UNIVERSIDADE ESTADUAL DE CAMPINAS

Instituto de Física “Gleb Wataghin”

Gianni Shigeru Setoue Liveraro

MACHINE LEARNING NA SELEÇÃO DE EVENTOS DO EXPERIMENTO ALICE DO LHC

Campinas

2022

Gianni Shigeru Setoue Liveraro

MACHINE LEARNING NA SELEÇÃO DE EVENTOS DO EXPERIMENTO ALICE DO LHC

Dissertação apresentada ao Instituto de Física "Gleb Wataghin" da Universidade Estadual de Campinas como parte dos requisitos exigidos para obtenção do título de Mestre em Física, na área de Física Aplicada.

Orientador: Prof. Dr. Jun Takahashi

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO GIANNI SHIGERU SETOUE LIVERARO, E ORIENTADA PELO PROF. DR. JUN TAKAHASHI.

Campinas

2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Física Gleb Wataghin
Lucimeire de Oliveira Silva da Rocha - CRB 8/9174

L75m

Liveraro, Gianni Shigeru Setoue, 1997-
Machine learning na seleção de eventos do experimento ALICE do LHC / Gianni Shigeru Setoue Liveraro. – Campinas, SP : [s.n.], 2022.

Orientador: Jun Takahashi.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Física Gleb Wataghin.

1. Aprendizado de máquina. 2. Experimento ALICE. 3. Grande Colisor de Hádrons (França e Suiça). I. Takahashi, Jun, 1971-. II. Universidade Estadual de Campinas. Instituto de Física Gleb Wataghin. III. Título.

Informações Complementares

Título em outro idioma: Machine learning for event selection in the ALICE experiment at the LHC
Palavras-chave em inglês:
Machine learning
ALICE experiment
Large Hadron Collider (France and Switzerland)
Área de concentração: Física Aplicada
Titulação: Mestre em Física
Banca examinadora:
Jun Takahashi [Orientador]
Rangel Arthur
Leandro Russovski Tessler
Data de defesa: 25-10-2022
Programa de Pós-Graduação: Física

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-9674-196X
- Currículo Lattes do autor: http://lattes.cnpq.br/6535470738548434

MEMBROS DA COMISSÃO EXAMINADORA DA DISSERTAÇÃO DE MESTRADO DO ALUNO GIANNI SHIGERU SETOUE LIVERARO - RA 265945 APRESENTADA E APROVADA AO INSTITUTO DE FÍSICA “GLEB WATAGHIN”, DA UNIVERSIDADE ESTADUAL DE CAMPINAS, EM 25/10/2022.

COMISSÃO JULGADORA:

- Prof. Dr. Jun Takahashi – Presidente e orientador (IFGW/UNICAMP)

- Prof. Dr. Leandro Russovski Tessler (IFGW/UNICAMP)

- Prof. Dr. Rangel Arthur (FT/UNICAMP)

OBS.: Ata da defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

CAMPINAS

2022

Dedico este trabalho aos meus pais,
Moacir Liveraro (in memoriam) e
Rute Setoue Liveraro.

Agradecimentos

Gostaria de expressar os meus agradecimentos àqueles sem os quais a realização deste trabalho não teria sido possível.

- Ao meu orientador, Prof. Dr. Jun Takahashi, pela confiança, atenção e pelos valiosos ensinamentos, assim como pela preocupação que sempre demonstrou com relação à minha formação.

- Ao Prof. Dr. David Dobrigkeit Chinellato pelas esclarecedoras discussões e sugestões que enriqueceram a minha pesquisa.

- Aos colegas e ex-colegas do grupo HadrEx pelo companheirismo durante este período e, em especial, ao Gabriel Reis Garcia por ter se mostrado sempre solícito e gentil em ajudar nas discussões sobre programação e *Machine Learning*.

- Aos Profs. Drs. Rangel Arthur, Leandro Tessler, Ernesto Kemp, Gabriela Castellano e Jean Rinkel pelas valiosas observações, discussões e sugestões em todas as bancas examinadoras.

- À UNICAMP e ao IFGW por fornecerem toda a estrutura para o desenvolvimento deste trabalho.

- Aos meus antigos professores, cujos nomes aqui não aparecem, mas que me apoiaram, de diferentes formas, a seguir uma carreira acadêmica.

- Aos meus pais, Moacir Liveraro (*in memoriam*) e Rute Setoue Liveraro, pelo carinho e apoio em todas as minhas escolhas.

- Aos membros das famílias Setoue e Liveraro pelo suporte de sempre.

- Aos antepassados.

- À Letícia Fernanda Alves pelo amor e paciência ao longo destes anos. Estendo este agradecimento à família Alves, que sempre me recebeu calorosamente.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001 e Processo 88887.514503/2020-00.

Esclarecimento sobre o padrão de texto adotado

Este trabalho foi realizado junto à colaboração internacional ALICE do experimento de mesmo nome no *Large Hadron Collider*. Os eventos utilizados, mesmo os simulados, foram gerados e são mantidos pela colaboração. Assim sendo, todas as análises sobre estes dados precisam ser aprovadas internamente antes de eventuais publicações ou apresentações em conferências.

Os nossos resultados foram documentados e submetidos em uma Nota de Análise da colaboração ALICE, que foi avaliada e certificada por membros internos. Por este motivo, os gráficos contidos nesta dissertação se encontram na lingua inglesa e estão rotulados como "*ALICE Performance*" - indicando a aprovação para uso em conferências.

Para manter a coerência entre os gráficos e o texto da dissertação, os números foram escritos utilizando pontos como separadores decimais e não virgulas, como seria o padrão da língua portuguesa. Pela mesma razão, também optamos por manter o nome de algumas variáveis na língua inglesa.

Resumo

O experimento ALICE do LHC foi construído para estudar a física das interações fortes e o Plasma de Quarks e Glúons. Em operações de alta luminosidade, é possível que múltiplas colisões ocorram suficientemente próximas no tempo de forma que os seus sinais sejam associados à um único evento. Este processo, conhecido como empilhamento (do inglês, *pile-up*), polui as informações das colisões de interesse e impacta negativamente as análises físicas. Atualmente, para mitigar tais efeitos, parte dos eventos com *pile-up* são removidos das análises por uma seleção baseada em cortes sobre propriedades globais dos eventos.

Nos últimos anos, os algoritmos de *Machine Learning* (ML) têm ganhado cada vez mais importância em muitas análises da física de altas energias devido ao seu alto desempenho em extrair padrões de grandes quantidades de dados. Neste trabalho, aplicamos as técnicas de ML sobre eventos simulados de colisões Pb-Pb à $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV para aprimorar a rejeição de *pile-up* no experimento ALICE. Os seguintes algoritmos foram treinados com as características globais dos eventos: *Random Forest*, *Boosted Decision Tree* (BDT), Máquinas de Vetores-Suporte, Regressão Logística e Rede Neural Artificial (*Multi-Layer Perceptron*, MLP). Uma arquitetura mais recente de Rede Neural Artificial, conhecida como *Graph Neural Network*, foi treinada com informações das trajetórias dos eventos. Um ganho de pelo menos 10% sobre o método de seleção padrão foi observado com a BDT e com a Rede MLP, que tiveram o melhor desempenho entre todos os modelos. As implementações de ML também se mostraram robustas e confiáveis para eventuais aplicações sobre eventos reais. A importância de cada atributo na classificação dos eventos foi avaliada e observamos que bons desempenhos podem ser obtidos mesmo retirando informações de alguns detectores dos treinamentos.

Palavras-chave: Large Hadron Collider; ALICE experiment; Machine Learning.

Abstract

The LHC's ALICE experiment is designed to study the physics of strong interactions and the Quark-Gluon Plasma. At large luminosity, multiple collisions can occur close enough in time that their signals are associated with a single event. This process, known as pile-up, pollutes information from collisions of interest and negatively impacts physical analyses. Currently, to mitigate such effects, part of the pile-up events are removed from analyses by a selection based on cuts on global event properties.

In recent years, Machine Learning (ML) algorithms have gained increasing importance in many analyzes of high-energy physics due to their high performance in extracting patterns from large amounts of data. In this work, we apply ML techniques on simulated Pb-Pb collision events at $\sqrt{s_{\mathrm{NN}}} = 5.02$ TeV to improve pile-up rejection in the ALICE experiment. The following algorithms were trained with the global event properties: Random Forest, Boosted Decision Tree (BDT), Support Vector Machines, Logistic Regression, and Artificial Neural Network (Multi-Layer Perceptron, MLP). A more recent architecture of Artificial Neural Network, known as Graph Neural Network, was trained with information from event tracks. A gain of at least 10% over the standard selection method was observed with BDT and MLP Network, which had the best performances among all models. The ML techniques also proved to be robust and reliable for eventual applications on real events. The importance of each feature on the classification task was also evaluated and we observed that good performances can be obtained even by removing information from some detectors in the training processes.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ADAM	*Adaptive Moment Estimation*.
ALICE	*A Large Ion Collider Experiment*.
ARI	Índice ajustado de Rand.
ATLAS	*A Toroidal LHC ApparatuS*.
AUC	*Area Under the Curve*.
BC	*Bunch Crossing*.
BDT	*Boosted Decision Tree*.
CERN	*Conseil Européen pour la Recherche Nucléaire*.
CMS	*Compact Muon Solenoid*.
CPU	*Central Processing Unit*.
DBSCAN	*Density-Based Spatial Clustering of Applications with Noise*.
DCA	*Distance of closest approach*.
DPG	*Data Preparation Group*.
EMCal	*Electromagnetic Calorimeter*.
FMD	*Forward Multiplicity Detector*.
FN	Falso negativo.
FP	Falso positivo.
FPR	*False Positive Rate*.

GAT	*Graph Attention Networks*.
GCN	*Graph Convolutional Network*.
GI	Ganho de Informação.
GNN	*Graph Neural Network*.
GPU	*Graphical Processing Unit*.
HLT	*High-Level Trigger*.
HMPID	*High Momentum Particle Identification Detector*.
IA	Inteligência Artificial.
ITS	*Inner Tracking System*.
L0	*Level 0* (*trigger*).
L1	*Level 1* (*trigger*).
L2	*Level 2* (*trigger*).
LBFGS	*Limited memory Broyden Fletcher Goldfarb Shanno Algorithm*.
LHC	*Large Hadron Collider*.
LHC-b	*Large Hadron Collider beauty*.
LR	Regressão Logística.
MB	*Minimum-Bias*.
MDI	*Mean Decrease Impurity*.
ML	*Machine Learning*.
MLP	*Multi-Layer Perceptron*.
MP	Modelo Padrão.
NMI	*Normalized Mutual Information*.
NN	*Artificial Neural Network*.

PDP *The Physics Data Processing project*.

PHOS *Photon Spectrometer*.

PMD *Photon Multiplicity Detector*.

QCD *Quantum Chromodynamics*.

QGP *Quark-Gluon Plasma*.

ReLU *Rectified Linear Unit*.

RF *Random Forest*.

RHIC *Relativistic Heavy Ion Collider*.

ROC *Receiver Operating Characteristic*.

SAG *Stochastic Average Gradient*.

SDD *Silicon Drift Detector*.

SGD *Stochastic Gradient Descent*.

SPD *Silicon Pixel Detector*.

SSD *Silicon Strip Detector*.

SVM *Support Vector Machines*.

TNR *True Negative Rate*.

TOF *Time Of Flight*.

TPC *Time Projection Chamber*.

TPR *True Positive Rate*.

TRD *Transition Radiation Detector*.

VN Verdadeiro negativo.

VP Verdadeiro positivo.

ZDC *Zero Degree Calorimeter*.

Sumário

Capítulo 1

Introdução e Contextualização

Ao longo das últimas décadas, o ramo da Física de Altas Energias tem se dedicado a estudar as estruturas básicas da matéria. Este esforço histórico culminou no desenvolvimento do Modelo Padrão (MP), que se consolidou como uma das mais bem sucedidas teorias da física - fornecendo uma descrição das partículas e forças fundamentais do Universo. Com exceção da gravidade, as demais forças fundamentais - força eletro-fraca e força nuclear forte - são explicadas pelo MP [1]. A força eletro-fraca é intermediada pelo fóton e pelos bósons $W^{\pm}$ e Z^0, sendo responsável pelas interações de natureza eletromagnética e pelos decaimentos das partículas. A força nuclear forte, por sua vez, é intermediada pelo glúon e se manifesta principalmente na interação de partículas conhecidas como quarks. Uma das características mais notáveis da física das interações fortes é o confinamento dos quarks e glúons, que não são encontrados livres na natureza, sendo presentes apenas em estados ligados conhecidos como hádrons.

O sucesso do MP em descrever a física das interações fundamentais tem uma ligação direta com o desenvolvimento de grandes experimentos. Um dos principais da atualidade, construído pela Organização Europeia para a Pesquisa Nuclear (CERN), é o *Large Hadron Collider* (LHC). Com 27,6 km de extensão, o LHC é um acelerador do tipo síncrotron projetado para colidir feixes de prótons ou íons pesados e, com isso, investigar a física na escala de TeV [2]. Atualmente, existem quatro grandes experimentos montados nas regiões de cruzamento destes feixes, dois de propósitos gerais - CMS [3] e ATLAS [4] - e dois de propósitos específicos - LHCb [5] e ALICE. Uma representação pictórica da estrutura do LHC e da disposição dos seus principais experimentos pode ser vista na Figura 1.1.

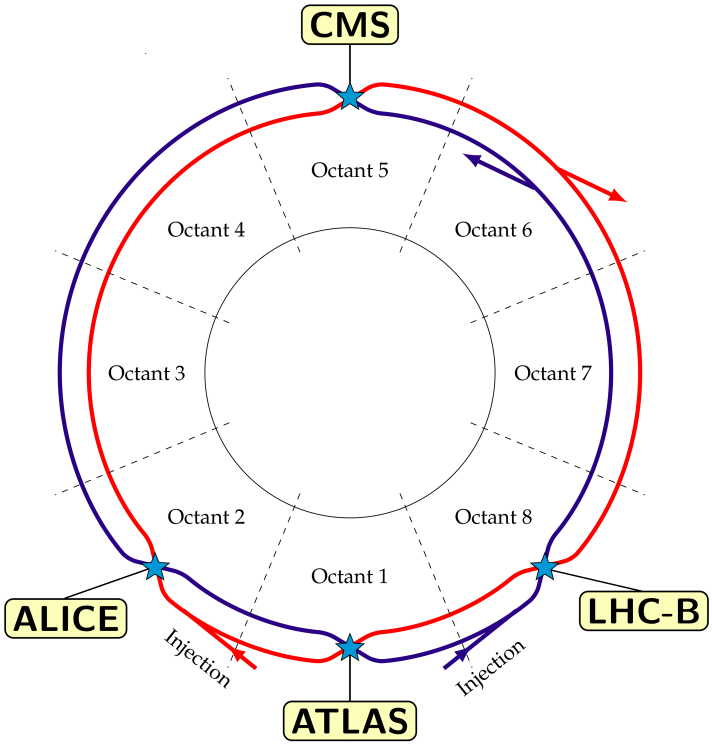


Figura 1.1: Estrutura geral do *Large Hadron Collider* contendo dois tubos nos quais os feixes de partículas se propagam em direções opostas. Nos pontos de cruzamento dos feixes estão localizados os principais experimentos: ALICE, ATLAS, LHCb e CMS. Figura modificada de [6].

O ALICE (do inglês, *A Large Ion Collider Experiment*) é um experimento construído com camadas de detectores especializados e dedicado ao estudo da física das interações fortes e da matéria sob condições extremas [7]. Para isso, são estudadas colisões próton-próton (pp), próton-núcleo (p-A) ou núcleo-núcleo (A-A) - sendo o chumbo (Pb) o núcleo mais utilizado. No ALICE, as colisões Pb-Pb e pp podem atingir, respectivamente, energias no centro de massa de até $\sqrt{s_{NN}}$ =5,02 TeV[1] e $\sqrt{s}$ =13 TeV. Nesta escala de energia, as colisões relativísticas de íons pesados produzem um estado da matéria conhecido como Plasma de Quarks e Glúons (do inglês, *Quark-Gluon Plasma*, QGP), que é caracterizado como um meio de alta temperatura e densidade de energia e que permite a existência de quarks e glúons desconfinados[2]. O estudo do QGP é feito através da medida de diversos observáveis experimentais, muitos deles com uma seção de choque pequena, como no caso da medida de quarks pesados. Tais análises necessitam de elevadas estatísticas de eventos. Por esta razão, a luminosidade dos feixes é calibrada de modo a fornecer ao ALICE uma taxa de colisões que seja próxima ao limite suportado pelo seus sistemas de detecção. Isto maximiza a quantidade de informações salvas das colisões e permite que sinais de fenômenos raros ou pouco frequentes possam ser estudados.

Durante o funcionamento do ALICE, colisões suficientemente próximas no tempo podem ocorrer -

[1] O subscrito *NN* indica medidas por par de núcleons.

[2] Evidências experimentais da existência da QGP já foram obtidas pelo experimento RHIC (do inglês, *Relativistic Heavy Ion Collider*) e podem ser consultadas nas referências [8–11].

de modo que os detectores, que possuem janelas de aquisição distintas, registrem os sinais das colisões como um único evento. Este processo é conhecido como empilhamento (do inglês, *pile-up*) e representa uma dificuldade operacional por tornar mais complexa a reconstrução dos eventos.

A ocorrência de *pile-up* é diretamente relacionada com a luminosidade dos feixes do LHC. Quanto maior for a luminosidade empregada, maior será a taxa de colisões nos *Runs* dos experimentos - e, portanto, maior será a ocorrência de eventos com *pile-up*. No ano de 2015, o LHC entregou ao experimento ALICE uma luminosidade integrada de 0.4 nb^{-1} para colisões Pb-Pb [12]. Neste período, de 30 a 40% dos eventos salvos continham *pile-up*. Com o aumento da luminosidade instantânea nos próximos *Runs* do LHC, espera-se que o experimento ALICE registre uma luminosidade integrada de até 7 nb^{-1} para colisões Pb-Pb - significando uma maior ocorrência de *pile-up* neste sistema. Assim sendo, o desenvolvimento de métodos para analisar eventos com múltiplas colisões deverá ganhar relevância nos próximos anos de operação do ALICE - em particular, nos estudos de sistemas Pb-Pb que, em geral, são mais complexos de se analisar do que eventos de sistemas pp devido a sua alta multiplicidade, como ilustra a Figura 1.2.

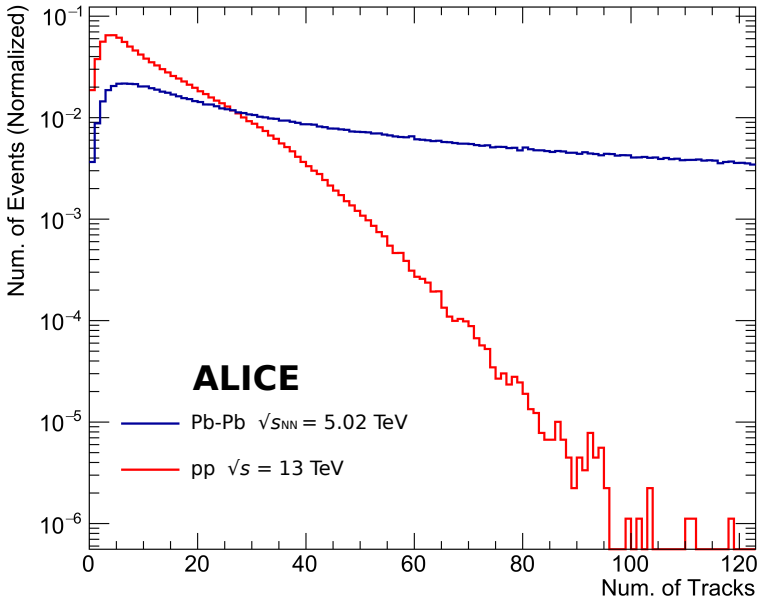


Figura 1.2: Distribuições normalizadas do número de trajetórias para sistemas pp e Pb-Pb no ALICE.

Soluções para os problemas envolvendo eventos com *pile-up* têm sido propostas em outros experimentos de altas energias, especialmente em estudos de sistemas pp, em que um mesmo evento pode ser salvo com até 30 colisões simultâneas em média [13]. Uma destas propostas envolve aplicar técnicas de análise multivariada, como *Machine Learning* (ML), para identificar trajetórias ou eventos com *pile-up* [14]. De fato, as técnicas de ML têm ganhado cada vez mais espaço na área de altas energias, sendo aplicadas na reconstrução de trajetórias [15], na classificação de eventos [16] e até no reconhecimento

de partículas [17] e jatos [18]. Este aumento no interesse da comunidade de altas energias pelas técnicas de ML pode ser vista pelo crescente número de publicações com esse tema nos últimos anos [19].

De forma objetiva, as técnicas de ML podem ser caracterizadas como algoritmos que resolvem problemas sem serem explicitamente programados para tal [20]. Na prática, isso significa que os algoritmos de ML são capazes de ajustar, com base em dados, modelos matemáticos que resolvem tarefas - de classificação, regressão ou agrupamento[3] - e, com isso, auxiliar na tomada de decisões. Além disso, estas técnicas podem operar em cenários com um alto número de dimensões. Tais propriedades são muito atrativas para a física experimental de altas energias, que trabalha com grandes quantidades de eventos - reais ou simulados - que são, por sua vez, comumente caracterizados por medidas de múltiplos detectores.

Até o momento, não existem aplicações das técnicas de ML para mitigar os efeitos do *pile-up* em eventos do experimento ALICE. Deste modo, o reconhecimento de eventos associados a múltiplas colisões, utilizando técnicas de *Machine Learning* em dados simulados do ALICE, é o tema principal deste trabalho. O Capítulo 2 descreverá o experimento ALICE e os seus principais detectores. O problema do *pile-up* também será definido em detalhes. Uma descrição aprofundada dos algoritmos e metodologias tradicionais envolvendo *Machine Learning* será dada no Capítulo 3. Neste Capítulo, abordaremos também as técnicas de *Deep Learning* e os métodos para análise da importância de atributos. No Capítulo 4, relataremos a metodologia aplicada, que envolveu o uso das técnicas de ML em eventos simulados com *pile-up* do experimento ALICE. Por fim, o Capítulo 5 apresenta e discute os resultados obtidos com dados simulados.

[3] Mais detalhes serão dados no Capítulo 3.

Capítulo 2

O Experimento ALICE

2.1 Visão Geral

O *Large Hadron Collider* (LHC) é um dos principais aparatos experimentais da atualidade voltados ao estudo da física na escala de TeV. Com as colisões pp, o objetivo do LHC é estudar tópicos como a quebra da simetria eletro-fraca, matéria e energia escura e física além do modelo-padrão. Por outro lado, as análises de colisões Pb-Pb são, basicamente, voltadas a investigação das propriedades do estado desconfinado de quarks e glúons.

Os dois feixes do LHC são constituídos por agrupamentos de partículas, cujo espaçamento temporal é de 25 ns. Em alguns pontos, estes feixes se cruzam e os produtos gerados das colisões são estudados pelos quatro grandes experimentos, sendo eles: ATLAS, CMS, LHCb e ALICE. Nestes experimentos, a taxa de cruzamento dos agrupamentos (denotados neste trabalho como *bunch crossings*) fornecida pelo LHC é de até 40 MHz, fazendo com que grandes quantidades de colisões ocorram por segundo. Por esta razão, os experimentos do LHC são preparados para selecionar e salvar as informações das colisões em tempo real. Esta alta taxa de colisões também implica em um problema técnico conhecido como empilhamento (*pile-up*), em que duas ou mais colisões ocorrem na mesma janela de aquisição dos detectores, fazendo com que os sinais das partículas produzidas sejam errôneamente associadas a uma única colisão - como será visto em mais detalhes na Seção 2.5.

Como comentado no capítulo 1, o experimento ALICE foi otimizado para estudar a matéria fortemente interagente através de colisões de altas energias entre íons pesados. Por isso, o ALICE conta com um sistema de detectores especializados, capazes de identificar hádrons, léptons e fótons em um amplo intervalo de momento (0.15-100 GeV/c) e em eventos com alta multiplicidade. Estes detectores, ilustrados na Figura 2.1, se encaixam em três categoriais principais: detectores centrais, detectores frontais e o espectrômetro de múons[21]. O espectrômetro de múons, localizado em uma região lateral do ALICE,

inclui apenas o *Forward Muon Spectrometer*, que é dedicado ao estudo de partículas que decaem no canal muônico. Já os detectores centrais, que estão imersos em um campo magnético de 0.5 T, incluem: *Inner Tracking System* (ITS), *Time Projection Chamber* (TPC), *Transition Radiation Detector* (TRD), *Time Of Flight* (TOF), *Photon Spectrometer* (PHOS), *Electromagnetic Calorimeter* (EMCal) e *High Momentum Particle Identification Detector* (HMPID). Na região central, se destacam o ITS e a TPC que são os principais detectores do ALICE - responsáveis por, dentre outras coisas, reconstruir trajetórias e identificar partículas.

Os detectores frontais estão localizados nas regiões mais externas do ALICE, sendo eles: *Photon Multiplicity Detector* (PMD), *Forward Multiplicity Detector* (FMD), T0, V0 e *Zero Degree Calorimeter* (ZDC). Dentre estes, se destacam os detectores T0 e V0, que são rápidos e atuam no sistema de seleção em tempo real dos eventos, como será visto posteriormente. Além disso, os detectores V0 e ZDC podem ser utilizados em conjunto para determinar o parâmetro de impacto das colisões.

Uma descrição detalhada sobre os detectores mais relevantes para o desenvolvimento deste trabalho será fornecida na Seção 2.3. Antes, porém, serão definidas algumas variáveis cinemáticas e parâmetros que caracterizam os eventos no ALICE.

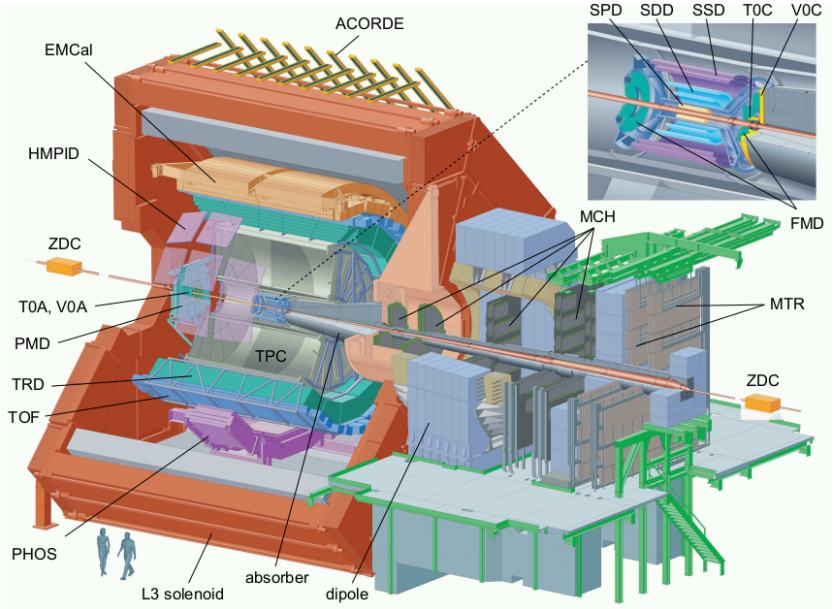


Figura 2.1: Estrutura do experimento ALICE do LHC, destacando os detectores centrais, frontais e espectrômetro de múons. Figura retirada de [21].

2.2 Caracterização dos eventos

O sistema de coordenadas do ALICE está indicado na Figura 2.2. O eixo z é alinhado com a direção de propagação dos feixes. O eixo x aponta para o centro do LHC e é alinhado com a direção horizontal. O eixo y é alinhado na direção perpendicular ao plano xz. As coordenadas xy caracterizam a direção

transversal do ALICE, ao passo que o eixo z representa a sua direção longitudinal. O ângulo polar θ é definido com relação ao eixo z, enquanto o ângulo azimutal ϕ é determinado sobre o plano transversal.

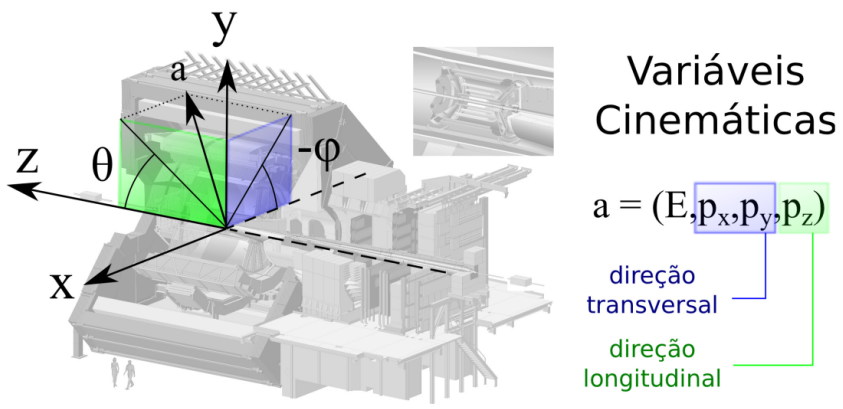


Figura 2.2: Sistema de coordenadas do experimento ALICE. Figura retirada de [22].

Algumas medidas no ALICE são feitas exclusivamente na direção transversal, como o momento transversal p_T:

$$p_T^2 = p_x^2 + p_y^2, \tag{2.1}$$

cuja distribuição é utilizada para a identificação de partículas, por exemplo. No sentido longitudinal, uma variável geométrica importante é a pseudorapidez η:

$$\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right], \tag{2.2}$$

utilizada, de forma prática, como o ângulo de espalhamento das partículas com relação ao eixo do feixe. Os detectores do ALICE são posicionados de acordo com intervalos de pseudorapidez: detectores centrais são limitados a região $|\eta| \leq 1,0$, detectores frontais possuem cobertura de $|\eta| \geq 1,5$ e o espectrômetro de muons é posicionado com $-4,0 \leq \eta \leq -2,5$. A Figura 2.3 ilustra a variação de η para diferentes valores de θ.

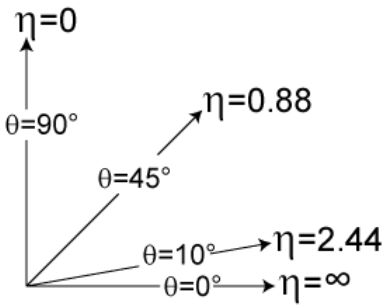


Figura 2.3: Pseudorapidez η em função do ângulo polar θ. Figura retirada de [23].

