UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Ciência de dados aplicada em predições de desempenho para processamento paralelo em GPUs

Benício Ramos Magalhães

Trabalho de Conclusão de Curso - MBA em Ciência de Dados (CEMEAI)

ICMC USP SÃO CARLOS

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Ciência de dados aplicada em predições de desempenho para processamento paralelo em GPUs

Benicio Ramos Magalhães

USP - São Carlos

2021

BENICIO RAMOS MAGALHÃES

Ciência de dados aplicada em predições de desempenho para processamento paralelo em GPUs

Trabalho de conclusão de curso apresentado ao Centro de Ciências Matemáticas Aplicadas à Indústria do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, como parte dos requisitos para conclusão do MBA em Ciência de Dados.

Área de concentração: Ciências de Dados

Orientador: Prof. Dr. Antonio Castelo Filho

USP - São Carlos

2021

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

M188c

Magalhaes, Benicio Ramos
Ciência de dados aplicada em predições de desempenho para processamento paralelo em GPUs / Benicio Ramos Magalhaes; orientador Antonio Castelo Filho. -- São Carlos, 2021.
112 p.

Trabalho de conclusão de curso (MBA em Ciência de Dados) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2021.

1. Trabalho de conclusão de curso. 2. MBA. 3. Ciência de dados. 4. Computação paralela. 5. Predição de desempenho. I. Filho, Antonio Castelo, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

ERRATA

FOLHA DE AVALIAÇÃO OU APROVAÇÃO

DEDICATÓRIA

A minha esposa Pâmella, pela compreensão, carinho e apoio incansável.

AGRADECIMENTOS

Agradeço a minha família por todo o apoio durante todos esses anos de estudo e trabalho. As dificuldades nessa vida são diversas e, contudo, a dedicação e o esforço são essenciais para conquistarmos nossos objetivos pessoais e profissionais. Tudo isso só se torna real graças ao incentivo que consigo por meio dessas pessoas especiais. Muito obrigado à minha esposa Pâmella por estar sempre ao meu lado, com toda a paciência, dedicação e apoio. Saiba que você faz toda a diferença na minha vida! Obrigado aos meus filhos (Isaac, Davi e Lucca) por me darem motivação de seguir sempre em frente. Obrigado aos meus pais (José Armando e Elisabete) por estarem sempre presentes, me apoiando e por serem os melhores exemplos que tenho na vida. Amo todos vocês.

Agradeço também ao meu orientador Professor Doutor Antonio Castelo Filho, pela fundamental orientação ao longo deste período para concretização deste trabalho.

Finalmente agradeço a todo o corpo docente do curso do MBA de ciência de dados da ICMC/USP pela excelência e qualidade técnica ímpar com a qual fomos agraciados.

“A persistência é o menor caminho do êxito.”

- Charles Chaplin

RESUMO

MAGALHÃES, B. R. **Ciência de dados aplicada em predições de desempenho para processamento paralelo em GPUs**. 2020. 112 f. Trabalho de conclusão de curso (MBA em Ciência de Dados) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2020.

Este trabalho é um estudo de técnicas de predição de desempenho de processamento de computação paralela em GPUs utilizando ciência de dados. O objetivo é modelar o comportamento desses sistemas e estudar diversas técnicas existentes na área de ciência de dados e com isso predizer seu comportamento com relação à métricas não conhecidas. Inicialmente, realizamos uma pesquisa bibliográfica acerca dos principais trabalhos relacionados à previsão de desempenho de programas paralelos e, em seguida, obtivemos uma base de dados de métricas de desempenho de processamento paralelo de GPUs. A partir dessa base, elaboramos um modelo representativo e, por fim, realizamos predições de desempenho com relação às métricas coletadas.

Palavras-chave: Modelagem. Avaliação de desempenho. Computação paralela. Ciência de dados. Análise de predição de dados.

ABSTRACT

MAGALHAES, B. R. **Data science applied in performance predictions for parallel processing in GPUs**. 2020. 122 p. Completion of course work (MBA em Ciência de Dados) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2020.

This work is a study of performance prediction techniques of parallel computing processing in GPUs using data science. The objective is to model the behavior of these systems and to study several existing techniques in the area of data science and thereby predict their behavior in relation to unknown metrics. Initially, we carried out a bibliographic search about the main works related to the performance prediction of parallel programs and, then, we obtained a database of performance metrics of parallel processing of GPUs. From this base, we elaborate a representative model and, finally, we make performance predictions regarding the collected metrics.

Keywords: Modeling. Performance evaluation. Parallel computing. Data science. Data prediction analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AET	–	Application Execution Time
AI	–	Artificial Intelligence
ALU	–	Arithmetic-Logic Unit
BSP	–	Bulk Synchronous Parallel Model
CPU	–	Central Processing Unit
CUDA	–	Compute Unified Device Architecture
CRISP-DM	–	Cross Industry Standard Process for Data Mining
DL	–	Deep Learning
DT	–	Decision Tree
FPU	–	Floating Point Unit
GPU	–	Graphics Processing Unit
GDDR	–	Graphics Double Data Rate
HPC	–	High Performance Computing
IQR	–	Interquartile Range
K-NN	–	K-Nearest Neighbors
LR	–	Linear Regression
LD	–	Load
MAE	–	Mean Absolute Error
MBA	–	Master in Business Adminstration
ML	–	Machine Learning
MPI	–	Message Passing Interface
NB	–	Naive Bayes
NNGE	–	Non-Nested Generalized Exemplars
MLP	–	Multi-Layer Perceptron
OpenCL	–	Open Computing Language
PAS2P	–	Parallel Application Signature for Performance Prediction
PET	–	Predicted Execution Time
PETE	–	Prediction Execution Time Error
RAE	–	Relative Absolute Error
RBF	–	Radial Basis Function
RF	–	Random Forest

RMSE	–	Root Mean Squared Error
RRSE	–	Root Relative Squared Error
SCC	–	Spearman Correlation Coefficient
SET	–	Signature Execution Time
SFU	–	Special Function Units
SGEMM	–	Single precision GEneral Matrix Multiply
SM	–	Streaming Multiprocessors
SMOTE	–	Synthetic Minority Oversampling Technique
SP	–	Streaming Processors
ST	–	Store
SVM	–	Support Vector Machines
TPC	–	Texture Processing Clusters
UCI	–	University of California Irvine

SUMÁRIO

1 INTRODUÇÃO

Em um mundo tecnológico e moderno, onde os sistemas computacionais oferecem diversos benefícios à sociedade, existe uma necessidade cada vez maior de trabalhar com aplicações de alto desempenho e isso tornou-se viável por meio da utilização de sistemas distribuídos. “Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente.” (TANENBAUM, STEEN, 2007, p.1). Com isso os desenvolvedores começaram a procurar meios para escrever aplicações distribuídas com alta eficiência e isso resultou na utilização de novas tecnologias para processamento massivo paralelo, como as GPUs (*Graphics Processing Unit*).

“Computação paralela é mais que uma estratégia para atingir um alto desempenho, ela é uma visão de como a computação pode ser escalonada para ter um poder computacional praticamente ilimitado.” (DONGARRA et al, 2003, p.3). Esse aumento de poder computacional com baixo custo tem viabilizado a resolução de problemas complexos aplicada em diversas áreas de conhecimento.

Para aferir e garantir que uma aplicação atenda aos requisitos não-funcionais de alto desempenho, exigidos principalmente por sistemas de missão crítica, existem diversas técnicas de avaliação de desempenho. Jain (1991) define uma avaliação de desempenho como uma arte, portanto, assim como uma obra de arte, toda a avaliação requer um conhecimento íntimo do que está sendo modelado e uma seleção cuidadosa da metodologia, carga de trabalho e ferramentas.

Outra área de interesse neste trabalho, envolve o estudo de modelos preditivos para estimar o comportamento dos sistemas com relação ao seu desempenho. De acordo com SAS (2020), os modelos preditivos utilizam resultados conhecidos para desenvolver (ou treinar) um modelo que pode ser usado para prever valores para dados diferentes ou novos.

Neste trabalho, vamos nos basear nas metodologias previstas na área de ciência de dados para análise de predição desempenho do resultado do tempo de processamento de um produto matricial executado em diversas configurações aplicadas em uma GPU, obtendo assim modelos variados, realizando simulações e comparando os resultados previstos com os dados reais medidos.

1.1 Objetivos

Este trabalho tem como objetivo principal estudar técnicas e metodologias de predição existentes em ciência de dados aplicados em avaliações de desempenho de sistemas computacionais. Para isso, utilizamos dados com tempos de processamento em GPUs e propomos algumas técnicas de predição de desempenho usando algoritmos de aprendizado de máquina e estatística.

Como objetivos específicos, montamos um modelo representativo para explicar as características principais de um sistema computacional com relação as suas funções e seus parâmetros. Também aplicamos simulações no modelo para predizer seu comportamento com relação à métricas não conhecidas.

1.2 Motivação

A ciência de dados é uma área que se tem mostrado promissora para resolver problemas reais de negócios, com o uso de métodos científicos e técnicas avançadas para captura, tratamento de dados, aprendizado de máquina, redes neurais e inteligência artificial.

De maneira geral, a principal motivação para avaliar o desempenho de programas computacionais é aplicar uma metodologia que viabilize obtermos respostas sobre o comportamento de sistemas de grande complexidade de forma prática, rápida e mais barata utilizando algoritmos de ciência de dados.

1.3 Justificativa

O intuito é aplicar os conhecimentos dessa área numa base de dados obtida através de testes de desempenho em GPUs para que seja possível modelar e prever o comportamento de programas sobre condições que ainda não foram avaliadas.

1.4 Metodologia

Primeiramente foi feito um levantamento bibliográfico acerca de trabalhos recentes contendo metodologias e técnicas aplicadas na predição de desempenho de dados aplicados em GPUs. Em seguida, foi obtida uma base de dados de métricas de desempenho de um programa de processamento matricial em GPUs, com resultados obtidos através de combinações nos parâmetros de entrada. Na sequência, aplicamos algoritmos de ciência de dados com as principais características do sistema e a partir desses modelos elaborados, realizamos predições de desempenho utilizando simulações com dados não conhecidos.

1.5 Organização do trabalho

O próximo capítulo apresentará alguns trabalhos relacionados à predição de desempenho de programas paralelos, seguida das técnicas mais utilizadas em ciência de dados. No capítulo 3, abordaremos de forma teórica o que é computação paralela focado na utilização de GPUs. O capítulo 4 apresenta quais são e como são aplicados os modelos de predição com aprendizado de máquina. Já o capítulo 5 é a implementação dos algoritmos, geração e análise dos resultados, verificando a diferença entre os modelos, discutindo sua eficácia e comparando as predições com os dados reais medidos em testes. Finalmente, algumas conclusões decorrentes deste estudo são apresentadas no capítulo 6.

2 REVISÃO BIBLIOGRÁFICA

Fizemos um levantamento bibliográfico onde destacamos alguns trabalhos relevantes para o tema de pesquisa deste trabalho. As principais teses pesquisadas foram realizadas por González (2018) que apresenta uma predição de desempenho de aplicações executadas em diversas GPUs usando tanto um modelo analítico quanto técnicas de aprendizado de máquina. Destacamos também o trabalho realizado em Baldini et al. (2014) que mostra que é possível predizer qual GPU oferece o melhor desempenho na hora de realizar a portabilidade de um sistema de processamento paralelo executado originalmente em CPUs, utilizando apenas técnicas de aprendizado de máquina.

Considerando que temos interesse também na predição de desempenho de aplicações paralelas como um todo, destacamos aqui dois estudos que não necessariamente estão relacionados com GPUs, mas que geraram interesse devido ao fato de apresentar uma metodologia mais genérica para avaliação de sistemas desta natureza, como a PAS2P proposta por Wong, Rexachs e Luque (2015) e CRISP-DM proposta em Chapman et al [1]* (2000 *apud* QAZDAR et al, 2019, p.3580).

Na proposta do trabalho desenvolvido por González (2018) ele realiza a medição de algumas aplicações em diferentes GPUs e utiliza o método analítico através de um modelo BSP (*Bulk Synchronous Parallel Model)* e depois usa três abordagens de aprendizado de máquina (Regressão Linear, Máquinas de Vetor de Suporte e Florestas Aleatórias). Mesmo sabendo que o modelo analítico fornece previsões melhores, o intuito é mostrar que as técnicas de aprendizado de máquina podem oferecer previsões aceitáveis para todas as aplicações mesmo sem a necessidade de uma análise e um conhecimento profundo do sistema, nas quais são exigidas na modelagem analítica.

Em Baldini et al. (2014) eles lidam com as questões de esforço de portabilidade dos códigos que utilizam processamento paralelo. Realizar a portabilidade de um sistema de processamento paralelo projetado originalmente para CPUs (*Central Processing Unit*) pode gerar um esforço grande, sendo que muitas vezes é necessário reescrever todo o código para uma linguagem específica da GPU. Eles mostram que com técnicas de aprendizado de máquina é possível criar modelos que conseguem predizer o desempenho de forma precisa sem requerer

[1] *Chapman, P., Clinton, J., Kerber, R., Khabza, T., Reinartz, T., Shearer, C., & Wilrth, R. (2000). CRISP-DM 1.0 step-by-step data mining guide. The CRISP-DM consortium.

de uma análise estática do código ou mesmo a criação de um modelo analítico, conseguindo até mesmo demonstrar quais são os melhores dispositivos para o sistema.

Em Wong, Rexachs e Luque (2015) é proposta uma metodologia chamada de PAS2P (*Parallel Application Signature for Performance Prediction*) onde dados de desempenho do sistema são coletados e caracterizados por suas fases de execução (comportamento de repetição do algoritmo) e pesos (valor associado às métricas coletadas). Com essa informação é realizada uma predição do tempo de execução da aplicação paralela e depois validada através de resultados experimentais.

Por fim, a metodologia proposta por Chapman et al* (2000 *apud* QAZDAR et al, 2019, p.3580) é chamada de CRISP-DM (*Cross Industry Standard Process for Data Mining*). A metodologia consiste em alguns passos que vai desde o entendimento do negócio, passando pelo entendimento e preparação dos dados, a modelagem em si, que envolve implementações de aprendizado de máquina e a validação do modelo. Uma vez que o modelo está criado, testado e validado, a última etapa consiste na implantação, que pode ser um relatório com os resultados obtidos.

2.1 Trabalho desenvolvido por González, M. T. A.

O trabalho desenvolvido por González (2018) apresenta uma análise de desempenho de aplicações executadas em diversas GPUs dividida em duas etapas, a primeira utilizando um modelo analítico e a segunda utilizando métodos de aprendizado de máquina.

Foram utilizadas cinco aplicações algébricas (multiplicação e adição matricial, adição vetorial, produto escalar e sublista contígua de maior soma) apresentadas na tabela 2.1 e mais seis aplicações utilizando Rodinia (*Back Propagation, Gaussian Elimination, Heart Wall, Hot Spot, LU Decomposition, Needleman-Wunsch*) que são *benchmarks* do tipo *open-source* que contém códigos CUDA (*Compute Unified Device Architecture* ou arquitetura de dispositivo de computação unificada). Esses últimos apresentados na tabela 2.2. E na tabela 2.3 são apresentados nove diferentes GPUs utilizadas para nos experimentos:

Tabela 2.1 – Aplicações de álgebra linear utilizado nos experimentos de González (2018)

Application	Param	Kernel	dimGrid	dimBlock	Shared Mem
Matrix Mul	1	MMGU MMGC MMSU MMSC	(GS, GS, 1)	(BS, BS, 1)	0 0 $(BS^2 \times 2 \times 4B)$
Matrix Add	1	MAU MAC	(GS, GS, 1)	(BS, BS, 1)	0
Vector Add	1	VAdd	(GS, 1, 1)	(BS, 1, 1)	0
Dot Product	1	dotP	(GS, 1, 1)	(BS, 1, 1)	$(BS \times 4B)$
Max. Sub Array	1	MSA	(48, 1, 1)	(128, 1, 1)	$(4096 \times 4B)$

Fonte: González (2018, p.15).

Tabela 2.2 – Aplicações Rodinia utilizado nos experimentos de González (2018)

Application	Berkeley Dwarf	Domain	Param.	Kernels	Samples
Back Propagation (BCK)	Unstructured Grid	Pattern Recognition	1 - [57]	layerforward adjust-weights	57
Gaussian Elimination (GAU)	Dense Linear Algebra	Linear Algebra	1 - [32]	Fan1 Fan2	34800
Heart Wall (HWL)	Structured Grid	Medical Imaging	1 - [84]	heartWall	5270
Hot Spot (HOT)	Structured Grid	Physics Simulation	2 - [5,4]	calculate-temp	396288
LU Decomposition (LUD)	Dense Linear Algebra	Linear Algebra	1 - [32]	diagonal perimeter internal	8448 8416 8416
Needleman-Wunsch (NDL)	Dynamic Programming	Bioinformatics	2 - [16,10]	needle-1 needle-2	21760 21600

Fonte: González (2018, p.18).

Tabela 2.3 – Especificações dos Hardwares das GPUs utilizados nos experimentos de González (2018)

Model	C.C.	Memory	Bus	Bandwidth	L2	Cores/SM	Clock
GTX-680	3.0	2 GB	256-bit	192.2 GB/s	0.5 M	1536/8	1058 Mhz
Tesla-K40	3.5	12 GB	384-bit	276.5 GB/s	1.5 MB	2880/15	745 Mhz
Tesla-K20	3.5	4 GB	320-bit	200 GB/s	1 MB	2496/13	706 MHz
Titan	3.5	6 GB	384-bit	288.4 GB/s	1.5 MB	2688/14	876 Mhz
Quadro K5200	3.5	8 GB	256-bit	192.2 Gb/s	1 MB	2304/12	771 Mhz
Titan X	5.2	12 GB	384-bit	336.5 GB/s	3 MB	3072/24	1076 Mhz
GTX-970	5.2	4 GB	256-bit	224.3 GB/s	1.75 MB	1664/13	1279 Mhz
GTX-980	5.2	4 GB	256-bit	224.3 GB/s	2 MB	2048/16	1216 Mhz
Pascal-P100	6.0 GB	16 GB	4096-bit	732 GB/s	4 MB	3584/56	1328 Mhz

Fonte: González (2018, p.19).

Para o modelo analítico ele se baseia em BSP (*Bulk Synchronous Parallel Model*) que oferece uma simples abstração das arquiteturas paralelas, dividindo seus resultados em três grandes fases: processamento, comunicação e sincronização dos dados. O modelo analítico foi implementado conforme apresentado na equação 2.1, onde o tempo de execução é dividido apenas entre o custo computacional e custo de comunicação. Maiores detalhes sobre a aplicação deste modelo podem ser encontrados em (GONZÁLEZ, 2018, p.32):

$$T_K = \frac{l\ .(comp+comm_{GM}+comm_{SM})}{R\ .\ P\ .\ \lambda} \quad (2.1)$$

Fonte: González (2018, p.32).

Onde:

T_K: tempo de execução aproximado do kernel em função do número de threads.

l: número de threads

$comp$: custo computacional

$comm_{GM}$: custo de comunicação com a memória global

$comm_{SM}$: custo de comunicação com a memória compartilhada

R: taxa de clock

P: número de núcleos da GPU

λ: parâmetro estimado como taxa entre o tempo de execução previsto do kernel e o tempo de execução medido

Para avaliar a acurácia do modelo, foram realizados experimentos, onde foram comparadas as predições do modelo com os resultados medidos. A medida final da acurácia foi dada pela taxa T_K/T_m ,sendo T_K (tempo de execução do modelo) e T_m (tempo de execução medido).

Para o aprendizado de máquina ele utiliza três métodos: LR (*linear regression* ou regressão linear), SVM (*Support Vector Machines*) e RF (*random forest* ou florestas aleatórias), com duas diferentes implementações: a primeira usando todas as variáveis preditoras existentes e a segunda utilizando técnicas para otimização do uso de variáveis preditoras, realizando a extração de algumas delas. O erro nas predições foi calculado utilizando o MAPE (*Mean Absolute Percentage Error* ou erro percentual absoluto médio) apresentado na equação 2.2:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{T_m - T_k}{T_m}\right| \quad (2.2)$$

Fonte: González (2018, p.47).

Onde:

T_K: tempo predito

T_m: tempo medido

n: número de pontos ajustados

Para avaliar os modelos de aprendizado de máquina foram utilizados os procedimentos de validação cruzada ou *leave-one-out cross validation*, selecionando parte dos dados como teste (tanto no contexto de GPUs quanto *Kernels*) e o restante para treinar os modelos.

A implementação com extração de variáveis preditivas foi realizada usando técnicas de correlação e clusterização de dados, no caso, o SCC (*Spearman Correlation Coefficient* ou coeficiente de correlação de Spearman). O SCC foi aplicado para todas as variáveis contra o tempo de execução dos *kernels* e aplicado um limite de 0.75 neste coeficiente, realizando assim a redução do número de variáveis no modelo.

Os resultados da primeira implementação (com todas as variáveis preditivas) foram comparadas com o modelo analítico e são apresentadas a seguir nas tabelas 2.4 e 2.5:

Tabela 2.4 – MAPE das predições em % com as aplicações de vetoriais e matriciais de González (2018)

Apps	MAPE ML Techniques			
	AM	LR	SVM	RF
MMGU	4.41±4.97	16.47±13.40	20.48±14.88	10.65±10.21
MMGC	7.35±3.49	15.53±11.28	13.93±7.49	10.02±7.44
MMSU	7.95±5.31	5.33±1.93	11.79±4.25	4.90±1.99
MMSC	8.31±6.44	14.83±10.22	17.04±8.21	8.41±2.87
MAU	9.71±3.09	69.51±51.77	14.76±9.62	36.76±22.23
MAC	11.00±5.27	12.65±2.77	17.67±8.52	10.83±3.85
dotP	6.89±5.56	3.30±1.42	15.37±8.31	4.67±1.35
vAdd	9.92±7.71	18.98±16.56	14.68±8.15	8.64±4.56
MSA	28.02±13.05	3.02±1.62	10.91±5.02	5.67±4.11

Fonte: González (2018, p.52).

