

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE SÃO JOÃO DA BOA VISTA

JOÃO RAPHAEL CIOFFI

Sistema de Planejamento de Vôo Autônomo Utilizando Inteligência Artificial

São João da Boa Vista

2022

João Raphael Cioffi

Sistema de Planejamento de Vôo Autônomo Utilizando Inteligência Artificial

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Aeronáutica do Campus de São João da Boa Vista, Universidade Estatal Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Aeronáutica .

Orientador: Profº Dr. Murilo Sartorato

Coorientador: Profº Dr. Cláudio Fabiano Motta
Toledo

São João da Boa Vista

2022

C576s	<p>Cioffi, João Raphael Sistema de planejamento de vôo autônomo utilizando inteligência artificial / João Raphael Cioffi. -- São João da Boa Vista, 2022 79 p. : il., tabs., mapas</p> <p>Trabalho de conclusão de curso (Bacharelado - Engenharia Aeronáutica) - Universidade Estadual Paulista (Unesp), Faculdade de Engenharia, São João da Boa Vista Orientador: Murilo Sartorato Coorientador: Cláudio Fabiano Motta Toledo</p> <p>1. Inteligência artificial. 2. Redes neurais (Computação). 3. Análise por agrupamento. 4. Drone. I. Título.</p>
-------	---

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Engenharia, São João da Boa Vista. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

**UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE SÃO JOÃO DA BOA VISTA**

JOÃO RAPHAEL CIOFFI

**ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO
REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE "GRADUANDO EM ENGENHARIA
AERONÁUTICA "**

**APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM
ENGENHARIA AERONÁUTICA**

**Profº Dr. DENILSON PAULO SOUZA DOS SANTOS
Coordenador**

BANCA EXAMINADORA:

**Profº Dr. Murilo Sartorato
Orientador/UNESP-SJBV**

**Profº Dr. Carlos do Carmo Pagani Junior
UNESP-SJBV**

**Profº Dr. Elmer Mateus Gennaro
Membro Externo**

Dezembro , 2022

Dedico este trabalho à minha família - em especial à minha querida mãe - e aos que fizeram parte da minha trajetória acadêmica - docentes e amigos.

“Without big data, you are blind and deaf and in the middle of a freeway.”
(Geoffrey Moore)

RESUMO

Com o aumento no uso de Veículos Aéreos não Tripulados (UAVs/Drones) (SADHU; ZONOUZ; POMPILI, 2020), o desenvolvimento de técnicas para detecção e identificação de falhas em tempo real vem sendo cada vez mais necessário para que se possa ter o devido controle e a recuperação do veículo em um cenário potencial de colisão. As causas da colisão podem estar relacionadas à falhas no sistema de sensoriamento/atuação, à falhas de componentes estruturais - como por exemplo *birdstrikes* (JHA; SATHYAMOORTHY; PRAKASH, 2019) ou quebra de hélices -, ou ainda ataques cibernéticos ao sistema do veículo (BEST et al., 2020).

Neste aspecto, adotando-se técnicas de Aprendizagem de Máquina, criou-se uma arquitetura geral para tomada de decisão no pré-voo (previsão de dados climáticos) e planejamento de missão em tempo real do voo (detecção e classificação de operações incorretas do drone). A arquitetura proposta deve ser capaz de aprender a dinâmica temporal do sistema de modo automático utilizando os dados brutos. Os resultados empíricos devem mostrar que a solução proposta aqui será capaz de detectar e classificar os principais tipos de comportamento inadequado do sistema.

PALAVRAS-CHAVE: Drones. VANTs. Dados climáticos. Dados de voo. Inteligência Artificial. Redes Neurais. Identificação de Anomalias. Análise por Agrupamento.

ABSTRACT

With the increase in the use of Unmanned Aerial Vehicles (UAVs/Drones) (SADHU; ZONOUZ; POMPILI, 2020), the development of techniques for detecting and identifying faults in real time is becoming increasingly necessary so that you can have proper control and recovery of the vehicle in a potential collision scenario. The causes of collisions may be related to sensors/actuators failures, components structural failures - such as birdstrikes (JHA; SATHYAMOORTHY; PRAKASH, 2019) or breaking propellers -, or even cyber attacks on vehicle's system (BEST et al., 2020).

In this aspect, by adopting Machine Learning techniques, a general architecture was created for a pre-flight decision making (climate data forecast) and a real-time flight mission planning (detection and classification of incorrect drone operations). The proposed architecture must be able to learn the temporal dynamics of the system automatically using raw data. The empirical results must show that the solution proposed here will be able to detect and classify the main types of system misbehavior.

KEYWORDS: Drones. UAVs. Weather Data. Flight Data. Artificial Intelligence. Neural Networks. Anomaly Detection. Cluster Analysis.

LISTA DE ILUSTRAÇÕES

Figura 1	Diagrama de Venn para os campos de estudo da computação numérica	17
Figura 2	Esquematização da programação clássica vs. Machine Learning	18
Figura 3	Parametrização de uma rede neural pelos pesos	19
Figura 4	Esquematização do fluxo de dados até a determinação da perda da rede	19
Figura 5	Esquematização completa do funcionamento da rede	20
Figura 6	Representações de um tensor	21
Figura 7	Representação de uma série temporal como uma sequência de funções densidade	23
Figura 8	Representação esquemática de uma Árvore de Decisão	24
Figura 9	Exemplificação do funcionamento de uma Árvore	25
Figura 10	Diagrama de um Algoritmo Random Forest	28
Figura 11	Rede neural elementar (Perceptron)	30
Figura 12	Funções de ativação comumente utilizadas	31
Figura 13	Multilayer Perceptron	32
Figura 14	Diferença entre uma rede neural simples e uma rede neural profunda	33
Figura 15	Algoritmo <i>Gradient-Descent</i>	34
Figura 16	Ajuste de pontos através de uma função Sigmoid	35
Figura 17	Representação básica de uma célula LSTM	36
Figura 18	Estrutura básica de uma operação Autoencoder com LSTM	38
Figura 19	Representação de <i>features</i> através de uma clusterização	40
Figura 20	Estágios de um processo de clusterização	41
Figura 21	Curva de Inércia	44
Figura 22	Arquitetura geral do sistema	46
Figura 23	Parte do repositório do Github criado para arquivar o projeto	47
Figura 24	Exemplo de estações de leitura METAR disponíveis no Brasil	48
Figura 25	Histograma de frequência para as principais variáveis numéricas do METAR	50
Figura 26	Matriz de correlação numérica (Pearson)	51
Figura 27	Umidade relativa [%] vs. temperatura do ar [°F] mapeados por suas densidades	52
Figura 28	Curvas de distribuição para as condições de voo	53
Figura 29	Análise proporcional das condições de voo	54
Figura 30	Sistema utilizado para a geração dos dados de voo por simulação	54
Figura 31	Histograma de frequência para as principais variáveis de voo	56
Figura 32	Matriz de correlação numérica (Pearson)	57
Figura 33	Exemplo da <i>Randomized Search</i> utilizada no processo	60
Figura 34	Matriz de correlação entre variáveis esperadas e variáveis previstas pelo algoritmo	61
Figura 35	Histograma para a temperatura do ar esperada e temperatura do ar prevista	61
Figura 36	Dispersão: temperatura do ar esperada vs. temperatura do ar prevista	62
Figura 37	Histograma para a umidade relativa do ar esperada e umidade relativa do ar prevista	62

Figura 38 Dispersão: umidade relativa do ar esperada vs. umidade relativa do ar prevista	63
Figura 39 Histograma para a velocidade do ar esperada e velocidade do ar prevista	63
Figura 40 Dispersão: velocidade do ar esperada vs. velocidade do ar prevista	64
Figura 41 Histograma para a pressão barométrica do ar esperada e pressão barométrica do ar prevista	64
Figura 42 Dispersão: pressão barométrica do ar esperada vs. pressão barométrica do ar prevista	65
Figura 43 Curvas de distribuição de densidade dos erros para algumas variáveis previstas em y	66
Figura 44 Trecho do código implementado para construir a rede	67
Figura 45 Exemplo da arquitetura da rede neural utilizada no processo	68
Figura 46 Decaimento da perda em função do número de Batch	68
Figura 47 Gráfico de dispersão 2D para as taxas temporais de roll, pitch e yaw	70
Figura 48 Gráfico de dispersão 3D para as taxas temporais de roll, pitch e yaw mapeadas pela taxa temporal de subida	70
Figura 49 Trecho do código implementado	71
Figura 50 Exemplo de tabela gerada como resposta contendo a coluna de labels criadas	72
Figura 51 Gráfico de radar mostrando a clusterização de anomalias com base nas variáveis de voo	72
Figura 52 Gráfico setores mostrando a relação percentual obtida na clusterização	73
Figura 53 Exemplo de execução do sistema através de um terminal	74
Figura 54 Exemplo de ruído gerado na taxa de subida da aeronave	74
Figura 55 Drone DJI Tello usado como protótipo para testes futuros	75

LISTA DE TABELAS

Tabela 1 – Configuração de Hardware e Software	45
Tabela 2 – Listagem das Principais Dependências do Python Utilizadas	45
Tabela 3 – Interpretação Estatística dos Dados	49
Tabela 4 – Definições para as diferentes categorias de voo	52
Tabela 5 – Interpretação Estatística dos Dados	56
Tabela 6 – Valores de R^2 encontrados para cada uma das variáveis em questão	59
Tabela 7 – Métricas finais obtidas na matriz resposta do modelo	60
Tabela 8 – Métricas finais obtidas na matriz resposta do modelo	69

LISTA DE ABREVIATURAS E SIGLAS

METAR	Meteorological Aerodrome Report
GPU	Graphics Processing Unit
CPU	Central Processing Unit
ML	Machine Learning
DL	Deep Learning
IA	Inteligência Artificial
VANT	Veículo Aéreo não Tripulado
UAV	Unmanned Aerial Vehicle
PIP	Package Installer for Python
API	Application Programming Interface
n/a	Não Aplicável

LISTA DE SÍMBOLOS

ϕ	Roll
θ	Pitch
ψ	Yaw
$E(s)$	Função de entropia
$G(s, x)$	Função ganho
w	Parâmetro peso de uma rede neural
b	Parâmetro viés de uma rede neural
$L(f(x))$	Função Perda
$\mathcal{L}(\beta)$	Função de Verossimilhança ou <i>Likelihood</i>
β	Coeficiente linear de uma regressão
X	Variáveis de entrada do modelo
y	Variáveis de resposta do modelo
$\{X_{train}, y_{train}\}$	Conjunto de dados de treino
$\{X_{test}, y_{test}\}$	Conjunto de dados de teste

SUMÁRIO

1	INTRODUÇÃO	15
1.1	O PARADOXO DAS ANOMALIAS	15
1.2	O USO DE INTELIGÊNCIA ARTIFICIAL	15
1.3	O USO DE REDES NEURAIS	16
2	REVISÃO BIBLIOGRÁFICA E FUNDAMENTAÇÃO TEÓRICA	17
2.1	CONCEITOS GERAIS	17
2.1.1	Inteligência Artificial	17
2.1.2	Machine Learning	18
2.1.3	Deep Learning	18
2.1.4	Estruturas de Dados em Redes Neurais	20
2.1.4.1	Conceito de Tensores	20
2.1.4.2	Exemplos Reais de Tensores	22
2.1.5	Séries Temporais	22
2.2	CONCEITOS DETALHADOS	22
2.2.1	Decision Trees	22
2.2.1.1	O que é uma Árvore de Decisão?	23
2.2.1.2	Tipos de Árvores	25
2.2.1.3	Atributos Inerentes aos Nós de uma Árvore	25
2.2.1.4	Decodificando Hiperparâmetros de uma Árvore	26
2.2.1.5	Random Forests	27
2.2.1.5.1	<i>Métodos Ensemble</i>	27
2.2.1.5.2	<i>O Algoritmo Random Forest</i>	27
2.2.1.6	Boosting dos Modelos	28
2.2.1.6.1	<i>Por que o Boosting é Importante?</i>	29
2.2.1.6.2	<i>Processo de Treinamento por Boosting</i>	29
2.2.1.6.3	<i>Tipos de Boosting</i>	29
2.2.2	Redes Neurais Artificiais	30
2.2.2.1	Perceptrons	30
2.2.2.2	Layers	31
2.2.2.3	Multilayer Perceptron (MLP)	32
2.2.2.4	Treinamento Supervisionado	33
2.2.2.5	Função Sigmoid (σ)	34
2.2.2.6	Long Short-Term Memory (LSTM)	36
2.2.2.7	LSTM Autoencoder	37
2.2.2.8	Hiperparâmetros de uma Rede Neural	38
2.2.3	Clusterização dos Dados	39

2.2.3.1	Componentes de uma Rotina de Clusterização	40
2.2.3.2	Definições e Notação	41
2.2.3.3	Medidas de Similaridade	42
2.2.3.4	Relação entre Inércia e Número de Clusters	43
3	METODOLOGIA, RESULTADOS E DISCUSSÃO	45
3.1	CONFIGURAÇÕES DE HARDWARE E SOFTWARE	45
3.2	ESCOPO GERAL DO TRABALHO	45
3.3	OBJETIVO GERAL	46
3.4	OBJETIVO ESPECÍFICO	46
3.5	REPOSITÓRIO DO PROJETO	47
3.6	PRIMEIROS PASSOS: GERANDO A BASE DE DADOS	47
3.6.1	Dados Meteorológicos	47
3.6.1.1	Abordagem Contextual	47
3.6.1.2	Abordagem Prática	49
3.6.1.2.1	<i>Análise Exploratória dos Dados</i>	49
3.6.1.2.2	<i>Feature Engineering</i>	51
3.6.2	Dados de Voo por Simulação	54
3.6.2.1	Abordagem Contextual	54
3.6.2.2	Abordagem Prática	55
3.6.2.2.1	<i>Análise Exploratória dos Dados</i>	56
3.7	MÓDULO 1: PREVISÃO DE DADOS METAR	58
3.7.1	Criação das Matrizes Adequadas ao Modelo	58
3.7.2	Seleção do Modelo	58
3.7.3	Ajuste do Modelo Através dos Hiperparâmetros	59
3.7.4	Resposta do Modelo	60
3.7.4.1	Resposta Geral	60
3.7.4.2	Respostas Isoladas por Variável	61
3.7.4.2.1	<i>Temperatura do Ar</i>	61
3.7.4.2.2	<i>Umidade Relativa do Ar</i>	62
3.7.4.2.3	<i>Velocidade do Ar</i>	63
3.7.4.2.4	<i>Pressão Barométrica</i>	64
3.7.4.3	Análise Distributiva dos Resíduos	65
3.8	MÓDULO 2: IDENTIFICAÇÃO DE PADRÕES DE ANOMALIA	66
3.8.1	Criação das Matrizes Adequadas ao Modelo	66
3.8.2	Configuração do Modelo	67
3.8.3	Resposta do Modelo	68
3.9	MÓDULO 3: CLASSIFICANDO ANOMALIAS EM SUBNÍVEIS PARA TOMADA DE DECISÃO	69
3.9.1	Criação das Matrizes Adequadas ao Modelo	69
3.9.2	Configuração do Modelo	71

3.9.3	Respostas do Modelo	71
3.10	RESPOSTA CAÓTICA DO SISTEMA: GERANDO RUÍDOS NA BASE DE DADOS	73
3.11	TRABALHOS FUTUROS	75
4	CONCLUSÃO	76
	REFERÊNCIAS	77

1 INTRODUÇÃO

1.1 O PARADOXO DAS ANOMALIAS

Um grande ponto de decisão sobre um conjunto de dados ser ou não classificado como anomalia deve levar em conta fatores que vão além de simplesmente categorizá-los como *outliers* (BEN-GAL, 2005). *Outliers*, neste sentido, são definidos como valores atípicos, isto é, observações que apresentam afastamento das demais da série, ou que sejam inconsistentes no contexto no qual. Numa análise descritiva dos dados, a presença deles pode prejudicar a interpretação dos resultados dos testes estatísticos aplicados às amostras. No sentido geral, é esperado que toda anomalia represente um *outlier* na distribuição dos dados, mas o contrário nem sempre será verdadeiro.

Imaginemos, por exemplo, o cenário de uma aeronave militar (KOWALECZKO; PIATKOWSKI, 2022). Dentro deste contexto operacional, é esperado que as respostas temporais das principais variáveis de voo (ϕ, θ, ψ) estejam bem acima da distribuição normal destes dados para situações de manobras críticas. Se partirmos da premissa de apenas olhar para *outliers* e classificá-los como indivíduos anômalos, teremos um viés por não conhecer de fato o contexto dos dados para os quais estamos olhando. Apesar de estarem fora da representação gaussiana esperada, situações desse tipo costumam ser bem controladas justamente por estarem inseridas no contexto da missão ou treinamento de pilotos militares. Desta forma, para classificar uma anomalia da maneira correta, devemos adotar técnicas mais robustas que vão além de simplesmente entender a nossa distribuição amostral ou populacional.

