Implementação de algoritmo KNN Concorrente em linguagem C usando biblioteca Pthreads

Igor Bispo de M. C. Correia

November 22, 2020

Abstract

Este trabalho se insere na disiplina Programação Concorrente da Universidade de Brasília e consiste na aplicação do algoritmo de classificação *k-Nearest-Neighbors* (KNN) [3] em linguagem C usando programação concorrente [1] a fim de acelerar o tempo de execução. Para avaliar o algoritmo, foi usada a base de dados do MNIST [4]. A estratégia de paralelização escolhida acelerou o tempo de execução em até 3.2x se comparada com a execução *single-threaded*. O fator de aumento de performance tende a ser maior à medida que a CPU é mais potente e capaz de executar mais threads simultâneas.

1 Introdução

O algoritmo KNN trata-se de um algoritmo de aprendizagem de máquina [3] utilizado como baseline para comparações em problemas de classificação por se tratar de um algoritmo de implementação relativamente simples e performance razoável em muitas aplicações.

Contudo, uma das principais desvantagens do KNN é o tempo de execução alto e crescente com relação ao número de amostras de treino. O algoritmo KNN pertence a $\mathcal{O}(n \cdot m \cdot k)$ em que n é o numero de amostras de treino, m é o número de amostras de teste e k é o número de *features* de cada amostra. Em virtude disso, o KNN não costuma ser executado em base de dados muito grandes ou com muitas *features*.

A fim de melhorar o tempo de execução, pode ser implementada uma estratégia de paralelismo em algum estágio do algoritmo. Nas seções seguintes deste documento será descrito o funcionamento do KNN e a estratégia de paralelismo escolhida para otimização.

2 Algoritmo KNN

O KNN é um algoritmo de aprendizagem de máquina não-paramétrico supervisionado [3] muito utilizado em problemas de classificação. A ideia por trás deste algoritmo consiste em supôr que dados próximos espacialmente, seguindo alguma métrica de distância pré-determinada, têm altas chances de pertencerem à mesma classe.

Uma implementação típica do KNN, com K como um hiperparâmetro, segue a seguinte estrutura:

1. Seja y_{pred} o vetor que conterá as predições do KNN para a base de dados de teste.
2. Para cada amostra X pertencente ao conjunto de dados de teste:

(a) Calcule a distância L2 [1] entre X e cada uma das amostras do conjunto de treinamento. Insira os resultados no vetor X_{dist}.
(b) Ordene as K primeiras posições do vetor X_{dist}.
(c) Faça um histograma H com os rótulos das K amostras de treinamento mais próximas de X, usando como referência X_{dist}.
(d) Seja y o rótulo de H que tem a maior frequência.
(e) Insira y em y_{pred}.

Como pode ser observado, a parte que mais toma tempo de execução do algoritmo está no cálculo das distâncias entre X e cada uma das amostras do conjunto de treinamento. A linha que calcula a norma L2 realizará na $\mathcal{O}(n \cdot k)$ operações para cada amostra na base de dados de teste, em que n é o # amostras de treinamento e k é o tamanho da entrada X.

Uma outra região bastante crítica com relação tempo de execução está na linha que ordena o vetor X_{dist}. Para cada amostra, será executada na $\mathcal{O}(n \cdot K)$ operações em que n é o # amostras de treinamento e K é o hiperparâmetro que representa a quantidade de vizinhos que serão analisados.