Tabela 2.5 – MAPE das predições em % com as aplicações Rodinia CUDA kernels de González (2018)

Kernels/Techn.	AM	LR	SVM	RF
BCK-K1	3.92±1.00	19.89±15.57	31.00±49.90	24.49±15.90
BCK-K2	4.86±3.33	86.42±158.74	140.21±283.86	97.49±188.53
GAU-K1	54.09±5.92	65.24±116.70	14.82±3.13	35.31±53.29
GAU-K2	63.72±5.12	26.62±12.38	18.52±11.73	18.60±7.48
HTW	3.71±2.23	100.79±192.76	41.63±38.04	26.29±25.13
HOT	5.53±2.14	167.16±293.56	57.61±83.96	116.92±183.91
LUD-K1	-	27.61±41.32	10.46±8.37	27.45±41.17
LUD-K2	-	57.70±79.18	42.50±54.74	48.16±70.49
LUD-K3	-	41.22±22.08	25.94±24.01	42.13±37.12
NDL-K1	-	16.09±5.27	15.54±10.09	14.03±3.81
NDL-K2	-	15.01±5.11	10.56±4.94	13.20±3.25

Fonte: González (2018, p.53).

Pode-se observar que o modelo analítico apresenta melhores resultados e que a previsão com o aprendizado de máquina não é muito boa em alguns dos algoritmos, e essas são listadas por diversas razões, como, por exemplo, o pouco número de amostras disponíveis. Mesmo assim, alguns algoritmos apresentaram uma boa predição, como observado no caso do RF para as aplicações vetoriais e matriciais. A análise das previsões é bem extensa e não é o objetivo deste trabalho apresentar o detalhamento de todas elas, mas ela pode ser encontrada em (GONZÁLEZ, 2018, p.52-59).

De maneira geral, ele consegue constatar que os modelos de aprendizado de máquina foram capazes de trazer resultados satisfatórios uma vez que apresentam uma abordagem mais generalizada para diferentes tipos de aplicativos e GPUs, sem precisar de conhecimento prévio a respeito dos mesmos, no qual é exigido na construção do modelo analítico.

2.2 Trabalho desenvolvido por Baldini, I., Fink, S. J., e Altman, E.

A metodologia aplicada em Baldini et al. (2014) endereça o problema de identificar o benefício de migrarmos uma aplicação originalmente desenvolvida para ser executada em CPUs para um ambiente com GPUs, sem o esforço de realizar a portabilidade do mesmo para este novo dispositivo, além disso, é possível demonstrar também que alguns modelos preditivos são até mesmo capazes de escolher qual o melhor dispositivo para aquela aplicação.

A figura 2.1 mostra um diagrama em alto-nível da abordagem realizada para prever o desempenho das aplicações em GPUs:

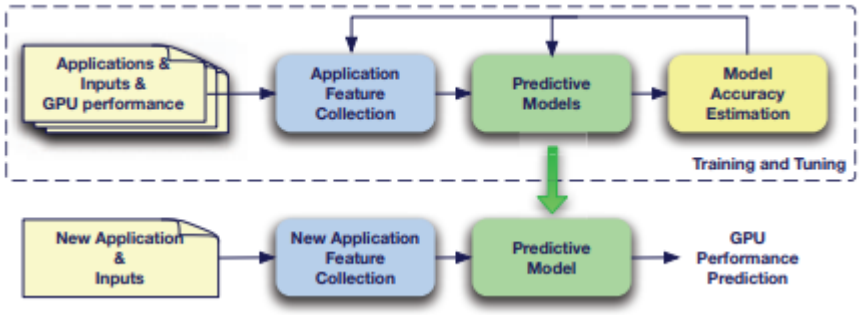

Figura 2.1 - Visão geral da metodologia para predição de desempenho de aplicações em GPUs baseado em duas fases: treino para calibração do modelo e predição, que aplica o modelo em uma nova aplicação.

Fonte: Baldini et al. (2014, p.255).

Foi realizada a escolha das variáveis preditivas pensando nas métricas mais relevantes para representar as características de uma aplicação paralela, sendo assim, as aplicações foram devidamente instrumentadas e as atenções restritas a estas variáveis. A tabela 2.6 apresenta as variáveis preditivas selecionadas em (BALDINI et al., 2014, p.256):

Tabela 2.6 – Variáveis preditivas utilizadas nas modelagens de aprendizado de máquina em Baldini et al. (2014)

Category	Feature	Mnemonic
Computation	Arithmetic and logic instructions	ALU
	SIMD-based instructions	SIMD
Memory	Memory loads	LD
	Memory stores	ST
	Memory fences	FENCE
Control flow	Conditional and unconditional branches	BR
OpenMP	Speedup of 12 threads over sequential execution	OMP
Aggregate	Total number of instructions	TOTAL
	Ratio of computation over memory	ALU-MEM
	Ratio of computation over GPU communication	ALU-COMM

Fonte: Baldini et al. (2014, p.256).

Todos os experimentos foram executados em um servidor com dois processadores de seis núcleos cada (totalizando 12 núcleos) e também equipados com duas placas GPUs. Eles utilizaram 18 *benchmarks* vindos da suíte de aplicações Parboil 2.0 e Rodinia 2.2 para a construção da sua base de dados e utilizaram procedimentos de validação cruzada ou *leave-one-out cross validation*, para selecionar parte dos dados como teste e o restante como treinamento para o desenvolvimento dos modelos.

O primeiro questionamento é se existiria uma forte correlação entre o ganho de desempenho medido na execução com CPUs (comparação utilizando 12 núcleos) contra o ganho medido nas GPUs. Chegaram à conclusão que não existia uma forte correlação e ainda que boa parte das execuções tinham desempenho pior em um ambiente com GPUs. Concluíram

que os benchmarks utilizados incluem uma boa quantidade de execuções nas quais uma aceleração vinda de GPUs não são efetivas, porém, outras poderiam apresentar benefícios.

Sendo assim, a segunda etapa é verificar se o ganho de desempenho da execução de um algoritmo em paralelo com relação à sua versão sequencial (*speedup*) justifica o esforço para realizar sua portabilidade. Neste caso, eles propõem uma solução formulando um problema de classificação binária, onde a classe identifica se a aplicação consegue um ganho em GPUs maior que um limite (*threshold*) pré-estabelecido. Os classificadores de aprendizado de máquina escolhidos foram o NNGE (*Non-Nested Generalized Exemplars*) e SVM (*Support Vector Machines*), sendo que foi obtida uma acurácia de aproximadamente 80% e o modelo treinado utilizando SVM teve uma acurácia um pouco melhor. Com isso foi possível criar um primeiro modelo de aprendizado de máquina que consegue verificar se o algoritmo é efetivo para uma aceleração vinda de GPUs.

Com este *sub set* de dados em mãos, o próximo passo é predizer qual o melhor dispositivo para realizar a portabilidade daquele algoritmo. Para isso foi utilizado o classificador NNGE com três classes, sendo elas correspondentes à uma CPU e duas GPUs distintas. Para cada execução do benchmark na base de dados, é feita uma classificação que corresponde ao dispositivo que obteve o melhor desempenho. Neste caso, o melhor desempenho foi baseado nas medições das métricas de operações de processamento e memória. Por fim, dadas as devidas considerações, obtiveram uma acurácia em torno de 91% para o preditor. A figura 2.2 apresenta os resultados obtidos para cada benchmark, sendo que cada *benchmark* teve três execuções (três grupos de barras apresentadas), o hardware com melhor desempenho em cada uma delas está pintada de azul escuro e as predições com erro foram apresentadas em vermelho escuro.

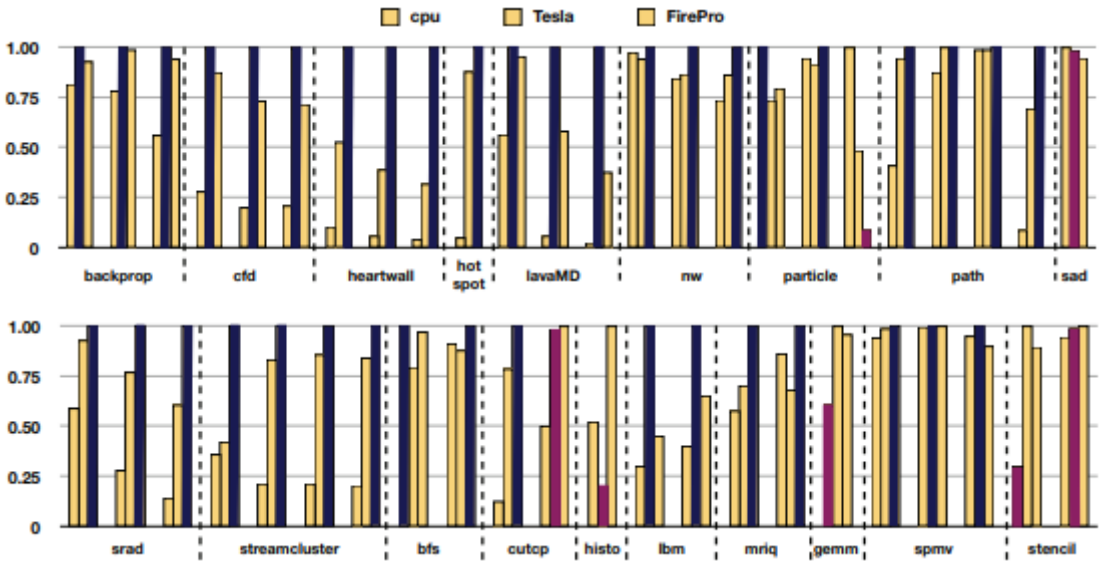

Figura 2.2 – Resultados da classificação de desempenho com três classes de dispositivos (CPU, Tesla GPU e FirePro GPU)

Fonte: Baldini et al. (2014, p.260).

2.3 PAS2P

PAS2P (*Parallel Application Signature for Performance Prediction*) proposta por Wong, Rexachs e Luque (2015) é um método de predição de desempenho empenhado em descrever uma aplicação baseada no seu comportamento. O PAS2P consiste basicamente em dois estágios:

1. Análise da aplicação e geração de sua assinatura;
2. Predição de desempenho.

A figura 2.3 apresenta uma visão geral da metodologia:

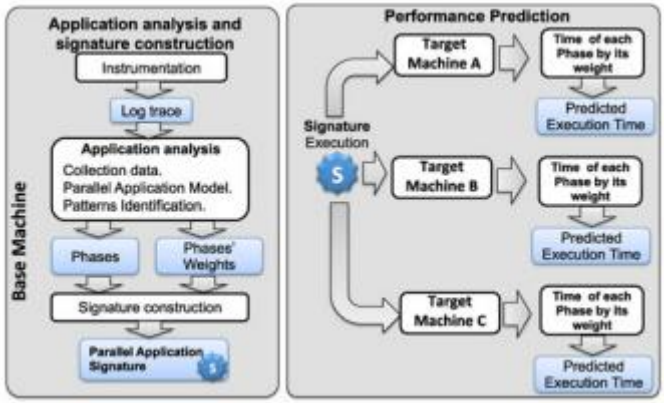

Figura 2.3 - Visão geral da metodologia PAS2P

Fonte: Wong, Rexachs, Luque (2015, p.2010).

A primeira etapa é o processo de gerar a assinatura da aplicação, que consiste em instrumentar o programa e executá-lo em uma máquina base gerando um log de rastreamento. Para modelar a aplicação, são coletadas métricas de tempo de execução dividindo as etapas de processamento em fases. Essas fases são agrupadas e é estipulado um peso para cada uma, que é definido pelo número de vezes em que elas ocorrem (repetições de um mesmo tipo de comunicação). A partir disso, as fases são marcadas (*checkpoints*) e finalmente são geradas as assinaturas, que nada mais são do que marcações do código instrumentado que sabem exatamente onde começa e termina cada fase.

A figura 2.4 exemplifica os passos para extração e determinação das fases do algoritmo:

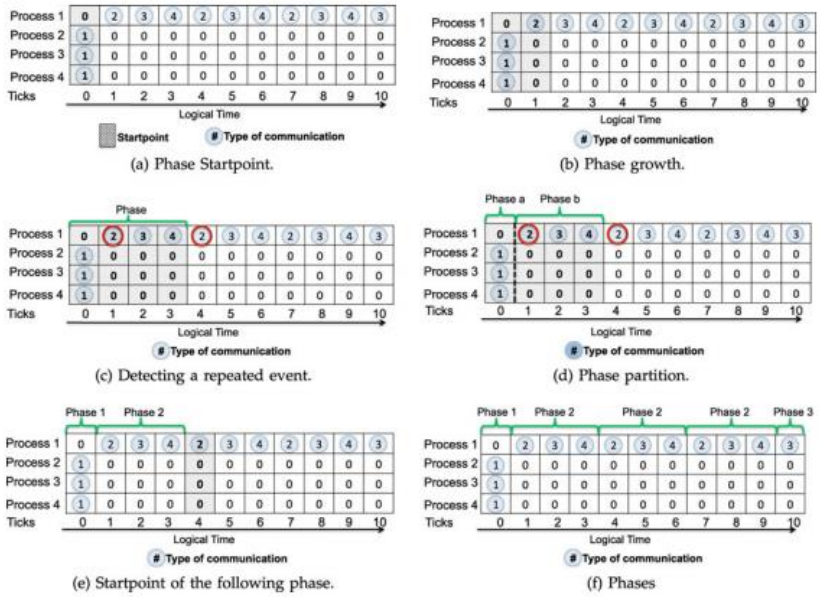

Figura 2.4 - Exemplificação do padrão do algoritmo de identificação das fases.

Fonte: Wong, Rexachs, Luque (2015, p.2013).

A figura 2.5 exemplifica uma tabela de fases que serve como base para construção da assinatura:

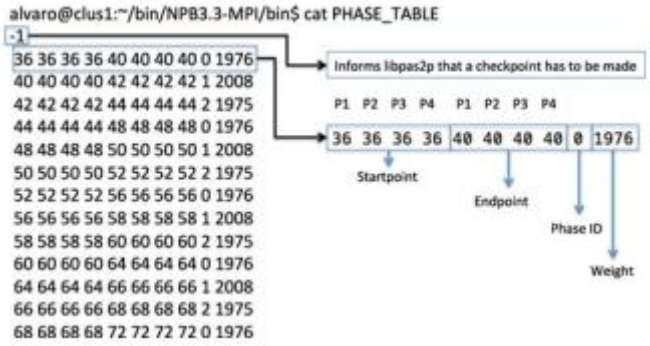

Figura 2.5 - Tabela de fases para construção da assinatura.

Fonte: Wong, Rexachs, Luque (2015, p.2014).

A segunda etapa consiste na predição de desempenho, que ocorre a partir da assinatura obtida na etapa um. Para predizer o tempo de execução PET (*Predicted Execution Time*) da aplicação é utilizada uma equação que consiste na multiplicação do tempo de execução de cada fase pelo seu peso.

Com as informações referentes à assinatura, executa-se um novo teste nas máquinas onde desejamos predizer o comportamento da aplicação. Essa execução mede o tempo de cada fase mapeada na assinatura e no final faz o cálculo da predição do tempo de execução. A principal vantagem é que a execução da assinatura da aplicação costuma ser muito mais eficiente do que a execução da aplicação em si, pois ele considera as estruturas de repetição como fases similares e os tempos preditos validados experimentalmente apresentam erros máximos de apenas 3%. Vale observar que nesta metodologia, caso a aplicação avaliada não apresente essa característica de repetição, o tempo de execução das fases será bem similar ao tempo de execução real da aplicação.

A tabela 2.7 apresenta um exemplo com os erros percentuais obtidos do tempo predito de algumas aplicações em relação ao tempo medido utilizando essa metodologia. Vale observar que o tempo de execução da assinatura SET (*Signature Execution Time*) é bem inferior ao tempo de execução da aplicação AET (*Application Execution Time*) e também que percentual de erro do tempo predito PETE (*Prediction Execution Time Error*) é bem pequeno, chegando ao máximo de 3% de variação.

Tabela 2.7 - Exemplos de resultados de predição de desempenho utilizando a metodologia PAS2P

Appl.	Cores	SET (Sec)	SET versus AET(%)	PET (Sec)	PETE(%)	AET (Sec)
CG-64	32	8.42	0.29	2793.42	1.90	2847.42
	64	4.87	0.32	1504.66	0.48	1511.91
BT-64	32	13.47	0.80	1652.65	0.9	1667.64
	64	10.19	0.77	1302.76	0.55	1309.91
SP-64	32	2.04	0.24	808.76	1.28	819.17
	64	2.08	0.51	388.367	3.05	400.55
SMG2k	32	16.75	2.63	633.23	0.38	635.61
64	64	8.37	10.15	162.87	2.32	166.74
Sweep	16	4.32	0.17	2494.36	0.06	2492.74
3d-32	32	3.01	0.22	1328.04	0.40	1322.62
POP-64	32	22.79	1.41	1608.85	0.17	1611.59
	64	18.36	1.79	1016.01	0.61	1022.28

SET: Signature Execution Time, SET versus AET: 100(SET/AET).
PET: Predicted Execution Time, AET: Application Execution Time.
PETE: Prediction Execution Time Error.

Fonte: Wong, Rexachs, Luque (2015, p.2016).

2.4 CRISP-DM

Mesmo não estando diretamente relacionada ao tema de predição de desempenho de programas paralelos, a metodologia apresentada por Chapman et al *apud* Qazdar et al chamou a atenção por conter um processo genérico para criação de modelos e predição de resultados.

A figura 2.6 apresenta uma visão geral da metodologia CRISP-DM*:*

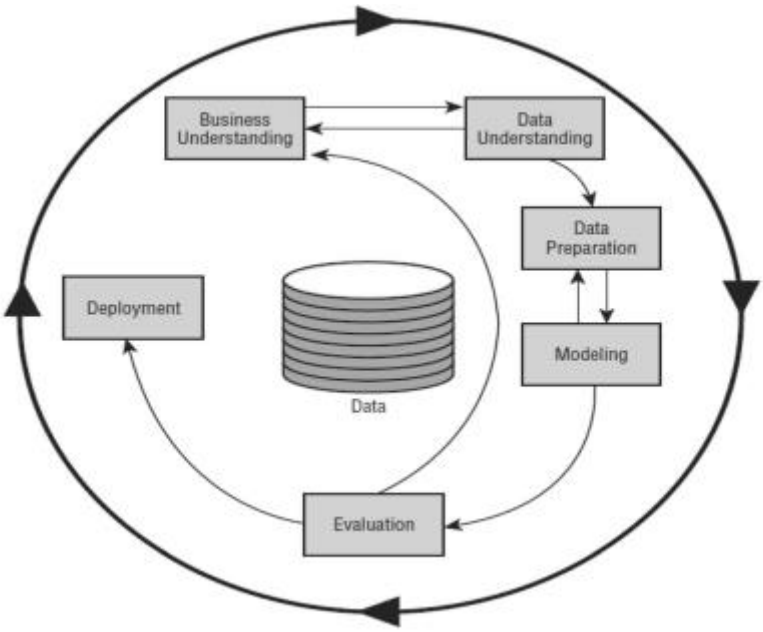

Figura 2.6 - Visão geral da metodologia CRISP-DM

Fonte: Qazdar et al. (2019, p.3581).

Para aplicarmos a metodologia CRISP-DM (*Cross Industry Standard Process for Data Mining*) são definidos os seguintes passos (QAZDAR et al, 2019, p.3580):

1. Entendimento do negócio em todos os seus aspectos, como por exemplo, os objetivos do projeto, os requisitos, regras aplicáveis, campo de aplicação, etc.;
2. Entendimento dos dados focado nas técnicas adequadas de coleta, descrição, exploração e manipulação;
3. Preparação dos dados. É a fase que cobre todas as atividades necessárias para o tratamento dos dados, obtendo assim uma base pronta para ser manipulada na fase de modelagem;
4. Modelagem. É a implementação de diferentes técnicas de aprendizado de máquina (regressão, classificação, clusterização, recomendação). A escolha desses algoritmos depende da necessidade do projeto, da base de dados e dos resultados almejados;

5. Avaliação do modelo. Aplicam-se técnicas de medidas de erro, como por exemplo, erro absoluto médio MAE (*Mean Absolute Error*), erro quadrático médio RMSE (*Root Mean Squared Error*), erro absoluto relativo RAE (*Relative Absolute Error*), erro quadrático relativo RRSE (*Root Relative Squared Error*), acurácia, entre outros. A escolha da avaliação do modelo está diretamente relacionada aos requisitos do projeto, o algoritmo usado e aos resultados desejados;
6. Entrega de resultados. Uma vez com o modelo criado, testado e avaliado, a entrega pode ser um relatório ou uma implementação do processo.

2.5 Considerações Finais

As fontes de pesquisa são de suma importância para a concepção de um novo trabalho. A ciência ganha cada vez mais credibilidade não só pelos esforços individuais de cientistas e pesquisadores, mas principalmente pelo fato de que a comunidade científica tem como ideal compartilhar conhecimento, ajudando a amadurecer idéias, firmar conceitos e principalmente evoluir. Reconhecendo a importância de estar inserido nesse contexto, seguimos para o próximo capítulo com a intenção de contribuir um pouco mais e compartilhar o conhecimento com todos aqueles que o anseiam.

3 COMPUTAÇÃO PARALELA COM GPUS

Computação paralela nada mais é do que escalar um processamento computacional por meio da utilização de diversas unidades de processamento trabalhando de forma simultânea com o objetivo de executar uma tarefa em comum. A técnica de paralelismo já existe há muitos anos e de acordo com Dongarra et. al (2003), ela é "uma visão atraente de como a computação pode ser escalonada de um único processador para um poder de computação virtualmente ilimitado".

Nos dias de hoje, este poder computacional tem sido cada vez melhor aproveitado, graças ao surgimento de aceleradores de hardware, como é o caso das GPUs (*Graphics Processing Unit)* ou unidade de processamento gráfico. Elas são basicamente processadores especializados em operações gráficas, sendo assim, foram projetadas de maneira específica para lidar com grandes conjuntos de dados de forma paralela, tornando-se mais eficientes do que as CPUs convencionais.

3.1 Introdução

A computação paralela traz uma série de desafios como, por exemplo, uma maior complexidade no desenvolvimento de softwares, problemas relacionados a comunicação e sincronização dos dados, desafios arquiteturais de memória compartilhada, paralelismo em CPUs, soluções de clusterização, entre outros. Assim sendo, todas as arquiteturas paralelas representam um peso entre o custo, complexidade, oportunidade e desempenho, porém, ela torna viável a resolução de problemas que anteriormente não conseguíamos resolver devido as limitações causadas pelo elevado tempo de processamento sequencial das informações.

A computação paralela faz uso de múltiplos núcleos de processamento, dividindo um problema em partes independentes, onde cada elemento de processamento consegue executar um algoritmo de forma simultânea. Dada a premissa de que podemos tirar um melhor proveito da computação paralela a partir de múltiplos núcleos é adequado explorarmos os conceitos básicos de funcionamento das GPUs.