1.2 O USO DE INTELIGÊNCIA ARTIFICIAL

Para uma devida compreensão de um modelo físico, sobretudo quando estamos lidando com princípios de robótica móvel (que é o caso de drones, por exemplo), seria necessário entender o equacionamento dinâmico da aeronave levando em consideração, dentre outras variáveis, as derivativas aerodinâmicas (WRIGHT; COOPER, 2008). Isso permite, além de uma modelagem robusta e fidedigna para compreensão dos nossos dados, que apliquemos uma filtragem baseada em Kalman (TING; THEODOROU; SCHAAL, 2007) para a detecção de anomalias.

Entretanto, o uso de IA se faz aqui extremamente necessário devido aos fatores apresentados a seguir:

1. Infraestrutura: devido a falta de um modelo físico previamente destinado à elaboração deste trabalho, toda a coleta de dados foi realizada através de voos simulados utilizando o gerenciador de missões *QGroundControl* (DRONECODE, 2022) e o simulador de robótica *Gazebo* (GAZEBO, 2022).
2. Estado da arte: apesar de técnicas mais tradicionais voltadas para a aproximação física dos modelos ("physics-based"), a combinação entre inteligência artificial e entendimento estatístico de nossos dados ("data-driven") está em alta e tem se mostrado cada vez mais eficaz no que diz

respeito à tradução direta e interpretação do mundo real através de números (AN; KIM; CHOI, 2015).

Portanto, devido à complexidade em se modelar uma aeronave real utilizando as equações dinâmicas e ao grande interesse pessoal em contribuir para o avanço técnico-científico dos estudos voltados para a inteligência artificial, optou-se por manter essas técnicas neste trabalho como as principais ferramentas.

1.3 O USO DE REDES NEURAIS

As principais razões para se escolher *Deep Learning* como uma das principais ferramentas neste trabalho são apresentadas a seguir:

1. Simplicidade: não há necessidade de aplicar *Feature Engineering*; o DL utiliza modelos treináveis de ponta a ponta amplamente desenvolvido pela comunidade e que podem ser facilmente construídos através de operações com tensores.
2. Escalabilidade: devido à paralelização das TPUs/GPUs, a Lei de Moore (MOLICK, 2006) oferece vantagens ao DL e muito além disso, pois os modelos são treinados através de iterações sequenciais sobre conjuntos de dados amostrais de dimensões arbitrárias.
3. Versatilidade e Reusabilidade: ao contrário das técnicas de aprendizado de máquina comumente usadas, os modelos de DL não exigem uma retomada de fluxo de trabalho partindo do zero para retreino dos dados adicionais - o que é um fator extremamente qualitativo quando estamos lidando com a produtização de nossos modelos. Além disso, a reusabilidade tem muito a ver com o que se espera neste trabalho: modelos treinados para a classificação de anomalias podem ser facilmente integráveis a uma *pipeline* e embarcados junto ao *software* principal de um sistema.

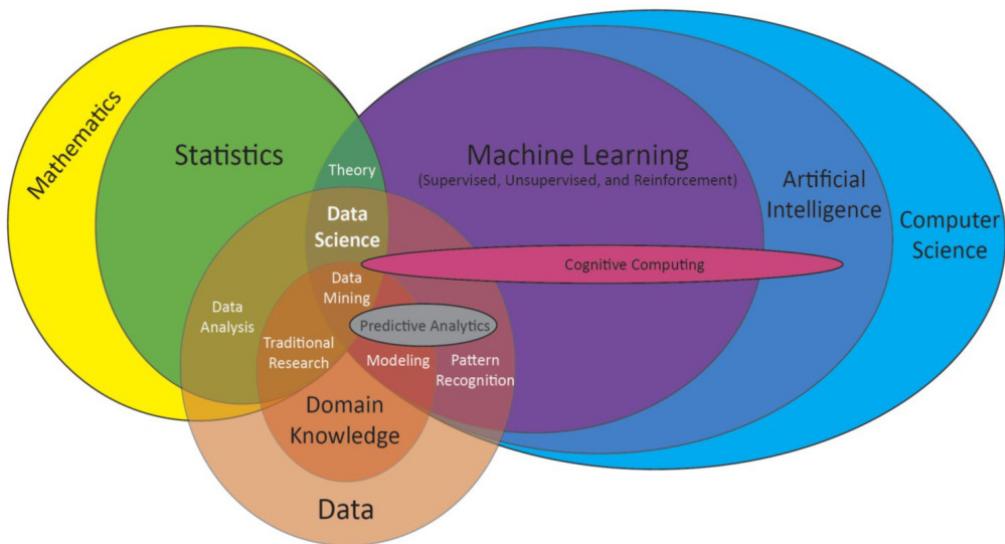
2 REVISÃO BIBLIOGRÁFICA E FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica será feita nas próximas seções, baseada nos conceitos introduzidos pela revisão bibliográfica das referências contidas neste trabalho.

2.1 CONCEITOS GERAIS

Nos últimos anos, a inteligência artificial de um modo geral tem sido um objeto de grande repercussão na comunidade científica e acadêmica, bem como na mídia (já que muitas vezes os termos *Machine Learning* e IA aparecem em inúmeros artigos, muitas vezes fora das publicações voltadas para a tecnologia) (GÉRON, 2022). Apesar do *hype* excessivo gerado pelo estado da arte desses termos, é importante termos uma visão macro, pois podem representar áreas distintas dentro da computação a depender do campo de estudo. A Figura 1 apresenta uma distinção entre eles:

Figura 1 – Diagrama de Venn para os campos de estudo da computação numérica.



Fonte: Adaptado de Ryan Urbanowicz, PhD, University of Pennsylvania (TWITTER, 2018).

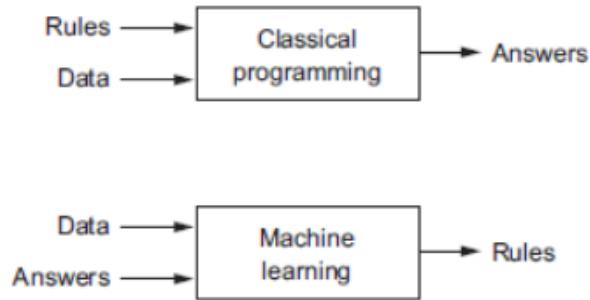
2.1.1 Inteligência Artificial

A definição de Inteligência Artificial (GÉRON, 2022) está baseada no esforço para automatizar tarefas intelectuais normalmente realizadas por humanos (como por exemplo o Teste de Alan Turing (FRENCH, 2000), o precursor da IA como a conhecemos hoje em dia). De um modo geral, é um campo que engloba as técnicas comuns de aprendizado de máquina e os algoritmos recentes de aprendizagem profunda, mas também pode incluir diversas outras abordagens que não envolvem, necessariamente, um aprendizado propriamente dito (isto é, a capacidade do computador reconhecer padrões num conjunto de dados e interpretar o mundo real através deles). Esta última abordagem é tida como uma IA simbólica e foi o paradigma dominante desde a década de 1950 até o *boom* dos sistemas computacionais em meados da década de 1980.

2.1.2 Machine Learning

Diferentemente de uma IA simbólica ou da programação clássica, um sistema integrado de ML é treinado ao invés de ser explicitamente programado pelo usuário. Para que ele execute uma determinada tarefa, são apresentados à ele inúmeros exemplos relevantes afim de que se verifique relações estatísticas entre as variáveis (podendo ser mútuas/correlatas ou não e, na maioria das vezes, não intuitivas para uma análise macro de um ser humano). Nessas interações é possível que, eventualmente, existam regras para a automação dos processos. A Figura 2 apresenta conceitualmente a diferença entre as arquiteturas de uma programação básica e de uma programação voltada para aprendizado de máquina:

Figura 2 – Esquematização da programação clássica vs. Machine Learning



Fonte: Adaptado de Aurélien Geron, 2022.

2.1.3 Deep Learning

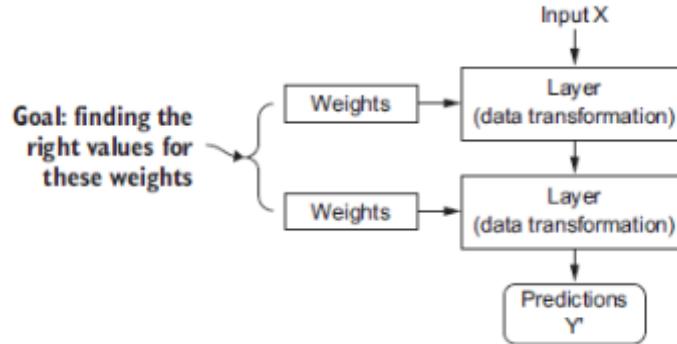
O *Deep Learning* é um subcampo específico do ML (GÉRON, 2022), em que a aprendizagem dos dados é feita através de camadas dispostas sucessivamente para que se tenha uma representação significativa. Esse conceito é a extrapolação do que, de um modo extremamente simplista, temos na representação da teoria dos grafos para solução de problemas complexos (NETTO, 2003).

O termo "*deep*" em DL não é uma referência a nenhum tipo de compreensão mais profunda alcançada pela abordagem; ao invés disso, ele representa essa ideia previamente citada sobre a disposição sucessiva das camadas de representações. O agrupamento de diversas camadas contribuindo hierarquicamente e mutuamente entre si é chamada de profundidade de um modelo.

O *Deep Learning* moderno geralmente envolve dezenas ou até centenas de camadas sucessivas de representações em camadas, que são (quase sempre) aprendidas por meio de modelos chamados redes neurais. O termo rede neural é uma referência à neurobiologia, mas embora alguns dos conceitos centrais do aprendizagem profunda tenham sido desenvolvidos em parte por inspiração em nossa compreensão do cérebro, os modelos mais genéricos não são modelos do cérebro humano (não há evidências científicas de que o cérebro implemente algo parecido com os mecanismos de um modelo conceitual de DL). De uma forma mais ampla, podemos estender os conceitos básicos de DL como sendo uma estrutura matemática capaz de aprender as representações dos dados aos quais ela é submetida (representações que vão desde a correlação numérica de Pearson, r , até a dependência temporal de modelos mais complexos).

Até este ponto, sabemos que a aprendizagem de máquina trata do mapeamento de cada variável de entrada através de uma sequência de transformações numéricas simples (camadas), e que essas transformações são aprendidas pela exposição a exemplos (dados).

Figura 3 – Parametrização de uma rede neural pelos pesos

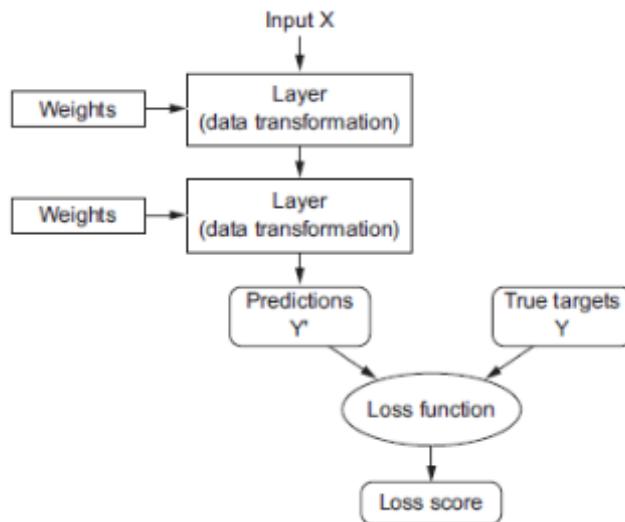


Fonte: Adaptado de Aurélien Geron, 2022.

Em termos técnicos, a transformação implementada por uma camada é parametrizada por seus pesos (Figura 3). Nesse contexto, aprender significa encontrar um conjunto de valores para os pesos de todas as camadas em uma rede, de modo que ela mapeie corretamente as entradas (exemplos) para seus alvos associados.

Após mapeadas as entradas, a rede faz a construção da função perda (também chamada de função objetivo) para controlar sua saída. Em outras palavras, a função perda pega as previsões da rede e calcula uma pontuação de distância entre elas e os pontos-alvo (quanto menor a distância, maior será o desempenho da rede).

Figura 4 – Esquematização do fluxo de dados até a determinação da perda da rede

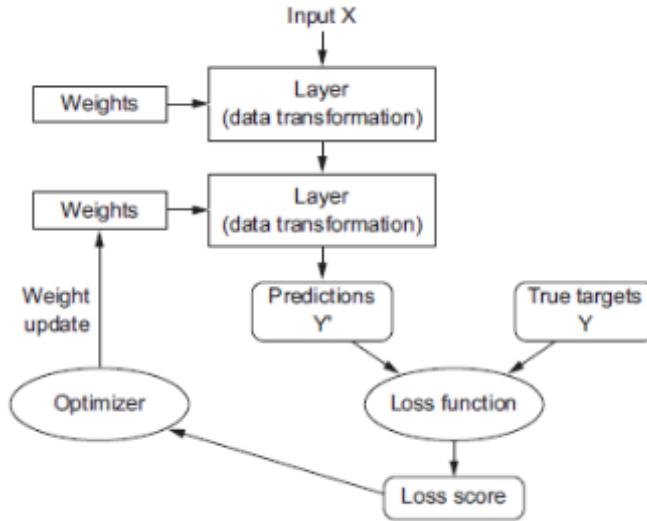


Fonte: Adaptado de Aurélien Geron, 2022.

Essa pontuação é usada como um *feedback* da rede para que ela se ajuste aos pesos em cada iteração e minimize ainda mais o valor da perda. Esse ajuste é o trabalho do otimizador, que implementa o

chamado algoritmo de *Backpropagation*.

Figura 5 – Esquematização completa do funcionamento da rede



Fonte: Adaptado de Aurélien Geron, 2022.

Inicialmente, são atribuídos valores aleatórios aos pesos de modo que a rede apenas implemente uma série randômica de transformações. Naturalmente, é esperado que se tenha uma pontuação de perda bem longe do esperado pois os pesos estão sobreajustados. A cada exemplo que a rede processa uma entrada, no entanto, os pesos serão reajustados ao passo que se tenha uma perda considerável ou estagnação no valor da função perda. Este é o *loop* de treinamento que, repetido um número suficiente de vezes (normalmente dezenas de iterações em milhares de exemplos), produz valores de peso que minimizem nossa função perda. Uma rede com função mínima é aquela para a qual as saídas estão o mais próximo possível dos alvos. Temos, neste último caso, um superajuste dos nossos pesos - existe, porém, um *tradeoff* ao lidarmos com superajuste de parâmetros (LAWRENCE; GILES, 2000). É importante considerar que a minimização ideal da função perda não necessariamente corresponderá à minimização global dos valores, uma vez que esta última pode provocar vieses não desejados na base de dados. Ela será, sim, a minimização mais próxima do ideal, sem que haja perda no balançoamento da rede como um todo.

2.1.4 Estruturas de Dados em Redes Neurais

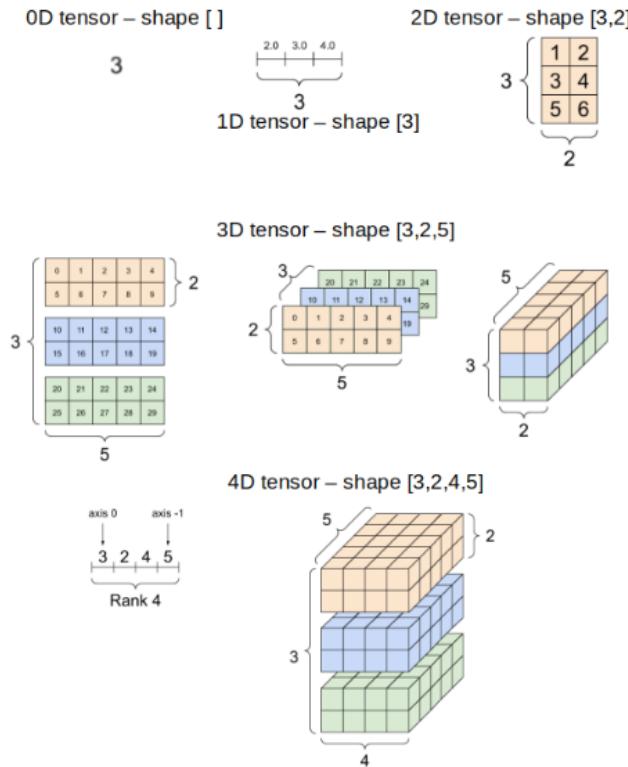
2.1.4.1 Conceito de Tensores

Em *Machine Learning*, para podermos lidar com os dados na sua forma unitária, devemos entender sua estrutura mais simples e básica de representação: os tensores. Em sua essência, um tensor é um *container* de dados - quase sempre numéricos. Tensores são uma generalização de matrizes para um número arbitrário de dimensões (no contexto de tensores, uma dimensão é frequentemente chamada de eixo). A seguir abordaremos, de uma forma geral, como são classificados os tensores computacionalmente:

- **Escalares (Tensores 0D):** um tensor que contém apenas um número é chamado de tensor escalar, podendo ser variáveis inteiras ou do tipo *float*, por exemplo.
- **Vetores (Tensores 1D):** uma matriz de números é chamada de tensor unidimensional. Diz-se que um tensor 1D possui exatamente um eixo. Representações computacionais desse tipo podem ser exemplificadas como uma lista de valores - numéricos ou não.
- **Matrizes (Tensores 2D):** uma matriz de vetores é um tensor bidimensional. Seus dois eixos são referenciados como sendo as linhas e colunas dessa estrutura.
- **Tensores 3D ou de dimensão superior:** se agruparmos as matrizes referidas anteriormente dentro de uma outra matriz, obteremos uma nova dimensão - ou simplesmente teremos um tensor 3D, que pode ser interpretado conceitualmente como um cubo de dados. Ao agruparmos esses novos tensores tridimensionais em uma nova matriz, teremos uma 4^a dimensão de dados (tensores 4D), e assim por diante.