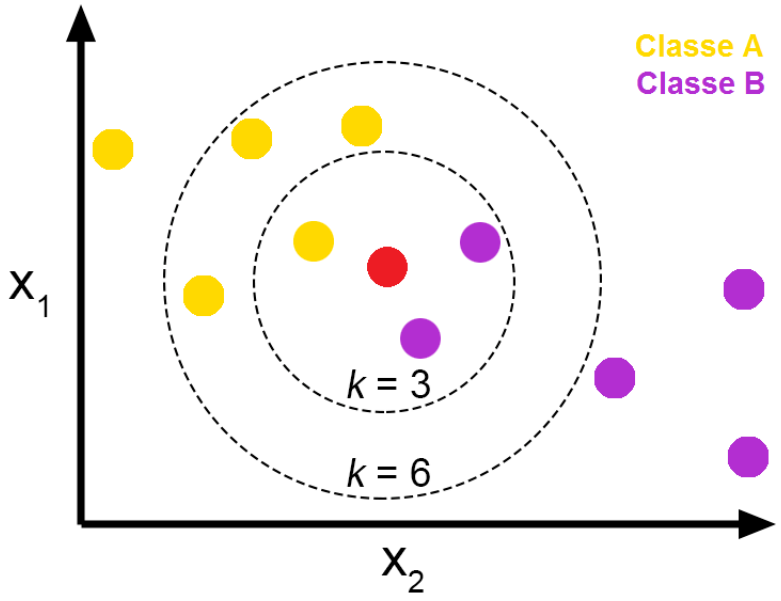

Figure 1: KNN aplicado para determinar o rótulo do ponto vermelho. Neste caso, com K = 6, o ponto seria rótulado como pertencente à classe A

2.1 Estratégias de Concorrência

Conforme citado na seção anterior, o cálculo das distâncias entre as amostras é a seção mais custosa do algoritmo quanto ao tempo de execução, em virtude disso, a aplicação de paralelismo [1] no cálculo das distâncias tipicamente trás um aumento significativo de performance do KNN.

Além disso, é desejável uma estretégia de paralelismo sobre a ordenação do vetor de distâncias. Contudo, não é possível aplicar uma simples estratégia de divisão de fronteiras sobre o vetor de distâncias visto que a ordenação de subpartições do vetor não necessariamente gera um vetor ordenado, sendo necessário a execução de um *merge* após a ordenação parcial dos sub-vetores.

[1] A distância L2 entre p e q é dada por $d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$

Por fim, a execução paralela do KNN sobre o vetor de amostras de teste é bastante interessante, visto que a determinação do rótulo de cada uma das amostras de teste é totalmente independente entre si.

3 KNN Concorrente

Com base no que foi exposto, foi construída neste projeto uma estratégia de concorrência sobre o KNN original.

O algoritmo implementado utiliza dois níveis de paralelismo, sendo o primeiro dentro da execução do KNN sobre a predição de cada amostra, e o segundo sobre o cálculo em toda a base de dados de teste.

O algoritmo implementado é decomposto em 4 funções escritas em linguagem C, a saber:

- thread_knn_predict, a qual roda o algoritmo típico sequencial do KNN sobre um subconjunto da base de dados de treinamento e classifica uma amostra;
- paralel_knn_predict, a qual executa a N instâncias da função thread_knn_predict, cada uma com um subconjunto da base de dados de treinamento.
- paralel_knn, a qual executa paralel_knn_predict sobre um subconjunto da base de dados de teste;
- knn_classifier, função que deve ser chamada pelo usuário, encapsula toda a execução do algoritmo KNN Concorrente.

Uma descrição mais aprofundada das estratégias de paralelismo implementadas será exposta nas seções a seguir.

3.1 Paralelismo Sobre Cada Amostra

Para realizar a predição de uma amostra, a função *paralel_knn_predict* dividirá a base de treinamento em N partes de mesmo tamanho, disjuntas, em que N é o número de threads que serão usadas na execução do algoritmo.

Em seguida, serão executadas paralelamente N instâncias do classificador KNN (representadas pela função *thread_knn_predict*), cada uma em uma subpartição distinta do vetor de treinamento.

Após a finalização de todas as threads paralelas, será feita uma votação entre os resultados de predição obtidos em cada uma das threads e o resultado mais votado será dado como predição final do algoritmo. Essa técnica pode ser vista como uma forma de *Bootstrap aggregating* (ou *bagging*)[2], porém, diferente do *bagging* típico, os subconjuntos de amostragem são totalmente disjuntos.