GPUs inicialmente foram projetadas apenas para funções gráficas, porém, seu potencial em acelerar o processamento de dados foi logo percebido em cenários modernos de HPC (*High*

Performance Computing). O HPC ou computação de alto desempenho é o ambiente onde contextuamos o processamento de uma grande quantidade de dados para tratar problemas de ML (*Machine Learning* ou aprendizado de máquina), DL (*Deep Learning* ou aprendizado profundo) e *AI* (*Artificial Intelligence* ou inteligência artificial). Muitos outros setores também extraem benefícios com a computação de alto desempenho, pois ela ajuda a prover uma melhor acurácia e um maior detalhamento de modelos matemáticos que necessitam de um processamento massivo de dados. Como exemplo, podemos citar algumas áreas: a cosmologia, sismologia, simulação de dispositivos semicondutores, inteligência artificial, sistemas de reconhecimento de imagens, mapeando genético, etc.

3.2 Arquitetura computacional de GPUs

Como já comentado, o propósito inicial das GPUs são renderizações de imagens e é muito utilizada na execução de jogos eletrônicos de computador e consoles, sendo que as duas maiores fabricantes de GPUs nos dias de hoje são a NVIDIA e a AMD.

A figura 3.1 apresenta a arquitetura de uma GPU modelo Geforce GTX 8800, que foi uma das primeiras a suportar a tecnologia C ou CUDA:

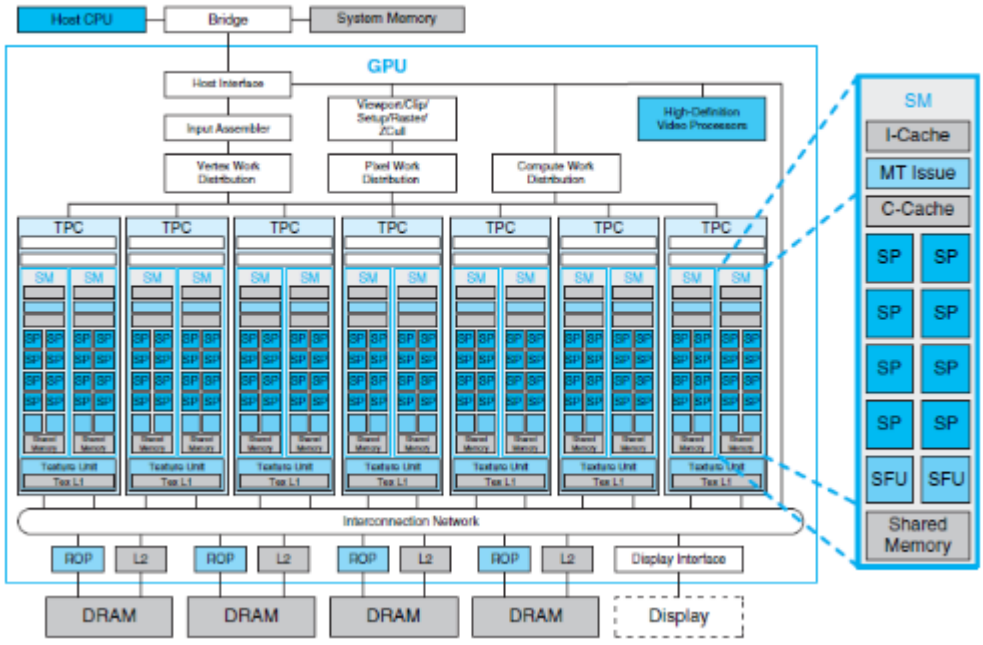

Figura 3.1 – GPU GeForce GTX 8800, primeira GPU a suportar plataforma CUDA.

Fonte: González (2018, p.10).

A seguir são explicados os principais componentes de uma GPU:

- **TPC (*Texture Processing Clusters*)**: Cada GPU consiste em múltiplas TPCs que são chips que agrupam múltiplos SMs (*Streaming Multiprocessors*). Cada TPC também contém um motor raster, responsável pela rasterização, que é o processo de converter uma imagem vetorial em uma imagem raster (formada por *pixels* ou pontos). De forma mais simplificada, este chip é arquitetado de forma a encapsular todos os principais componentes de processamento gráfico juntos.
- **SM (*Streaming Multiprocessors*)**: São processadores projetados para executar milhares de *threads* simultaneamente. Para gerenciar uma quantidade tão grande de *threads*, cada SM consiste em um barramento de cache L1 dedicado com seus respectivos núcleos associados e uma memória compartilhada (*shared memory*) cache L2, que armazenam os processamentos antes de extrair os dados de uma DRAM global GDDR (*Graphics Double Data Rate*), que são memórias específicas para gráficos. Temos também dentro de cada SM um conjunto de unidades de hardware responsáveis por executar instruções específicas de processamento, como SP (*Streaming Processors* ou *CUDA cores*), SFUs (*Special Function Units*) e LD/ST (*Load/Store*).
- **SP (*Streaming Processors*)**: Os SP ou *CUDA cores* são processadores dedicados para executar cálculos de lógica aritmética de 32-bits (ALU ou *Arithmetic-Logic Unit*) e cálculos de ponto flutuante de 64-bits (FPU ou *Floating Point Unit*).
- **SFU (*Special Function Units*)**: Os SFU são unidades de hardware responsáveis por executar instruções transcendentais, como, por exemplo, seno, cosseno e raiz quadrada.
- **LD/ST (*Load/Store*)**: LD/ST são unidades responsáveis em ler e escrever nas unidades de memória global.

Dado essa estrutura básica, temos que as GPUs andam avançando sua tecnologia e aumentando cada vez mais seu poder computacional, como o número de transistores, SPs, largura de banda, quantidade de memória, entre outras características. Não iremos entrar nos detalhes de cada arquitetura existente, porém, podemos citar algumas delas: Tesla, Fermi, Kepler, Maxwell, Pascal, Volta e Turing.

3.3 Modelos de programação paralela com GPUs

A complexidade arquitetural de uma GPU implica também em uma maior complexidade no desenvolvimento de aplicações. É fácil perceber que sem um modelo de programação projetado especificadamente para abstrair esse conceito de paralelismo torna inviável a implementação de soluções eficientes e com alto desempenho.

GPUs em sua natureza arquitetural são processadores *stream* (grande conjunto ordenado de dados), ou seja, podem operar em paralelo, executando um kernel em muitos registros em um fluxo de uma só vez (GPGU, 2020). Existem diversas linguagens de programação em GPUs, porém, as duas principais são CUDA (*Compute Unified Device Architecture*) e OpenCL (*Open Computing Language*).

Tomando como exemplo o CUDA temos que, de acordo com Resios (2011), o conceito principal é particionar o problema em subproblemas que podem ser resolvidos de forma independente através de blocos de threads paralelas e, portanto, cada subproblema dividido em partes ainda menores que podem ser resolvidos em paralelo pelos threads desses blocos. Sendo esses blocos independentes um dos outros, conseguimos obter boa escalabilidade.

Podemos citar algumas áreas que se beneficiam dos modelos de programação paralela, como: clusters HPC utilizando MPI (*Message Passing Interface*), processamento de imagem digital, computação geométrica, computação científica utilizada para previsão do tempo, modelagem molecular, astrofísica e também áreas de estudo mais recentes como redes neurais artificiais, aprendizado de máquina, aprendizado profundo e inteligência artificial.

3.4 Considerações finais

É imprescindível entender que o maior salto tecnológico que temos até o momento depende e muito das tecnologias envolvidas no processamento de computação paralela utilizando GPUs. Dessa forma, este capítulo nos ajuda a compreender um pouco mais sobre o seu funcionamento e dá uma base de conhecimento para avançarmos para os próximos tópicos, que é um estudo de modelos de predição de desempenho com base de dados de execuções de testes em GPUs.

4 MODELOS DE PREDIÇÃO EM CIÊNCIA DE DADOS

O mundo moderno atual vive um gigantesco impacto da transformação digital aplicada em milhares de serviços e que tem facilitado muito o dia-a-dia das pessoas, tanto em aspectos de comunicação, pesquisa, prestação de serviços, autoatendimento, compras, vendas, entre outros. Todo esse mecanismo envolve dados que são monitorados através de diversas plataformas, como, por exemplo, rastreios de navegação em websites, informações de localização em celulares, relógios inteligentes com coleta de monitoração de atividades físicas, casas inteligentes que coletam hábitos de moradia, enfim, temos uma infinidade de dados disponíveis prontos para serem coletados e analisados. Por trás desses dados, podemos encontrar respostas escondidas para questões nunca antes imaginadas e quem sabe também, não as utilizar para prever um comportamento ou antecipar um problema?

4.1 Introdução

Um cientista de dados utiliza ferramentas como a estatística, a matemática e a computação para extrair conhecimento de dados desorganizados, transformando-os em conhecimento e isso pode ser aplicado em praticamente qualquer área existente nos dias de hoje. As aplicações são praticamente ilimitadas, tanto para desvendar uma tendência, mapear um comportamento, prever um evento, treinar uma máquina para resolver problemas, entre outros.

A definição para previsão em estatística é o processo de estimativas em situações de incertezas (PREVISÃO, 2020). Baseado neste conceito, verificamos também a definição de estimação como sendo o processo de atribuição de um valor a um parâmetro, para o qual não se conhece o valor absoluto (ESTIMAÇÃO, 2020).

Sendo assim, podemos considerar que a partir da observação de uma amostra dos dados, podemos inferir ou estimar um parâmetro que se aproxime de seu valor real, dada uma certa probabilidade de erro ou acerto. O fato de podermos ter essa noção dos acontecimentos que nos cercam, faz da previsão estatística uma ferramenta poderosa.

4.2 Algoritmos de aprendizado de máquina

Durante todo o desenvolvimento de pesquisa realizado para a elaboração deste trabalho, as principais metodologias aplicadas para predição de desempenho, envolvem a modelagem analítica. Os passos, de forma geral, envolvem um processo de instrumentação de código, modelagem de *pipelines* para cada trecho do algoritmo estudado e criação de equações que modelem os tempos de resposta. Essa abordagem, apesar de eficiente, é muito custosa, ao ponto de que ela, na maioria das vezes, é limitada para ser aplicada apenas na arquitetura atual modelada e a cada mudança, é necessário rever todo o processo, muitas vezes inviabilizando o projeto.

Nesse aspecto, nossa intenção no trabalho é apresentar uma metodologia de modelagem de desempenho que simplifique e diminua o custo desse processo. Para isso, os algoritmos de aprendizado de máquina têm a vantagem de não necessitarem de conhecimento específico da aplicação em que se pretende modelar, pois ele olha apenas para os dados gerados, analisando suas distribuições, correlações, entre outras métricas estatísticas. Além disso, qualquer mudança arquitetural do sistema, pode ser facilmente adaptada no modelo, gerando uma nova base de dados para treinamento.

No que segue, serão apresentadas breves descrições dos principais algoritmos e métodos utilizados neste trabalho.

4.2.1 KNN (*K Nearest Neighbor* ou K Vizinhos Mais Próximos)

É um algoritmo que pode ser usado tanto para classificação quanto para regressão. Ele é do tipo não paramétrico, ou seja, ele não considera que as relações entre a entrada e a saída dos dados possam seguir algum tipo de função matemática. Outra característica é ele ser do tipo *lazy* (preguiçoso) que é um método onde o modelo é gerado apenas no momento em que os dados são inseridos e requisitado seu processamento.

Em Luz (2019) é descrito o funcionamento do algoritmo KNN na qual existe uma variável K que direcionará a quantidade de vizinhos mais próximos da amostra em que desejamos realizar a classificação.

A figura 4.1 ilustra o processo de funcionamento do KNN:

Figura 4.1 – Processo de funcionamento do KNN.

Fonte: Luz (2019).

Exemplificando um K=3, iremos calcular a distância dos 3 vizinhos mais próximos do ponto em que desejamos predizer a classe. Para identificar a classe do ponto desejado, é realizada uma votação onde a maioria dos pontos irá dizer qual a classe o ponto pertence. No exemplo apresentado na figura 4.1, o ponto em questão seria classificado como pertencente ao grupo B.

4.2.2 SVM (*Support Vector Machine* ou Máquina de Vetores)

É um algoritmo que pode ser usado tanto em classificação quanto em regressão e é uma ferramenta poderosa devido à sua tendência de não se ajustar excessivamente aos dados, mantendo um bom desempenho em muitos casos.

O objetivo é encontrar um hiperplano que separa os dados em suas classes em potencial. O hiperplano deve ser posicionado na distância máxima dos hiperplanos e os dados que estão localizados com a menor distância desse hiperplano são chamados de vetores de suporte. (MARIUS, 2020).

A figura 4.2, ilustra o mecanismo de classificação do SVM com a representação dos hiperplanos e dos vetores de suporte:

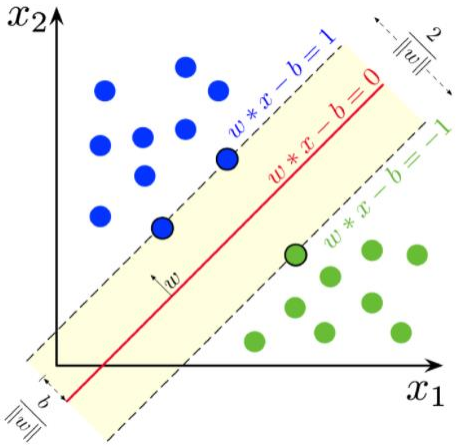

Figura 4.2 – Representação do hiperplano e dos vetores de suporte para classificação SVM.

Fonte: Marius (2020).

O SVM utiliza funções de kernel (por exemplo, linear, *radial basis function* ou RBF, polinomial e sigmoid) que conseguem se adaptar melhor a certos tipos de dados gerando uma melhor representatividade dos hiperplanos. A figura 4.3 ilustra os tipos de kernel mais populares e a aparência dos hiperplanos para um mesmo conjunto de dados.

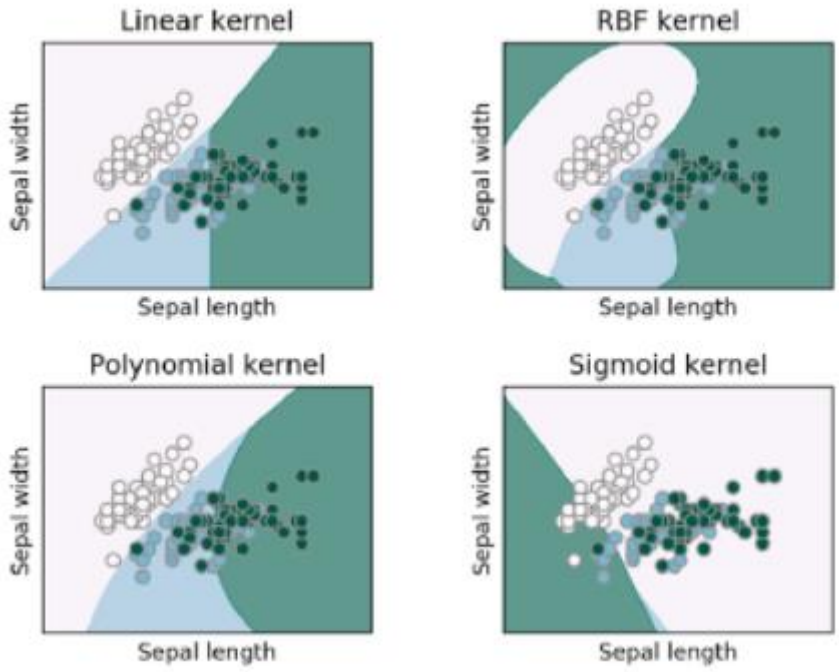

Figura 4.3 – Tipos de kernel SVM e diferença entre seus hiperplanos

Fonte: Marius (2020).

4.2.3 DT (*Decision Tree* ou Árvore de Decisão)

Uma árvore de decisão usa uma estrutura em formato de árvore (estruturas de dados formados por um conjunto de elementos que armazenam informações chamados nós) para representar um número de possíveis caminhos de decisão e apresentam um resultado para cada caminho. (GRUS, 2016). A figura 4.4 a seguir ilustra um exemplo de uma árvore de decisão para o problema de "adivinhe o animal":

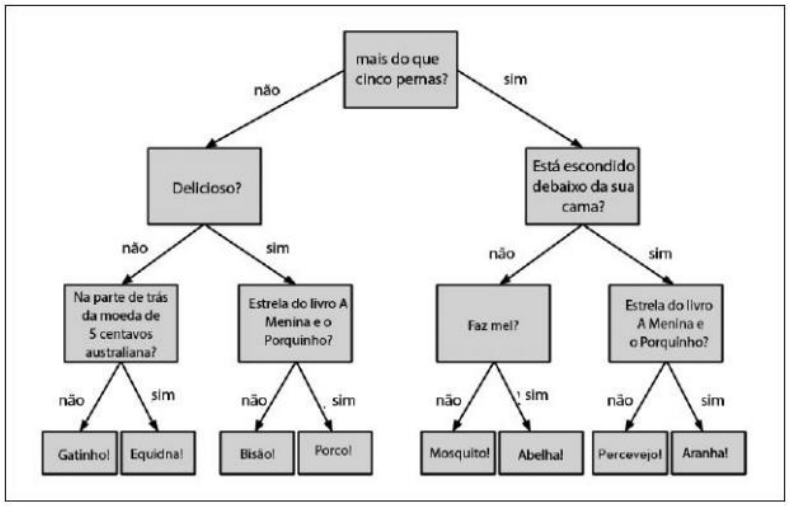

Figura 4.4 – Exemplo de uma árvore de decisão para o problema "adivinhe o animal".

Fonte: Grus (2016).

Esses algoritmos são supervisionados não-paramétricos e podem ser usados tanto para regressão quanto para classificação. Algumas vantagens é que eles são fáceis de interpretar e lidam bem com uma mistura de atributos numéricos e categóricos.

Conforme descrito em Campos (2017) temos que para construir uma árvore de decisão deve-se particionar o espaço recursivamente em retângulos (sub-regiões), levando em consideração uma variável por vez, nas quais um modelo simples é aprendido. Para encontrarmos o melhor ponto de corte ou divisão dos nós são utilizadas algumas medidas de impureza como, por exemplo, a entropia e o índice gini, que são definidos pelas equações 4.1 e 4.2 a seguir:

$$entropia\ (R) = -\sum p(c \mid R) \log\left(p(c \mid R)\right) \qquad (4.1)$$

$$gini\ (R) = \sum p(c \mid R)\ (1 - \ p(c \mid R)) \qquad (4.2)$$

Fonte: Campos (2017).

onde p (c | R) é a probabilidade de um ponto na região de corte R pertencer a classe c.

Assim sendo a cada pergunta feita (ou nós da árvore) pode-se calcular as impurezas e determinar com qual método obteve-se o melhor ganho de informação. Após todos esses passos, só resta a definição de um critério de parada, como por exemplo, atingir um maior grau de pureza ou uma profundidade máxima pré-estabelecida.

Ainda de acordo com Campos (2017), algumas vantagens e desvantagens das árvores de decisão são:

Vantagens:

- fácil interpretação;
- pouco esforço na preparação dos dados;
- lidar com problemas de múltiplos rótulos.

Desvantagens:

- uma profundidade muito elevada da árvore pode gerar *overfitting* dos dados;
- modelo muito sensível a uma pequena variação nos dados de treinamento;
- não garante a construção da melhor estrutura para os dados de treino.

4.2.4 Naive Bayes

É um classificador probabilístico que tem como base o teorema de Bayes que descreve a probabilidade de um evento, baseado em um conhecimento *a priori* que pode estar relacionado ao evento. (TEOREMA DE BAYES, 2020)

A definição formal é dada pela equação 4.3:

$$P(A \mid B) = \frac{P(B \mid A) \,.\, P(A)}{P(B)} \qquad (4.3)$$

Fonte: Teorema de Bayes (2020).

em que A e B são eventos e P(B) ≠ 0.

Em ciência de dados esse algoritmo é muito usado em problemas de classificação e se tornou popular principalmente em problemas de categorização de textos baseado na frequência

de palavras utilizadas. Por ser muito simples e rápido é muito usado para previsões em tempo real e ele recebe o nome de *naive* (ingênuo) por desconsiderar completamente a correlação entre as variáveis (features) do problema.

4.2.5 Ensemble

Um classificador ensemble é basicamente um processo que usa múltiplos modelos de aprendizado de máquina de forma combinada com a intenção de obter algum benefício nos seus resultados, como por exemplo, um melhor desempenho na predição dos dados (ENSEMBLE LEARNING, 2020).

Algumas estratégias mais comuns de ensembles são o *bootstrap aggregating* ou *bagging, boosting* e *stacking.*

Em Aniceto (2017) diz que no *bagging* a idéia principal é gerar um conjunto de dados por amostragem *bootstrap* dos dados originais, ou seja, vai gerando conjuntos de dados selecionando exemplos aleatórios do conjunto original por re-amostragem e usa um algoritmo base para modelar esses dados. Após esse processo, ele roda novamente o algoritmo base, só que dessa vez, montando um novo conjunto de dados, definido de forma aleatória com o mesmo conjunto original de treinamento. Por fim, para cada estimador, ele obtém um resultado diferente e usa um método de agregação para definir o resultado final, como por exemplo, o resultado que for mais abundante na votação individual de cada estimador.

Em Silva (2020) diz que no *boosting* diferentemente do *bagging* os classificadores fracos são treinados com um conjunto de dados de forma sequencial, onde um modelo base depende dos demais e no final são combinados. O resultado do primeiro modelo apresenta um resíduo, que é a distância entre o que foi previsto e o valor real. O modelo seguinte é ajustado em cima do resíduo do modelo anterior e recalculado. Isso vai se repetindo até que a distância entre o valor previsto e o valor real seja o menor possível. A figura 4.5 ilustra o funcionamento do algoritmo de boosting:

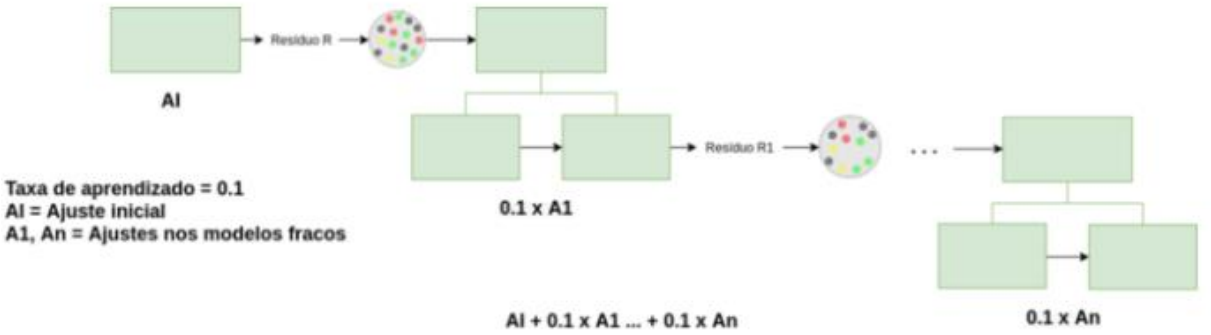

Figura 4.5 – Exemplo de funcionamento do algoritmo de boosting.