Mais uma variável geométrica, definida nos sentidos transversal e longitudinal, é a distância de máxima aproximação (do inglês, *distance of closest approach*, DCA) das trajetórias com relação aos vér-

tices primários[1]. Normalmente, esta informação é empregada na seleção trajetórias de acordo com os objetivos físicos de cada estudo. A Figura 2.4 exemplifica, para o plano transversal, duas trajetórias e os respectivos DCAs com relação ao vértice primário. Como será visto adiante, a informação da componente z do DCA (DCA_z) pode ser aproveitada por modelos preditivos para identificar eventos com ocorrência de *pile-up*.

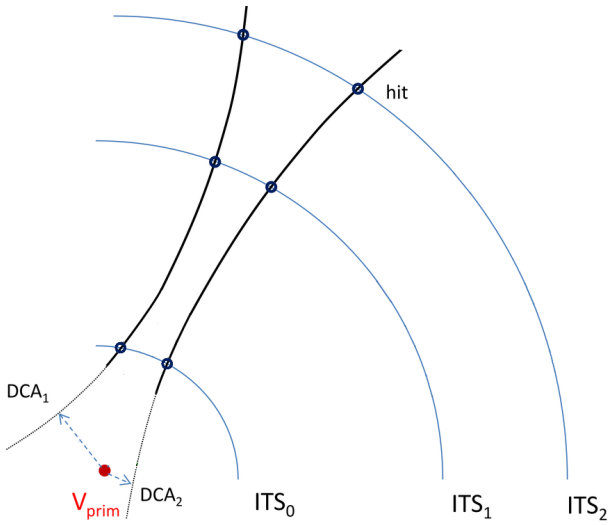


Figura 2.4: Representação da distância de máxima aproximação com relação ao vértice primário (em vermelho) no plano transversal. As linhas sólidas representam as trajetórias, enquanto as suas extrapolações estão indicadas pelas linhas pontilhadas. Figura modificada de [21].

Finalmente, um conceito importante para a caracterização de eventos no ALICE é o de centralidade, que representa o parâmetro de impacto das colisões de íons pesados[21]. Colisões são ditas centrais se o parâmetro de impacto envolvido for pequeno, enquanto colisões com alto parâmetro de impacto são classificadas como periféricas. Na prática, o grau de centralidade dos eventos/colisões é designado como um percentil. Eventos com percentis de centralidade maiores (como 70-100%) estão associados a colisões mais periféricas, ao passo que percentis de centralidade menores (como 0-10%) representam eventos com colisões centrais.

Experimentalmente, a determinação da centralidade é feita usando a multiplicidade de partículas. Eventos com maior multiplicidade estão, em geral, associados a colisões mais centrais e, por isso, são classificados com um percentil de centralidade menor. Inversamente, eventos de baixa multiplicidade são relacionados a percentis de centralidade maiores por estarem relacionados a colisões mais periféricas.

2.3 Principais detectores

As análises contidas neste trabalho foram feitas sobre dados simulados nas condições de operação do ALICE entre os anos de 2015 e 2018 (período conhecido como *Run* 2 do LHC). Por esta razão, a

[1] De forma mais ampla, o conceito de DCA também abrange vértices secundários. Porém, neste trabalho, focaremos na definição de DCA como a distância de máxima aproximação com relação ao vértice primário apenas.

descrição dos sistemas de detecção e aquisição de eventos contemplará apenas este período[2]. A seguir, será fornecida uma descrição dos detectores do ALICE que foram mais relevantes para as análises contidas neste trabalho. Mais informações acerca dos demais sistemas de detecção podem ser encontradas em [25].

2.3.1 *Inner Tracking System* (ITS)

O ITS é o sistema de detecção mais interno da região central do ALICE, sendo constituído por seis camadas de detectores de silício localizados a uma distância que varia de 4 a 44 cm com relação ao eixo do feixe (Figura 2.5). Os principais objetivos do ITS são relacionados a determinação dos vértices primários[3] e secundários[4], reconstrução trajetórias e identificação de partículas.

O sistema ITS opera com três tipos de detectores de silício. As duas camadas mais internas são do tipo *Silicon Pixel Detector* (SPD), enquanto as duas camadas intermediárias e as duas mais externas são dos tipos *Silicon Drift Detector* (SDD) e *Silicon Strip Detector* (SSD), respectivamente. Todas as camadas possuem uma boa resolução espacial e são capazes de operar com tempos razoavelmente rápidos de leitura quando comparados aos detectores mais lentos do ALICE, como a TPC.

Pelo fato de serem utilizados diferentes tipos de detectores de silício, os tempos de leitura do ITS variam de acordo com as camadas. A janela de leitura do SPD é de 300 ns, período no qual ocorrem 12 *bunch crossings* (BCs) no ALICE. Já os detectores SDD e SSD possuem, respectivamente, tempos de leitura de 6.4 µs (256 BCs) e $\approx$ 1µs ($\approx$ 40 BCs), tornando o SPD o detector com a menor janela de aquisição no ITS. Por esta razão, dentre os detectores centrais, o SPD é o menos suscetível aos efeitos do *pile-up*. Além disso, a rápida resposta do SPD permite que as suas informações sejam utilizadas para a seleção dos eventos em tempo real através do sistema de gatilho (*trigger*).

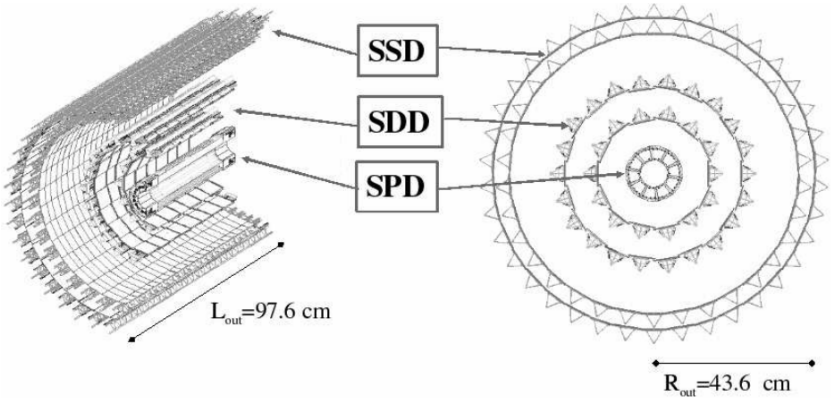


Figura 2.5: Estrutura e dimensões do detector ITS. Figura retirada de [25].

[2]Para mais detalhes sobre as atualizações do ALICE para os *Runs* 3 e 4, consultar [24].
[3]Ponto geométrico em que as colisões do feixe ocorreram.
[4]Ponto geométrico em que uma partícula decai.

2.3.2 *Time Projection Chamber* (TPC)

Sendo o principal detector central do ALICE, a TPC é ótimizada para reconstruir trajetórias, identificar e medir o momento de partículas carregadas e determinar os vértices primários e secundários dos eventos. Com notável versatilidade, a TPC também é capaz de fornecer uma alta resolução da posição das trajetórias e operar em cenários de alta multiplicidade - como é o caso de colisões Pb-Pb.

A TPC está posicionada ao redor do ITS e possui um formato cilíndrico, em que o seu volume é preenchido por um gás ($Ne/CO_2/N_2$) imerso em um campo elétrico (Figura 2.6). O campo é estabelecido entre o eletrodo central e as extremidades da TPC, onde estão localizados os detectores de carga (*pads*). Basicamente, as partículas das colisões produzem cargas de ionização no gás que são transportadas até as extremidades da TPC pela influência do campo elétrico. O tempo de transporte (*drift*) de uma carga de ionização até os *pads* de detecção é de até 100 µs, sendo esta a janela de aquisição da TPC. Deste modo, a TPC é considerada o sistema de detecção mais lento do ALICE e especialmente suscetível a ocorrência *pile-up*, uma vez que 4000 BCs ocorrem neste intervalo de tempo.

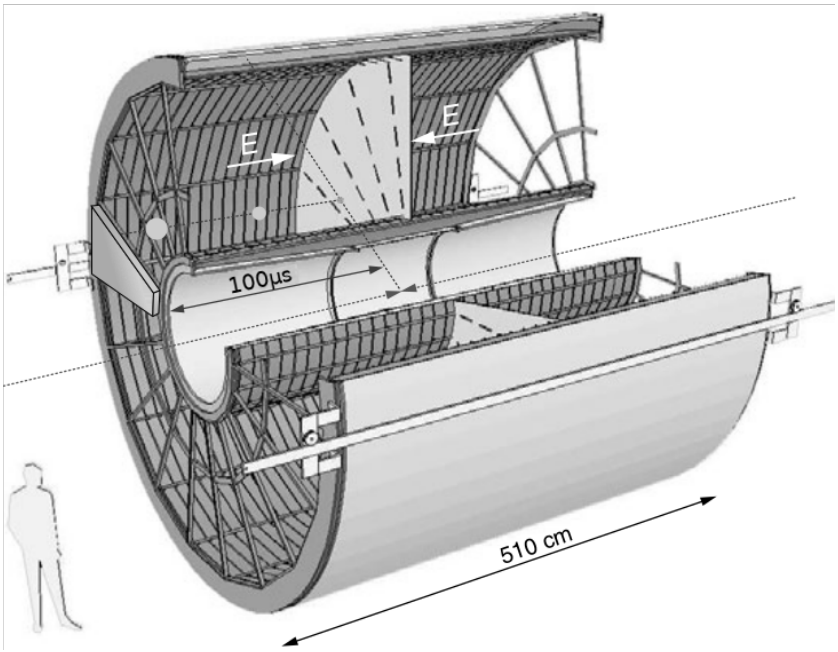


Figura 2.6: Estrutura do detector TPC. Figura modificada de [25].

Dado que a velocidade de transporte das cargas pelo gás da TPC é de $\approx 2.5\text{cm}/\mu\text{s}$ e que o instante inicial da colisão é estimado pelo detector T0, é possível estimar posição longitudinal das trajetórias que passam pela TPC de acordo com o tempo com que as respectivas cargas de ionização são coletadas. Esta dinâmica é especialmente relevante quando se considera trajetórias de interações *pile-up* vindas de um *bunch crossing* diferente daquele que ativou o sistema de *trigger*, como se verá adiante.

2.3.3 V0 detector

O detector V0 é constituido por dois discos cintiladores (V0A e V0C) posicionados assimetricamente em regiões de alta pseudorapidez - com $2,8 < \eta < 5,1$ para V0A e $-3,7 < \eta < -1,7$ para V0C. É um dos detectores mais rápidos do ALICE, com uma janela de aquisição pequena o suficiente para detectar as partículas vindas da colisão principal (1 BC). Desta forma, o detector V0 tem um papel central no sistema de *trigger*, que requer respostas rápidas para a seleção de eventos em tempo real. Ademais, outras atribuições do detector V0 incluem o controle de luminosidade e a estimação da centralidade dos eventos. A Figura 2.7 indica a posição dos detectores V0 e T0 com relação ao ITS.

Devido a sua alta resolução temporal, as informações do V0 são cruzadas com as informações de detectores mais lentos, como a TPC, para identificar eventos com ocorrência de *pile-up*, como será discutido.

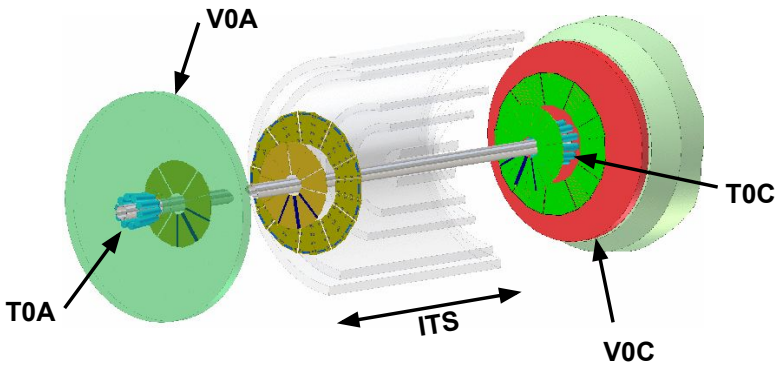


Figura 2.7: Posicionamento dos detectores V0 e T0 ao redor do ITS, que está representado pelas suas seis camadas (em cinza). Figura modificado de [26].

2.3.4 T0 detector

O detector T0 é constituído por dois conjuntos de contadores de radiação Cherenkov, T0A e T0C, também posicionados em regiões de alta pseudorapidez ao redor do ITS[26]. Da mesma forma como o V0, o detector T0 possui uma alta precisão temporal ($\approx$ 50ps), sendo utilizado para marcar o tempo em que as colisões de interesse ocorrem - informação aproveitada por detectores como o *Time of Flight* na identificação de partículas. Outra capacidade do T0 é a de fornecer uma estimativa inicial para a multiplicidade do evento, que auxilia o sistema de *trigger* em seleções de eventos que consideram a centralidade.

Em segundo plano, o T0 também é responsável por fornecer uma primeira estimativa da posição do vértice primário com uma precisão de $\pm 1,5$ cm - informação utilizada para discriminar interações

feixe-feixe de interações feixe-gás que acontecem no LHC.

2.4 Sistema de *Trigger*

Durante a operação do LHC, mais de 10^4 colisões Pb-Pb podem ocorrer por segundo, não sendo viável armazenar todas as informações registradas pelos detectores. Por esta razão, o experimento ALICE conta com um sistema de *trigger*, responsável por selecionar em tempo real as informações de colisões consideradas interessantes para serem salvas. O sistema de *trigger* funciona em três níveis hierárquicos de decisão[27], que representam análises dos eventos utilizando critérios de qualidade pré-estabelecidos[5]. Além disso, cada nível do *trigger* avalia apenas eventos que passaram pelos níveis anteriores.

A decisão do primeiro nível de *trigger* (L0) deve ser extremamente rápida, ocorrendo em $\approx 0,9$ µs após a colisão. Para isto, detectores com grande precisão temporal - como T0 e V0 - devem ser utilizados. Em seguida, a decisão de nível 1 (L1) é feita em até $\approx 6,5$ µs após L0. Finalmente, a decisão de nível 2 (L2) ocorre 100 µs após a colisão - note que este é o tempo de transporte de todas as cargas de ionização no volume da TPC. Por fim, eventos que passem por estes três níveis são enviados ao *High-Level Trigger* (HLT), que realiza uma rápida reconstrução e avalia se as informações obtidas são interessantes ou não para alguma análise física. Eventos selecionados pelo HLT são salvos em mídia permanente para futuras análises.

2.5 Produção de eventos com empilhamento

Durante a operação do ALICE, colisões adicionais a de interesse podem ocorrer, de modo que os produtos das múltiplas colisões acabam sendo salvas como um único evento. Este é o processo de empillhamento (*pile-up*) introduzido anteriormente. Neste trabalho, as colisões que ativaram o sistema de *trigger* - e que, em geral, são de maior interesse para análises físicas - serão denotadas como "colisões principais".

Existem dois tipos de *pile-up* que podem ocorrer nos experimentos do LHC:

- *Same-bunch pile-up*: Duas ou mais colisões ocorrem no mesmo *bunch crossing* da colisão principal. Os produtos destas colisões são registrados por todos os detectores.
- *Out-of-bunch pile-up*: As colisões adicionais ocorrem em *bunch crossings* anteriores ou posteriores a aquele que ativou o sistema de *trigger*. Neste cenário, os detectores são afetados de formas diferentes de acordo com a sua janela de aquisição

[5] Para mais detalhes, consultar [21]

Neste trabalho, haverá um foco maior em *out-of-bunch pile-up*, uma vez que este é o processo mais frequente em colisões Pb-Pb. Segundo documentações internas do ALICE, a ocorrência de *same-bunch pile-up* foi rara nos eventos salvos do *Run 2*. Portanto, o seu efeito é negligível na maioria (senão todas) as análises.

Essencialmente, a ocorrência de *pile-up* polui as informações salvas dos eventos, dificultando os processos de reconstrução e elevando a incerteza das medidas experimentais. Em particular, a qualidade de reconstrução das trajetórias é mais suscetível aos efeitos do *pile-up* em sistemas Pb-Pb, uma vez que a elevada multiplicidade destas colisões pode gerar uma ocupação extremamente alta na TPC. Nestes casos, o valor de χ^2 sobre o número de *clusters*[6] na TPC[7] tende a ser, em média, consideravelmente maior para eventos com *pile-up* (> 3) do que para eventos normais (< 2.5). Por conta desta menor qualidade de reconstrução, parte das trajetórias de eventos com *pile-up* nem sempre passam por critérios de seleção, não sendo aproveitadas para análises físicas.

O *out-of-bunch pile-up* também afeta as distribuições de DCA_z das trajetórias reconstruídas. Fundamentalmente, posição z_{trk} das trajetórias na TPC é determinada levando em consideração o tempo t_0 em que a colisão principal ocorreu e o tempo t_{trk} que as cargas de ionização levam pra chegar até os leitores de carga (posicionados em z_{TPC}):

$$z_{trk} = z_{TPC} - vD \times \Delta t, \tag{2.3}$$

em que vD (≈ 2.5 cm/μs) é a velocidade de deriva das cargas na TPC e $\Delta t = t_{trk} - t_0$. Por conta desta dependência temporal, as trajetórias de interações *pile-up* que ocorreram N_{BC} *bunch crossings* antes ou após a colisão principal acabam sendo deslocadas no eixo z (em direção aos extremos ou ao centro da TPC) por aproximadamente $0.7N_{BC}$ mm. Como resultado, os valores estimados para o DCA_z também são deslocados - tornando esta quantidade potencialmente útil para reconhecer eventos com *out-of-bunch pile-up*.

Com a ocorrência de *pile-up*, outro problema que surge é a degradação da identificação de elétrons pela TPC, que é feita pela estimativa da sua deposição específica de energia dE/dx. Segundo documentações internas do ALICE, devido a alta multiplicidade de partículas carregadas, o valor medido de dE/dx acaba sendo enviesado para os elétrons, prejudicando a análise da produção de partículas que decaem neste canal, como J/ψ. Além da identificação de partículas, o *pile-up* também adiciona incertezas nas análises físicas dos eventos. No ALICE, é comum que as distribuições de observáveis sejam normalizadas com relação ao número de eventos, assumindo que cada evento contém apenas uma colisão. No entanto, a existência de eventos com *pile-up* - que não respeitam esta suposição - inevitavelmente eleva a incerteza das normalizações.

[6] *Cluster* é o nome dado a um conjunto de sinais (*hits*) em um detector que pertencem a uma mesma trajetória.

[7] Esta é uma medida de qualidade das trajetórias reconstruídas na TPC. Quanto menor for este valor, mais confiável foi o ajuste da trajetória.

Finalmente, a consequência mais direta do *out-of-bunch pile-up* é o fato da multiplicidade dos eventos ser discordante para detectores com diferentes janelas de leitura. A princípio, isto significa que a estimativa da centralidade é limitada a detectores extremamente rápidos, como o V0. Porém, esta discrepância nas medidas de multiplicidade também é explorada pelo ALICE para tentar rejeitar eventos com *out-of-bunch pile-up* e, assim, reduzir os malefícios deste processo sobre as análises físicas. Esta seleção de eventos consiste em ajustar um corte sobre a correlação da amplitude total de sinal do detector V0 (*VZERO Amplitude*) com o número de trajetórias propagadas na TPC (*Num. of TPC Tracks*). Na prática, o corte ajustado seleciona apenas os eventos com *VZERO Amplitude* maior que um certo limiar, que é definido pela relação:

$$\text{VZERO Amplitude} = 3.4 \times 10^{-5}(\text{Num. of TPC Tracks})^2 + 1.42(\text{Num. of TPC Tracks}) - 300. \quad (2.4)$$

A correlação dos detectores V0 e TPC com o limiar de seleção pode ser vista na Figura 2.8. É possível notar uma distribuição alongada de *Num. of TPC Tracks* mesmo para eventos com baixas multiplicidades segundo o detector V0. Este efeito é um indicativo claro da ocorrência de *pile-up*, uma vez que a TPC salva as trajetórias de todas as possíveis colisões enquanto o V0 registra apenas os sinais da colisão principal. Para eventos contendo apenas a colisão principal, espera-se uma proporcionalidade de sinais entre estes dois detectores - o que pode ser percebido como a distribuição diagonal da Figura 2.8.

Esta seleção de eventos empregada pelo ALICE mantém $\approx 100\%$ dos eventos principais e elimina

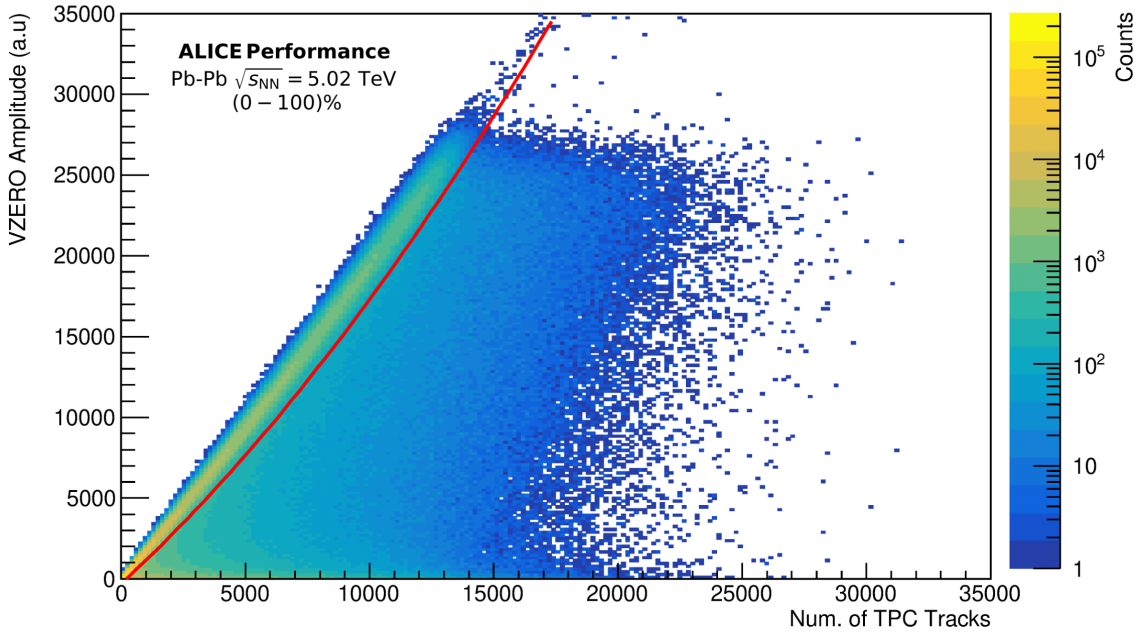


Figura 2.8: Correlação entre o número de trajetórias propagadas na TPC com a amplitude total do detector V0 para um conjunto de eventos. A linha vermelha representa o corte aplicado na correlação, que rejeita eventos no lado direito da curva.

cerca de $\approx 63\%$ dos eventos com *pile-up*. Perante o exposto, propomos aprimorar este método através da aplicação das técnicas de *Machine Learning*, que têm se mostrado promissoras não apenas em análises do LHC mas também em outras frentes de pesquisa.

Capítulo 3

Machine Learning e Ciência de Dados

Neste capítulo, forneceremos uma introdução à teoria do aprendizado de máquina. Os elementos fundamentais e os paradigmas envolvidos no aprendizado de algoritmos serão apresentados focando especialmente nas tarefas de classificação, que tiveram maior relevância no desenvolvimento deste trabalho. Finalmente, apresentaremos um conjunto de métodos, tanto estatísticos quanto baseados em aprendizado de máquina, que estimam a importância de cada variável do conjunto de dados.

3.1 Visão Geral

3.1.1 Elementos básicos do Aprendizado de Máquina

A teoria do Aprendizado de Máquina (do inglês, *Machine Learning*, ML) é voltada ao estudo de algoritmos capazes de extrair conhecimento de conjunto de dados e, com isso, criar modelos matemáticos que fazem predições e auxiliam na tomada de decisões[20, 28]. A extração de conhecimento (ou aprendizado) está relacionada com a capacidade de um algoritmo aumentar a sua performance em realizar tarefas conforme é exposto a dados disponíveis[1]. Com o processo de aprendizado, que ocorre de forma automatizada, busca-se obter um algoritmo que tenha a máxima capacidade de generalização - isto é, um modelo capaz de utilizar o conhecimento adquirido para trabalhar com dados nunca vistos anteriormente.

Em essência, o ML é uma área multidisciplinar fortemente baseada em estatística, modelagem matemática, otimização e ciência da computação - carregando também inspirações da neurociência, biologia e física. Mesmo sendo considerado uma sub-especialidade da Inteligência Artificial (IA), o ML não se propõe a desenvolver algoritmos que reproduzam as funções cognitivas humanas, sendo limitada a solução de problemas específicos.

Os tópicos a seguir resumem os elementos básicos e os passos gerais envolvidos na maioria das aplicações de ML:

[1] Por completeza, vale comentar existem estratégias para aproveitar aquilo que já foi aprendido por um certo algoritmo no aprendizado de outro. Abordagens deste tipo são conhecidas como pré-treinamento ou *Transfer Learning* [29].

- **Seleção de dados**: Inicialmente, os dados ou observações são coletados e agrupados. Neles, deverão estar contidos os padrões ou correlações a serem aprendidas pelos modelos de ML para a resolução da tarefa de interesse. Nesta etapa, também pode ser feito um pré-processamento dos dados, o que inclui: seleção de variáveis, mudanças de escala e tratamento de dados faltantes. Geralmente o grupo de dados pré-processado é repartido em conjuntos de treinamento e de teste.
  O aprendizado dos algoritmos de ML ocorre com a extração de conhecimento do grupo de treinamento, enquanto a qualidade do aprendizado sintetizado é avaliada com o grupo de teste. As observações destes dois conjuntos devem ser obrigatoriamente diferentes - evitando possíveis vieses e garantindo a independência dos processos de treinamento e teste.

- **Seleção da técnica de ML**: Nesta etapa, a técnica de ML deve ser escolhida. Em geral, esta escolha pode levar em consideração fatores como: adequação para realizar a tarefa de interesse, flexibilidade para lidar com não-linearidades, custo computacional e viabilidade de aplicação. Estes aspectos estão relacionados com as particularidades dos modelos matemáticos que caracterizam cada técnica. Contudo, vale destacar que uma característica comum dos algoritmos de ML é o fato dos seus respectivos modelos matemáticos possuírem parâmetros livres. São estes parâmetros que sofrem ajustes na etapa de aprendizado, de modo a tornar os algoritmos aptos a resolver problemas.

- **Configurações do treinamento e dos hiperparâmetros**: Em muitos casos, após a escolha da técnica de ML é necessário definir uma função objetivo, cujo propósito é o de avaliar a taxa de erro do algoritmo. Durante a etapa de treinamento, deseja-se a otimização desta função, que é feita por métodos que podem ou não ser definidos *a priori* pelo usuário. Em alguns casos, os modelos de ML já possuem métodos fixados de otimização, não cabendo ao usuário esta escolha. O objetivo da otimização é o de ajustar os parâmetros livres do algoritmo de ML de modo a minimizar (ou, em alguns casos, maximizar) a função objetivo[2]. O método de Newton e o método de gradiente descendente são exemplos de técnicas de otimização comumente empregadas, como será visto adiante.
  Além de escolher a função objetivo e o método de otimização, é necessário determinar os hiperparâmetros, que são parâmetros internos dos algoritmos de ML ajustados antes do treinamento. Os hiperparâmetros definem as condições em que o aprendizado ocorrerá, podendo representar tanto aspectos estruturais dos modelos quanto características do treinamento em si. Alguns exemplos de hiperparâmetros incluem: o número de iterações, a taxa de aprendizado, número de árvores em algoritmos baseados em árvores de decisão, número de neurônios em redes neurais artificiais e etc. A qualidade do treinamento esta sujeita ao ajuste adequado dos hiperparâmetros - como será

[2]Nos casos em que se busca a minimização, normalmente a função objetivo passa a ser denotada como função custo (do inglês, *loss function*).

discutido na Subseção 3.1.3.

- **Treinamento**: Etapa em que os dados do grupo de treinamento são utilizados para o aprendizado do modelo de ML. Formalmente, o treinamento consiste no ajuste dos parâmetros livres do algoritmo de modo a otimizar a função objetivo escolhida. O objetivo do treinamento é o de obter um modelo ajustado capaz de realizar com competência a tarefa desejada.

- **Teste e análise**: Após o treinamento, podem ser utilizadas métricas que avaliam o desempenho do algoritmo treinado em trabalhar com dados não vistos durante o treinamento. Normalmente, esta análise é feita com os dados do grupo de teste.

3.1.2 Paradigmas de aprendizado

A forma como os modelos aprendem na etapa de treinamento pode variar de acordo com as características dos dados e da tarefa que se pretende atacar. Existem, pelo menos, quatro formas do processo de aprendizado ocorrer[30]:

- **Aprendizado supervisionado**: As observações empregadas no treinamento já possuem as respostas (saídas), que são utilizadas para indicar o comportamento desejado para os modelos de ML. Um exemplo de técnica que utiliza aprendizado supervisionado são as *Random Forests* - como se verá na Seção 3.2.

  Para grande parte das aplicações do aprendizado supervisionado, dois espaços devem ser considerados: o espaço X das variáveis de entrada (*input*) e o espaço Y das saídas (*output*). No espaço n-dimensional X, cada observação i, $i = 1, ..., k$ do conjunto é caracterizada por um vetor $x_i = [a_{i0}, a_{i1}, a_{i2}, ..., a_{in}]$, cujos elementos representam as variáveis preditoras ou atributos (*features*). Para cada x_i, haverá um vetor de saída y_i correspondente no espaço Y que representa a resposta de interesse[3]. Logo, o objetivo do aprendizado supervisionado é o de sintetizar um modelo $f : X \rightarrow Y$ de modo que cada vetor de entrada x_i seja mapeado ao seu correspondente vetor de saída y_i - isto é, $f(x_i) = y_i$. A tarefa a ser realizada pelo algoritmo é de regressão se a saída desejada y_i for numérica. Caso y_i seja do tipo categórico, tem-se uma tarefa de classificação.

