Estratégias para Redução do Custo de Implementação de um Classificador Geométrico por Arestas de Suporte

Alan Cândido de Souza

Programa de Pós-Graduação em Engenharia Elétrica

Universidade Federal de Minas Gerais

Orientador: Prof. Cristiano Leite de Castro

Coorientador: Prof. Janier Arias Garcia

Dissertação de

Mestrado em Engenharia Elétrica

07/2019

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

ESTRATÉGIAS PARA REDUÇÃO DO CUSTO DE IMPLEMENTAÇÃO DE UM CLASSIFICADOR GEOMÉTRICO POR ARESTAS DE SUPORTE

Alan Cândido de Souza

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Cristiano Leite de Castro

Belo Horizonte - MG

Julho de 2019

DISSERTAÇÃO DE MESTRADO Nº 1130

ESTRATÉGIAS PARA REDUÇÃO DO CUSTO DE IMPLEMENTAÇÃO DE UM CLASSIFICADOR GEOMÉTRICO POR ARESTAS DE SUPORTE

Alan Cândido de Souza

DATA DA DEFESA: 12/07/2019

"Estratégias Para Redução do Custo de Implementação de Um Classificador Geométrico Por Arestas de Suporte"

Alan Cândido de Souza

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 12 de julho de 2019.

Por:

Prof. Dr. Cristiano Leite de Castro
DEE (UFMG) - Orientador

Prof. Dr. Janier Arias García
DELT (UFMG) - Coorientador

Prof. Dr. Frederico Gualberto Ferreira Coelho
DEE (UFMG)

Prof. Dr. Luiz Carlos Bambirra Torres
ICEA (UFOP)

Agradecimentos

Agradeço a Deus, pelas infinitas graças derramadas.

Aos meus pais, João e Luciene, que sempre foram a base de tudo e me incentivaram em todos os momentos.

Aos meus irmãos Geraldo, Edson, Aline e Marília, pelo suporte e apoio.

À minha namorada Raquel, pelo companheirismo e dedicação.

À toda a minha família. De forma especial aos meus tios Antônio, Socorro, Edson e Célia, pela acolhida nestes dias em que estive em Belo Horizonte. Aos tios Sincero e Lucimara, que sempre me auxiliaram.

Agradeço ainda aos meus orientadores Cristiano e Janier, e ao professor Antônio Braga, os quais sempre foram exemplo de excelência.

Aos colegas do LITC e os amigos do PPGEE.

À CAPES pelo apoio financeiro.

The best way to predict the future is to invent it

Alan Kay

Resumo

Este trabalho avalia estratégias para a redução do custo de implementação dos classificadores da família CHIP-clas, o qual é um classificador baseado na informação estrutural dos dados e é independente de algoritmos de otimização e ajustes de parâmetros. São abordadas duas propostas. A primeira busca avaliar o compromisso entre a redução da precisão numérica em relação ao desempenho do classificador. São avaliados dois formatos de representação em ponto-flutuante de 16 bits, os quais são comparados com a implementação de 32 bits. Os resultados indicaram que a redução da precisão numérica não compromete o desempenho do modelo para os casos avaliados, proporcionando um desempenho estatisticamente equivalente ao modelo de 32 bits além de apresentar maior eficiência e menor demanda de recursos de memória. A segunda proposta avalia uma técnica de computação paralela na fase de treinamento do classificador. Os resultados mostraram também um desempenho estatisticamente equivalente em relação ao modelo sem implementação da técnica paralela e ainda uma redução significativa do tempo de processamento em algumas das bases de dados avaliadas.

Abstract

This work evaluates strategies to reduce the implementation cost of classifiers based on the CHIP-clas model, which is independent of hyperparameter tuning and optimizations algorithms. The first proposal aims to evaluate the trade-off among numerical precision and model performance. Two 16-bit floating-point formats were compared to the 32-bit precision implementation. The results indicate that the model is robust to low precision computation, providing statistically equivalent results compared to the base model while reducing in a half the memory demand. The second proposal evaluates a method that implements a parallel computation technique to the classifier's training stage. Results also indicated statistically equivalent performance and a speed-up in processing time in some databases.

Sumário

Lista de Abreviaturas

$\mathcal{AS}$ Arestas de Suporte.

API *Application Programming Interface*.

ASIC *Application-Specific Integrated Circuits*.

CI Circuitos Integrados.

CLB *Configurable Logic Block*.

CPLD *Complex Programmable Logic Devices*.

DSP Processamento Digital de Sinais.

EP Elemento de Processamento.

FPGA *Field Programmable Gate Arrays*.

GG Grafo de Gabriel.

GPGPU *General Purpose Computing on Graphics Processing Units*.

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos.

K-NN *K-Nearest Neighbors*.

LUT *Look-up Tables*.

MSB *Most Significant Bit*.

MSE Erro Médio Quadrático.

PCA *Principal Component Analysis*.

QP *Quadratic Programming Problem*.

RTL *Register-transfer level*.

SVM Máquinas de Vetores de Suporte.

VHDL *Very High Speed Integrated Circuits Hardware Description Language*.

VLSI *Very Large System Integration*.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Um sistema *Cyber*-Físico é definido por (Lee & Seshia, 2016) como a integração de computação e sistemas físicos, redes de monitoramento e sistemas embarcados. Esse contexto engloba ainda dispositivos de IoT (*Internet of Things*), que juntos formam uma rede de objetos físicos que possuem sistemas dedicados para geração e transmissão de dados, sensoriamento e comunicação com o ambiente externo. A empresa de pesquisa e consultoria *Gartner Inc.*, estima que até 2020, haverão cerca de 20 bilhões de dispositivos de IoT conectados (Hung, 2017).

Paralelo ao crescimento do número de dispositivos de IoT conectados, a grande quantidade de dispositivos móveis (*smartphones* e aparelhos "vestíveis", como relógios inteligentes, dentre outros) e o aumento do poder computacional contribuem de forma significativa para o crescimento da disponibilidade de dados em diferentes contextos e aplicações. Com isso, algoritmos de aprendizado de máquina estão cada vez mais presentes no dia a dia das pessoas e empresas, uma vez que tais ferramentas possibilitam a automatização de tarefas e otimização de processos — através dos dados disponíveis — que vão desde sistemas de *software* até aplicações em processos industriais.

Entretanto, a complexidade computacional de alguns algoritmos de aprendizado dificulta sua implementação em sistemas embarcados ou dispositivos com recursos limitados em termos de tamanho, processamento e memória. Geralmente, tais algoritmos são implementados nesse tipo de sistema através da com-

putação em nuvem. Nesse contexto, os modelos são previamente treinados em servidores remotos uma vez que geralmente esta etapa demanda mais recursos computacionais. No entanto, essa abordagem traz perigos à segurança dos dados pois estes precisam ser transferidos pela Internet entre servidores, que por sua vez podem ser comprometidos por *cyber* ataques, colocando em risco informações importantes de usuários e empresas.

Observa-se ainda nos últimos anos, um crescimento na preocupação em relação à privacidade dos dados de usuários, o que tem motivado o desenvolvimento de metodologias e algoritmos que possam ser treinados utilizando recursos dos próprios dispositivos (Deng, 2019), eliminando a necessidade do fluxo de dados externo.

Dessa forma, alguns trabalhos na literatura apresentam soluções para o desenvolvimento de algoritmos de aprendizado de máquina que possam ser executados de forma adequada em plataformas de *hardware* de sistemas embarcados ou dispositivos móveis com limitação de recursos. Essas abordagens exploram a implementação de modelos de aprendizado mais simples e ainda o desenvolvimento de novas arquiteturas. As vantagens desse tipo de metodologia são evidentes, garantindo segurança nos dados de usuários e empresas e redução dos custos de computação em nuvem.

1.1.1 Aprendizado de máquina em *hardware*

Após o hiato no financiamento e produção científica na área de aprendizado de máquina e reconhecimento de padrões na década de 1970, provocados por questionamentos acerca da viabilidade do algoritmo *perceptron* (Rosenblatt, 1958), que fora o modelo precursor do aprendizado de máquina de abordagem conexionista, ocorreu em 1980 um retorno significativo da pesquisa na área, relacionado ainda à aplicação do algoritmo de *backprogration* (Rumelhart *et al.*, 1985) (Lippmann, 1987). Na década de 1990 ocorreu o desenvolvimento das Máquinas de Vetores de Suporte (SVM) (Cortes & Vapnik, 1995) e nas duas primeiras décadas do século *XXI* ocorreu a consolidação da subárea de inteligência computacional voltada para redes neurais de camada profunda (*Deep Learning*) (LeCun *et al.*, 2015),

as quais apresentaram resultados expressivos de reconhecimento de padrões em aplicações de processamento de imagens e sinais de áudio.

A evolução e desenvolvimento dos circuitos integrados contribuiu de forma significativa para o crescimento da área de aprendizado de máquina. No final da década de 1980, foi apresentado o ETANN (*Electrically Trainable Artificial Neural Network*) (Holler *et al.*, 1989), proposto pela Intel, o qual possuía sinapses adaptativas mas no entanto necessitava de um computador externo para realizar as operações mais complexas. Com a efetivação da Lei de *Moore* acerca do crescimento no número de transistores em um circuito integrado, os modelos de aprendizado de máquina se beneficiaram do elevado poder computacional dos novos processadores. Pesquisadores viram ainda a possibilidade da utilização de GPUs (*Graphics Processing Unit*) para acelerar os modelos de aprendizado, uma vez que os mesmos implementam diversas operações vetoriais e matriciais, as quais são executadas de forma mais eficiente neste tipo de arquitetura em comparção com processadores de uso geral.

Nesse contexto surgiu ainda o interesse por parte das empresas em desenvolver ASICs (*Application-Specific Integrated Circuit*) voltados para execução eficiente de operações matriciais utilizadas em algoritmos de aprendizado de máquina. Como exemplo pode-se citar a TPU (*Tensor Processing Unit*) desenvolvida pela Google (Jouppi *et al.*, 2018) e a *Tensor Cores* desenvolvida pela Nvidia (Markidis *et al.*, 2018). Dessa forma, ocorreu um renascimento do mercado de desenvolvimento e fabricação de *chips* específicos para aplicações de aprendizado de máquina. Como exemplo pode-se citar *startups* como a *Graphcore*, que desenvolveu uma arquitetura de processamento baseada em grafos denominado IPU (*Intelligent Processing Unit*) (Manne *et al.*, 2017), sendo o acesso a essas novas arquiteturas facilitado pelo crescimento das tecnologias de computação em nuvem.

1.2 Contextualização e Objetivos

Em (Torres *et al.*, 2015c) é apresentado um novo classificador denominado CHIPclas (Classificador por Arestas de Suporte), o qual utiliza a informação estrutural dos dados a partir da construção do grafo de Gabriel, de forma que se possa obter

a margem máxima de separação entre as classes em problemas de classificação binária para aplicações de aprendizado de máquina supervisionado. Além de apresentar resultados estatisticamente equivalentes às Máquinas de Vetores de Suporte (SVMs) com *kernel* RBF e Polinomial, em testes com bases de dados reais (Torres, 2016), esse classificador destaca-se pelo fato de ser independente de ajustes de hiperparâmetros, o que elimina a necessidade de implementação de técnicas de busca em *grid* ou validação cruzada (Torres *et al.*, 2015a).

A partir do CHIP-clas, surgiram novos algoritmos e metodologias baseadas em sua estrutura, como a metodologia para seleção de parâmetros em redes neurais de base radial (Torres *et al.*, 2014) e um classificador que utiliza a estrutura do grafo e classifica os dados a partir de um modelo de mistura de Gaussianas (Torres *et al.*, 2015b). Foram desenvolvidos ainda um modelo de aprendizado incremental (de Freitas Diadelmo, 2016) e um método que propõe a utilização de aprendizagem de métrica na etapa de filtragem do modelo (Gomes *et al.*, 2017), dentre outros trabalhos.

As características do CHIP-clas relacionadas à utilização da estrutura dos dados para classificar novas amostras o torna adequado para aplicações em sistemas embarcados, uma vez que não há necessidade de configurações adicionais por parte do usuário e resolução de problemas de otimização. Um dos problemas no entanto é o seu custo computacional, dada a necessidade de se obter a distância entre todas as amostras a partir de uma métrica predefinida — como a Distância Euclidiana por exemplo — para organizar a estrutura do grafo de Gabriel.

Nesse contexto, (dos Reis Gade *et al.*, 2017) aborda o problema do custo computacional do CHIP-clas. O novo método, denominado NN-clas, visa simplificar a etapa de classificação do modelo de forma a reduzir seu custo computacional e viabilizar sua implementação em sistemas embarcados. Os resultados mostraram desempenho estatisticamente equivalente ao CHIP-clas nas bases de dados avaliadas.

Em (dos Reis Gade, 2018) é proposta um arquitetura para implementação do CHIP-clas em *hardware* digital. Os resultados no entanto, mostraram um crescimento do consumo de recursos associado ao número de amostras e dimensões das bases de dados avaliadas. Dessa forma, este presente trabalho visa dar continuidade às propostas apresentadas em (dos Reis Gade, 2018), assim como propor

novas soluções para o problema do custo computacional relativo à potencial utilização do CHIP-clas em sistemas embarcados ou dispositivos com recursos de processamento limitados.

São avaliadas duas propostas cujo objetivo é reduzir o consumo de recursos do NN-clas em sistemas embarcados. Apresenta-se um estudo da viabilidade de implementação do NN-clas com precisão numérica reduzida e ainda é avaliada uma proposta de computação paralela, de forma a promover a escalabilidade do modelo em bases de dados com quantidades significativas de amostras e reduzir a demanda de recursos associada ao crescimento amostral dos dados.

1.2.1 Objetivo Geral

Este trabalho tem por objetivo geral propor novas soluções e metodologias para o problema do custo computacional do CHIP-clas no contexto de sua implementação em sistemas embarcados.

1.2.2 Objetivos Específicos

1. Analisar os trabalhos propostos em relação à melhoria da eficiência na implementação do CHIP-clas, de forma a identificar pontos de melhoria e continuidade;

2. Fazer um estudo da etapa de treinamento do CHIP-clas, relativo à construção do grafo de Gabriel, com o objetivo de propor novas soluções de implementação mais eficientes;

3. Identificar e implementar as propostas mais promissoras associadas à redução do custo computacional do modelo

4. Quantificar o desempenho do NN-clas para valores reduzidos de precisão numérica, de forma a validar a viabilidade da implementação do mesmo com precisão numérica reduzida;

1.3 Contribuições

Dentre as contribuições deste trabalho destacam-se:

- Criação de um *software* em Matlab® para conversão de valores decimais para binários em ponto flutuante nos formatos IEEE 754 e *bfloat* [1].
- Implementação e análise do impacto da redução da precisão numérica para o desempenho do classificador.
- Proposta de um técnica de construção do grafo de Gabriel em paralelo de forma a melhorar a eficiência do NN-clas na etapa de treinamento.
- Criação de uma base de códigos em *python* para implementação do CHIP-clas, com as diferentes opções de configuração, através do algoritmo de classificação do NN-clas [2].
- Publicação do artigo: Souza, Alan C., et al. *"Improving the Efficiency of Gabriel Graph-based Classifiers for Hardware-optimized Implementations."* 2019 *XXII Symposium on Image, Signal Processing and Artificial Vision* (STSIVA). IEEE, 2019.

1.4 Organização do texto

Este trabalho está organizado da seguinte forma: No capítulo 2 é feita uma breve revisão da teoria dos grafos, a fim de introduzir a estrutura básica do CHIP-clas. São abordados de forma sucinta alguns classificadores de margem máxima. Após isso é feita uma análise sobre alguns trabalhos relacionados, tais como a proposta do NN-clas e sua respectiva implementação em *hardware*.

No capítulo 3 são abordadas propostas de redução de precisão na representação numérica e no capítulo 4 é apresentada uma metodologia para melhoria da eficiência na etapa do treinamento do CHIP-clas a partir da implementação de um método de computação paralela. No capítulo 5 são descritos os experimentos

[1] Disponível em: https://github.com/alancsouza/IEEE754_float2binary_converter
[2] Disponível em: https://github.com/alancsouza/chip_clas

executados de forma a validar as hipóteses levantadas, assim como os resultados obtidos nesses experimentos. Por fim, no capítulo 6 é realizada uma análise sobre os resultados encontrados e apresentadas propostas de continuidade do trabalho.

Capítulo 2

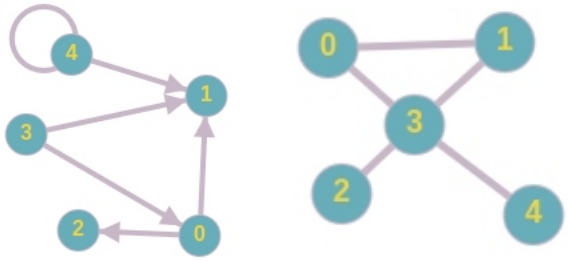
Referencial Teórico

2.1 Teoria de Grafos

Um Grafo (G) é um estrutura de dados abstrata e pode ser definido matematicamente como um conjunto de vértices ou nós ($\mathcal{V}$) e arestas ou *links* (A), sendo $G = (\mathcal{V}, \mathcal{A})$. Essa abstração é capaz de modelar diversas aplicações que variam desde ferramentas de busca na Internet até problemas de otimização de roteamento de veículos em empresas de transporte (Ziviani, 2010). A Figura 2.1 apresenta dois grafos, onde os vértices estão indicados pelos números e as arestas pelas conexões formadas entre eles.

Um grafo pode ser definido como direcionado e não direcionado. Nos grafos direcionados (ou dirigidos), também chamados de digrafos, cada aresta possui uma orientação (Bondy *et al.*, 1976). Neste caso pode haver um vértice orientado para si mesmo, chamado de *self-loop* (ver vértice 4 na na Figura 2.1a). Isto não ocorre em grafos não direcionados, como apresentado na Figura 2.1b, uma vez que não há uma orientação explícita de arestas (Ziviani, 2010).

Outro atributo importante em grafos é o grau do vértice, que é definido como o número de arestas incidentes sobre o mesmo. No caso de grafos direcionados, existem duas formas de avaliar o grau do vértice, em relação às arestas que entram (grau de entrada) e que saem (grau de saída) do vértice (Prestes, 2016).



(a) (b)

Figura 2.1: Grafo direcionado e não direcionado

2.1.1 Formas de representação

Grafos podem ser representados de forma matricial, através das matrizes de incidência e de adjacência. A matriz de incidência associa cada vértice à sua respectiva aresta, sendo uma matriz $\mathcal{V} \times \mathcal{A}$. Já a matriz de adjacência é uma matriz $\mathcal{V} \times \mathcal{V}$, onde cada aresta é representada pela associação entre dois vértices, para grafos não-direcionados e não-ponderados (sem peso entre arestas) (Bondy *et al.*, 1976). A matriz de adjacência da Figura 2.1b é apresentada abaixo, onde a formação de uma aresta entre dois vértices $\mathcal{A}_{ij}$ é representada como "1"na matriz $n \times n$, onde n representa o número de vértices e $\{i, j = 1, \ldots, n\}$ representam os índices dos vértices nas linhas e colunas da matriz, respectivamente.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

2.1.2 Subgrafos

Determinado grafo G' é considerado um subgrafo de G caso seu conjunto de vértices e arestas estejam contidos em G. Matematicamente temos que $G' = (\mathcal{V}', \mathcal{A}')$ é um subgrafo de $G = (\mathcal{V}, \mathcal{A})$ se e somente se $\mathcal{V}' \subseteq \mathcal{V}$ e $\mathcal{A}' \subseteq \mathcal{A}$ (Ziviani, 2010).