Fonte: Silva (2020).

Em Dutta (2019) diz que *stacking* visa atacar o mesmo problema com diferentes tipos de modelo que são capazes de aprender uma parte do todo. Com os resultados obtidos desses diversos modelos é construída uma predição intermediária então é adicionado um novo modelo que aprenderá a partir das predições intermediárias. Ao final do processo, diz-se que o modelo final está empilhado sobre os demais, podendo assim obter um desempenho que é melhor do que os modelos tratados de maneira individual. A figura 4.6 ilustra o funcionamento do algoritmo de *stacking*:

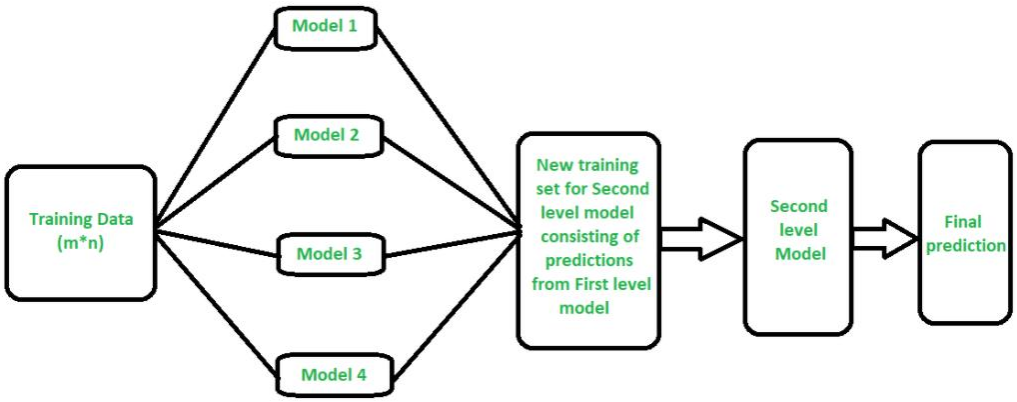

Figura 4.6 – Exemplo de funcionamento do algoritmo de stacking.

Fonte: Dutta (2019).

4.2.6 Redes Neurais Perceptron e MLP (Multi-Layer Perceptron)

Uma rede neural artificial é um modelo preditivo baseado no funcionamento do cérebro humano. Ele utiliza-se de neurônios artificiais, onde cada um deles olha para a saída de outros neurônios que os alimentam, faz um cálculo e então dispara um sinal, dependendo se o cálculo exceder, por exemplo, algum limite. (GRUS, 2016).

A figura 4.7 ilustra o modelo de um neurônio artificial, onde cada sinal de entrada xi é multiplicado pelo peso wi e então conectado ao neurônio que é um combinador linear responsável por somar os valores de entrada (x1 . w1 + x2 . w2 + x3 . w3) gerando um potencial de ativação u. Esse potencial u passa por uma função de ativação (exemplos: sigmoidal, tangente hiperbólica, relu, leaky relu, etc.) que através da função g(u) , calcula o sinal da saída do neurônio, ativando-o ou não.

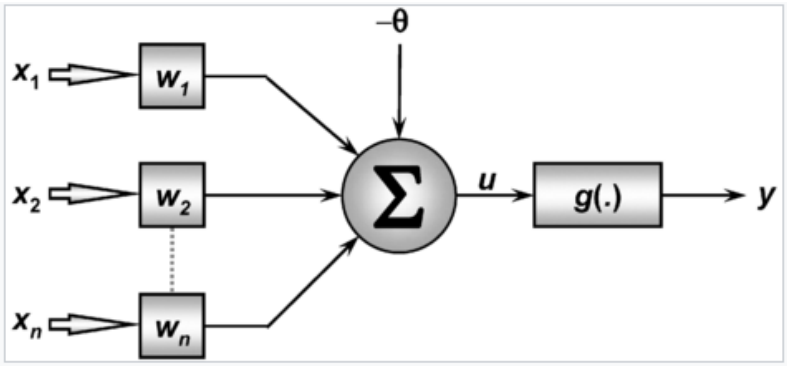

Figura 4.7 – Modelo de um neurônio artificial.

Fonte: Perceptron Multicamadas (2020).

O Perceptron é a rede neural mais simples que existe, sendo basicamente um classificador binário que computa a soma ponderada de suas entradas e ativa se essa soma for zero ou maior. Uma limitação é que o Perceptron só é capaz de solucionar problemas que são linearmente separáveis.

A definição formal é dada pela equação 4.4:

$$f(x) = \begin{cases} 1 \; if \;\; w.x + b \geq 0 \\ 0 \; else \end{cases} \tag{4.4}$$

Fonte: Perceptron (2020).

onde w é um vetor de peso real, o w.x é o produto escalar (que computa a soma dos pesos) e o b é o viés, um termo constante que não depende de qualquer valor de entrada.

O MLP ou Perceptron Multicamadas é também uma rede neural, parecida com o Perceptron, porém, composta por mais do que uma camada de neurônios, gerando assim mais de uma reta classificadora. O aprendizado é dado pelo algoritmo de *back propagation* ou retro propagação do erro. Em linhas gerais, o *back propagation* consiste na inicialização dos dados com pesos aleatórios, computando as ativações de todos os neurônios (processamento direto) e calculando o erro, calculando os novos pesos para cada neurônio da rede no sentido reverso (da saída para entrada – processamento reverso) e por fim, testando o critério de parada, se o critério for satisfeito o algoritmo termina, senão ele repete desde a etapa do processamento direto. (PERCEPTRON MULTICAMADAS, 2020). A figura 4.8 ilustra o mecanismo de *back propagation*:

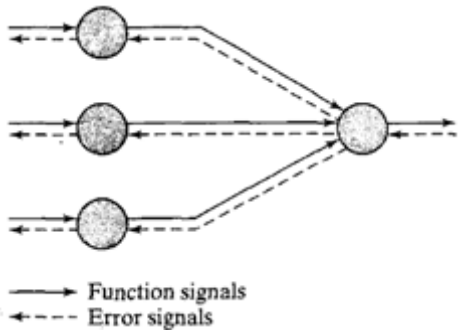

Figura 4.8 – Modelo do algoritmo de back propagation.

Fonte: Perceptron Multicamadas (2020).

4.3 Considerações finais

É imprescindível entender o potencial benefício que as técnicas de aprendizado de máquina e redes neurais podem trazer para o nosso dia-a-dia. Dessa forma, este capítulo nos ajuda a compreender um pouco mais sobre os principais conceitos e algoritmos utilizados em ciência de dados e dá base para avançarmos para o capítulo onde colocamos esses conceitos em prova, aplicando-os em uma base de dados reais para predição de desempenho computacional de GPUs.

5 IMPLEMENTAÇÃO, RESULTADOS E DISCUSSÃO

Uma vez descrito toda a parte conceitual necessária para a compreensão prática deste trabalho, iremos agora apresentar o problema levantado, as bases de estudo, as técnicas de ciências de dados aplicadas, os resultados obtidos, assim como as devidas discussões e aprendizados adquiridos.

5.1 Introdução

A concepção inicial do tema de estudo está relacionada diretamente à área de interesse profissional do autor, que lida diariamente com avalições de desempenho de sistemas de missão crítica. Dado a complexidade em encontrar bases consistentes para estudo, onde seja possível aplicar conceitos de aprendizado de máquina e aprendizado profundo, conseguimos obter um conjunto de dados referente as medições de desempenho com o tempo de execução de um kernel GPU SGEMM (*Single precision GEneral Matrix Multiply*) para multiplicação de matrizes. O repositório oficial da base de dados pode ser encontrado em (DUA, GRAFF, 2019).

A proposta de contribuição deste trabalho é poder aplicar abordagens de aprendizado de máquina para conseguir obter um modelo para classificar e predizer o desempenho do sistema de multiplicação de matrizes sem a necessidade de realizarmos uma análise profunda ou obter um conhecimento específico do sistema em questão. Neste caso, queremos mostrar que é possível obter uma boa acurácia sem requerer de uma análise estática do código ou mesmo de criar um modelo analítico específico.

5.2 Descrição do problema

Implementar modelos de aprendizado de máquina para prever o tempo de processamento de uma GPU aplicada a um sistema de multiplicação de matrizes SGEMM.

Para tal problema iremos abordar técnicas de classificação como, por exemplo, SVM (*Support Vector Machines* ou máquina de vetores), K-NN (*K-Nearest Neighbors* ou k-vizinhos mais próximos), DT (*Decision Tree* ou árvore de decisão), NB (Naive Bayes), redes neurais artificiais, entre outros.

Dessa forma, pretendemos realizar alguns experimentos, como por exemplo, criar novos modelos de classificação por meio de *ensembles* e por fim apurar e realizar a comparação dos modelos, verificando se de fato conseguimos obter bons resultados de predição apenas com algoritmos de aprendizado de máquina e redes neurais.

5.3 Descrição das atividades realizadas

Iremos agora abordar todos os passos implementados para a resolução do problema proposto, desde a concepção da base, passando pela preparação e tratamento dos dados, experimentos, as metodologias, análises realizadas e por fim os resultados obtidos.

5.3.1 Coleta e descrição dos dados

Esta base de dados foi obtida diretamente do repositório oficial de aprendizado de máquina da UCI (*University of California Irvine*) disponível em (DUA, GRAFF, 2019).

Conforme descrito em Dua e Graff (2019), este conjunto de dados mede o tempo de execução de um sistema de multiplicação de matrizes A * B = C, onde todas as matrizes tem tamanho 2048 X 2048 usando um *kernel* de GPU SGEMM parametrizável com 241.600 combinações de possíveis parâmetros. Para cada combinação testada, quatro medições de desempenho foram realizadas e seus resultados são relatados nas 4 últimas colunas. Todos os tempos estão medidos em milissegundos. Existem 14 parâmetros, onde os 10 primeiros são ordinais e as quatro últimas são variáveis binárias. Das 1.327.104 combinações de parâmetros totais, apenas 241.600 são viáveis, devido a várias restrições do *kernel*, e este conjunto contém o resultado de todas essas combinações viáveis. O experimento (obtenção dos dados) foi realizado em uma estação de trabalho rodando Linux Ubuntu v.16.04 com processador Intel Core i5 (3.5 GHz), 16 GB de memória RAM, uma GPU NVIDIA GeForce GTX 680 4GB e uma GeForce GTX 580 1.5GB. *Kernel* utilizado foi o ‘gemm_fast’ da biblioteca OpenCL ‘CLTune’.

A tabela 5.1 apresenta uma pequena amostra da base de dados, com seus parâmetros, número de linhas e número de colunas:

Tabela 5.1 – Amostragem da base de dados com medições de tempo de uma GPU kernel SGEMM

Número de linhas e colunas: (241600, 18)

	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	STRM	STRN	SA	SB	Run1 (ms)	Run2 (ms)	Run3 (ms)	Run4 (ms)
0	16	16	16	8	8	8	8	2	1	1	0	0	0	0	115.26	115.87	118.55	115.80
1	16	16	16	8	8	8	8	2	1	1	0	0	0	1	78.13	78.25	79.25	79.19
2	16	16	16	8	8	8	8	2	1	1	0	0	1	0	79.84	80.69	80.76	80.97
3	16	16	16	8	8	8	8	2	1	1	0	0	1	1	84.32	89.90	86.75	85.58
4	16	16	16	8	8	8	8	2	1	1	0	1	0	0	115.13	121.98	122.73	114.81

Fonte: tabela elaborada pelo autor.

A figura 5.1 apresenta os parâmetros relacionadas aos atributos preditivos do conjunto de dados da multiplicação de matrizes:

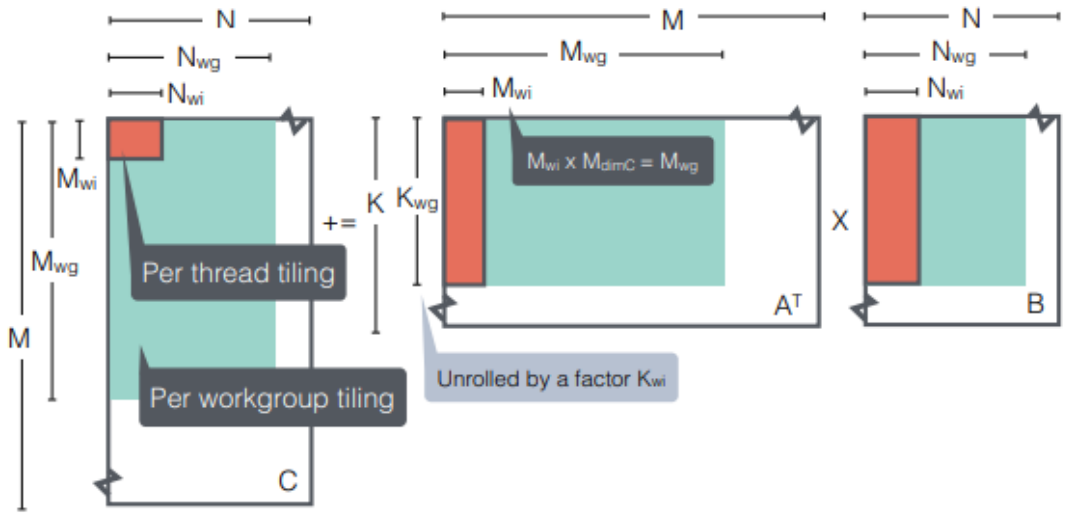

Figura 5.1 – Parâmetros da base de dados de multiplicação de matrizes

Fonte: Nugteren, Codreanu (2015, p.201).

Descrição dos parâmetros:

- **MWG, NWG e KWG**: mosaico 2D aplicado em um *workgroup* de *threads*. Esses parâmetros correspondem as dimensões das matrizes M, N e K respectivamente.
- **MDIMC, NDIMC**: parâmetros ajustáveis que definem o tamanho dos *workgroups* de *threads*. A unidade desses *workgroups* está definida na figura 5.1 como MWI e NWI, que são basicamente o tamanho do mosaico por *thread*. Eles são definidos pela seguinte relação: MWI = MWG/MDIMC e NWI = NWG/NDIMC. A sigla WI é uma abreviação de *work item*, ou seja, corresponde a um *thread*, considerando que estamos implementando um mosaico 2D à nível de *threads*.

- **SA, SB:** esses parâmetros binários são usados para armazenar o *workgroup* em memória. Quando eles estão setados para 1 = sim, a memória local é habilitada.
- **MDIMA, NDIMB**: parâmetros que são usados quando a memória local está ativada. Sendo assim elas podem ser ajustadas da seguinte forma: MDIMC x NDIMC = MDIMA x KDIMA = KDIMB x NDIMB. Portanto, MDIMA e NDIMB são parâmetros extras de *tunning* e o KDIMA e KDIMB são calculados de acordo com a equação apresentada.
- **STRM, STRN:** esses parâmetros binários são usados quando queremos habilitar ou desabilitar o acesso de um *thread* a memória externa (*off-chip memory*) do *workgroup* de *threads* definido. STRM é usada para a matriz A e C e STRN para a matriz B. STR vem do termo *stride* (passo). Se estiver habilitado o *stride* é ajustado para MDIMA e NDIMB, caso contrário ele é ajustado para 1 (*no stride*).
- **VWM, VWN:** a largura do vetor (VW: *vector width*) para carregamento e armazenamento de dados podem ser configuradas por estes dois parâmetros. VWM para as matrizes A e C e VWN para a matriz B.
- **KWI:** um fator onde é possível controlar o *kernel-loop* do KWG, fazendo um *loop-unrolling.* O *loop-unrolling* é basicamente uma técnica de transformação de um *loop* que tenta otimizar a velocidade de um programa em relação ao seu tamanho binário, esta é uma abordagem conhecida como compensação espaço-tempo. (LOOP UNROLLING, 2020)

A tabela 5.2 apresenta uma descrição estatística da base de dados para todos os seus parâmetros:

Tabela 5.2 – Estatística descritiva da base de dados

	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM
count	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000
mean	80.415364	80.415364	25.513113	13.935894	13.935894	17.371126	17.371126	5.000000	2.448609
std	42.469220	42.469220	7.855619	7.873662	7.873662	9.389418	9.389418	3.000006	1.953759
min	16.000000	16.000000	16.000000	8.000000	8.000000	8.000000	8.000000	2.000000	1.000000
25%	32.000000	32.000000	16.000000	8.000000	8.000000	8.000000	8.000000	2.000000	1.000000
50%	64.000000	64.000000	32.000000	8.000000	8.000000	16.000000	16.000000	5.000000	2.000000
75%	128.000000	128.000000	32.000000	16.000000	16.000000	32.000000	32.000000	8.000000	4.000000
max	128.000000	128.000000	32.000000	32.000000	32.000000	32.000000	32.000000	8.000000	8.000000

VWN	STRM	STRN	SA	SB	Run1 (ms)	Run2 (ms)	Run3 (ms)	Run4 (ms)
241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000	241600.000000
2.448609	0.500000	0.500000	0.500000	0.500000	217.647852	217.579536	217.532756	217.527669
1.953759	0.500001	0.500001	0.500001	0.500001	369.012422	368.677309	368.655118	368.677413
1.000000	0.000000	0.000000	0.000000	0.000000	13.290000	13.250000	13.360000	13.370000
1.000000	0.000000	0.000000	0.000000	0.000000	40.660000	40.710000	40.660000	40.640000
2.000000	0.500000	0.500000	0.500000	0.500000	69.825000	69.930000	69.790000	69.820000
4.000000	1.000000	1.000000	1.000000	1.000000	228.530000	228.310000	228.320000	228.320000
8.000000	1.000000	1.000000	1.000000	1.000000	3339.630000	3375.420000	3397.080000	3361.710000

Fonte: tabela elaborada pelo autor.

5.3.2 Preparação dos dados

Nesta seção vamos descrever toda a metodologia aplicada na preparação dos dados, realizando os devidos tratamentos para aplicar os algoritmos de aprendizado de máquina.

Considerando que a base de dados é relativamente grande, o primeiro trabalho realizado foi de limitar o seu uso para que as simulações e experimentos pudessem ser viáveis para execução. Algumas tentativas frustradas de executar os modelos com a massa completa levaram mais de 10h de duração sem finalização e, por conta disso, o volume foi reduzido em 50% dos dados de forma aleatória. O experimento (processamento dos modelos) foi realizado em uma estação de trabalho rodando Windows 10 Pro com processador Intel Core i7-4500U (1.8 GHz), 12 GB de memória RAM e uma GPU NVIDIA GeForce GT 750M.

A tabela 5.3 apresenta uma amostragem dos dados que foram efetivamente utilizados para o processamento dos modelos de aprendizado de máquina. O tamanho original de 241.600 linhas foi reduzido para 120.800, mantendo todas as variáveis preditoras e variáveis alvo.

Tabela 5.3 – Amostragem da base de dados com 50% do total das medições

Número de linhas e colunas: (120800, 18)

	MWG	NWG	KWG	MDIMC	NDIMC	MDIMA	NDIMB	KWI	VWM	VWN	STRM	STRN	SA	SB	Run1 (ms)	Run2 (ms)	Run3 (ms)	Run4 (ms)
172756	128	64	16	8	16	8	16	2	2	4	0	1	0	0	325.74	329.21	329.49	328.71
19971	32	16	16	16	8	8	8	8	1	1	0	0	1	1	79.09	79.06	79.10	77.22
45592	32	64	32	32	8	8	16	2	1	2	1	0	0	0	62.10	61.83	62.44	61.40
227329	128	128	32	8	32	32	16	8	2	1	0	0	0	1	172.85	175.17	175.38	175.13
239256	128	128	32	32	8	32	8	8	4	2	1	0	0	0	233.98	234.08	234.03	234.42

Fonte: tabela elaborada pelo autor.

O primeiro passo no tratamento dos dados foi verificar duplicidade e dados faltantes, porém, nada foi identificado. A seguir, foi verificado a ocorrência de *outliers* nas variáveis alvo de tempo de execução. A figura 5.2 apresenta o gráfico *boxplot* com os outliers identificados:

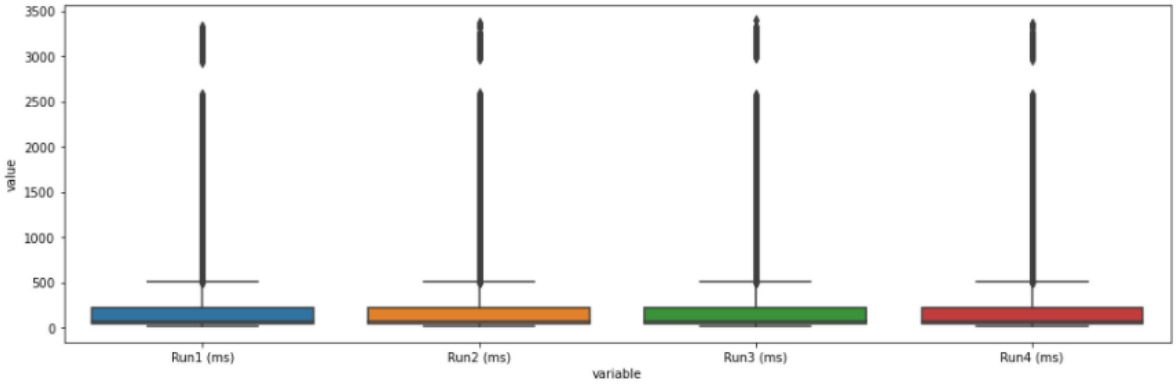

Figura 5.2 – Gráfico de boxplot com os outliers para os tempos de execução

Fonte: figura elaborada pelo autor.

