UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
PROGRAMAÇÃO CONCORRENTE

JOÃO PEDRO SILVA DE SOUSA - DRE: 122122366
RAFAEL PASSOS ORIOLI GUIMARÃES - DRE: 122141077
GABRIEL GUIMARÃES HENRICI - DRE: 122137997

IMPLEMENTAÇÃO CONCORRENTE DO ALGORITMO K-VIZINHOS MAIS PŔOXIMOS

RELATÓRIO PARCIAL

RIO DE JANEIRO
2025

SUMÁRIO

1. Descrição do problema geral

1.1 Fundamentação teórica do KNN

O algoritmo **K-Vizinhos Mais Próximos (K-Nearest Neighbors - KNN)** é um dos métodos mais fundamentais e amplamente utilizados em aprendizado de máquina supervisionado. Proposto originalmente por Fix e Hodges em 1951, o KNN baseia-se em um princípio intuitivo: elementos semelhantes tendem a estar próximos no espaço de características. Essa hipótese de **proximidade local** permite que o algoritmo classifique novos dados com base na classe predominante de seus vizinhos mais próximos.

O KNN é considerado um algoritmo de **aprendizado preguiçoso** (*lazy learning*), pois não constrói um modelo explícito durante a fase de treinamento. Em vez disso, todo o conjunto de dados de treinamento é armazenado em memória, e o processamento ocorre no momento da predição. Essa característica confere ao algoritmo grande flexibilidade, mas também impõe desafios computacionais significativos, especialmente em conjuntos de dados volumosos.

1.2 Funcionamento do algoritmo sequencial clássico

Na implementação **sequencial**, o KNN executa as seguintes etapas para classificar uma nova amostra:

1. **Cálculo de distâncias**: Para cada ponto de teste, calcula-se a distância entre esse ponto e **todos os pontos** do conjunto de treinamento. A métrica mais comum é a distância euclidiana, definida como:

   $$ d(x, y) = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2} $$

   onde D é o número de dimensões (atributos) dos dados.

2. **Seleção dos K vizinhos**: As distâncias calculadas são ordenadas em ordem crescente, e os **K menores valores** são selecionados. Alternativamente, pode-se usar estruturas de dados como *heaps* ou *priority queues* para evitar ordenação completa.

3. **Votação majoritária**: Entre os K vizinhos selecionados, determina-se a classe mais frequente. No caso de classificação, a classe majoritária é atribuída ao ponto de teste. Para regressão, calcula-se a média (ou mediana) dos valores dos vizinhos.

4. **Resolução de empates**: Em caso de empate entre classes, diferentes estratégias podem ser aplicadas: escolher a classe do vizinho mais próximo, usar K ímpar, ou aplicar pesos baseados na distância.

1.3 Proposta de Trabalho

O trabalho irá se concentrar numa implementação concorrente para determinar os K vizinhos mais próximos de uma dada observação, não se estendendo a realizar uma predição sobre o dado de entrada, seja para regressão ou classificação. Dessa forma, deixaremos de fora as partes do algoritmo sequencial relacionadas a **Cálculo da Média dos Valores dos Vizinhos**, no caso de regressão, ou de **Votação seguida de Resolução de Empates** no caso de classificação.

Portanto, a concorrência será explorada nos seguintes aspectos

- Dividir o conjunto de teste e designar cada subconjunto a um fluxo de processamento.
- Dividir o conjunto de treinamento e paralelizar o cálculo das distâncias para uma observação.

1.4 Dados de entrada e saída

A entrada do programa consiste em:

- **Conjunto de treinamento**: Matriz de dimensões $N \times D$, onde N é o número de amostras e D é o número de atributos (features). Cada linha representa uma amostra com seus respectivos atributos e classe associada.
- **Conjunto de teste**: Matriz de dimensões $M \times D$, contendo as amostras a serem classificadas, onde M é o número de pontos de teste.
- **Parâmetro K**: Número inteiro positivo que define quantos vizinhos serão considerados na classificação. A escolha de K influencia diretamente o desempenho: valores pequenos podem causar *overfitting*, enquanto valores grandes podem *sub-ajustar* o modelo.
- **Métrica de distância**: Embora a distância euclidiana seja padrão, outras métricas como Manhattan, Minkowski ou Hamming podem ser especificadas.

A saída gerada pelo programa incluirá:

- **Índices dos K vizinhos**: Para cada ponto de teste, os índices dos K vizinhos mais próximos no conjunto de treinamento.

1.5 Complexidade computacional e limitações

A complexidade computacional do KNN sequencial é significativa e pode ser analisada em duas fases:

Fase de predição:

- Cálculo de distâncias: $O(N \times M \times D)$
- Ordenação ou seleção dos K menores: $O(M \times N \log K)$ usando *heaps*
- Votação: $O(M \times K)$

A complexidade total é dominada pelo termo $O(N \times M \times D)$, tornando o algoritmo computacionalmente custoso para grandes conjuntos de dados. Por exemplo, considerando:

- $N = 1.000.000$ amostras de treinamento
- $M = 10.000$ amostras de teste
- $D = 100$ dimensões

O número total de operações de distância seria de aproximadamente 10^{12} cálculos, cada um envolvendo D subtrações, quadrados e uma raiz quadrada.

1.6 Justificativa para solução concorrente

O KNN apresenta características que o tornam um **candidato ideal para paralelização**:

Paralelismo de dados natural: O cálculo das distâncias entre um ponto de teste e os pontos de treinamento é **completamente independente** do cálculo para outros pontos de teste. Não há dependências de dados entre diferentes amostras de teste, permitindo processamento simultâneo.