2.1.3 Diagrama de *Voronoi*

O diagrama de *Voronoi* é uma construção geométrica que possui aplicações em diversas áreas, como por exemplo meteorologia, planejamento urbano, estatística, cartografia, dentre outras (Rezende *et al.*, 2000). Dado um conjunto finito de pontos distintos $P = (p_1, p_2, p_3, \ldots, p_n)$ em um plano, o diagrama de *Voronoi* divide o plano de forma que cada espaço esteja associado ao ponto mais próximo (Okabe *et al.*, 2009). Essa divisão ocorre a partir do conceito de dominância, conforme apresentado em (2.1). Sendo dois pontos arbitrários $p_i, p_j \in P$, o domínio de p_i sobre p_j é definido como um subconjunto do espaço que é mais próximo de p_i em relação à p_j (Aurenhammer, 1991). A Figura 2.2c apresenta a representação do Diagrama de Voronoi.

$$dom(p_1, p_j) = x \in \mathbb{R}^2 \mid \delta(x, p_i) \leq \delta(x, p_j) \tag{2.1}$$

onde $\delta(.)$ é a função que calcula a distância Euclidiana.

Essa divisão cria $n - 1$ polígonos convexos, formando semiplanos limitados pela bissetriz do seguimento de reta formado por dois pontos vizinhos $\overline{p_i, p_j}$. Dessa forma, o diagrama de *Voronoi* é considerado um grafo planar, no qual os seguimentos de reta que delimitam cada polígono formam arestas e a interseção entre eles formam vértices (Aurenhammer, 1991).

2.1.4 Triangulação de *Delaunay*

Considerando o contexto apresentado na subseção anterior, a triangulação de *Delaunay* associa cada ponto em P ao ponto pertencente ao espaço vizinho, de forma que uma aresta é formada entre dois pontos ($p_i, p_j \in P$) se e somente se as suas respectivas regiões de *Voronoi* compartilharem uma aresta entre si (Aurenhammer, 1991), conforme mostrado na Figura 2.2b.

2.1.5 Grafo de Gabriel

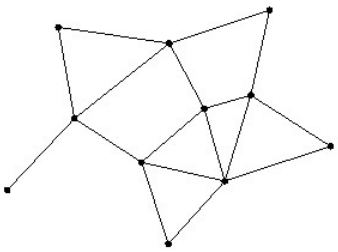
O Grafo de Gabriel (GG) é um grafo $G(\mathcal{V}, \mathcal{E})$, planar não-direcionado conforme mostrado na Figura 2.2a, onde cada elemento pertencente ao conjunto de pontos é um vértice, e dois pontos formam uma aresta se e somente se não houver

qualquer outro vértice na circunferência cujo diâmetro é formado por esses dois pontos (Gabriel & Sokal, 1969). Essa afirmação pode ser estendida para uma hiperesfera em espaços de maior dimensão. Formalmente temos:

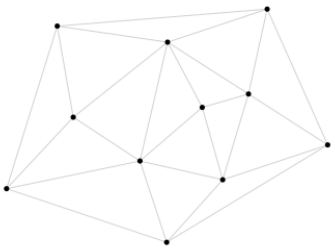
$$\delta^2(x_i, x_j) \leq \delta^2(x_i, x_z) + \delta^2(x_j, x_z), \quad \forall\, x_z \in \mathcal{V} \quad \text{e} \quad i \neq j \neq z \tag{2.2}$$

onde $\delta(.)$ é o operador que calcula a distância Euclidiana entre dois pontos.

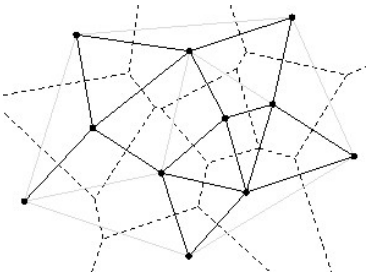
O Grafo de Gabriel é um subgrafo da Triangulação de *Delaunay* (Berg *et al.*, 2008). A Figura 2.2 apresenta a relação entre o Grafo de Gabriel, a Triangulação de Delaunay e o Diagrama de Voronoi.



(a) Grafo de Gabriel



(b) Triangulação de Delaunay



(c) Grafo de Gabriel e Triangulação de Delaunay sobrepostos ao Diagrama de Voronoi

Figura 2.2: Relação entre os grafos planares. Adaptado de (Maignan & Gruau, 2011)

2.2 Problemas de classificação binária e maximização de margem

Em aprendizado supervisionado, seja uma base de dados de treinamento $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$ formada por N pares de amostras de entrada e seus respectivos rótulos, sendo $[\mathbf{x} = x_1, x_2, \ldots, x_N]^T, \mathbf{x} \in \mathbb{R}^n$ e $\mathbf{y}_i \in \{-1, 1\}$. Caso as classes sejam linearmente separáveis, existe um conjunto infinito de hiperplanos que podem dividir os dados. Dessa forma, deseja-se obter o hiperplano que melhor represente a separação das classes (James *et al.*, 2013). Essa definição do hiperplano ótimo pode ser feita através da análise da margem de separação. Dada a equação do hiperplano:

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{2.3}$$

onde $\mathbf{w}$ é o vetor normal, $\phi(.)$ é a função de mapeamento (*kernel*) para o espaço de características e b representa o *bias*.

A margem de separação é definida como a menor distância entre a fronteira de separação e as amostras de treinamento. Dessa forma, a maximização de margem busca obter o menor erro de generalização (Bishop, 2006). A distância perpendicular entre determinada amostra de treinamento e o hiperplano $y(\mathbf{x}) = 0$ é definida por $|y(\mathbf{x})|/||\mathbf{w}||$, onde $y(\mathbf{x})$ é dado por 2.3. Dessa forma, a solução de margem máxima é dada por:

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \left\{ \frac{1}{||\mathbf{w}||} \min_i [y_i(\mathbf{w}^T \phi(\mathbf{x}) + b)] \right\} \tag{2.4}$$

Uma vez os pontos de interesse que definem a margem de separação — denominados vetores de suporte — são os pontos mais próximos da borda de separação, pode-se definir:

$$y_i(\mathbf{w}^T \phi(\mathbf{x}) + b) = 1 \tag{2.5}$$

De forma que todos os pontos satisfaçam:

$$y_i(\mathbf{w}^T \phi(\mathbf{x}) + b) \geq 1 \quad \forall \quad i \in \{1, \ldots, N\} \tag{2.6}$$

sendo $y_i \in \{+1, -1\}$.

Dessa forma, a Equação (2.4) passa a maximizar $||\mathbf{w}||^{-1}$ e pode ser reescrita como um problema de minimização, como segue:

$$\begin{aligned} &\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2 \\ s.a. \quad &y_i(\mathbf{w}^T\phi(\mathbf{x}) + b) \geq 1 \end{aligned} \tag{2.7}$$

2.2.1 Máquinas de Vetores de Suporte

O algoritmo SVM (*Support Vector Machines*) (Cortes & Vapnik, 1995) busca maximizar a margem de separação através da resolução de um problema de otimização convexo. Conforme mostrados na Figura 2.3, o conjunto de vetores de suporte é definido pelas amostras que satisfaçam as seguintes condições:

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 0, & y_i = +1 \\ \mathbf{w}^T\mathbf{x}_i + b < 0, & y_i = -1 \end{cases} \tag{2.8}$$

Para os casos em que há sobreposição de classes, onde os dados não podem ser separados linearmente no espaço de características, é utilizado um parâmetro de relaxamento, com o objetivo de flexibilizar o critério de margem rígida, e ainda um parâmetro de regularização (c) que controla a quantidade de erros de classificação e a generalização do modelo.

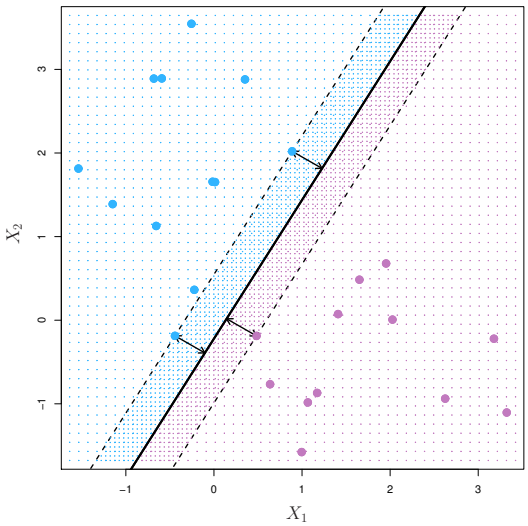


Figura 2.3: Vetores de suporte, extraído de (James *et al.*, 2013).

2.3 CHIP-clas

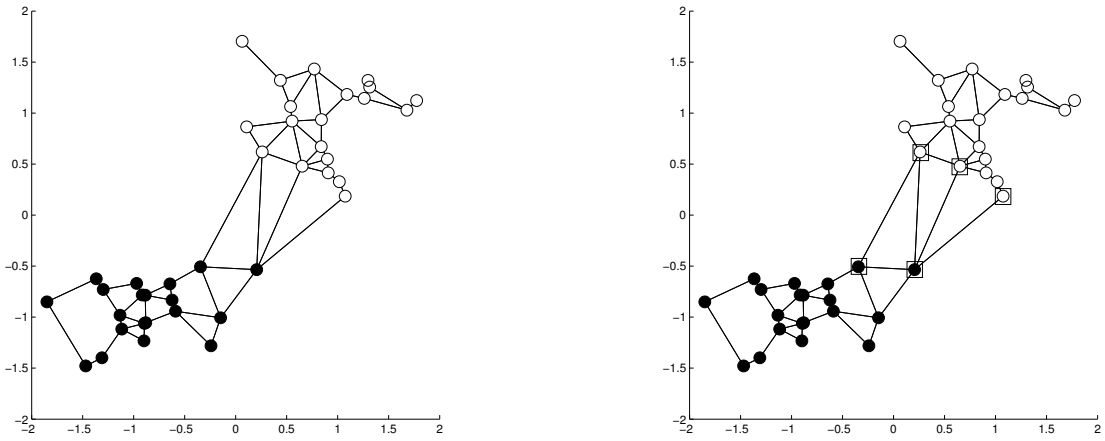
O Chip-clas (Torres *et al.*, 2015c), é um classificador binário de margem máxima, que utiliza a estrutura dos dados para obter informações sobre a margem de separação dos dados, a partir da construção do Grafo de Gabriel (Gabriel & Sokal, 1969). As etapas da implementação do CHIP-clas são descritas a seguir:

2.3.1 Etapa de Treinamento

A etapa de treinamento do CHIP-clas se dá através da construção do Grafo de Gabriel, de forma a definir os pares de dados que formam uma aresta entre si. Essa definição é feita a partir de (2.2). Neste sentido, cado amostra é considerada como um vértice do grafo e a construção das arestas é feita a partir do cálculo da distância entre um vértice e os demais vértices pertencentes ao grafo. A estrutura do Grafo de Gabriel é apresenta na Figura 2.4a.

A partir da estrutura do grafo de Gabriel e ainda com informações a priori sobre os rótulos das classes, é definido um conjunto especial de arestas denominado Arestas de Suporte ($\mathcal{AS}$). Este nome é uma alusão aos vetores de suporte do classificador SVM. O conjunto $\mathcal{AS}$ constitui a saída da etapa de treinamento e é formado pelas arestas que conectam vértices de classes opostas. Tais vértices

são denominados vetores geométricos (Torres, 2016) e estão destacados na Figura 2.4b. A definição do conjunto $\mathcal{AS}$ é importante uma vez que definem a margem de separação das classes.



(a) Grafo de Gabriel

(b) Conjunto de arestas de suporte

Figura 2.4: Treinamento do CHIP-clas

2.3.2 Filtragem dos dados

Uma vez que o CHIP-clas depende da disposição topológica dos dados — principalmente os que estão próximos à margem de separação — para classificar corretamente novos dados, ele sofre influência negativa de bases com dados ruidosos ou que apresentam sobreposição entre as classes. A Figura 2.5a mostra uma situação comum em bases reais, onde as classes se sobrepõem de forma que não há uma separação clara entre as mesmas, fazendo com que a obtenção das arestas de suporte seja dificultada. Para solucionar este problema, em (Torres, 2016) é proposto um método de filtragem, de forma que os dados situados na região de sobreposição são considerados como ruídos e removidos da base de dados.

Esse processo de filtragem ocorre da seguinte forma: o Grafo de Gabriel é construído sobre a base de dados, conforme mostrado na Figura 2.5b. Nesta figura pode-se observar a dificuldade da obtenção do conjunto de arestas de suporte, devido à sobreposição dos dados. Portanto, é calculada uma medida de qualidade $q(.)$ baseada no grau do vértice (Aupetit & Catz, 2005), cuja finalidade é avaliar

topologicamente sua relação com vértices vizinhos de classes opostas, conforme apresentado em (2.9).

$$q(\mathbf{x}_i) = \frac{\hat{\mathcal{A}}(\mathbf{x}_i)}{\mathcal{A}(\mathbf{x}_i)}, \tag{2.9}$$

onde $\mathcal{A}(\mathbf{x}_i)$ representa o grau do vértice e $\hat{\mathcal{A}}(\mathbf{x}_i)$ o número de arestas da mesma classe conectadas a $\mathbf{x}_i$.

Caso essa medida de qualidade seja $q(\mathbf{x}_i) = 0$, isso indica que $\mathbf{x}_i$ pode ser considerado um ruído. Caso seja $q(\mathbf{x}_i) = 1$, todos os vértices conectados a $\mathbf{x}_i$ são da mesma classe, o que indica que este dado não é um ruído. Entretanto, quando esse valor varia ($0 < q(\mathbf{x}_i) < 1$), isso pode indicar tanto um ruído quanto uma amostra candidata ao conjunto $\mathcal{AS}$.

Dessa forma, para determinar se os dados cuja medida de qualidade estão entre $0 < q(\mathbf{x}_i) < 1$, podem ser considerados como ruídos, (Torres, 2016) apresenta uma metodologia para definição de um valor limiar de $q(.)$, avaliado separadamente em cada classe, conforme apresentado em (2.10) e (2.11).

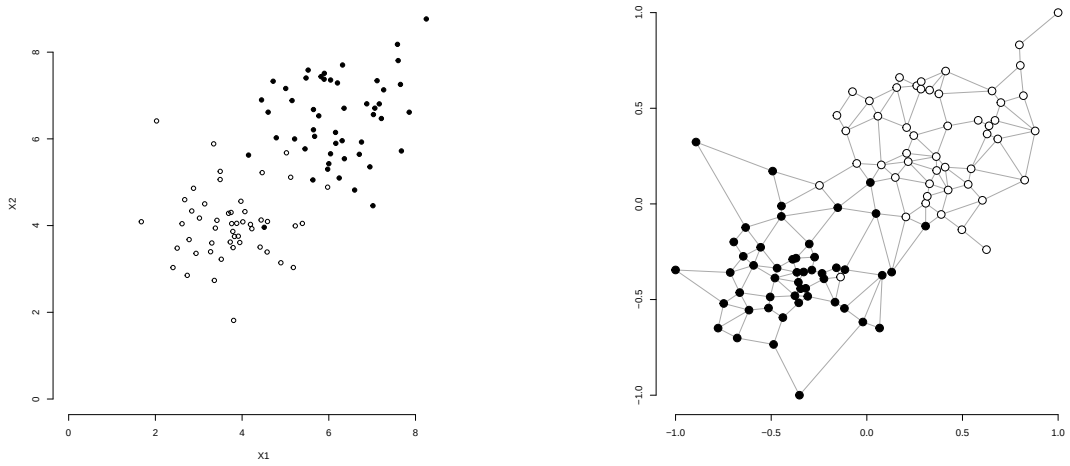
$$t^+ = \frac{\sum\limits_{q(\mathbf{x}_i)\in\mathcal{Q}^+} q(\mathbf{x}_i)}{|\mathcal{Q}^+|} \tag{2.10}$$

$$t^- = \frac{\sum\limits_{q(\mathbf{x}_i)\in\mathcal{Q}^-} q(\mathbf{x}_i)}{|\mathcal{Q}^-|} \tag{2.11}$$

onde $\mathcal{Q}^+$ e $\mathcal{Q}^-$ representam a medida de qualidade para os dados com rótulo $+1$ e -1 respectivamente.

Dessa forma, as amostras de treinamento cuja medida de qualidade é menor que o limiar respectivo à sua classe, são identificadas e removidas, conforme apresentado na Figura 2.6.

Dado que muitas bases com aplicações reais possuem dados sobrepostos, a etapa de filtragem é crucial para um desempenho adequado do CHIP-clas, de forma que esta etapa é utilizada na fase inicial do classificador, antes da etapa de treinamento. Alguns trabalhos na literatura avaliam outras metodologias de filtragem e remoção dos dados ruidosos.

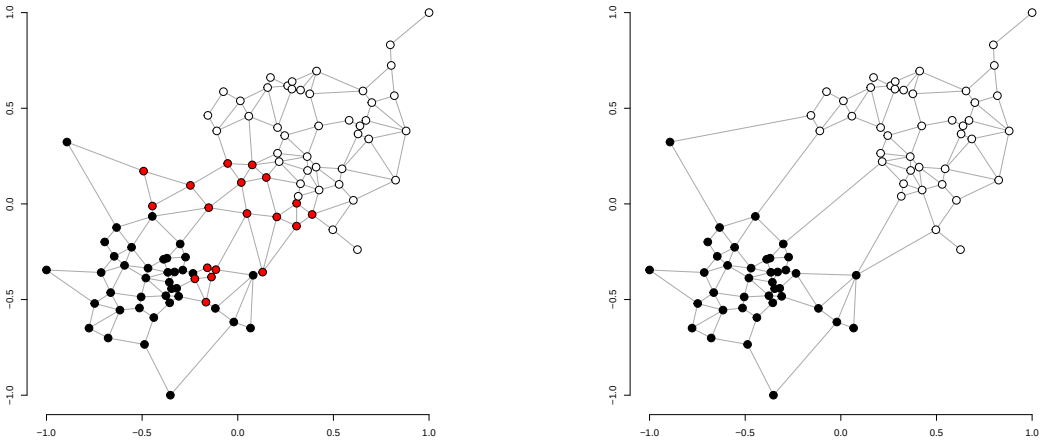


(a) Classes sobrepostas (b) Grafo de Gabriel

Figura 2.5: Grafo de Gabriel para amostras com bases sobrepostas

Em (Gomes *et al.*, 2017) é proposta uma medida de suavização da margem máxima de separação, através de uma técnica de aprendizado de métrica, a qual é utilizada como uma metodologia para remoção dos ruídos. Essa técnica indica uma redução do número de amostras removidas, apresentando um impacto positivo — especialmente para bases de dados com uma quantidade menor de amostras— uma vez que, ao utilizar a técnica tradicional baseada na medida topológica de qualidade, comumente ocorre uma redução significativa do número de amostras, o que pode significar perda de informação em determinadas bases de dados.

Outra proposta de filtragem dos dados na borda de separação é proposta em (Salgado, 2019), onde são exploradas características intrínsecas da teoria de Grafos, como medidas de centralidade e métricas de distância, aplicadas à regularização do classificador.



(a) Identificação das amostras consideradas como ruído

(b) Eliminação dos ruídos e construção do Grafo de Gabriel

Figura 2.6: Identificação e filtragem dos ruídos

2.3.3 Etapa de Classificação

Uma vez que se obtém, através da saída da etapa de treinamento, o conjunto de arestas de suporte $\mathcal{AS}$, a etapa de classificação se baseia nos pontos médios destas arestas, conforme mostrado na Figura 2.7. Em cada ponto médio é traçado um hiperplano local, de forma que a região de máxima separação das classes é obtida através da combinação hierárquica de todos os hiperplanos através de uma *Gating network* (Figura 2.8).