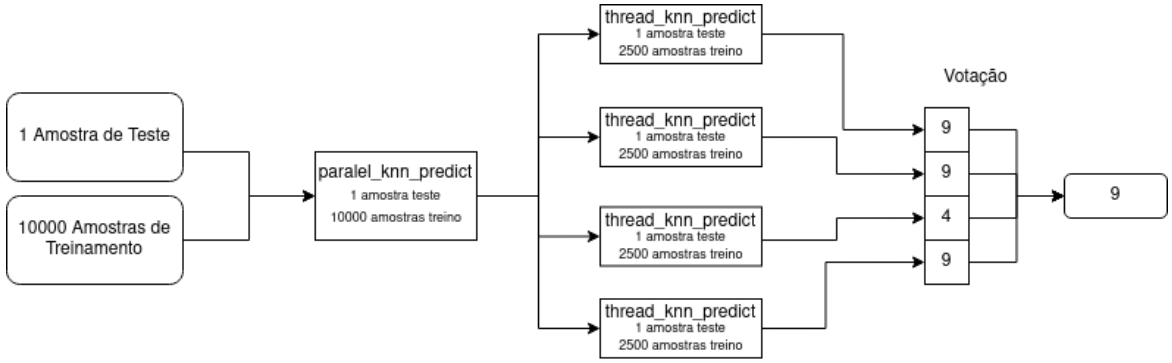

Figure 2: Função paralel_knn_predict aplicada para classificar uma amostra usando uma base de treinamento de 10000 amostras e 4 threads. O algoritmo classificou X como pertencente à classe 9.

3.2 Paralelismo Sobre A Base de Dados

A fim de otimizar a perfomance do algoritmo, foi aplicada uma segunda estratégia de paralelizacão concomitante à primeira.

O paralelismo sobre a base de dados de teste consiste em dividir a base em N subconjuntos disjuntos (em que N é o número de threads) e, em seguida, aplicar a função *paralel_knn* sobre cada um dos subconjuntos. A função *paralel_knn*, por sua vez, executará *paralel_knn_predict* para cada uma das amostras do subconjunto.

Os resultados serão salvos em um vetor compartilhado entre as threads que executam *paralel_knn*, preservando a posição original em que as amostras estão na base de dados de teste.

Após as threads finalizarem, a função *knn_classifier*, a qual realiza a computação descrita nesta seção, retornará um vetor com as predições para todas as amostras da base de teste.

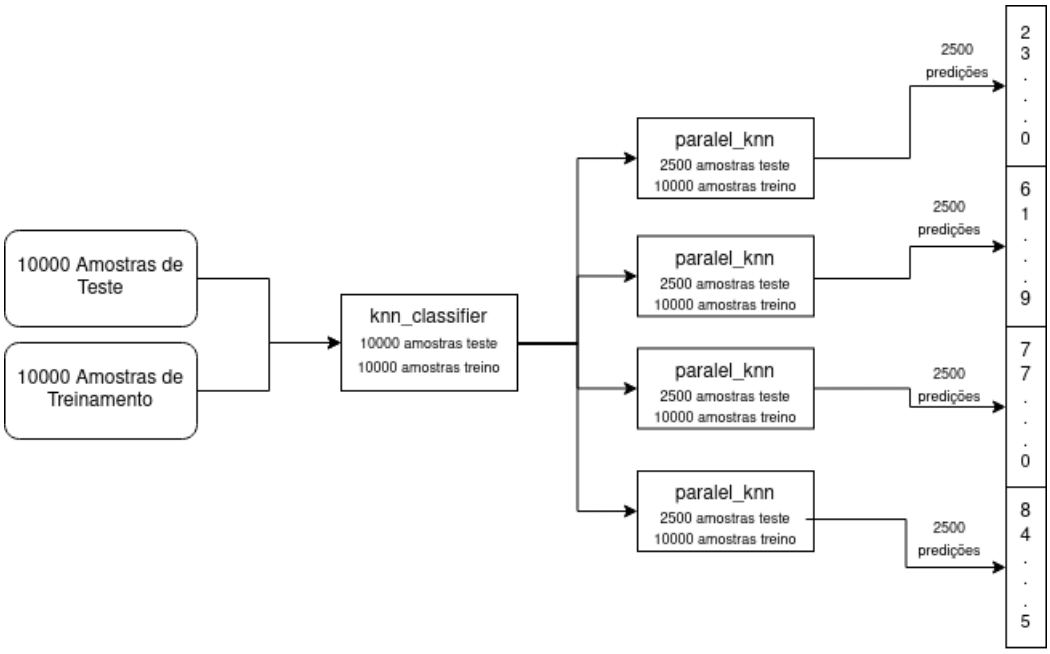

Figure 3: Função knn_classifier aplicada para classificar 10000 amostras de teste usando 10000 amostras de treinamento e 4 threads. O vetor de saída é compartilhado entre as threads de paralel_knn

4 Resultados

Com o intuito de comparar a performance do algoritmo KNN Concorrente implementado com o KNN Original, foram realizados testes de tempo de execução e acurácia com ambos algoritmos sobre a base de dados do MNIST [4].