Operações computacionalmente intensivas: Cada cálculo de distância envolve múltiplas operações aritméticas (subtrações, multiplicações, adições e raiz quadrada), tornando o algoritmo **CPU-bound** e ideal para distribuição entre núcleos de processamento.

2. Projeto da solução concorrente

2.1 Estratégias de paralelização

Diferentes estratégias podem ser empregadas para paralelizar o algoritmo KNN, cada uma com suas vantagens e desafios:

2.1.1 Estratégia 1: Divisão por pontos de teste

Nesta abordagem, cada thread recebe um **subconjunto dos pontos de teste** e calcula independentemente os K vizinhos mais próximos para cada ponto de seu subconjunto.

Vantagens:

- **Mínima comunicação entre threads**: Cada thread trabalha em dados completamente independentes.
- **Simplicidade de implementação**: Não requer sincronização complexa durante o processamento.
- **Balanceamento de carga natural**: Se M (pontos de teste) for divisível pelo número de threads, a carga é naturalmente balanceada.

Desvantagens:

- **Replicação de dados**: Todo o conjunto de treinamento precisa ser acessível por todas as threads (memória compartilhada).
- **Escalabilidade limitada por M**: Se o número de pontos de teste for pequeno, pode haver subutilização de threads.

2.1.2 Estratégia 2: Divisão por blocos de treinamento

Cada thread processa um **subconjunto do conjunto de treinamento**, calculando distâncias parciais para todos os pontos de teste. Ao final, os resultados parciais precisam ser combinados.

Vantagens:

- **Melhor para conjuntos de teste pequenos**: Escala com N em vez de M.
- **Redução de pressão na memória cache**: Cada thread trabalha com um subconjunto menor de dados.

Desvantagens:

- **Necessidade de combinação de resultados**: Requer uma fase adicional para mesclar os K vizinhos parciais de cada thread.
- **Complexidade de sincronização**: Precisa de estruturas de dados thread-safe para agregação de resultados.
- **Overhead de comunicação**: Threads precisam compartilhar resultados intermediários.

2.2 Estratégia escolhida e justificativa

A estratégia a ser escolhida será uma intercalação das duas estratégias abordadas. O conjunto de treino será dividido e distribuído para fluxos diferentes, e cada fluxo irá, por sua vez, paralelizar o cálculo das distâncias euclidianas de uma observação a outra.

2.3 Parâmetros configuráveis

A implementação permitirá ajuste dinâmico de:

- **Número de threads**: De 1 até o número de núcleos disponíveis
- **Valor de K**: Número de vizinhos considerados

3. Casos de teste de corretude e desempenho

3.1 Testes de corretude

A validação da corretude é fundamental para garantir que a paralelização não introduza erros nos resultados. Os testes compararão a saída da versão concorrente com a versão sequencial.

3.1.1 Conjuntos de dados sintéticos

Para criação dos casos de teste, serão criados três conjuntos de treino e teste, de forma que, para cada caso, serão sorteados valores de M, N, D e K. As entradas das matrizes serão uniformemente distribuídas em um intervalo de [a, b] a serem também sorteados.

Dessa forma, para avaliar o desempenho, pode-se variar o conjunto de threads usadas para obter a solução do problema.

3.2 Testes de desempenho

Os experimentos de desempenho avaliarão a escalabilidade e eficiência da solução concorrente.

3.2.1 Métricas de avaliação

Tempo de execução (T)

- Média de 5 execuções para reduzir variância

Aceleração (A) $$S_p = \frac{T_{sequencial}}{T_{paralelo}(p)}$$ onde p é o número de threads

Eficiência (E) $$E_p = \frac{S_p}{p} \times 100%$$

Escalabilidade

- Avaliar como a aceleração varia com o aumento de threads.
- Identificar o ponto de saturação.
-

3.4 Resultados esperados

3.4.1 Corretude

Espera-se **100% de concordância** entre as versões sequencial e concorrente em todos os casos de teste, independentemente do número de threads.

3.4.2 Desempenho

Fatores limitantes

- **Lei de Amdahl**: Overhead de inicialização e finalização (~5% sequencial)
- **Contenção de memória**: Acesso concorrente ao conjunto de treinamento
- **Cache**: Thrashing em conjuntos muito grandes

Gráficos planejados

1. Speedup vs. Número de threads (várias configurações de N, M, D)
2. Eficiência vs. Número de threads
3. Tempo de execução vs. Tamanho do problema (log-log)

---

4. Referências bibliográficas

Livros e artigos fundamentais

- JAMES, G.; WITTEN, D.; HESTIE, T.;TIBSHIRANI, R.; TAYLOR, J. *an Introduction to Statistical Learning with Applications in Python.* Springer.

- TAN, P.-N.; STEINBACH, M.; KUMAR, V**.** *Introduction to Data Mining.* 2. ed. Pearson Education, 2019. Capítulos 4–5: Classificação e KNN.

- BISHOP, C. M**.** *Pattern Recognition and Machine Learning.* Springer, 2006. Seção 2.5.2: Nearest Neighbor Methods.

- HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2. ed. Springer, 2009. Capítulo 13: Prototype Methods.

- SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts.* 10. ed. Wiley, 2020. Capítulos 4–6: Threads, Sincronização e Deadlocks.

- PACHECO, P. *An Introduction to Parallel Programming.* Morgan Kaufmann, 2011. Capítulos 4–5: Programação com Pthreads.