Dado um conjunto de m hiperplanos locais $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$, uma amostra arbitrária de teste $\mathbf{x}$ é classificada em relação ao hiperplano $\mathcal{H}_i$ por:

$$\mathcal{H}_i(\mathbf{x}) = \text{sign}(\mathbf{x}^T \mathbf{w}_i - b_i) \tag{2.12}$$

onde sign(.) é a função sinal e $\mathbf{w_i} = (x_i - x_j)$ é o vetor normal do hiperplano, para duas amostras (x_i, x_j) pertencentes a classes opostas, com orientação de $\mathbf{w}$ definida por:

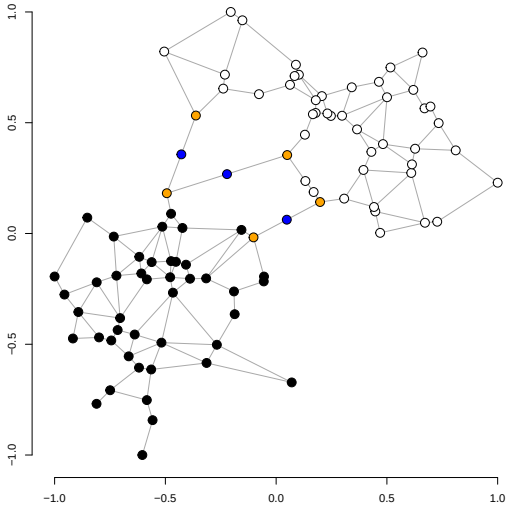


Figura 2.7: Pontos médios formados a partir das arestas de suporte

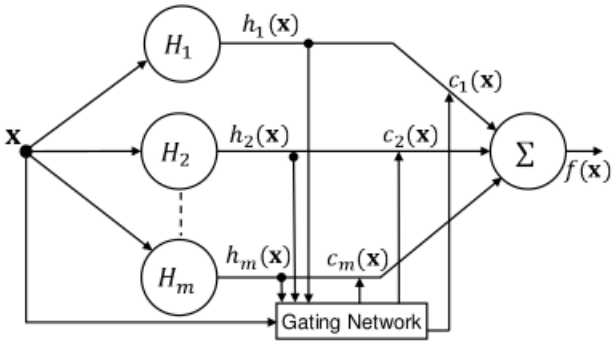


Figura 2.8: Modelo de Mistura Hierárquica de Especialistas, extraído de (Torres *et al.*, 2015c)

$$\begin{aligned} \mathbf{x}^T\mathbf{w}_i - b_i \geq 0, \\ \mathbf{x}^T\mathbf{w}_i - b_i < 0 \end{aligned} \tag{2.13}$$

sendo o *bias* b_i definido através de (2.14):

$$b_i = \left(\frac{x_i + x_j}{2} \right) \mathbf{w}_i^T \tag{2.14}$$

Cada ponto médio que forma o hiperplano local pode ser definido diretamente pelos vértices de $\mathcal{AS}$:

$$p_i = \left(\frac{x_i + x_j}{2} \right) \tag{2.15}$$

O parâmetro de ponderação c_i utilizado na *Gating Network* é definido pela Equação (2.16).

$$c_i(\mathbf{x}) = \exp - \left(\frac{(\max(\delta(\mathbf{x}, p_k)))^2}{\delta(\mathbf{x}, p_i)} \right) \quad \forall \quad k = 1, \ldots, m. \tag{2.16}$$

Onde posteriormente é feita uma normalização de forma que a soma dos valores de $c(\mathbf{x})$ seja igual a 1.

Por fim, é implementada a função de decisão $f(x)$ apresentada em (2.17):

$$f(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^{m} h_i(\mathbf{x}) c_i(\mathbf{x}) \right) \tag{2.17}$$

onde os pesos $c_i(\mathbf{x})$ são associados ao hiperplano $h_i(\mathbf{x})$ e apresentam valores maiores para hiperplanos mais próximos. A Figura 2.9 apresenta a região de separação gerada através do CHIP-clas.

2.3.4 CHIP-clas reduzido

O CHIP-clas reduzido apresenta uma proposta de simplificação da etapa de classificação do modelo, com o objetivo de tornar o classificador adequado para implementação em circuitos integrados. Ao contrário da utilização de um modelo de mistura hierárquica de especialistas, no CHIP-clas reduzido a classificação é realizada de acordo com o hiperplano mais próximo. Dado um padrão arbitrário $\mathbf{x}$ a ser classificado, busca-se a menor distância entre a amostra e o conjunto de pontos médios p, obtidos através das arestas de suporte, como segue:

$$\underset{i}{\text{argmin}}\, \delta(\mathbf{x}, p_i), \quad \forall \quad i = 1, \ldots, m. \tag{2.18}$$

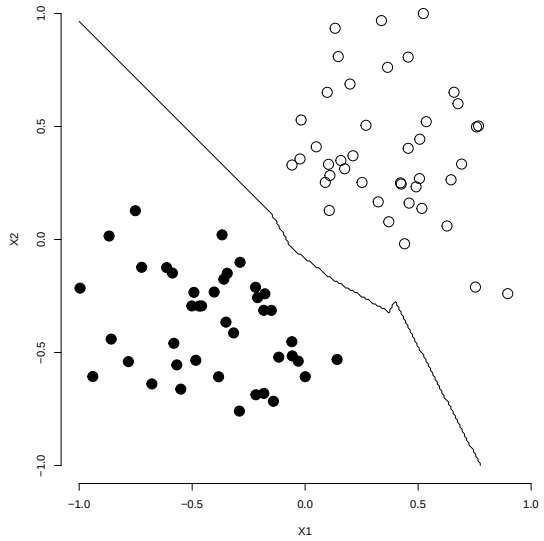


Figura 2.9: Superfície de separação gerada pelo CHIP-clas

Nesse contexto, o hiperplano de separação que passa pelo ponto médio é definido por $\mathcal{H}_i(\mathbf{x}) = (\mathbf{x}^T\mathbf{w}_i - b_i)$, sendo $b_i = [(1/2)(x_i + x_j)]\mathbf{w}_i$, onde (x_i, x_j) são os vetores geométricos que formam a aresta de suporte (dos Reis Gade, 2018). Dessa forma, para um problema de classificação binária definido pelas classes $(+1, -1)$, a amostra $\mathbf{x}$ é classificada da seguinte forma:

$$\begin{cases} +1, & \text{se} \quad \mathcal{H}_i(\mathbf{x}) > 0 \\ -1 & \text{caso contrário} \end{cases} \tag{2.19}$$

2.3.5 Custo Computacional

A análise de complexidade do algoritmo CHIP-clas pode ser feita pelo custo computacional relacionado à construção do Grafo de Gabriel. Segundo (Zhang & King, 2002), para a construção do Grafo através da técnica de força bruta, onde são calculadas as distâncias entre uma amostra e todas as demais, de forma que essa processo seja repetido em cada amostra de treinamento, a complexidade é de

$\mathcal{O}(dn^2)$ no melhor cenário, considerando que cada ponto é visitado apenas uma vez, e $\mathcal{O}(dn^3)$ no médio e pior caso, uma vez que para cada par de pontos (a, b), são necessárias $\mathcal{O}(dn)$ operações, onde n é a quantidade de amostras de treinamento e d o número de atributos dos dados. Entretanto, ao se utilizar a estrutura da Triangulação de Delaunay, essa complexidade pode cair para $\mathcal{O}(n \log n)$ (Matula & Sokal, 1980). No entanto, essa abordagem aumenta o custo de memória, visto a necessidade de armazenar a estrutura da Triangulação de Delaunay.

2.4 Plataformas de *Hardware*

No contexto de sistemas embarcados, várias plataformas de hardware oferecem alternativas para implementação de algoritmos de aprendizado de máquina, dentre as quais destacam-se GPGPUs (*General Purpose Computing on Graphics Processing Units*), FPGAs (*Field Programmable Gate Arrays*) e ASICs (*Application-Specific Integrated Circuits*) (Ovtcharov *et al.*, 2015).

2.4.1 ASIC

O desenvolvimento de novos processos para fabricação do silício e a consequente evolução de Circuitos Integrados (CI) levou ao advento, a partir de 1980, de sistemas mais robustos, com milhões de transistores em um único *chip*, denominados VLSI (*Very Large System Integration*). Nesse contexto, ASICs surgiram da necessidade de criação de circuitos integrados específicos para executar determinadas tarefas, os quais são geralmente uma combinação entre CIs de aplicação geral com alguns blocos projetados para aplicações específicas. *Full-Custom ASICs* por sua vez são uma classe de ASICs projetados inteiramente para uma nova função. Essa abordagem no entanto possui a tendência de ser mais demorada em relação ao tempo de acesso ao mercado (uma vez que os projetistas precisam redesenhar todo o circuito de forma a ser aplicado à tarefa especificada) e apresenta ainda um custo elevado, dado que não se beneficia do fator de produção em larga escala comum em aplicações mais generalistas (Smith, 1997).

2.4.2 FPGA

FPGAs são uma classe de dispositivos lógicos programáveis, desenvolvidos como evolução dos CPLDs (*Complex Programmable Logic Devices*), os quais possuíam limitação em termos de portas lógicas. FPGAs no entanto são adequados para aplicações com requisitos de alta performance (operações acima de 100 MHz por exemplo) e baixo consumo de energia. O fato de serem programáveis (através de linguagens de descrição de *hardware* como VHDL e *Verilog*) é uma vantagem em relação aos ASICs, garantindo um tempo menor de acesso ao mercado e maior flexibilidade em relação às tarefas à serem executadas. FPGAs usam o conceito de blocos lógicos *Configurable Logic Block* (CLB), o que propicia a execução de processos de forma paralela, aumentando assim a capacidade computacional do sistema (Wilson, 2007). A Figura 2.10 apresenta um exemplo de CLB da fabricante *Xilinx*® .

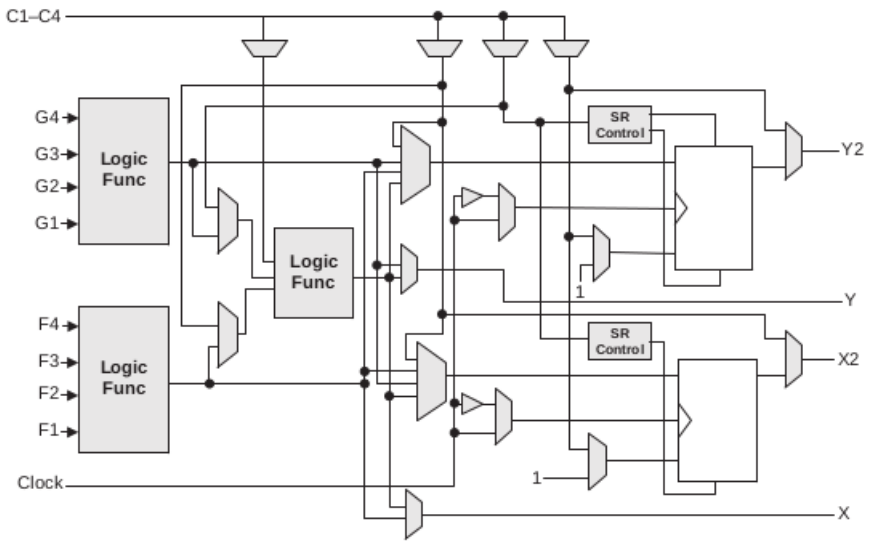


Figura 2.10: CLB da empresa *Xilinx*, extraído de (Wilson, 2007)

Segundo (de Oliveira *et al.*, 2011), o desenvolvimento de projetos em FPGA pode ser dividido da seguinte forma:

- Especificação e entrada de projeto
- Síntese e Mapeamento da tecnologia

- Posicionamento e roteamento
- Verificação e teste
- Programação do FPGA

Devido à grande complexidade dos projeto atuais, o mapeamento é facilitado através do uso de linguagens de descrição de *hardware* como VHDL e Verilog.

2.4.2.1 Restrições de Projetos em FPGA

A otimização de projetos em FPGA busca atender a requerimentos relacionados à melhor utilização dos recursos disponíveis. Segundo (Cofer & Harding, 2006), as principais restrições em projetos de FPGA são:

1. Síntese: restrições relacionadas ao processo de transformação entre as linguagens de descrição de *hardware* (VHDL e Verilog) e sua conversão à nível RTL (*Register-transfer level*);
2. Atribuição de pinos: relacionada à atribuição dos sinais aos pinos de entrada e saída;
3. Tempo: relacionada às características de frequência do projeto, atrasos associados ao sincronismo entre *flip-flops* e registradores;
4. Área: restrição associada ao mapeamento dos diferentes blocos do projeto e alocação dos recursos disponíveis.

Outros itens de restrições podem ser ainda avaliados, como o consumo de energia por exemplo. No caso da representação numérica, essa escolha está implicitamente associada à restrição de área do projeto, visto que a utilização dos recursos de memória está diretamente relacionada ao tamanho da precisão numérica escolhida.

2.5 Representação Numérica

Em projetos de circuitos digitais, a escolha da forma de representação numérica apresenta comumente um compromisso entre eficiência e acurácia (Inacio & Ombres, 1996). Geralmente essa escolha é definida entre representação em ponto-fixo e ponto-flutuante. Embora a representação em ponto-fixo apresente a vantagem de realizar operações aritméticas de forma mais eficiente, o seu baixo alcance de representação e a necessidade de normalização podem inviabilizar sua implementação em determinadas aplicações. A seguir são abordadas as formas de representação em ponto-fixo e ponto-flutuante.

2.5.1 Representação em Ponto-fixo

Baseia-se na representação onde a posição do ponto decimal ou binário é pré-determinada. Dessa forma, existe uma limitação na extensão de números que podem ser representados. Para a representação de números inteiros positivos, a extensão de representação varia de $0 \leq \alpha < 2^n$, sendo n o número de *bits*. Valores comuns para esse tipo de representação variam entre $8, 16, 32$ e 64 *bits*

Para a representação de números com sinal, o bit mais significativo (MSB), representa um número positivo caso seu valor seja o valor binário 0 e um número negativo caso seja 1 . No entanto, o valor da magnitude (valor absoluto do número) pode mudar dependendo da forma de representação. As principais formas de representação são:

1. Sinal-magnitude: onde o MSB representa o sinal e os demais bits a magnitude;

2. Representação com complemento de 1: onde, caso o MSB seja 1 (valor negativo), todos os demais $n-1$ bits são trocados pelo seu complemento;

3. Representação em complemento de 2: neste caso,dado que o MSB seja 1, os demais $n-1$ são substituídos pelo seu complemento e adicionado o valor 1_2.

Nos casos de Sinal-magnitude e Representação em complemento de 1, existe um problema associado à dupla representação do valor 0, o qual possui duas representações (positivo e negativo). Outro ponto em questão é a necessidade de se tratar números positivos e negativos de forma diferente. A representação em complemento de 2 soluciona esses problemas, de forma que é mais adequada para implementação de operações aritméticas binárias (adição, subtração, etc), sendo comumente usada em arquiteturas de processadores. A extensão de representação em Complemento de 2 é $(-2^n \leq \alpha \leq 2^n - 1)$.

2.5.2 Representação em Ponto-flutuante

Ao contrário da representação em ponto-fixo, a representação em ponto-flutuante apresenta números reais em notação científica, o que permite uma maior extensão de representação numérica. Neste caso, os bits são divididos em três componentes: sinal (MSB), expoente e significando (mantissa).

2.5.2.1 Forma Normalizada

Na representação normalizada, existe um valor implícito de 1 na parte fracionária (significando). Existe ainda um valor de deslocamento do expoente, denominado *bias*, com o objetivo de representar números positivos e negativos no expoente. Dessa forma, temos a representação em ponto flutuante na forma normalizada representada em (2.20):

$$\alpha = (-1)^s \times 1.F^{(E-bias)} \tag{2.20}$$

onde s representa o MSB referente ao sinal, F a parte fracionária (significando) e E o expoente.

2.5.2.2 Forma Não-Normalizada

Um dos problemas da representação normalizada é a impossibilidade de representação do valor decimal 0, uma vez que existe o valor implícito na parte fracionária. A representação Não-Normalizada apresenta uma solução para este

caso, de forma que, quando o expoente for 0, o valor implícito na parte fracionária é substituído por 0 e o valor do expoente é substituído, de forma que seja possível além da representação do valor decimal 0, outros valores positivos e negativos próximos de zero.

2.5.3 Padrão IEEE 754

Lançado originalmente em 1985 pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) e revisado em 2008 (Zuras *et al.*, 2008), o documento visa padronizar a utilização de ponto-flutuante em ambiente de *hardware*, *software* ou a combinação de ambos. As propostas de padronização apresentadas incluem ainda, dentre outros pontos, os formatos de representação (representação de valores especiais, como *infinito* e *NaN* (*not a number*)), regras de arrendondamento, operações aritméticas em ponto-flutuante e diretrizes para tratamento de exceções.

A proposta de padronização divide as representações em ponto-flutuante em níveis de precisão e tamanho de *bits*, sendo os mais comuns: *half-precision* com 16 *bits*, *single-precision* com tamanho de 32 *bits* e *double-precision* com 64 *bits*. A Tabela 2.1 apresenta diferentes tamanhos de representação numérica e a divisão dos valores de expoente e significando, assim como o valor de deslocamento do expoente (*bias*).

Tabela 2.1: Diferentes formas de representação em Ponto-Flutuante IEEE 754

Precisão	Expoente	Significando	*Bias*
16 bits	5	10	15
32 bits	8	23	127
64 bits	11	52	1023

2.6 NN-clas

Com o objetivo de simplificar o algoritmo do CHIP-clas de maneira que o mesmo seja adequado para implementação em sistemas embarcados com restrições de

recursos de processamento e memória, é apresenta em (dos Reis Gade *et al.*, 2017) uma nova abordagem para a etapa de classificação do modelo. O novo método, denominado NN-clas — como uma referência ao algoritmo *K-Nearest Neighbors* (K-NN) (Cover *et al.*, 1967)— é baseado na distância entre um novo dado na fase de testes e os dados que fazem parte do conjunto de arestas de suporte $\mathcal{AS}$, de forma que esse dado recebe a classe da amostra de menor distância.

Testes realizados em 15 bases de dados do repositório da UCI e ainda em 4 bases sintéticas mostraram que, embora o método não possa ser considerado como de margem máxima, seus resultados são estatisticamente equivalentes ao CHIP-clas. Contudo, o NN-clas é mais simples, uma vez que não utiliza o método de mistura hierárquica de especialistas, facilitando assim sua implementação em *hardware*. O pseudocódigo abaixo apresenta o algoritmo de classificação proposto no método NN-clas:

Algorithm 1 Classificação NN-clas

input: Conjunto de arestas de suporte $\mathcal{AS}$, classes y_v e o conjunto de amostras de teste $\mathbf{X}_t$
output: O valor estimado $\hat{y}$ das amostras de teste $\mathbf{X}_t$
for i in $\mathbf{X}_t$ **do**
    for j in $\mathcal{AS}$ **do**
        Calcula a distância entre as amostras de treinamento e os vértices de $\mathcal{AS}$
        $dist(i) \leftarrow \delta(\mathbf{X}_t(i), \mathcal{AS}(j))$
    end for
    $\hat{y} \leftarrow$ rótulo da amostra de menor distância em $\mathcal{AS}$
end for
return $\hat{y}$

2.7 Arquitetura de *Hardware* do NN-clas

A implementação do NN-clas em *Hardware* apresentada em (dos Reis Gade, 2018) foi desenvolvida através de uma arquitetura em *pipeline*, onde os blocos foram divididos de acordo com cada estágio do classificador: cálculo da distância entre

as amostras, construção do GG, obtenção das arestas de suporte e por fim, a classificação da amostra de teste utilizando a classe da amostra mais próxima pertencente ao conjunto de $\mathcal{AS}$. Vale ressaltar a interdependência de cada estágio, sendo a saída do estágio anterior a entrada para o estágio seguinte.

Os blocos foram implementados em FPGA, explorando as características de paralelismo intrínsecas desta plataforma. A precisão numérica utilizada foi de 32 *bits* em ponto-flutuante, no padrão IEEE 754. Este valor de precisão simples foi escolhido de forma arbitrária e apresentou resultados semelhantes — em termos de desempenho do modelo — quando comparados com os resultados obtidos através da implementação em *software*.