  Na classificação, busca-se atribuir rótulos aos dados de acordo com os seus atributos. Para uma tarefa de classificação binária, por exemplo, o espaço Y poderia ser caracterizado pelos valores {-1, 1} ou {0, 1}. As tarefas de regressão, por sua vez, buscam prever um número com base nos atributos dos dados - neste caso, o espaço das saídas poderia ser definido pelo conjunto dos números reais, por exemplo.

[3] É possível, para fins de ilustração, imaginar que o i-ésimo evento do ALICE em um certo *Run* pode ser representado por um vetor de atributos x_i. Neste caso, os elementos de x_i poderiam ser medidas (ou quantidades derivadas) de diversos detectores. Enquanto o vetor de saída y_i poderia indicar uma quantidade ou rótulo associado ao evento.

- **Aprendizado não-supervisionado**: As observações não possuem as respostas desejadas para os modelos de ML - isto é, apenas o espaço atributos X é conhecido. Existem dois problemas principais que podem ser resolvidos neste cenário: agrupamento e estimação de densidades.

  Em tarefas de agrupamento, o objetivo é classificar em um mesmo grupo os dados que possuam atributos similares. O algoritmo *k-means* é um exemplo tradicional de técnica de agrupamento [31].

  Em tarefas de estimação de densidades, o intuito é obter a lei de distribuição de probabilidades que supostamente gerou as observações de treinamento. Aprender esta lei permite a geração de dados sintéticos com atributos próximos ou iguais aos dados reais. A Rede Adversária Generativa (do inglês, *Generative Adversarial Network*) é um exemplo de técnica que faz estimação de densidades [32].

- **Aprendizado semi-supervisionado**: Apenas uma parte muito pequena dos dados possuem as respostas desejadas para os modelos de ML. Desta forma, os algoritmos treinados de forma semi-supervisionada devem ser capazes de aprender as informações destes poucos dados para operar com dados reais não rotulados. Um exemplo de algoritmo que opera via aprendizado semi-supervisionado são as *Graph Neural Networks* quando utilizadas para classificar nós em grafos.

- **Aprendizado por reforço**: Nesta categoria, não existe uma saída definida explicitamente. Os modelos são treinados a realizar uma sequência de ações, que serão avaliadas com uma recompensa ou uma punição. Portanto, o objetivo do treinamento por reforço é obter um modelo que maximize o ganho de recompensas a longo prazo - aprendendo, então, a melhor estratégia para realizar decisões sequenciais.

Este trabalho está totalmente baseado no aprendizado supervisionado, pelo fato de apenas dados simulados (e que já possuíam as saídas desejadas) terem sido utilizados - mais detalhes no Capítulo 4. Por esta razão, as descrições a seguir se concentrarão neste paradigma de aprendizado. Contudo, por completeza, realizamos algumas análises com os algoritmos de agrupamento k-means e DBSCAN, que se encontram no escopo do aprendizado não-supervisionado. A descrição deste estudo e dos resultados obtidos podem ser conferidos no Apêndice A.

3.1.3 Generalização e regularização

A flexibilidade (ou complexidade) dos algoritmos de ML representa a capacidade dos modelos de capturar as informações contidas nos dados de treinamento. Em geral, o grau de flexibilidade de um modelo depende do ajuste sobre os hiperparâmetros, que deve ser feito com cuidado. Caso um certo

modelo tenha uma flexibilidade muito alta, poderá sofrer com sobreajuste (do inglês, *overfitting*). Neste cenário, os algoritmos se ajustam excessivamente sobre os dados de treinamento e acabam apresentando uma baixa performance em trabalhar com dados inéditos (baixa capacidade de generalização). Por outro lado, um modelo com baixa flexibilidade não consegue tirar o melhor proveito das informações contidas nos dados de treinamento. Nesta situação, conhecida como sub-ajuste (do inglês, *underfitting*), os modelos treinados também podem apresentar uma performance ruim ao trabalhar com dados não vistos. A Figura 3.1 contém uma representação pictórica dos cenários de *overfitting* e *underfitting*.

Para diagnosticar cenários de *underfitting/overfitting*, uma solução comum é monitorar o erro dos modelos preditivos durante ou após o treinamento. O erro é monitorado separadamente sobre os conjuntos de treinamento e teste. Em alguns casos, um conjunto extra conhecido como grupo de validação é utilizado, cujo papel é o de exclusivamente monitorar a ocorrência de *underfitting/overfitting*. Idealmente, busca-se no treinamento um modelo que minimiza o erro junto aos dados de treinamento e de validação (e/ou teste). A Figura 3.2 exibe uma representação deste tipo de análise. A eficácia deste procedimento está relacionada com a independência dos conjuntos de treinamento, teste e validação - além disso, os conjuntos devem ser suficientemente representativos do problema a ser atacado . Outra solução para evitar o *overfitting* é recorrer a técnicas de regularização, que visam controlar a flexibilidade dos modelos de aprendizado . Muitas destas técnicas envolvem adicionar termos na funções objetivo, que irão penalizar o aprendizado e tornar o modelo menos complexo[4]. Algumas das técnicas de regularização mais conhecidas são: LASSO (regularização L1), *Ridge Regression* (regularização L2) e *Elastic Net* - que serão definidas em mais detalhes na Seção 3.2.2.

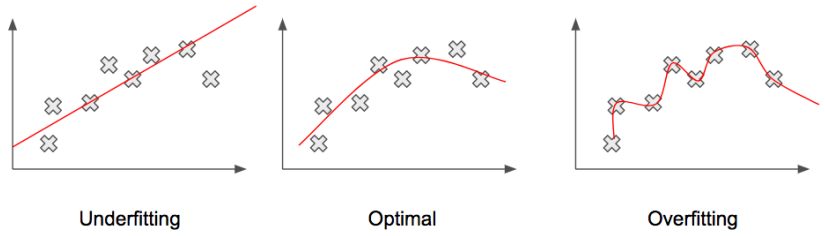


Figura 3.1: Representação pictórica dos possíveis ajustes feitos por um modelo de ML (linha vermelha) sobre uma distribuição qualquer de dados (em cinza). Os três cenários de ajuste estão ilustrados: *underfitting*, ajuste ótimo e *overfitting*. Figura retirada de [33].

[4]Existem outras formas da regularização que não necessariamente envolvem adicionar termos na função objetivo - como a técnica de *dropout* em redes neurais artificiais, por exemplo.

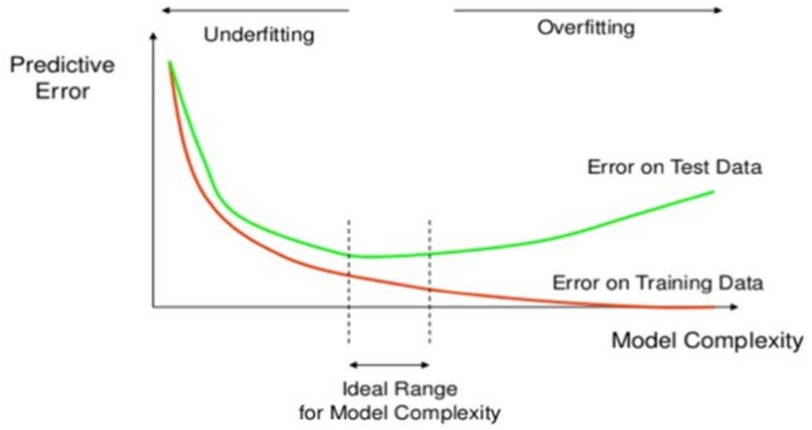


Figura 3.2: Representação do erro cometido por um modelo preditivo sobre conjuntos de treinamento e teste de acordo com a sua flexibilidade (complexidade). A flexibilidade ideal é aquela que minimiza conjuntamente o erro nos grupos de treinamento e teste (ou validação). Ajustes inadequados da flexibilidade podem levar a cenários de *underfitting* ou *overfitting*, que são caracterizados por erros maiores junto ao conjunto de teste/validação. Figura retirada de [34].

3.2 Técnicas de classificação

Nesta seção, serão apresentadas algumas técnicas de classificação que tiveram maior importância para este trabalho. A descrição dos modelos será feita sem grande aprofundamento matemático. Uma descrição mais detalhada de algumas passagens matemáticas e das motivações envolvidas por trás de algumas das técnicas podem ser facilmente encontradas na literatura.

3.2.1 Redes Neurais Artificiais

Rede Neural Artificial (do inglês, *Artificial Neural Network*, NN) é um modelo de ML baseado no funcionamento do cérebro humano. Por esta razão, a unidade fundamental das NNs é o neurônio artificial (também conhecido como perceptron)[35]. Matemáticamente, a saída de um neurônio artificial k, considerando um vetor de entrada $x_i = [a_{i0} = 1, a_{i1}, ..., a_{in}]$, é:

$$y_k = f(\sum_{j=0}^{n} w_{kj} a_j) = f(w^T x_i), \tag{3.1}$$

em que $w = [w_{k0}, w_{k1}, ..., w_{kn}]$ é o vetor de pesos sinápticos e $f()$ é a função de ativação. O termo de limiar (*bias*) w_{k0} é incluído para evitar que vetores de entrada exclusivamente nulos sempre produzam saídas nulas. A Figura 3.3 mostra uma representação pictórica do modelo matemático do neurônio.

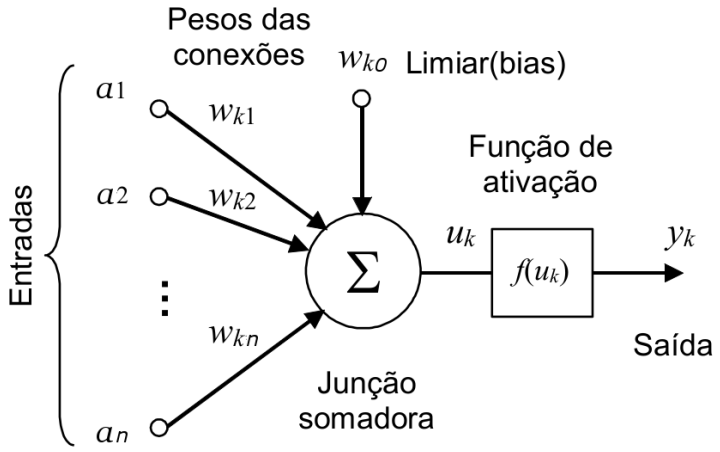


Figura 3.3: Modelo matemático de um neurônio artificial. Figura adaptada de [36].

As funções de ativação $f()$ transformam a soma ponderada das entradas na saída do neurônio. De forma prática, são as funções de ativação que definem tanto o limiar para os sinais de entrada ativarem os neurônios quanto a intensidade dos sinais de saída. Existem diversas funções de ativação conhecidas da literatura, que podem ser lineares ou não-lineares - alguns exemplos incluem:

- Identidade:

$$f(x) = x \tag{3.2}$$

- Degrau:

$$f(x) = 1, \text{ se } x \geq 0$$
$$f(x) = 0, \text{ se } x < 0$$

- Tangente Hiperbólica:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.3}$$

- *Rectified Linear Unit* (ReLU):

$$f(x) = max(0, x) \tag{3.4}$$

- *Softmax*:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3.5}$$

A escolha das funções de ativação deve ser feita pelo usuário *a priori*.

Rede perceptron de múltiplas camadas

Devido a sua menor flexibilidade, os neurônios sozinhos podem não ser adequados para resolver problemas não-lineares de classificação. Uma alternativa para atacar problemas mais complexos consiste em agregar sequencialmente camadas de neurônios, formando então as redes neurais artificiais. Uma das arquiteturas mais conhecidas dentre as NNs é o perceptron de múltiplas camadas (do inglês, *Multilayer perceptron*, MLP)[5]. Como todas as camadas nesta estrutura são plenamente conectadas, a saída de cada neurônio em uma certa camada é a entrada para todos os neurônios da próxima camada. Além disso, todos os neurônios de cada camada estão sujeitos a uma função de ativação específica. Normalmente, funções de ativação não-lineares são utilizadas nas redes MLP, permitindo que estas arquiteturas consigam combinar os sinais de entrada de formas mais complexas e extrair padrões não-lineares dos dados.

Na arquitetura MLP, existem três blocos de camadas fundamentais: (1) a camada de entrada (*input*), (2) as camadas intermediárias ou ocultas (*hidden layers*) e (3) a camada de saída (*output*) - como mostra a Figura 3.4. Se uma rede tiver duas ou mais camadas intermediárias, é chamada de rede neural profunda (do inglês, *Deep Neural Network*). Tradicionalmente, o número de neurônios na camada de saída é igual ao número de classes para problemas de classificação. Ademais, é comum utilizar a função *softmax* na última camada, que restringe os valores de saída ao intervalo $[0,1]$ permitindo interpretações probabilísticas.

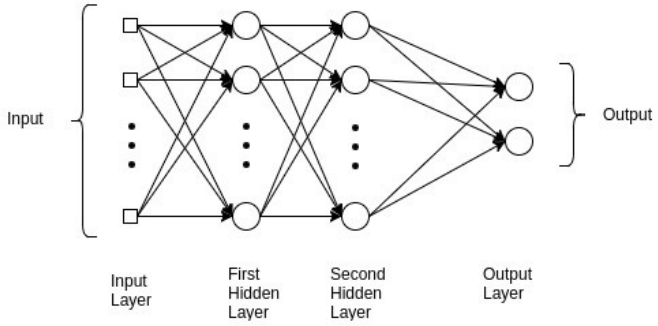


Figura 3.4: Representação da arquitetura de uma rede neural do tipo MLP. Figura retirada de [37].

O aprendizado das redes neurais ocorre com o ajuste de seus pesos sinápticos. Isto é feito aplicando métodos de otimização sobre uma função custo, que avalia os erros durante o treinamento. Para tarefas de classificação, existem duas funções custo comuns que avaliam a diferença entre as saídas obtidas $\hat{y}_l$ e as saídas esperadas y_l ($l = 1,2,...,N$):

[5]Neste texto, daremos foco exclusivo as redes do tipo MLP, que estão no escopo das redes neurais não-recorrentes. Mais informações sobre outros tipos de redes neurais podem ser vistas em [36].

- Erro quadrático médio:

$$L = \frac{1}{N}\sum_{l=1}^{N}(\hat{y}_l - y_l)^2. \tag{3.6}$$

- Entropia Cruzada (*Cross-Entropy*):

$$L = -\frac{1}{N}\sum_{l=1}^{N}(y_l \log(\hat{y}_l) + (1-y_l)\log(1-\hat{y}_l)). \tag{3.7}$$

Na prática, o processo de aprendizado nas redes neurais pode ser dividido em duas etapas sequenciais: *forward* e *backward propagation*. Na etapa de *forward propagation*, é feita a soma ponderada dos elementos de cada vetor de entrada x_i através dos pesos sinápticos da camada intermediária. Em seguida, a respectiva função de ativação é aplicada, gerando uma saída que é utilizada como entrada para os neurônios da próxima camada. Este processo se repete até a última camada da rede, em que um valor de saída é produzido e comparado, pela função custo, com a saída esperada. Após isto, se inicia o processo de *backward propagation*, em que os pesos sinápticos da camada de saída até a camada de entrada são ajustados visando a minimização do erro estimado pela função custo.

Considerando uma rede neural com P pesos sinápticos, pode-se imaginar a função custo $L()$ como uma superfície de erro sobre o espaço $\mathbb{R}^{P+1}$. Neste sentido, o objetivo do *backward propagation* é o de encontrar o ponto em $\mathbb{R}^P$ que minimiza $L()$. Esta busca é feita de forma iterativa através do método de gradiente descendente, que calcula as derivadas parciais da função custo em relação aos pesos sinápticos de cada camada. Logo, a k-ésima atualização de um peso sináptico qualquer da rede neural é dada pela regra geral:

$$w_{k+1} = w_k - \lambda * \nabla L(w_k), \tag{3.8}$$

em que λ é um hiperparâmetro conhecido como taxa de aprendizado (do inglês, *learning rate*), que dita o tamanho do passo sobre a superfície de erro e, portanto, a velocidade de ajuste sobre os pesos sinápticos. O procedimento de *backward propagation* acaba quando os pesos sinápticos de todas as camadas forem atualizados. A Figura 3.5 ilustra um exemplo de superfície de erro para pesos sinápticos no espaço $\mathbb{R}^2$.

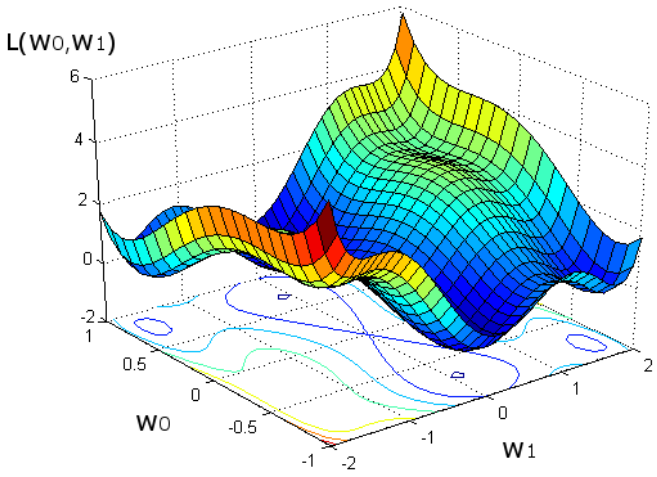


Figura 3.5: Representação de uma superfície de erro definida por pesos sinápticos em $\mathbb{R}^2$. Figura adaptada de [38].

Durante o treinamento das redes neurais, os processos de *forward* e *backward propagation* são repetidos inúmeras vezes até um critério de parada ser satisfeito. Um dos critérios mais comuns é o de realizar o treinamento até o número máximo de épocas ser atingido, em que uma época indica quando todas as observações de treinamento são apresentadas à rede neural. Outros critérios incluem definir um limiar para o erro cometido pela rede ou encerrar o treinamento quando o erro junto aos dados do conjunto de validação for mínimo (técnica conhecida como *holdout*). Definir qual critério de parada será utilizado garante um ajuste adequado da rede neural, o que evita cenários indesejados de *underfitting* ou *overfitting*[36].

A expressão 3.8 é a regra mais básica de atualização dos pesos sinápticos e está restrita a um valor fixo de *learning rate*. A convergência do treinamento pode ser lenta se o *learning rate* for muito pequeno, enquanto a convergência pode não ocorrer no caso do *learning rate* ter um valor muito elevado. Para resolver estes problemas podem ser empregados algoritmos de otimização adaptativos, que ajustam a taxa de aprendizado conforme o treinamento progride. Um dos mais conhecidos é o *Adaptive Moment Estimation* (ADAM)[39], que ajusta taxas de aprendizado específicas para cada peso da rede neural[6]. Formalmente, a regra de atualização do algoritmo ADAM para o k-ésimo ajuste de um peso sináptico qualquer é:

$$w_{k+1} = w_k - \frac{\lambda}{\sqrt{\hat{v}_k} + \varepsilon} \hat{m}_k, \tag{3.9}$$

em que:

$$\hat{m}_k = \frac{\gamma_1 m_{k-1} + (1-\gamma_1)\nabla L(w_k)}{1-\gamma_1^k}, \tag{3.10}$$

[6]Outros exemplos de algoritmos adaptativos incluem o *Stochastic Gradient Descent* (SGD) com *Momentum*, o Adagrad e o Adadelta.

$$\hat{v_k} = \frac{\gamma_2 v_{k-1} + (1-\gamma_2)(\nabla L(w_k))^2}{1-\gamma_2^k}. \tag{3.11}$$

Na implementação do ADAM, os vetores iniciais m_0 e v_0 são nulos e os hiperparâmetros γ_1, γ_2 e ε podem ser ajustados pelo usuário - os valores recomendados são: $\gamma_1 = 0,9$, $\gamma_2 = 0,999$ e $\varepsilon = 10^{-8}$.

Graph Neural Networks

As aplicações de *Deep Learning* ganharam popularidade pela sua versatilidade em trabalhar com dados Euclidianos - isto é, que possuem uma estrutura fixa, como imagens, tabelas, textos, aúdios e vídeos. Contudo, a maioria das suas arquiteturas tradicionais, como a MLP ou as Redes Convolucionais, não são capazes de lidar com dados na forma de grafos, que são não-Euclidianos. Eis que nos últimos anos surgiu uma nova arquitetura conhecida como *Graph Neural Network* (GNN), desenvolvida com o intuito de expandir os conceitos tradicionais de *Deep Learning* para operar especificamente com dados deste tipo [40].

Os grafos são estruturas irregulares de dados capazes de descrever objetos e suas relações. Formalmente, a arquitetura de um grafo G é composta por um grupo de nós (do inglês, *nodes*, V) que se relacionam por um conjunto de conexões (do inglês, *edges*, E):

$$G = G(V,E). \tag{3.12}$$

Um grafo é não-direcionado se as suas conexões forem bi-direcionais entre os nós, caso contrário ele é direcionado. Matematicamente, um grafo G com n nós pode ser definido por uma matriz de adjacência A com tamanho $n \times n$, cujos elementos a_{ij} indicam se dois nós i e j quaisquer estão conectados. A Figura 3.6 exemplifica um grafo e a matriz de adjacência correspondente. Geralmente, os nós também são caracterizados por um vetor de atributos.

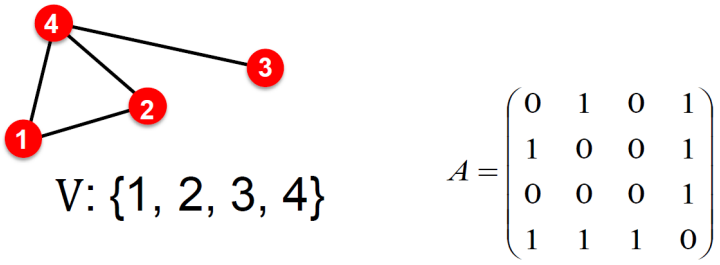


Figura 3.6: Estrutura de um grafo indireto com um conjunto V de nós e a respectiva matriz de adjacência A. Para nós conectados, os elementos de A são iguais a um - caso contrário, são iguais a zero. Figura retirada de [41].

Em geral, grafos são estruturas complexas de se analisar por algumas razões:

- Essencialmente, grafos não residem no espaço Euclidiano. Isto significa que pode ser pouco intuitivo realizar interpretações de sua estrutura em representações 2D ou 3D.
- Grafos não possuem uma forma ou tamanho fixos, implicando que dois grafos quaisquer, visualmente distintos, podem ter matrizes de adjacência parecidas.
- Em geral, os nós de um grafo não são dispostos de forma ordenada ou respeitando qualquer sequência.
- Além das técnicas tradicionais de ML (como uma rede MLP, algoritmos baseados em árvores, Regressão Logística e etc) não conseguirem lidar com dados de estrutura irregular, eles assumem uma independência entre as observações. Isto pode não ser verdade para grafos, uma vez que os nós se relacionam explicitamente por conexões.

Alguns dos problemas que podem ser atacados pela GNN sobre estas estruturas incluem: classificação de nós, previsão de conexões e classificação de grafos [40]. A classificação de nós envolve rotular cada nó de um grafo com base nos seus atributos e ligações - normalmente, os treinamentos são semi-supervisionados e ocorrem sobre um único grafo. Com a previsão de conexões, busca-se detectar se dois nós quaisquer de um grafo podem ter alguma ligação ou não. Finalmente, a classificação de grafos consiste na atribuição de rótulos a grafos inteiros com base nas suas características estruturais e nos atributos dos seus nós - nesta tarefa, os treinamentos e testes são feitos sobre vários grafos.

Em geral, a idéia chave por trás das GNNs é a de gerar uma representação matemática alternativa para os grafos que possibilite a aplicação de métodos tradicionais de ML. Isto é feito através do mecanismo de *Message Passing*, cujo objetivo é o de mapear os atributos dos nós em um espaço d-dimensional (conhecido como *node embedding*) capturando, ao mesmo tempo, a topologia do grafo. Para isso, o mapeamento é otimizado de modo que as distâncias no *node embedding* reflitam às posições relativas aos nós no grafo original [40]. A Figura 3.7 exemplifica este procedimento.

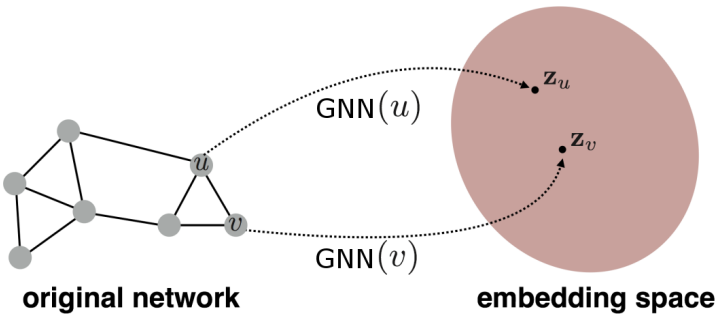


Figura 3.7: Ilustração do mapeamento dos nós de um grafo para o *node embedding* através da GNN. Figura modificada de [40].

O mapeamento via *Message Passing* está fundamentado no compartilhamento de informações entre nós vizinhos. Formalmente, para o k-ésimo passo de *Message Passing*, o vetor de atributos x_u^k de cada nó $u \in V$ em um grafo $G = G(V,E)$ é atualizado com as informações agregadas da sua vizinhança $N(u)$. Esta atualização, que ocorre paralelamente para cada nó de G, é expressa como:

$$x_u^{(k+1)} = f(W_{self}^{(k)} x_u^{(k)} + W_{neigh}^{(k)} \sum_{v \in N(u)} x_v^{(k)} + b^{(k)}), \tag{3.13}$$

em que $W_{self}^{(k)}$ e $W_{neigh}^{(k)} \in \mathbb{R}^{d^{(k+1)} \times d^{(k)}}$ são parâmetros treináveis, f é a função de ativação e $b^{(k)} \in \mathbb{R}^{d^{(k+1)}}$ é o termo de limiar (*bias*). Cada camada da GNN representa um passo de *Message Passing*, de modo que para $k = 0$, os atributos dos nós são iguais aos seus atributos de entrada originais. Além disso, a dimensão d das camadas é um hiperparâmetro e pode ser definida pelo usuário *a priori*.

De forma prática, o que motiva o *Message Passing* é o fato de que os nós conectados - e que, portanto compartilharam informações - terão atributos similares e tenderão a ser mapeados próximos no *node embedding*. É desta forma que a GNN efetivamente captura as informações estruturais dos grafos.

Após algumas iterações de *Message Passing*, o *node embedding* sintetizado pode ser utilizado de diferentes formas para realizar a tarefa de interesse. Especificamente para a classificação de grafos, emprega-se uma camada de leitura para agregar ou resumir as informações do *node embedding*. Este procedimento cria um novo espaço conhecido como *graph embedding*, em que cada ponto sintetiza as características de um grafo de treinamento. É sobre o *graph embedding* que se treina um classificador final - normalmente uma rede MLP - para realizar as previsões. A Figura 3.8 exemplifica uma arquitetura dedicada a classificação de grafos.

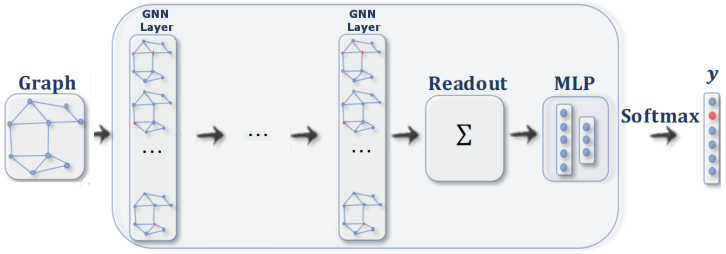


Figura 3.8: Exemplo de uma arquitetura básica para a classificação de grafos. As camadas da GNN são seguidas por uma camada de leitura, que resume as informações do *node embedding* em um *graph embedding*. Uma rede MLP é treinada sobre o *graph embedding* para realizar as classificações. Figura modificada de [42].

Na literatura, existem muitas camadas de leitura disponíveis, sendo a mais comum a *Global Mean Pool* que toma a média dos vetores de atributos $x_i^{(NE)}$ do *node embedding*:

$$x_G = \frac{1}{N_G} \sum_{i=i}^{N_G} x_i^{(NB)}. \tag{3.14}$$

A relação 3.13 é a mais tradicional e básica do mecanismo de *Message Passing* para uma camada

da GNN. Porém, existem algumas modificações desta formulação que não serão mostradas aqui mas que podem ser enconstradas facilmente na literatura, como: *Graph Convolutional Network* (GCN) [40], GraphConv [43], GraphSage [44] e *Graph Attention Networks* (GAT) [45]. Finalmente, vale ressaltar que o treinamento das GNNs ocorre da mesma que forma que nas redes neurais tradicionais, através dos processos de *forward-backward propagation* e da otimização baseada no método de gradiente descendente.

3.2.2 Regressão Logística

Problemas de regressão podem ser abordados por técnicas de ML que utilizam modelos lineares em sua estrutura. Para técnicas lineares, a saída desejada y_i de uma observação $x_i = [1, a_{i1}, a_{i2}, ..., a_{ik}]$ pode ser obtida por uma combinação linear dos atributos de entrada:

$$y_i = x_i^T w = w_0 + w_1 a_{i1} + w_{i2} a_{i2} + ... + w_k a_{ik}, \tag{3.15}$$

em que $w = [w_0, w_1, w_2, ..., w_k]$ é o vetor de parâmetros livres, que deve ser ajustado no treinamento. Este conceito por trás dos algoritmos lineares também pode ser expandido para problemas de classificação, como é o caso da Regressão Logística (do inglês, *Logistic Regression*)[46].

A Regressão Logística é um modelo linear de classificação capaz de fornecer a probabilidade de uma dada observação pertencer a uma certa categoria (classe). Apesar de ser expansível para cenários multi-classe, é utilizada especialmente em problemas de classificação binária[7]. A interpretação probabilística na Regressão Logística é possível pelo uso de uma função sigmóide, que restringe os valores da saída $\hat{y}_i$ ao intervalo $[0, 1]$:

$$\hat{y}_i = \frac{1}{1 + e^{-x_i^T w}}. \tag{3.16}$$

A expressão 3.16, que pode ser generalizada como $g(Z) = \frac{1}{1+e^{-Z}}$, é denominada função logística e está ilustrada na Figura 3.9.