Neste caso, a metodologia utilizada foi do IQR (*InterQuartile Range* ou intervalo interquartil) para a identificação dos *outliers* e foi optado pela estratégia de remoção desses dados. Após aplicar esta metodologia, tivemos uma redução na base de dados de 120.800 para 76.569 linhas. A figura 5.3 ilustra os gráficos de boxplot após remoção de *outliers*:

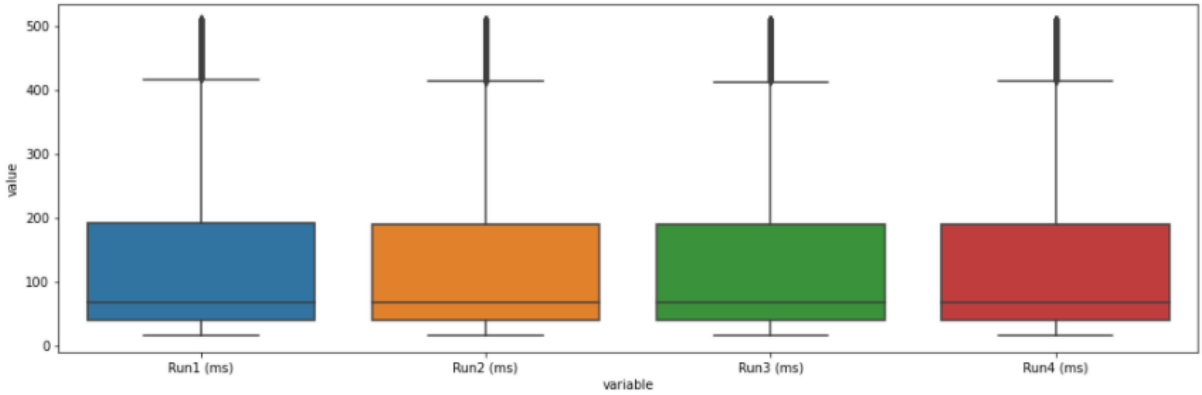

Figura 5.3 – Gráfico de boxplot com os outliers removidos para os tempos de execução

Fonte: figura elaborada pelo autor.

Foi verificado a correlação existentes entre as variáveis e pôr fim a distribuição das variáveis alvo de tempo de execução. A figura 5.4 apresenta a matriz de correlação no formato mapa de calor, entre todos os atributos da base e a figura 5.6 apresenta o histograma com a distribuição dos dados:

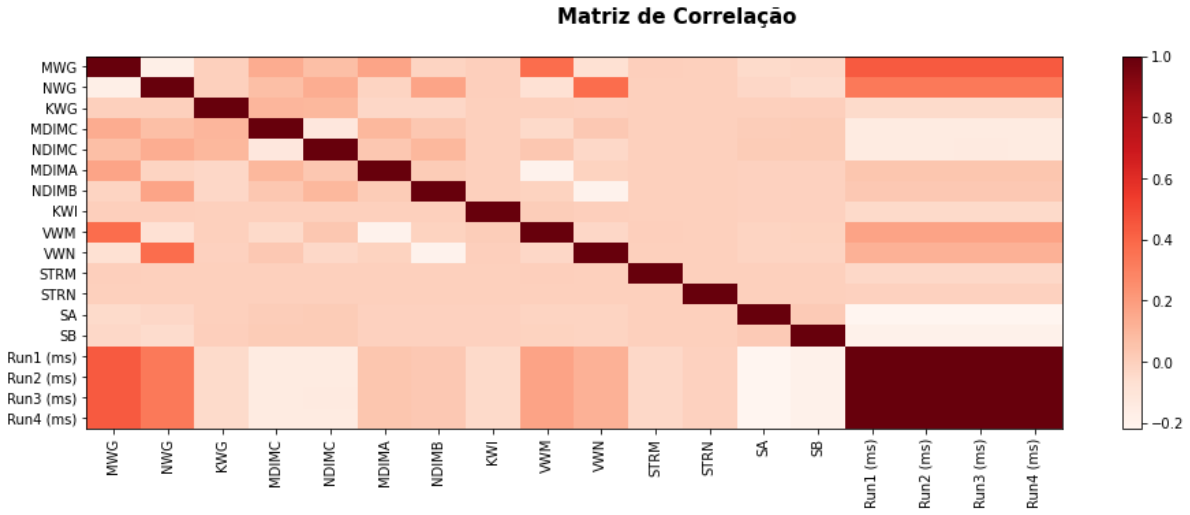

Figura 5.4 – Matriz de correlação entre todos os atributos da base de dados

Fonte: figura elaborada pelo autor.

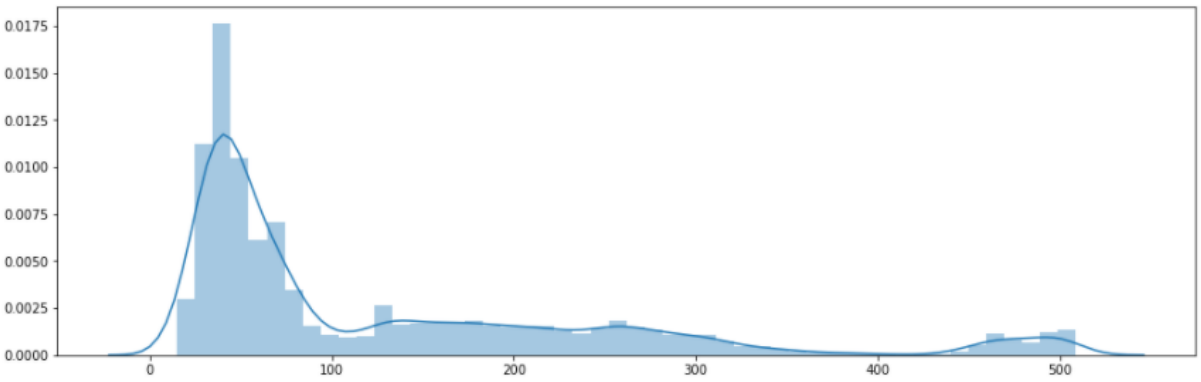

Figura 5.6 – Histograma com a distribuição dos dados de tempo de execução

Fonte: figura elaborada pelo autor.

Com a existência de quatro atributos alvo (testes executados), foi necessário realizar a consolidação em apenas uma coluna do *dataframe* renomeada de Run (ms). Ela é basicamente a média simples consolidada das outras quatro execuções.

Na metodologia utilizada o trabalho foi executado como sendo um problema de classificação e, por conta disso, foi necessário realizar a discretização dos dados, transformando a coluna Run (ms) em variáveis categóricas. Foram adicionadas três novas colunas ao *dataframe*, sendo elas o Run Discretizado (ms), Run Discretizado Mapping e o Run Binário.

Para a criação do Run Discretizado (ms) foi criado *thresholds* e aplicados na coluna Run (ms). Os intervalos foram obtidos de forma empírica chegando na seguinte distribuição de grupos de intervalos em milissegundos: [0, 20], [20, 40], [40, 60], [60, 80], [80, 100], [100,

200], [200, 300], [300, 400], [400, 500] e [500, 600], totalizando 10 classes. A figura 5.7 ilustra o histograma dos dados da coluna Run Discretizado (ms) e a tabela 5.4 apresenta a distribuição dos dados categorizados:

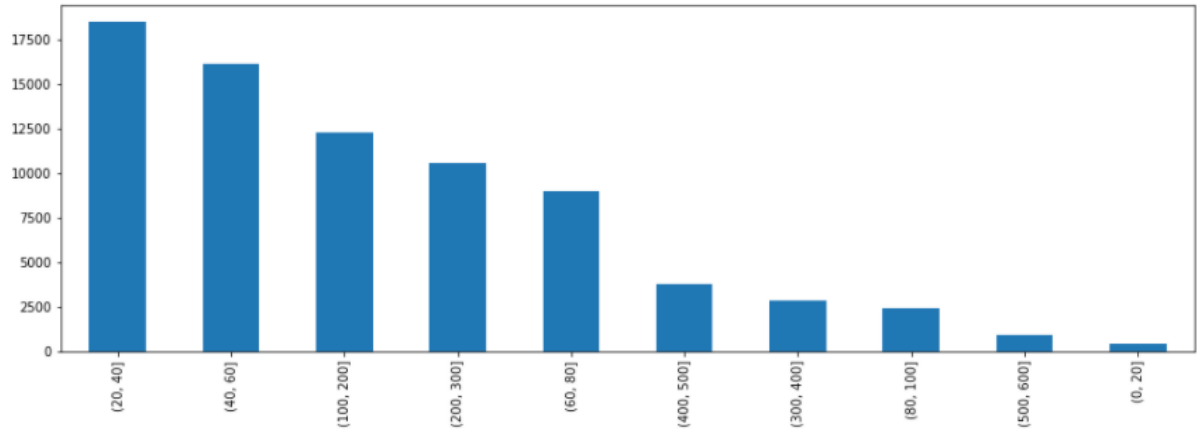

Figura 5.7 – Histograma com a distribuição dos dados categorizadas do Run Discretizado (ms)

Fonte: figura elaborada pelo autor.

Tabela 5.4 – Distribuição das classes categorizadas da coluna Run Discretizado (ms)

```
Distribuição das classes:
[000,020]: 0.52 %
[020,040]: 24.15 %
[040,060]: 21.00 %
[060,080]: 11.67 %
[000,100]: 3.17 %
[100,200]: 15.99 %
[200,300]: 13.77 %
[300,400]: 3.70  %
[400,500]: 4.86  %
[500,600]: 1.17  %
```

Fonte: tabela elaborada pelo autor.

A coluna Run Discretizado Mapping foi criada com o intuito de facilitar a aplicação dos modelos de aprendizado de máquina. Ela nada mais é do que a identificação dos intervalos criados no Run Discretizado (ms) para números inteiros. As 10 classes criadas foram de 0 até 9, sendo: [0, 20] – classe 0, [20, 40] – classe 1, [40, 60] – classe 2, [60, 80] – classe 3, [80, 100] – classe 4, [100, 200] – classe 5, [200, 300] – classe 6, [300, 400] – classe 7, [400, 500] – classe 8 e [500, 600] – classe 9. A figura 5.8 ilustra o histograma dos dados da coluna Run Discretizado Mapping:

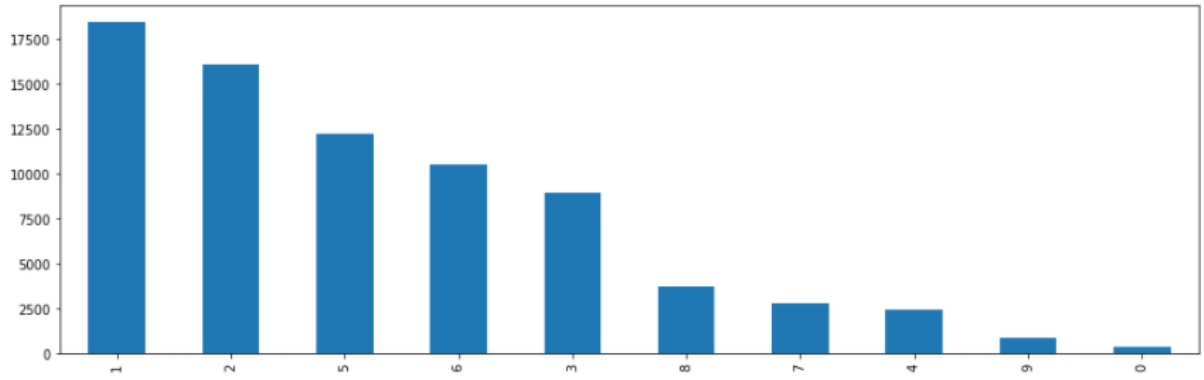

Figura 5.8 – Histograma com a distribuição dos dados categorizadas do Run Discretizado Mapping

Fonte: figura elaborada pelo autor.

Por último, a coluna Run Binário foi criada para a comparação e estudo da execução dos modelos, simplificando o problema em classes binárias. Vale considerar que o autor não considera esse tipo de abordagem adequada do ponto de vista do negócio e da natureza do problema em questão, pois com a classificação binária perde-se muito da informação relevante da análise de desempenho de sistemas de missão crítica. Uma das utilidades principais dessa modelagem é poder prever as condições em que os sistemas podem sofrer atrasos ou mesmo necessidade de recapacitação por conta de uma mudança no ambiente ou requisito novo, portanto, a abordagem multiclasse, contribui mais nesse sentido do que a classificação binária, porém, considerando que a experimentação e o estudo dos métodos também é objetivo deste trabalho, foi criada essa coluna no *dataframe*, sendo que valores menores ou iguais a média geral do Run (ms) foram classificados como 0 e valores maiores que a média geral foram classificados como 1. A figura 5.9 ilustra o histograma dos dados da coluna Run Binário e a tabela 5.5 apresenta a distribuição dos dados categorizados de forma binária:

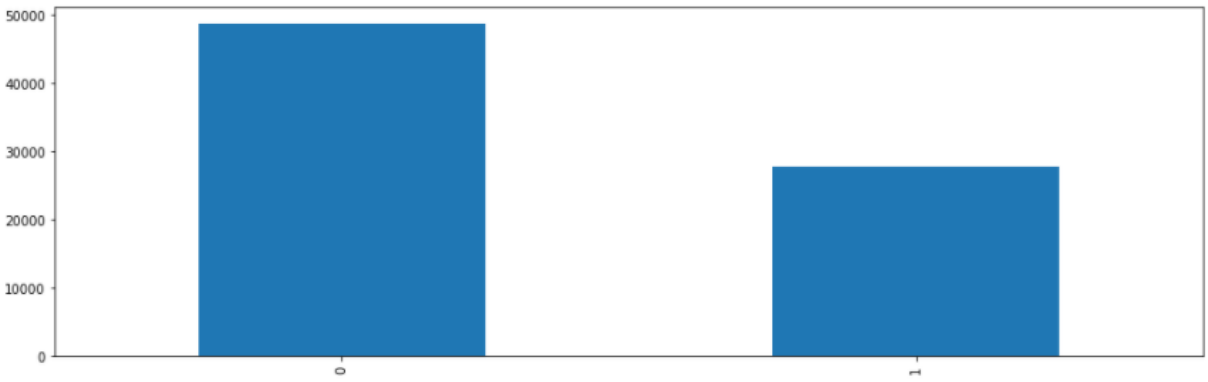

Figura 5.9 – Histograma com a distribuição dos dados categorizadas do Run Binário

Fonte: figura elaborada pelo autor.

Tabela 5.5 – Distribuição das classes categorizadas da coluna Run Binário

```
Distribuição das classes:
 <= Média: 63.68 %
 >  Média: 36.32 %
```

Fonte: tabela elaborada pelo autor.

Em linhas gerais, esses foram os tratamentos realizados na base de dados com o objetivo de prepará-los para a aplicação dos experimentos. Como próximos passos iremos começar a desenvolver os modelos de predição baseados nessa evolução realizada.

5.3.3 Experimentos e análises de resultados

A primeira etapa dos experimentos foi dividir a base de dados em um conjunto de treinamento e teste. Para isso, foi escolhida a proporção de 80-20, sendo 80% dos dados reservados para treinamento e 20% dos dados para testes. Foram criadas bases de treinamento e teste para as colunas Run Discretizado Mapping e Run Binário. O tamanho final da base de treinamento foi de 61.255 linhas e da base de teste 15.314 linhas.

Antes de iniciar a aplicação dos modelos de aprendizado de máquina, foi realizada uma tentativa de otimização na distribuição dos dados. A técnica escolhida para tal finalidade foi a aplicação do SMOTE (*Synthetic Minority Oversampling Technique*) nas três abordagens: *oversampling*, *undersampling* e *overundersampling.* A técnica de SMOTE basicamente tenta equalizar a quantidade de classes desbalanceadas com exemplos sintéticos, seguindo uma das estratégias citadas, sendo que o *oversampling* tende a aumentar as observações das classes minoritárias, o *undersampling* tende a diminuir as observações das classes majoritárias e o *overundersampling* faz uma mistura das outras duas estratégias. Após ser aplicada as três estratégias (código fonte pode ser encontrado no Apêndice A), é necessário avaliar se alguma delas foi melhor avaliada dentro de um critério único. Neste caso, foi realizado uma modelagem simples do tipo SVM para as quatro distribuições de dados: dados originais, dados com oversampling, dados com undersampling e dados com overundersampling. A figura 5.10 apresenta a matriz de confusão para os resultados obtidos:

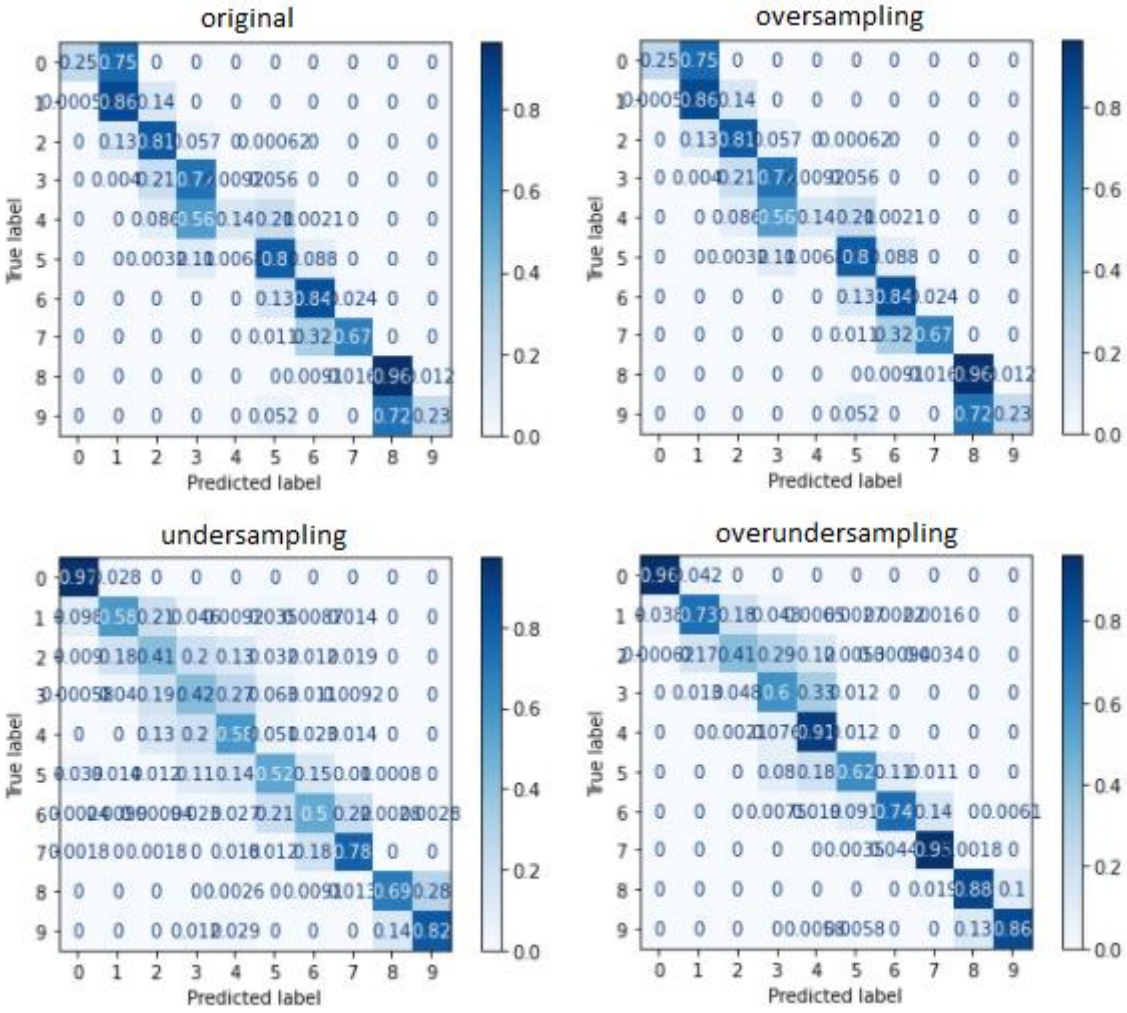

Figura 5.10 – Matriz de confusão dos dados aplicada técnicas de SMOTE

Fonte: figura elaborada pelo autor.

É possível observar que as técnicas de undersampling e overundersampling obtiveram uma matriz pior em comparação com os dados originais. Já o oversampling aparentemente está igual, porém, para questões de desempate, foi avaliada as métricas do modelo SVM com os dados com SMOTE com relação ao *precision ou* precisão, *recall ou* recuperação, *f1-score* e acurácia.

Apenas para contextualizar as métricas de validação dos modelos, temos que a acurácia é a média global do acerto do modelo aos classificar as classes, ou seja, é uma relação entre a quantidade de acertos da matriz de confusão (verdadeiros positivos + verdadeiros negativos) com o total de classificações (verdadeiros positivos + verdadeiros negativos + falsos positivos + falsos negativos). O *precision*, também conhecido como valor predito positivo, traz a informação de quantas observações o modelo classificou corretamente como verdadeiro

positivos dentre os elementos classificados. O *recall*, também conhecido como sensibilidade, mostra quantos elementos relevantes (variável desejada) foram realmente recuperadas no modelo, ou seja, de todos os elementos relevantes disponíveis, quantos ele conseguiu classificar. O *f1-score* nada mais é do que uma média harmônica entre o recall e o precision, resumindo a informação das duas métricas. As tabelas 5.6 e 5.7 apresentam os resultados obtidos para os dados originais e para os dados com SMOTE *oversampling*:

Tabela 5.6 – Avaliação do modelo SVM base com dados originais

Resultados com o balanceamento original:

	precision	recall	f1-score	support
0	0.90	0.25	0.39	72
1	0.87	0.86	0.87	3689
2	0.74	0.81	0.77	3205
3	0.63	0.72	0.67	1739
4	0.68	0.14	0.23	487
5	0.80	0.80	0.80	2491
6	0.81	0.84	0.83	2121
7	0.86	0.67	0.75	566
8	0.86	0.96	0.91	772
9	0.81	0.23	0.35	172
accuracy			0.79	15314
macro avg	0.80	0.63	0.66	15314
weighted avg	0.79	0.79	0.78	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.7 – Avaliação do modelo SVM base com dados SMOTE oversampling

Resultados com o balanceamento Over Sampling:

	precision	recall	f1-score	support
0	0.52	0.92	0.66	72
1	0.89	0.83	0.86	3689
2	0.77	0.76	0.77	3205
3	0.69	0.73	0.71	1739
4	0.40	0.82	0.54	487
5	0.86	0.75	0.80	2491
6	0.87	0.82	0.84	2121
7	0.75	0.89	0.82	566
8	0.96	0.90	0.93	772
9	0.66	0.83	0.73	172
accuracy			0.79	15314
macro avg	0.74	0.82	0.77	15314
weighted avg	0.81	0.79	0.80	15314

Fonte: tabela elaborada pelo autor.