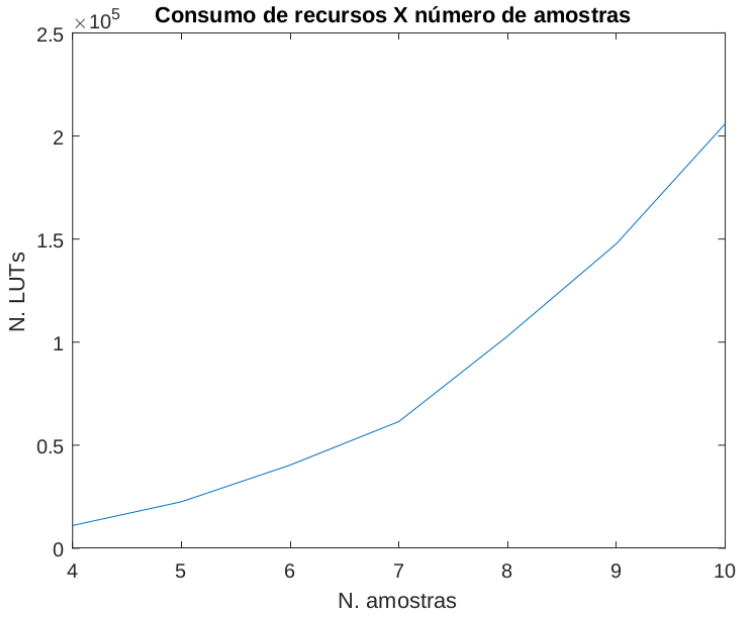
Entretanto, os resultados associados à eficiência da implementação mostraram um crescimento exponencial do consumo de recursos, em termos de *Look-up Tables* (LUT) e blocos de processamento digital de sinais (DSPs), em relação ao número de amostras, como mostrado na Figura 2.11a e um crescimento linear em relação ao número de dimensões, conforme Figura 2.11b.

Uma vez que a metodologia apresentada no capítulo 3, utiliza como base de desenvolvimento a arquitetura proposta no trabalho de (dos Reis Gade, 2018), aqui será apresentada uma análise mais detalhada desta arquitetura.

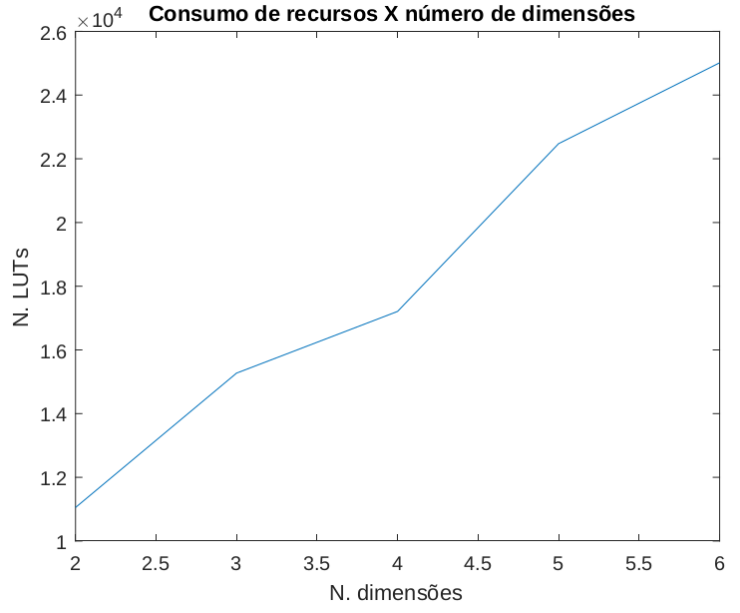
2.7.1 Cálculo da distância

Nesta subetapa foram avaliadas em (dos Reis Gade, 2018) duas métricas para o cálculo da distância: distância Euclidiana quadrática e Distância de *Manhattan*. As duas métricas apresentaram resultados semelhantes em termos de precisão e desempenho, de forma que não houve uma redução significativa do consumo de recursos, apesar da Distância de *Manhattan* não necessitar de blocos multiplicadores como ocorre no cálculo da Distância Euclidiana quadrática.

Foi desenvolvido um bloco denominado *fifo*, o qual representa um elemento de memória local em estrutura de fila, responsável pela leitura dos dados de entrada e fornecimento de um novo dado a cada ciclo de *clock*. Para sincronizar o fluxo de dados para o correto cálculo das métricas de distância foram utilizados blocos multiplexadores e atrasadores.



(a)



(b)

Figura 2.11: Consumo de recursos em relação ao número de amostras e dimensões (adaptado de (dos Reis Gade, 2018))

2.7.2 Cálculo do grafo de proximidade

Esta subetapa é responsável pela criação da matriz de adjacência na etapa de treinamento. A cada ciclo de *clock*, com uma latência de $3 + 3\,[\log_2 m] + 2$ ciclos de *clock* – onde m representa o número de dimensões da amostra – é fornecida a distância Euclidiana quadrática entre duas amostras. Dessa forma, as distâncias são comparadas a fim de verificar se existe a formação de uma aresta entre os dois vértices, através de (2.2). Os primeiros blocos desta subetapa são multiplicadores uma vez que a comparação é feita sobre a distância ao quadrado. Novamente são usados blocos multiplexadores e atrasadores para sincronizar a comparação entre as amostras de entrada.

Após isso, são utilizados blocos somadores responsáveis por realizar a soma em pares das distâncias ao quadrado. Por fim, são utilizados blocos comparadores, através da implementação da operação lógica *AND*, sendo a saída o valor binário referente à formação ou não de uma aresta entre as duas amostras comparadas. A Figura 2.12 apresenta a arquitetura do grafo de proximidade proposto por (dos Reis Gade, 2018), para 4 amostras de treinamento.

2.7.3 Cálculo e armazenamento do grafo de borda

Esta subetapa é a última da fase de treinamento do modelo e consiste na obtenção dos dados referentes ao conjunto $\mathcal{AS}$. Para tanto, é necessário verificar se as amostras que possuem uma aresta entre si são também de classes distintas. Dessa forma, foi implementado um bloco denominado *fifo label* o qual representa um elemento de memória local com a informação das classes às quais as amostras de treinamento pertencem. Foram utilizadas as operações lógicas *AND*, *OR* e *XOR*, como mostra a Figura 2.13, para 4 amostras de treinamento, onde L_i representa o rótulo da i-ésima amostra.

A porta lógica *XOR* retorna o valor binário 1 caso os rótulos sejam diferentes, sendo esta uma condição necessária para a formação de uma aresta de suporte. Após isso, a porta lógica *AND* recebe como entrada a saída da porta lógica *XOR* e a informação da matriz de adjacência $G_{i,j}$, sobre a existência de uma aresta entre os dados i e j. Caso ambos sejam iguais ao valor binário 1, isso indica que as amostras i e j pertencem a $\mathcal{AS}$.

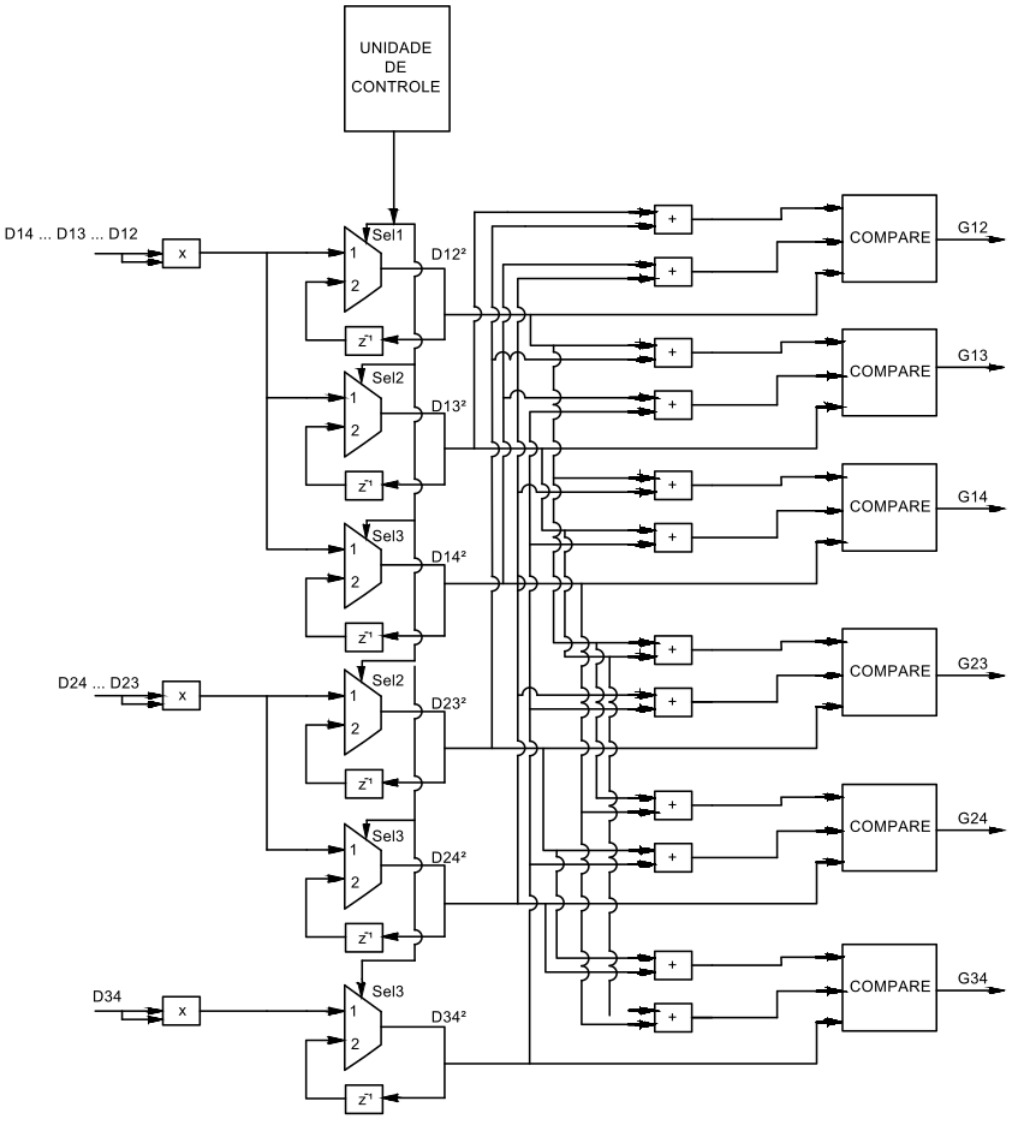


Figura 2.12: Arquitetura do grafo de proximidade (dos Reis Gade, 2018)

Os dados pertencentes às arestas de suporte são armazenados em um elemento de memória local. As entradas deste estágio são: os resultados do cálculo do grafo de borda, o rótulo das amostras de treinamento e suas respectivas dimensões. A Figura 2.14 apresenta a arquitetura para o armazenamento das arestas de suporte, proposta por (dos Reis Gade, 2018). Os blocos que recebem como entrada os rótulos e a informação se a amostra pertence ao conjunto $\mathcal{AS}$ armazenam os valores de entrada quando o sinal de controle *enable* estiver ativo.

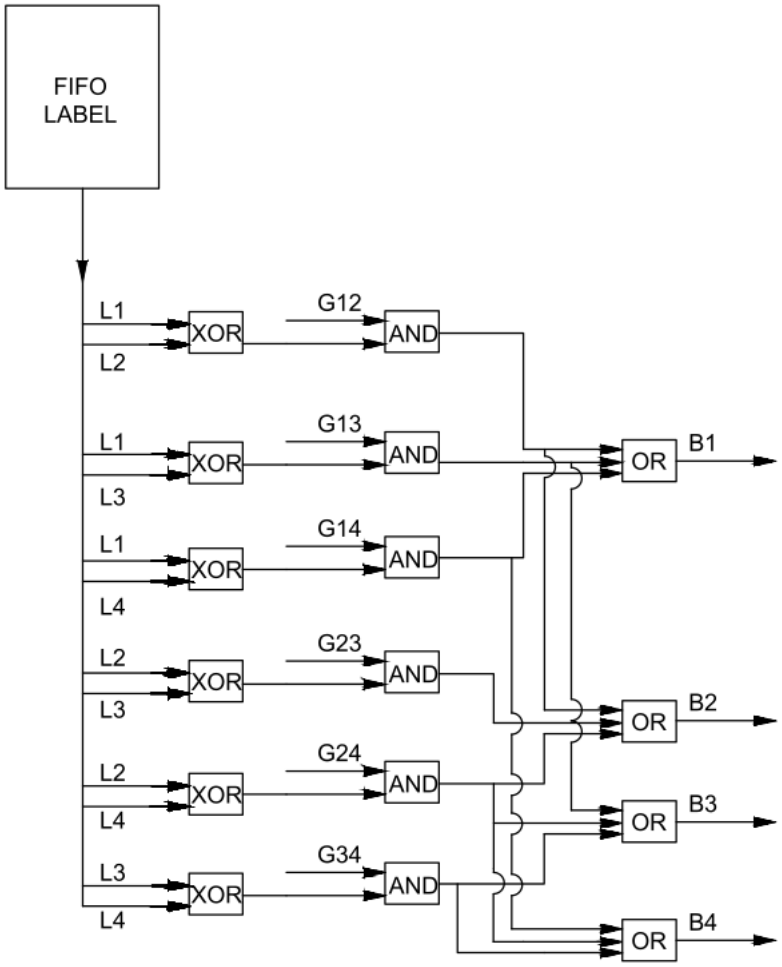


Figura 2.13: Arquitetura do grafo de borda (dos Reis Gade, 2018)

2.7.4 Etapa de classificação

Na fase de classificação, foi implementado o algoritmo do NN-clas (dos Reis Gade *et al.*, 2017), onde a classificação de um novo dado é feita através da distância entre as amostras de teste e os dados que foram definidos como pertencentes ao conjunto das arestas de suporte, sendo uma amostra arbitrária de teste $\mathbf{x}_t$ classificada com o rótulo do dado de menor distância pertencente a $\mathcal{AS}$. Para isso, a arquitetura implementada por (dos Reis Gade, 2018) utiliza o Elemento de Processamento (EP) semelhante ao cálculo de distância da etapa de treinamento, mas ao contrário desta etapa, os dados de entrada são as amostras pertencentes

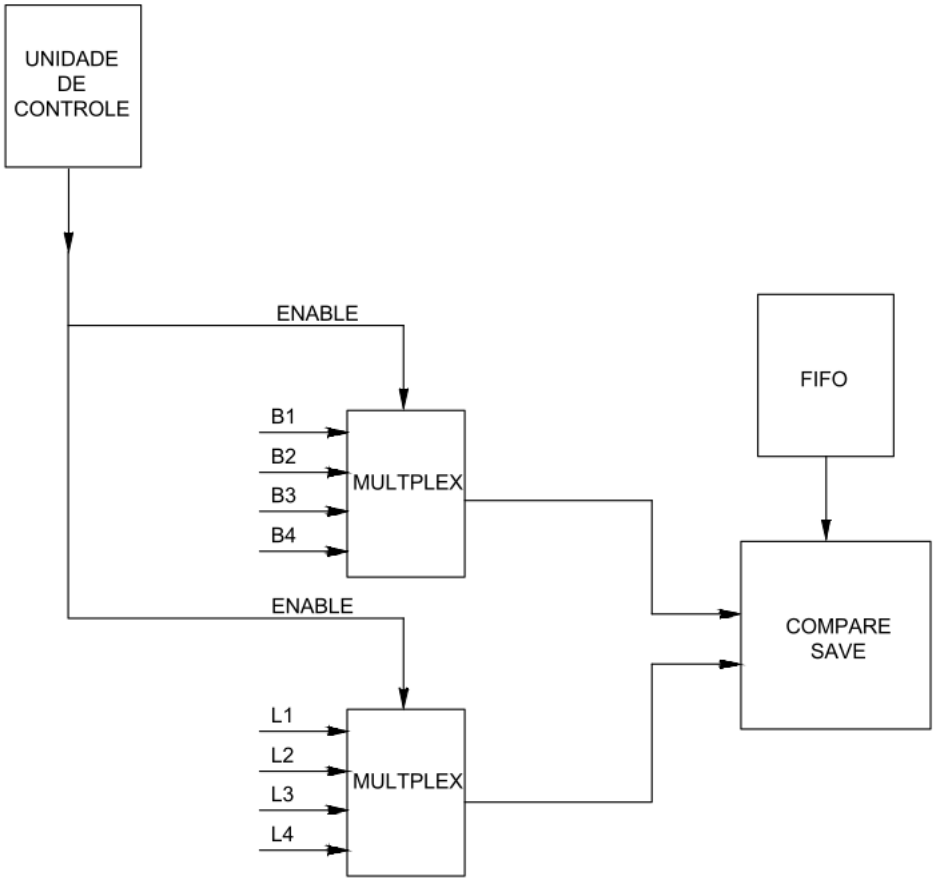


Figura 2.14: Armazenamento de dados localizados na borda de separação (dos Reis Gade, 2018)

a $\mathcal{AS}$, conforme apresentado na Figura 2.15.

Por fim, a partir das distâncias fornecidas pelo EP anterior, a última subetapa da fase de classificação consiste em comparar os valores de distâncias, de forma a obter o menor valor. A arquitetura desta subetapa, apresentada na Figura 2.16, possui como entrada os rótulos dos dados e a matriz de distância entre o novo dado a ser classificado e os dados pertencentes a $\mathcal{AS}$

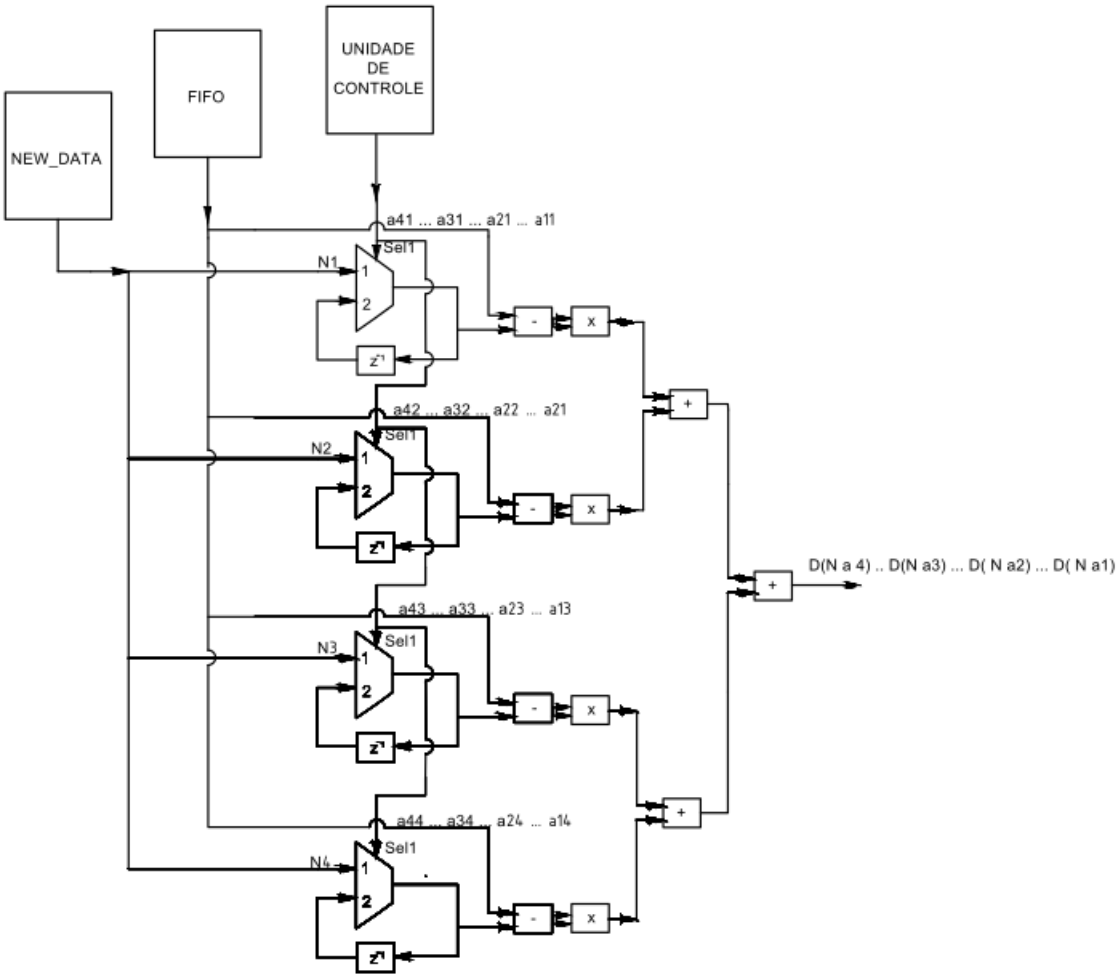


Figura 2.15: Cálculo da distância na etapa de treinamento (dos Reis Gade, 2018)

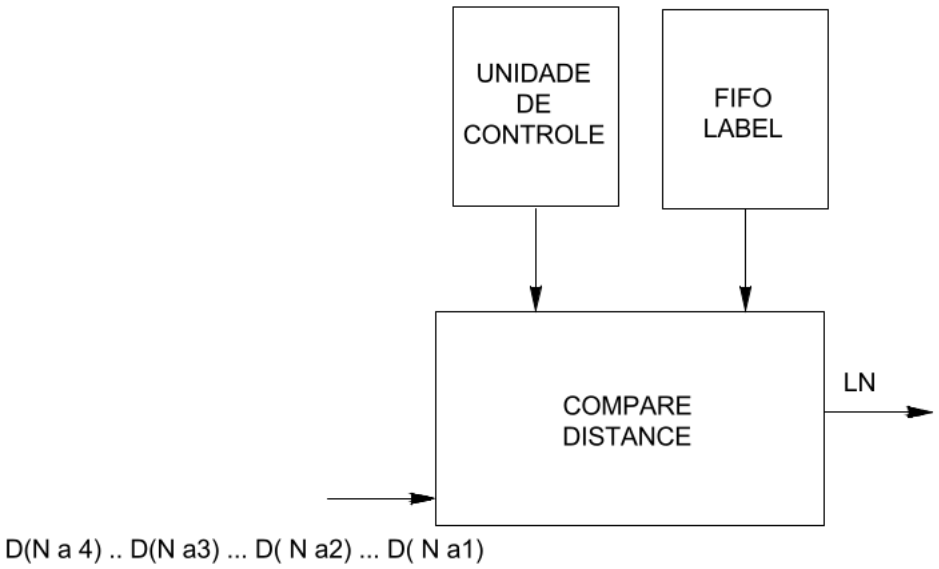


Figura 2.16: Comparação dos valores de distância (dos Reis Gade, 2018)

Capítulo 3

Proposta de aumento da eficiência através da redução de precisão numérica

Dado o impacto da representação numérica no contexto de implementação de algoritmos de aprendizagem de máquina em sistemas embarcados, busca-se através dessa proposta, avaliar e quantificar o compromisso entre o desempenho do modelo e a precisão da representação numérica no classificador NN-clas, uma vez que este é um modelo da família de classificadores por aresta de suporte e apresentou desempenho estatisticamente equivalente ao CHIP-clas (dos Reis Gade *et al.*, 2017).