A Figura 7 mostra, de uma forma esquemática, como podemos entender a representação dessa estrutura de dados nas suas formas estendidas de dimensão (TENSORFLOW, 2022).

Figura 6 – Representações de um tensor



Fonte: Adaptado de TensorFlow, 2022.

2.1.4.2 Exemplos Reais de Tensores

Após definidos os tipos de tensores, podemos exemplificar cada categoria com base em sua aplicação no mundo real. Os dados que manipulamos através de técnicas de *DL*, quase sempre são representados como segue:

- **Vector Data:** tensores 2D no formato [*samples, features*], associados a cada tipo de dado ou variável na nossa amostra - similar às linhas e colunas de uma tabela por exemplo.
- **Séries temporais ou dados sequenciais:** tensores 3D no formato [*samples, timestep, features*], associados ao registro temporal de cada um dos indivíduos.
- **Imagens:** tensores 4D no formato [*samples, height, width, channels*], em que temos dimensões associadas ao tamanho (largura e altura) e também ao canal de renderização ou formato de cor, por exemplo.
- **Vídeos:** tensores 5D no formato [*samples, frames, height, width, channels*]. É basicamente uma extensão do tensor de imagem, adotando, porém, um eixo referente aos *frames* (taxa temporal da renderização).

2.1.5 Séries Temporais

Uma série temporal é um conjunto de observações feitas sequencialmente no tempo (BOX et al., 2015). Muitos conjuntos de dados aparecem como séries temporais: uma sequência mensal da quantidade de mercadorias expedidas de uma fábrica, uma série semanal do número de acidentes rodoviários, valores diários de precipitação, observações feitas sobre um processo químico de hora em hora e assim por diante. Exemplos de séries temporais são abundantes e largamente utilizadas em áreas como a economia, negócios e finanças, engenharia, ciências naturais - especialmente geofísica e meteorologia -, ciências sociais, dentre outras.

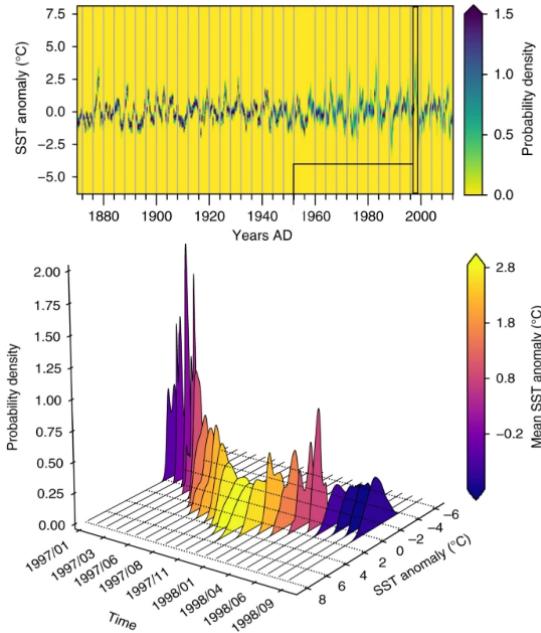
Uma característica intrínseca de uma série temporal é que, normalmente, observações adjacentes são dependentes. A natureza dessa dependência entre observações de uma série temporal é de considerável interesse prático. A análise de séries temporais (GOSWAMI et al., 2018) está preocupada com técnicas que extraiam essa dependência de forma explícita. Isso requer o desenvolvimento de modelos estocásticos e dinâmicos para dados destas séries. Os métodos discutidos neste trabalho são apropriados para sistemas discretos (dados amostrais no tempo), em que as observações do sistema ocorreram em intervalos de tempo equiespaçados (quando não foi possível, porém, verificar o equiespaçamento, aplicaram-se técnicas intrínsecas de cada modelo para garantir a qualidade das observações).

2.2 CONCEITOS DETALHADOS

2.2.1 Decision Trees

Árvores de Decisão (ou *Decision Trees*) são a base para muitos algoritmos clássicos de ML, como *Random Forests*, *Baggings* e *Boosted Decision Trees* (QUINLAN, 1986). Elas foram propostas pela primeira vez por Leo Breiman, um estatístico da Universidade da Califórnia, em Berkley. Sua ideia

Figura 7 – Representação de uma série temporal como uma sequência de funções densidade



Fonte: Adaptado de Goswami, 2018.

era representar os dados como uma árvore onde cada nó interno denota um teste em um atributo (basicamente uma condição), cada ramificação representa um resultado do teste e cada nó folha (nó terminal) contém um rótulo de classe.

2.2.1.1 O que é uma Árvore de Decisão?

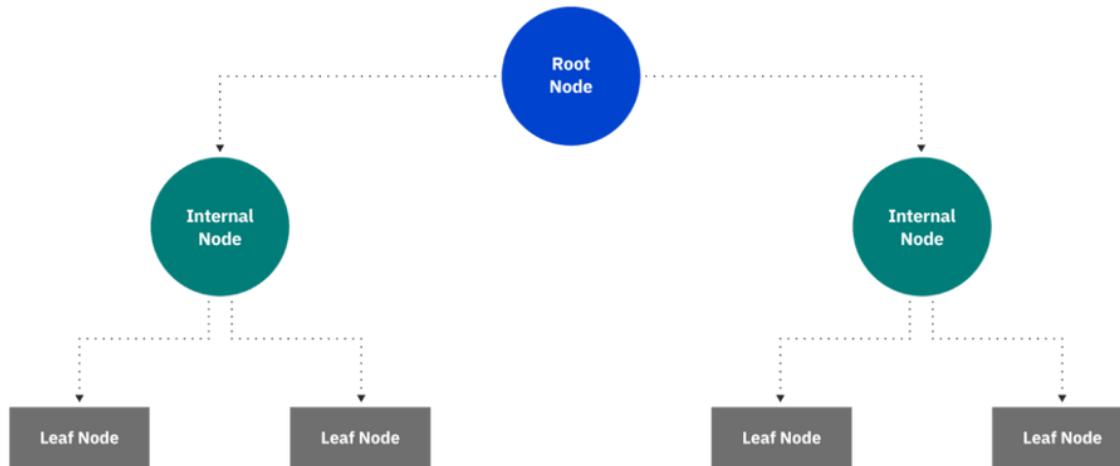
Uma Árvore de Decisão, em sua essência, é um algoritmo de aprendizagem de máquina supervisionado não-paramétrico (IBM, 2022a). Uma forma básica de dividir os algoritmos mais comuns de ML pode ser apresentada como segue:

- **Algoritmos de Aprendizagem Paramétrica:** suposições podem simplificar muito o processo de aprendizado, mas também podem limitar o que pode ser aprendido. Algoritmos que simplificam a função para uma forma conhecida são chamados de algoritmos paramétricos de aprendizado de máquina. No geral, esses algoritmos seguem duas etapas: selecionar o formato da função para ajustar os pontos e aprender os coeficientes da função a partir do treinamento dos dados. Algoritmos comumente utilizados em aprendizagem paramétrica: *Logistic Regression, Linear Discriminant Analysis, Perceptrons, Naive Bayes*, dentre outros.
- **Algoritmos de Aprendizagem Não-Paramétrica:** são algoritmos que não fazem fortes suposições sobre a forma da função de mapeamento. Ao não fazer suposições, eles estão livres para aprender qualquer forma funcional dos dados de treinamento. Os métodos não paramétricos procuram ajustar melhor os dados de treinamento na construção da função de mapeamento, mantendo alguma capacidade de generalizar para dados não vistos. Como tal, eles são capazes de se adequar a um grande número de formas funcionais. Algoritmos comumente utilizados em

aprendizagem não-paramétrica: *k-Nearest Neighbors*, *Decision Trees - CART e C4.5*, *Support Vector Machine (SVM)*, dentre outros.

Os algoritmos de Árvore possuem uma estrutura hierárquica, que consiste em: nó raiz (*root node*), galhos (*branches*), nós internos (*internal nodes*) e nós folha (*leaf nodes*). É válido notar que, da mesma forma que ocorre com as camadas ocultas de uma Rede Neural (que será abordada com mais detalhes na próxima seção), aqui não existe uma quantidade específica de nós internos para formar uma árvore - um único agrupamento sequencial de nó raiz, galho, nó interno e folhas já é a estrutura mínima de uma árvore (IBM, 2022a).

Figura 8 – Representação esquemática de uma Árvore de Decisão

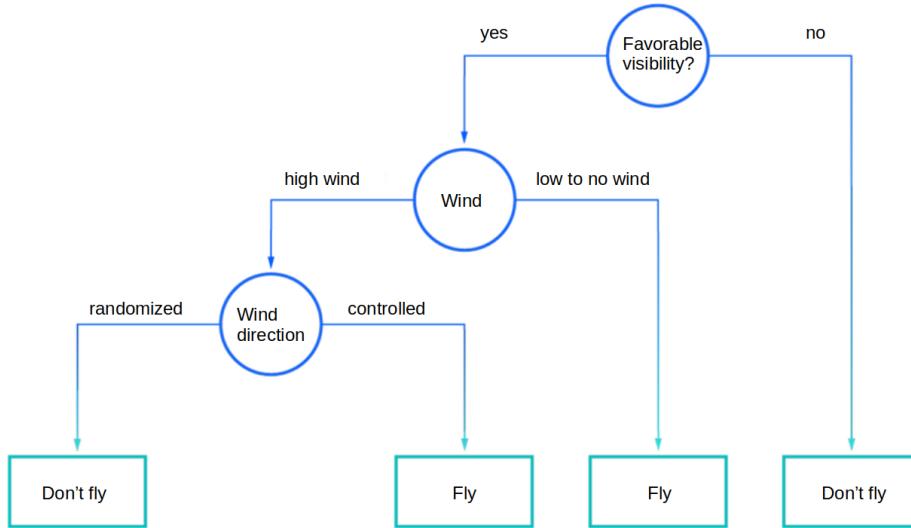


Fonte: Adaptado de IBM, 2022.

O diagrama da Figura 8 mostra que uma árvore de decisão começa com um nó raiz, que não possui ramificações de entrada. As ramificações de saída do nó raiz alimentam os nós internos, também conhecidos como nós de decisão. Com base nas características disponíveis, ambos os tipos de nós realizam avaliações para formar subconjuntos homogêneos, que são denotados por nós-folhas ou nós terminais. Os nós folha representam todos os resultados possíveis dentro do conjunto de dados.

Como exemplo prático, podemos imaginar que estamos avaliando sobre a tomada de decisão entre voar ou não um drone.

Figura 9 – Exemplificação do funcionamento de uma Árvore



Fonte: Produção do próprio autor.

Esse tipo de estrutura de fluxograma (Figura 9) também cria uma representação fácil de digerir da tomada de decisão, permitindo que diferentes grupos em uma organização entendam melhor por que uma decisão foi tomada.

2.2.1.2 Tipos de Árvores

As Árvores de Decisão podem ser classificadas em dois tipos (PAPERSPACEBLOG, 2020), a depender do tipo das variáveis alvo com as quais estamos trabalhando:

- **Categorical Variables Decision Trees:** são árvores que lidam com variáveis categóricas na resposta esperada (*target*). Voltemos ao esquema apresentado na Figura 9: se, por exemplo, pudéssemos classificar a decisão entre voar e não voar em condição favorável e condição desfavorável, respectivamente, teríamos uma saída categórica, uma vez que foi atribuída uma *label* para nossa resposta final.
- **Continuous Variables Decision Trees:** neste caso, as variáveis a serem determinadas como resposta são estritamente numéricas. Novamente no exemplo da Figura 9, caso desejássemos determinar apenas a velocidade ou direção do vento, adotar um algoritmo deste tipo seria factível.

2.2.1.3 Atributos Inerentes aos Nós de uma Árvore

Embora existam várias maneiras de selecionar o melhor atributo em cada nó, dois métodos, ganho de informação (*information gain*) e impureza de Gini (*Gini impurity*), atuam como critérios de divisão populares para modelos de árvore de decisão. Eles ajudam a avaliar a qualidade de cada condição de teste e o quanto bem ela será capaz de classificar as amostras em uma classe.

O ganho de informação descreve a quantidade de informação que é obtida por um atributo. Ele nos diz o quanto importante é o atributo. Como a construção da Árvore de Decisão trata de encontrar o nó de divisão correto que garante alta precisão, o Ganho de Informação trata de encontrar os melhores

nós que retornam o maior ganho de informação. Isso é calculado usando um fator conhecido como Entropia. A entropia define o grau de desorganização de um sistema. Quanto maior esse grau, maior será a entropia.

Isso atua como o fator base na determinação do ganho de informação. Entropia e Ganho de Informação juntos são usados para construir a Árvore de Decisão, e o algoritmo é conhecido como ID3 (QUINLAN, 1986). A entropia pode ser descrita numericamente como segue:

$$E(s) = \sum_{i=1}^c -p_i \log_2 p_i \quad (1)$$

Na equação apresentada, s é o conjunto de dados no qual a entropia é calculada, c representa as classes nesse conjunto p é a probabilidade de sucesso, isto é, a proporção de dados que pertencem à classe c naquele conjunto. Os valores de entropia podem variar entre 0 e 1 - quando a amostra é totalmente homogênea, então a entropia acaba sendo zero, e se a amostra é parcialmente organizada, então a entropia acaba sendo igual a um. Para selecionar o melhor recurso para dividir e encontrar a árvore de decisão ideal, o atributo com a menor quantidade de entropia deve ser usado.

O ganho de informação representa a diferença de entropia antes e depois de uma divisão em um determinado atributo. O atributo com o maior ganho de informação produzirá a melhor divisão, pois está fazendo o melhor trabalho ao classificar os dados de treinamento de acordo com sua classificação na resposta final. O ganho de informação é descrito numericamente como sendo:

$$G(T, x) = E(T) - E(T, x) \quad (2)$$

em que $E(T, x)$ representa a entropia de todos os atributos de entrada, dada por:

$$E(T, x) = \sum_{c \in X} P(c)E(c) \quad (3)$$

sendo X o atributo de entrada, T o atributo de saída, $P(c)$ a probabilidade de pontos estarem presentes em X e $E(c)$ a entropia dos pontos de dados possíveis c .

A impureza de Gini é a probabilidade de classificar incorretamente um ponto aleatório no conjunto de dados se ele for rotulado com base na distribuição de classe desse mesmo conjunto de dados. Similar à entropia, se o conjunto s é puro, ou seja, pertencente a uma classe, então sua impureza é zero. Isso é denotado numericamente como segue:

$$Gini = 1 - \sum_{i=1}^c (P_i)^2 \quad (4)$$

2.2.1.4 Decodificando Hiperparâmetros de uma Árvore

A biblioteca *Scikit-learn* (LEARN, 2022) fornece algumas funcionalidades ou parâmetros que devem ser usados em uma Árvore de Decisão para melhorar a precisão do modelo de acordo com os dados fornecidos. Os principais hiperparâmetros são:

- **Criterion:** parâmetro usado para medir a qualidade da amostragem dos dados. O valor padrão para ele é definido como "*Gini*". Caso desejarmos que a medida seja calculada pelo ganho de entropia, podemos alterá-lo para "*entropy*".
- **Splitter:** Este parâmetro é usado para escolher a divisão em cada nó. Se desejarmos que as sub-árvores tenham a melhor divisão possível, podemos definir esse parâmetro como "*best*".
- **Max Depth:** Este é um parâmetro inteiro através do qual podemos limitar a profundidade da árvore. O valor padrão desse parâmetro é definido como "*None*".
- **Min Samples Split:** Este parâmetro é usado para definir o número mínimo de amostras necessárias para dividir um nó interno.
- **Max Leaf Nodes:** Parâmetro que indica o número máximo de folhas da árvore e pode ser usado também como refinamento do modelo. O valor padrão é definido como "*None*".

2.2.1.5 Random Forests

Random Forests são algoritmos de aprendizagem de máquina comumente usado, registrado por Leo Breiman e Adele Cutler, que combina a saída de várias Árvores de Decisão para alcançar um único resultado (IBM, 2022b).

Embora as Árvores de Decisão sejam algoritmos comuns de aprendizado supervisionado, elas podem estar sujeitas a problemas, como viéses (*bias*) e superajuste (*overfitting*). No entanto, quando várias Árvores formam um conjunto no algoritmo da *Random Forest*, elas acabam por prever resultados mais precisos, sobretudo quando as Árvores isoladas não estão correlacionadas umas com as outras.