Após a análise de ambos os modelos, foi decidido utilizar os dados de SMOTE com balanceamento oversampling. A justificativa é que mesmo ambos apresentando uma acurácia igual, o modelo com oversampling conseguiu aumentar o *recall*, crescimento de 0.63 para 0.82, com isso, é possível atingir um maior número de elementos relevantes para a análise dos dados.

A seguir são apresentados os dados de forma visual, para entendermos melhor como eles estão distribuídos e agrupados no plano. A figura 5.11 apresenta os dados originais discretizados com *mapping number*, a figura 5.12 apresenta os dados com classificação binária e a figura 5.13 apresenta a nova distribuição com SMOTE oversampling que será utilizada nos modelos multiclasse.

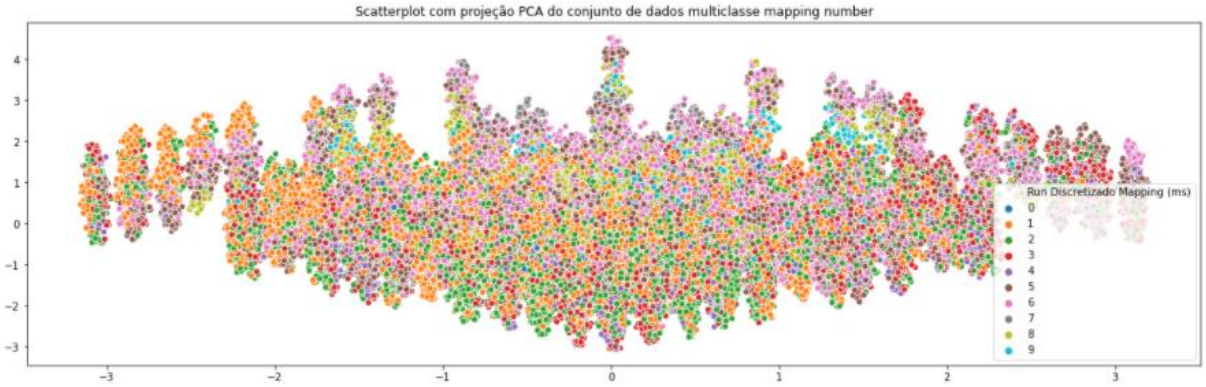

Figura 5.11 – Scatterplot com os dados originais Discretizado Mapping Number

Fonte: figura elaborada pelo autor.

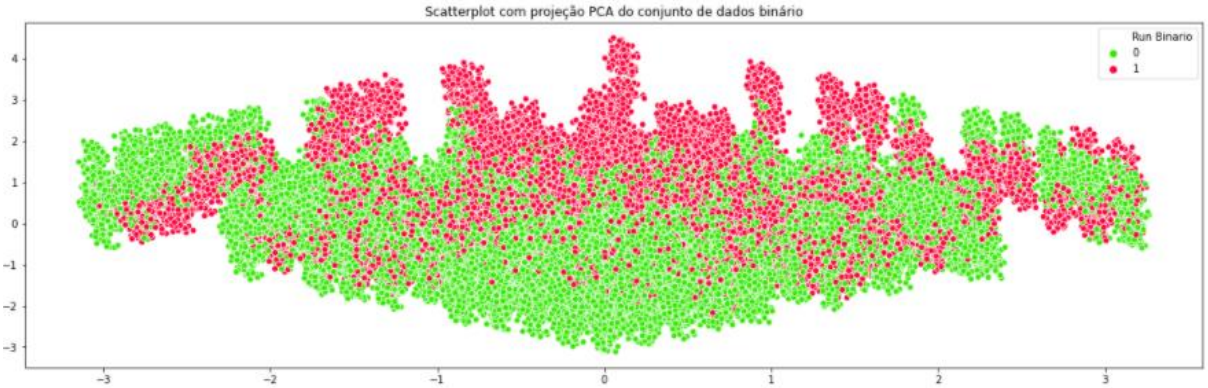

Figura 5.12 – Scatterplot com os dados originais Discretizado Binário

Fonte: figura elaborada pelo autor.

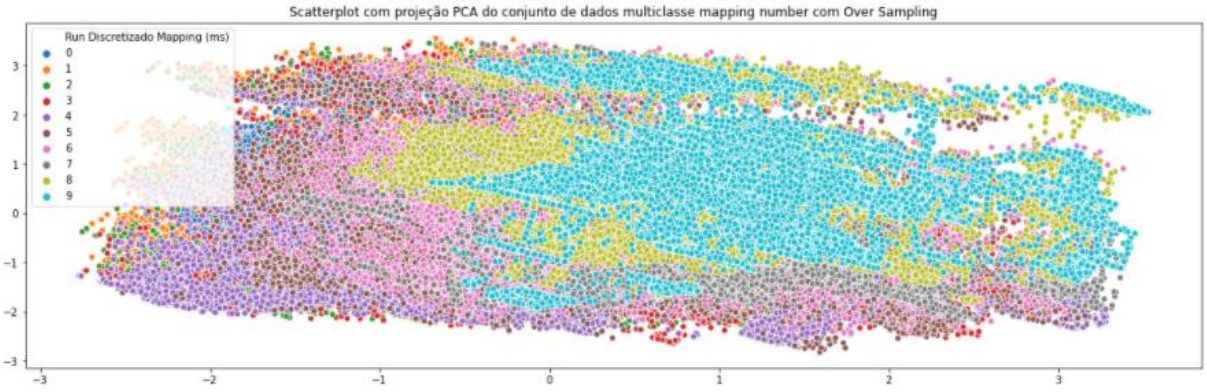

Figura 5.13 – Scatterplot com os dados balanceados usando SMOTE oversampling

Fonte: figura elaborada pelo autor.

É possível notar claramente na figura 5.13 o aumento das classes minoritárias quando em comparação com os dados originais. A seguir, a segunda etapa dos experimentos refere-se à aplicação das estratégias de algoritmos de aprendizado de máquina e redes neurais.

A primeira estratégia avaliada foi o KNN (K-vizinhos mais próximos). Como a idéia desse algoritmo é simplesmente classificar as amostras com relação a distância dos seus K vizinhos mais próximos, foi adotada uma estratégia de simular o melhor valor de K a ser aplicado na base completa de treinamento. Para isso, foi definida uma amostra menor do conjunto de treinamento (código disponível no Apêndice A), pois essa simulação implica na execução do modelo com diferentes valores de K, sendo que realizar este tipo de avaliação com todo o volume de treinamento seria inviável. Essa abordagem foi executada apenas com o intuito de tentar otimizar o tempo gasto na identificação dos parâmetros experimentais do modelo e os resultados são apresentados na figura 5.14, onde o melhor valor para K foi 7.

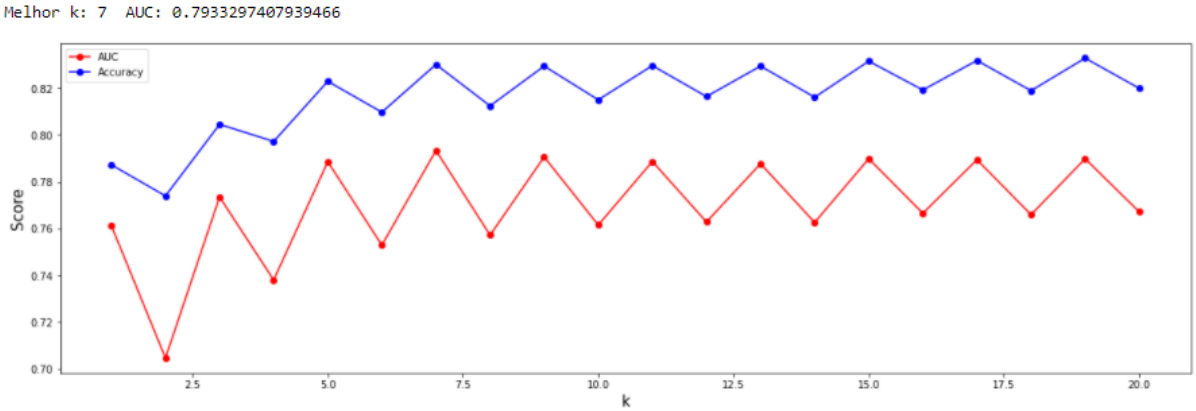

Figura 5.14 – Simulação do melhor parâmetro K para o modelo KNN

Fonte: figura elaborada pelo autor.

Os modelos treinados para KNN foram para multiclasse com oversampling e binário, ambos usando métrica euclidiana como cálculo da distância. As tabelas 5.8 e 5.9 apresentam os resultados obtidos para os dados com *oversampling e* binário:

Tabela 5.8 – Avaliação do modelo KNN com dados SMOTE oversampling

	precision	recall	f1-score	support
0	0.21	0.75	0.33	72
1	0.69	0.57	0.63	3689
2	0.50	0.39	0.44	3205
3	0.32	0.40	0.36	1739
4	0.16	0.41	0.23	487
5	0.63	0.52	0.57	2491
6	0.59	0.52	0.56	2121
7	0.39	0.68	0.49	566
8	0.73	0.82	0.77	772
9	0.42	0.66	0.51	172
accuracy			0.51	15314
macro avg	0.46	0.57	0.49	15314
weighted avg	0.55	0.51	0.52	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.9 – Avaliação do modelo KNN com dados binários

	precision	recall	f1-score	support
0	0.88	0.92	0.90	9679
1	0.85	0.79	0.82	5635
accuracy			0.87	15314
macro avg	0.87	0.85	0.86	15314
weighted avg	0.87	0.87	0.87	15314

Fonte: tabela elaborada pelo autor.

Conseguimos acurácia de 0.51 para o modelo multiclasse. É possível observar que a classe 4 recebeu o pior score de todos com apenas 0.23. No geral, é um modelo que não apresentou muita eficiência na predição, ficando com uma precisão e um recall não tão significativos. Para efeitos de comparação, vale a pena mencionar o ganho obtido com o modelo de classificação binária, indo para 0.87 de acurácia. Com isso, é possível verificar que o algoritmo trabalha muito melhor com um sistema de classificação binária, apesar deste não ser muito útil para atender as necessidades do negócio em si, dada a natureza do problema.

A segunda estratégia avaliada foi o SVM (máquina de vetores). Seguindo a mesma estratégia de definir um melhor número de parâmetros, montamos uma simulação semelhante ao caso aplicado no KNN para identificar o melhor valor de C (código disponível no Apêndice A), parâmetro que serve para ajustar a rigidez ou a suavidade da classificação de margem grande. Essa abordagem foi executada apenas com o intuito de tentar otimizar o tempo gasto na identificação dos parâmetros experimentais do modelo e os resultados são apresentados na figura 5.15, onde o melhor valor para C foi 10.

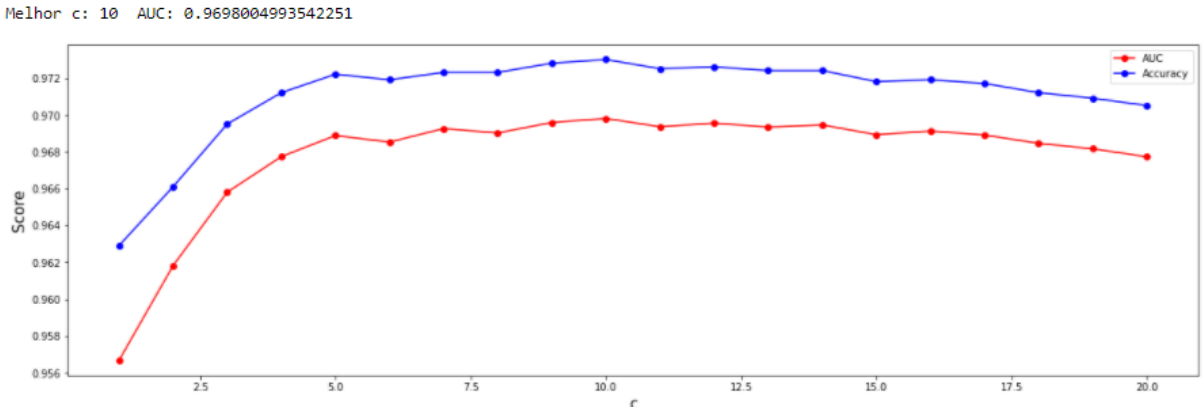

Figura 5.15 – Simulação do melhor parâmetro C para o modelo SVM

Fonte: figura elaborada pelo autor.

Os modelos treinados para SVM foram para multiclasse com oversampling e binário. Também foram avaliados dois tipos de kernel (linear e radial rbf), pois observando graficamente a distribuição dos dados, suspeitou-se que talvez um kernel radial poderia se aplicar melhor aos dados. As tabelas 5.10 e 5.11 apresentam os resultados obtidos para os dados com *oversampling* com kernel linear e radial e as tabelas 5.12 e 5.13 apresentam os resultados obtidos para os dados binários com kernel linear e radial:

Tabela 5.10 – Avaliação do modelo SVM com dados SMOTE oversampling kernel linear

	precision	recall	f1-score	support
0	0.11	0.93	0.19	72
1	0.66	0.46	0.54	3689
2	0.46	0.20	0.27	3205
3	0.28	0.41	0.33	1739
4	0.09	0.38	0.15	487
5	0.55	0.58	0.56	2491
6	0.60	0.46	0.52	2121
7	0.39	0.65	0.49	566
8	0.91	0.61	0.73	772
9	0.30	0.68	0.41	172
accuracy			0.43	15314
macro avg	0.43	0.54	0.42	15314
weighted avg	0.52	0.43	0.45	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.11 – Avaliação do modelo SVM com dados SMOTE oversampling kernel radial rbf

	precision	recall	f1-score	support
0	0.64	0.79	0.71	72
1	0.92	0.89	0.90	3689
2	0.84	0.83	0.84	3205
3	0.80	0.84	0.82	1739
4	0.59	0.81	0.68	487
5	0.91	0.88	0.89	2491
6	0.90	0.89	0.89	2121
7	0.86	0.90	0.88	566
8	0.96	0.93	0.95	772
9	0.76	0.83	0.80	172
accuracy			0.87	15314
macro avg	0.82	0.86	0.84	15314
weighted avg	0.87	0.87	0.87	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.12 – Avaliação do modelo SVM com dados binários kernel linear

	precision	recall	f1-score	support
0	0.96	0.95	0.95	9679
1	0.91	0.93	0.92	5635
accuracy			0.94	15314
macro avg	0.94	0.94	0.94	15314
weighted avg	0.94	0.94	0.94	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.13 – Avaliação do modelo SVM com dados binários kernel radial rbf

	precision	recall	f1-score	support
0	0.99	0.99	0.99	9679
1	0.98	0.98	0.98	5635
accuracy			0.99	15314
macro avg	0.99	0.99	0.99	15314
weighted avg	0.99	0.99	0.99	15314

Fonte: tabela elaborada pelo autor.

Estes foram os modelos que individualmente apresentaram os melhores resultados. De fato, confirma-se a suspeita de que um kernel radial se aplicaria melhor os dados, sendo que houve um ganho de acurácia de 0.43 para 0.87. Neste caso também, é possível confirmar as vantagens de modelagem frente à classificação binária, chegando a acurácia de 0.99 para kernel radial.

A terceira estratégia avaliada foi o DT (árvore de decisão). Diferentemente do KNN e SVM, nela não é necessário identificar um melhor parâmetro para otimização, porém, é necessário definir o critério de cálculo das impurezas. Os modelos treinados para DT foram para multiclasse com oversampling e binário, utilizando os critérios de impureza com entropia e gini. As tabelas 5.14 e 5.15 apresentam os resultados obtidos para os dados com *oversampling* com critério de entropia e gini e as tabelas 5.16 e 5.17 apresentam os resultados obtidos para os dados binários com critério de entropia e gini:

Tabela 5.14 – Avaliação do modelo DT com dados SMOTE oversampling critério entropia

	precision	recall	f1-score	support
0	0.20	0.68	0.31	72
1	0.61	0.41	0.49	3689
2	0.40	0.43	0.41	3205
3	0.31	0.58	0.41	1739
4	0.22	0.63	0.33	487
5	0.62	0.59	0.60	2491
6	0.46	0.29	0.36	2121
7	0.27	0.28	0.27	566
8	0.83	0.15	0.26	772
9	0.09	0.04	0.06	172
accuracy			0.43	15314
macro avg	0.40	0.41	0.35	15314
weighted avg	0.49	0.43	0.43	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.15 – Avaliação do modelo DT com dados SMOTE oversampling critério gini

	precision	recall	f1-score	support
0	0.25	0.65	0.36	72
1	0.86	0.53	0.66	3689
2	0.49	0.40	0.44	3205
3	0.34	0.59	0.43	1739
4	0.21	0.73	0.32	487
5	0.77	0.73	0.75	2491
6	0.85	0.69	0.77	2121
7	0.79	0.71	0.75	566
8	0.98	0.85	0.91	772
9	0.56	0.92	0.70	172
accuracy			0.60	15314
macro avg	0.61	0.68	0.61	15314
weighted avg	0.68	0.60	0.62	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.16 – Avaliação do modelo DT com dados binários critério entropia

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9679
1	1.00	0.99	0.99	5635
accuracy			1.00	15314
macro avg	1.00	1.00	1.00	15314
weighted avg	1.00	1.00	1.00	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.17 – Avaliação do modelo DT com dados binários critério gini

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9679
1	0.99	0.99	0.99	5635
accuracy			1.00	15314
macro avg	1.00	0.99	1.00	15314
weighted avg	1.00	1.00	1.00	15314

Fonte: tabela elaborada pelo autor.

Estes modelos DT apresentaram um desempenho inferior ao SVM, porém, um ponto positivo é a facilidade de implementação dos mesmos. Os resultados pelo critério de gini apresentaram melhores acurácias, indo de 0.43 para 0.60 na média. Com relação aos dados binários, o DT demonstrou até melhor que o SVM, obtendo 100% de acerto na base de testes.

A quarta implementação aplicada foi o Naive Bayes. Considerando que ele é um modelo muito rápido e simples, que não leva em consideração as correlações entre as variáveis preditoras, não foi gerado muitas expectativas com relação aos seus resultados. O principal valor nesse caso, foi validar uma maior quantidade de modelos distintos focado no aprendizado das técnicas. As tabelas 5.18 e 5.19 apresentam os resultados obtidos para os dados com *oversampling e* binário:

Tabela 5.18 – Avaliação do modelo NB com dados SMOTE oversampling

	precision	recall	f1-score	support
0	0.00	0.00	0.00	72
1	0.51	0.46	0.48	3689
2	0.41	0.33	0.37	3205
3	0.27	0.18	0.21	1739
4	0.08	0.49	0.14	487
5	0.44	0.28	0.34	2491
6	0.39	0.16	0.22	2121
7	0.19	0.47	0.27	566
8	0.46	0.35	0.40	772
9	0.11	0.63	0.19	172
accuracy			0.33	15314
macro avg	0.29	0.34	0.26	15314
weighted avg	0.40	0.33	0.34	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.19 – Avaliação do modelo NB com dados binários

	precision	recall	f1-score	support
0	0.79	0.88	0.83	9679
1	0.75	0.59	0.66	5635
accuracy			0.78	15314
macro avg	0.77	0.74	0.75	15314
weighted avg	0.77	0.78	0.77	15314

Fonte: tabela elaborada pelo autor.

Como esperado, este modelo apresentou uma acurácia de apenas 0.33 para multiclasse e 0.78 para binário.

A seguir foi feita uma primeira tentativa de obter um modelo aprimorado e, para isso, foi escolhido executar a técnica de *ensemble stacking*. O exemplo de funcionamento pode ser visto na figura 4.6, onde a idéia é atacar o problema utilizando vários tipos de modelos distintos, esperando assim obter algum resultado otimizado. Então, foi selecionado todos os principais modelos avaliados anteriormente para fazer parte desse ensemble (código disponível no Apêndice A), sendo eles KNN oversampling, SVM oversampling radial, DT oversampling gini e NB oversampling. Como modelo empilhado, foi utilizado regressão logística, que verificado

na literatura, é um dos mais adequados para problemas de classificação. As tabelas 5.20 e 5.21 apresentam os resultados obtidos para os dados com *oversampling e* binário:

Tabela 5.20 – Avaliação do modelo ensemble stacking com dados SMOTE oversampling

	precision	recall	f1-score	support
0	0.00	0.00	0.00	72
1	0.89	0.57	0.69	3689
2	0.54	0.50	0.52	3205
3	0.34	0.47	0.39	1739
4	0.10	0.37	0.16	487
5	0.64	0.70	0.67	2491
6	0.83	0.54	0.66	2121
7	0.47	0.72	0.57	566
8	0.96	0.71	0.82	772
9	0.45	0.85	0.59	172
accuracy			0.57	15314
macro avg	0.52	0.54	0.51	15314
weighted avg	0.66	0.57	0.59	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.21 – Avaliação do modelo ensemble stacking com dados binários

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9679
1	0.99	0.99	0.99	5635
accuracy			1.00	15314
macro avg	1.00	0.99	1.00	15314
weighted avg	1.00	1.00	1.00	15314

Fonte: tabela elaborada pelo autor.

Os resultados obtidos não foram tão satisfatórios, considerando que ele pode ter sofrido algum tipo de penalização vinda de modelos com desempenho pior. É possível observar, por exemplo, que para a classe 0, o modelo não conseguiu acertar nenhum resultado, igualmente aconteceu no modelo NB individual, então, possivelmente ele pode ter contribuído para a queda de resultados. No final de todas as execuções, será realizada uma nova tentativa de ensemble, porém, considerando apenas os dois melhores resultados obtidos nos modelos, para verificarmos novamente, se haverá algum tipo de ganho nesse processo.

Além dos algoritmos de aprendizado de máquina, também foi avaliado o comportamento da predição utilizando redes neurais. Para este caso, considerando o formato dos dados e o tempo disponível para finalização do trabalho, foi possível aplicar apenas duas estratégias de redes neurais, sendo elas o Perceptron e o MLP (Perceptron Multicamadas).