Entretanto, a implementação da arquitetura de *hardware* do NN-clas, descrita na seção 2.7, apresentou pontos de melhoria em termos da utilização de recursos associada à quantidade de amostras das bases de dados. Essa arquitetura possui ainda a vantagem da reutilização do EPs de cálculo de distância tanto na fase de treinamento quando na fase de inferência. Dessa forma, o NN-clas, implementado com precisão de 32 *bits* em *hardware* e 64 *bits* em software, serve como estado da arte para comparação do desempenho dos modelos com redução de precisão numérica.

3.1 Trabalhos Relacionados

Diferentes formas de representação numérica têm sido exploradas na literatura em termos de implementação de algoritmos de aprendizado de máquina em *hardware*. Em (Anguita *et al.*, 2003) é proposta uma arquitetura com representação em ponto-fixo, para implementação do algoritmo SVM em FPGA, analisando os efeitos de quantização em relação aos erros de classificação. Já em (Lin *et al.*, 2016) é proposto um processo de quantização para converter Redes Neurais Convolucionais (CNN) que foram implementadas em ponto-flutuante para ponto-fixo.

Em (Courbariaux *et al.*, 2014) é realizada uma comparação da implementação de uma rede neural de camada profunda usando representação em ponto-flutuante, ponto-fixo e ponto-fixo dinâmico. Os autores concluíram que a representação de precisão baixa é suficiente para a obtenção de bons resultados nas etapas de treinamento e teste dos modelos avaliados. A literatura apresenta ainda alternativas ao padrão IEEE 754, como por exemplo no trabalho de (Johnson, 2018), onde é proposta uma nova abordagem para computação em ponto-flutuante, com o objetivo de aumentar a eficiência em termos de processamento e consumo de energia.

3.2 Definição da representação numérica

Com o objetivo de avaliar o compromisso entre a redução da precisão numérica e o desempenho do modelo, foi definida a precisão de 16 *bits* em ponto-flutuante, de forma a comparar os resultados com o modelo implementado em precisão simples de 32 *bits*. A representação em ponto-flutuante foi escolhida em detrimento de ponto-fixo, pela sua maior extensão de representação e ainda devido ao fato de não ser necessário realizar uma normalização para executar as operações como no caso da representação em ponto-fixo.

As vantagens da representação em ponto-flutuante de 16 *bits* (*half-precision*) no padrão IEEE 754 estão associadas à uma menor área (menor demanda de memória), maior velocidade de processamento e menor consumo de energia. Entretanto, essa representação possui uma menor extensão de representação numérica,

maior erro numérico devido ao arredondamento e ainda introduz erros de quantização, em comparação à implementação em 32 *bits*. Nesse contexto, com o objetivo de superar algumas dessas desvantagens, aplicações recentes buscam alternativas para a representação de 16 *bits* em ponto-flutuante além da implementação no padrão IEEE 754.

Dentre tais alternativas, destaca-se o formato denominado *bfloat*16 (*Brain Floating Point*), que vem sendo utilizado de maneira crescente em arquiteturas como a TPU (Google), nos *chips Nervana* NNP-L1000, processadores *Xeon* e FPGAs da emrpesa Intel® e ainda na biblioteca de processamento distribuído de algoritmos de aprendizado de máquina *TensorFlow* (Girija, 2016).

As vantagens da utilização do formato *bfloat* em relação ao padrão IEEE estão associadas à maior extensão numérica, de forma que este formato busca explorar a extensão do alcance da representação em 32 *bits* em precisão simples, uma vez que utiliza o mesmo número de *bits* no expoente. Entretanto a mantissa é truncada em 7 *bits*, ao contrário de 10 *bits* no formato IEEE *half-precision* (FP16) e 23 *bits* no formato de precisão simples. Resultados na literatura mostraram o potencial do uso do formato *bfloat* no treinamento de algoritmos de aprendizado de máquina (Kalamkar *et al.*, 2019). A Figura 3.1 apresenta uma comparação entre a distribuição de *bits* nos diferentes formatos de representação numérica.

Figura 3.1: Comparação dos formatos de representação em ponto-flutuante, extraído de (Intel, 2018)

Dessa forma, na etapa de implementação em *hardware* é avaliada a utilização dos formatos de representação numérica em ponto-flutuante *bfloat16* (BF16) e a representação no padrão IEEE 754 em ponto-flutuante de 16 *bits* (FP16).

3.3 Implementação em *software*

De forma a validar a viabilidade da redução da precisão numérica do modelo NN-clas, é proposta inicialmente sua implementação em *software*, a fim de avaliar se a redução de precisão implica em uma perda direta de desempenho do classificador. Para isso, foi escolhida a linguagem de programação *python*,uma vez que a mesma apresenta diversas ferramentas de manipulação de operações matriciais e ainda APIs de visualização e processamento de dados no contexto de implementação de algoritmos de aprendizado de máquina.

Para implementação do NN-clas, foi realizada a etapa de remoção de ruídos descrita na seção 2.3.2. A métrica de distância escolhida foi a Euclidiana Quadrática. Foram criadas funções para cada estágio do classificador, sendo a precisão numérica gerenciada através da biblioteca *numpy* (Ascher *et al.*, 2001). Dessa forma, o NN-clas foi implementado com redução de precisão numérica para 16 *bits* no padrão IEEE 754 e comparado ao modelo original sem alteração na representação numérica (64 *bits*).

3.4 Implementação em *hardware*

Para a implementação em *hardware*, foi escolhida a plataforma FPGA, dada suas características intrínsecas de processamento em paralelo e menor consumo de energia em comparação com outras plataformas, no contexto de aplicações em sistemas embarcados. A implementação se deu através da linguagem de descrição de *hardware* VHDL. Utilizou-se como base de implementação a arquitetura em *pipeline* do NN-clas proposta por (dos Reis Gade, 2018).

O fluxo de dados do NN-clas é mostrado na Figura 3.2. Os elementos de processamento (EPs) de cada subetapa foram alterados para realizar operações com precisão reduzida de 16 *bits*, nos formatos de representação FP16 (padrão IEEE 754) e BF16 (*bfloat*) e posteriormente comparados à implementação em representação simples de 32 *bits* também no padrão IEEE (FP32). Dessa forma, foi realizada a simulação de cada subetapa do classificador através do *software* Modelsim®.

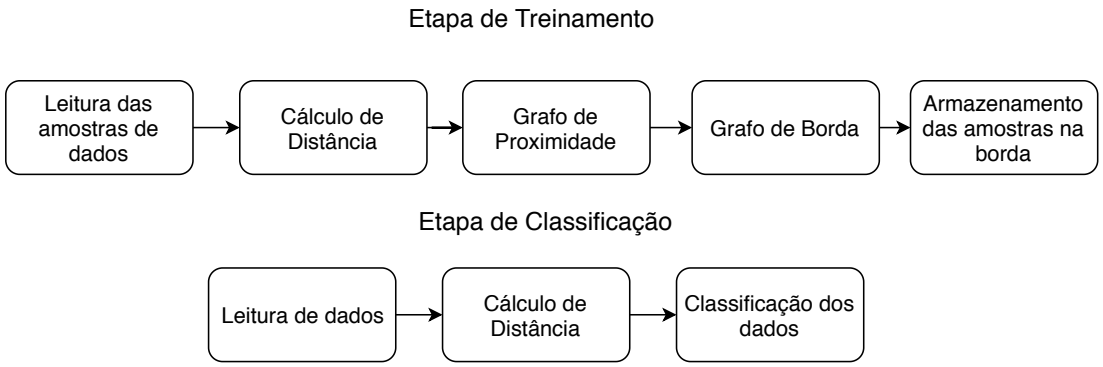


Figura 3.2: Fluxo de dados da arquitetura no NN-clas, adaptado de (dos Reis Gade, 2018)

3.4.1 Conversão decimal para ponto-flutuante

Para avaliar o desempenho do modelo com redução de precisão numérica, é necessário, para cada subetapa da implementação, a conversão dos valores em ponto-flutuante para decimal. Para isso, foi desenvolvido um algoritmo em Matlab®, o qual implementa a conversão definida em (2.20), a partir da entrada de um conjunto de dados em formato numérico ou *string*. Esse algoritmo foi ainda adaptado para realizar a conversão entre números decimais e binários utilizando a representação BF16.

3.4.2 Avaliação da redução de área

As Figuras 2.13, 2.14, 2.15, 2.16, as quais representam os diagramas das subetapas da construção do NN-clas, possuem em comum um elemento de memória local, seja para armazenamento das informações de dados de treinamento e teste assim como para armazenamento dos rótulos das amostras de treinamento. A proposta de redução da representação em ponto-flutuante indica uma queda significativa na demanda por memória e consequentemente na área do circuito. Na representação em 32 *bits*, cada elemento de memória possui o tamanho de $(32 \times n \times d)$, onde n representa a quantidade de amostras dos dados e d a dimensão. Já no caso da implementação em 16 *bits*, esse tamanho cai pela metade, sendo $(16 \times n \times d)$.

Capítulo 4

Proposta de Computação Paralela

Dado o crescimento, nos últimos anos, da disponibilidade de dados, busca-se otimizar algoritmos de aprendizado de máquina a fim de minimizar o tempo de processamento nas etapas de treinamento e inferência, de forma que possam ser executados com eficiência em aplicações de maior escala.

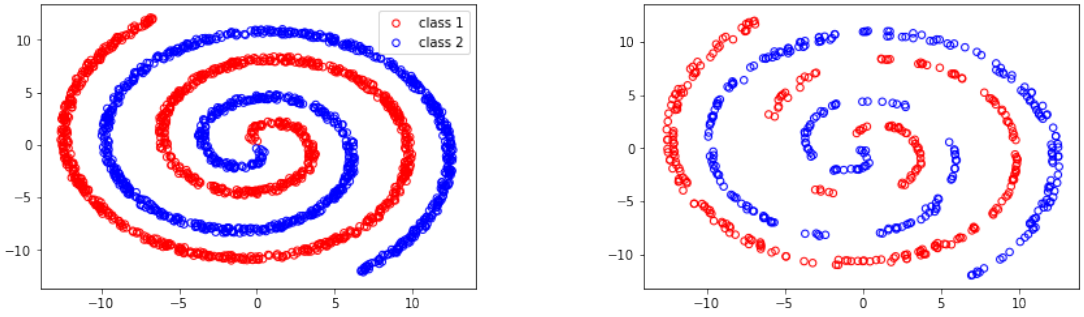
4.1 Análise de desempenho do CHIP-clas

Conforme apresentado na seção 2.3.5, a complexidade da construção do GG na etapa de treinamento do CHIP-clas é de $\mathcal{O}(dn^3)$ no médio e pior caso considerando a construção do grafo a partir do cálculo de distância entre as amostras de treinamento, onde n representa a quantidade de amostras dos dados e d o número de dimensões. Já, na etapa de classificação o custo pode ser de $\mathcal{O}(dn)$, através da utilização do método de classificação do NN-clas, onde n neste caso é a quantidade de amostras no conjunto $\mathcal{AS}$. Neste sentido, o custo computacional do modelo sofre bastante influência da etapa de treinamento, uma vez que o tempo de processamento cresce à medida que o número de amostras na fase de treinamento também cresce, com relação polinomial.

A construção do GG possui uma característica sequencial, dada a necessidade de computar a distância entre uma amostra arbitrária e as demais amostras da base de treinamento. Considerando a etapa de treinamento, busca-se definir

quais dados (vértices) fazem parte do conjunto de $\mathcal{AS}$. Com isso, tem-se uma busca por um conjunto comumente reduzido de dados ($\mathcal{AS}$), mas que requer a avaliação de todos os demais dados de treinamento. Portanto, faz-se necessária a implementação de novas estratégias, com o objetivo de reduzir o espaço de busca, de maneira que as arestas de suporte possam ser encontradas de forma mais eficiente.

A Figura 4.1 apresenta um exemplo de redução do espaço de busca do conjunto de treinamento na base sintética *"espirais"*. Pode-se observar que o espaço é reduzido à medida que se eliminam amostras que estão longe da borda de separação das classes, as quais apresentam menor probabilidade de pertencerem ao conjunto de $\mathcal{AS}$.



(a) Dados de treinamento (b) Espaço de busca reduzido

Figura 4.1: Exemplo sintético da redução do espaço de busca

4.2 Terminologia

As terminologias mais utilizadas em relação à computação de processos são: concorrente, paralela e distribuída. Computação concorrente refere-se à utilização de múltiplas *threads* por um mesmo processador. Computação paralela, por sua vez, refere-se à exploração de mais de um núcleo de processamento, buscando executar tarefas da maneira mais rápida possível. Já computação distribuída executa processos em diferentes máquinas, geralmente conectadas entre si (Prasad *et al.*, 2015). No contexto deste trabalho, será usada a terminologia de computação paralela para execução de processos em diferentes núcleos de processamento.

4.3 Trabalhos Relacionados

4.3.1 *Parallel Cascade* SVM

Por se tratar de um problema de programação quadrática (QP), o custo computacional do algoritmo SVM tende a crescer de forma polinomial com o crescimento do número de amostras de treinamento ($\mathcal{O}(n^3)$). Dessa forma, em (Graf *et al.*, 2005), é proposto um algoritmo que implementa técnicas de computação paralela, através de divisões sucessivas dos dados com o objetivo de processar otimizações locais e distribuir o processamento da implementação do classificador SVM em uma arquitetura em cascata. Esta arquitetura apresentada na Figura 4.2, mostra o processo de divisões dos dados de treinamento (TD) e consequente obtenção de vetores de suporte (SV) em cada janela de dados.

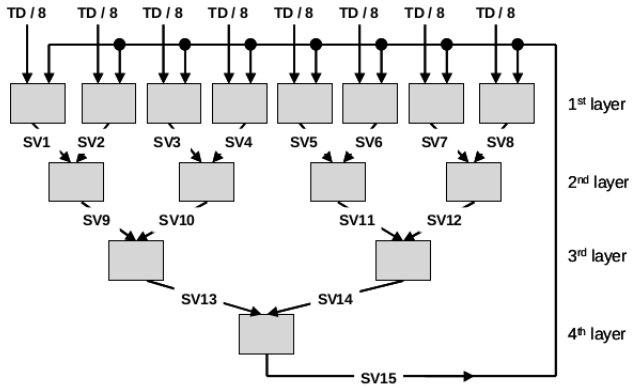


Figura 4.2: Arquitetura do SVM em cascata (Graf *et al.*, 2005)

Os testes realizados nesta arquitetura, em bases de dados com grandes quantidades de amostras (*mnist*, *forest* e NORB), mostraram um redução significativa do tempo de execução, em comparação com o algoritmo tradicional, além de obter um bom resultado de acurácia e menor consumo de memória.

4.3.2 CHIP-clas Incremental

Em (de Freitas Diadelmo, 2016), são apresentadas quatro propostas para implementação de aprendizado incremental com memória parcial, a partir do CHIP-clas. É apresentado um estudo da construção do GG na etapa de treinamento

através de janelas temporais de dados, de maneira iterativa, sendo que o grafo é atualizado a partir da chegada de novos dados.

4.4 Computação paralela do CHIP-clas

A partir da combinação entre a abordagem de computação paralela do SVM apresentada em (Graf *et al.*, 2005) e a proposta de utilização de janelas de dados temporais apresentada em (de Freitas Diadelmo, 2016), esta seção apresenta a proposta de uma metodologia para construção do GG de forma paralela, com o objetivo de explorar aplicações em arquiteturas com múltiplos núcleos de processamento.

Uma vez que a otimização da etapa de classificação do CHIP-clas já fora abordada em trabalhos anteriores (CHIP-clas reduzido e NN-clas), busca-se neste trabalho uma avaliação da etapa de treinamento. O principal desafio é a validação da possibilidade da construção do GG de forma paralela, uma vez que o mesmo possui características de computação sequencial, dado que é necessário saber a distância entre uma amostra e os demais dados para que se possa definir a existência ou não de uma aresta entre dois vértices.

Dessa forma, essa metodologia pretende diminuir o espaço de busca a ser computado por cada núcleo de processamento, através da criação de janelas de dados, de forma que as amostras de treinamento sejam divididas uniformemente, e processadas de forma paralela. Nesse sentido, a alocação dos dados nas janelas deve ser feita de forma aleatória, com o objetivo de não interferir no processo de obtenção das $\mathcal{AS}$.

Para essa construção, foi criado o conceito do conjunto de arestas de suporte locais e globais. O conjunto local faz referência às $\mathcal{AS}$ obtidas a partir da construção do GG em cada janela de dados. No entanto, vale ressaltar que tais amostras são apenas candidatas ao conjunto de arestas de suporte, uma vez que, em um contexto global, podem estar localizadas distantes da borda de separação. Essa etapa pode ser encarada como um filtro, onde os dados que não possuem informações relevantes para a classificação são removidos. Por fim, os conjuntos de arestas de suporte locais são unidos para a construção de um nova base de

dados de onde o GG será novamente construído e as $\mathcal{AS}$ globais serão obtidas. A Figura 4.3 apresenta um resumo da metodologia proposta.