Igual as técnicas de regressão lineares, os elementos do vetor de pesos w da Regressão Logística devem ser ajustados durante a fase de treinamento. O ajuste é feito de modo que a probabilidade $\hat{y}_i$ predita pelo modelo para uma certa observação x_i seja a mais próxima da sua classe verdadeira.

A função custo utilizada no ajuste dos coeficientes de w é obtida pelo método da máxima verossimilhança (do inglês, *maximum-likelihood*)[30]. Matematicamente, a saída da Regressão Logística segue a distribuição de Bernoulli, o que significa que a probabilidade P de observar o valor y_i de uma observação

[7] Neste trabalho, a descrição do modelo de Regressão Logística está feita exclusivamente sobre caso de classificação binária. Para ver a extensão do modelo para cenários multi-classe, consultar [30]

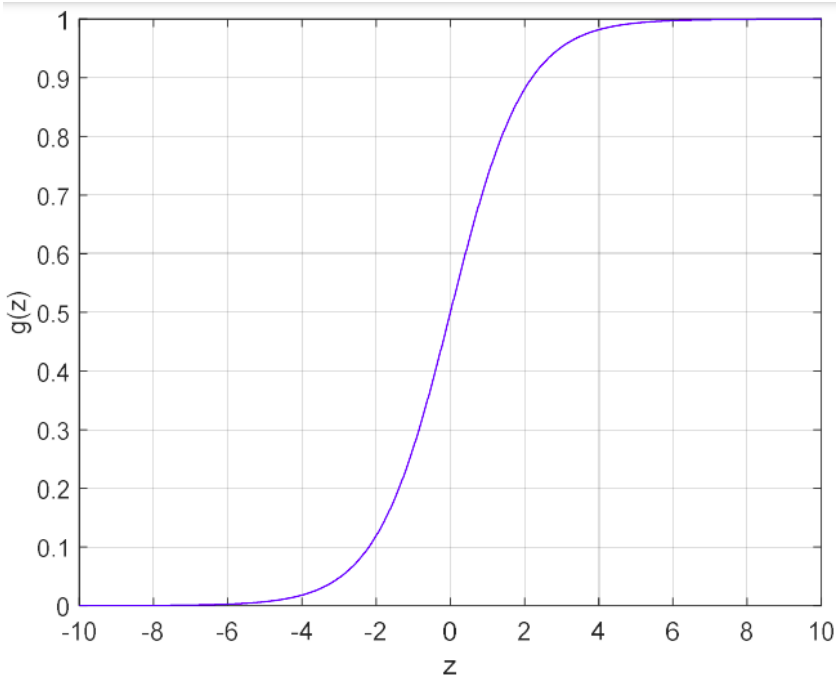


Figura 3.9: Gráfico da função logística. Figura retirada de [30].

x_i pode ser escrita de forma compacta como:

$$P(y_i) = \hat{y_i}^{y_i}(1-\hat{y_i})^{1-y_i}, \tag{3.17}$$

em que $P(y_i = 1) = \hat{y_i}$ e $P(y_i = 0) = 1 - \hat{y_i}$. Assumindo que cada uma das N observações no grupo de treinamento é independente, a função de verossimilhança para o conjunto é:

$$P(y_1, y_2, ..., y_N) = \prod_{i=1}^{N} P(y_i) = \prod_{i=1}^{N} \hat{y_i}^{y_i}(1-\hat{y_i})^{1-y_i}, \tag{3.18}$$

que, com a aplicação da função $-\log()$, pode ser reescrita como:

$$L(y_1, y_2, ..., y_N) = -\log(P(y_1, y_2, ..., y_N)) = -\sum_{i=1}^{N} y_i \log(\hat{y_i}) + (1-y_i)\log(1-\hat{y_i}). \tag{3.19}$$

A expressão de Entropia Cruzada 3.19 é a função custo a ser minimizada no ajuste da Regressão Logística. É comum que a Entropia Cruzada seja aprimorada com a adição de um termo de regularização $R(w)$ para reduzir as chances de *overfitting*. Adicionando este termo e rearrajando a relação 3.19, a função custo passa a ter a forma final:

$$L(X, y; w) = R(w) + C\sum_{i=1}^{N} \log(e^{-y_i(x_i^T w)} + 1), \tag{3.20}$$

em que C é a constante de regularização[8]. Alguns termos de regularização comumente utilizados são:

[8]Quanto menor for o valor desta constante, maior será a regularização sobre o modelo.

- LASSO (regularização L1):

$$R(w) = \sum_{i=1}^{k} |w_i|. \tag{3.21}$$

- *Ridge Regression* (regularização L2):

$$R(w) = \frac{1}{2} w^T w. \tag{3.22}$$

- *Elastic-Net*:

$$R(w) = \frac{1-\alpha}{2} w^T w + \alpha \sum_{i=1}^{k} |w_i|, \tag{3.23}$$

em que o hiperparâmetro $\alpha \in [0, +1]$.

Durante a fase de treinamento, a Eq. 3.20 pode ser minimizada por diferentes métodos iterativos. Pelo método de Gradiente Descendente, a k-ésima atualização do vetor de coeficientes w é dado por:

$$w_{k+1} = w_k - \lambda * \nabla L(w_k), \tag{3.24}$$

em que λ é a taxa de aprendizado. Outra técnica amplamente utilizada na otimização da Eq. 3.20 é o Método de Newton, em que a atualização do vetor de coeficientes é dada por:

$$w_{k+1} = w_k - (\nabla^2 L(w_k))^{-1} \nabla L(w_k). \tag{3.25}$$

O ajuste do vetor w ocorre até o número máximo de iterações ser atingido ou até a função custo cumprir algum critério de parada pré-determinado. Finalmente, vale destacar que existem ainda outros métodos de otimização que podem ser empregados no treinamento da Regressão Logística e que não serão detalhados aqui, como: *Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm* (LBFGS) e *Stochastic Average Gradient* (SAG).

3.2.3 Métodos baseados em árvores de decisão

Os métodos baseados em árvores têm como estratégia dividir o espaço de atributos de um conjunto de dados, de modo que as amostras com a mesma classe (ou valor-alvo) fiquem concentradas em uma mesma região. Nestes métodos, o aprendizado é baseado no ajuste de cortes ótimos sobre os atributos, que depois são empregados para classificar dados não vistos durante o treinamento. Existem diferentes técnicas que estão fundamentadas neste princípio, as principais estarão descritas na sequência.

Árvores de decisão

As árvores de decisão (do inglês, *decision trees*) são estruturas que dividem o espaço de atributos em um conjunto de regiões retângulares, que são utilizadas para determinar a saída do modelo preditivo[47].

Formalmente, as árvores de decisão são grafos em que dois vértices/nós quaisquer são ligados por

apenas um único caminho. Nesta estrutura, existe um "nó raíz"(*input* do modelo) que realiza a primeira partição no espaço de atributos, criando dois sub-nós[9] - e, portanto, dois sub-espaços iniciais. Os sub-nós podem ser rotulados como "nós de decisão", caso se propaguem em novos sub-nós, ou como "nós folha", que geram a variável resposta do modelo (*output*).

Logo, em uma árvore de decisão treinada, uma certa entrada é mapeada na saída desejada ao ser propagada do nó raíz até um nó folha. Note também que os nós de decisão representam os valores de corte sobre os atributos, de modo que a passagem de informações por eles está sujeita a condições binárias. A Figura 3.10 ilustra um exemplo de árvore de decisão e a respectiva partição no espaço formado por atributos genéricos X_1 e X_2 - os cortes nos atributos e os sub-espaços produzidos estão denotados, respectivamente, pelas letras t e R.

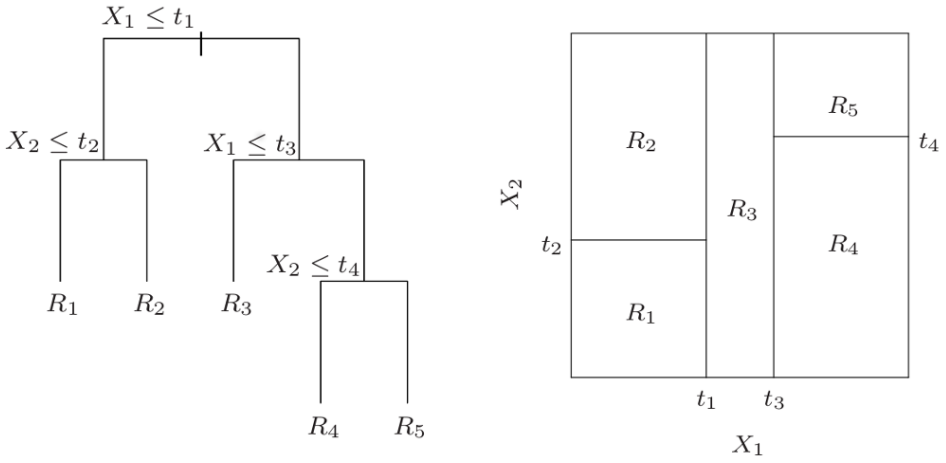


Figura 3.10: Exemplo de árvore de decisão e a partição correspondente do espaço de atributos bi-dimensional. Os atributos estão indicados por X, enquanto os valores de corte e as regiões do espaço estão representadas por t e R, respectivamente. Figura retirada de [47].

Uma das formas de se construir uma árvore de decisão é utilizando o algoritmo CART (do inglês, *Classification And Regression Tree*)[10]. Este algoritmo define como é feita a construção de um nó qualquer m da árvore a partir de um conjunto de dados Q_m - em que Q_m contém N exemplos de treinamento, com vetores de atributos $x_i = [a_{i1}, a_{i2}, a_{i3}, ..., a_{in}]$ e saídas y_i, $i = 1, 2, ..., N$. Para cada atributo de entrada é feita uma busca pelo ponto ótimo de corte. Este é um procedimento numérico e cobre todo o intervalo de valores dos atributos.

Formalmente, para o j-ésimo atributo, $j = 1, 2, ..., n$, tem-se que um valor de corte qualquer t_{jm} cria

[9] Estamos considerando um modelo comum de árvore de decisão, em que as decisões são feitas com base em condições binárias - isto é, cada processo de divisão gera apenas duas saídas. Contudo, de forma mais ampla, a construção das árvores de decisão não está restrita a esta abordagem.

[10] A construção de árvores de decisão não está restrita ao algoritmo CART. Existem outras abordagens, como ID3 e C4.5 [48].

dois subconjuntos Q_{jm}^{left} e Q_{jm}^{right}, definidos para todo i por:

$$Q_{jm}^{left} = \{(x,y)|a_{ij} <= t_{jm}\}$$
$$Q_{jm}^{right} = \{(x,y)|a_{ij} > t_{jm}\}$$

A qualidade do corte t_{jm} é avaliada usando uma função custo H, que mede a impureza dos subconjuntos Q_{jm}^{left} e Q_{jm}^{right} gerados. O grau de impureza medido pelas funções H representa o quanto um certo conjunto de dados possui elementos de uma mesma classe. A impureza é nula se todos os dados pertencerem a uma única classe e máxima se as classes estiverem uniformemente distribuídas no conjunto. Algumas medidas comuns de impureza, considerando K classes, são:

- Índice de Gini:
$$H(Q_m) = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}). \tag{3.26}$$
- Entropia:
$$H(Q_m) = -\sum_{k=1}^{K} p_{mk} * \log p_{mk}. \tag{3.27}$$
- Erro de Classificação:
$$H(Q_m) = 1 - p_{mk}, \tag{3.28}$$
em que:
$$p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k). \tag{3.29}$$

A Figura 3.11 ilustra, para K=2, as medidas de impureza em função da proporção de uma das classes.

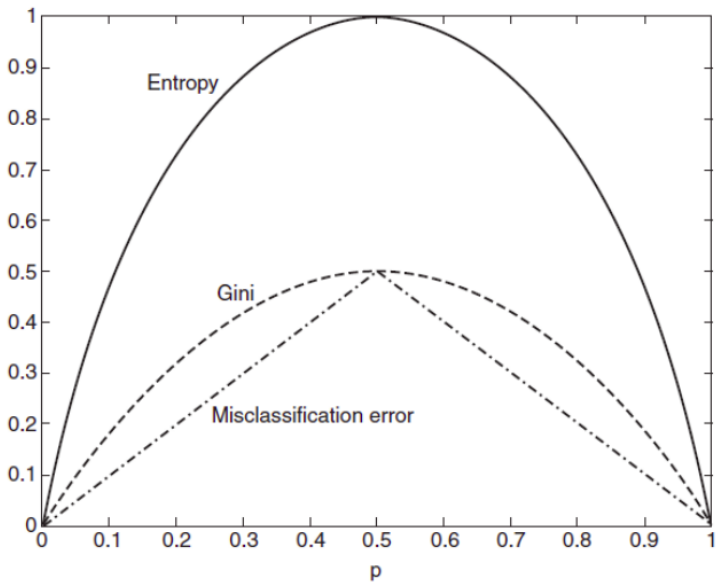


Figura 3.11: Relações de impureza (Gini, Entropia e erro de classificação) para $k = 2$ em função da proporção de uma das classes. Figura retirada de [49].

Portanto, de todas as partições possíveis para o j-ésimo atributo, define-se o corte ótimo $t*_{jm}$ como

aquele que minimiza a função H escolhida. Este procedimento é repetido para cada um dos n atributos disponíveis. Por fim, o atributo $j*$ escolhido para dividir os dados em m é aquele cujo corte ótimo $t*_{j*m}$ maximiza o Ganho de Informação (GI):

$$GI(Q_m, t*_{j*m}) = H(Q_m) - \frac{N_{left}}{N} H(Q*_{j*m}^{left}) - \frac{N_{right}}{N} H(Q*_{j*m}^{right}), \tag{3.30}$$

Todo este procedimento é repetido na inclusão de novos nós na árvore. No caso de m ser um nó-folha, a saída é simplesmente p_{mk}, que representa a probabilidade de uma certa entrada da árvore pertencer a classe k. Em geral, novos nós são adicionados até a profundidade máxima da árvore ou a quantidade mínima de dados (ajustadas *a priori*) em um certo nó serem atingidas.

Random Forest

Apesar de serem de fácil interpretação e pouco custosas computacionalmente, as árvores de decisão são muito propensas a cenários de *overfitting*. Ajustes inadequados de hiperparâmetros, como a profundidade da árvore, podem levar a partições muito pequenas no espaço de atributos - o que pode tornar as árvores de decisão especialistas em classificar apenas os dados de treinamento. Neste contexto, a *Random Forest* surgiu como uma alternativa robusta e menos sensível a ocorrência de *overfitting*.

A idéia fundamental por trás das *Random Forests* é a de utilizar múltiplas árvores de decisão para fazer as previsões[47, 50, 51]. As árvores são construídas com subconjuntos do grupo de treinamento, que possuem o mesmo tamanho do conjunto total por serem criados com reposição[11] - o que significa que os mesmos dados podem ser utilizados mais de uma vez. Além disso, os subconjuntos possuem apenas uma parcela, escolhida aleatoriamente, dos atributos originais. Devido a esta natureza aleatória, as *Random Forests* acabam se mostrando menos propensas a cenários de *overfitting* e tendem a superar a performance das árvores de decisão tradicionais.

Por fim, após o treinamento, a saída da *Random Forest* é obtida ao agregar as saídas individuais de todas as árvores de decisão ajustadas. Em tarefas de classificação, isto é feito através do voto majoritário, em que a predição da *Random Forest* é igual a classe mais votada dentre todas as árvores. Caso a saída tenha que ser probabilística, toma-se o número de votos atribuídos a cada classe dividido pelo número total de árvores. A Figura 3.12 ilustra um classificador *Random Forest* com três árvores de decisão.

[11] Este tipo de amostragem é conhecida como *Bootstrap*.

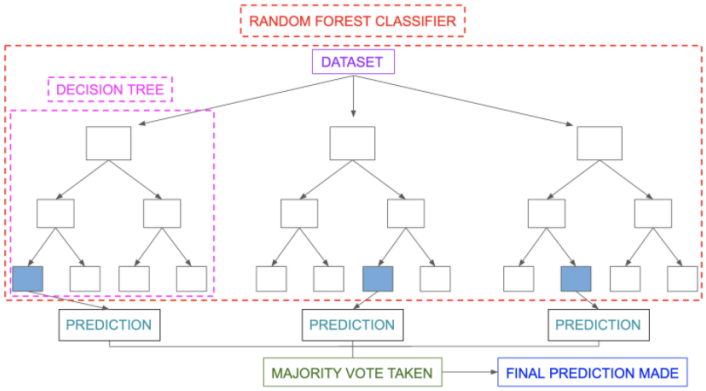


Figura 3.12: Visão simplificada do processo de classificação de uma *Random Forest* contendo três árvores de decisão. Figura retirada de [52].

Boosted Decision Tree

O *gradient boosting* é uma abordagem de ML que consiste em utilizar um comitê de máquinas formado por classificadores fracos para realizar predições[47]. Classificadores fracos são modelos simples, cuja performance é ligeiramente superior à de um classificador aleatório. O treinamento pelo método *gradient boosting* consiste em adicionar os classificadores fracos em sequência - de modo que cada novo classificador seja ajustado para corrigir os erros (*residuals*) do classificador anterior. Deste modo, com a adição de muitos modelos fracos, deseja-se gerar um comitê de máquinas robusto o suficiente para realizar predições acuradas.

A idéia de *gradient boosting* é aplicável para muitos modelos. Para o caso específico das *Boosted Decision Trees* (BDT), os classificadores fracos são as árvores de decisão, que a princípio são rasas e possuem poucos nós. Como as BDTs estão basedas em árvores de decisão, os seus principais hiperparâmetros são o número e profundidade das árvores. A Figura 3.13 apresenta a arquitetura de uma BDT com k árvores de decisão.

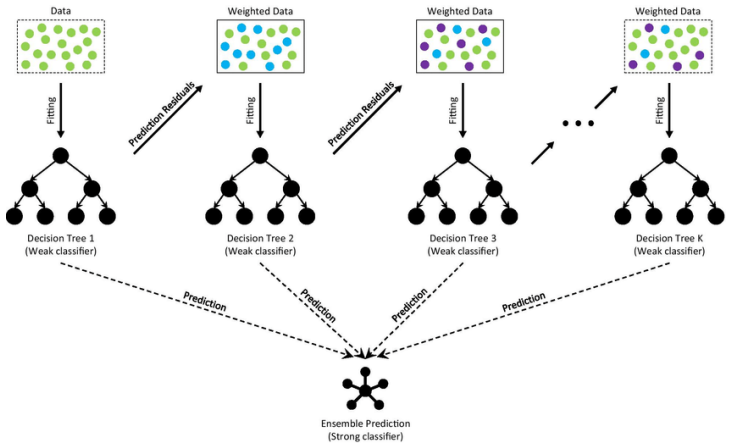


Figura 3.13: Visão simplificada do processo de classificação de uma *Boosted Decision Tree* contendo k árvores de decisão. Figura retirada de [53].

Formalmente, o treinamento da BDT ocorre de forma aditiva[12]. A saída $y_i^{(t)}$ do modelo após a adição da t-ésima árvore - considerando um conjunto de treinamento Q, árvores $f()$ e vetores de entrada x_i - é:

$$y_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = y_i^{(t-1)} + f_t(x_i).$$

Durante o treinamento, deseja-se que a adição de cada nova árvore $f()$ reduza o erro de classificação. Para que isso ocorra, árvores com estruturas adequadas devem ser adicionadas. No caso da BDT, a qualidade de uma árvore candidata com T nós folha é medida pela seguinte função objetivo:

$$objetivo = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in Q_j} g_i)^2}{\sum_{i\in Q_j} h_i + \lambda} + \gamma T, \tag{3.31}$$

em que λ e γ representam, respectivamente, a taxa de aprendizado e o coeficiente de regularização. Essencialmente, a taxa de aprendizado dita a velocidade com que o aprendizado irá ocorrer enquanto o coeficiente de regularização é utilizado para reduzir a complexidade do modelo e as chances de *overfitting*. Os termos g_i e h_i são definidos como:

$$g_i = \partial_{y_i^{(t-1)}} l(y_i, y_i^{(t-1)}),$$
$$h_i = \partial^2_{y_i^{(t-1)}} l(y_i, y_i^{(t-1)}),$$

em que $l()$ representa a função custo, definida *a priori* pelo usuário. Exemplos de função custo que podem ser utilizadas com a BDT incluem o Erro Quadrático Médio e a Entropia Cruzada.

Idealmente, a estrutura de cada nova árvore seria definida pela minimização da Eq. 3.31 - que tem um papel análogo às funções de impureza das árvores de decisão. Contudo, é proibitivo considerar todas as possíveis arquiteturas de árvores em cada nova adição. Logo, as árvores adicionadas na BDT são construídas a cada nível, em que cada nó é dividido em dois sub-nós. Da mesma forma como nas árvores de decisão tradicionais, o ajuste das partições na BDT é feito com a iteração sobre todos os atributos e valores por atributo, onde cada valor de corte produz subconjuntos Q_{left} e Q_{right}. A partição escolhida é aquela que maximiza a seguinte relação de ganho:

$$Ganho = \frac{1}{2}\left[\frac{(\sum_{i\in Q_{left}} g_i)^2}{\sum_{i\in Q_{left}} h_i + \lambda} + \frac{(\sum_{i\in Q_{right}} g_i)^2}{\sum_{i\in Q_{right}} h_i + \lambda} - \frac{(\sum_{i\in Q} g_i)^2}{\sum_{i\in Q} h_i + \lambda}\right] - \gamma. \tag{3.32}$$

O valor do *Ganho* para a melhor partição deve ser positivo, caso contrário o crescimento da árvore é parado.

[12]Esta decrição está baseada exclusivamente no modelo de BDT do XGBoost[54], que foi utilizado neste trabalho.

3.2.4 Máquinas de Vetores-Suporte

O algoritmo de Máquinas de Vetores-Suporte (do inglês, *Support Vector Machines*, SVM) está fundamentado no ajuste de fronteiras de decisão lineares para realizar classificações[36, 55]. Matematicamente, o treinamento do SVM consiste em ajustar um hiperplano separador das classes na forma:

$$g(x) = w^T * x + b = 0, \tag{3.33}$$

em que w é o vetor de pesos, x o vetor de entrada e b o intercepto.

Essencialmente, a classificação de uma observação é feita comparando a sua localização no espaço de atributos com relação ao hiperplano ajustado. No caso de um problema de classificação binária, a seguinte estratégia é adotada para rotular as observações:

- $y = 1$, se $g(x) \geq +1$
- $y = -1$, se $g(x) \leq -1$

Em geral, a formulação matemática do SVM re-escala as classes de modo que a parte superior do hiperplano rotule os dados como +1 e a parte inferior rotule como -1.

Os hiperplanos sempre são construídos com uma margem de separação, que representa a distância perpendicular até as observações mais próximas. Para o caso em que os dados são perfeitamente separáveis, infinitos hiperplanos podem ser ajustados no espaço de atributos durante o treinamento. Contudo, o hiperplano (ótimo) escolhido é sempre aquele que maximiza o tamanho ρ da margem de separação. As observações de treinamento utilizadas para determinar a margem de separação são denominadas vetores-suporte. Na prática, são os vetores-suporte que definem a localização da fronteira de decisão no espaço de atributos. A Figura 3.14 ilustra um exemplo de hiperplano ótimo ajustado em $\mathbb{R}^2$.[13]

A obtenção do hiperplano ótimo não é possível sem o ajuste adequado do vetor de pesos w. Formalmente, o tamanho da margem de separação está sujeita a[14]:

$$\rho = \frac{1}{||w||}, \tag{3.34}$$

indicando que o hiperplano ótimo, cuja margem é máxima, é obtido através da minimização da norma euclidiana de w. Tradicionalmente, esta minimização deve ser feita utilizando técnicas de otimização sobre alguma função custo. Considerando um cenário prático em que as observações de treinamento podem ter algum nível de sobreposição entre as classes, pode-se definir a expressão a ser minimizada como:

$$\frac{1}{2}||w|| + C\sum_{i=1}^{n} \zeta_i, \tag{3.35}$$

[13] Note que para $\mathbb{R}^2$, o hiperplano é representado por uma reta. Generalizando para um cenário com K atributos, o hiperplano ótimo seria ajustado sobre $\mathbb{R}^k$.

[14] Estamos considerando que o vetor de pesos está normalizado.

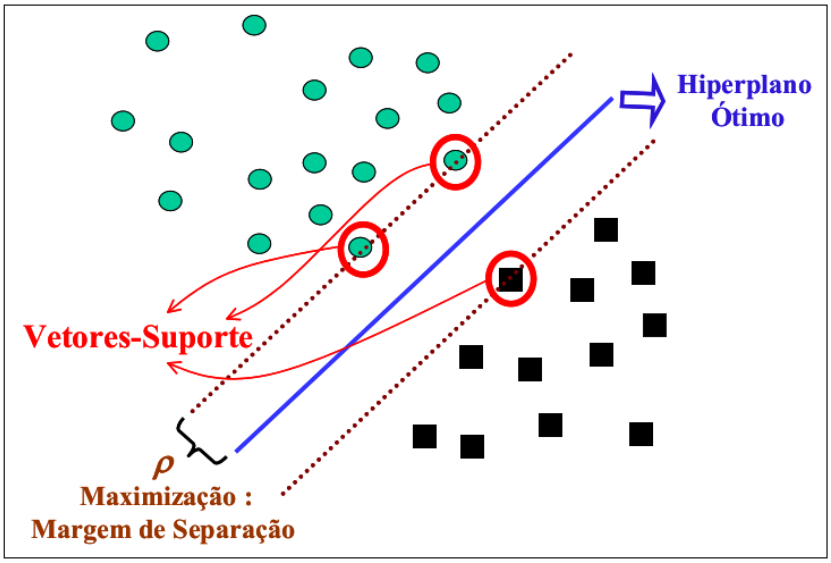


Figura 3.14: Hiperplano de máxima margem com vetores-suporte ajustado em $\mathbb{R}^2$. Figura retirada de [36].

$$\text{Sujeito a } \; y_i(w^T x_i + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, ..., n,$$

em que ζ é conhecido como variável de relaxação (do inglês, *slack variable*). Para observações classificadas corretamente, $\zeta = 0$, caso contrário ζ é igual a distância da observação até a sua margem correta. O termo $C\sum_{i=1}^{n} \zeta_i$ representa o quão aceitável será o erro de classificação do modelo, cuja intensidade é configurada pela constante de regularização C. Quanto maior C, maior será a margem do hiperplano ajustado e mais observações serão permitidas em seu interior. Apesar disto aumentar o erro junto aos dados de treinamento, acaba elevando a capacidade de generalização do modelo. Se $C = 0$, o hiperplano será ajustado sem admitir que observações fiquem dentro da margem, tornando-a mais estreita e reduzindo a capacidade de generalização do modelo. A Figura 3.15 representa o hiperplano ótimo sobre $\mathbb{R}^2$ considerando a atuação das variáveis de folga.

A Eq. 3.35 é minimizada através do método dos multiplicadores de Lagrange. Como resultado, a saída $f(x)$ do SVM tem a forma:

$$f(x) = sgn(\sum_{i=1}^{N} y_i \alpha_i x_i^T x + b), \tag{3.36}$$

em que os coeficientes α_i são os multiplicadores de Lagrange, cujo valor é diferente de zero apenas para os vetores-suporte - significando que, para N_{SV} vetores-suporte, a expressão 3.36 pode ser reescrita como:

$$f(x) = sgn(\sum_{i=1}^{N_{SV}} y_i \alpha_i x_i^T x + b). \tag{3.37}$$

A Eq. 3.37 é adequada para classificar dados linearmente separáveis apenas, não sendo ideal para

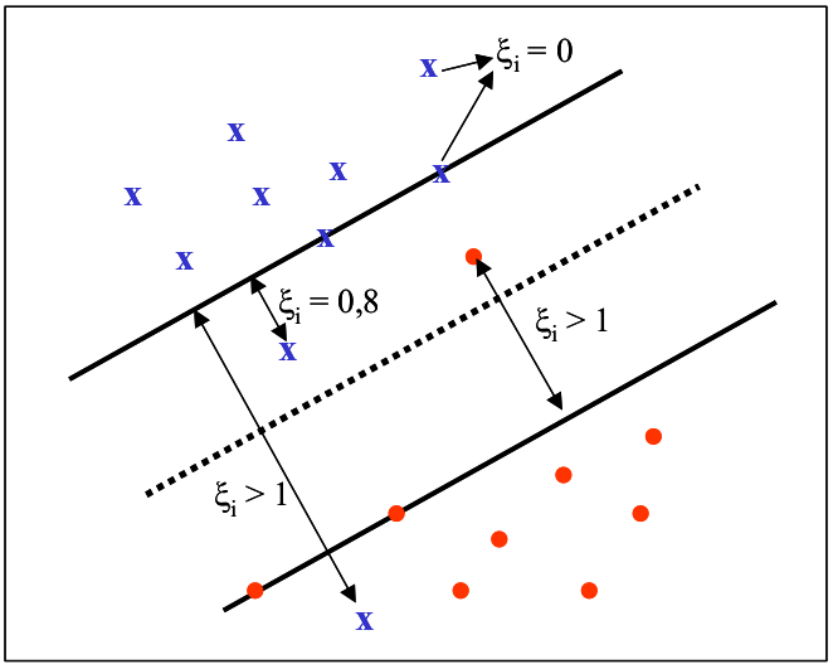


Figura 3.15: Hiperplano ajustado em $\mathbb{R}^2$ considerando o efeito das variáveis de folga. Figura retirada de [36].

realizar classificações não-lineares. Mesmo assim, é possível modificar esta formulação para atacar problemas não-lineares utilizando um artifício conhecido como Truque do Kernel.

A idéia fundamental do Truque do Kernel é a de transformar o problema de classificação não-linear em um problema linear. Para isso, deve ser possível realizar um mapeamento das observações do espaço de atributos inicial em um espaço de maior dimensionalidade utilizando uma função $\phi()$. Este mapeamento, que pode ser não-linear, é capaz de distorcer a posição dos dados, tornando-os linearmente separáveis. Com isso, pode-se então aplicar a formulação da 3.37 no novo espaço de alta dimensionalidade e realizar a tarefa de classificação. A Figura 3.16 ilustra estes conceitos.