2.2.1.5.1 Métodos Ensemble

Métodos de aprendizagem por *ensemble* são compostos de um conjunto de classificadores - por exemplo, árvores de decisão - e suas previsões são agregadas para identificar o resultado mais recorrente. Os métodos de ensemble mais conhecidos são *bagging*, também conhecido como agregação por *bootstrap*, e *boosting*. No método de *bagging* (BREIMAN, 1996), uma amostra aleatória de dados em um conjunto de treinamento é selecionada com substituição, o que significa que pontos individuais desses dados podem ser escolhidos mais de uma vez.

Depois que várias amostras de dados são geradas, esses modelos são treinados independentemente e dependendo do tipo de tarefa, ou seja, regressão ou classificação - a média ou a maioria dessas previsões produz uma estimativa mais precisa. Essa abordagem é comumente usada para reduzir a variação dentro de um conjunto de dados ruidoso.

2.2.1.5.2 O Algoritmo Random Forest

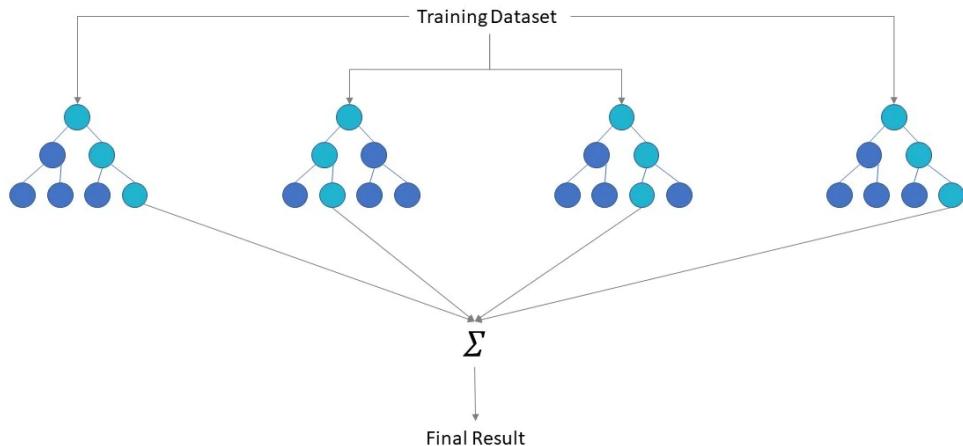
O algoritmo de floresta aleatória é uma extensão do método *bagging*, pois utiliza *bagging* e aleatoriedade de recursos para criar uma floresta não correlacionada de árvores de decisão. Aleatoriedade das *features*, também conhecida como *features bagging* ou "método do subespaço aleatório" gera um

subconjunto aleatório dessas *features*, o que garante baixa correlação entre as árvores de decisão. Essa é uma diferença fundamental entre Árvores de Decisão e *Random Forests* (BREIMAN, 2001). Enquanto as Árvores consideram todas as possíveis divisões de recursos, as *Random Forests* selecionam apenas um subconjunto desses recursos. Ao contabilizar toda a variabilidade potencial nos dados, podemos reduzir o risco de superajuste, viés e variação geral, resultando em previsões mais precisas.

Este algoritmo possui três hiperparâmetros principais, que precisam ser definidos antes do treinamento. Isso inclui o tamanho do nó (*node size*), o número de árvores (*number of trees*) e o número de recursos amostrados (*number of features sampled*). O algoritmo de *Random Forest* é composto por uma coleção de árvores de decisão, e cada árvore do conjunto é composta por uma amostra de dados extraída de um conjunto de treinamento com reposição, chamada de amostra por *bootstrap*. Dessa amostra de treinamento, um terço é reservado como dados de teste, conhecido como amostra "*out-of-bag*" (OOB).

Outra instância de aleatoriedade é injetada por meio do empacotamento (*bagging*) dos recursos, adicionando mais diversidade ao conjunto de dados e reduzindo a correlação entre as árvores de decisão. Dependendo do tipo de problema, a determinação da previsão irá variar. Para uma tarefa de regressão, será calculada a média das árvores de decisão individuais e, para uma tarefa de classificação, um voto majoritário, ou seja, a variável categórica mais frequente - produzirá a classe prevista. Por fim, a amostra OOB é então usada para validação cruzada, finalizando essa previsão.

Figura 10 – Diagrama de um Algoritmo Random Forest



Fonte: Adaptado de IBM, 2022.

2.2.1.6 Boosting dos Modelos

Boosting é um método usado em aprendizado de máquina para reduzir erros na análise preditiva de dados. Um único modelo de aprendizado de máquina pode cometer erros de previsão, dependendo da precisão do conjunto de dados de treinamento. O *Boosting* tenta superar esse problema treinando vários modelos sequencialmente para melhorar a precisão do sistema geral (SERVICES, 2022).

2.2.1.6.1 Por que o Boosting é Importante?

O *Boosting* melhora a precisão preditiva e o desempenho dos modelos convertendo vários aprendizes fracos em um único modelo de aprendizagem forte (SERVICES, 2022). Os modelos de ML podem ser do tipo fraco ou forte, como definido a seguir:

- **Weak Learners:** também tidos como "aprendizes fracos", têm baixa precisão de previsão, semelhante à uma estimativa aleatória da resposta. Eles são propensos ao *overfitting*, ou seja, não podem classificar dados que variam muito de seu conjunto de dados original.
- **Strong Learners:** também tidos como "aprendizes fortes", têm maior precisão de previsão. O *Boosting* converte um sistema de aprendizes fracos em um único sistema de aprendizado forte.

2.2.1.6.2 Processo de Treinamento por Boosting

O método de treinamento varia dependendo do tipo de processo de reforço chamado de *Boosting*. No entanto, um algoritmo segue as seguintes etapas gerais para treinar o modelo:

1. O algoritmo de reforço atribui peso igual a cada amostra de dados. Ele alimenta os dados para o primeiro modelo, chamado de algoritmo base. O algoritmo base faz previsões para cada amostra de dados.
2. O algoritmo avalia as previsões do modelo e aumenta o peso das amostras com um erro mais significativo. Ele também atribui um peso com base no desempenho do modelo. Um modelo que produz boas previsões terá uma grande influência sobre a decisão final.
3. O algoritmo passa os dados ponderados para a próxima árvore de decisão.
4. O algoritmo repete as etapas 2 e 3 até que as instâncias de erros de treinamento estejam abaixo de um determinado limite (*threshold*).

2.2.1.6.3 Tipos de Boosting

A seguir estão os três principais tipos de *boosting*:

- **Adaptive Boosting (AdaBoost):** o AdaBoost (ou *Boosting* Adaptativo) foi um dos primeiros modelos de reforço desenvolvidos. Ele se adapta e tenta se autocorrigir em cada iteração do processo de reforço - inicialmente dá o mesmo peso para cada conjunto de dados. Em seguida, ajusta automaticamente os pesos dos pontos de dados após cada árvore de decisão. Esse processo acaba atribuindo mais peso aos itens classificados incorretamente para corrigí-los na próxima iteração. Esse processo é repetido até que o erro residual, ou a diferença entre os valores reais e previstos, esteja abaixo de um limite aceitável. Podemos utilizar o AdaBoost com muitos preditores e normalmente ele não é tão sensível quanto outros algoritmos de *boosting*. Essa abordagem, no entanto, não funciona bem quando existe uma correlação entre as *features* ou alta dimensionalidade na nossa base de dados. De um modo geral, o AdaBoost é um tipo adequado de *boosting* algoritmo problemas de classificação (SERVICES, 2022).

- **Gradient Boosting (GB):** o *Gradient Boosting* é similar ao AdaBoost, pois também é uma técnica de treinamento sequencial. A diferença entre o AdaBoost e o GB é que o GB não dá mais peso aos itens classificados incorretamente. Ao invés disso, o GB optimiza sua função de perda gerando aprendizes de base dispostos sequencialmente para que o aprendiz atual seja sempre mais eficaz do que o anterior. Esse método tenta gerar resultados precisos inicialmente, em vez de corrigir erros ao longo do processo, como o AdaBoost. Por esse motivo, modelos baseados em GB podem levar a resultados mais precisos. O *Gradient Boosting* pode ajudar com problemas baseados em classificação e regressão (SERVICES, 2022).
- **Extreme Gradient Boosting (XGBoost):** o XGBoost melhora o GB para ter uma velocidade computacional escalável de várias maneiras. O XGBoost usa vários núcleos na CPU para que o aprendizado ocorra em paralelo durante o treinamento. É um algoritmo de *boosting* que pode lidar com conjuntos de dados extensos, o que o torna atraente para aplicações de *Big Data*. Os principais recursos do XGBoost são paralelização, computação distribuída, otimização de cache e processamento fora do núcleo (SERVICES, 2022).

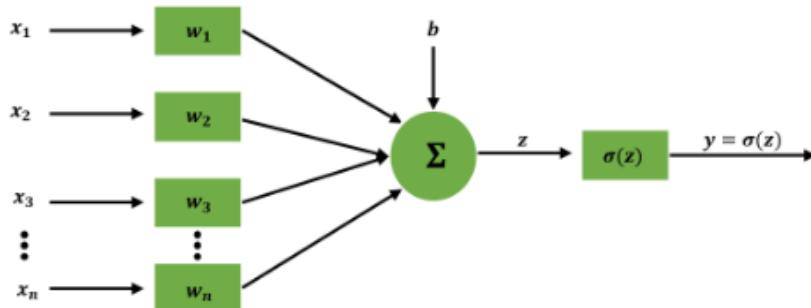
2.2.2 Redes Neurais Artificiais

Redes neurais artificiais (FENG et al., 2019) são compostas por unidades de processamento cujos sinais de entrada do neurônio são representados pelo vetor $X = [x_1, x_2, x_3, \dots, x_n]$ e são multiplicados pelos respectivos pesos sinápticos definidos por $W = [w_1, w_2, w_3, \dots, w_n]$.

2.2.2.1 Perceptrons

O Perceptron é a unidade elementar de uma rede neural. Ele recebe os sinais de entrada e retorna um valor único na saída, que pode ser a saída global da rede ou pode também servir como sinal de entrada para os demais perceptrons (DU; SWAMY, 2013). A Figura 11 apresenta de forma esquemática a estrutura de um Perceptron.

Figura 11 – Rede neural elementar (Perceptron)



Fonte: Produção do próprio autor.

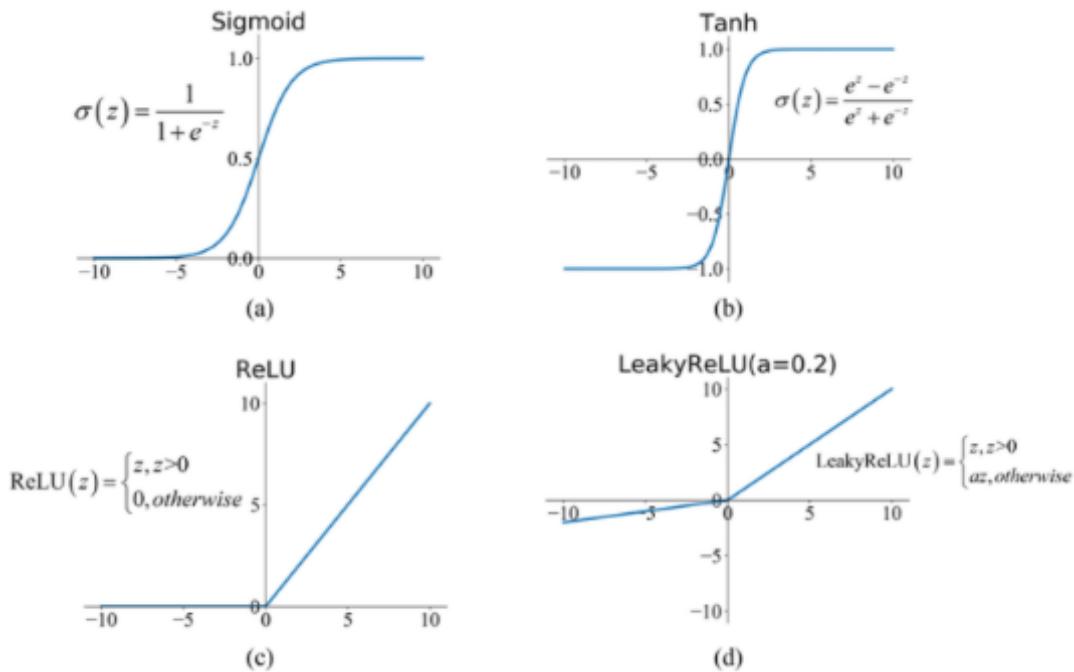
Os sinais de entrada x , após serem multiplicados pelos pesos w , são somados com b - *bias* ou viés, que representa uma parcela adicional ao grau de liberdade do nosso sistema -, produzindo o potencial

de ativação z . Deste modo, teremos a saída definida como sendo:

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases} \quad (5)$$

O potencial de ativação z é então passado para uma função matemática não-linear chamada de função de ativação ou *Sigmoid*, $\sigma(z)$, responsável por limitar a saída do neurônio a um certo intervalo como descrito na equação anterior (FENG et al., 2019). Outras funções de ativação (SHARMA; SHARMA; ATHAIYA, 2017), além da *Sigmoid*, também podem ser utilizadas no processo, como mostra a Figura 12

Figura 12 – Funções de ativação comumente utilizadas



Fonte: Feng, 2019.

2.2.2.2 Layers

Layers ou camadas são segmentações estruturais de uma rede neural (DŽAKULA et al., 2019). Algumas destas segmentações já foram citadas, de forma sucinta, nas seções anteriores, porém faremos uma descrição mais detalhada de cada uma delas a seguir.

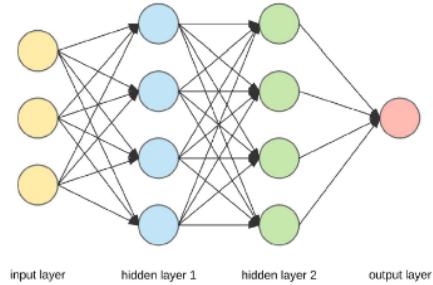
1. **Input Layer:** também chamada de camada de entrada, é a primeira camada de uma rede neural. À esta camada associamos o domínio de valores do nosso problema de forma diretamente proporcional - ou seja, quanto maior a quantidade de variáveis independentes do problema, maior será a quantidade de neurônios associados à camada (em quantidades idênticas). É nessa etapa onde os padrões são apresentados à rede.

2. **Hidden Layers:** também chamadas de camadas ocultas, são dispostas em sequência após termos a saída da camada anterior. No geral não existe uma regra definida para a quantidade n de camadas ocultas do problema, uma vez que são usadas apenas para refinar os resultados globais na saída da rede. No entanto, existe um *tradeoff* entre essa quantidade de camadas e a resposta esperada, pois o aumento excessivo de n pode gerar viés estatístico e fazer com que haja divergência de valores na saída (UZAIR; JAMIL, 2020).
3. **Output Layer:** também chamada de camada de saída, à ela associamos o domínio das variáveis dependentes do nosso problema. É nessa etapa que temos o resultado final do processo.

2.2.2.3 Multilayer Perceptron (MLP)

Perceptron Multicamadas (ou *Multilayer Perceptrons*) são redes estruturadas a partir da combinação de Perceptrons isolados (POPESCU et al., 2009). Neste caso, o conceito da saída (*output*) muda, pois cada camada pega como *input* o resultado da camada anterior - isso é a extensão prática do que definimos anteriormente como *Deep Learning*. A Figura 13 apresenta uma estrutura genérica de MLP:

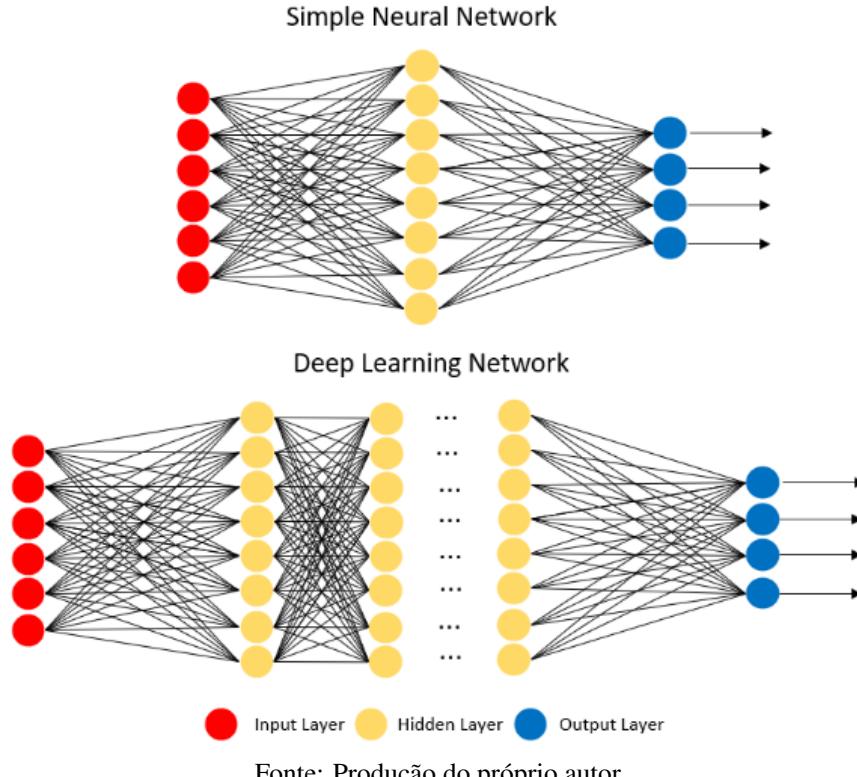
Figura 13 – Multilayer Perceptron



Fonte: Produção do próprio autor.