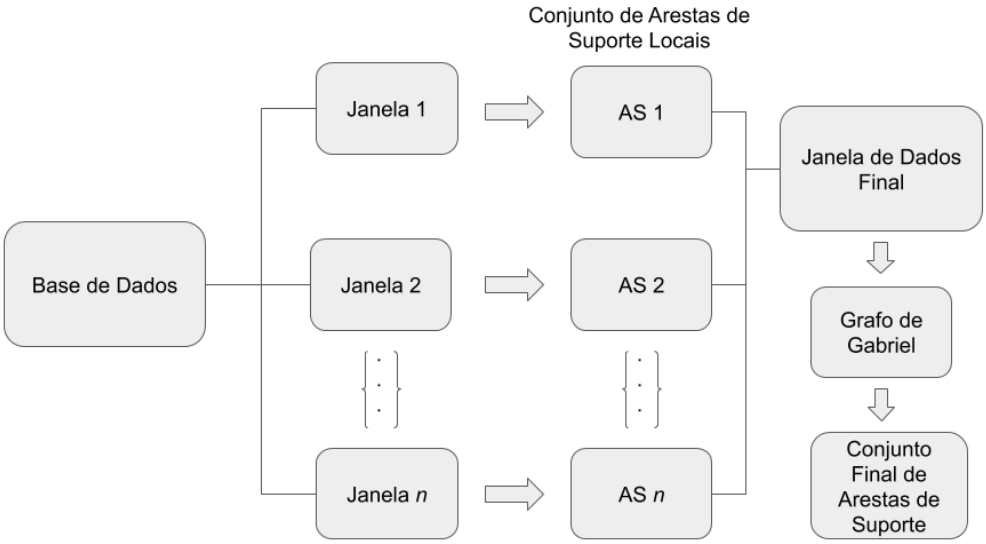


Figura 4.3: Esquema de treinamento do CHIP-clas de forma paralela

O Algoritmo 2 apresenta o método para construção do Grafo de Gabriel de forma paralela. A Figura 4.4 apresenta ainda a construção de blocos de processamento, onde a Figura 4.4a apresenta uma base de dados sintética com 200 amostras de treinamento, e as demais figuras apresentam as janelas aleatórias de dados, onde, a partir de cada janela será obtido um conjunto de arestas de suporte local. O modelo foi implementado em *python*, utilizando o módulo *concurrent futures* o qual permite a execução do algoritmo de forma paralela. Novamente é realizada a etapa de filtragem dos dados ruidosos antes da etapa de treinamento, sendo a métrica de distância para construção do GG a distância Euclidiana Quadrática.

4.4.1 Tamanho das janelas de dados

Uma das principais vantagens do CHIP-clas em relação a outros classificadores é que não existem hiperparâmetros a serem otimizados e necessidade de configurações adicionais por parte do usuário. Entretanto, a definição do tamanho

Algorithm 2 Algoritmo de treinamento do CHIP-clas de forma paralela

input: Pares de amostras de treinamento $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$, Fator de divisão de janelas $\mathcal{K}$
output: Conjunto de Arestas de suporte globais $\mathcal{AS}$
for i in $\mathcal{K}$ **do** ▷ para $i = 1, \ldots, \mathcal{K}$
    Constrói o GG para cada janela de dados k_i
    Calcula a matriz de adjacência $\mathbf{A}_i$
    $\mathcal{V}_i \leftarrow (v_{i,1}, v_{i,2}) \in \mathbf{A}_i$
    if sign$(v_{i,1})$ == sign$(v_{i,2})$ **then**
        $\hat{\mathcal{AS}}_i \leftarrow (v_{i,1}, v_{i,2})$ ▷ Conjunto local de arestas de suporte $\hat{\mathcal{AS}}$
    end if
end for
$\mathbf{X}_{new}$ é a combinação de todas as janelas de suporte locais $\hat{\mathcal{AS}}$
$\mathbf{A}$ é a matriz de adjacência de $\mathbf{X}_{new}$
for j in $\mathbf{X}_{new}$ **do**
    $\mathcal{V} \leftarrow (v_{j,1}, v_{j,2}) \in \mathbf{A}$
    if sign$(v_{j,1})$ == sign$(v_{j,2})$ **then**
        $\mathcal{AS} \leftarrow (v_{j,1}, v_{j,2})$ ▷ Conjunto global de arestas de suporte $\mathcal{AS}$
    end if
end for
return $\mathcal{AS}$

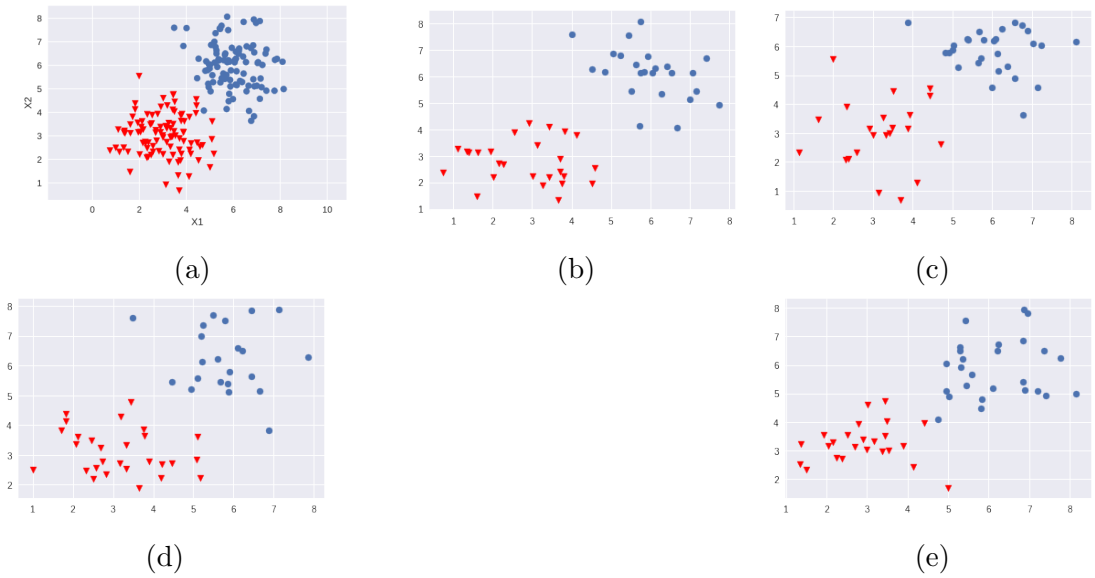


Figura 4.4: a- Dataset completo, b-e Janelas aleatórias

das janelas pode ser considerada como um possível parâmetro a ser otimizado. Nesse sentido, foi criado um número arbitrário de limiar que classifica a base de dados de acordo com o seu tamanho e busca executar mais divisões dos dados para bases com maiores quantidades de amostras, de forma automática, a fim de que o modelo se adapte à aplicação e distribua os dados de maneira uniforme entre núcleos de processamento.

Capítulo 5

Experimentos e Resultados Computacionais

Os experimentos foram projetados de forma que os métodos propostos nos capítulos 3 e 4 fossem avaliados, em relação ao desempenho e eficiência dos modelos. Os resultados foram organizados em relação à implementação em *software* e em *hardware*.

5.1 Validação em *Software*

Inicialmente foi avaliado, através da implementação em software, o compromisso entre o desempenho do modelo em testes quantitativos e a redução da precisão numérica. Os métodos foram implementados em 12 bases de dados do repositório da *University of California Irvine* (Dheeru & Karra Taniskidou, 2017). A Tabela 5.1 apresenta os valores médios da Área sob a Curva ROC (AUC-ROC), assim como o desvio padrão. A escolha da métrica AUC se deve ao fato de que é uma métrica robusta para avaliação de classificadores em bases de dados com distribuição desbalanceada de classes.

Os atributos das bases de dados foram normalizados entre $\{0, 1\}$ e a média dos resultados foi obtida através da validação cruzada, utilizando *10-fold cross validation*. Os nomes das bases de dados são apresentados assim como a quantidade de amostras (n) e dimensões (d) dos dados.

Os modelos são descritos da seguinte forma: O classificador NN-clas, implementado em ponto-flutuante de precisão dupla de 64 *bits* (NN-clas 64) serve como base de comparação para o mesmo método, com redução de precisão numérica, de 16 *bits* (NN-clas 16). O método de computação paralela descrito no capítulo 4 também é apresentado para comparação, em precisão dupla (Par-clas 64) e de 16 *bits* (Par-clas 16) em ponto-flutuante. Todos os modelos foram implementados com representação numérica no padrão IEEE 754. Foi utilizada ainda a técnica de filtragem apresentada na seção 2.3.2.

A definição do valor de divisão de janelas (k) no modelo Par-clas foi configurado de forma automática. Foi definido um valor de limiar que caracteriza o tamanho da base de dados de forma a efetuar mais divisões nas bases com maior quantidade de amostras. Uma vez que a maior base de dados avaliada possui 1372 amostras, o limiar foi definido como sendo 400 amostras. Dessa forma, as bases com quantidade de amostras maior que o valor de limiar têm o valor de k configurado através da divisão do número de amostras por um fator de escala, neste caso 100. Já paras as bases com quantidade de amostras menor que o limiar, esse fator de escala foi de 50. Tais valores foram escolhidos de forma arbitrária a partir do conhecimento prévio dos tamanhos das bases de dados avaliadas. Os valores destacados em negrito foram os que apresentaram melhores resultados em relação à métrica AUC.

Tabela 5.1: Média da AUC para os classificadores com precisão de 16 e 64 bits

Bases de dados	n	d	NN-clas16	NN-clas64	Par-clas16	Par-clas64
Banknote Auth.	1372	4	0.993 ± 0.009	$\mathbf{1.000 \pm 0.000}$	0.998 ± 0.003	$\mathbf{1.000 \pm 0.000}$
Austra. cred.	690	14	$\mathbf{0.815 \pm 0.009}$	0.772 ± 0.056	0.808 ± 0.060	0.772 ± 0.056
Fertility	100	9	$\mathbf{0.762 \pm 0.194}$	$\mathbf{0.762 \pm 0.194}$	$\mathbf{0.762 \pm 0.194}$	$\mathbf{0.762 \pm 0.194}$
Haberman's S.	306	3	$\mathbf{0.548 \pm 0.158}$	0.477 ± 0.112	0.546 ± 0.110	0.484 ± 0.097
P.I.diabetes	768	8	0.656 ± 0.070	$\mathbf{0.675 \pm 0.090}$	0.642 ± 0.052	0.673 ± 0.090
Breast cancer	683	9	0.942 ± 0.034	$\mathbf{0.955 \pm 0.029}$	0.913 ± 0.055	0.951 ± 0.035
Climate M.S.C.	540	18	0.658 ± 0.093	$\mathbf{0.663 \pm 0.093}$	0.649 ± 0.080	$\mathbf{0.663 \pm 0.093}$
German Credit	1000	24	0.598 ± 0.066	$\mathbf{0.625 \pm 0.056}$	0.585 ± 0.064	$\mathbf{0.625 \pm 0.056}$
Parkinsons	197	23	$\mathbf{0.913 \pm 0.023}$	$\mathbf{0.913 \pm 0.023}$	$\mathbf{0.913 \pm 0.023}$	$\mathbf{0.913 \pm 0.023}$
Sonar. M R	208	60	$\mathbf{0.902 \pm 0.115}$	0.848 ± 0.117	$\mathbf{0.902 \pm 0.115}$	0.848 ± 0.117
Statlog heart	270	13	$\mathbf{0.812 \pm 0.060}$	0.778 ± 0.022	$\mathbf{0.812 \pm 0.060}$	0.778 ± 0.022
ILPD	583	10	$\mathbf{0.557 \pm 0.056}$	0.536 ± 0.079	0.537 ± 0.051	0.529 ± 0.081

Os resultados foram avaliados através do teste de Friedman (Friedman, 1937), que é um teste não paramétrico, robusto para casos onde não se pode garantir as premissas de homocedasticidade e distribuição normal dos dados e adequado para comparação entre mais de dois métodos (Demšar, 2006).

As hipóteses analisadas foram:

$$\begin{cases} H_0 : \tau_i = 0, \quad \forall i \in \{1, 2, \ldots, a\} \\ H_1 : \exists\ \tau_i \neq 0 \end{cases}$$

onde τ representa a diferença média de desempenho entre os métodos analisados, sendo a o total de algoritmos, neste caso, 4.

Dessa forma, a hipótese nula representa a ausência de diferenças significativas entre os métodos analisados, já a hipótese alternativa avalia a existência de diferenças de desempenho entre os algoritmos. Com um nível de significância $\alpha = 0.001$, a estatística *Friedman* $\chi^2 = 0.30612$ e $p_{value} = 0.9589 > \alpha$, os resultados não apresentam evidências suficientes para rejeitar a hipótese nula de que não há diferenças entre os resultados médios dos modelos comparados.

Este resultado é confirmado através da análise gráfica do desempenho médio dos algoritmos. A Figura 5.1, apresenta a distribuição das médias do desempenho dos modelos, indicando uma semelhança entre os 4 modelos avaliados. Essa semelhança é confirmada ainda pela Figura 5.2, onde as médias dos valores de AUC para cada base de dados avaliada são comparadas entre os algoritmos.

5.1.1 Avaliação do tempo de execução

Nesta seção é apresentada uma análise sobre a eficiência dos métodos em relação ao tempo de execução. Uma vez que foi implementada a validação cruzada, em cada execução dos algoritmos, o tempo de processamento foi armazenado e em seguida obtida a média aritmética dos valores de tempo de execução.

A Tabela 5.2 apresenta a média dos tempos de execução (em segundos), para o NN-clas e Par-clas, ambos implementados em 16 e 64 *bits* em ponto-flutuante. Para fins de reprodutibilidade, os testes foram executados em um servidor com processador Intel® *Xeon* E5645, 2.40 GHz, no sistema operacional Linux versão 4.15.0-43. Os valores em negrito apresentam os menores tempos de execução.

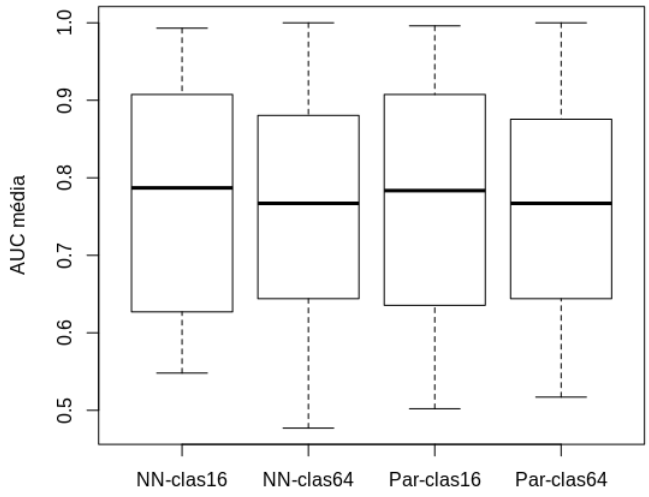


Figura 5.1: Comparação da distribuição dos valores médios de AUC

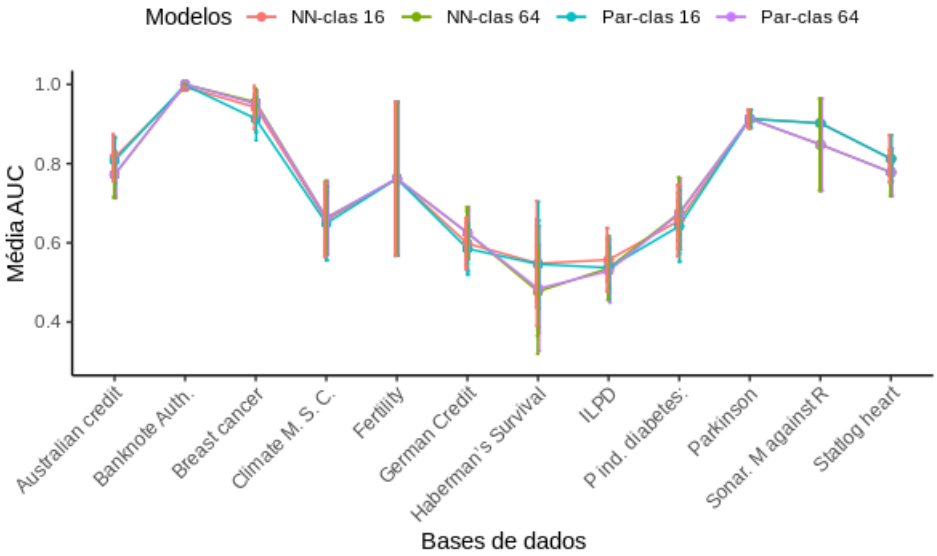


Figura 5.2: Média AUC nas diferentes bases de dados

Novamente, o teste Friedman foi realizado para analisar o desempenho dos métodos. As hipóteses analisadas foram:

Tabela 5.2: Tempo de execução (s) do NN-clas e Par-clas

Datasets	NN-clas16	NN-clas64	Par-clas16	Par-clas64
Bank Auth.	1,524.725	1,250.717	346.552	**317.585**
Aust. cr.	216.339	**171.441**	208.860	173.957
Fertility	2.074	**1.986**	4.002	3.954
Haberman	31.451	25.969	23.121	**18.231**
Diabetes	241.091	211.145	**134.637**	202.378
Brea. ca.	342.741	283.129	58.028	**48.291**
Climate	82.071	**69.097**	79.369	70.455
Germ. Cr.	545.567	**330.612**	532.542	355.853
Parkinson	10.074	9.062	8.477	**7.935**
Sonar	10.960	9.191	10.704	**9.163**
S. heart	18.204	**16.921**	22.138	20.448
ILPD	110.569	97.546	**57.514**	81.759

$$\begin{cases} H_0 : \tau_i = 0, \quad \forall i \in \{1, 2, \ldots, a\} \\ H_1 : \exists\ \tau_i \neq 0 \end{cases}$$

onde τ representa diferença média do tempo de execução, sendo a o total de algoritmos, neste caso, 4.

A hipótese nula representa a ausência de diferenças significativas no tempo de execução dos métodos analisados, já a hipótese alternativa avalia a existência de diferenças de desempenho no tempo de execução dos métodos. Com um nível de significância $\alpha = 0.001$, a estatística *Friedman* $\chi^2 = 16.4$ e $p_{value} = 0.0009387 \approx \alpha$, os resultados não apresentam evidências suficientes para rejeitar a hipótese nula da existência de diferenças na média dos tempos de execução dos modelos avaliados.

Entretanto, através de uma análise gráfica do *bloxplot* da Figura 5.3 e do gráfico de comparação do desempenho mostrado na Figura 5.4, observa-se uma redução significativa do tempo de execução, associada à utilização da técnica de computação paralela em algumas das bases de dados avaliadas. Observa-se por exemplo uma redução de 70% do tempo de execução na base de dados *"Banknote Authentication"*, com 1372 amostras. Um comportamento similar é observado

na base *"Breast Cancer"* (Mangasarian, 1990), com 683 amostras, apresentando uma redução de cerca de 80%.

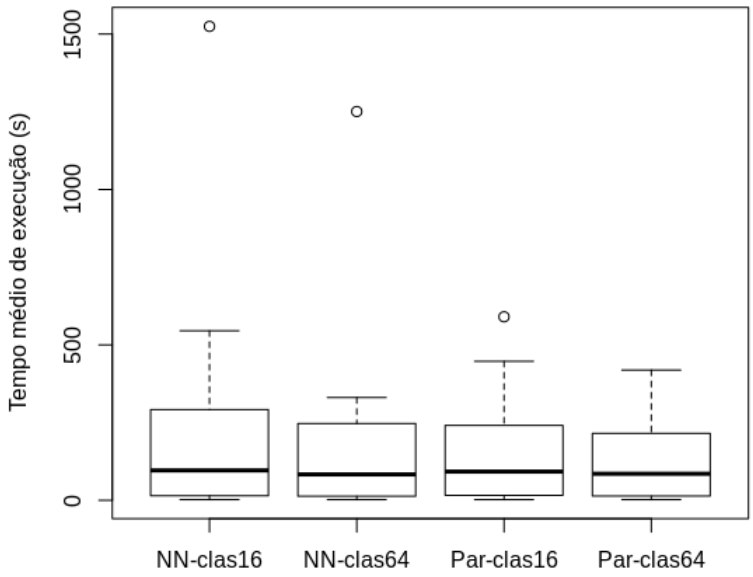


Figura 5.3: *Boxplot* comparando a média do tempo de execução dos métodos

Para bases com menores quantidades de amostras no entanto, não é observado melhora significativa no desempenho do método que implementa computação paralela. Isso pode ser analisado do ponto de vista do tamanho amostral reduzido, dado que bases com menores quantidades de amostras não possuem tamanho amostral suficiente para que ocorram divisões nas janelas de dados de forma a acelerar a construção do grafo na fase de treinamento.