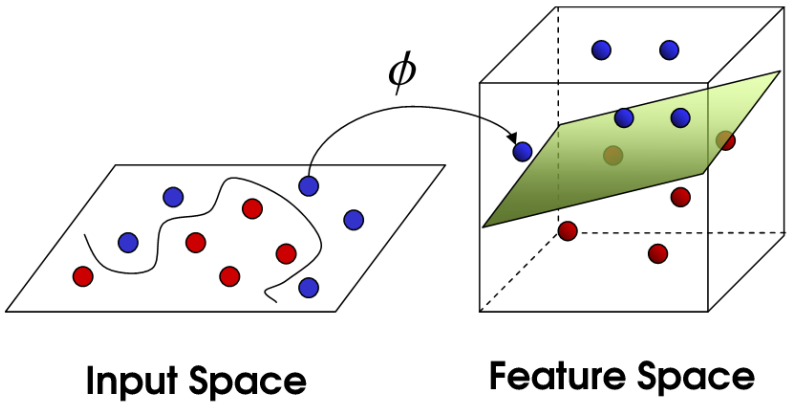


Figura 3.16: Atuação das funções kernel no mapeamendo dos dados para um espaço de maior dimensão. Figura retirada de [56].

Matematicamente, a equação do hiperplano no espaço de maior dimensionalidade é:

$$y = w^T * \phi(x) + b, \tag{3.38}$$

enquanto a saída do SVM treinado, obtido também via multiplicadores de Lagrange, teria a forma:

$$f(x) = sgn[\sum_{i=1}^{N_{SV}} y_i \alpha_i \phi^T(x_i)\phi(x) + b]. \tag{3.39}$$

A Eq. 3.39 indica que a saída do SVM depende do produto escalar $\phi^T(x_i)\phi(x)$ no espaço de maior dimensionalidade. Apesar desta formulação apontar a necessidade de se conhecer a forma explícita da função mapeadora $\phi()$, isto não é necessário. É suficiente apenas conhecer o resultado $K(x_i,x) = \phi^T(x_i)\phi(x)$ dos produtos escalares no espaço de maior dimensionalidade - sendo $K(x_i,x)$ conhecido como função kernel. Para a função kernel poder ser utilizada como uma generalização do produto escalar de ϕ, é necessário apenas que ela respeite a seguinte condição[15]:

$$\int\int K(x_i,x)g(x_i)g(x)dx_i dx > 0, \tag{3.40}$$

para toda função $g()$ tal que:

$$\int g^2(x_i)dx_i < \infty. \tag{3.41}$$

Uma vez que existam funções de kernel $K(x_i,x)$ que respeitem estas condições, a saída do SVM para classificações não-lineares pode ser escrita como:

$$f(x) = sgn[\sum_{i=1}^{N_{SV}} y_i \alpha_i K(x_i,x) + b], \tag{3.42}$$

em que $K(x_i,x)$ é definida *a priori* pelo usuário. Existem algumas funções kernel conhecidas na literatura e amplamente utilizadas nas aplicações de SVM:

- Função de base radial gaussiana:

$$K(u,v) = e^{(-\gamma||u-v||^2)}, \tag{3.43}$$

  em que o hiperparâmetro $\gamma > 0$.

- Função polinomial de grau d:

$$K(u,v) = (u^T v + 1)^d \tag{3.44}$$

[15] Esta restrição é conhecida como condição de Mercer.

- Função linear[16]:

$$K(u,v) = u^T v. \tag{3.45}$$

em que u e v são vetores quaisquer.

Finalmente, vale comentar que o modelo da SVM não realiza previsões probabilísticas naturalmente. Mesmo assim, é possível obter probabilidades utilizando uma técnica denominada *Platt scalling*[57], que consiste em treinar, via Regressão Logística, um classificador que tem como entrada as previsões do SVM treinado. Por utilizar uma função sigmóide, a Regressão Logística atua como um mapeador da saída do SVM para algum valor correspondente de probabilidade.

3.3 Técnicas de análise da importância de atributos

A performance dos algoritmos de ML em resolver tarefas está intrinsecamente relacionada com a escolha de atributos para as etapas de treinamento e teste. Quanto menos informativos forem os atributos, menor tende a ser a performance dos modelos preditivos. Neste sentido, torna-se relevante saber determinar o peso de cada variável em uma análise de ML. Além disto possibilitar um entendimento mais profundo do problema a ser atacado, também auxilia na exclusão de atributos pouco úteis ou possívelmente nocivos para o treinamento. Vantagens associadas à seleção de variáveis incluem [58]:

- Atributos altamente correlacionados podem indicar que uma mesma informação está sendo medida por eles. Remover um (ou mais) destes atributos pode simplificar a estrutura dos modelos treinados, sem impactos negativos na sua performance.

- Alguns modelos (como os lineares) podem ser afetados por atributos de variância nula - isto é, que possuem praticamente um único valor. Além de evitar este efeito, remover tais atributos gera um impacto praticamente nulo na qualidade dos treinamentos, uma vez que estas variáveis apresentam pouca ou nenhuma informação útil.

- Uma redução na performance dos modelos preditivos pode ocorrer em cenários com muitos atributos (alta dimensionalidade) para poucas observações de treinamento (baixa estatística). Este fenômeno é conhecido como Maldição da Dimensionalidade (do inglês, *Curse of Dimensionality*)[47]. Conforme a dimensionalidade aumenta, o volume do espaço de atributos se eleva ao ponto das observações se tornarem esparsas, reduzindo a capacidade dos modelos de sintetizar padrões ou correlações importantes para realizar as predições. Nestas situações, excluir atributos pouco informativos ou redundantes tende a reduzir os efeitos da Maldição da Dimensionalidade e a elevar a performance dos algoritmos preditivos.

[16] É válido comentar que a formulação obtida através do Truque do Kernel tem como caso especial os problemas de classificação lineares, afinal $K(u,v)$ é uma generalização do produto escalar.

- Trabalhar com menos atributos leva a um menor custo computacional nas etapas de treinamento e teste.

A seleção de atributos é primordialmente aplicada para excluir variáveis redundantes ou pouco informativas para o treinamento. Contudo, de forma mais ampla, estas mesmas técnicas também podem ser aproveitadas para medir a importância prática de cada variável em um estudo - informação que pode ser eventualmente empregada na tomada de decisões. Em ambos os casos, os métodos que avaliam o peso dos atributos podem ser divididos em três categorias principais: Métodos de Filtragem (*Filter Methods*), Métodos de Empacotamento (*Wrapper Methods*) e Métodos Embarcados (*Embedded Methods*).

3.3.1 Métodos de Filtragem

Métodos de Filtragem avaliam a importância de atributos sem utilizar algoritmos de ML, englobando normalmente técnicas tradicionais da inferência estatística. Em muitos casos, os Métodos de Filtragem são aplicados na etapa de pré-processamento dos dados para avaliar se cada atributo, individualmente[17], possui alguma relação significativa com a variável resposta. É comum que estas avaliações sejam feitas com testes de hipótese, em que o critério de qualidade para definir se uma variável possui alguma relação com outra é fornecida pelo cálculo do p-valor. A escolha de qual método estatístico usar deve considerar tanto os tipos de atributos analisados - se são contínuos, categóricos, ordinais - quanto as suas condições específicas de aplicação - como a suposição de normalidade ou não das distribuições.

As técnicas de filtragem escolhidas para as análises deste trabalho foram:

- **Correlação de Spearman** [59]: é uma técnica não-paramétrica[18] que avalia o grau de monotonicidade entre duas variáveis. Coeficientes próximos a +1 ou -1 indicam uma relação monotônica perfeita, positiva ou negativa, respectivamente. Por outro lado, coeficientes de Spearman próximos a 0 representam nenhuma correlação - ou nenhum grau de monotonicidade - entre duas variáveis X e Y quaisquer. Os coeficientes ρ de Spearman, considerando os *ranks*[19] X_r e Y_r, são calculados por:

$$\rho = \frac{\mathrm{COV}(X_r, Y_r)}{\mathrm{STD}(X_r)\mathrm{STD}(Y_r)}, \tag{3.46}$$

  em que COV() é a covariância e STD() é o desvio padrão. As hipóteses por trás da correlação de Spearman, avaliadas com o p-valor associado, são as seguintes:

  Hipótese Nula (H0): Não existe uma relação monotônica entre as duas variáveis.

  Hipótese Alternativa (H1): Existe uma relação monotônica entre as duas variáveis.

[17] Muitos métodos de filtragem são univariados.

[18] Isto é, que não assume condições prévias para o seu uso - como a normalidade das distribuições, por exemplo.

[19] *Rank* ou posto é o número que representa a posição de cada observação de uma variável. Para isso, as observações devem ser organizadas de forma crescente ou decrescente. Para observações com valores idênticos, toma-se o valor médio do *rank*.

- **Teste-U de Mann Whitney** [60]: indica o quanto uma variável é capaz de discriminar duas populações - no caso deste trabalho, as populações seriam equivalentes as classes (eventos com e sem ocorrência de *pile-up*, como se verá). Portanto, este é um teste univariacional não-paramétrico utilizado para verificar quais atributos podem dar um poder preditivo maior aos modelos de ML. Para duas variáveis X e Y com respectivos tamanhos n_1 e n_2, os valores U do teste são calculados como:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1,$$
$$U_2 = n_1 n_2 - U_1,$$

em que R_1 e R_2 representam as somas dos *ranks* de cada variável. O menor valor entre U_1 e U_2 é escolhido como o *score* do Teste-U. Quanto maior for o valor deste *score* para uma variável, maior tende a ser o seu poder discriminativo.

As hipóteses envolvidas no Teste-U são:

Hipótese Nula (H0): As distribuições das duas populações são iguais.

Hipótese Alternativa (H1): As distribuições das duas populações não são iguais.

3.3.2 Métodos de Empacotamento

Os Métodos de Empacotamento avaliam como a performance dos modelos de ML varia com relação aos atributos[58]. Neste sentido, é comum que estas técnicas envolvam eliminar ou alterar os atributos de entrada e monitorar o possível impacto destas ações nas saídas dos modelos preditivos.

Os algoritmos de empacotamento aplicados neste trabalho foram [61]:

- **Permutação de atributos**: Com um modelo de ML treinado, uma métrica de performance *baseline* é calculada com a classificação da amostra de testes. Em seguida, os valores de uma variável específica do mesmo conjunto de testes são permutados e a métrica de performance é re-calculada. Este processo é repetido independentemente para cada variável. Finalmente, a importância de cada atributo é avaliada como a queda percentual de performance após a sua permutação (*Performance drop*):

$$\text{Performance drop } (\%) = 100 * \left(\frac{\text{Métrica baseline} - \text{Métrica após permutação}}{\text{Métrica baseline}} \right). \quad (3.47)$$

- **Exclusão de atributos**: Inicialmente, o modelo de ML é treinado com todos os atributos disponíveis e uma métrica de performance *baseline* é calculada com a amostra de testes. Em seguida, um atributo é excluído dos conjuntos de treino e teste e o modelo de ML é re-treinado. Com este novo modelo, a métrica de performance é então re-calculada. Igual a Permutação de

Atributos, o mecanismo de Exclusão é repetido independentemente para cada atributo e as medidas de importância são calculadas por:

$$\text{Performance drop } (\%) = 100 * (\frac{\text{Métrica baseline} - \text{Métrica após exclusão}}{\text{Métrica baseline}}). \tag{3.48}$$

Note que, quanto maior for o impacto na performance com a permutação ou exclusão de uma certa variável, maior deverá ser a sua importância na tarefa de classificação. As possíveis métricas de performance para o cálculo do *Performance drop* serão definidas no Capítulo 4.

3.3.3 Métodos Embarcados

Métodos Embarcados utilizam modelos de ML que fazem a seleção de variáveis como parte de seu processo de aprendizado. A importância dada a cada atributo é, portanto, extraída diretamente dos modelos treinados. Duas tecnicas de ML foram escolhidas para estas análises:

- ***Random Forests***: como visto anteriormente, as *Random Forests* fazem uso de uma medida de impureza para determinar quais atributos participarão da construção dos nós das árvores de decisão. Atributos que reduzem o nível de impureza dos dados são utilizados com maior frequência e considerados de maior relevância. Formalmente, a estimativa de importância feita pela *Random Forest* sobre cada atributo é dada pelo cálculo do Decréscimo Médio da Impureza (do inglês, *Mean Decrease Impurity*, MDI) [62]. Atributos que possuem alguma correlação com a variável resposta tendem a apresentar valores maiores de MDI.

  Apesar de ser um método interessante e prático, alguns cuidados devem ser tomados ao se medir a relevância de atributos usando as *Random Forests*. Basicamente, este método pode inflar a importância de atributos contínuos ou categóricos de alta cardinalidade[20], que são empregados repetidas vezes na construção das árvores de decisão. Desta forma, para evitar conclusões incorretas, é relevante complementar a análise com outros métodos, como os de filtragem e de empacotamento.

- **Regressão Logística**: a Regressão Logística é fundamentada em um ajuste linear dos atributos de entrada, como descrito anteriormente. Neste modelo, a importância dos atributos (normalizados) pode ser estimada a partir da magnitude dos seus respectivos coeficientes, que são ajustados durante o treinamento. Atributos importantes (altamente correlacionados com a saída) devem ter um peso maior para o modelo, sendo associados a coeficientes com elevada magnitude. Inversamente, atributos pouco informativos devem possuir uma baixa influência sobre o modelo preditivo, com coeficientes próximos à 0.

[20] Atributos contendo muitas categorias.

Capítulo 4

Classificação de eventos com empilhamento

Neste capítulo, descreveremos a aplicação das técnicas de ML para reconhecer eventos com a ocorrência de *pile-up*. Porém, antes de esmiuçar os passos deste estudo, vamos discorrer brevemente sobre as métricas de análise utilizadas.

4.1 Métricas de análise

Como foi descrito anteriormente, a etapa de testes busca quantificar a qualidade das previsões feitas pelos modelos de ML considerando observações inéditas. Tradicionalmente, para problemas de classificação, a saída dos modelos preditivos é categórica - isto é, para uma classificação binária, a saída pode ser 0 ou 1, por exemplo. Com estas saídas é possível construir uma matriz de confusão, que fornece métricas úteis para avaliar o desempenho de classificadores[49]. Essencialmente, a matriz de confusão é uma tabela que cruza as classificações feitas pelos modelos preditivos com as respostas esperadas. A Figura 4.2 ilustra esta estrutura considerando um problema de classificação binária (com uma categoria "positiva"e outra "negativa"), em que os elementos são definidos por:

- Verdadeiro negativo (VN): número de observações negativas classificadas corretamente.
- Falso positivo (FP): número de observações negativas classificadas incorretamente como positivas.
- Falso negativo (FN): número de observações positivas classificadas incorretamente como negativas.
- Verdadeiro positivo (VP): número de observações positivas classificadas corretamente.

Existem algumas métricas de desempenho que podem ser extraídas a partir dos elementos da matriz de confusão, são elas:

		Classe predita	
		+	-
Classe verdadeira	+	VP	FN
	-	FP	VN

Figura 4.1: Estrutura da matriz de confusão para duas classes. Figura retirada de [49]

- Acurácia (*Accuracy*, ACC): fração de observações corretamente classificadas. É definida pelo traço da matriz de confusão dividido pelo número total de observações:

$$ACC = \frac{VP+VN}{VP+VN+FP+FN}. \tag{4.1}$$

- Taxa de Verdadeiro Positivo (*True Positive Rate*, TPR): fração de observações positivas classificadas corretamente. É definida como:

$$TPR = \frac{VP}{VP+FN}. \tag{4.2}$$

- Taxa de Falso Positivo (*False Positive Rate*, FPR): fração de observações negativas classificadas incorretamente como positivas. É dada por:

$$FPR = \frac{FP}{VN+FP}. \tag{4.3}$$

- Taxa de Verdadeiro Negativo (*True Negative Rate*, TNR): fração de observações negativas classificadas corretamente. É expressa por:

$$TNR = \frac{VN}{VN+FP}. \tag{4.4}$$

- Taxa de Falso Negativo (*False Negative Rate*, FNR): fração de observações positivas classificadas incorretamente como negativas. É escrita como:

$$TNR = \frac{FN}{VN+FN}. \tag{4.5}$$

Como comentado ao longo da seção 3.2, as técnicas de classificação podem fornecer a probabilidade P de uma observação pertencer a uma determinada classe. Pelo fato desta saída P ser contínua e restrita ao intervalo $[0,1]$, a classificação em si ocorre ao se determinar um parâmetro livre conhecido como Limiar de Decisão (do inglês,*Decision Threshold*), que representa um valor de corte sobre as probabilidades. Considerando um cenário com duas classes, uma observação é classificada como positiva se $P \geq$*Decision Threshold* - caso contrário, ela é rotulada como sendo negativa.

Cada valor de *Decision Threshold* produz uma matriz de confusão diferente, modificando também

as suas métricas de desempenho. Mesmo assim, ainda é possível avaliar a performance geral de classificadores binários recorrendo a um método conhecido como *ROC Curve* (do inglês, *Receiver Operating Characteristic Curve*) [47], que não depende da escolha de qualquer limiar sobre as probabilidades. A *ROC Curve* é construída plotando as Taxas de Verdadeiro Positivo e Falso Positivo para todos os valores de *Decision Threshold*. A Figura 4.2 exemplifica a forma de algumas *ROC Curves*, onde a linha diagonal representa o desempenho de um classificador aleatório[1]. Graficamente, quanto mais próxima do ponto $(0,1)$ estiver a *ROC Curve* de um classificador, melhor é sua performance geral. Os pontos $(1,1)$ e $(0,0)$ indicam, respectivamente, os valores extremos 0.0 e 1.0 do *Decision Threshold*.

Uma quantidade que sintetiza as informações das *ROC Curves* é a Área Sob a Curva (do inglês, *Area Under the Curve*, AUC), que varia no intervalo $[0,1]$. O valor da AUC indica a capacidade de um classificador discriminar observações com base em seus atributos, sendo uma boa métrica para comparar diferentes modelos. Quanto maior for este valor para um classificador, maior é a sua competência em realizar previsões. Para classificadores aleatórios - um dos piores cenários possíveis - o valor da AUC é igual a 0.5.

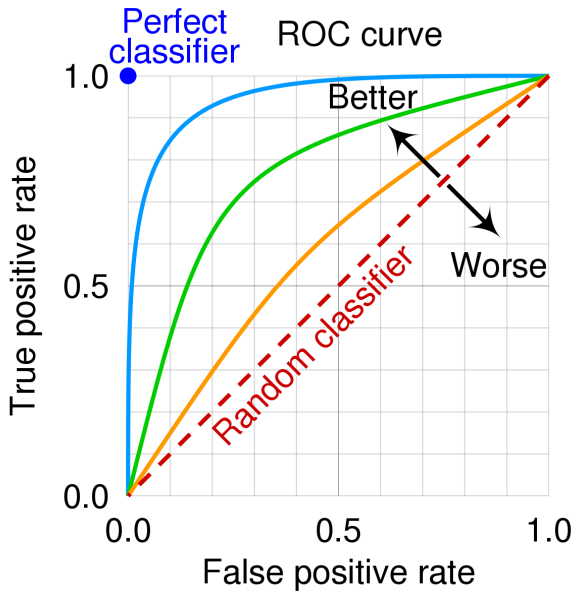


Figura 4.2: Ilustração de algumas *ROC* Curves. Os eixos x e y representam, respectivamente, as Taxas de Falso e Verdadeiro Positivo. A linha diagonal representa a performance de um classificador aleatório. Três classificadores estão exemplificados pelas curvas azul, laranja e verde. Figura retirada de [63]

.

Mesmo recorrendo as *ROC Curves* para fazer avaliações gerais dos modelos é fato que, na prática, a qualidade das classificações depende fortemente do valor escolhido para o *Decision Threshold*. Algumas análises adotam o seu valor padrão de 0.5. No entanto, o ideal é que a escolha deste limiar leve em consideração as demandas e particularidades de cada estudo. Em situações em que o erros de classificação possuem um custo alto, pode ser mais adequado trabalhar com valores mais elevados de *Decision Th-*

[1] Classificadores com *ROC Curves* que se encontram abaixo desta linha são considerados piores que os aleatórios.

reshold. Por outro lado, se os erros de classificação forem menos importantes do que identificar a classe positiva, pode ser mais interessante recorrer a *Decision Thresholds* menores. De todo modo, existem algumas figuras de mérito que podem ser empregadas para otimizar esta escolha, como o Índice de Youden (do inglês, *Youden's Index*)[64], que mede a diferença entre as Taxas de Verdadeiro e Falso Positivo:

$$\text{Youden's Index} = TPR - FPR. \tag{4.6}$$

Neste caso, o *Decision Threshold* escolhido deve ser aquele que maximiza a relação 4.6, por representar o valor de corte em que a quantidade de acertos do classificador é máxima com relação aos seus erros.

Finalmente, vale comentar que existem outras métricas para busca de *Decision Threshold* ótimo mas que não serão abordadas aqui, como: *G-Mean*, *F-Score* e o coeficiente de correlação de Matthews[64].

4.2 Conjunto de dados

Para as análises deste trabalho, foram utilizados dados simulados de colisões Pb-Pb à $\sqrt{s_{NN}} = 5.02$ TeV referentes às condições de operação do experimento ALICE no ano de 2015 (*Run 2* do LHC).

Dentro da colaboração ALICE, as simulações são feitas a partir de geradores de eventos, como o PYTHIA[65] e o HIJING[66], que reproduzem as condições iniciais das colisões. Em seguida, a propagação das partículas geradas e a sua interação com os detectores é simulada por bibliotecas como o *GEANT*[67, 68]. Por fim, os sinais (*hits*) nos detectores são combinados e reconstruídos da mesma forma como ocorre com os dados de colisões reais.

O grupo de eventos utilizado neste trabalho é categorizado como *Minimum-Bias* (MB), por conter colisões selecionadas a partir de sinais de *trigger* coincidentes nos detectores V0A e V0C e por englobar todas as classes de centralidade (0-100%). Os eventos também foram selecionados de modo que as posições dos vértices primários estivessem restritas ao intervalo $|z| < 10$ cm. Do total de eventos desta amostra, aproximadamente 40% possuem ocorrência de *pile-up*. Segundo documentações internas da colaboração ALICE, a ocorrência de *same-bunch pile-up* foi desprezível neste período, sendo que a maioria destes eventos contém *out-of-bunch pile-up* apenas.

Os atributos dos eventos empregados neste trabalho podem ser divididos em duas categorias: variáveis globais e variáveis de trajetória. As variáveis globais dizem respeito a medidas totais dos detectores do ALICE para cada evento, como o número total de trajetórias detectadas ou a amplitude de sinal. Por outro lado, as variáveis de trajetória, como sugere nome, correspondem a atributos das trajetórias reconstruídas dos eventos - como o momento, ângulos de espalhamento e etc. As trajetórias analisadas neste trabalho foram selecionadas considerando um intervalo de $|\eta| \leq 1.0$. A descrição de todas as variáveis estudadas e os seus tipos pode ser encontrada nas Tabelas 4.1 e 4.2.

Tabela 4.1: Descrição dos atributos globais dos eventos. A variável classe está destacada em negrito.

Atributos globais dos eventos		
Variáveis	Tipo de variável	Descrição
V0A	Contínua	Amplitude de sinal no detector *V0A*
V0C	Contínua	Amplitude de sinal no detector *V0C*
SPD Clusters	Contínua	N. de *clusters* no SPD
Global Tracks	Contínua	N. de trajetórias ITS+TPC que apontam para o vértice primário
Num. of TPC Tracks	Contínua	N. de trajetórias reconstruídas na TPC
IsEventPileup	Categórica	Indica se houve ocorrência de *pile-up* no evento

Tabela 4.2: Descrição dos atributos das trajetórias dos eventos.

Atributos das trajetórias		
Variáveis	Tipo de variável	Descrição
SPD_1	Categórica	Presença de sinal na 1ª camada do SPD
SPD_2	Categórica	Presença de sinal na 2ª camada do SPD
SSD_1	Categórica	Presença de sinal na 1ª camada do SSD
SSD_2	Categórica	Presença de sinal na 2ª camada do SSD
SDD_1	Categórica	Presença de sinal na 1ª camada do SDD
SDD_2	Categórica	Presença de sinal na 2ª camada do SDD
TPC	Categórica	Indica se houve propagação da trajetória na TPC
DCA_z	Contínua	Distância de máxima aproximação com relação a posição do vértice primário no eixo z
DCA_{xy}	Contínua	Distância de máxima aproximação com relação a posição do vértice primário no plano transversal
PT	Contínua	Momento transversal
Eta	Contínua	Pseudorapidez
Phi	Contínua	Ângulo azimutal

4.3 Aplicação das técnicas de ML

Nesta seção, serão descritos os passos envolvidos na classificação de eventos com as técnicas de ML. As nossas análises estão divididas entre aquelas que fazem uso de atributos globais dos eventos e aquelas que utilizam variáveis por trajetória. Nos dois casos, busca-se a discriminação de eventos com e sem *pile-up*. Para isso, a variável global *IsEventPileup* é empregada como classe, sendo utilizada para os treinamentos, validações e testes - inclusive, neste trabalho, os eventos com *pile-up* serão sempre denotados como a categoria "positiva". A primeira análise envolveu usar os atributos da Tabela 4.1 para o treinamento supervisionado de cinco modelos: *Random Forest*, *Boosted Decision Tree*, Redes Neurais

Artificiais (MLP), Regressão Logística e Máquina de Vetores-Suporte. A segunda análise consistiu em criar grafos dos eventos com as informações por trajetória (Tabela 4.2) e realizar a tarefa de classificação com uma *Graph Neural Network* (GNN). As etapas de cada investigação estão detalhadas na sequência.

Em todos os estudos, a estrutura inicial dos algoritmos foi ajustada seguindo uma busca em *grid*, que considera diferentes combinações de hiperparâmetros. Para os algoritmos baseados em árvores de decisão, o refinamento dos hiperparâmetros consistiu em variar o número e a profundidade das árvores. Na Regressão Logística, alguns métodos de otimização (como o de Newton e o Gradiente descendente) e de regularização (L1 e L2) foram avaliados considerando diferentes valores do coeficiente de regularização. A arquitetura final da Máquina de Vetores-Suporte também foi obtida por meio de testes sobre o coeficiente de regularização. Finalmente, as arquiteturas das redes MLP e GNN foram aprimoradas explorando combinações com variados tipos e quantidades de camadas, números de neurônios (dimensões), funções de ativação e valores de *learning rate*.

Durante a busca em *grid*, o desempenho dos modelos candidatos era avaliado por métricas como a acurácia e a AUC, enquanto a ocorrência de *overfitting* foi monitorada comparando os erros dos conjuntos de treinamento e validação. Os hiperparâmetros dos modelos finais podem ser vistos em mais detalhes no Apêndice B.

4.3.1 Análise com atributos globais dos eventos

Classificação de eventos Minimum-Bias

Inicialmente, foram criados grupos de treinamento e teste com 600×10^3 eventos MB em cada. A validação dos modelos foi feita com 20% dos eventos de treinamento. Em particular, para o ajuste do SVM, foram empregados apenas 200×10^3 eventos. Para evitar possíveis vieses no aprendizado dos algoritmos, a proporção de eventos com e sem *pile-up* foi mantida a mesma em todos os conjuntos de treinamento mencionados.

As previsões dos modelos treinados sobre os eventos de teste foram avaliadas pelas Taxas de Verdadeiro e Falso Positivo considerando todo o intervalo de *Decision Threshold*. A comparação entre modelos se deu pela montagem da *ROC Curve* e pelo cálculo dos valores de AUC correspondentes. Possíveis escolhas para o *Decision Threshold* foram estimadas com a maximização do Índice de Youden.

Finalmente, o desempenho dos classificadores foi comparado com a da seleção de eventos padrão utilizada pela colaboração ALICE, cuja eficiência é equivalente a $TPR \approx 0.63$ e $FPR \approx 0.0$ para eventos MB.

Análise técnica

Para entender a validade dos resultados obtidos pelos modelos de classificação, foram feitas algumas investigações adicionais. O intuito foi averiguar a confiabilidade de resultados obtidos via ML e compreender o funcionamento dos modelos em diferentes cenários de treinamento e teste. Estas análises incluem:

- **Estatística mínima de treinamento**: o desempenho dos modelos preditivos possui uma dependência com quantidade de observações que são empregadas no treinamento. Em geral, não há um número padrão de observações necessárias para o treinamento de cada técnica, por isto se tratar de uma questão dependente do problema. Logo, com o intuito de descobrir o número mínimo de eventos necessários para gerar bons classificadores, treinamos os modelos de ML utilizando diversos conjuntos, contendo de 10 até 200×10^3 eventos. O desempenho de cada algoritmo treinado foi avaliado pelo conjunto de testes com 600×10^3 eventos.

- **Validação com V0 e TPC**: comparar as saídas dos modelos de ML com o conhecimento de campo é importante para verificar se os algoritmos, de fato, aprenderam a discriminar as observações com base nos seus atributos. Por isso, observamos a disposição dos eventos classificados pelos modelos treinados na correlação de sinais dos detectores V0 e TPC, cujas distribuições para eventos com e sem *pile-up* são razoavelmente conhecidas.

- **Dependência com proporção de *pile-up***: conforme o experimento ALICE realiza a tomada de dados, a quantidade de agrupamentos do feixe tende a diminuir com o tempo, assim como a taxa de *bunch crossings* e o número de colisões por segundo. Isto significa que diferentes amostras de dados do ALICE podem conter proporções distintas de eventos com *pile-up*. Dito isto, verificamos se conjuntos de teste com diferentes porcentagens de eventos com *pile-up* poderiam impactar significativamente os resultados dos modelos preditivos. Para esta investigação, utilizamos conjuntos de teste com 300×10^3 eventos, com proporções de *pile-up* que variavam de 10 até 95%.

- **Tempo de teste**: pelo fato do ALICE lidar com grandes quantidades de dados dos eventos, os seus sistemas de *hardware* e *software* devem ser otimizados para entregar a maior eficiência computacional possível. Pensando neste fator, monitoramos o tempo médio que os modelos gastavam para classificar cada evento.

- **Dependência com a centralidade**: uma vez que a centralidade é uma característica importante dos eventos, foi investigada a variação de desempenho de modelos treinados e testados com eventos em diferentes intervalos de centralidade: (0-10)%, (10-30)%, (30-50)%, (50-70)% e (70-100)%. Os conjuntos de treino e teste empregados continham 100×10^3 eventos cada.