Podemos ainda estender essa ideia retomando os conceitos apresentados no início deste trabalho sobre Deep Learning. A Figura 14 apresenta, de forma geral, a diferença entre uma Rede Neural simples e uma Rede Neural Profunda. Basicamente o conceito do DL começa a tomar notoriedade quando temos a sucessão de diversas camadas ocultas.

Figura 14 – Diferença entre uma rede neural simples e uma rede neural profunda



Fonte: Produção do próprio autor.

2.2.2.4 Treinamento Supervisionado

O treinamento supervisionado de uma rede neural consiste em ajustar os pesos e os valores do *bias* ou viés. Quando um padrão é inicialmente apresentado à rede, ela produz uma saída. Após medir a distância entre a resposta atual e a desejada, são realizados os ajustes apropriados nos pesos das conexões de modo a reduzir essa distância. Esse procedimento é conhecido como "Regra Delta"(ALSMADI et al., 2009). Desta forma, teremos o seguinte esquema de treinamento:

- Iniciar todas as conexões com pesos aleatórios;
- Calcular a resposta da rede;
- Calcular o erro;
- Se o erro E não for satisfatoriamente pequeno ($E > e$), então atualizar os pesos:

$$w_{i,j,new}^k = \left(w_{i,j,previous}^k \right) - \alpha \left(\frac{dE}{dw_{i,j}^k} \right) \quad (6)$$

- Repetir até que o erro E seja satisfatoriamente pequeno ($E = e$);

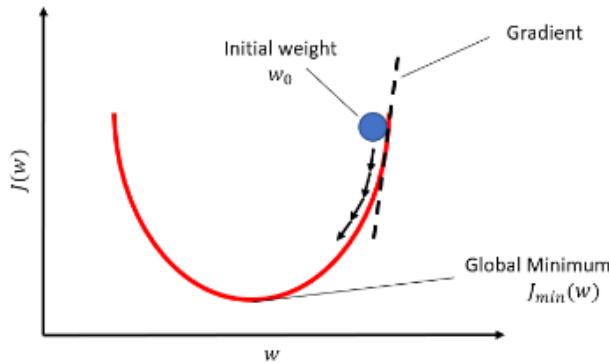
A ideia do algoritmo *Backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986) é, com base no cálculo do erro ocorrido na camada de saída da rede neural, recalcular o valor dos pesos do vetor

W da última camada e assim proceder para as camadas anteriores, no sentido reverso - isto é, atualizar todos os pesos w das camadas a partir da última até atingir a camada de entrada.

Os erros são também chamados de Função Custo, que de uma forma geral é a diferença entre o valor real do conjunto de dados e o valor pertencente à curva de regressão, obtidos na saída da rede. Um exemplo de função custo é o Erro Quadrático Médio ou *Mean Squared Error (MSE)*, que segue a equação:

$$MSE = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (7)$$

Figura 15 – Algoritmo *Gradient-Descent*



Fonte: Produção do próprio autor.

O cálculo dos melhores pesos de uma rede é um processo custoso e requer um método de otimização. Neste aspecto, surge a ideia do *Gradient-Descent* (Figura 15), no qual teremos:

- Chutar um valor para w ;
- Calcular o vetor gradiente de J em w ;
- Atualizar w na direção oposta à do vetor gradiente (que indica a direção e o sentido de maior crescimento da função);

2.2.2.5 Função Sigmoid (σ)

Antes de entendermos como uma função *Sigmoid* funciona e sua importância para a rede neural, devemos retomar alguns conceitos de regressão linear (CHEIN, 2019), que no geral pode ser modelada como segue:

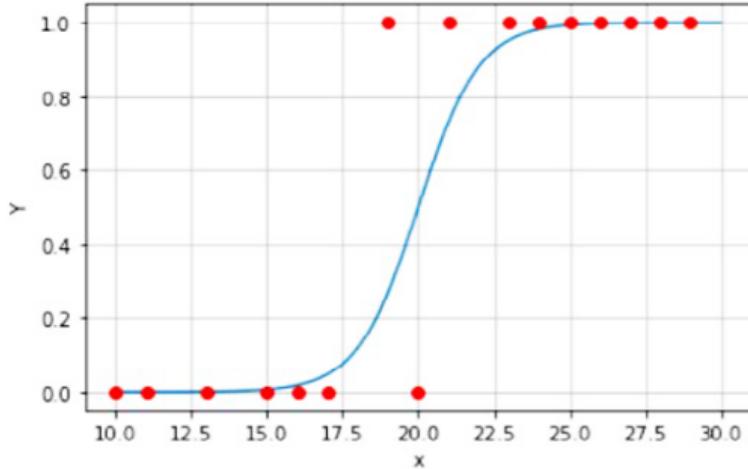
$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_n X_n = \beta^T X \quad (8)$$

em que y é a variável dependente (também chamada de "target"), X é a matriz dos regressores (variáveis independentes) e β^T é o vetor transposto dos coeficientes da regressão. Se considerarmos a saída y como um valor binário (ou inteiro) no intervalo $[0, 1]$, podemos usar o modelo de regressão logística para limitar nossos dados.

É nesta etapa que surge o conceito da função *Sigmoid*, que pode ser escrita como sendo:

$$\sigma(z) = \frac{1}{(1 + e^{-z})} \quad (9)$$

Figura 16 – Ajuste de pontos através de uma função Sigmoid



Fonte: Produção do próprio autor.

A probabilidade bayesiana de resposta no intervalo binário é definida a seguir:

$$P = \begin{cases} p(y = 1|x) = \frac{1}{1 + e^{-\beta^T X}}, & \text{if } y = 1 \\ p(y = 0|x) = 1 - P = \frac{1}{1 + e^{\beta^T X}}, & \text{if } y = 0 \end{cases} \quad (10)$$

A equação ainda porde ser reescrita de forma compacta usando o modelo de Bernoulli:

$$P(y = K) = p^K (1 - p)^{1-K} \quad (11)$$

em que:

$$\begin{cases} K = 1, p(y = 1) = p \\ K = 0, p(y = 0) = 1 - p \end{cases} \quad (12)$$

Para encontramos o valor de β usamos a função de verossimilhança (ou *Likelihood*):

$$\mathcal{L}(\beta) = \prod_{i=1}^n p^{y_i} (1 - p)^{1-y_i} \quad (13)$$

O principal objetivo é maximizar a função para obter o melhor ajuste possível da curva de *Sigmoid* (Figura 16). Para isso, devemos aplicar uma transformação *log* nos pontos, obtendo a partir da equação anterior sua derivação parcial:

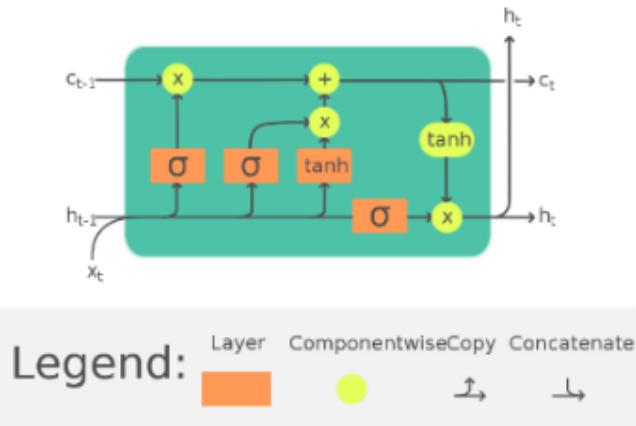
$$\frac{\partial \log \mathcal{L}(\beta)}{\partial \beta} = \sum_{i=1}^n \left[X_i \left(y_i - \frac{e^{\beta^T x_i}}{1 + e^{\beta^T}} \right) \right] = 0 \quad (14)$$

Após definido o problema, devemos determinar os valores de β através de métodos numéricos (EPPERSON, 2021), obtendo assim a curva que melhor ajuste os pontos e que minimize o erro associado a eles.

2.2.2.6 Long Short-Term Memory (LSTM)

Redes LSTM (NGUYEN et al., 2021) são definidas como sendo um tipo específico de Redes Neurais Recorrentes (RNN) que retém as dependências de longo-termo entre os dados num dado tempo através de alguns *timesteps* anteriores. Seu formato é baseado em módulos repetidos de redes neurais, o que inclui *control gates*, *forget gates*, *input gates* e *output gates*.

Figura 17 – Representação básica de uma célula LSTM



Fonte: Adaptado de Chevalier, 2020.

As camadas *Sigmoid* produzem números no intervalo $[0, 1]$, representando uma parte da informação de entrada que deve ser deixada passar.

Assim como acontece com o uso de um RNN para dados de séries temporais, o LSTM lê uma sequência de vetores de entrada $X = x_1, x_2, \dots, x_t, \dots$, em que $x_t \in R_m$ representa um vetor m -dimensional gerado a partir de m variáveis num dado instante de tempo t (ZHAO et al., 2018). Consideremos o cenário em que várias dessas séries temporais podem ser obtidas tomando uma janela (janela ou window é uma função genérica que extrai o subconjunto do objeto x observado entre os tempos de início e término; se uma frequência for especificada, a série é então reamostrada na nova frequência) sobre uma série temporal maior. Mesmo o LSTM podendo trabalhar com qualquer dado de série temporal, deve-se considerar que seu desempenho nem sempre é o mesmo, pois pode variar dependendo da entrada.

Dada a nova informação x_t no estado t , o módulo LSTM funciona da seguinte forma: num primeiro momento ele decide quais informações antigas devem ser esquecidas produzindo um número dentro do intervalo *Sigmoid* $[0, 1]$, digamos f_t como sendo

$$f_t = \sigma_1(W_f[h_{t-1}, x_t] + b_f) \quad (15)$$

onde $h_{(t-1)}$ é a saída no estado $(t - 1)$, W_f e b_f são a matriz de peso e o *bias* (*bias* ou viés é a simplificação das suposições feitas pelo modelo para tornar a função de destino mais fácil de aproximar; em outras palavras, é a diferença entre o valor esperado de um ajuste do modelo - ou média das previsões - e o valor real) do gate de esquecimento. Então, x_t é processado antes de ser armazenado no estado da célula. O valor é determinado na porta de entrada juntamente com um vetor de valores candidatos \tilde{C}_t gerado pela camada *tanh* ao mesmo tempo para ser atualizado no novo estado da célula C_t , em que

$$i_t = \sigma_2(W_i[h_{t-1,x_t} + b_i]) \quad (16)$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1,x_t} + b_c]) \quad (17)$$

e

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (18)$$

onde (W_i, b_i) e (W_c, b_c) são as matrizes de peso e os vieses da porta de entrada e do estado da célula de memória, respectivamente. Finalmente, a porta de saída, que é definida por

$$o_t = \sigma_3(W_o[h_{t-1,x_t} + b_o]) \quad (19)$$

$$h_t = o_t \cdot \tanh(C_t) \quad (20)$$

onde W_o e b_o são a matriz de peso e a polarização da porta de saída, determina além do estado da célula que está sendo emitido. A Figura 17, apresenta uma ilustração da estrutura e do princípio de funcionamento de um módulo LTSM típico. Nesta figura, o estado da célula percorre toda a cadeia, mantendo as informações sequenciais em um estado interno e permitindo que o LSTM persista o conhecimento acumulado de etapas de tempo subsequentes (NGUYEN et al., 2021).

2.2.2.7 LSTM Autoencoder

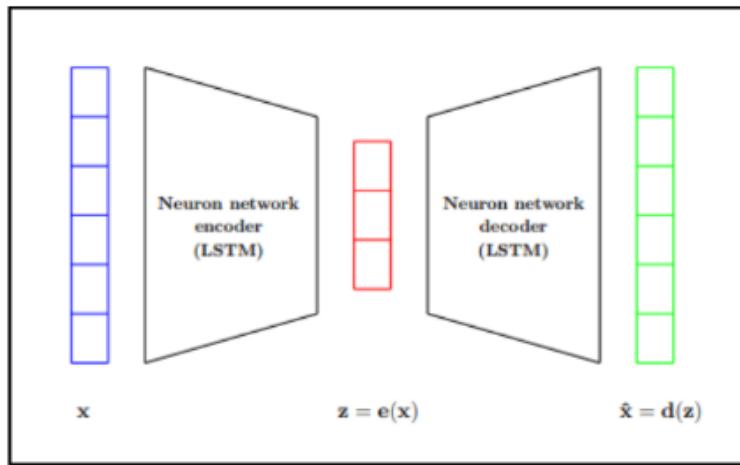
Autoencoder (NGUYEN et al., 2021) é uma rede neural não supervisionada que visa aprender o melhor esquema de codificação-decodificação dos dados. Em geral, consiste em uma camada de entrada, uma camada de saída, uma rede neural codificadora, uma rede neural decodificadora e um espaço latente. Quando os dados são alimentados na rede, o codificador os compacta no espaço latente, enquanto o decodificador descompacta a representação codificada na camada de saída. A saída "autoencodada" é então comparada com os dados iniciais e o erro é verificado através da arquitetura para atualizar os pesos da rede. Em particular, dada a entrada $x \in R_m$, o codificador comprime x para obter uma representação codificada $z = e(x) \in R_n$

O autoencoder é treinado minimizando o erro de reconstrução:

$$L = \frac{1}{2} \sum_{i=1}^n \|x_i - \hat{x}_i\|^2 \quad (21)$$

O objetivo principal do autoencoder não é simplesmente copiar a entrada para a saída. Ao restringir o espaço latente a ter uma dimensão menor que a entrada, ou seja, $n < m$, o autoencoder é forçado a aprender as características mais salientes dos dados de treinamento. Em outras palavras, um recurso importante no projeto do autoencoder é que ele reduz as dimensões dos dados, mantendo as principais informações da estrutura de dados.

Figura 18 – Estrutura básica de uma operação Autoencoder com LSTM



Fonte: Adaptado de Nguyen, 2020.

Vários tipos de autoencoders têm sido propostos na literatura, como o autoencoder vanilla, o autoencoder convolucional, o autoencoder regularizado e o autoencoder LSTM. Dentre esses tipos, o autoencoder LSTM refere-se ao autoencoder que tanto o codificador quanto o decodificador fazem parte da rede LSTM. A capacidade do LSTM de aprender padrões em dados em longas sequências os torna adequados para previsão de séries temporais ou detecção de anomalias. Ou seja, o uso da célula LSTM é para capturar dependências temporais em dados multivariados. Um modelo codificador-decodificador aprendido usando apenas as sequências normais pode ser usado para detectar anomalias em multivariadas séries temporais. O codificador-decodificador que só viu instâncias normais durante o treinamento e aprendeu a reconstruir-las, quando alimentado com uma sequência anômala pode não ser bem reconstruído - o que levaria a erros maiores (NGUYEN et al., 2021). Isso tem um significado prático, pois nem sempre dados anômalos estão disponíveis ou é impossível cobrir todos os tipos desses dados. Por este motivo, durante a fase de treino do modelo tomaremos o devido cuidado em extrair anomalias muito bem definidas (e reconhecidas) dentro de cada voo ou simulação.

2.2.2.8 Hiperparâmetros de uma Rede Neural

Assim como discutido na seção anterior, os hiperparâmetros são utilizados para controlar o processo de aprendizado de um modelo. São estes os parâmetros que controlam o processo e os demais

parâmetros - e por isso estão em um nível acima -, não sofrendo alterações durante o processo de aprendizagem dinâmica (como ocorre com os pesos e viéses) (CHOLLET, 2021). Os hiperparâmetros de uma rede neural podem ser entendidos como segue:

- **Learning Rate:** também denotado de taxa de aprendizagem, é o ajuste que determina o passo de cada iteração (ou *step*) durante o processo de minimização de uma função custo. Entretanto existe um balanceamento entre custo computacional e essa taxa, pois passos muito pequenos podem levar iterações pouco viáveis para modelos em produção devido ao tempo decorrido até a convergência, enquanto passos maiores - apesar de serem menos custosos computacionalmente e também mais rápidos - podem divergir completamente e não conseguirem encontrar um mínimo global da função.
- **Número de Camadas:** refere-se, necessariamente, à quantidade de camadas ocultas da rede, uma vez que as camadas de entrada e saída são inerentes a qualquer modelo de rede neural. Como abordado anteriormente, não existe uma regra definida para se adotar um número fixo de camadas, uma vez que elas servem para refinar os pesos e viéses encontrados. Entretanto, aqui também existe um custo computacional relativo, uma vez que a adição de camadas exigirá maior poder de processamento.
- **Número de Neurônios por Camada:** para qualquer rede, o número de neurônios das camadas de entrada e saída são fixos. Entretanto, podemos estimar a quantidade de neurônios das camadas ocultas (BENGIO, 2012) seguindo a equação:

$$N_h = \frac{N_s}{\alpha(N_i + N_o)} \quad (22)$$

em que N_h é a quantidade de neurônios da camada oculta, N_i é a quantidade de neurônios na camada de entrada, N_o é a quantidade de neurônios na camada de saída e α é um parâmetro de correção arbitrário entre 5 e 10.