Observa-se ainda que algumas implementações em 16 *bits* apresentam um tempo de execução relativamente maior, em comparação com 64 *bits*. Isso pode ter relação ao fato de que os modelos de 16 *bits* precisaram passar por sucessivas conversões de representação numérica para realização dos testes em *software*.

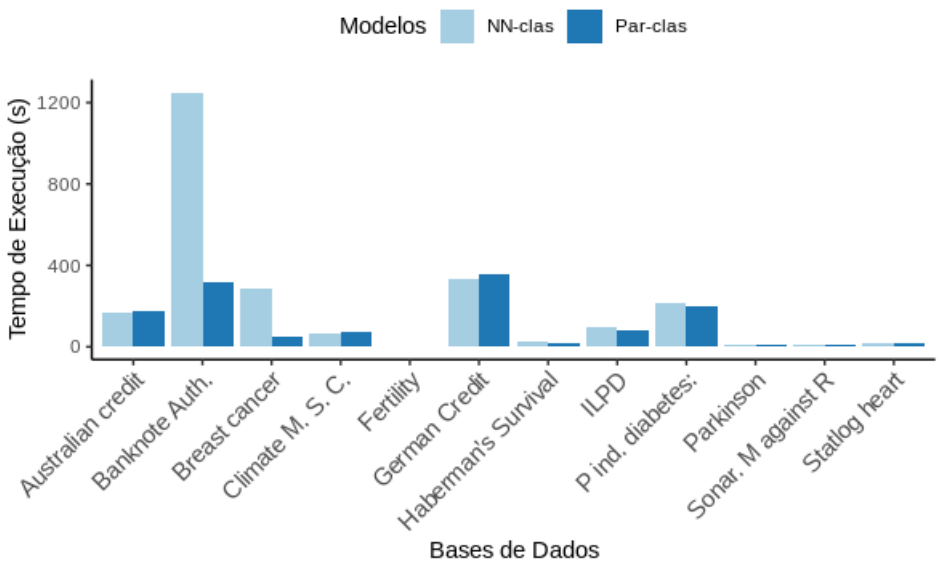


Figura 5.4: Comparação do tempo de execução para os modelos em 64 *bits*

5.1.2 Avaliação do tamanho da janela de dados

Na proposta de construção do grafo de Gabriel de forma paralela na etapa de treinamento, os experimentos foram projetados de maneira que se mantivesse a ausência de parâmetros a serem otimizados, uma vez que este é um dos fatores que tornam o CHIP-clas um classificador adequado para implementação em sistemas embarcados e em aplicações que não necessitem de entrada de dados ou configurações adicionais por parte do usuário.

Portanto, dado um parâmetro k que representa a quantidade de divisões às quais os dados serão expostos, o tamanho da janela de dados é definido por (n/k), onde n representa a quantidade de amostras dos dados de treinamento. Nos experimentos apresentados na seção anterior, o número de divisões dos dados foi definido de forma automática, como um fator de escala a partir do número de amostras da base de dados, de forma que estas divisões em janelas pudessem ser exploradas em bases de dados com maiores quantidades de amostras.

Entretanto, faz-se necessária uma avaliação do impacto da escolha do valor de k em relação ao desempenho e velocidade de processamento do modelo. Dessa forma, foram escolhidas cinco bases de dados, dentre as que foram avaliadas na

seção anterior, de forma a analisar o impacto da técnica de computação paralela na eficiência do modelo, a partir da variação do parâmetro de divisão de janelas de dados (k). Foram escolhidas as bases de dados com maiores quantidades de amostras e dimensões.

A Tabela 5.3 apresenta os resultados da avaliação da variação do parâmetro k. São reportados a quantidade de amostras (n) e dimensões (d) das bases de dados, assim como a quantidade de amostras restantes após a filtragem dos dados ruidosos (n*). A Tabela apresenta ainda informações da quantidade de amostras de dados de treinamento X_{train} (foi utilizada a divisão de 80% da base de dados após a filtragem para ser utilizada na etapa de treinamento), tempo de execução, em segundos e desempenho, através da métrica AUC. A coluna $\mathcal{W}$ apresenta o tamanho da última janela de dados, que é resultado do processo de filtragem realizado pela divisão de janelas. Esta porcentagem está diretamente associada à eficiência do modelo, uma vez que a complexidade da fase de treinamento do classificador diminui com a redução do número de amostras de treinamento.

Por fim, é apresentada a quantidade de amostras do conjunto de arestas de suporte global ($\mathcal{AS}$) e a porcentagem de redução dos dados, computada a partir do número de amostras após a etapa de filtragem inicial (n*) em relação à quantidade de amostras em $\mathcal{W}$.

São avaliados cinco valores para o parâmetro de (k). Os valores que apresentaram melhores resultados associados à eficiência do modelo, em relação ao tempo de execução, foram destacados em negrito. Para $k = 1$, a técnica de divisão de janelas não é utilizada, uma vez que considera-se toda a base de dados em uma única janela de processamento, de forma que esta configuração não possui os valores de $\mathcal{W}$ e porcentagem de redução dos dados.

Em relação ao desempenho do Par-clas para os diferentes valores de k, o resultado da métrica AUC sofreu pouca ou nenhuma alteração. Já em relação à eficiência associada ao tempo de processamento, observa-se um comportamento semelhante entre as bases de dados avaliadas, conforme apresentado na Figura 5.5.

O tempo de processamento cai de forma acentuada na primeira divisão de dados avaliada ($k = 5$) em comparação com o modelo sem implementação da divisões ($k = 1$). Entretanto, para $k > 5$, a divisão dos dados não se traduz de

Tabela 5.3: Avaliação de desempenho para diferentes valores de k

Base de dados	n	d	n*	X_{train}	k	Tempo(s)	AUC	$\mathcal{W}$	$\mathcal{AS}$	Redução (%)
Banknote Auth.	1372	4	1178	942	1	1024	1.0000	–	163	–
					5	213	1.0000	397	158	66.30%
					10	261	1.0000	476	163	59.59%
					20	317	1.0000	530	152	55.01%
					30	368	1.0000	579	158	50.85%
					50	431	1.0000	620	155	47.37%
German Credit	1000	24	594	475	1	509	0.5483	–	475	–
					5	271	0.5483	474	474	20.20%
					10	264	0.5483	475	475	20.03%
					20	261	0.5483	474	474	20.20%
					30	261	0.5483	475	475	20.03%
					50	247	0.5483	462	462	22.22%
Breast cancer	698	9	557	445	1	246	0.9167	–	144	–
					5	34	0.9167	152	121	72.71%
					10	35	0.9042	174	124	68.76%
					20	38	0.9167	183	117	67.15%
					30	41	0.9167	192	121	65.53%
					50	46	0.9042	203	120	63.55%
Australian credit	689	14	433	346	1	265	0.7593	–	342	–
					5	138	0.7593	346	342	20.09%
					10	133	0.7593	344	340	20.55%
					20	135	0.7593	344	340	20.55%
					30	128	0.7500	336	336	22.40%
					50	117	0.7593	321	318	25.87%
Diabetes	768	8	482	385	1	306	0.7060	–	359	–
					5	145	0.6984	355	343	26.35%
					10	131	0.6984	344	331	28.63%
					20	129	0.6984	342	328	29.05%
					30	119	0.7070	329	317	31.74%
					50	101	0.7307	304	293	36.93%

forma direta em melhora significativa de eficiência. De fato, para algumas das bases de dados, a eficiência tende a melhorar suavemente à medida que se aumenta o valor de k. Entretanto, para outras bases a eficiência piora para valores maiores de k, como no caso da base *Banknote Authentication*, indicando uma saturação do ganho de eficiência em relação ao aumento no número de divisões dos dados.

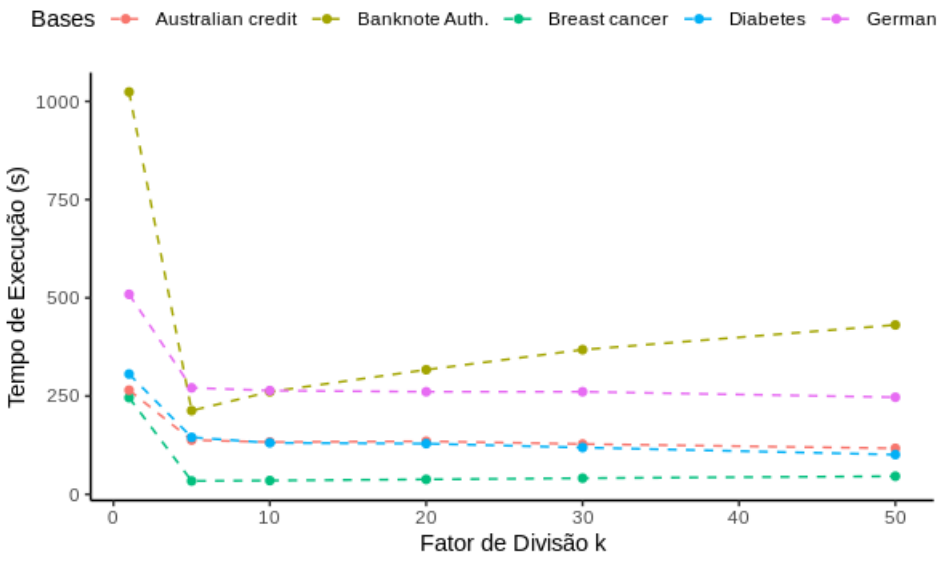


Figura 5.5: Comparação de diferentes valores de divisão de janelas de dados

5.1.3 Testes de Escalabilidade

Com o objetivo de avaliar a eficiência do algoritmo de treinamento paralelo do NN-clas em bases de dados com uma quantidade elevada de amostras, apresenta-se nesta seção a avaliação dos modelos Par-clas e NN-clas na base de dados *mnist* (LeCun *et al.*, 1998), que é uma base clássica para reconhecimento de padrões. A base original possui 60 mil imagens de 28×28 *pixels*, indicando um total de 784 dimensões.

Cada imagem pode representar um dígito de 0 a 9, tratando-se de um problema de classificação de múltiplas classes. Entretanto, uma vez que os modelos da família do CHIP-clas são aplicados em problemas de classficação binária, definiu-se um rótulo (dígito) a ser classificado contra os demais dígitos (neste caso o dígito "2" foi escolhido para representar uma das classes e os demais dígitos agrupados na outra classe). Foi selecionada uma subamostra do conjunto total contendo 10 mil imagens, sendo que destas foram escolhidas 80% como amostras de treinamento. Uma vez que o número original de dimensões é muito elevado, optou-se por realizar a técnica de redução de dimensionalidade PCA (Hotelling, 1933), reduzindo de 784 para 16 dimensões.

A Tabela 5.4 apresenta os resultados de AUC, tempo de execução, em segundos, e ainda os valores de quantidades de amostra na última janela de dados ($\mathcal{W}$) e o tamanho do conjunto de $\mathcal{AS}$. Novamente foram avaliados diferentes valores do parâmetro de divisão de janelas de dados. Inicialmente foi avaliado o modelo sem divisões ($k = 1$). Foram ainda implementados os valores de ($k = 5$), uma vez que este número de divisões apresentou resultados promissores na seção 5.1.2 e um valor arbitrário de divisões ($k = 1000$) por se tratar de uma base com grandes quantidades de amostras. O valor de (k) que apresentou menor tempo de execução é destacado em negrito. Neste caso, o processo de filtragem dos dados mostrado na seção 2.3.2 não foi utilizado, uma vez que aumentaria o tempo de processamento dado a necessidade de construção do Grafo de Gabriel.

Tabela 5.4: Resultados na base *mnist*

Base de dados	n	d	X_{train}	k	Tempo(s)	AUC	$\mathcal{W}$	$\mathcal{AS}$	Redução (%)
mnist	10000	16	8000	1	1.47e + 05	0.9620	–	–	–
				5	1.51e + 05	0.9705	7981	7903	0.24%
				1000	1.41e + 05	0.9715	7852	7768	1.85%

Os resultados não indicaram ganho de eficiência na utilização da técnica de divisão dos dados, para os valores de k avaliados, uma vez que o tempo de execução foi semelhante nos três casos. Algumas hipóteses podem ser levantadas em relação à esses resultados. Observa-se que não houve uma filtragem adequada dos dados a partir das divisões em janelas de forma que implicasse em um ganho de desempenho, conforme mostrado na última coluna da Tabela 5.4.

Outra questão que pode ser analisada é a arquitetura proposta do algoritmo paralelo. Como pode-se observar na construção do algoritmo *cascade SVM* apresentada na Figura 4.2, existem várias camadas de divisão dos dados. Entretanto, na arquitetura proposta e validada na base *mnist*, foi realizada uma divisão simples dos dados, o que pode não ser suficiente para impactar o desempenho final do modelo em termos de tempo de execução, devido à grande quantidade de amostras na etapa de treinamento. Dessa forma, pode-se ainda analisar a viabilidade da implementação de camadas sucessivas de divisão dos dados no algoritmo Par-clas.

5.2 Validação em *Hardware*

Para avaliar o impacto da redução da representação numérica do classificador, implementado em *hardware*, foi realizada, para cada subetapa do modelo, a simulação do NN-clas em FPGA com precisão simples de 32 *bits* no padrão IEEE 754 (FP32) e nos formatos de 16 *bits* em ponto-flutuante: FP16 (*half-precision* no padrão IEEE) e BF16 (*bfloat*), através do *software* Modelsim®.

5.2.1 Etapa de Treinamento

Para avaliação da etapa de treinamento, foram geradas bases de dados sintéticos com diferentes valores de quantidade de amostras e dimensões. Foi então realizada uma comparação entre o resultado da subetapa de cálculo de distância através da simulação em FPGA dos formatos FP32, FP16 e BF16, com os resultados obtidos através do software Matlab®, tendo como métrica de comparação o Erro Médio Quadrático (MSE). A métrica de distância usada para o cálculo da matriz de distâncias foi a Euclidiana quadrática. O EP de cálculo da distância foi escolhido um vez que sua correta execução é crucial para a construção do GG e consequente obtenção do conjunto de $\mathcal{AS}$. A Tabela 5.5 sumariza os resultados obtidos.

Tabela 5.5: MSE do Cálculo de distância para diferentes dimensões

Dimensões	FP32	FP16	BF16
4×4	3.1526e−12	4.7165e−4	0.0122
16×4	9.1061e−12	3.7442e−4	0.0237
30×8	2.5876e−11	0.0017	0.0792
60×10	5.8860e−11	0.0029	0.1617
120×20	5.8860e−11	0.0161	0.9598

É perceptível que o MSE cresce de forma proporcional ao número de amostras e de dimensões dos dados, uma vez que o número de operações a serem executadas aumenta. Os resultados corroboram com os apresentados em (dos Reis Gade, 2018). A Figura 5.6 apresenta o aumento do erro associado à implementação de 16 *bits* em relação ao de precisão simples de 32 *bits*, sendo o MSE deste

último quase nulo. Nota-se ainda que o erro associado ao modelo que utiliza representação BF16 é maior que os demais, uma vez que esse formato possui a menor quantidade de *bits* na mantissa.

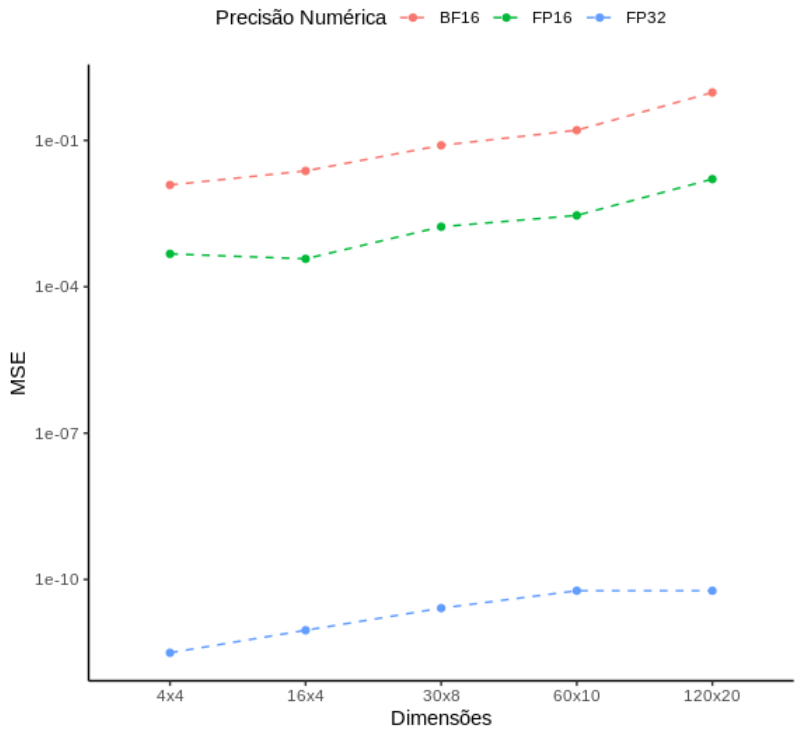


Figura 5.6: Comparação do MSE para diferentes dimensões dos dados

5.2.2 Etapa de Classificação

Para validar a etapa de classificação, foi realizada a simulação através do *software* Modelsim® da etapa de classificação do NN-clas. Foram utilizados os elementos de processamento do cálculo e comparação dos valores de distância, implementados nos formatos FP32, FP16 e BF16.

Para termos de comparação, foi utilizada a metodologia de testes apresentada em (dos Reis Gade, 2018). Esta metodologia considera toda a base de dados de treinamento como o conjunto de $\mathcal{AS}$, uma vez que a etapa de classificação do NN-clas pode ser comparada com o algoritmo K-NN, com o valor de $k = 1$, com o objetivo de simplificar os testes da etapa de classificação do modelo.

Portanto, a Tabela 5.6 apresenta os valores médios de AUC assim como o desvio padrão do desempenho do classificador NN-clas, nos formatos de precisão numérica avaliados. Novamente foram utilizadas nos testes, as bases de dados avaliadas no desempenho da implementação em *software* (Tabela 5.1), com adição das bases "*breast hess*" (Hess *et al.*, 2006), "*bupa (liver disorders)*" e "*golub*" (Golub *et al.*, 1999). Os resultados foram obtidos através da validação cruzada 10-*fold*. Os valores destacadas em negrito são os que obtiveram os melhores resultados.

Tabela 5.6: Média da AUC da simulação em FPGA

Datasets	n	d	FP32	FP16	BF16
Bank Auth.	1372	4	**0.9987 ± 0.0028**	0.9986 ± 0.0029	**0.9987 ± 0.0028**
Aust. cr.	690	14	**0.7904 ± 0.0473**	0.4118 ± 0.0478	**0.7904 ± 0.0473**
Fertility	100	9	**0.5778 ± 0.2241**	**0.5778 ± 0.2241**	**0.5778 ± 0.2241**
Haberman	306	3	**0.5299 ± 0.0802**	0.5274 ± 0.0808	**0.5299 ± 0.0802**
Diabetes	768	8	0.6722 ± 0.0579	0.6712 ± 0.0584	**0.6752 ± 0.0571**
Brea. ca.	683	9	**0.9489 ± 0.0307**	**0.9489 ± 0.0307**	**0.9489 ± 0.0307**
Climate	540	18	0.5941 ± 0.0878	0.5941 ± 0.0878	**0.5951 ± 0.0883**
Germ. Cr.	1000	24	**0.6948 ± 0.0642**	0.6048 ± 0.0642	0.6041 ± 0.0652
Parkinsons	197	23	**0.9454 ± 0.1255**	0.9426 ± 0.1251	**0.9454 ± 0.1255**
Sonar	208	60	**0.8720 ± 0.0615**	0.8649 ± 0.0752	**0.8720 ± 0.0615**
S. heart	270	13	0.7898 ± 0.0863	**0.7948 ± 0.0792**	0.7863 ± 0.0822
ILPD	583	10	0.5953 ± 0.0718	0.5953 ± 0.0718	**0.5988 ± 0.0740**
breast hess	133	30	**0.7144 ± 0.2279**	**0.7144 ± 0.2279**	**0.7144 ± 0.2279**
bupa	345	7	**0.5967 ± 0.0786**	0.5944 ± 0.0785	0.5895 ± 0.0820
golub	72	50	**0.7450 ± 0.2999**	**0.7450 ± 0.2999**	**0.7450 ± 0.2999**

Foram realizadas ainda as análises estatística e gráfica do desempenho do NN-clas a partir da simulação, para os valores de precisão avaliados. O teste Friedman foi utilizado para avaliação dos resultados obtidos. As hipóteses levantadas foram:

$$\begin{cases} H_0 : \tau_i = 0, \quad \forall i \in \{1, 2, \ldots, a\} \\ H_1 : \exists\ \tau_i \neq 0 \end{cases}$$

onde τ representa a diferença média de desempenho do NN-clas em relação à variação da precisão numérica, sendo a o total de formatos avaliados, neste caso os três formatos (FP32, FP16, BF16).