Análise de importância dos atributos

Para inferir a importância dos atributos globais dos eventos, foram empregadas as técnicas descritas na seção 3.3. A configuração dos modelos de ML e os grupos de treino e teste deste estudo foram os mesmos da classificação de eventos MB descrita anteriormente.

Por não envolver o uso do ML, a análise com os métodos de filtragem foi feita sobre uma única amostra com todos os eventos disponíveis. Os métodos de empacotamento foram aplicados com a Rede Neural Artificial e com a *Boosted Decision Tree* apenas, sendo que a métrica de desempenho escolhida para calcular o *Performance drop* foi a Acurácia. Finalmente, a *Random Forest* e a Regressão Logística foram escolhidas como os métodos embarcados deste estudo.

Após estimar a relevância de cada atributo, realizamos treinamentos e testes apenas com aqueles que se mostraram mais relevantes. O objetivo foi verificar se a tarefa de classificação poderia ser realizada com qualidade mesmo simplificando o problema para um cenário com menor dimensionalidade.

Inclusão do DCAz como atributo

Em geral, atributos pouco explicativos ou redundantes podem ser eliminados com o intuito de simplificar e, em alguns casos, de elevar o desempenho dos modelos preditivos, como foi discutido na seção 3.3. Em contrapartida, também é possível aumentar o poder preditivo dos algoritmos de ML fornecendo atributos com novas informações. Neste sentido, estudamos a possibilidade de incluir novas variáveis que pudessem aumentar a capacidade dos modelos de discriminar os eventos principais dos eventos com *pile-up*.

Devido ao tempo de deriva da TPC, sabe-se que a posição reconstruída z das trajetórias vindas de processos *out-of-bunch pile-up* é deslocada com relação a sua posição real. E, como discutido na seção 2.5, isto tem um efeito direto sobre o DCA_z destas trajetórias. Dito isto, tentamos codificar esta informação por trajetória como uma característica global dos eventos com o intuito de treinar novos classificadores. A estratégia escolhida envolveu compactar a informação de DCA_z de cada evento da seguinte forma:

$$\overline{DCA_z} = \frac{\sum_{i=1}^{N} DCA_{zi}^2}{N} \tag{4.7}$$

em que as N trajetórias são selecionadas como primárias pelo corte do ALICE[21]:

$$|DCA_{xy}| < (0.0182 + 0.0350\,\mathrm{GeV}/c\,p_T^{-1})\,\mathrm{cm}. \tag{4.8}$$

Finalmente, os classificadores foram treinados e testados incluindo $\overline{DCA_z}$ como atributo. Para possibilitar uma comparação justa, os hiperparâmetros, partições de dados e eventos empregados nesta etapa foram os mesmos da análise sobre eventos MB.

4.3.2 Análise com atributos por trajetória

Intuitivamente, as variáveis por trajetória carregam as informações mais rudimentares dos eventos e, portanto, devem ser mais informativas que as variáveis globais. Neste sentido, seria salutar aplicar as técnicas tradicionais de ML - BDT, SVM, *Random Forest* e etc - para classificar os eventos com base nos atributos de suas trajetórias. Entretanto, pelo fato dos eventos possuírem quantidades variadas de trajetórias e uma estrutura irregular, seria proibitivo realizar treinamentos e testes com estas técnicas. A solução foi tentar resolver esta tarefa de classificação com as *Graph Neural Networks*, que foram desenvolvidas para trabalhar com dados irregulares.

O desafio inicial desta abordagem foi estruturar cada evento do ALICE como um grafo. Consideramos os nós como sendo as trajetórias, que foram caracterizadas pelos atributos da tabela 4.2. Em adição aos atributos por trajetória, todos os nós também foram contemplados com as informações dos atributos globais (tabela 4.1) - o objetivo foi o de tentar atribuir a cada grafo um "peso global"que auxiliasse a GNN a discriminar os eventos. Para evitar que os grafos ficassem com um tamanho proibitivo, foram utilizadas apenas as trajetórias que passassem pela seleção de primários definida pela relação 4.8. Finalmente, padronizamos que cada nó comportasse quatro conexões, que eram estabelecidas apenas entre nós que possuíssem valores similares de $|DCA_z|$. Esta escolha foi feita com o intuito de separar grosseiramente as trajetórias vindas da colisão principal das possíveis colisões *pile-up*, o que poderia facilitar a discriminação de eventos pela GNN.

As análises realizadas com a GNN seguiram etapas parecidas com o estudo feito a partir dos atributos globais. Para verificar a estabilidade de desempenho, treinamos a GNN sobre amostras que continham de 10 até 1×10^4 eventos MB. Todos os conjuntos treinamento possuíam a mesma proporção de eventos por classe.

Foram empregados para validação e teste, respectivamente, 1×10^3 e 9×10^3 eventos. Os resultados obtidos com os testes foram comparados com a análise de atributos globais e com o corte padrão do ALICE. Por fim, também verificamos a dependência dos resultados com a centralidade dos eventos e estimamos o tempo que a GNN treinada gastava para classificar os eventos do grupo de teste. A arquitetura da GNN final, empregada em todos os estudos, pode ser consultada no Apêndice B.

Capítulo 5

Resultados

Neste capítulo, serão apresentados os resultados da aplicação de ML para discriminar eventos com e sem *pile-up*. O desempenho dos modelos de ML será discutido para as análises com atributos globais (Seção 5.1) e por trajetória (Seção 5.2). Nos dois casos, a performance dos algoritmos será comparada ao método de seleção padrão do ALICE. Finalmente, também serão mostradas as validações e os resultados técnicos feitos sobre os modelos de ML, além do estudo sobre a importância dos atributos globais.

5.1 Seleção de eventos com atributos globais

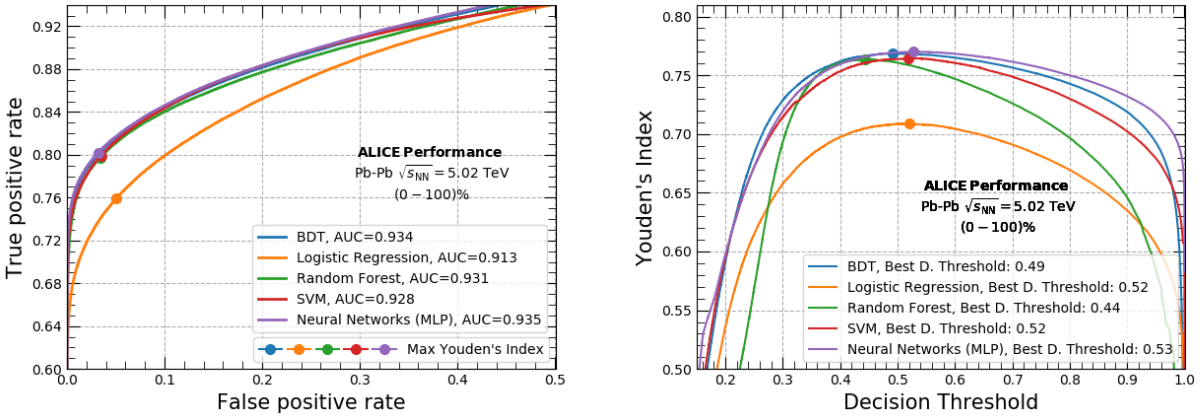
5.1.1 Análise de eventos *Minimum-Bias*

As ROC *Curves* ilustrando a variação das Taxas de Verdadeiro e Falso Positivo para os classificadores de ML podem ser vistas na Figura 5.1a. As curvas e os respectivos valores de AUC correspondem ao desempenho de cada algoritmo de ML em classificar os 600×10^3 eventos MB do grupo de testes. A comparação dos diferentes modelos mostra que todos possuem uma boa capacidade de discriminar eventos com e sem *pile-up*, com valores de AUC superiores à 0.9. A Rede Neural MLP e a BDT se mostraram os modelos de maior capacidade preditiva, com valores de AUC próximos a ≈ 0.935. Por outro lado, a Regressão Logística apresentou o menor desempenho geral, com uma AUC=0.913.

Para cada curva da Figura 5.1a, o ponto de *Decision Threshold* que maximiza o Índice de Youden (círculos) também é indicado. Com excessão da Regressão Logística, todas as técnicas possuem Taxas de Verdadeiro Positivo em torno de 80% e Taxas de Falso Positivo de 4% no valor máximo do Índice de Youden. A variação do Índice de Youden com relação ao *Decision Threshold*, para todas as técnicas de ML, está ilustrada na Figura 5.1b. Os valores de *Decision Threshold* que maximizam cada curva (círculos) também estão inclusos. De forma geral, os cortes ótimos de *Decision Threshold* variam em torno de 0.5, que é o valor padrão para tarefas de classificação.

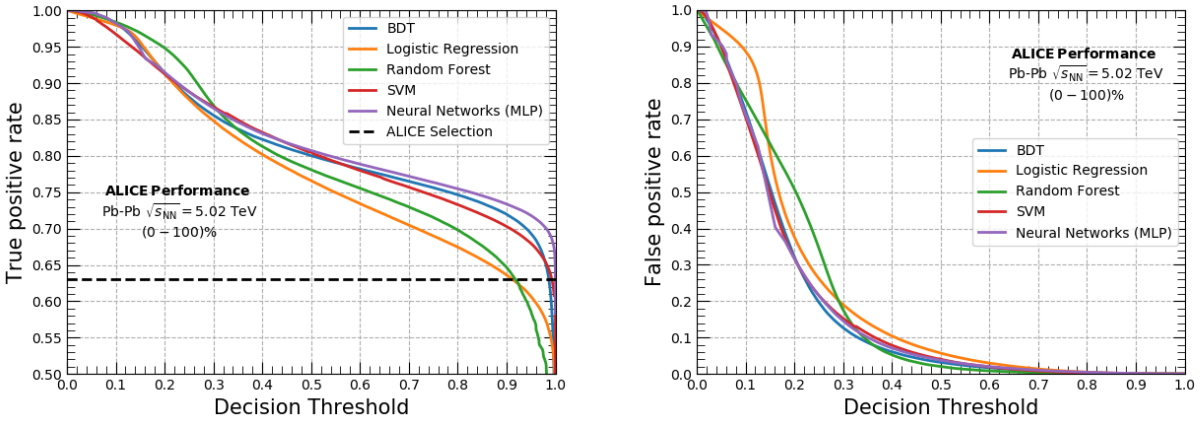
Comparamos o desempenho dos modelos preditivos com a seleção de eventos padrão aplicada pelo

ALICE, cujas Taxas de Verdadeiro e Falso Positivo são de, respectivamente, $\approx 63\%$ e $\approx 0.0\%$. Em geral, todos os modelos se mostraram melhores que a seleção do ALICE considerando valores mais elevados de *Decision Threshold*, como pode ser visto na Figura 5.2. Para um *Decision Threshold*$= 0.9$, a BDT e a Rede MLP apresentam Taxas de Verdadeiro Positivo em torno 73% (Fig. 5.2a) e Taxas de Falso Positivo que tendem a 0.0% (Fig. 5.2b), indicando um ganho de aproximadamente 10% sobre a seleção de eventos padrão.



(a) (b)

Figura 5.1: (a) ROC *Curves* e valores correspondentes de AUC para as diferentes técnicas de ML: Regressão Logística, BDT, Random Forest, SVM e Rede MLP. O pontos circulares indicam os valores de *Decision Threshold* que maximizam o Índice de Youden. (b) Variação do Índice de Youden de todas as técnicas de ML em função do *Decision Threshold*. Os valores ótimos de *Decision Threshold* estão inclusos.



(a) (b)

Figura 5.2: (a) Taxas de Verdadeiro Positivo e de (b) Falso Positivo em função do *Decision Threshold* para todas as técnicas de ML. A linha tracejada no gráfico da esquerda representa o desempenho da seleção de eventos do ALICE ($\approx$0.63). No gráfico da direita, o desempenho do corte padrão pode ser representado por uma linha constante em torno de $\approx$0.0.

Os resultados das Figuras 5.1 e 5.2 foram obtidos para treinamentos com 600×10^3 eventos. Para ga-

rantir que esta estatística era suficiente, estudamos a variação de desempenho de modelos treinados com diferentes quantidades de eventos. A Figura 5.3 apresenta os resultados obtidos, considerando um intervalo de 10 até 200×10^3 eventos e os valores correspondentes de AUC para cada classificador. Modelos mais simples ou menos flexíveis, como a Regressão Logística e a BDT, conseguem apresentar desempenhos razoáveis mesmo para treinamentos com menor estatística ($\leq 10^4$ eventos). Por outro lado, as Redes MLP exibem uma menor capacidade preditiva neste cenário - além dos treinamentos terem apresentado instabilidades e, em alguns casos, não convergirem. Este comportamento é um indicativo de que técnicas de ML mais flexíveis, como as Redes MLP, podem não ser adequadas para treinamentos com baixos volumes de dados - afinal, o risco de *overfitting* neste caso tende a ser elevado. Para quantidades maiores que $\geq 100 \times 10^3$ eventos, o desempenho de todos os classificadores se mostrou bom e suficientemente estável. Por isso, todos os treinamentos com atributos globais foram feitos com, pelo menos, esta quantidade eventos.

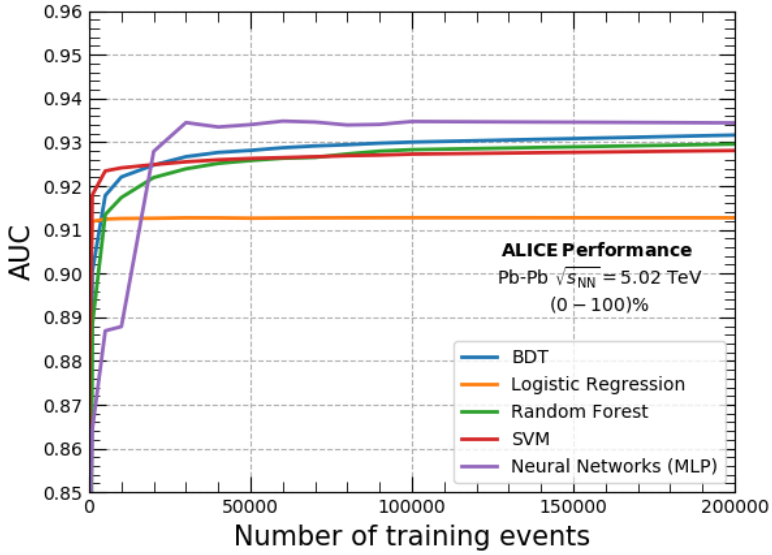


Figura 5.3: Desempenho (AUC) de cada técnica de ML em função do número de eventos no conjunto de treinamento. A quantidade de eventos com e sem *pile-up* foi a mesma em todas as amostras.

Também verificamos se o desempenho dos modelos preditivos apresentava qualquer dependência em relação a proporção de eventos com *pile-up* na amostra de testes. A Figura 5.4 mostra a variação das Taxas de Verdadeiro Positivo (*Decision Threshold*$= 0.5$) em função da porcentagem de eventos com *pile-up* para todos os modelos preditivos e para o método de seleção padrão. Apesar das pequenas flutuações, causadas pelas particularidades das amostras de testes, o desempenho de todos os métodos se mostra razoávelmente estável. Neste sentido, podemos ver que todas as abordagens são adequadas para uso nos diferentes *runs* do experimento ALICE, que podem naturalmente conter variadas proporções de *pile-up*.

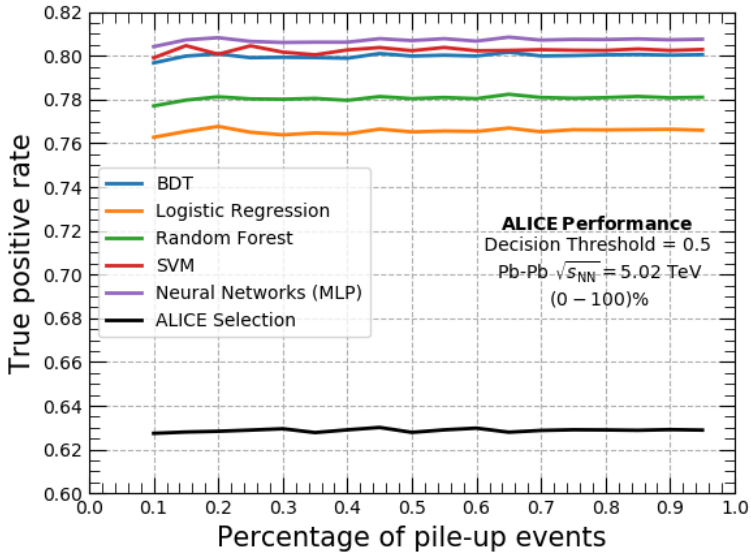


Figura 5.4: Taxa de Verdadeiro Positivo em função da porcentagem de eventos *pile-up* na amostra de testes. O desempenho dos classificadores foi calculado considerando um *Decision Threshold*= 0.5. A linha preta representa o desempenho do corte padrão do ALICE.

A partir de agora, apresentaremos alguns resultados focando especialmente nos modelos BDT e Rede MLP, que tiveram o maior desempenho geral dentre todos os algoritmos.

Para avaliar o aprendizado dos classificadores, analisamos a disposição dos eventos da amostra de testes na correlação dos detectores V0 e TPC. A rede MLP foi capaz de reconhecer satisfatoriamente as regiões com maiores concentrações de eventos com e sem *pile-up*, como mostra a Figura 5.5. Isto é reforçado pelo alto nível de similaridade das distribuições dos eventos classificados (gráficos inferiores) com as distribuições verdadeiras (gráficos superiores), mostrando também que os algoritmos de ML são capazes de aprender as principais diferenças físicas dos eventos.

5.1.2 Dependência com a centralidade dos eventos

Como comentado nas seções 2.2 e 4.3, a centralidade é um parâmetro importante para a caracterização dos eventos salvos no ALICE. A variação de desempenho (AUC) da BDT ao classificar eventos em diferentes intervalos do percentil de centralidade pode ser vista na Figura 5.6. Aqui, a medida de centralidade se refere apenas à colisão principal. A comparação dos valores de AUC mostra que eventos periféricos são consideravelmente mais fáceis de se classificar (AUC≈ 0.97) do que os eventos mais centrais (AUC≈ 0.89).

Colisões principais com alta centralidade produzem um número elevado de partículas e, portanto, sinais expressivos nos detectores mais rápidos e na TPC. Nestes eventos, é possível que as colisões *pile-up* sejam mais periféricas e acabem adicionando poucas trajetórias na contagem total da TPC. Como resultado, a discrepância no número de trajetórias salvas na TPC com os sinais dos detectores mais rápidos

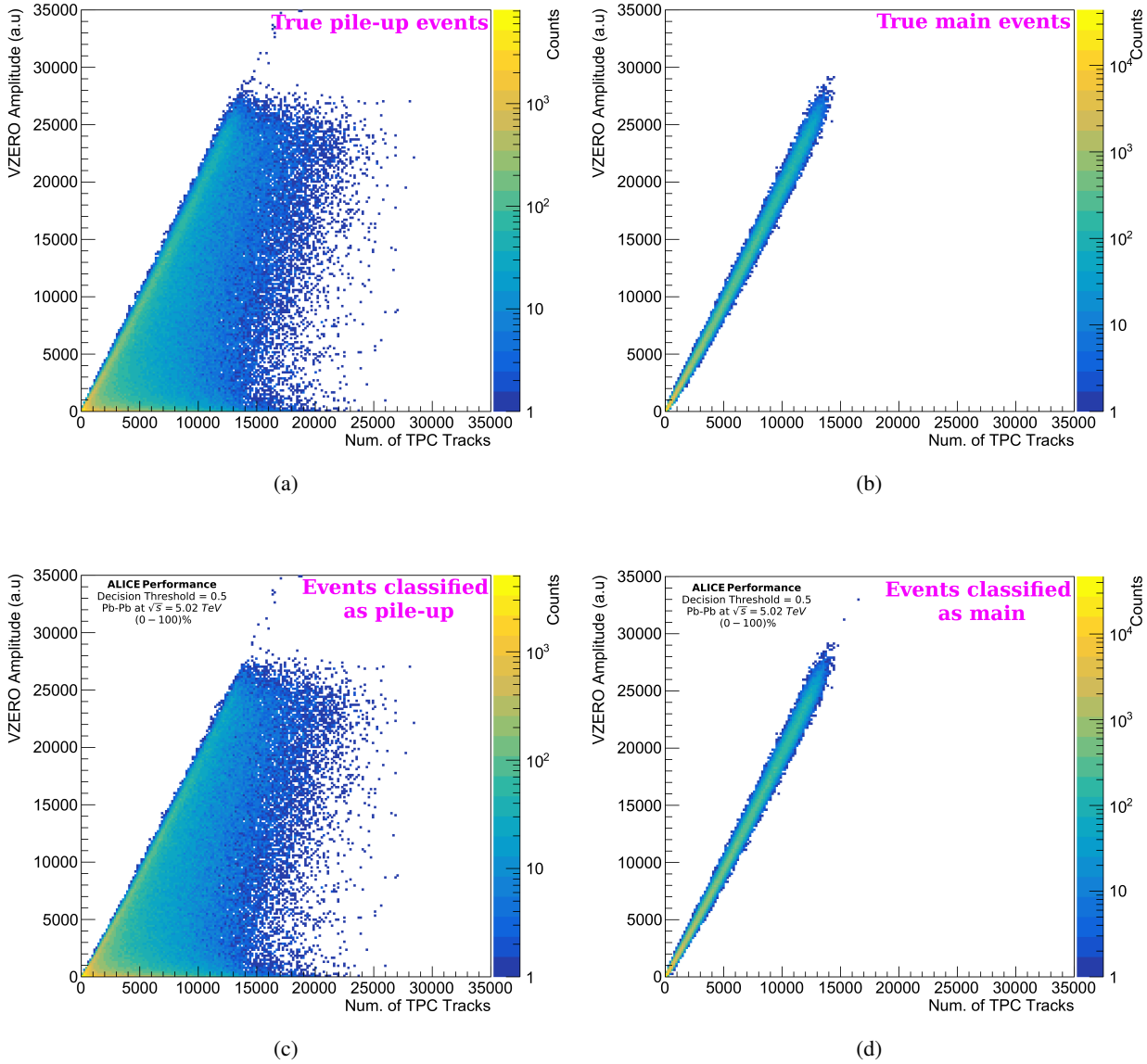


Figura 5.5: Correlação de informações dos detectores V0 e TPC para os eventos do conjunto de testes. Os gráficos superiores ((a) e (b)) mostram as distribuições verdadeiras, enquanto os gráficos inferiores ((c) e (d)) apresentam as distribuições dos eventos classificados pela rede MLP considerando um *Decision Threshold*= 0.5.

tende a ser muito pequena, dificultando o reconhecimento de *pile-up* nestes eventos. Inversamente, eventos com colisões principais periféricas e colisões *pile-up* centrais apresentam sinais muito discrepantes nos detectores mais rápidos e na TPC, o que facilita a identificação da ocorrência *pile-up*.

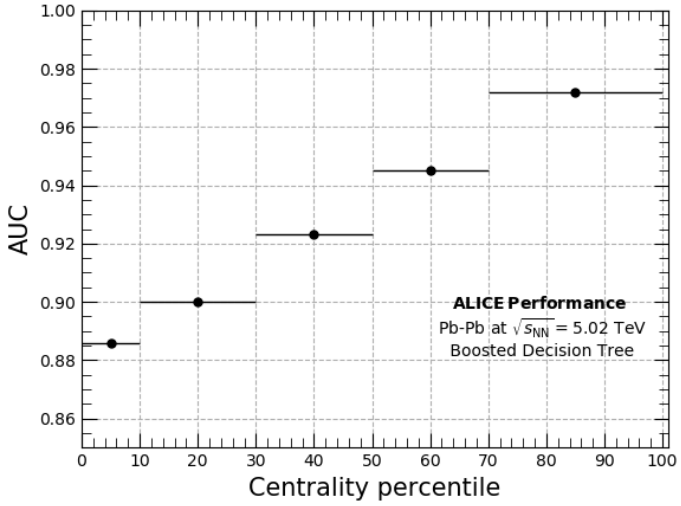


Figura 5.6: Valores de AUC para a *Boosted Decision Tree* treinada e testada com eventos em diferentes intervalos do percentil de centralidade: $(0-10)\%$, $(10-30)\%$, $(30-50)\%$, $(50-70)\%$ e $(70-100)\%$.

5.1.3 Análise de importância dos atributos

Estimamos a importância dos atributos globais aplicando os métodos de filtragem, de empacotamento e embarcados que foram descritos na Seção 3.3. Estabelecemos que os resultados dos testes estatísticos seriam considerados significativos apenas se os p-valores correspondentes fossem menores que 0.05.

Os coeficientes de correlação de Spearman (ρ) calculados para todos os atributos estão reportados na Tabela 5.1. Todos os atributos possuem um alto nível de correlação (monotonicidade) com os demais ($\rho > 0.7$ e p-valor< 0.05). Atributos relacionados a detectores rápidos - como o V0, SPD e ITS - apresentam as correlações mais altas ($\rho \approx 1.0$). Isto é intuitivo pelo fato destes detectores registrarem, em muitos casos, os sinais da colisão principal apenas. A alta correlação também pode indicar algum nível de redundância entre estes atributos. O número de trajetórias reconstruídas na TPC (*Num. of TPC Tracks*) foi o atributo menos correlacionado com os demais ($\rho \approx 0.82$), possivelmente pelo fato da TPC ter uma janela de aquisição maior e capturar informações tanto da colisão principal quanto das possíveis colisões *pile-up*.

Tabela 5.1: Coeficientes de correlação de Spearman (ρ) para os atributos globais (p-valor < 0.05). Os atributos menos correlacionados com os demais estão destacados em negrito.

Atributos	V0A	V0C	SPD Clusters	Global Tracks	Num. of TPC Tracks
V0A	1.0	-	-	-	-
V0C	0.9939	1.0	-	-	-
SPD Clusters	0.9954	0.9960	1.0	-	-
Global Tracks	**0.9907**	**0.9910**	**0.9940**	**1.0**	
Num. of TPC Tracks	**0.8233**	**0.8235**	**0.8245**	**0.8374**	**1.0**

Os resultados obtidos para cada atributo com o Teste-U de Mann-Whitney (*U-scores*) e os seus respectivos p-valores podem ser vistos na Tabela 5.2. Os atributos *Num. of TPC Tracks* and *Global Tracks* apresentaram os maiores *U-scores*, indicando que estas são as variáveis com o maior poder de discriminar significamente (p-valor< 0.05) os eventos com e sem *pile-up*. Por outro lado, os atributos relacionados apenas com detectores V0 e SPD, sozinhos, não possuem poder significativo (p-valor>0.05) para separar os dois tipos de evento.

Tabela 5.2: Resultados (*U-scores*) do teste-U de Mann-Whitney para todos os atributos com os respectivos p-valores. Os atributos com os maiores *scores* estão destacados em negrito.

Atributos	*U-score*	p-valor
V0A	1.35	$>$ **0.05**
V0C	1.37	$>$ **0.05**
SPD Clusters	1.43	$>$ **0.05**
Global Tracks	**10.3**	$<$ **0.05**
Num. of TPC Tracks	**333.7**	$<$ **0.05**

Através das técnicas de ML, foi possível avaliar a importância dos atributos pelos métodos de empacotamento. A Tabela 5.3 mostra a queda percentual de desempenho (acurácia) da BDT e da Rede MLP quando os atributos são permutados ou excluídos. As duas abordagens concordam que o atributo *Num. of TPC Tracks* é o mais importante para a classificação dos eventos, seguido por *Global Tracks*. Também é possível notar que, em geral, a exclusão ou permutação dos atributos relacionados a detectores rápidos tiveram pouco impacto no desempenho dos classificadores.

Tabela 5.3: Valores de *Performance drop* da Permutação e Exclusão de atributos com os modelos BDT e Rede MLP. Os atributos mais importantes estão destacados em negrito.

	Boosted Decision Trees		Rede MLP	
Atributos	Permutação Perf. Drop (%)	Exclusão Perf. Drop (%)	Permutação Perf. Drop (%)	Exclusão Perf. Drop (%)
V0A	14.30	1.149	21.13	0.95
V0C	14.30	1.149	21.13	0.95
SPD Clusters	13.71	0.6191	12.46	0.26
Global Tracks	**15.55**	**1.295**	**26.32**	**0.95**
TPC Tracks	**43.30**	**40.39**	**43.56**	**40.54**

Por fim, mensuramos a importância dos atributos com os métodos embarcados. A Tabela 5.4 mostra a influência de cada atributo global nos treinamentos da Regressão Logística e da *Random Forest*. Na Regressão Logística, a importância é medida pela magnitude dos coeficientes que foram ajustados no treinamento, ao passo que na *Random Forest* isto é medido pelo *Mean Decrease Impurity* (MDI). Segundo estas técnicas, os atributos *Num. of TPC Tracks* e *Global Tracks* se mostraram como os mais importantes para a tarefa de classificação. Este nível de importância foi razoavelmente menor para os atributos relacionados ao detector V0.

Tabela 5.4: Importância dos atributos globais segundo a *Random Forest* (MDI) e a Regressão Logística (coeficientes). Os melhores atributos estão destacados em negrito. Os valores das colunas foram rescalados de modo que a somar 1.

Features	*Random Forest*	Regressão Logística
	MDI	|Coeficientes|
V0A	0.0837	0.0897
V0C	0.0963	0.0776
SPD Clusters	0.158	0.0317
Global Tracks	**0.159**	**0.220**
Num. of TPC Tracks	**0.503**	**0.581**

Todos os métodos de importância dos atributos foram concordantes em apontar *Num. of TPC Tracks* como a variável mais relevante para a discriminação de eventos. Isto é compreensível sob o ponto de vista experimental, dado que o detector TPC é o mais lento do ALICE e registra todas as possíveis colisões adicionais dos eventos. Outro atributo que se mostrou importante é o *Global Tracks*, que representa o número de trajetórias reconstruídas com informações dos detectores ITS e TPC e que apontam para o vértice primário. Trajetórias deste tipo são, normalmente, associadas apenas a colisão principal. Por isto, é razoável considerar que a correlação de informações entre *Num. of TPC Tracks* e *Global Tracks* também pode ser útil para identificar eventos com *pile-up*.