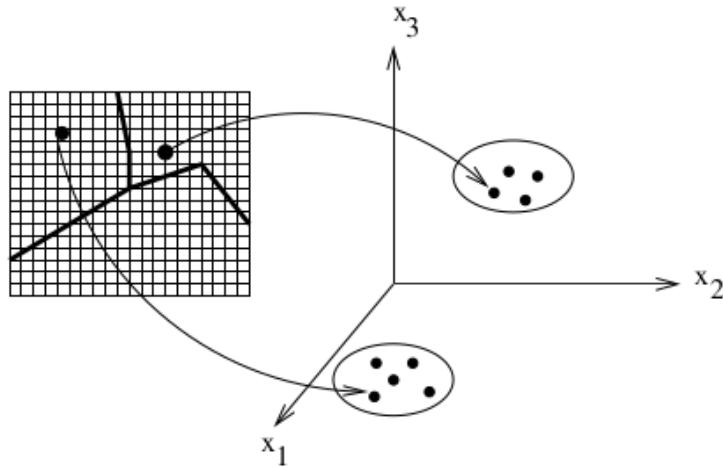
- **Batch Size:** O tamanho de *batch* é o hiperparâmetro que define o número de amostras a serem trabalhadas antes de atualizar os parâmetros do modelo interno. Podem ser do tipo: "*Batch Gradient Descent*" (em que o tamanho de *batch* é igual ao tamanho do conjunto de dados de treino), "*Stochastic Gradient Descent*" (tamanho de *batch* igual a 1), ou "*Mini-Batch Gradient Descent*" (sendo o tamanho de *batch* entre 1 e o tamanho do conjunto de treino).
- **Número de Epochs:** época é o termo referente ao número de iterações ou passagens durante o processo de treinamento do modelo. Para este caso também não existe um valor previamente idealizado, sendo necessário, portanto, utilizá-lo como parâmetro de refinamento do modelo com o auxílio da curva de convergência (análise gráfica e retreino, caso necessário).

2.2.3 Clusterização dos Dados

Clusterização, ou *Clustering* (JAIN; MURTY; FLYNN, 1999), é a classificação não supervisionada de padrões (observações, itens de dados, ou vetores de recursos) em grupos (*clusters*).

Os procedimentos de análise de dados podem ser categorizados como sendo exploratórios ou confirmatórios - com base na disponibilidade de modelos apropriados para a fonte dos dados -, mas um elemento-chave em ambos os tipos de procedimentos (seja por formação de hipótese para uma tomada de decisão) é o agrupamento ou classificação de medidas com base na qualidade de ajuste do modelo ou nos agrupamentos naturalmente revelados através análise exploratória. A análise de cluster é a organização de uma coleção de padrões (geralmente representados como um vetor de medidas ou um ponto em um espaço multidimensional) em clusters com base na similaridade entre eles. Intuitivamente, os padrões dentro de um cluster válido são mais semelhantes entre si do que quando comparados a um outro agrupamento (como por exemplo na Figura 19, que apresenta as medidas de uma imagem e suas posições, transformadas em *features*. Os agrupamentos no espaço de atributos correspondem aos segmentos desta imagem).

Figura 19 – Representação de *features* através de uma clusterização



Fonte: Adaptado de Jain, 1999.

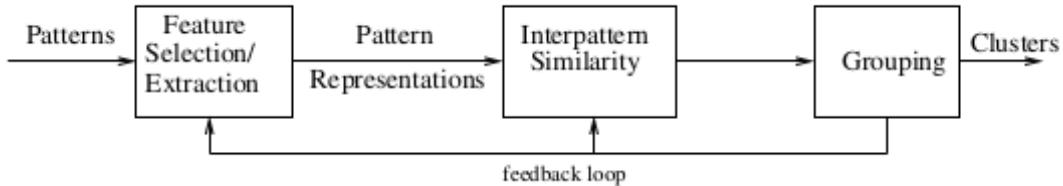
2.2.3.1 Componentes de uma Rotina de Clusterização

Uma rotina de clusterização típica (JAIN; DUBES, 1988), segue as etapas dispostas a seguir:

1. Representação de padrões (opcionalmente incluindo extração de *features* e/ou seleção).
2. Definição de uma medida de proximidade padrão apropriada para o domínio dos dados.
3. Clusterização ou agrupamento seletivo.
4. Abstração dos dados (se necessário).
5. Avaliação de saída (se necessário).

A Figura 20 descreve um sequenciamento típico para as três primeiras dessas etapas, incluindo um caminho de feedback onde o agrupamento na saída do processo pode afetar a extração das *features* (atributos) e o cálculo das similaridades entre elas.

Figura 20 – Estágios de um processo de clusterização



Fonte: Adaptado de Jain, 1999.

Pattern representation, ou representação de padrões, se refere ao número de classes, ao número de padrões disponíveis, e ao número, tipo e escala dos atributos disponíveis para o algoritmo de clusterização. *Feature selection*, ou seleção de atributos, é o processo de identificação dos subconjuntos mais relevantes a partir da base de dados bruta a serem utilizados nos agrupamentos. *Feature extraction*, ou extração de atributos, é a utilização de uma ou mais transformações nos atributos originais para produzir novas *features*.

Pattern proximity, ou proximidade dos padrões, é usualmente medido por uma função de distância definida nos pares de padrões. Uma medida de distância simples, como a Distância Euclideana, pode ser usada para refletir a similaridade entre dois padrões, enquanto outras medidas de similaridade podem ser usadas para caracterizar a semelhança conceitual entre dois padrões. *Grouping*, ou etapa de agrupamento, pode ser realizada de várias maneiras. Os *clusters* (ou agrupamentos) na saída podem ser do tipo *hard* (partições dos dados brutos em subgrupos) ou do tipo *fuzzy* (em que cada padrão tem um grau variável de associação em cada um dos os clusters de saída).

2.2.3.2 Definições e Notação

Alguns termos e notações serão apresentados. Usaremos eles para fazer uma descrição sucinta da modelagem numérica dos principais algoritmos de clusterização (JAIN; DUBES, 1988). São eles:

- Um padrão (ou vetor de *features*, observações ou *datum*), X , é um item de dado único utilizado pelo algoritmo. Tipicamente consiste em um vetor de medidas d , dado por: $X = (x_1, x_2, \dots, x_d)$.
- Os componentes escalares individuais, x_i , de um padrão X , são chamados atributos (ou *features*).
- d é a dimensionalidade do padrão ou do domínio de X .
- Um conjunto de padrões é denotado por $\mathcal{X}' = \{X_1, X_2, \dots, X_n\}$. O i -ésimo padrão em \mathcal{X}' é denotado por $X_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,n}\}$. Em muitos casos, um conjunto de padrões, para ser clusterizado, é visto como uma matriz de padrões na dimensão $n \times d$.
- Uma classe, em resumo, se refere a um estado inerente ao processo de geração de padrões em alguns casos. Mais concretamente, uma classe pode ser visualizada como um conjunto de padrões cuja distribuição no espaço das *features* é descrita por uma função densidade de probabilidade.

- Técnicas de clusterização do tipo *Hard* atribuem um rótulo de classe l_i para cada padrão X_i , identificando suas classes. O conjunto de todos os rótulos (ou *labels*) para um conjunto de padrões \mathcal{X}' é $\mathcal{L} = \{l_1, l_2, \dots, l_n\}$, com $l_i \in \{1, 2, \dots, k\}$, em que k é o número de *clusters*.
- Procedimentos de clusterizações do tipo *Fuzzy* atribuem para cada padrão de entrada X_i um grau fracionário de adesão f_{ij} em cada cluster de saída j .
- A medida de distância (uma especialização da medida de proximidade) é a métrica (ou quasi-métrica) do espaço de atributos utilizada para quantificar a similaridade dos padrões.

2.2.3.3 Medidas de Similaridade

Como a similaridade é fundamental para a definição de um *cluster*, medir tal similaridade entre dois padrões obtidos a partir do mesmo espaço de atributos é essencial para a maioria dos processos de agrupamento. Devido à variabilidade de tipos de atributos, a medida de distância deve ser escolhida com certo cuidado. É mais comum determinar a não-similaridade entre dois padrões usando a medida de distância definida no espaço de atributos (JAIN; DUBES, 1988).

A métrica mais popular para atributos contínuos é a Distância Euclidiana. Ela pode ser interpretada numericamente como sendo:

$$d_2(X_i, X_j) = \left(\sum_{k=1}^d (x_{i,k} - x_{j,k})^2 \right)^{1/2} = \|X_i - X_j\|_2 \quad (23)$$

para o caso especial ($p = 2$) da métrica de Minkowski

$$d_p(X_i, X_j) = \left(\sum_{k=1}^d |x_{i,k} - x_{j,k}|^p \right)^{1/p} = \|X_i - X_j\|_p \quad (24)$$

A Distância Euclideana tem um forte apelo e é comumente usada para avaliar a proximidade de objetos em espaços bidimensionais ou tridimensionais. A desvantagem do uso direto das métricas de Minkowski é a tendência do atributo de maior escala ter grande influência nos demais, passando a interferir diretamente no centroide do agrupamento. Soluções para este problema incluem a normalização dos atributos contínuos (para um intervalo comum ou variância conhecida) ou outros esquemas de ponderação. A correlação linear entre os atributos também pode distorcer as medidas de distância; esta distorção pode ser atenuada pela aplicação de uma transformação nos dados ou usando a distância quadrática de Mahalanobis, dada por:

$$d_M(X_i, X_j) = (X_i - X_j)\Sigma^{-1}(X_i - X_j)^T \quad (25)$$

em que os padrões X_i e X_j são adotados como sendo vetores linha, e Σ é a matriz de covariância amostral dos padrões ou a matriz de covariânciada extraída do processo de geração desses padrões; $d_M(., .)$ atribui diferentes pesos a diferentes atributos com base em suas variações e correlações lineares pareadas. Aqui, assume-se implicitamente que as densidades condicionais de classe são unimodais

e caracterizadas por uma dispersão multidimensional, isto é, que as densidades são multivariadas Gaussianas.

Alguns algoritmos de clusterização funcionam em uma matriz de valores de proximidade ao invés do conjunto de padrões original. Isso acaba sendo útil em situações nas quais se deseja determinar os $n(n - 1)/2$ valores de distância, par a par, para todos os n padrões e armazená-los em uma matriz simétrica.

2.2.3.4 Relação entre Inércia e Número de Clusters

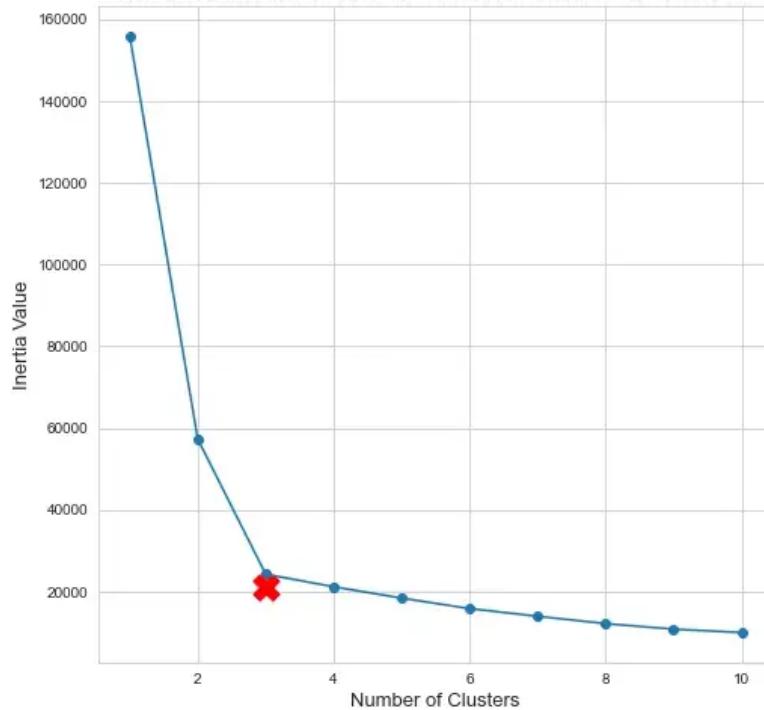
Em algoritmos de aprendizagem supervisionada (SCIENCE, 2021), podemos interpretar diferentes hiperparâmetros, ter diversos números de clusters e calcular diretamente algumas métricas de erro, como a precisão (GOUTTE; GAUSSIER, 2005). O conjunto de hiperparâmetros e o número de *clusters* que levam à maior precisão podem ser usados para o modelo final. Mas isso não é possível para aprendizado não-supervisionado, devido à falta de valores de exemplo a serem ajustados no processo de treino.

Uma forma de contornar isso para podermos avaliar o modelo e comparar os diferentes hiperparâmetros e número de *clusters* seria calcular a soma dos quadrados dentro do cluster, também chamada de Inércia. O número ideal de clusters é encontrado usando os valores de inércia calculados e um método denominado *Elbow-Method*, que é aplicado na curva de inércia. O valor de Inércia ou a soma dos quadrados dentro de um *cluster* dá uma indicação de quão coerentes são os diferentes *clusters* encontrados. Ela pode ser escrita numericamente como sendo:

$$I = \sum_{i=1}^N (x_i - C_k)^2 \quad (26)$$

em que N é o número de amostras no conjunto de dados e C é o centroide (ou centro de um *cluster*). Deste modo, a inércia simplesmente calcula a distância quadrática para cada amostra num dado agrupamento ao centro deste mesmo *cluster*. Esse processo é feito para cada *cluster* e para todos os conjuntos amostrais da base de dados. Quanto menor o valor de inércia, mais coerente são os diferentes agrupamentos obtidos. Para determinar o número ótimo de *clusters* através do valor de inércia, devemos utilizar o método descrito anteriormente: o *Elbow-Method*.

Figura 21 – Curva de Inércia



Fonte: Adaptado de Towards Data Science, 2021.

Na Figura 21, podemos notar claramente o "cotovelo"(ou *elbow point*), representado pela marcação *X* em vermelho. Ele nos fornece o número ótimo de *clusters* para o nosso modelo, que para o caso exemplificado seria igual a 3. Quando adicionamos mais agrupamentos, o valor da inércia tende a descrescer, mas também a informação contida num *cluster* diminui ainda mais. Uma quantidade de agrupamentos além do ideal resulta em decréscimo de performance do nosso modelo, uma vez que as posições dos centroides dos diferentes *clusters* ficam tão próximas umas das outras, que é praticamente impossível distinguir os limites deles no espaço de atributos.

3 METODOLOGIA, RESULTADOS E DISCUSSÃO

3.1 CONFIGURAÇÕES DE HARDWARE E SOFTWARE

Toda a metodologia e a obtenção de resultados dispostos neste capítulo foram realizados com a utilização de *Hardware* e *Software* configurados de acordo com a Tabela 1:

Tabela 1 – Configuração de Hardware e Software

Segmento	Arquitetura ou Fabricante	Descrição
Sistema Operacional	Ubuntu (Linux)	22.04 LTS
CPU	Intel	i7-8565U
Memória (RAM)	Kingston HyperX	16.00GB
Memória Física (HD)	n/a	1.0TB
Memória Física (SSD)	Kingston	120.00GB
Driver GPU Dedicado	NVIDIA (Pascal)	GeForce MX150
Driver GPU Secundário	Dell	WhiskeyLake-U (UHD Graphics 620)

fonte: Produção do Próprio Autor.