Para a aplicação do Perceptron foi utilizado o método de validação *K-fold* ou validação cruzada, que basicamente consiste em dividir o conjunto de dados em k subconjuntos de mesmo tamanho, sendo que um subconjunto é utilizado para testes e o restante para estimação dos parâmetros, obtendo assim medias mais robustas sobre a capacidade de predição do modelo. As tabelas 5.22 e 5.23 apresentam os resultados obtidos para os dados com *oversampling e* binário:

Tabela 5.22 – Avaliação do modelo Perceptron com dados SMOTE oversampling

	precision	recall	f1-score	support
0	0.06	0.71	0.12	72
1	0.32	0.13	0.19	3689
2	0.35	0.13	0.19	3205
3	0.16	0.23	0.19	1739
4	0.05	0.16	0.07	487
5	0.20	0.19	0.19	2491
6	0.26	0.30	0.28	2121
7	0.14	0.46	0.21	566
8	0.78	0.53	0.63	772
9	0.10	0.34	0.15	172
accuracy			0.21	15314
macro avg	0.24	0.32	0.22	15314
weighted avg	0.29	0.21	0.22	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.23 – Avaliação do modelo Perceptron com dados binários

	precision	recall	f1-score	support
0	0.93	0.81	0.87	9679
1	0.74	0.89	0.81	5635
accuracy			0.84	15314
macro avg	0.83	0.85	0.84	15314
weighted avg	0.86	0.84	0.84	15314

Fonte: tabela elaborada pelo autor.

É possível observar bem as limitações do Perceptron, sendo que de maneira geral, aplica-se mais em situações de classificação binária. Assim a acurácia obtida para multiclasse foi de apenas 0.21, enquanto que no problema binário a acurácia obtida foi de 0.84.

Com o objetivo de adequar melhor o problema de multiclasse utilizando redes neurais, foi aplicado o MLP ou Perceptron Multicamada, que justamente por usar mais do que uma camada de neurônios, consegue gerar mais retas classificatórias. A estratégia de validação-cruzada k-fold foi utilizada novamente e o número de iterações foi definida em 1000. Além disso, o otimizador foi o adam (valor padrão), que pela documentação, é mais adequado para um grande volume de dados de treinamento. As tabelas 5.24 e 5.25 apresentam os resultados obtidos para os dados com *oversampling e* binário:

Tabela 5.24 – Avaliação do modelo Perceptron multicamadas com dados SMOTE oversampling

	precision	recall	f1-score	support
0	0.64	0.85	0.73	72
1	0.92	0.84	0.88	3689
2	0.79	0.82	0.80	3205
3	0.80	0.77	0.78	1739
4	0.51	0.80	0.62	487
5	0.86	0.89	0.87	2491
6	0.88	0.81	0.85	2121
7	0.86	0.91	0.88	566
8	0.98	0.94	0.96	772
9	0.81	0.93	0.87	172
accuracy			0.84	15314
macro avg	0.80	0.86	0.82	15314
weighted avg	0.85	0.84	0.84	15314

Fonte: tabela elaborada pelo autor.

Tabela 5.25 – Avaliação do modelo Perceptron multicamadas com dados binários

	precision	recall	f1-score	support
0	0.99	0.99	0.99	9679
1	0.99	0.99	0.99	5635
accuracy			0.99	15314
macro avg	0.99	0.99	0.99	15314
weighted avg	0.99	0.99	0.99	15314

Fonte: tabela elaborada pelo autor.

Os resultados melhoraram consistentemente com relação ao que foi obtido no Perceptron. Tivemos um *precision* e um *recall* médio de 0.8, além de um acurácia de 0.84, ou seja, esta foi uma solução que atingiu resultados satisfatórios.

A última parte dos experimentos consiste em tentar otimizar os resultados obtidos, através do ensemble stacking com os dois melhores modelos gerados. Os melhores resultados foram para o algoritmo de aprendizado de máquina SVM com kernel radial e para a rede neural MLP. A tabela 5.26, coloca ambos os resultados lado a lado para comparação:

Tabela 5.26 – Resultados individuais dos dois melhores modelos (SVM e MLP)

SVM kernal radial

	precision	recall	f1-score	support
0	0.64	0.79	0.71	72
1	0.92	0.89	0.90	3689
2	0.84	0.83	0.84	3205
3	0.80	0.84	0.82	1739
4	0.59	0.81	0.68	487
5	0.91	0.88	0.89	2491
6	0.90	0.89	0.89	2121
7	0.86	0.90	0.88	566
8	0.96	0.93	0.95	772
9	0.76	0.83	0.80	172
accuracy			0.87	15314
macro avg	0.82	0.86	0.84	15314
weighted avg	0.87	0.87	0.87	15314

MLP otimizador adam

	precision	recall	f1-score	support
0	0.64	0.85	0.73	72
1	0.92	0.84	0.88	3689
2	0.79	0.82	0.80	3205
3	0.80	0.77	0.78	1739
4	0.51	0.80	0.62	487
5	0.86	0.89	0.87	2491
6	0.88	0.81	0.85	2121
7	0.86	0.91	0.88	566
8	0.98	0.94	0.96	772
9	0.81	0.93	0.87	172
accuracy			0.84	15314
macro avg	0.80	0.86	0.82	15314
weighted avg	0.85	0.84	0.84	15314

Fonte: tabela elaborada pelo autor.

Como a predição no geral está bem equilibrada e com valores aceitáveis, a idéia é verificar onde o algoritmo consegue gerar um ganho, nem que seja mínimo, para garantirmos as melhores métricas possíveis. As classes com piores precision são a 0 e 4, com valores em cerca de 0.6. Para o restante das métricas (recall e acurácia) todos os outros valores estão bem avaliados, sendo que quase todas as métricas ficam próximas de 80% para mais. A tabela 5.27 apresenta os resultados do modelo ensemble stacking para SVM e MLP:

Tabela 5.26 – Resultados individuais dos dois melhores modelos (SVM e MLP)

	precision	recall	f1-score	support
0	0.79	0.72	0.75	72
1	0.89	0.87	0.88	3689
2	0.81	0.80	0.81	3205
3	0.80	0.84	0.82	1739
4	0.59	0.81	0.68	487
5	0.89	0.87	0.88	2491
6	0.89	0.86	0.87	2121
7	0.86	0.89	0.88	566
8	0.96	0.96	0.96	772
9	0.87	0.86	0.87	172
accuracy			0.85	15314
macro avg	0.84	0.85	0.84	15314
weighted avg	0.86	0.85	0.85	15314

Fonte: tabela elaborada pelo autor.

É possível observar pequenos ganhos que trazem um pouco mais de confiabilidade nas predições. Foi possível verificar um ganho no precision da classe 0, indo para 0.79 contra 0.64 dos modelos individuais. No geral, o *precision* teve um aumento na média para 0.84 contra 0.82 do melhor modelo individual. Mesmo não tendo ganhos significativos de acurácia, o ensemble estabilizou o precision e o recall, fazendo uma junção dos dois modelos e equilibrando os pontos fracos de cada um.

5.4 Considerações finais

Os principais resultados já estão expostos e o capítulo a seguir apresenta opiniões pessoais, considerações do autor sobre todo o conteúdo apresentado e também possíveis trabalhos futuros relacionados ao tema.

6 CONCLUSÕES E TRABALHOS FUTUROS

6.1 Conclusões e trabalhos futuros

O cenário competitivo atual do mercado de tecnologia, traz um desafio enorme para algumas empresas: como manter seus sistemas de missão crítica com o melhor desempenho possível para atender um crescimento de demanda e alavancar seus negócios? Dado esse contexto, sabe-se que os testes não funcionais de software, como desempenho, capacidade, estabilidade, entre outros, são a escolha certa para lidar com esta questão, porém, nem sempre os prazos e custos envolvidos são atraentes para o negócio. Uma alternativa a este processo, são a criação de modelos de software que possam prever resultados de uma maneira mais rápida e a baixo custo. No geral, as soluções de modelagem são desenvolvidas por meio de modelos analíticos, onde é necessário um entendimento profundo das arquiteturas e da programação envolvida no sistema para que se possa criar equações robustas que representem seu comportamento com fidelidade. Um dos problemas da modelagem analítica é que o custo de manutenção é elevado, pois qualquer alteração, seja ela de requisito, hardware ou mesmo atualização de tecnologia, acarreta praticamente no desenvolvimento do zero de um novo modelo.

Este trabalho buscou explorar as técnicas de ciência de dados e aplicá-las no contexto de modelagem de predição de desempenho de sistemas. A principal vantagem de usar os algoritmos de aprendizado de máquina e redes neurais é justamente o fato de não precisarmos obter um conhecimento profundo da arquitetura do sistema em questão para poder desenvolver predições com um grau aceitável de confiabilidade.

Com relação ao tratamento dos dados, transformou-se as variáveis alvo (tempo de processamento em GPU) em um problema de classificação multiclasse e binário. Os resultados binários, serviram apenas para questões de estudo e também para verificar qual a diferença de ganho de predição dos algoritmos aplicados nesse tipo de problema simplificado. A grande questão de transformar uma análise de desempenho em problema binário é que ele não agrega muita informação relevante com relação ao comportamento do software avaliado.

A primeira otimização nos dados foi aplicar o SMOTE com *oversampling*. Essa técnica manteve a mesma acurácia quando feito um *benchmarking* com os dados originais, porém,

gerou um *recall* bem maior, crescimento médio de 0.63 para 0.82, com isso foi possível atingir uma maior quantidade de dados relevantes para dentro da modelagem.

Dentre os modelos de aprendizado de máquina o SVM com kernel radial apresentou a melhor taxa de acertos, com 0.87 de acurácia, média de 0.82 de *precision* e 0.86 de *recall*. Estes resultados mostraram que é possível obter uma predição muito apurada e precisa, com quase 90% de taxa de acerto, apenas aplicando algoritmos de ciência de dados.

Com relação as redes neurais, o modelo MLP também apresentou bons resultados, com 0.84 de acurácia, média de 0.80 de *precision* e 0.86 de *recall*. Praticamente tão bom quanto o algoritmo de aprendizado de máquina.

Por fim, um *ensemble stacking* com os dois melhores modelos foi executado e mesmo mantendo a acurácia de 0.85, constatou-se um ganho com relação a predição de algumas classes, além de melhor estabilizar os resultados de *precision* e *recall* como um todo, mostrando ser um modelo mais robusto e eficiente.

Para uma primeira análise de desempenho aplicando ciência de dados, este trabalho foi capaz de atender as expectativas, porém, abrimos aqui a possibilidade de alguns trabalhos futuros, como por exemplo:

- Propor uma nova metodologia de avaliação considerando aplicar modelos de regressão ao invés de classificação e averiguar as possíveis vantagens e desvantagens práticas entre as duas abordagens;
- Adaptar a metodologia e aplicar em outros contextos focados para melhorias de sistemas de TI, como por exemplo, modelo de classificação para predição de acessos indevidos focados em melhorias de segurança da informação e modelo de classificação de usabilidade do sistema, mapeando quais são as funcionalidades mais acessadas;
- Utilizar algoritmos de aprendizado profundo para o treinamento de modelos de inteligência artificial, podendo não apenas prever resultados de desempenho, mas também, apresentar sugestões de melhorias nos sistemas ou alertas de monitoração de forma proativa.

6.2 Considerações finais

De maneira geral foi possível aplicar um conhecimento sólido há muito tempo almejado dentro da minha carreira profissional e que a instituição conseguiu prover com muito esmero.

O curso de MBA em ciência de dados foi de extremo valor e contribuiu muito para o meu crescimento profissional, considerando que foi possível colocar em prática a maior parte dos conceitos aprendidos ao longo deste período, com uma temática atualizada e que corrobora muito com as necessidades e tendências do mercado de trabalho. Agradeço demais a dedicação de todos os professores, monitores e coordenadores que nos acompanharam e deram todo o apoio possível para conquistarmos mais essa etapa.

REFERÊNCIAS

ANICETO, M. **Classificadores Ensemble, tipos Bagging e Boosting**. 2017. Disponível em: https://lamfo-unb.github.io/2017/09/27/BaggingVsBoosting/. Acesso em: 19 dez. 2020

BALDINI I., FINK S. J., and ALTMAN E., **Predicting gpu performance from cpu runs using machine learning**. In Computer Architecture and High PerformanceComputing (SBAC-PAD), 2014 IEEE 26th International Symposium on, pages 254–261. doi:10.1109/SBAC-PAD.2014.30.

BALLESTER-RIPOLL, R., PAREDES, E. G., PAIROLA, R., **Sobol Tensor Trains for Global Sensitivity Analysis**. In arXiv Computer Science / Numerical Analysis e-prints, 2017

CAMPOS, R. **Árvores de Decisão.** 2017. Disponível em: https://medium.com/machine-learning-beyond-deep-learning/%C3%A1rvores-de-decis%C3%A3o-3f52f6420b69. Acesso em: 19 dez. 2020.

DONGARRA, J., FOSTER, I., FOX, G., GROPP, W., KENNEDY, K., TORCZON, L., WHITE, A., **Sourcebook Of Parallel Computing**. San Francisco: Morgan Kaufmann Publishers, 2003.

DUA, D., GRAFF, C., (2019). **UCI Machine Learning Repository** [http://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance]. Irvine, CA: University of California, School of Information and Computer Science. Acesso em: 05 out. 2020.

DUTTA, A. **Stacking in Machine Learning**. 2019. Disponível em: https://www.geeksforgeeks.org/stacking-in-machine-learning/. Acesso em: 19 dez. 2020.

ENSEMBLE LEARNING, 2020. Disponível em: https://pt.wikipedia.org/wiki/Estima%C3%A7%C3%A3o. Acesso em: 19 dez. 2020.

ESTIMAÇÃO, 2020. Disponível em: https://pt.wikipedia.org/wiki/Estima%C3%A7%C3%A3o. Acesso em: 19 dez. 2020.

FALCH, T. L., ELSTER, A. C., **Machine Learning Based Auto-Tuning for Enhanced OpenCL Performance Portability**. 2015 Ieee International Parallel And Distributed Processing Symposium Workshop, [S.L.], p. 1231-1240, maio 2015. IEEE. http://dx.doi.org/10.1109/ipdpsw.2015.85.

GONZÁLEZ, M. T. A., **Performance Prediction Of Applications Executed on GPUs using a Simple Analytical Model and Machine Learning Techniques**. 2018. 109 f. Tese (Doutorado) - Curso de Mathematics And Statistics, Institute Of Mathematics And Statistics Of University Of São Paulo, Universidade de São Paulo, São Paulo, 2018.

GPGPU, 2020. Disponível em: https://pt.wikipedia.org/wiki/GPGPU. Acesso em: 03 out. 2020.

GRUS, J. **Data Science do Zero**: Primeiras Regras com o Python / Joel Grus; traduzido por Welington Nascimento. - Rio de Janeiro: Alta Books, 336 f., 2016.

JAIN, R., **The Art of Computer Systems Performance Analysis:** techiniques for experimental desing, measurement, simulation and modeling. New York: John Wiley & Sons, 1991.

LOOP UNROLLING, 2020. Disponível em: https://en.wikipedia.org/wiki/Loop_unrolling. Acesso em: 05 out. 2020.

LUZ, F. **ALGORITMO KNN PARA CLASSIFICAÇÃO**. 2019. Disponível em: https://inferir.com.br/artigos/algoritimo-knn-para-classificacao/. Acesso em: 19 dez. 2020.

MARIUS, Hucker. **Multiclass Classification with Support Vector Machines (SVM), Dual Problem and Kernel Functions**. 2020. Disponível em: https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02. Acesso em: 19 dez. 2020.

NUGTEREN, C., CODREANU, V., **CLTune**: A Generic Auto-Tuner for OpenCL Kernels. In: MCSoC: 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip. IEEE, 2015

PERCEPTRON, 2020. Disponível em: https://pt.wikipedia.org/wiki/Perceptron. Acesso em: 19 dez. 2020.

PERCEPTRON MULTICAMADAS, 2020. Disponível em: https://pt.wikipedia.org/wiki/Perceptron_multicamadas. Acesso em: 19 dez. 2020.

PREVISÃO, 2020. Disponível em: https://pt.wikipedia.org/wiki/Previs%C3%A3o_(estat%C3%ADstica).Acesso em: 19 dez. 2020.

QAZDAR, A., ER-RAHA, B., CHERKAOUI, C., MAMMASS, D., **A machine learning algorithm framework for predicting students performance**: A case study of baccalaureate students in Marocco. Springer Science+Business Media, LLC, part of Springer Nature, 2019.

RESIOS, A., **GPU Performance Prediction using Parametrized Models**. 2011. 66 f. Tese (Doutorado) - Computer Science, Utrecht University, Utrecht, 2011.

SAS, **Análises Preditivas: o que são e qual sua importância?** SAS. Disponível em: https://www.sas.com/pt_br/insights/analytics/analises-preditivas.html#:~:text=An%C3%A1lises%20preditivas%20usam%20dados%2C%20algoritmos,que%20poder%C3%A1%20acontecer%20no%20futuro Acesso em: 26 Jun. 2020.

SILVA, J. **Uma breve introdução ao algoritmo de Machine Learning Gradient Boosting utilizando a biblioteca Scikit-Learn**. 2020. Disponível em: https://medium.com/equals-lab/uma-breve-introdu%C3%A7%C3%A3o-ao-algoritmo-de-machine-learning-gradient-boosting-utilizando-a-biblioteca-311285783099#:~:text=Na%20t%C3%A9cnica%20de%20Boosting%2C%20cada,combinados%20de%20uma%20maneira%20determin%C3%ADstica Acesso em: 19 dez. 2020.

TANENBAUM, A., STEEN, M. V., **Sistemas distribuídos:** princípios e paradigmas. 2.ed. Tradução de Arlete Simille Marques. São Paulo: Pearson Prentice Hall, 2007.

TEOREMA DE BAYES, 2020. Disponível em: https://pt.wikipedia.org/wiki/Teorema_de_Bayes. Acesso em: 19 dez. 2020.

WONG, A., REXACHS, D., LUQUE, E., **Parallel Application Signature for Performance Analysis and Prediction**, in IEEE Transactions on Parallel and Distributed Systems, v.26, n.7, p. 2009-2019, 1 July 2015, Doi: 10.1109/TPDS.2014.2329688