A base de dados MNIST contém dígitos de 0 à 9, manuscritos, representados por uma matriz dimensões 28x28 em que cada elemento é um número de 0 à 255 correspondente ao valor de brilho da imagem na respectiva posição. O dataset pode ser obtido gratuitamente no site Kaggle e está em formato CSV.

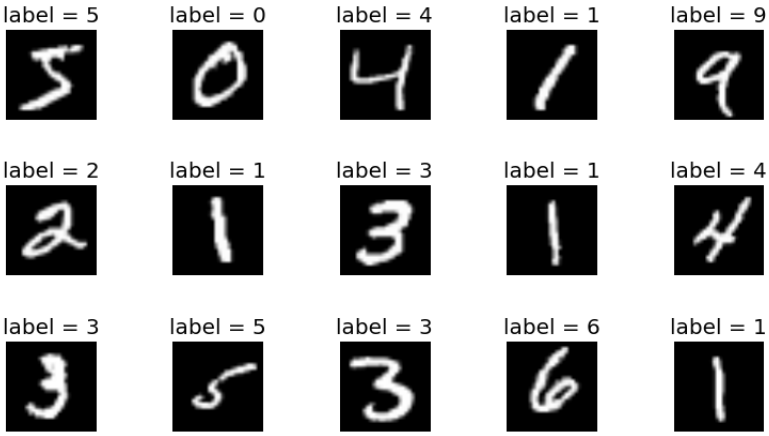

Figure 4: Exemplo de imagens da base de dados MNIST

Ao todo, cada algoritmo recebeu uma base de treinamento com 60000 amostras e uma base de teste com 10000 amostras. Os testes foram feitos em sistema operacional Ubuntu 20.04, com processador Intel Core i7 2600K e SSD. Para cada número de threads, os algoritmos foram executados 10 vezes.

Os resultados obtidos estão exibidos na tabela a seguir:

	Tempo de Execução +- σ	Acurácia
1 Thread	319.79 +- 1.05 s	97.06%
2 Threads	113.33 +- 1.9 s	96.05%
4 Threads	100.46 +- 0.29 s	96.06%
8 Threads	97.49 +- 0.26 s	95.54%

Conforme mostram os resultados, a versão concorrente performou até 3.2x mais rápido do que a versão sequencial, se comparada execução com 8 threads vs. 1 thread. Entretanto, foi observada uma perda de 1.5% de acurácia média na versão com mais threads.

A perda de acurácia ocorre porque a versão concorrente do algoritmo trata-se de um *ensemble* [5] de classificadores KNN, em que cada um dos classificadores é treinado apenas em um subconjunto da base de dados de treinamento. Em virtude disso, cada um dos subclassificadores tende a ser mais fraco que um classificador treinado com a base de dados inteira.

5 Conclusões

A implementação concorrente do algoritmo KNN desenvolvida apresentou melhora significativa do tempo de execução se comparada à implementação sequencial. Contudo, a

implementação sequencial apresenta em média uma acurácia 1% superior a versão concorrente.

É razoável supôr que ganho de performance tende a ser maior em computadores com CPUs com mais threads, visto que, na máquina em que foram realizados os testes, a CPU chegou à praticamente 100% de uso no cenário de 4 threads. Vale ressaltar que o i7 2600k utilizado nos testes têm 4 núcleos e 8 threads.

Por fim, em estudos futuros podem ser estudadas outras técnicas de concorrência com potencial de trazerem um ganho de performance, como por exemplo, a aplicação de paralelismo no calculo de cada distância.

References

[1] M. Ben-Ari and M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice-Hall international series in computer science. Prentice Hall, 1990. ISBN: 9780137118212. URL: https://books.google.com.br/books?id=jz4PAQAAMAAJ.

[2] Leo Breiman. "Bagging predictors." In: *Machine learning* 24.2 (1996), pp. 123–140.

[3] T. Cover and P. Hart. "Nearest neighbor pattern classification." In: *Information Theory, IEEE Transactions on* 13.1 (1967), pp. 21–27.

[4] L. Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]." In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 141–142. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2211477.

[5] Richard Maclin and David W. Opitz. "Popular Ensemble Methods: An Empirical Study." In: *CoRR* abs/1106.0257 (2011). arXiv: 1106.0257. URL: http://arxiv.org/abs/1106.0257.