Além disso, a implementação dos algoritmos de ML, bem como a arquitetura geral do sistema, foi feita utilizando a linguagem *Python* na sua versão 3.10.6. As dependências (bibliotecas do gerenciador PIP) estão dispostas na tabela a seguir, em suas respectivas versões:

Tabela 2 – Listagem das Principais Dependências do Python Utilizadas

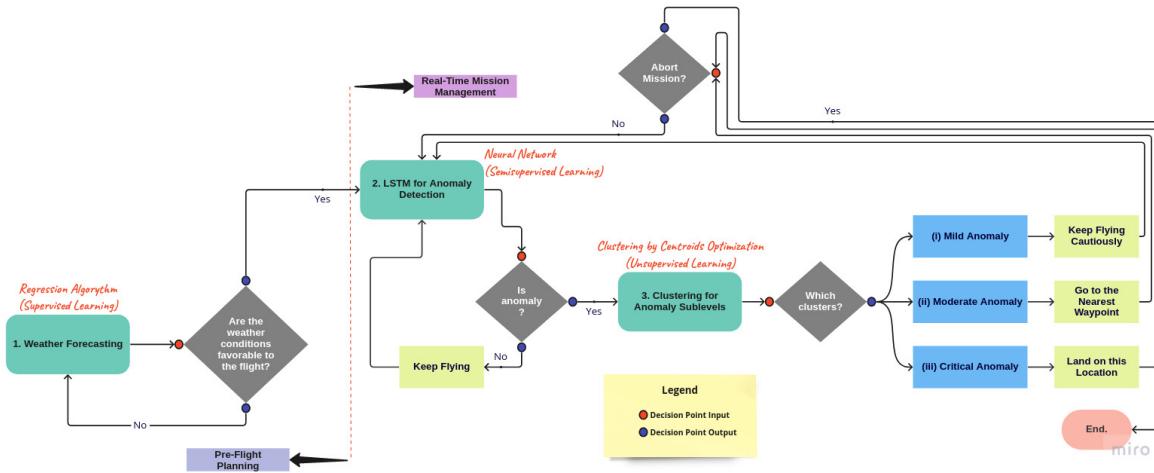
Dependência	Versão	Utilidade
colorama	0.4.4	personalização de cores no terminal
matplotlib	3.3.4	visualização de dados
numpy	1.19.5	tratativa numérica e manipulação de matrizes
pandas	1.2.5	manipulação de estruturas de dados organizadas (tabelas)
progressbar	2.5	report do progresso de uma aplicação
scikit-learn	0.24.1	IA, ML e DL
scipy	1.6.2	tratativa numérica e análise estatística
seaborn	0.11.1	visualização de dados
shap	0.40.0	interpretabilidade de modelos
tensorflow	2.5.0	manipulação de tensores e arquitetura de redes neurais
termcolor	1.1.0	personalização de cores no terminal

fonte: Produção do Próprio Autor.

3.2 ESCOPO GERAL DO TRABALHO

Para o desenvolvimento deste trabalho, foram abordadas técnicas gerais de *Machine Learning* e *Deep Learning*. A Figura 22 apresenta a arquitetura geral para o fluxo de dados, desde a leitura de variáveis climáticas no pré-voo até a classificação final dos níveis de anomalia e tomada de decisão em tempo real.

Figura 22 – Arquitetura geral do sistema



Fonte: Produção do próprio autor.

Esta arquitetura foi desenvolvida pensando nas boas práticas de engenharia de *Software* (ALLBEE, 2018). Neste diagrama, podemos notar os componentes principais de um fluxo de dados inseridos em um processo computacional integrado.

3.3 OBJETIVO GERAL

O objetivo geral deste trabalho foi o de arquitetar um sistema computacional auto-suficiente para o planejamento e gerenciamento em tempo real de voo autônomo executado por um drone. O drone utilizado para a modelagem inicial foi do tipo quadricóptero, simulado através do Gazebo (GAZEBO, 2022).

3.4 OBJETIVO ESPECÍFICO

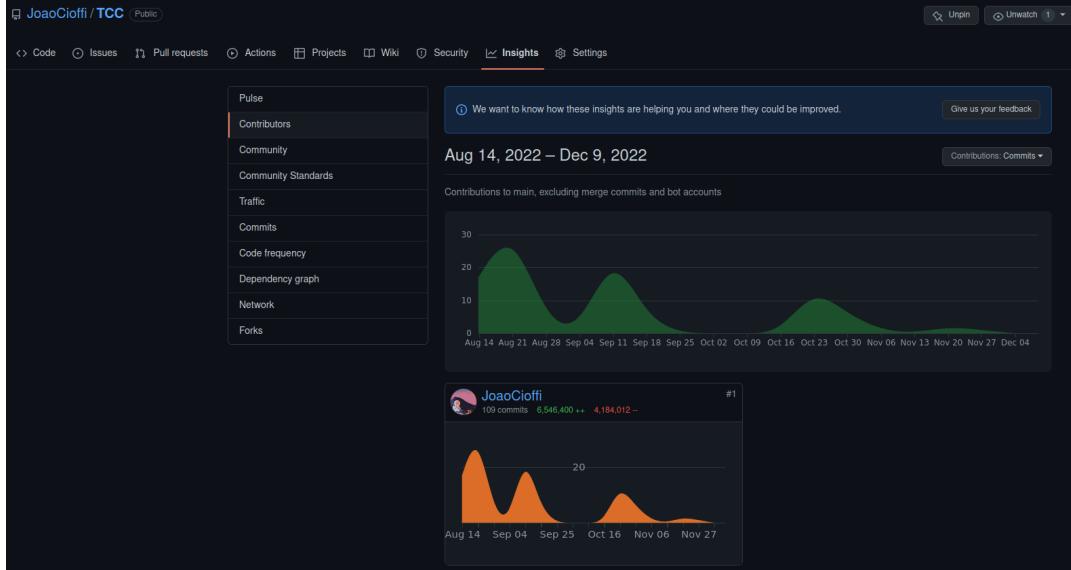
Para atender ao objetivo geral, faz-se necessário o entendimento da arquitetura do sistema previamente apresentado como segue:

- Módulo do pré-voo: tem por objetivo fazer previsões de variáveis climáticas e atmosféricas associadas a uma leitura padrão de um informe METAR. Caberá ao supervisor da missão (ou piloto) acionar o voo da aeronave quando for conveniente.
- Classificação binária dos dados: tem por objetivo classificar de forma positiva ou negativa um conjunto de variáveis como sendo anomalias num dado instante de tempo.
- Clusterização dos dados: tem por objetivo tomar como entrada os dados anteriormente classificados como anomalias para seu devido agrupamento em subníveis (anomalia leve, anomalia moderada ou anomalia crítica).

3.5 REPOSITÓRIO DO PROJETO

Toda a arquitetura deste trabalho, bem como os principais elementos do projeto prático, podem ser acessados publicamente e estão dentro do repositório do *Github*: <<https://github.com/JoaoCioffi/TCC>>.

Figura 23 – Parte do repositório do Github criado para arquivar o projeto



Fonte: Produção do Próprio Autor.

É válido ressaltar que o repositório em questão, apesar de estar em domínio público, faz parte integralmente deste trabalho. Portanto, qualquer reprodução pode ser realizada através do devido referenciamento do autor.

3.6 PRIMEIROS PASSOS: GERANDO A BASE DE DADOS

3.6.1 Dados Meteorológicos

3.6.1.1 Abordagem Contextual

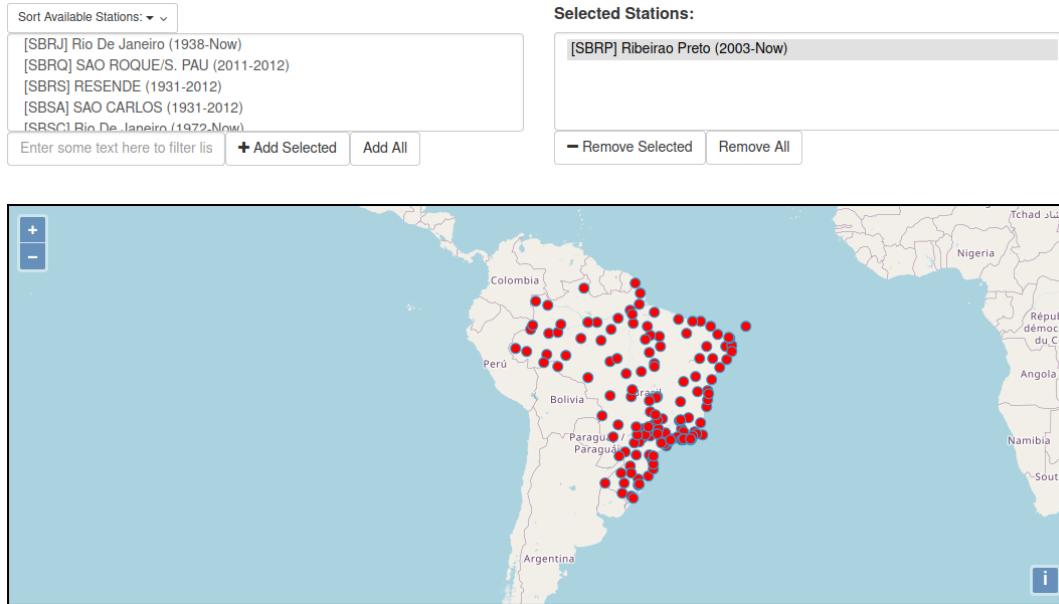
Para a geração dos dados meteorológicos, foi feita uma requisição da API pública desenvolvida pela *Iowa State University* (MESONET, 2022), que tem o objetivo de prover relatórios METAR acumulados ao longo dos anos - relatórios estes referentes aos principais aeródromos ou aeroportos localizados ao redor do mundo.

Para o presente trabalho, foi realizada uma filtragem de 5 anos consecutivos (01/01/2017 a 31/12/2021), com relatórios da localidade do Aeroporto Estadual Dr. Leite Lopes (SBRP). Os motivos que levaram à adoção desses registros e da localidade em questão são:

- **Localidade:** Ribeirão Preto, cidade do interior do estado de São Paulo, possui um forte atrativo para o desenvolvimento do agronegócio nacional. Por este motivo, é muito comum vermos aplicações em lavouras da região utilizando inteligência artificial e, sobretudo, drones de pulverização aeroagrícola.

- **Periodicidade:** Não existe uma regra específica para ser adotada como período mínimo de geração de uma base de dados histórica (série temporal). A escolha foi completamente arbitrária, porém decidiu-se filtrar os dados mais recentes possíveis até o presente momento da elaboração deste trabalho, adotando os últimos 5 anos anteriores (consolidados) para que se tivesse uma massa de dados relevante.

Figura 24 – Exemplo de estações de leitura METAR disponíveis no Brasil



Fonte: Adaptado de IEM, 2022.

A própria API (Figura 24) fornece uma documentação para as variáveis contidas em sua base. Denominaremos metadados todas essas variáveis, incluindo as que foram utilizadas na modelagem. São alguns desses metadados:

- **station:** código do aeródromo ou aeroporto de onde foi extraído o relatório METAR.
- **valid:** Registro do horário da observação.
- **tempf:** Temperatura do ar no momento da leitura [$^{\circ}F$].
- **feel:** Temperatura aparente ou sensação térmica [$^{\circ}F$].
- **dwpf:** Temperatura do ponto de orvalho [$^{\circ}F$].
- **relh:** Umidade relativa do ar [%].
- **drct:** Direção do vento relativa ao Norte Verdadeiro [°].
- **sknt:** Velocidade do vento [knots].
- **p01i:** Precipitação de uma hora para o período desde o horário de observação até o horário da redefinição da precipitação horária anterior.

- **alti:** Pressão atmosférica [*inHg*].
- **skyc1:** Código da primeira formação de nuvens visíveis acima do solo (*FEW* - poucas nuvens, *SCT* - nuvens esparsas, *BKN* - nublado, *OVC* - nuvens tempestuosas ou muito densas).
- **sky11:** Altitude relativa da primeira formação de nuvens visíveis acima do solo [*feet*].
- **vsby:** Visibilidade [*miles*].

3.6.1.2 Abordagem Prática

Toda a elaboração da abordagem prática foi realizada utilizando a ferramenta integrada *Jupyter Notebook* (JUPYTER, 2022), desde a etapa analítica inicial da base de dados até a implementação final do modelo.

3.6.1.2.1 Análise Exploratória dos Dados

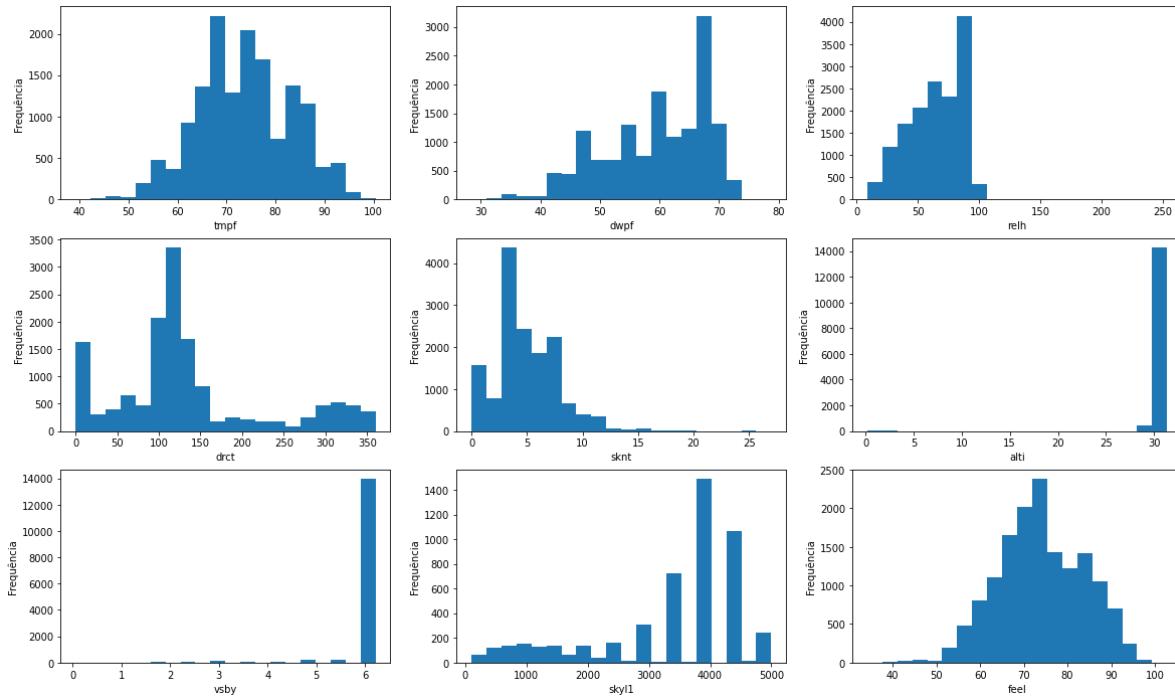
Nesta etapa foi realizada a interpretação de distribuição dos dados brutos afim de compreender o comportamento estatístico da nossa base. Os resultados estão dispostos na Tabela 3.

Tabela 3 – Interpretação Estatística dos Dados

Medida	tmpf	dwpf	relh	drct	sknt	alti	vsby	sky11	feel
total de registros	43208	43208	43208	43208	43208	43208	43208	43208	43208
média	74.72	59.64	64.18	135.97	5.08	29.91	6.05	3334.53	74.88
desvio padrão	9.63	8.61	22.25	92.56	2.95	1.59	0.70	760.83	9.71
mínimo	33.80	14.00	7.16	0.00	0.00	0.18	0.00	100.00	30.75
máximo	104.00	84.20	253.63	360.00	30.00	31.36	6.21	10000.00	118.28
percentil 25%	68.00	53.60	46.41	90.00	3.00	29.91	6.21	3334.53	68.00
percentil 50%	73.40	60.80	67.22	120.00	5.00	29.97	6.21	3334.53	73.79
percentil 75%	82.40	66.20	83.09	160.00	7.00	30.06	6.21	3334.53	82.27

fonte: Produção do Próprio Autor.

Figura 25 – Histograma de frequência para as principais variáveis numéricas do METAR



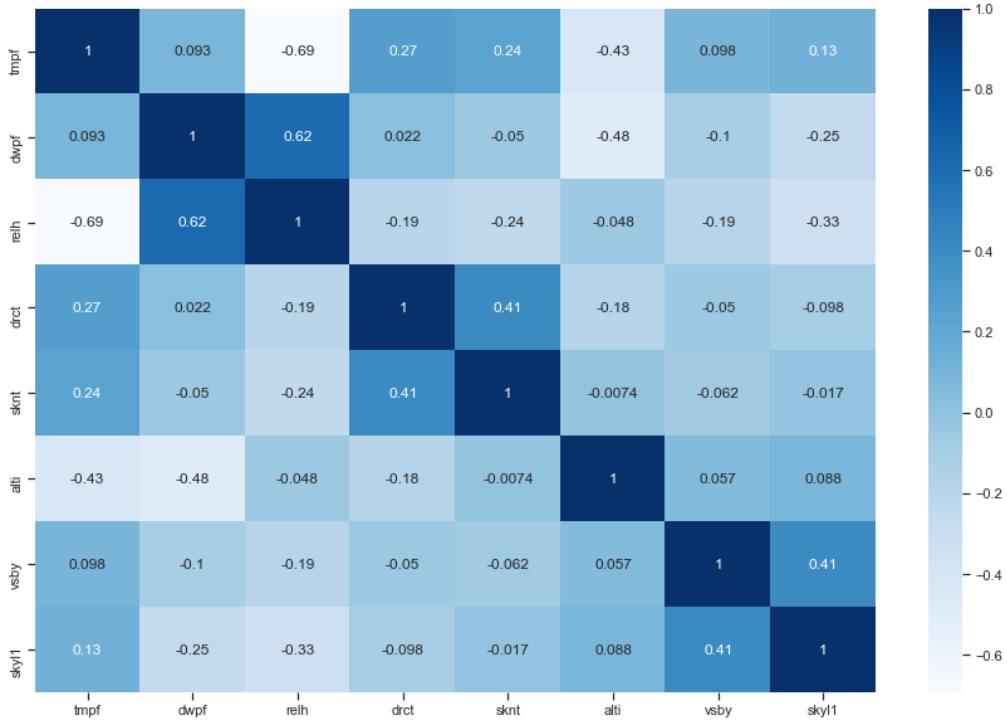
Fonte: Produção do Próprio Autor.