De forma geral, os métodos aplicados também indicaram que os atributos dos detectores V0 e SPD possuem um baixo poder preditivo e/ou discriminatório. Com isto, poderíamos ingenuamente afirmar que estas variáveis não são relevantes para o treinamento dos modelos de ML; entretanto, sabemos que a correlação de detectores rápidos e lentos (como V0 e TPC) é importante para o reconhecimento dos eventos com *pile-up* - como discutido na seção 2.5. O ponto é que estes atributos devem carregar informações redundantes sobre os eventos (como apontou a Correlação de Spearman) e isto tem um impacto direto sobre a efetividade de alguns métodos, como os de Permutação e de Exclusão. Em geral, permutar ou excluir um atributo redundante tem um efeito pequeno sobre o desempenho dos modelos preditivos, uma vez que as informações excluídas ou alteradas ainda estão preservadas em parte dos atributos restantes. A mesma lógica é válida para os métodos embarcados, onde parte dos atributos redundantes tende a ser considerada menos importante para os treinamentos. Neste sentido, uma possível solução seria remover um ou mais atributos redundantes e aplicar novamente os métodos de importância.

5.1.4 Impacto da inclusão e retirada de atributos

O estudo da importância de atributos mostrou uma possível redundância entre as variáveis *SPD Clusters*, *V0A* e *V0C*. Como o detector V0 é especialmente utilizado pelo ALICE para a identificação de *pile-up*, buscamos avaliar se seria possível obter um modelo alternativo, com boa capacidade preditiva, mas que utilizasse apenas as informações do detector SPD. Em paralelo, também avaliamos qual seria o impacto de incluir o atributo $\overline{DCA_z}$ sobre os treinamentos, que foi calculado com a relação 4.7.

A Figura 5.7 mostra as ROC *Curves* e os valores de AUC da BDT. A curva (azul) referente ao treinamento com os atributos originais (*Baseline*) também está inclusa para a comparação. A retirada das informações do detector V0 ($V0A$ e $V0C$) teve um impacto muito pequeno sobre a qualidade do aprendizado. Nesta configuração, o desempenho da BDT ($AUC = 0.929$) ainda é muito próximo ao que se tinha originalmente ($AUC = 0.934$). Isto mostra que os atributos *SPD Clusters* e *Global Tracks* são substitutos adequados de $V0A$ e $V0C$ para as predições. Finalmente, é possível observar que a inclusão do $\overline{DCA_z}$ como atributo global elevou consideravelmente o desempenho da BDT (AUC$= 0.950$). Este resultado é interessante por confirmar que o DCA_z das trajetórias, de fato, pode ser empregado para reconhecer eventos com ocorrência de *pile-up*.

5.2 Seleção de eventos com atributos das trajetórias

5.2.1 Número mínimo de eventos para o treinamento

Após estruturar os eventos do ALICE na forma de grafos, estudamos qual seria a estatística necessária para um treinamento adequado e estável da GNN. A Figura 5.8 mostra a variação de desempenho (AUC)

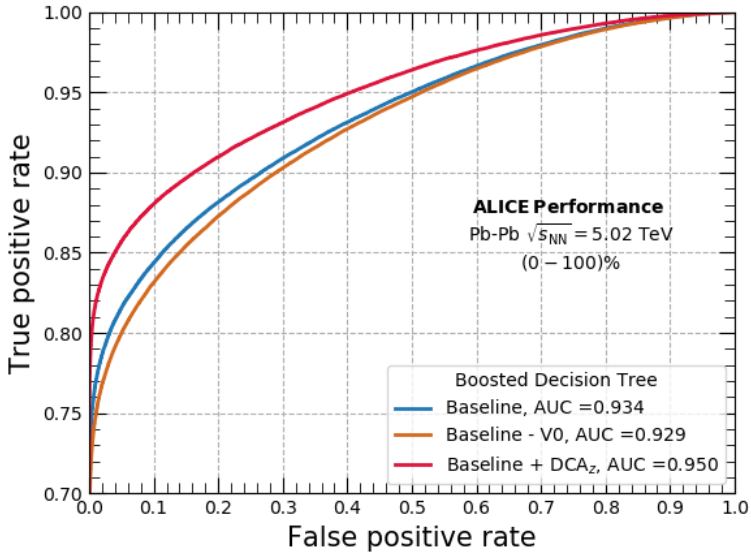


Figura 5.7: ROC *Curves* com os respectivos valores de AUC para a BDT treinada com diferentes configurações de atributos. O modelo com os atributos originais (*Baseline*) está ilustrado em azul, sem a informação de V0 (*Baseline-V0*) em laranja e com a informação de DCAz (*Baseline*+DCA$_z$) em vermelho.

da GNN em função da quantidade de eventos de treinamento, que variou de 10 até 10^4. No cenário de menor estatística ($<$ 1000 eventos), a GNN apresenta um desempenho de classificação razoável, com uma AUC de até 0.89. Este desempenho se estabilizou em torno de AUC$=$ 0.90 para maiores quantidades de eventos. Assim sendo, os resultados que serão apresentados na sequência foram obtidos para treinamentos com mais de 1000 eventos.

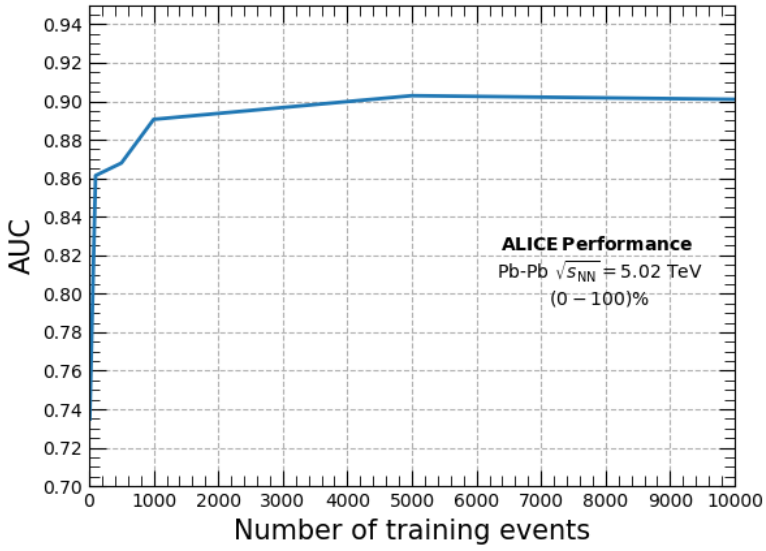
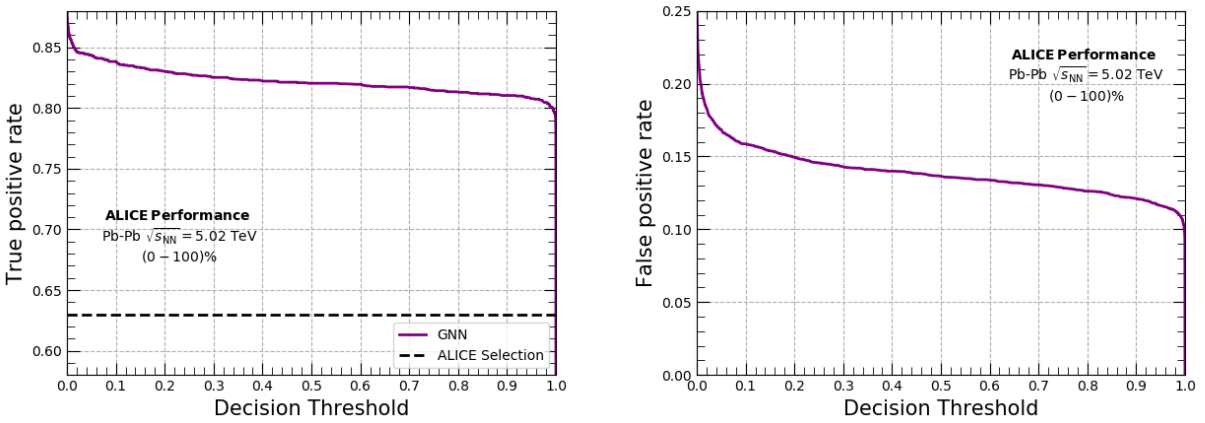


Figura 5.8: Desempenho (AUC) da GNN em função do número de eventos no conjunto de treinamento. A quantidade de eventos com e sem *pile-up* foi a mesma em todas as amostras.

5.2.2 Desempenho da GNN

Comparamos o desempenho da GNN treinada com a seleção de eventos do ALICE, como mostra a Figura 5.9. Em geral, para todo o intervalo de *Decision Threshold*, a GNN apresenta uma Taxa de Verdadeiro Positivo maior que 80%, superando o corte padrão do ALICE (Figura 5.9a). Entretanto, a Taxa de Falso Positivo é consideravelmente elevada, sendo maior que 10% para a maior parte do *Decision Threshold*. Em suma, apesar da GNN reconhecer mais eventos com *pile-up*, uma fração significativa dos eventos principais acaba sendo perdida - o que torna a GNN um método de mitigação pior que a seleção de eventos padrão, que preserva praticamente todos os eventos principais.



(a) (b)

Figura 5.9: (a) Taxas de Verdadeiro Positivo e de (b) Falso Positivo em função do *Decision Threshold* para a *Graph Neural Network*. A linha tracejada no gráfico da esquerda representa o desempenho da seleção de eventos do ALICE (≈0.63). No gráfico da direita, o corte padrão pode ser representado por uma linha constante com uma Taxa de Falso Positivo de ≈0.0.

5.2.3 Dependência com a centralidade

Avaliamos a dependência do desempenho da GNN (AUC) com a centralidade dos eventos, como mostra a Figura 5.10. A GNN apresenta uma capacidade preditiva consideravelmente menor ao classificar eventos centrais (AUC≈ 0.84) quando comparada a classificação de eventos mais periféricos (AUC≈ 0.92). Este efeito, que também foi visto na análise com atributos globais, indica que a diferença de centralidade das colisões principais também dificulta a discriminação dos eventos, mesmo utilizando os atributos por trajetória. Uma possível causa para este efeito pode estar relacionado com o fato da GNN tomar a média do *node embedding* para fazer as classificações. Eventos com colisões principais centrais sustentam muitas trajetórias com atributos parecidos - o que é amplificado pelo mecanismo de *Message Passing*. Nestes eventos, as características das trajetórias de colisões *pile-up* mais periféricas têm um peso muito menor no cálculo da média sobre *node embedding* - dificultando a identificação da

ocorrência de *pile-up* pela GNN. O mesmo argumento pode explicar o cenário inverso, onde a GNN tem maior facilidade para reconhecer eventos com colisões *pile-up* centrais e colisões principais periféricas.

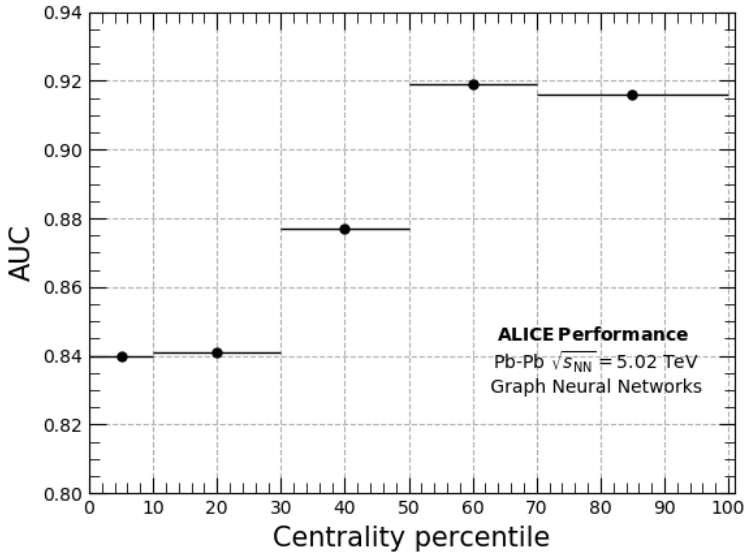


Figura 5.10: Valores de AUC para a *Graph Neural Network* treinada e testada com diferentes intervalos do percentil de centralidade: $(0-10)\%$, $(10-30)\%$, $(30-50)\%$, $(50-70)\%$ e $(70-100)\%$.

5.3 Comparação de modelos

5.3.1 Desempenho de classificação

Finalmente, comparamos a capacidade preditiva de modelos treinados com atributos globais e com atributos por trajetória. A Figura 5.11 apresenta as ROC *Curves* e os valores de AUC correspondentes a cada algoritmo. A curva da BDT (azul) se refere ao treinamento sobre atributos globais, enquanto a curva da GNN (roxa) indica o treinamento com atributos por trajetória. Os pontos (círculos) representando o valor máximo do Índice de Youden também estão inclusos. A comparação entre os modelos mostra que o treinamento com os atributos globais produz o melhor resultado preditivo (AUC=0.934). Mesmo assim, a GNN ainda apresenta um poder preditivo razoável (AUC=0.901), com uma Taxa de Verdadeiro Positivo de $\approx 78\%$ e uma Taxa de Falso Positivo de $\approx 8\%$ no valor de corte do Índice de Youden.

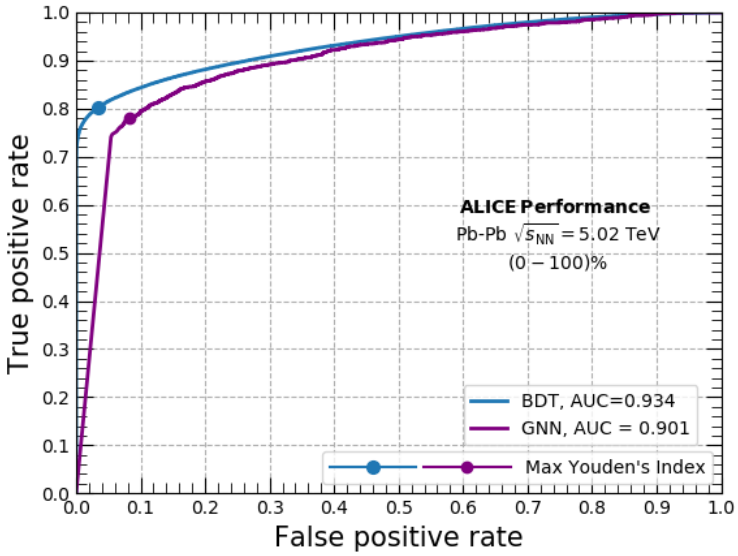


Figura 5.11: ROC *Curves* e valores correspondentes de AUC para a BDT (azul), treinada com os atributos globais, e para a GNN (roxo), treinada com os atributos por trajetória. O pontos circulares indicam os valores de *Decision Threshold* que maximizam o Índice de Youden.

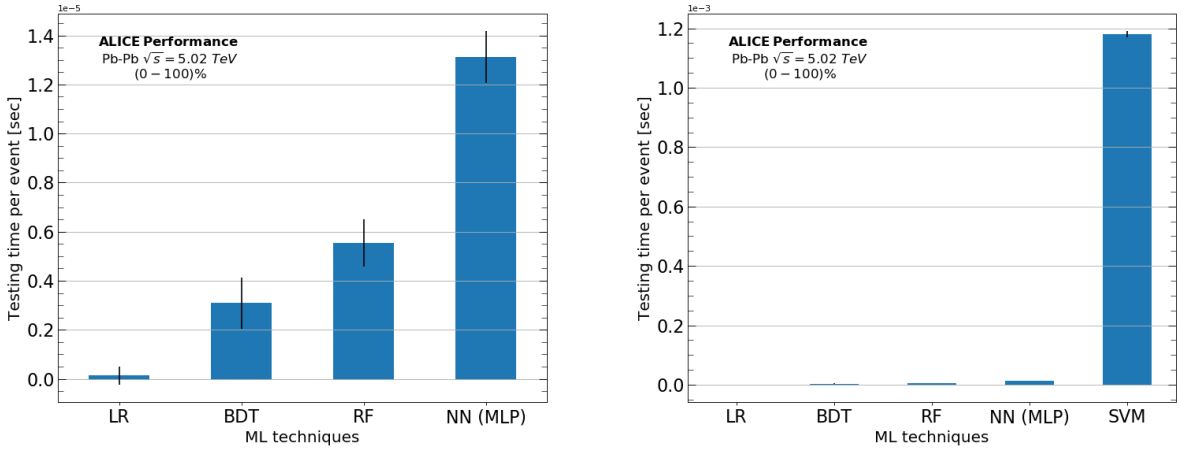
5.3.2 Tempo de Teste

Durante as etapas de testes, avaliamos o custo computacional das técnicas de ML ao classificar eventos MB. A Figura 5.12 apresenta o tempo médio (barras azuis) que cada algoritmo de ML levou (em segundos) para classificar cada evento dos conjuntos de teste. Estão inclusas tanto as técnicas aplicadas sobre os atributos globais (gráficos superiores) quanto a comparação do SVM com a GNN (gráfico inferior). O tempo médio de classificação foi avaliado testando cada algoritmo 10 vezes. A comparação dos modelos mostra que a Regressão Logística (LR) tem o menor custo computacional, levando $\approx 1.4 \times 10^{-7}$ segundos para classificar cada evento (Figura 5.12a). Os modelos BDT, *Random Forest* (RF) e Rede MLP também apresentaram custos computacionais razoáveis. Dentre os modelos treinados sobre atributos globais, o SVM foi o que levou mais tempo para classificar cada evento ($\approx 1.2 \times 10^{-3}$ segundos), sendo a técnica mais custosa (Figura 5.12b). Contudo, este tempo ainda é razoávelmente inferior quando comparado ao tempo médio que a GNN leva para classificar cada evento ($\approx 5.2 \times 10^{-2}$ segundos), como pode ser visto na Figura 5.12c.

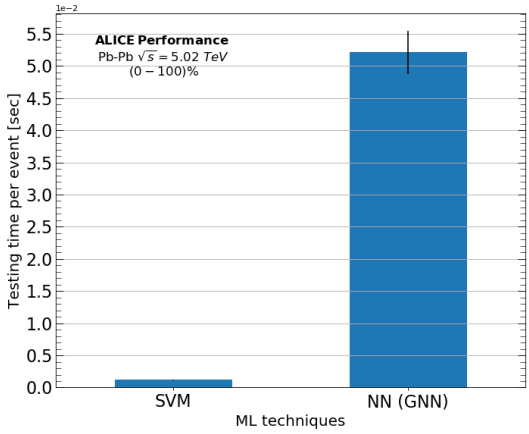
Os resultados da Figura 5.12 são relativamente intuitivos. O processo de classificação dos modelos mais velozes ocorre através de operações computacionalmente simples, como multiplicações matriciais (Regressão Logística e Rede MLP) ou avaliações condicionais (*Random Forest* e BDT). A saída do modelo SVM é baseado no cálculo de distância das novas observações com os vetores-suporte, que são criados com as observações de treinamento. Quanto mais vetores-suporte tiver o modelo final do SVM, maior será o custo computacional do seu uso. No caso da GNN, o tempo maior para as classificações é

justificado pelo fato dos eventos serem estruturados como um grafos, onde cada nó comportava quatro conexões e era representado por um vetor com 17 atributos. Esta estrutura é computacionalmente custosa, especialmente quando lembramos que eventos Pb-Pb possuem, em geral, uma multiplicidade de partículas elevada.

Por fim, uma ressalva que deve ser feita sobre este estudo é que todos os testes foram processados por uma CPU (do inglês, *Central Processing Unit*), que é a unidade de processamento comum dos computadores. Porém, sabe-se que o custo computacional de técnicas baseadas em Redes Neurais Artificiais é consideravelmente menor quando se faz uso de uma GPU (do inglês, *Graphical Processing Unit*), que é capaz de paralelizar muitas operações e otimizar o tempo de treino e teste destes modelos [69].



(a) (b)



(c)

Figura 5.12: Tempo médio gasto (em segundos) para classificar cada evento (barras azuis). O gráfico (a) mostra tempo de teste da Rede MLP (NN(MLP)), *Random Forest* (RF), BDT e Regressão Logística (LR), enquanto o gráfico (b) inclui o modelo SVM na comparação. O gráfico (c) mostra a comparação direta do SVM com a *Graph Neural Network* (NN(GNN)). As barras pretas representam um desvio-padrão.

Para futuras consultas, tanto os resultados da Figura 5.12 como os desempenhos obtidos com os

modelos de ML (treinados e testados sobre eventos MB) foram resumidos na Tabela 5.5.

Tabela 5.5: Desempenho obtido pelos classificadores e o seus respectivos custos computacionais. Os valores de TPR e FPR foram calculados considerando o limiar cujo Índice de Youden é máximo.

Modelos de classificação	TPR (%)	FPR (%)	AUC (%)	Tempo médio de teste por evento (10^{-6} s)
Random Forest	79.74	3.401	93.1	5.54
Regressão Logística	75.90	5.011	91.3	0.138
BDT	80.21	3.355	93.4	3.10
SVM	79.93	3.494	92.8	1180.0
Rede MLP	80.21	3.213	93.5	13.1
GNN	78.06	8.194	90.1	52100.0

Capítulo 6

Conclusões e perspectivas

Neste trabalho, estudamos a ocorrência de *pile-up* em colisões Pb-Pb à 5.02 TeV nas condições do experimento ALICE do LHC. Descrevemos como este processo ocorre durante a aquisição de dados e o método empregado pelo ALICE para mitigar os seus efeitos nas análises físicas. Neste contexto, propomos uma nova abordagem para reduzir os efeitos do *pile-up* utilizando técnicas de ML. Este estudo foi feito com eventos simulados e diferentes técnicas foram aplicadas: *Random Forest*, *Boosted Decision Tree*, Regressão Logística, Máquina de Vetores-Suporte, Rede Neural MLP e *Graph Neural Network*. As análises foram conduzidas tanto com atributos globais dos eventos (amplitudes de sinal, número de *clusters* e contagens de trajetórias) como com atributos das trajetórias (momento transversal, distância de máxima aproximação, pseudorapidez e etc).

Tivemos sucesso em identificar eventos com ocorrência de *pile-up* com os métodos de ML. Os resultados obtidos com os atributos globais mostraram um ganho de desempenho considerável sobre o método de mitigação tradicional do ALICE. Para verificar a confiabilidade dos resultados, analisamos a dependência do desempenho destes algoritmos com a quantidade de eventos no treino, com a proporção de eventos *pile-up* no teste e com o percentil de centralidade. Também comparamos a disposição dos eventos classificados na correlação dos detectores V0 e TPC com a sua distribuição verdadeira. Todas as análises mostraram que os modelos de ML são confiáveis e robustos para classificar eventos.

Algumas técnicas para descobrir a importância dos atributos globais foram estudadas, sendo elas: Correlação de Spearman, Teste-U, Permutação e Exclusão de atributos, *Random Forests* e Regressão Logística. Os atributos globais mais importantes, segundo estes métodos, foram *Num. of TPC Tracks* e *Global Tracks*. Devido às possíveis redundâncias, os atributos relacionados aos detectores mais rápidos, V0 e SPD, tiveram a sua importância subestimada. Esta redundância foi comprovada ao observarmos que a retirada das informações do detector V0 teve um impacto desprezível sobre o desempenho dos modelos; mostrando que a mitigação de *pile-up* pode ser feita de forma razoável sem o uso deste detector. Ademais, também observamos que a inclusão das informações do DCA_z das trajetórias levou a um

aumento substancial da capacidade preditiva dos algoritmos.

Para a análise com atributos das trajetórias, observamos que a GNN consegue reconhecer uma grande fração dos eventos com *pile-up*. Entretanto, uma parte significativa dos eventos principais acaba sendo identificada errôneamente; implicando que a GNN, neste ponto, não supera o desempenho da seleção de eventos do ALICE, que mantém $\approx 100\%$ dos eventos principais. Por outro lado, uma pequena quantidade de eventos de treinamento se mostrou necessária para a obtenção de uma GNN com desempenho estável. Por fim, tanto os modelos treinados sobre atributos globais quanto a GNN mostraram uma dependência do seu desempenho com a centralidade dos eventos. Constatamos que quanto mais periféricas forem as colisões principais dos eventos, mais fácil é o reconhecimento da ocorrência de *pile-up*.

Na seção 5.3, comparamos a capacidade preditiva da BDT, treinada com atributos globais, com a da GNN, treinada com atributos por trajetória. A GNN não apresentou um desempenho melhor que a BDT para discriminar eventos. Neste sentido, uma investigação mais profunda acerca de novas arquiteturas da GNN ou métodos para a construção dos grafos deve ser feita; uma vez que os atributos por trajetória são, intuitivamente, mais informativos que os atributos globais e devem ser capazes de levar a modelos com desempenho superior. Por último, observamos que a BDT apresentou um baixo custo computacional em comparação com a maioria dos modelos, sendo uma técnica interessante para aplicações sobre grandes quantidades de dados. Diferente da GNN, que foi a técnica mais custosa para classificar cada evento na fase de testes.

Finalmente, tendo também em vista o calendário de operação do ALICE para os próximos anos, esperamos que este trabalho contribua para o desenvolvimento de novas soluções baseadas em Inteligência Artificial para estudos e análises da área de altas energias. Com relação às futuras perspectivas, seria salutar validar os resultados deste trabalho aplicando os algoritmos treinados para classificar eventos reais do ALICE. Atualmente, esta implementação ainda é desafiadora pelo fato do sistema de computação (*Grid*) da colaboração não ter suporte para algumas das bibliotecas empregadas neste trabalho. Entretanto, um esforço tem sido feito para integrar muitos destes pacotes (baseados na linguagem Python) aos sistemas do ALICE e, por esta razão, temos confiança de que os nossos resultados poderão ser aproveitados nos próximos anos.

Bibliografia

1. Thomson, M. *Modern particle physics* (Cambridge University Press, 2013).
2. Evans, L. & Bryant, P. LHC Machine. *Journal of Instrumentation* **3,** S08001–S08001 (2008).
3. CMS Collaboration. The CMS experiment at the CERN LHC. *Journal of Instrumentation* **3,** S08004–S08004 (2008).
4. ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation* **3,** S08003–S08003 (2008).
5. LHCb Collaboration. The LHCb Detector at the LHC. *Journal of Instrumentation* **3,** S08005–S08005 (2008).
6. Battistoni, G. *et al.* The Application of the Monte Carlo Code FLUKA in Radiation Protection Studies for the Large Hadron Collider. *Progress in Nuclear Science and Technology* **2,** 358–364 (2011).
7. ALICE Collaboration. The ALICE experiment at the CERN LHC. *Journal of Instrumentation* **3,** S08002–S08002 (2008).
8. Adams, J. *et al.* Experimental and theoretical challenges in the search for the quark–gluon plasma: The STAR Collaboration's critical assessment of the evidence from RHIC collisions. *Nuclear Physics A* **757,** 102–183 (2005).
9. Adcox, K. *et al.* Formation of dense partonic matter in relativistic nucleus–nucleus collisions at RHIC: Experimental evaluation by the PHENIX Collaboration. *Nuclear Physics A* **757,** 184–283 (2005).
10. Back, B. *et al.* The PHOBOS perspective on discoveries at RHIC. *Nuclear Physics A* **757,** 28–101 (2005).
11. Arsene, I. *et al.* Quark–gluon plasma and color glass condensate at RHIC? The perspective from the BRAHMS experiment. *Nuclear Physics A* **757,** 1–27 (2005).

12. Noferini, F. *et al.* ALICE results from Run-1 and Run-2 and perspectives for Run-3 and Run-4. *Journal of Physics: Conference Series* **1014,** 012010 (2018).

13. CMS Collaboration. Pileup mitigation at CMS in 13 TeV data. *Journal of Instrumentation* **15,** P09018–P09018 (2020).

14. Martınez, J. A., Cerri, O., Spiropulu, M., Vlimant, J. R. & Pierini, M. Pileup mitigation at the Large Hadron Collider with graph neural networks. *The European Physical Journal Plus* **134,** 333 (2019).

15. Duarte, J. & Vlimant, J. Graph Neural Networks for Particle Tracking and Reconstruction. *Artificial Intelligence For High Energy Physics,* 387 (2022).

16. Terashi, K. *et al.* Event classification with quantum machine learning in high-energy physics. *Computing and Software for Big Science* **5,** 1–11 (2021).

17. Abbas, M., Khan, A., Qureshi, A. S. & Khan, M. W. Extracting signals of Higgs boson from background noise using deep neural networks. arXiv: 2010.08201 [hep-ph] (2020).

18. Romero, A., Whiteson, D., Fenton, M., Collado, J. & Baldi, P. Safety of quark/gluon jet classification. *arXiv preprint arXiv:2103.09103* (2021).

19. Feickert, M. & Nachman, B. *A Living Review of Machine Learning for Particle Physics* 2021. arXiv: 2102.02770 [hep-ph].

20. Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* **3,** 210–229 (1959).

21. ALICE Collaboration. Performance of the ALICE experiment at the CERN LHC. *International Journal of Modern Physics A* **29,** 1430044 (2014).

22. Chinellato, D. D. *Estudo de estranheza em colisões próton-próton no LHC.* Tese de doutorado (Universidade Estadual de Campinas, 2012).

23. Chiochia, V.; Accelerators and Particle Detectors. *Lecture notes from University of Zurich* (2010).

24. ALICE Collaboration. New ALICE detectors for Run 3 and 4 at the CERN LHC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **958,** 162116 (2020).

25. ALICE Collaboration. ALICE: Physics Performance Report, Volume I. *Journal of Physics G: Nuclear and Particle Physics* **30,** 1517–1763 (2004).

26. Bondila, M. *et al.* ALICE T0 detector. *IEEE transactions on nuclear science* **52,** 1705–1711 (2005).

27. Bhasin, A. *et al. Implementation of the alice trigger system* em *2007 15th IEEE-NPSS Real-Time Conference* (2007), 1–8.

28. Géron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (O'Reilly Media, Inc., 2019).

29. Pan, S. J. & Yang, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* **22,** 1345–1359 (2010).

30. Boccato, L., Attux, R.; *Notas de aula do curso "Tópicos em Sistemas Inteligentes II - Aprendizado de Máquina"*. 2019. https://www.dca.fee.unicamp.br/~lboccato/ia006_2s2019.html.

31. Lloyd, S. Least squares quantization in PCM. *IEEE transactions on information theory* **28,** 129–137 (1982).

32. Goodfellow, I. *et al.* Generative adversarial nets. *Advances in neural information processing systems* **27** (2014).

33. Deshpande, M. *A Guide to Improving Deep Learning's Performance.* https://pythonmachinelearning.pro/a-guide-to-improving-deep-learnings-performance/. 2017.