Apêndice A – Implementações dos algoritmos em python

```
#bibliotecas

import random #randomização
import numpy as np #tratamentos numéricos
import pandas as pd #dataframes
import seaborn as sns #distribuições
import tensorflow as tf #tensorflow
from sklearn import tree #arvore de decisão
from sklearn import metrics #metricas
from sklearn.svm import SVC #técnica SVM
from tensorflow import keras #redes neurais com keras
from imblearn import combine #técnica de SMOTE (combine)
from numpy.random import seed #sementes randomicas
import matplotlib.pyplot as plt #gráficos
from imblearn import over_sampling #técnica de SMOTE (over_sampling)
from imblearn import under_sampling #técnica de SMOTE (under_sampling)
from sklearn.metrics import f1_score #avaliar modelo pela métrica f1_score
from sklearn.decomposition import PCA #técnica PCA
from tensorflow.random import set_seed #sementes tf
from sklearn.metrics import recall_score #avaliar modelo pela métrica recall_score
from sklearn.model_selection import KFold #validacao cruzada
from sklearn.metrics import roc_auc_score #avaliar modelo pela métrica roc_auc_score
from sklearn.naive_bayes import GaussianNB #naive bayes
from sklearn.metrics import accuracy_score #avaliar modelo pela métrica accuracy_score
from sklearn.linear_model import Perceptron #rede neural perceptron
from sklearn.metrics import precision_score #avaliar modelo pela métrica precision_score
from sklearn.metrics import confusion_matrix #matriz de confusão
from sklearn.neural_network import MLPClassifier #multi-layer perceptron
from sklearn.preprocessing import StandardScaler #padronização
from sklearn.metrics import plot_confusion_matrix #análise de matriz de confusão
from sklearn.linear_model import LinearRegression #regressão linear
from sklearn.metrics import classification_report #metricas de validação
from mlxtend.classifier import StackingClassifier #ensemble stack
from sklearn.neighbors import KNeighborsClassifier #k-vizinhos
from sklearn.model_selection import StratifiedKFold #validação cruzada
from sklearn.linear_model import LogisticRegression #regressão logistica que irá juntar os modelos no
ensemble
```

```
#descrição dos dados

df1 = pd.read_csv('sgemm_product.csv',sep=',')
df = df1.sample(frac=0.5) #limitar o tamanho apenas para processamento mais rápido e testar técnicas
print('Número de linhas e colunas:',df.shape)
df.head()

#descrição estatística
df.describe()


#tratamento e transformação dos dados

#dados duplicados
#verificando se temos duplicidade dos dados
df[df.duplicated() == True]

#dados faltantes
#verificando se temos dados faltantes
df.isnull().sum()

#outliers
#verificando outliers dos tempos de resposta
plt.figure(figsize=(16,5))
sns.boxplot(x="variable", y="value", data=pd.melt(df.iloc[:,-4:]))
plt.show()

#detectando e removendo outliers
def detect_remove_outlier(data):
   Q1 = data.quantile(0.25)
   Q3 = data.quantile(0.75)
   IQR = Q3-Q1
   mask = ((data > (Q1 - 1.5 * IQR)) & (data < (Q3 + 1.5 * IQR)))
   data_clean = data[mask]
   return data_clean

df_clean = detect_remove_outlier(df)
df_clean.head()
```

```
#verificando dados faltantes obtidos pela função de outlier
df_clean.isnull().sum().sort_values(ascending=False).head(18)

#dropando as linhas com NaN
df_clean = df_clean.dropna()
df_clean = df_clean.reset_index()
print('Tamanho da base sem outliers:',df_clean.shape)
df_clean.isnull().sum().sort_values(ascending=False).head(18)

#verificando boxplot dos tempos de resposta após tratamento de outliers
plt.figure(figsize=(16,5))
sns.boxplot(x="variable", y="value", data=pd.melt(df_clean.iloc[:,-4:]))
plt.show()

#dados correlacionados
#matriz de correlação
plt.figure(figsize=(16,5))

plt.imshow(df_clean.corr(), cmap='Reds', interpolation='none', aspect='auto')
plt.xticks(range(len(df_clean.corr())), df_clean.corr().columns, rotation='vertical')
plt.yticks(range(len(df_clean.corr())), df_clean.corr().columns)
plt.suptitle('Matriz de Correlação', fontsize=15, fontweight='bold')
plt.grid(False)
plt.colorbar()
plt.show()

#dados distribuídos e balanceados
#Consolidando os atributos de predição em um único apenas, considerando suas médias
run = df_clean.iloc[:,-4:].mean(axis=1)
df_clean['Run (ms)'] = pd.Series(run)

data = df_clean.copy()
data

#verificando a distribuição dos dados
plt.figure(figsize=(16,5))
fig = sns.distplot(data.iloc[:,-1:])
```

```
#aplicando log na variável alvo
run_log = np.log(data['Run (ms)'])
data.insert(20,'Run Log (ms)',run_log)

plt.figure(figsize=(16,5))
fig = sns.distplot(data.iloc[:,-1:])

#caso queira apagar a última coluna para dar rollback nos dados
#data.drop(data.iloc[:,-1:],axis = 1, inplace = True)
data

#aplicando discretização dos dados no tempo de resposta

#classificação utilizando os dados reais
threshold = [0,20,40,60,80,100,200,300,400,500,600]
run_disc = pd.cut(data['Run (ms)'], bins=threshold)

#inserindo na base
data.insert(21,'Run Discretizado (ms)',run_disc)

#realizando o mapping numbers (transformando categorias em números)
data['Run Discretizado Mapping (ms)'] = data['Run Discretizado (ms)'].astype('category').cat.codes
data['Run Discretizado Mapping (ms)'].unique()

#problema de classificação binária (maior ou menor que a média)
#classificação utilizando os dados reais
avg = data['Run (ms)'].mean()
data.loc[data['Run (ms)'] <= avg, 'Run Bin'] = 0
data.loc[data['Run (ms)'] >  avg, 'Run Bin'] = 1
#obtendo dados
run_bin = data[['Run Bin']].values.astype('int')
data.drop(data.iloc[:,-1:],axis = 1, inplace = True)
#inserindo na base de dados
data.insert(23,'Run Binario',run_bin)

print('Se menor ou igual que ',avg,'então será 0')
print('Se maior que ',avg,'então será 1')
```

```
#verificando balanceamento entre as variáveis categóricas (multiclasse)
fig = plt.figure(figsize=(16,5))
fig = data['Run Discretizado (ms)'].value_counts().plot(kind='bar')

#verificando balanceamento entre as variáveis categóricas mapping (multiclasse)
fig = plt.figure(figsize=(16,5))
fig = data['Run Discretizado Mapping (ms)'].value_counts().plot(kind='bar')

#classificação utilizando os dados reais
mask0 = (data['Run Discretizado Mapping (ms)'] == 0)
mask1 = (data['Run Discretizado Mapping (ms)'] == 1)
mask2 = (data['Run Discretizado Mapping (ms)'] == 2)
mask3 = (data['Run Discretizado Mapping (ms)'] == 3)
mask4 = (data['Run Discretizado Mapping (ms)'] == 4)
mask5 = (data['Run Discretizado Mapping (ms)'] == 5)
mask6 = (data['Run Discretizado Mapping (ms)'] == 6)
mask7 = (data['Run Discretizado Mapping (ms)'] == 7)
mask8 = (data['Run Discretizado Mapping (ms)'] == 8)
mask9 = (data['Run Discretizado Mapping (ms)'] == 9)

print('Distribuição das classes:')
print("[000,020]: %.2f" % (len(data[mask0].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[020,040]: %.2f" % (len(data[mask1].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[040,060]: %.2f" % (len(data[mask2].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[060,080]: %.2f" % (len(data[mask3].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[000,100]: %.2f" % (len(data[mask4].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[100,200]: %.2f" % (len(data[mask5].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
print("[200,300]: %.2f" % (len(data[mask6].index.values)/len(data['Run Discretizado Mapping (ms)']) *
100),'%')
```

```
print("[300,400]: %.2f" % (len(data[mask7].index.values)/len(data['Run Discretizado Mapping (ms)']) * 100),' %')
print("[400,500]: %.2f" % (len(data[mask8].index.values)/len(data['Run Discretizado Mapping (ms)']) * 100),' %')
print("[500,600]: %.2f" % (len(data[mask9].index.values)/len(data['Run Discretizado Mapping (ms)']) * 100),' %')

#verificando balanceamento entre as variáveis binárias
fig = plt.figure(figsize=(16,5))
fig = data['Run Binario'].value_counts().plot(kind='bar')

mask0 = (data['Run Binario'] == 0)
mask1 = (data['Run Binario'] == 1)

print('Distribuição das classes:')
print(" <= Média: %.2f" % (len(data[mask0].index.values)/len(data['Run Binario']) * 100),'%')
print(" >  Média: %.2f" % (len(data[mask1].index.values)/len(data['Run Binario']) * 100),'%')

#separando os dados em conjunto de treinamento e testes, na proporção 80-20
perc = 0.8
#salvando y como seria valores reais em logaritimo
x_train_log, y_train_log = data.iloc[0:int(len(data)*perc),1:-9],data['Run Log (ms)'].iloc[0:int(len(data)*perc)]
x_test_log, y_test_log = data.iloc[int(len(data)*perc):int(len(data)),1:-9],data['Run Log (ms)'].iloc[int(len(data)*perc):int(len(data))]

#salvando y como serie classificação multiclasse
x_train_m, y_train_m = data.iloc[0:int(len(data)*perc),1:-9],data['Run Discretizado (ms)'].iloc[0:int(len(data)*perc)]
x_test_m, y_test_m = data.iloc[int(len(data)*perc):int(len(data)),1:-9],data['Run Discretizado (ms)'].iloc[int(len(data)*perc):int(len(data))]

#salvando y como serie classificação multiclasse mapping number
x_train_map, y_train_map = data.iloc[0:int(len(data)*perc),1:-9],data['Run Discretizado Mapping (ms)'].iloc[0:int(len(data)*perc)]
x_test_map, y_test_map = data.iloc[int(len(data)*perc):int(len(data)),1:-9],data['Run Discretizado Mapping (ms)'].iloc[int(len(data)*perc):int(len(data))]
```

```
#salvando y como classificação binário
x_train_b, y_train_b = data.iloc[0:int(len(data)*perc),1:-9],data['Run
Binario'].iloc[0:int(len(data)*perc)]
x_test_b, y_test_b = data.iloc[int(len(data)*perc):int(len(data)),1:-9],data['Run
Binario'].iloc[int(len(data)*perc):int(len(data))]


sc = StandardScaler()
x_train_m = sc.fit_transform(x_train_m)
x_test_m = sc.fit_transform(x_test_m)

x_train_map = sc.fit_transform(x_train_map)
x_test_map = sc.fit_transform(x_test_map)

x_train_b = sc.fit_transform(x_train_b)
x_test_b = sc.fit_transform(x_test_b)

print('Tamanho da amostra de treinamento:',len(x_train_b))
print('Tamanho da amostra de teste:',len(x_test_b))

#otimizando a distribuição

#aplicando técnicas de SMOTE para verificar se conseguimos uma estratégia melhor de balanceamento
dos dados para validar os modelos

#Over Sampling
oversamp = over_sampling.SMOTE()
Xo, Yo = oversamp.fit_resample(x_train_map, y_train_map)

plt.figure(figsize=(16,5))
h = plt.hist(Yo)

#Under Sampling
undersamp = under_sampling.RandomUnderSampler()
Xu, Yu = undersamp.fit_resample(x_train_map, y_train_map)

plt.figure(figsize=(16,5))
h = plt.hist(Yu)
```

```
#Over Under Sampling (estratégia mista)
overunder = combine.SMOTEENN(sampling_strategy='all')
Xc, Yc = overunder.fit_resample(x_train_map, y_train_map)

plt.figure(figsize=(16,5))
h = plt.hist(Yc)

#avaliando qual a melhor estratégia de SMOTE aplicando uma modelagem do tipo SVM

#treinando um classificador com o balanceamento original
clf = SVC(gamma='auto')
clf.fit(x_train_map,y_train_map)
ZY_ = clf.predict(x_test_map)

#treinando um classificador com o balanceamento Over Sampling
clf_ov = SVC(gamma='auto')
clf_ov.fit(Xo,Yo)
ZYov_ = clf_ov.predict(x_test_map)

#treinando um classificador com o balanceamento Under Sampling
clf_un = SVC(gamma='auto')
clf_un.fit(Xu,Yu)
ZYun_ = clf_un.predict(x_test_map)

#treinando um classificador com o balanceamento Over Under Sampling
clf_co = SVC(gamma='auto')
clf_co.fit(Xc,Yc)
ZYco_ = clf_co.predict(x_test_map)

#avaliando a matriz confusão dos modelos com as diferentes estratégias de balanceamento

#matriz de confusão do balanceamento original
disp = plot_confusion_matrix(clf, x_test_map, y_test_map,
                             cmap=plt.cm.Blues,
                             normalize='true')
```

```
#matriz de confusão do balanceamento Over Sampling
disp = plot_confusion_matrix(clf, x_test_map, y_test_map,
                      cmap=plt.cm.Blues,
                      normalize='true')

#matriz de confusão do balanceamento Under Sampling
disp = plot_confusion_matrix(clf_un, x_test_map, y_test_map,
                      cmap=plt.cm.Blues,
                      normalize='true')

#matriz de confusão do balanceamento Over Under Sampling
disp = plot_confusion_matrix(clf_co, x_test_map, y_test_map,
                      cmap=plt.cm.Blues,
                      normalize='true')

#avaliando os resultados com as diferentes estratégias de balanceamento
print('+-----------------------------------------------------------------+')
print('Resultados com o balanceamento original:')
print('+-----------------------------------------------------------------+')
print(classification_report(y_test_map, ZY_))
print('+-----------------------------------------------------------------+')
print('Resultados com o balanceamento Over Sampling:')
print('+-----------------------------------------------------------------+')
print(classification_report(y_test_map, ZYov_))
print('+-----------------------------------------------------------------+')
print('Resultados com o balanceamento Under Sampling:')
print('+-----------------------------------------------------------------+')
print(classification_report(y_test_map, ZYun_))
print('+-----------------------------------------------------------------+')
print('Resultados com o balanceamento Over Under Sampling:')
print('+-----------------------------------------------------------------+')
print(classification_report(y_test_map, ZYco_))
print('+-----------------------------------------------------------------+')

#As estratégias com os dados originais e com o balanceamento Over Sampling, ficaram praticamente
#equivalentes. Tivemos um pequeno ganho com relação ao recall, subindo a média um pouco para o
#over sampling. Mesmo considerando pouco o ganho, vamos utilizar essa estratégia na nossa
#modelagem.
```

```
#apresentando os dados graficamente

#aplicando PCA e plotando dados com conjunto original de dados de treino (multiclasse)
pca = PCA(n_components=10, random_state=1)
pca_result = pca.fit_transform(x_train_m)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_result[:,0],y=pca_result[:,1],alpha='auto', hue=y_train_m, palette='tab10')
plt.title('Scatterplot com projeção PCA do conjunto de dados multiclasse original')
plt.show()

#apresentando os dados graficamente

#aplicando PCA e plotando dados com conjunto original de dados de treino (multiclasse mapping
number)
pca = PCA(n_components=10, random_state=1)
pca_result = pca.fit_transform(x_train_map)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_result[:,0],y=pca_result[:,1],alpha='auto', hue=y_train_map, palette='tab10')
plt.title('Scatterplot com projeção PCA do conjunto de dados multiclasse mapping number')
plt.show()

#aplicando PCA e plotando dados com conjunto original de dados de treino (binário)
pca = PCA(n_components=2, random_state=1)
pca_result = pca.fit_transform(x_train_b)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_result[:,0],y=pca_result[:,1],alpha='auto', hue=y_train_b, palette='prism')
plt.title('Scatterplot com projeção PCA do conjunto de dados binário')
plt.show()
```

```
import warnings
warnings.filterwarnings("ignore")

#aplicando PCA e plotando dados com conjunto original de dados de treino (multiclasse mapping number) com Over Sampling
pca = PCA(n_components=10, random_state=1)
pca_result = pca.fit_transform(Xo)

#realizando o plot
fig = plt.figure(figsize=(20,6))
sns.scatterplot(x=pca_result[:,0],y=pca_result[:,1],alpha='auto', hue=Yo, palette='tab10')
plt.title('Scatterplot com projeção PCA do conjunto de dados multiclasse mapping number com Over Sampling')
plt.show()

#estratégias de modelagem

#KNN

#identificando melhor parâmetro K para aplicar nos modelos

cv_knn = StratifiedKFold(n_splits=10, shuffle=True, random_state=42) #validação cruzada

#fatiando ainda mais os dados, para viabilizar a simulação em várias execuções
#estamos realizando em cima dos dados binários, pois tivemos muitos problemas/erros no cálculo das métricas
#para um problema de classificação multi-classe.
train_x = x_train_b[:10000]
train_y = y_train_b[:10000]

mauc_knn = []
macc_knn = []
vk = []
```

```
for k in range(1, 21):
   vauc_knn = []
   vacc_knn = []
   for train_index, test_index in cv_knn.split(train_x, train_y):
      x_train, x_test = x_train_b[train_index], x_train_b[test_index]
      y_train, y_test = y_train_b[train_index], y_train_b[test_index]
      model_knn = KNeighborsClassifier(n_neighbors=k, metric = 'euclidean')
      model_knn.fit(x_train,y_train)
      y_pred_knn = model_knn.predict(x_test)
      score = accuracy_score(y_pred_knn, y_test)
      vauc_knn.append(roc_auc_score(y_test, y_pred_knn))
      vacc_knn.append(accuracy_score(y_pred_knn, y_test))
   macc_knn.append(np.mean(vacc_knn))
   mauc_knn.append(np.mean(vauc_knn))
   vk.append(k)

#identificando o melhor parâmetro K, dada a acurácia e o AUC
best_k = np.argmax(mauc_knn)+1
print('Melhor k:', best_k, ' AUC:',mauc_knn[best_k-1])
plt.figure(figsize=(20,6))
plt.plot(vk, mauc_knn, '-ro', label= 'AUC')
plt.plot(vk, macc_knn, '-bo', label = 'Accuracy')
plt.xlabel('k', fontsize = 15)
plt.ylabel('Score', fontsize = 15)
plt.legend()
plt.show()

#K-vizinhos, realizando a modelagem com o melhor k para multiclasse com mapping number Over
Sampling

model_KNNo = KNeighborsClassifier(n_neighbors = best_k, metric='euclidean')
model_KNNo.fit(Xo,Yo)
y_pred_KNNo = model_KNNo.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_KNNo))
print('+---------------------------------------------------------------+')
```

```
#K-vizinhos, realizando a modelagem com o melhor k para classificação binária

model_KNNb = KNeighborsClassifier(n_neighbors = best_k, metric='euclidean')
model_KNNb.fit(x_train_b,y_train_b)
y_pred_KNNb = model_KNNb.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_KNNb))
print('+---------------------------------------------------------------+')


#SVM

#identificando melhor parâmetro c para aplicar nos modelos

cv_svm = StratifiedKFold(n_splits=10, shuffle=True, random_state=42) #validação cruzada

#fatiando ainda mais os dados, para viabilizar a simulação em várias execuções
#estamos realizando em cima dos dados binários, pois tivemos muitos problemas/erros no cálculo das métricas
#para um problema de classificação multi-classe.
train_x = x_train_b[:10000]
train_y = y_train_b[:10000]

mauc_svm = []
macc_svm = []
vc = []
```

```
for c in range(1, 21):
   vauc_svm = []
   vacc_svm = []
   for train_index, test_index in cv_svm.split(train_x, train_y):
      x_train, x_test = x_train_b[train_index], x_train_b[test_index]
      y_train, y_test = y_train_b[train_index], y_train_b[test_index]
      model_svm = SVC(C = c, gamma = 'auto')
      model_svm.fit(x_train,y_train)
      y_pred_svm = model_svm.predict(x_test)
      score = accuracy_score(y_pred_svm, y_test)
      vauc_svm.append(roc_auc_score(y_test, y_pred_svm))
      vacc_svm.append(accuracy_score(y_pred_svm, y_test))
   macc_svm.append(np.mean(vacc_svm))
   mauc_svm.append(np.mean(vauc_svm))
   vc.append(c)

#identificando o melhor parâmetro c, dada a acurácia e o AUC
best_c = np.argmax(mauc_svm)+1
print('Melhor c:', best_c, ' AUC:',mauc_svm[best_c-1])
plt.figure(figsize=(20,6))
plt.plot(vc, mauc_svm, '-ro', label= 'AUC')
plt.plot(vc, macc_svm, '-bo', label = 'Accuracy')
plt.xlabel('c', fontsize = 15)
plt.ylabel('Score', fontsize = 15)
plt.legend()
plt.show()

#SVM, realizando a modelagem com o melhor c com kernel linear para multiclasse com mapping
number Over Sampling

model_SVMol = SVC(kernel='linear', C = best_c, random_state=1)
model_SVMol.fit(Xo,Yo)
y_pred_SVMol = model_SVMol.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_SVMol))
print('+---------------------------------------------------------------+')
```

```
#SVM, realizando a modelagem com o melhor c com kernel radial (rbf) para multiclasse com mapping number Over Sampling

model_SVMor = SVC(kernel='rbf', C = best_c, random_state=1)
model_SVMor.fit(Xo,Yo)
y_pred_SVMor = model_SVMor.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_SVMor))
print('+---------------------------------------------------------------+')

#SVM, realizando a modelagem com o melhor c com kernel linear para classificação binária

model_SVMolb = SVC(kernel='linear', C = best_c, random_state=1)
model_SVMolb.fit(x_train_b,y_train_b)
y_pred_SVMolb = model_SVMolb.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_SVMolb))
print('+---------------------------------------------------------------+')

#SVM, realizando a modelagem com o melhor c com kernel radial (rbf) para classificação binária

model_SVMorb = SVC(kernel='rbf', C = best_c, random_state=1)
model_SVMorb.fit(x_train_b,y_train_b)
y_pred_SVMorb = model_SVMorb.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_SVMorb))
print('+---------------------------------------------------------------+')
```

```
#DT

#arvore de decisão critério entropy para multiclasse com mapping number Over Sampling

model_DTCoe = tree.DecisionTreeClassifier(criterion = 'entropy', random_state=42)
model_DTCoe.fit(Xo,Yo)
y_pred_DTCoe = model_DTCoe.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_DTCoe))
print('+---------------------------------------------------------------+')

#arvore de decisão critério gini para multiclasse com mapping number Over Sampling

model_DTCog = tree.DecisionTreeClassifier(criterion = 'gini', random_state=42)
model_DTCog.fit(Xo,Yo)
y_pred_DTCog = model_DTCog.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_DTCog))
print('+---------------------------------------------------------------+')

#arvore de decisão critério entropy para classificação binária

model_DTCbe = tree.DecisionTreeClassifier(criterion = 'entropy', random_state=42)
model_DTCbe.fit(x_train_b,y_train_b)
y_pred_DTCbe = model_DTCbe.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_DTCbe))
print('+---------------------------------------------------------------+')
```

```
#arvore de decisão critério gini para classificação binária

model_DTCbg = tree.DecisionTreeClassifier(criterion = 'gini', random_state=42)
model_DTCbg.fit(x_train_b,y_train_b)
y_pred_DTCbg = model_DTCbg.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_DTCbg))
print('+---------------------------------------------------------------+')


#NB

#naive bayes para multiclasse com mapping number Over Sampling

model_NBCo = GaussianNB()
model_NBCo.fit(Xo,Yo)
y_pred_NBCo = model_NBCo.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_NBCo))
print('+---------------------------------------------------------------+')

#naive bayes para classificação binária

model_NBCb = GaussianNB()
model_NBCb.fit(x_train_b,y_train_b)
y_pred_NBCb = model_NBCb.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_NBCb))
print('+---------------------------------------------------------------+')
```

```
#ensemble stacking parte I

#ensemble stacking para multiclasse com mapping number Over Sampling

lro = LogisticRegression()
model_Ensembleo = StackingClassifier(classifiers=[model_KNNo, model_SVMor, model_DTCog,
model_NBCo], meta_classifier=lro)
model_Ensembleo.fit(Xo,Yo)
y_pred_Ensembleo = model_Ensembleo.predict(x_test_map)

print('+--------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_Ensembleo))
print('+--------------------------------------------------------------+')

#ensemble stacking para classificação binária

lrb = LogisticRegression()
model_Ensembleb = StackingClassifier(classifiers=[model_KNNb, model_SVMorb, model_DTCbg,
model_NBCb], meta_classifier=lrb)
model_Ensembleb.fit(x_train_b,y_train_b)
y_pred_Ensembleb = model_Ensembleb.predict(x_test_b)

print('+--------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_Ensembleb))
print('+--------------------------------------------------------------+')

#redes neurias
#perceptron
#perceptron para multiclasse com mapping number Over Sampling

cv = KFold(n_splits=10, shuffle=True, random_state=42)
model_PERCEPTRONo = Perceptron(random_state=0)
model_PERCEPTRONo.fit(Xo,Yo)
y_pred_PERCEPTRONo = model_PERCEPTRONo.predict(x_test_map)

print('+--------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_PERCEPTRONo))
print('+--------------------------------------------------------------+')
```

```
#perceptron para classificação binária

cv = KFold(n_splits=10, shuffle=True, random_state=42)
model_PERCEPTRONo = Perceptron(random_state=0)
model_PERCEPTRONo.fit(x_train_b,y_train_b)
y_pred_PERCEPTRONo = model_PERCEPTRONo.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_PERCEPTRONo))
print('+---------------------------------------------------------------+')


#MLP

#multi-layer perceptron para multiclasse com mapping number Over Sampling

cv = KFold(n_splits=10, shuffle=True, random_state=42)
model_MLPo = MLPClassifier(max_iter=1000)
model_MLPo.fit(Xo,Yo)
y_pred_MLPo = model_MLPo.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_MLPo))
print('+---------------------------------------------------------------+')

#multi-layer perceptron para classificação binária

cv = KFold(n_splits=10, shuffle=True, random_state=42)
model_MLPb = MLPClassifier(max_iter=1000)
model_MLPb.fit(x_train_b,y_train_b)
y_pred_MLPb = model_MLPb.predict(x_test_b)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_b, y_pred_MLPb))
print('+---------------------------------------------------------------+')
```

```
#ensemble stacking parte II

#Ensemble considerando os dois melhores modelos: SVM radial e MLP para multiclasse com Over Sampling

lrmso = LogisticRegression()
model_Ensemblemso = StackingClassifier(classifiers=[model_MLPo, model_SVMor],
meta_classifier=lrmso)
model_Ensemblemso.fit(Xo,Yo)
y_pred_Ensemblemso = model_Ensemblemso.predict(x_test_map)

print('+---------------------------------------------------------------+')
print(classification_report(y_test_map, y_pred_Ensemblemso))
print('+---------------------------------------------------------------+')
```