Dessa forma, a hipótese nula representa a ausência de diferenças significativas na média da AUC entre as configurações avaliadas, já a hipótese alternativa avalia a existência de diferenças na média da AUC das configurações. Com um nível de significância $\alpha = 0.001$, a estatística *Friedman* $\chi^2 = 5.8919$, e $p_{value} = 0.05255 > \alpha$, os resultados não apresentam evidências suficientes para rejeitar a hipótese nula de que não há diferenças entre os resultados médios dos diferentes valores de precisão numérica comparados.

A Figura 5.7 apresenta a distribuição das médias de AUC dos três formatos de precisão numérica implementados e a Figura 5.8 apresenta a comparação dos resultados nas diferentes bases de dados avaliadas. A análise gráfica corrobora com os resultados apresentados no teste estatístico, indicando semelhanças no desempenho do NN-clas nos formatos avaliados.

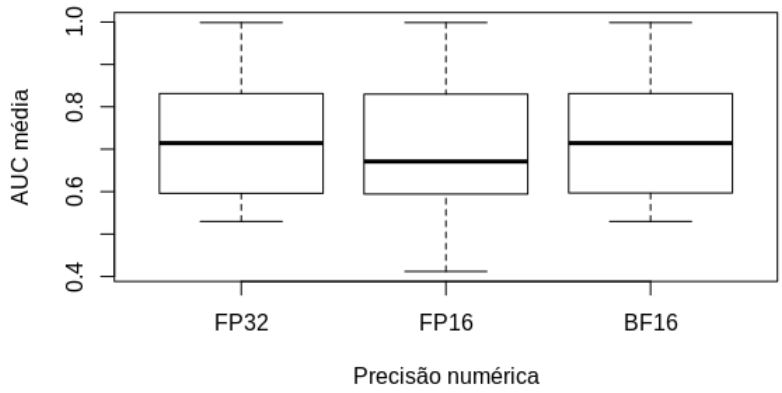


Figura 5.7: *Boxplot* comparando o desempenho da implementação em *hardware*

Observa-se no entanto que, para a base "*Australian credit*"o desempenho da implementação em ponto-flutuante de 16 *bits* no padrão IEEE 754 foi excessivamente menor em comparação com a implementação em 32 *bits* também no padrão IEEE. Entretanto, o resultado da implementação no formato *bfloat* apresentou a mesma média AUC em comparção com a implementação em precisão simples (FP32). Isso indica que, em relação à redução da precisão numérica neste caso avaliado, o formato *bfloat* apresenta vantagens em relação ao padrão IEEE.

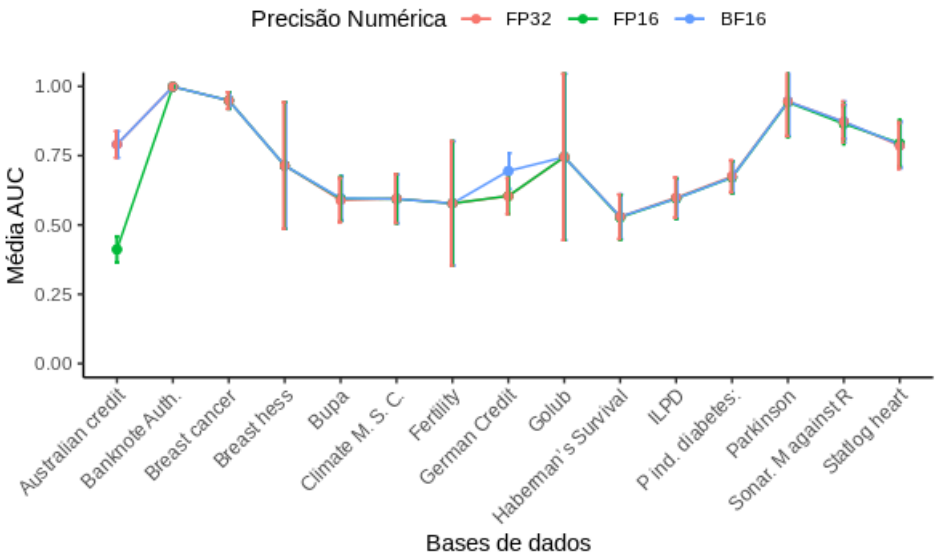


Figura 5.8: Média AUC da simulação em FPGA

Essa vantagem está associada à maior quantidade de *bits* no expoente (apesar da redução na mantissa), garantindo uma maior extensão de representação numérica.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho apresentou duas propostas com o objetivo de melhorar a eficiência computacional de classificadores baseados no Grafo de Gabriel, como o CHIP-clas e o NN-clas. Uma das propostas foi a avaliação de um método de construção do GG de forma paralela, explorando potenciais aplicações baseadas no paralelismo intrínseco de plataformas de *hardware* como FPGAs, GPUs e processadores modernos. Outra proposta explorou a utilização de representação numérica de precisão reduzida, avaliando o compromisso entre o desempenho do classificador e a precisão numérica utilizada. Foram implementadas análises estatísticas e gráficas a fim de avaliar os resultados e traçar conclusões sobre as propostas apresentadas.

Em relação à redução de precisão numérica, os experimentos computacionais indicaram que os classificadores baseados no CHIP-clas são robustos para aplicações com computação em precisão numérica reduzida nas bases de dados e formatos de representação numérica avaliados. Na simulação da implementação do NN-clas em *hardware*, os três formatos de representação numérica avaliados (FP32, FP16 e BF16) apresentaram resultados estatisticamente equivalentes. Entretanto, o modelo com representação em ponto-flutuante de 16 *bits* no padrão IEEE não obteve resultados adequados de desempenho em uma das bases de dados avaliadas (*Australian Credit*). Já o formato *bfloat* de 16 *bits* em ponto-flutuante (BF16) se mostrou robusto e atingiu resultado equivalente à implementação de 32 *bits* neste caso específico, indicando o potencial de uso desse

formato em detrimento do padrão IEEE 754 em aplicações onde faz-se necessária a utilização de uma maior extensão de representação numérica.

Esses resultados são extremamente significativos do ponto de vista da implementação em sistemas embarcados com recursos limitados de armazenamento, de forma que mostram a possibilidade da obtenção do mesmo nível de desempenho, com um consumo de memória reduzido pela metade. Essa redução de recursos implica ainda em uma redução do custo de implementação, uma vez que apresenta uma menor demanda de recursos de memória e área do circuito.

Os resultados indicaram ainda a viabilidade da proposta de construção do grafo de Gabriel de forma paralela na etapa de treinamento, uma vez que também apresentou resultados estatisticamente equivalentes ao modelo NN-clas utilizado como base de comparação. Este resultado é importante uma vez que o custo computacional da implementação do GG através de iterações consecutivas entre as amostras para obtenção da distância entre as mesmas e consequentemente definição da formação de arestas no grafo é proibitivo para aplicações com bases de dados com um número elevado de amostras. Nesse sentido, um método de distribuição dessas amostras entre núcleos de processamento é fundamental para que o algoritmo seja escalável para problemas mais complexos.

Apesar das indicações de que a implementação da técnica de computação paralela na etapa de treinamento propicia um ganho de eficiência do modelo em termos de tempo de processamento, a análise do impacto do número de divisões em relação ao ganho computacional mostrou que a definição do parâmetro k, o qual realiza o ajuste no tamanho das janelas de dados é importante para o desempenho do classificador. Os resultados para as bases de dados avaliadas indicaram que o ganho de eficiência é limitado a um número específico de divisões, de forma que, a partir desse valor, não ocorre um aumento real da eficiência do modelo.

A avaliação da escalabilidade do modelo na base de dados *"mnist"* mostrou que precisam ser realizadas melhorias na metodologia de divisão dos dados proposta. Novos estudos podem ser realizados de forma a analisar o impacto do número de dimensões dos dados em relação ao desempenho computacional da técnica de construção do GG de forma paralela. Pode-se ainda avaliar a implementação de camadas sucessivas de divisão dos dados, a fim de melhorar o

desempenho em bases de dados com um elevado número de amostras de treinamento.

6.1 Trabalhos futuros

A partir dos resultados iniciais em relação ao compromisso entre precisão numérica e desempenho, futuros trabalhos podem estender essa proposta, avaliando ainda outras formas de representação além da representação em ponto flutuante. Pode-se avaliar a viabilidade da utilização de representação em ponto-fixo, propondo novas formas de normalização. Nesse contexto, seria interessante ainda avaliar a implementação de formatos alternativos de representação numérica os quais sejam mais eficientes, de maneira similar à proposta de utilização do formato *bfloat*. Trabalhos futuros podem ainda estender essa proposta para outros modelos de aprendizado de máquina que utilizam métricas de distância como base de implementação.

Outra questão importante que ainda está em aberto é o desenvolvimento de uma arquitetura da metodologia de remoção de dados ruidosos na etapa de pre-processamento apresentada na seção 2.3.2. Uma vez que a remoção destes dados é crucial para um correto desempenho do classificador, faz-se necessária a implementação dessa etapa em *hardware*. Existem ainda alguns desafios a serem solucionados, uma vez que a classificação de dados como ruído depende do grau do vértice que por sua vez depende da construção do grafo. Entretanto essa construção é a etapa mais complexa computacionalmente, e neste caso deveria ser realizada para a determinação e consequente remoção dos dados ruidosos e novamente para a obtenção das arestas de suporte na etapa de treinamento, tornando o processo inviável em aplicações com quantidades maiores de dados. Portanto, seria interessante a avaliação de novos métodos de classificação e remoção de dados ruidosos os quais não dependam diretamente da construção do grafo de Gabriel, para uma melhor implementação em sistemas embarcados.

Uma vez que a definição do parâmetro de divisões de janela k é importante para a eficiência da utilização da técnica de computação paralela, trabalhos futuros podem fazer um estudo mais detalhado sobre a definição desse parâmetro, de forma a se desenvolver uma metodologia de configuração do mesmo a partir da

distribuição dos dados, com o objetivo de garantir a premissa de independência de parâmetros que é característica fundamental dos classificadores da família CHIP-clas.

Finalmente, novos estudos podem ser feitos no sentido de avaliar e propor novos métodos para avaliação de outras restrições de projetos da implementação do modelo CHIP-clas em sistemas embarcados, como quantificar a analisar o consumo de energia do mesmo. Aplicações futuras podem ainda expandir a proposta de paralelismo apresentada neste trabalho, de forma a validar sua eficácia em diferentes plataformas de *hardware*, como por exemplo GPUs. Nesse sentido, poderiam ser feitos estudos de caso em ambientes reais, como por exemplo a implementação de sistemas embarcados em plantas industriais, de forma a avaliar tais métodos em ambientes não controlados, com diferentes tipos de dados e sujeito a dados mais ruidosos.

Referências

Anguita, D., Boni, A. & Ridella, S. (2003). A digital architecture for support vector machines: theory, algorithm, and fpga implementation. *IEEE Transactions on neural networks*, **14**, 993–1009. 38

Ascher, D., Dubois, P.F., Hinsen, K., Hugunin, J., Oliphant, T. *et al.* (2001). Numerical python. 40

Aupetit, M. & Catz, T. (2005). High-dimensional labeled data analysis with topology representing graphs. *Neurocomputing*, **63**, 139–169. 15

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, **23**, 345–405. 10

Berg, M.d., Cheong, O., Kreveld, M.v. & Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd edn. 11

Bishop, C.M. (2006). *Pattern recognition and machine learning*. springer. 12

Bondy, J.A., Murty, U.S.R. *et al.* (1976). *Graph theory with applications*, vol. 290. Citeseer. 8, 9

Cofer, R. & Harding, B.F. (2006). *Rapid System Prototyping with FPGAs: Accelerating the Design Process*. Elsevier. 24

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, **20**, 273–297. 2, 13

Courbariaux, M., Bengio, Y. & David, J.P. (2014). Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*. 38

Cover, T.M., Hart, P.E. *et al.* (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, **13**, 21–27. 28

de Freitas Diadelmo, M.V. (2016). *Aprendizado Incremental com Memória Parcial via Grafo de Gabriel*. Master's thesis, Universidade Federal de Minas Gerais, Brasil. 4, 44, 45

de Oliveira, C.A., de Aguiar, J.A., Fontanini, M.G.S. & Wendling, M. (2011). Dispositivos lógicos programáveis. 23

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, **7**, 1–30. 51

Deng, Y. (2019). Deep learning on mobile devices: a review. In *Mobile Multimedia/Image Processing, Security, and Applications 2019*, vol. 10993, 109930A, International Society for Optics and Photonics. 2

Dheeru, D. & Karra Taniskidou, E. (2017). UCI machine learning repository. 49

dos Reis Gade, L. (2018). *Estudo e desenvolvimento arquitetural para implementação de um classificador geométrico de margem larga em sistemas embarcados*. Master's thesis, Universidade Federal de Minas Gerais, Brasil. x, 4, 21, 28, 29, 30, 31, 32, 33, 34, 35, 36, 40, 41, 60, 61

dos Reis Gade, L., de Castro, C.L., Torres, L.C.B., Coelho, F.G.F., Braga, A.P., Garcia, J.A. & Torres, F.S. (2017). Nn-clas: classificador geométrico de margem larga baseado na regra do vizinho mais próximo. 4, 28, 33, 37

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, **32**, 675–701. 51

Gabriel, K.R. & Sokal, R.R. (1969). A new statistical approach to geographic variation analysis. *Systematic zoology*, **18**, 259–278. 11, 14

Girija, S.S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Software available from tensorflow. org*. 39

Golub, T.R., Slonim, D.K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J.P., Coller, H., Loh, M.L., Downing, J.R., Caligiuri, M.A. *et al.* (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *science*, **286**, 531–537. 62

Gomes, I.P., Torres, L.C.B. & de Pádua Braga, A. (2017). Aprendizado de métrica supervisionado para classificador por arestas de suporte. 4, 17

Graf, H.P., Cosatto, E., Bottou, L., Dourdanovic, I. & Vapnik, V. (2005). Parallel support vector machines: The cascade svm. In *Advances in neural information processing systems*, 521–528. xi, 44, 45

Hess, K.R., Anderson, K., Symmans, W.F., Valero, V., Ibrahim, N., Mejia, J.A., Booser, D., Theriault, R.L., Buzdar, A.U., Dempsey, P.J. *et al.* (2006). Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer. *Journal of clinical oncology*, **24**, 4236–4244. 62

Holler, M., Tam, S., Castro, H. & Benson, R. (1989). An electrically trainable artificial neural network (etann) with 10240 floating gate synapses. In *International Joint Conference on Neural Networks*, vol. 2, 191–196. 3

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, **24**, 417. 58

Hung, M. (2017). Leading the iot – gartner insights on how to lead in a connected world. Tech. rep., Gartner, Inc. 1

Inacio, C. & Ombres, D. (1996). The dsp decision: Fixed point or floating? *IEEE Spectrum*, **33**, 72–74. 25

Intel (2018). BFLOAT16 – Hardware Numerics Definition. Tech. rep., Intel Corporation. x, 39

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An introduction to statistical learning*, vol. 112. Springer. x, 12, 14

Johnson, J. (2018). Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*. 38

Jouppi, N., Young, C., Patil, N. & Patterson, D. (2018). Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, **38**, 10–19. 3

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D.T., Jammalamadaka, N., Huang, J., Yuen, H. *et al.* (2019). A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*. 39

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *et al.* (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**, 2278–2324. 58

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *nature*, **521**, 436. 2

Lee, E.A. & Seshia, S.A. (2016). *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press. 1

Lin, D., Talathi, S. & Annapureddy, S. (2016). Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, 2849–2858. 38

Lippmann, R.P. (1987). Anintroduction to computing with neural nets,". *IEEE Assp magazine*, **4**, 4–22. 2

Maignan, L. & Gruau, F. (2011). Gabriel graphs in arbitrary metric space and their cellular automaton for many grids. *TAAS*, **6**, 12:1–12:14. x, 11

Mangasarian, O.L. (1990). Cancer diagnosis via linear programming. *SIAM news*, **23**, 1–18. 54

Manne, S., Chin, B. & Reinhardt, S.K. (2017). If you build it, will they come? *IEEE Micro*, **37**, 6–12. 3

Markidis, S., Der Chien, S.W., Laure, E., Peng, I.B. & Vetter, J.S. (2018). Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 522–531, IEEE. 3

Matula, D.W. & Sokal, R.R. (1980). Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis*, **12**, 205–222. 22

Okabe, A., Boots, B., Sugihara, K. & Chiu, S.N. (2009). *Spatial tessellations: concepts and applications of Voronoi diagrams*, vol. 501. John Wiley & Sons. 10

Ovtcharov, K., Ruwase, O., Kim, J.Y., Fowers, J., Strauss, K. & Chung, E.S. (2015). Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, **2**. 22

Prasad, S.K., Gupta, A., Rosenberg, A.L., Sussman, A. & Weems, C.C. (2015). *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. 43

Prestes, E. (2016). Introdução à teoria dos grafos. 8

Rezende, F.A.V.S., Almeida, R.M.V. & Nobre, F.F. (2000). Diagramas de voronoi para a definição de áreas de abrangência de hospitais públicos no município do rio de janeiro. *Cadernos de Saúde Pública*, **16**, 467–475. 10

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**, 386. 2

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1985). Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science. 2

Salgado, M.N. (2019). *Regularizaçao De Classificadores Geométricos De Margem Larga Baseados No Grafo De Gabriel*. Ph.D. thesis, Universidade Federal de Minas Gerais. 17

Smith, M.J.S. (1997). *Application-specific integrated circuits*, vol. 7. Addison-Wesley Reading, MA. 22

Torres, L.C., Lemos, A.P., Castro, C.L. & Braga, A.P. (2014). A geometrical approach for parameter selection of radial basis functions networks. In *International Conference on Artificial Neural Networks*, 531–538, Springer. 4

Torres, L.C., Castro, C.L. & Braga, A.P. (2015a). A parameterless mixture model for large margin classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–6, IEEE. 4

Torres, L.C., Lemos, A., Castro, C. & Braga, A. (2015b). Gabriel graph for dataset structure and large margin classification: A bayesian approach. In *Proc. Eur. Symp. Artificial Neural Netw.*, 237–242. 4

Torres, L.C.B. (2016). *Classificador por arestas de suporte(CLAS): Métodos de aprendizado baseados em grafos de Gabriel*. Ph.D. thesis, Universidade Federal de Minas Gerais, Brasil. 4, 15, 16

Torres, L.C.B. *et al.* (2015c). Distance-based large margin classifier suitable for integrated circuit implementation. *Electronics Letters*, **51**, 1967–1969. x, 3, 14, 19

Wilson, P. (2007). *Design Recipes for FPGAs: Using Verilog and VHDL (Embedded Technology)*. Embedded Technology Series, Newnes. x, 23

Zhang, W. & King, I. (2002). A study of the relationship between support vector machine and gabriel graph. In *Proceedings of IEEE World Congress on Computational Intelligence—International Joint Conference on Neural Networks*. 21

Ziviani, N. (2010). *PROJETO DE ALGORITMOS COM IMPLEMENTAÇÕES EM JAVA EC+*. Cengage Learning Edições Ltda. 8, 9

Zuras, D., Cowlishaw, M., Aiken, A., Applegate, M., Bailey, D., Bass, S., Bhandarkar, D., Bhat, M., Bindel, D., Boldo, S. *et al.* (2008). Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, 1–70. 27