34. Al-Behadili, H. N. K., Ku-Mahamud, K. R. & Sagban, R. *Rule pruning techniques in the ant-miner classification algorithm and its variants: A review* em *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)* (2018), 78–84.

35. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning* (MIT press, 2016).

36. Von Zuben, F. J. *Notas de aula do curso "Redes Neurais"*. 2021. https://www.dca.fee.unicamp.br/~vonzuben/courses/ia353_1s21.html.

37. Peixoto, F. *A Simple overview of Multilayer Perceptron(MLP).* 2020. https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/.

38. Sowmya Yellapragada. *Understanding Loss Functions in Computer Vision.* 2020. https://medium.com/ml-cheat-sheet/winning-at-loss-functions-2-important-loss-functions-in-computer-vision-b2b9d293e15a.

39. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

40. Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning* **14,** 1–159 (2020).

41. Leskovec, J. Stanford cs224w: Machine Learning with Graphs. *Traditional Methods for Machine Learning in Graphs* (2019).

42. Wu, Z. *et al.* A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* **32,** 4–24 (2021).

43. Morris, C. *et al. Weisfeiler and leman go neural: Higher-order graph neural networks* em *Proceedings of the AAAI conference on artificial intelligence* **33** (2019), 4602–4609.

44. Hamilton, W., Ying, Z. & Leskovec, J. Inductive representation learning on large graphs. *Advances in neural information processing systems* **30** (2017).

45. Veličković, P. *et al.* Graph attention networks. *stat* **1050,** 20 (2017).

46. Hosmer Jr, D. W., Lemeshow, S. & Sturdivant, R. X. *Applied logistic regression* (John Wiley & Sons, 2013).

47. Hastie, T., Tibshirani, R. & Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction* (Springer, 2009).

48. Scikit-Learn developers. *Decision Trees.* 2007. https://scikit-learn.org/stable/modules/tree.html.

49. Faceli, K., Lorena, A. C., Gama, J. & Carvalho, A. C. P. L. F. *Inteligência artificial: uma abordagem de aprendizado de máquina* (LTC, 2011).

50. Breiman, L. Random forests. *Machine learning* **45,** 5–32 (2001).

51. Scikit-Learn developers. *Ensemble methods.* 2007. https://scikit-learn.org/stable/modules/ensemble.html.

52. Kashyap, K. *Machine Learning - Decision Trees and Random Forest Classifiers.* 2019. https://medium.com/analytics-vidhya/machine-learning-decision-trees-and-random-forest-classifiers-81422887a544.

53. Deng, H., Zhou, Y., Wang, L. & Zhang, C. Ensemble learning for the early prediction of neonatal jaundice with genetic features. *BMC medical informatics and decision making* **21,** 1–11 (2021).

54. Chen, T. & Guestrin, C. *Xgboost: A scalable tree boosting system* em *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), 785–794.

55. Zhang, T. An introduction to support vector machines and other kernel-based learning methods. *Ai Magazine* **22,** 103–103 (2001).

56. Moradi, S., Tapak, L. & Afshar, S. Identification of Novel Noninvasive Diagnostics Biomarkers in the Parkinson's Diseases and Improving the Disease Classification Using Support Vector Machine. *BioMed Research International* **2022** (2022).

57. Arat, M. M.; *Can you interpret probabilistically the output of a Support Vector Machine?.* 2019. https://mmuratarat.github.io/2019-10-12/probabilistic-output-of-svm.

58. Kuhn, M. & Johnson, K. *Applied predictive modeling* (Springer, 2013).

59. Spearman, C. The Proof and Measurement of Association between Two Things. *The American Journal of Psychology* **15,** 72–101 (1904).

60. Mann, H. B. & Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics,* 50–60 (1947).

61. Parr, T. and Turgutlu, K. and Csiszar, C. and Howard, J. *Beware Default Random Forest Importances* 2018. https://explained.ai/rf-importance/index.html.

62. Louppe, G., Wehenkel, L., Sutera, A. & Geurts, P. *Understanding variable importances in forests of randomized trees* em *Advances in Neural Information Processing Systems* (ed. Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K. Q.) **26** (Curran Associates, Inc., 2013). https://proceedings.neurips.cc/paper/2013/file/e3796ae838835da0b6f6ea37bcf8bcb7-Paper.pdf.

63. Thoma, M. *Receiver Operating Characteristic (ROC) curve with False Positive Rate and True Positive Rate.* 2018. https://commons.wikimedia.org/wiki/File:Roc-draft-xkcd-style.svg.

64. Berrar, D. Performance measures for binary classification (2019).

65. Sjöstrand, T. *et al.* An introduction to PYTHIA 8.2. *Computer physics communications* **191,** 159–177 (2015).

66. Wang, X. & Gyulassy, M. HIJING: A Monte Carlo model for multiple jet production in pp, pA, and AA collisions. *Physical Review D* **44,** 3501 (1991).

67. Brun, R., McPherson, A., Zanarini, P., Maire, M. & Bruyant, F. *GEANT 3: user's guide Geant 3.10, Geant 3.11* rel. técn. (CERN, 1987).

68. Agostinelli, S. *et al.* GEANT4—a simulation toolkit. *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506,** 250–303 (2003).

69. Buber, E. & Banu, D. *Performance analysis and CPU vs GPU comparison for deep learning* em *2018 6th International Conference on Control Engineering & Information Technology (CEIT)* (2018), 1–6.

70. Scikit-Learn developers. *Clustering.* 2007. https://scikit-learn.org/stable/modules/clustering.html.

71. Chen, Y. & Lai, Y. Sparse dynamical Boltzmann machine for reconstructing complex networks with binary dynamics. *Physical Review E* **97** (2018).

72. Ester, M. *et al. A density-based algorithm for discovering clusters in large spatial databases with noise* em. **96** (1996), 226–231.

73. Roman, V.; *Unsupervised Machine Learning: Clustering Analysis.* 2019. https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e.

74. Mckinney, W. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python High Performance Science Computer* (2011).

75. Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* **12,** 2825–2830 (2011).

76. Ketkar, N. & Santana, E. *Deep learning with Python* (Springer, 2017).

77. Abadi, M. *et al. TensorFlow: a system for Large-Scale machine learning* em *12th USENIX symposium on operating systems design and implementation (OSDI 16)* (2016), 265–283.

78. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17,** 261–272 (2020).

79. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems* **32** (2019).

80. Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

Apêndice A

Estudos com técnicas de Aprendizado Não-Supervisionado

Por completeza, realizamos algumas análises sobre os atributos globais dos eventos utilizando algoritmos de agrupamento, que estão fundamentados no escopo do aprendizado não-supervisionado. O objetivo do agrupamento é particionar o conjunto de dados em grupos (*clusters*). A princípio, observações associadas a um mesmo *cluster* devem apresentar atributos semelhantes, enquanto observações de *clusters* distintos devem ter características dissimilares.

Nossas análises foram feitas com dois algoritmos clássicos de agrupamento: k-means e DBSCAN. Porém, antes de apresentar tais técnicas e os resultados obtidos, introduziremos algumas novas métricas para avaliações de desempenho

A.1 Métricas de desempenho

A avaliação de desempenho dos algoritmos de agrupamento não é simples como no caso dos algoritmos de classificação. Neste caso, as métricas de desempenho não devem levar em conta os valores absolutos associados aos *clusters* sintetizados, mas sim se o processo de agrupamento separou as observações de forma similar às suas categorias verdadeiras - ou, no caso de não se ter acesso a estas informações, tais métricas também devem ser capazes de medir características gerais do agrupamento, como a compacidade dos *clusters* e o grau de separação entre eles.

Abaixo, discutiremos sobre algumas figuras de mérito que usam as informações das categorias verdadeiras[1] das observações para avaliar o desempenho dos algoritmos de agrupamento [70].

A.1.1 Índice ajustado de Rand

Mede a similaridade entre os *clusters* verdadeiros e os sintetizados pelos algoritmos. Consiste, grosso modo, na contagem dos pares de observações que são associadas ao mesmo ou a diferentes *clusters* (tanto nos *clusters* verdadeiros quanto nos *clusters* preditos).

Considerando o conjunto de amostras de dados associadas aos L rótulos verdadeiros $P^* = \{C_1^*, C_2^*, ..., C_L^*\}$

[1] Para análises sem estas informações, existem outras métricas que podem ser facilmente encontradas na literatura, como o Coeficiente de Silhueta e o Índice de Calinski-Harabasz.

e o grupo de K *clusters* sintetizados $P = \{C_1, C_2, ..., C_K\}$, a similaridade entre eles pode ser calculada pelo Índice ajustado de Rand (*ARI*) como:

$$ARI(P^*, P) = \frac{\sum_{ij}\binom{N_{ij}}{2} - [\sum_i \binom{N_i}{2} \sum_j \binom{N_j}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_i \binom{N_i}{2} + \sum_j \binom{N_j}{2}] - [\sum_i \binom{N_i}{2} \sum_j \binom{N_j}{2}]/\binom{N}{2}}, \tag{A.1}$$

em que N é o número total de observações e N_{ij} é o número de observações da classe $C_j^* \in P^*$ associadas ao cluster C_i na partição P. N_i é o número de observações no cluster C_i da partição P, e N_j é o número de observações na classe C_j^*. O *ARI* apresenta valores próximos a 0.0 para classificações aleatórias ou exatamente 1.0 quando os *clusters* correspondem as classes verdadeiras.

A.1.2 Informação Mútua Normalizada

O *score* de Informação Mútua Normalizada (do inglês, *Normalized Mutual Information*, *NMI*) mede o grau de pureza que se atinge com o agrupamento. Agrupamentos bem sucedidos geram *clusters* com elevados graus de pureza, isto é, a maioria, senão todas, as observações ali contidas pertencem à mesma classe - nestes casos, o valor de *NMI* é em torno de 1.0. Por outro lado, processos de agrupamento mal sucedidos produzem *clusters* contendo observações de várias classes e, portanto, pouco puros - para estas situações, o *NMI* fica em torno de 0.0. De forma similar ao cálculo do Ganho em Árvores de Decisão, desejamos que o processo de agrupamento reduza a impureza de uma amostra de observações com relação as classes.

Matemáticamente, o *NMI* pode ser escrito como:

$$NMI(P^*, P) = \frac{2 \times [H(P^*) - H(P^*|P)]}{[H(P^*) + H(P)]}, \tag{A.2}$$

Em que $H()$ é a função de entropia definida no Capítulo 3 (Eq. 3.27). O termo $H(P^*|P)$ se trata da entropia condicional, que é definida de forma geral para duas variáveis A e B quaisquer por:

$$H(B|A) = -\sum_{a,b} p(a,b) \log(p(a,b)/p(a)), \tag{A.3}$$

em que $p(a)$ é a probabilidade de $A = a$ e $p(a,b)$ denota a probabilidade de $B = b$ dado que $A = a$.

A.1.3 Homogeneidade e completeza

Outras métricas relacionadas ao cálculo da entropia são a homogeneidade (h) e a completeza (c). A homogeneidade mede o quanto cada *cluster* contém observações de uma única classe, sendo definida por:

$$h = 1 - \frac{H(P^*|P)}{H(P^*)}. \tag{A.4}$$

No cenário em que todas as observações em um *cluster* k qualquer pertencem à mesma classe, temos $h = 1.0$.

A completeza, por sua vez, mede o quão as observações de uma determinada classe são colocadas juntas pelo algoritmo de agrupamento:

$$c = 1 - \frac{H(P|P^*)}{H(P)}. \tag{A.5}$$

Neste caso, se todas as observações de uma certa classe forem associadas ao mesmo *cluster*, temos $c = 1.0$.

A.2 K-means

O k-means é um dos algoritmos de agrupamento mais populares[47], sendo aplicado exclusivamente sobre atributos numéricos. Antes do treinamento, é necessário especificar o número de *clusters* (k) que devem ser sintetizados.

O algoritmo k-means divide um conjunto com observações x_i ($i = 1,...,N$) em k *clusters* distintos (C); cada um descrito pelo ponto médio u_j ($j = 1,...,k$) das observações no *cluster*. Comumente estes pontos médios são chamados de centróides[2]. Na prática, o treinamento do k-means, que é iterativo, se realiza com o seguintes passos:

- Passo 1: as posições iniciais dos k centróides são escolhidas aleatóriamente.
- Passo 2: todas as observações são associadas ao seu centróide mais próximo. Tal associação está baseada no cálculo das respectivas distâncias euclidianas.
- Passo 3: novos centróides são criados tomando a média das observações associadas aos centróides anteriores.
- Passo 4: a diferença entre os novos centróides e os anteriores é calculada. Os passos 2 e 3 são repetidos até este valor ser menor que um certo limiar - em outras palavras, tal repetição é feita até que a posição dos centróides não mude significativamente (convergência).

Uma representação pictórica dos passos descritos pode ser vista na Figura A.1.

Vale destacar algumas das desvantagens do algoritmo k-means. Por trabalhar unicamente com distâncias euclidianas, o k-means tem menor efetividade em lidar com *cluster* alongados ou com formatos mais complexos. Pela mesma razão, este algoritmo também não é capaz de trabalhar com atributos categóricas. Finalmente, a escolha inicial do número de *cluster* pode não ser trivial para muitos problemas - especialmente quando o usuário não possui qualquer conhecimento de domínio.

[2]Note que o vetor u_j é representado no mesmo espaço das observações x_i, apesar de não fazer parte do conjunto de dados de entrada.

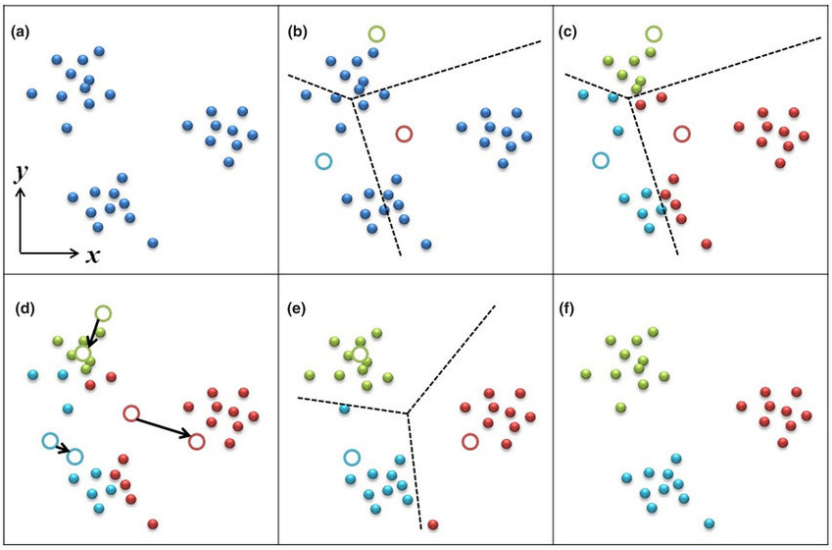


Figura A.1: Representação pictórica do agrupamento bidimensional de observações pelo algoritmo k-means. (a) As observações (círculos sólidos azuis) a serem agrupadas no espaço bidimensional. (b) Cada observação é associada ao centróide mais próximo (círculos azul claro, verde e vermelho escuro), cujas posições são inicialmente aleatórias. (c) O espaço bidimensional é dividido por três fronteiras de decisão (linhas pretas tracejadas). (d) Cada centróide se move em direção ao ponto médio das observações associadas ao seu atual *cluster*. (e) As observações são novamente associadas aos centróides, mas agora considerando as suas posições atualizadas. Os passos (c) e (d) são repetidos até o algoritmo convergir. (f) *Clusters* obtidos após a convergência. Figura retirada de [71].

A.3 DBSCAN

O DBSCAN (do inglês, *Density-Based Spatial Clustering of Applications with Noise*) é um algoritmo que trabalha considerando a densidade dos dados no espaço de atributos [72]. Os *clusters* são definidos como regiões com altas densidades de dados, que são limitadas por áreas de baixa densidade - por esta razão, o DBSCAN é capaz de encontrar *clusters* com formatos complexos. Além disso, não é necessário que o número de *clusters* seja definido *a priori*.

Os principais hiperparâmetros deste algoritmo são:

- ε: distância máxima para duas observações serem consideradas do mesmo *cluster*.
- $MinPts$: Número mínimo de observações para uma região ser considerada um *cluster*.

Durante o ajuste, o DBSCAN define as observações de três formas: os núcleos (*cores*), as bordas (*borders*) e os ruídos (*noises*). Uma observação é categorizada como núcleo se possuir pelo menos $MinPts$ dentro de um raio ε de distância. Caso uma observação esteja dentro do raio ε de um núcleo mas não tenha $MinPts$ observações no seu raio, é considerada como uma borda. Por fim, ruídos são observações que não estão no raio ε de um núcleo e que não possuem $MinPts$ observações no seu raio. A Figura A.2 ilustra estes conceitos.

De forma simples, o ajuste do DBSCAN - que é iterativo - ocorre com os seguintes passos:

- Passo 1: Escolha um ponto (observação) arbitrário que não foi visitado ainda.

- Passo 2: Determine a vizinhança deste ponto a partir da distância ε e o classifique como núcleo, borda ou ruído.
- Passo 3: Caso o ponto seja um núcleo, todos os pontos da sua vizinhança em um raio ε formam um *cluster* inicial. Os passos 1 e 2 são feitos para todos os demais pontos deste *cluster* - tal processo se repete até todos os pontos em uma distância ε do *cluster* serem visitados e rotulados.
- Passo 4: Uma vez que terminamos com o *cluster* atual, repetimos os passos 1-3 na busca de novos *clusters*.

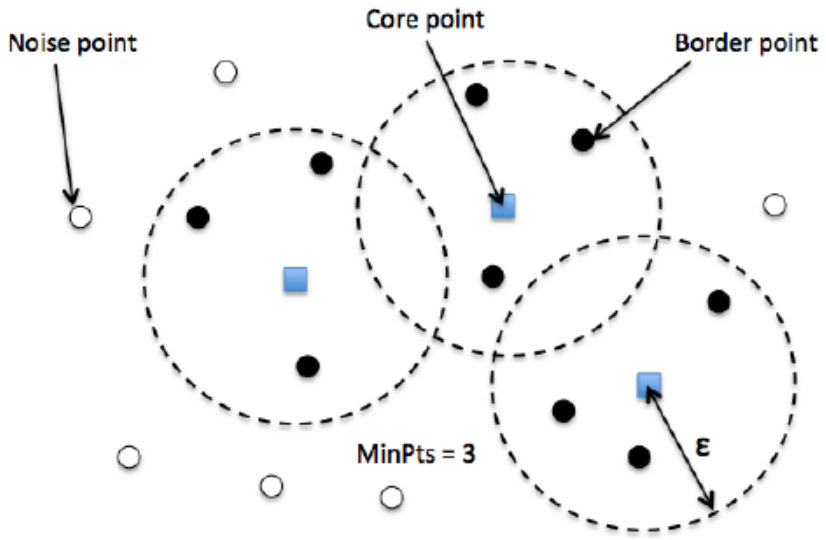


Figura A.2: Representação de como o DBSCAN classifica as observações durante o treinamento - considerando $MinPts = 3$ e um valor arbitrário de ε. Os núcleos, as bordas e os ruídos estão representados pelos quadrados azuis, pelos círculos sólidos pretos e pelos círculos vazios, respectivamente. Figura retirada de [73].

A maior desvantagem do DBSCAN está relacionada a sua baixa capacidade em diferenciar *clusters* com densidades variáveis - afinal, a determinação dos *clusters* é fortemente dependente dos valores escolhidos para ε e MinPts. Além disso, o DBSCAN também não é capaz de lidar com variáveis categóricas.

A.4 Agrupamento de eventos

Empregamos os algoritmos k-means e DBSCAN para agrupar 10^5 eventos caracterizados pelos atributos globais da tabela 4.1. Os hiperparâmetros das técnicas foram ajustados realizando uma busca em *grid* - a escolha dos modelos finais levou em consideração a otimização das métricas apresentadas na seção A.1. Pelo fato dos eventos possuírem dois rótulos conhecidos (eventos com e sem *pile-up*), escolhemos $k = 2$ para o algoritmo k-means. O DBSCAN foi configurado com $\varepsilon = 400$ e $MinPts = 200$.

Os resultados obtidos com as duas técnicas estão resumidos na Tabela A.1. Todas as métricas de

desempenho apresentaram valores próximos à 0.0 para o algoritmo k-means, apontando uma baixa capacidade deste algoritmos em separar os eventos. O DBSCAN, por sua vez, apresentou um desempenho considerávelmente superior ao do k-means (duas ordens de grandeza). Contudo, todas as figuras de mérito ainda se mostraram próximas à zero - indicando que este algoritmo também não foi capaz de discriminar adequadamente os dois tipos de eventos.

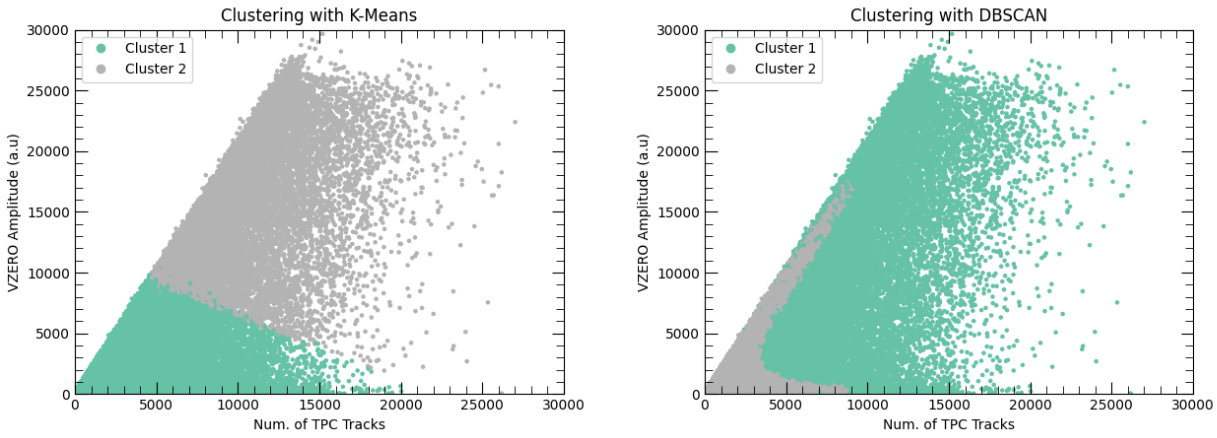
Tabela A.1: Resultados obtidos com o agrupamento de eventos pelos algoritmos k-means e DBSCAN.

Métricas	*k-means* (10^{-4})	DBSCAN (10^{-2})
Índice ajustado de Rand	7.98	6.92
Informação Mútua	5.17	4.73
Homogeneidade	7.45	6.82
Completeza	8.94	8.38

Além das métricas da seção A.1, é possível verificar a qualidade dos *clusters* sintetizados através de uma inspeção visual sobre os dados. A Figura 5.9 apresenta a correlação de amplitude do V0 com o número de trajetórias na TPC para os eventos agrupados pelos algoritmos k-means e DBSCAN. O k-means (Fig. A.3a) separou grosseiramente os eventos de acordo a amplitude do detector V0. O primeiro *cluster* está definido na região com eventos de baixa amplitude em V0, enquanto o segundo engloba eventos com altas amplitudes. A fronteira de decisão, que separa os *clusters*, apresenta um aspecto linear.

O DBSCAN (Fig. A.3b) produziu *clusters* cuja fronteira de decisão é não-linear. O *cluster* 2 (cinza) se estendeu por parte da região diagonal do gráfico, onde se encontram os eventos principais. Contudo, este *cluster* também se estendeu para a região de baixa amplitude em V0 e alta multiplicidade na TPC, onde se encontra grande parte dos eventos com ocorrência de *pile-up*. O *cluster* 2 foi definido como a região com maior densidade de dados em toda a distribuição. Já os eventos associados ao *cluster* 1 (verde) foram considerados como ruídos, por estarem em regiões menos densas do espaço de atributos.

Com base nos resultados da tabela A.1 e da Figura A.3, é possível concluir que tanto o k-means quanto o DBSCAN não se mostraram alternativas razoáveis - frente aos algoritmos de aprendizado supervisionado - para discriminar eventos. Nos dois casos, os algoritmos não conseguiram definir fronteiras de decisão adequadas - que, idealmente, deveriam ter um formato próximo ao da expressão 2.4. Como resultado, todos os *clusters* produzidos apresentaram um baixo grau de pureza, por conterem quantidades significativas de eventos com e sem *pile-up*.



(a) (b)

Figura A.3: Correlação de informações dos detectores V0 e TPC para os eventos agrupados pelos algoritmos (a) k-means e (b) DBSCAN.

Apêndice B

Especificações técnicas

B.1 Bibliotecas

Os resultados deste trabalho foram gerados com os seguintes pacotes e bibliotecas:

Python: v.2.716 (e v.3.6.9, para o treinamento de Redes Neurais Artificias apenas);

Pandas [74]: v. 1.1.5;

Scikit-Learn [75]: v. 1.0.1;

Keras [76]: v. 2.7.0;

Tensorflow [77]: v. 2.7.0;

XGBoost [54]: v. 0.82;

Scipy [78]: v. 1.2.2 (para aplicação dos métodos de filtragem);

PyTorch [79]: v. 1.10.2;

PyTorch Geometric [80]: v. 2.0.3;

B.2 Hiperparâmetros e configurações

As técnicas de ML empregadas neste estudo e a configuração dos seus principais hiperparâmetros estão descritas abaixo[1]:

Redes Neurais Artificiais (MLP):

- Biblioteca: Tensorflow com Keras.
- Arquitetura: Rede MLP formada por cinco camadas com 5, 5, 4, 3 e 2 neurônios respectivamente.
- Funções de ativação: Nas primeiras quatro camadas, a função de ativação foi a ReLU, enquanto a camada de saída foi construída com a função *Softmax*.

[1] Hiperparâmetros não citados foram deixados nos valores-padrão das bibliotecas.

- Função custo: Entropia Cruzada.
- Otimizador: ADAM.
- *Learning rate*: 1.0e-3.

Random Forest:

- Biblioteca: Scikit-Learn.
- Número de árvores (estimadores): 30.
- Profundidade máxima: 20.
- Critério: Entropia.

Boosted Decision Tree:

- Biblioteca: XGBoost.
- Número de árvores (estimadores) 150.
- Profundidade máxima: 10.
- *Learning rate*: 0.1.
- Objective: binary-logistic.

Regressão Logística :

- Biblioteca: Scikit-Learn.
- *Solver*: Newton-CG.
- Parâmetro de Regularização (C): 10.
- Penalidade: L2.

Máquinas de Vetores-Suporte:

- Biblioteca: Scikit-Learn.
- Função Kernel: Função de base-radial (rbf).
- Parâmetro de Regularização (C): 10.
- *Probability: True*.

Redes Neurais Artificiais (GNN):

- Biblioteca: PyTorch com PyTorch Geometric.
- Arquitetura: Rede formada por três camadas GraphConv com 32, 32 e 16 dimensões respectivamente. A camada de leitura consiste em *Global Mean Pool* e é seguida por uma rede MLP formada por quatro camadas com 10, 8, 4 e 2 neurônios respectivamente.
- Funções de ativação: Nas camadas GraphConv e MLP as funções de ativação empregadas foram, respectivamente, tangente hiperbólica e ReLU. A saída foi construída com a função *Softmax*.
- Função custo: Entropia Cruzada.
- Otimizador: ADAM.
- *Learning rate*: 1.0e-4.

K-means:

- Biblioteca: Scikit-Learn.
- Número de *clusters* (k): 2.
- Algoritmo: Elkan.
- N. máximo de iterações: 1000.

DBSCAN:

- Biblioteca: Scikit-Learn.
- ε: 400.
- MinPts: 200.
- Métrica: Euclidean.

Apêndice C

Atuação dentro da colaboração ALICE

Normalmente, um aluno da colaboração ALICE deve cumprir tarefas internas - conhecidas como *service tasks* - e participar dos turnos de tomadas de dados do experimento - conhecidos como *shifts*. Apesar destas atividades não serem obrigatórias para alunos de Mestrado, seria salutar participar delas nesta etapa da formação. Contudo, em decorrência da pandemia de Covid-19, não foi possível viajar e atuar presencialmente no experimento. Mesmo assim, isto não me impediu de realizar atividades remotas dentro da colaboração, que incluem:

Operar códigos do ALICE

A maioria das análises de eventos do ALICE são feitas em alta velocidade pelo *Grid*, que é um sistema de computação do CERN formado por vários *clusters*. As tarefas são submetidas ao *Grid* na forma de códigos em C++ que estão baseados na biblioteca AliRoot. Durante o período de Mestrado, fui introduzido a este *framework* e aprendi a editar alguns códigos da colaboração para criar o conjunto de dados utilizado neste trabalho. Nesse ínterim, também realizei algumas correções em outros códigos de análise. Tanto a edição dos códigos quanto a submissão de tarefas para o *Grid* foram feitas com o apoio do Prof. Dr. David Dobrigkeit Chinellato, também membro da colaboração ALICE.

Revisão de artigos

Uma das atribuições das instituições que fazem parte da colaboração ALICE é a de, eventualmente, participar da avaliação de artigos que poderão ser submetidos para publicação. Como participante do ALICE, atuei na revisão de alguns destes artigos em conjunto com os demais membros do grupo HadrEx que fazem parte da colaboração.

Submissão de uma Nota de Análise

As Notas de Análise (*Analysis Notes*) são documentos cuja finalidade é a de detalhar a metodologia e fornecer as principais conclusões de análises efetuadas com dados do ALICE. Se trata, portanto, de uma documentação confidencial que deve ser submetida para avaliações internas apenas. Parte dos resultados apresentados neste trabalho também foram documentados desta forma e submetidos à colaboração.

Data Preparation Group

Alguns dos nossos resultados também foram apresentados e discutidos no *Data Preparation Group* (DPG), que é a divisão responsável por - dentre outras coisas - coordenar a reconstrução de eventos, produções de Monte Carlo e realizar os controles de qualidade dos dados coletados pelo experimento ALICE.

Aprovação e apresentação da análise

Finalmente, os resultados contidos neste trabalho foram submetidos e aprovados pelos grupos DPG e PDP (*The Physics Data Processing project*) do ALICE. Esta aprovação me possibilitou apresentar a análise em nome da colaboração nos eventos da área, como a "XXIX Reunião de Trabalho sobre Interações Hadrônicas"que ocorreu em maio de 2022.