join('Data', 'C:/UFABC/DADOS/ETFs/Dados_Teste.txt')

# lê o arquivo com os dados e carrega em um RDD
DadosRDD1 = (sc.textFile(arquivo, 8).collect())
```

Figura 3: Código de Carregamento do RDD para execução em modo Sequencial

Com esse modelo adotado os dados de teste, apresentou um tempo total de 2 minutos e 54 segundos para carregar os dados e efetuar o processamento das transformações.

Já no segundo modelo 2: modelo paralelizado, foi configurado na carga do Data set de dados (RDD) o inicio do paralelismo de processamento, conforme o código abaixo:

```
arquivo = os.path.join('Data', 'C:/UFABC/DADOS/ETFs/Dados_Teste.txt')

# lê o arquivo com os dados e carrega em um RDD
DadosRDD1 = (sc.textFile(arquivo, 8).collect())

DadosRDD = sc.parallelize(DadosRDD1,6)
```

Figura 4: Código de Carregamento do RDD para execução em modo Paralelo

Com esse modelo os testes apresentaram uma melhora significativa no tempo aonde levou certa de 1 Minuto e 52 segundos para carregar os dados e efetuar o processamento das transformações. Gerando no total uma eficiência de 36% do tempo de processamento.

## Referências

[1] Fukunaga, K.; Narendra, P. M. A branch and bound algorithm for computing k-nearest neighbors. IEEE Transactions on Computers, v. 100, n. 7, p. 750–753, 1975.