Os dados dispostos na Tabela 3, juntamente com a interpretação gráfica trazida pela Figura 25, nos permite afirmar que:

1. No geral, apesar de não termos uma tendência de distribuição do tipo Gaussiana Normal (com exceção de *feel*), a maioria das variáveis possuem um bom comportamento e dentro do esperado - isto é, não foi observado nenhum valor fora do contexto real (apesar de as distribuições não serem do tipo gaussiana normal, são valores factíveis e possíveis de serem representados no contexto real em que estão inseridos). Isso sugere, num primeiro momento, uma boa qualidade dos dados apresentados.
2. Com relação à *outliers*, não verificaram-se grandes discrepâncias. Apenas podemos notar dados fora do esperado dentro dos domínios reais de *skyl* (que apresentou valor máximo de 1000.00 ft) e *vsby* (com valor mínimo de 0.00 mi). Isso pode ser facilmente tratado nas etapas subsequentes, como veremos adiante.

Em seguida, foi verificada a correlação numérica de Pearson (σ) para algumas dessas variáveis apresentadas anteriormente. O resultado é apresentado na Figura 26.

Figura 26 – Matriz de correlação numérica (Pearson)



Fonte: Produção do Próprio Autor.

Da matriz apresentada anteriormente, podemos verificar que existe pouca correlação entre as variáveis (o que pode ser um ponto positivo num primeiro momento, pois quando temos muitas delas com uma correlação alta, teremos o que chamamos de multicolinearidade - isto é, forte dependência entre as variáveis, que acaba interferindo no ajuste dos modelos no geral). As únicas variáveis que mais se mostraram evidentes foram *dwpf* e *relh*, sendo $\sigma \approx 0.62$ (isso é bastante factível se imaginarmos, por exemplo, que a umidade relativa do ar ao longo do dia depende até certo ponto da temperatura do ponto de orvalho durante a manhã). Além disso, também temos uma alta correlação (em módulo) entre *tmpf* e *relh*, sendo $\sigma \approx -0.69$.

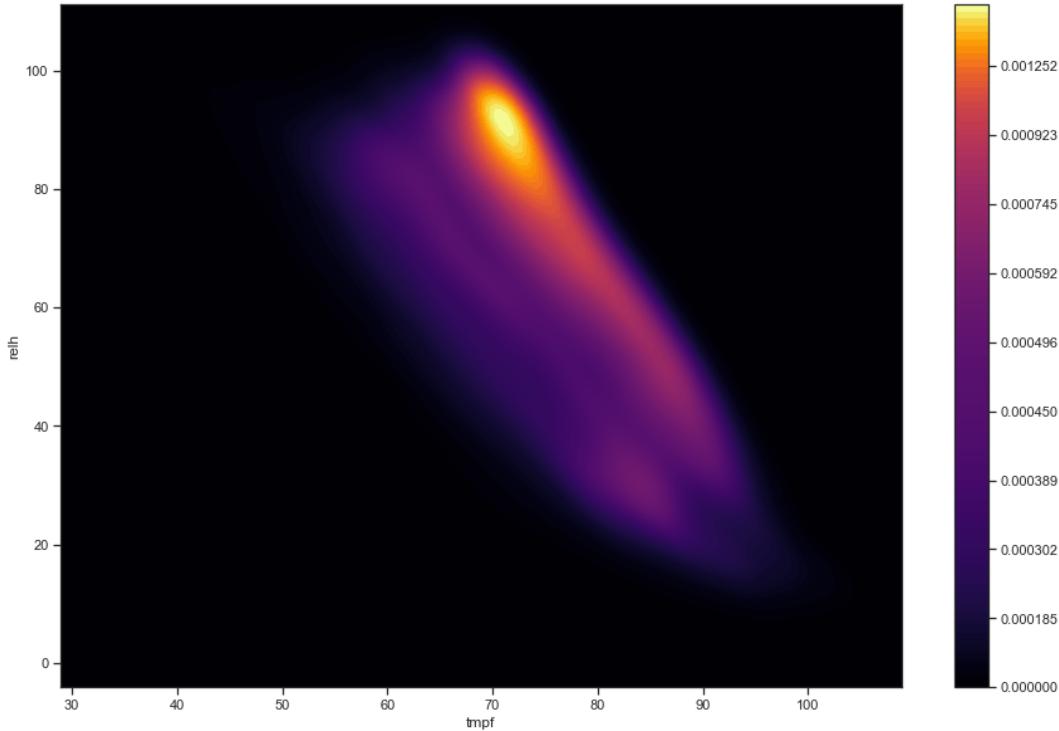
Desta forma, verificada que existiu a maior correlação entre *tmpf* e *relh*, foi feita uma interpretação mais a fundo para entender a distribuição de densidade dessas variáveis (Figura 27).

Podemos notar que existe uma maior densidade de dados nos domínios de *relh*:[80, 100] e *tmpf*:[70, 80]. Isso também é bastante factível, pois podemos interpretar que a maior umidade relativa do ar (na faixa dos 80% a 100%) é mantida quando se tem uma temperatura ambiente (próxima dos 80°F ≈ 26.67°C).

3.6.1.2.2 Feature Engineering

A etapa de engenharia de *features* (ou, simplesmente, *Feature Engineering*) é o processo de transformar dados brutos em novas características, ou seja, em representações numéricas que melhor representem os dados e que serão aproveitados de forma mais eficiente pelos modelos de ML.

Figura 27 – Umidade relativa [%] vs. temperatura do ar [°F] mapeados por suas densidades



Fonte: Produção do Próprio Autor.

Desta forma, a primeira transformação realizada foi a criação de novas classificações dos dados de acordo com as condições de categoria de voo definidas pelo NWS (SERVICE, 2022). Essas classificações são mostradas na Tabela 4.

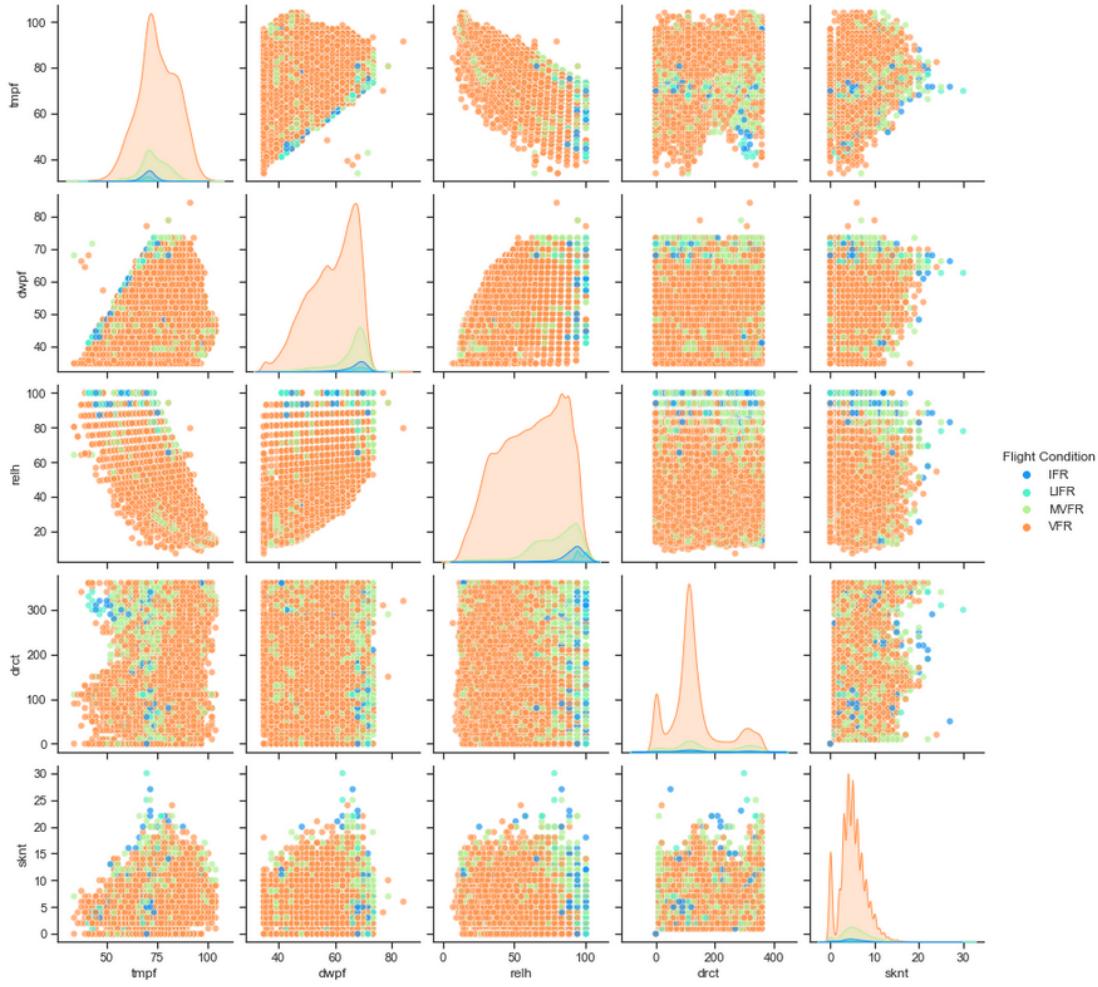
Tabela 4 – Definições para as diferentes categorias de voo

Código	Descrição	Visibilidade	Teto
VFR	Regra de Voo Visual	acima de 8km	acima de 3000ft
MVFR	Regra de Voo Marginal Visual	5km – 8km	1000ft – 3000ft
IFR	Regra de Voo por Instrumentos	1.5km – 5km	500ft – 1000ft
LIFR	Regra de Voo por Instrumentos na Condição Baixa	abaixo de 1.5km	abaixo de 500ft

fonte: Produção do Próprio Autor.

Para as condições descritas anteriormente, foi adotada a variável *vsby* como sendo a principal para os domínios citados. Desta forma, foi possível obter a distribuição numérica das diferentes condições registradas ao longo dos 5 anos.

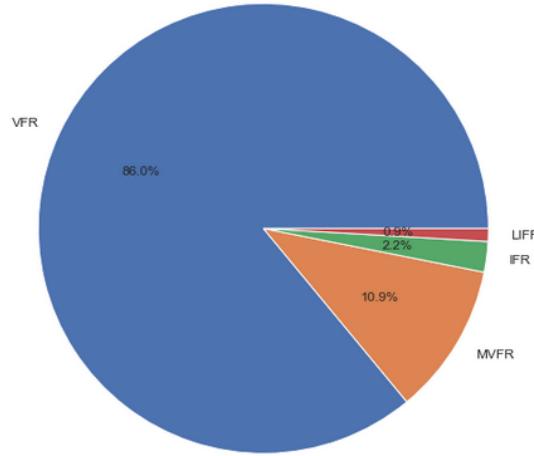
Figura 28 – Curvas de distribuição para as condições de voo



Fonte: Produção do Próprio Autor.

Como podemos perceber, a maior densidade de distribuição nos dados ocorre na condição de voo visual (VFR), representado na Figura 28. Isso é corroborado pela Figura 29, que revela que esta condição, ao longo dos 5 anos, representou cerca de 86% dos dados climáticos. Para uma missão de pulverização aeroagrícola efetuada por drones, por exemplo, isso é extremamente vantajoso.

Figura 29 – Análise proporcional das condições de voo



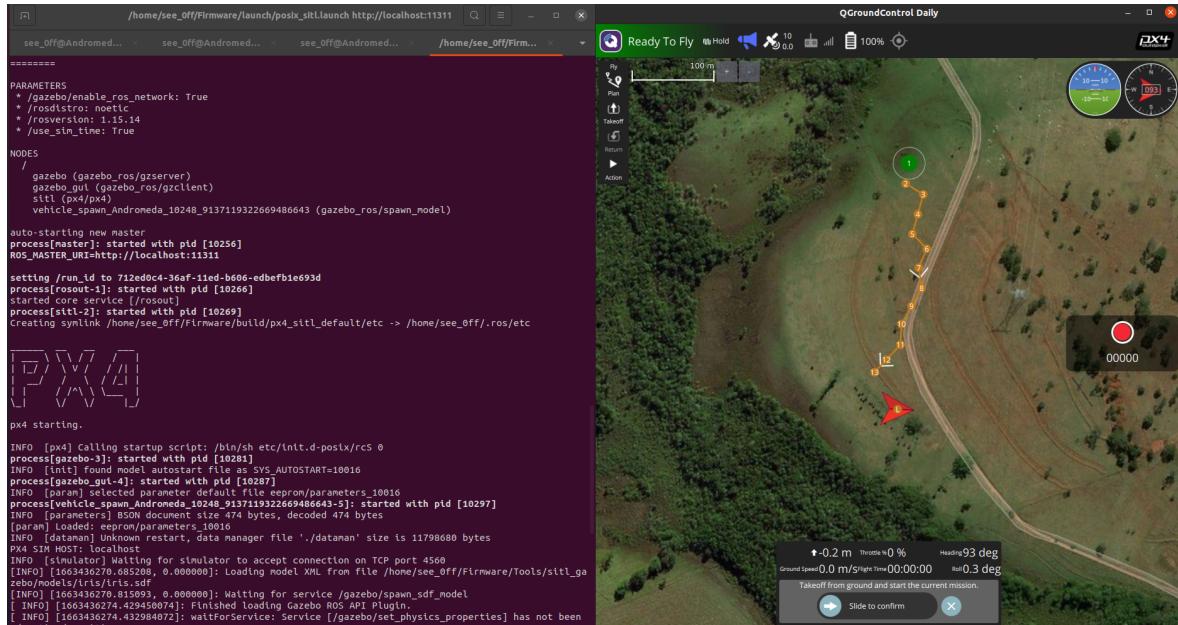
Fonte: Produção do Próprio Autor.

3.6.2 Dados de Voo por Simulação

3.6.2.1 Abordagem Contextual

Para a geração dos dados de voo, foi feita a coleta de voos através do gerenciador de missões *QGroundControl* (DRONECODE, 2022) acoplado ao módulo de pacotes de simulação em robótica *Gazebo* (GAZEBO, 2022). A Figura 30 mostra a interface gráfica desse sistema em execução.

Figura 30 – Sistema utilizado para a geração dos dados de voo por simulação



Fonte: Produção do Próprio Autor.

A base de dados foi construída com a leitura de um total de 200 voos simulados de um drone quadricóptero. Cada missão executada (considera-se aqui uma missão a série temporal de dados consecutivos lidos desde a decolagem até o pouso final da aeronave, de forma ininterrupta) levou cerca de 180 registros (sendo a taxa de amostragem de aproximadamente 1 leitura por segundo). Desta forma, obtêve-se um conjunto de dados final de 36000 linhas. A variância dos dados foi obtida por alguns fatores externos, tais como: alteração abrupta da rota, desaceleração dos motores num dado trecho da missão, envios de comandos para obtenção das principais respostas de voo (ϕ, θ, ψ), dentre outros fatores.

Os metadados obtidos das simulações foram:

- **roll:** Também denominada de rolagem (ϕ), é a medida de rotação ou defasagem angular em torno do eixo x da aeronave [dg].
- **pitch:** Também denominada de arfagem (θ), é a medida de rotação ou defasagem angular em torno do eixo y da aeronave [dg].
- **yaw:** Também denominada guinada (ψ), é a medida de rotação ou defasagem angular em torno do eixo z da aeronave [dg].
- **heading:** Também chamada de proa magnética ou direção, é a medida de defasagem angular do nariz da aeronave em relação a um referencial - pode ou não ser assumido como sendo o Norte Verdadeiro - [dg].
- **rollRate:** Também chamada de taxa de rolagem, é a variação temporal dessa grandeza [dg/s].
- **pitchRate:** Também chamada de taxa de arfagem, é a variação temporal dessa grandeza [dg/s].
- **yawRate:** Também chamada de taxa de guinada, é a variação temporal dessa grandeza [dg/s].
- **groundSpeed:** É a medida de velocidade relativa ao solo [m/s].
- **airSpeed:** É a medida de velocidade real da aeronave [m/s].
- **climbRate:** É a taxa de subida da aeronave [m/s].
- **altitudeRelative:** É a altitude relativa ao solo [m].
- **altitudeAMSL:** É a altitude da aeronave em relação ao nível do mar [m].
- **flightDistance:** É a distância total percorrida em relação ao ponto de decolagem [m].
- **flightTime:** É o tempo decorrido da missão [HH:MM:SS].
- **throttlePct:** É o nível de rotação dos motores num dado instante [%].

3.6.2.2 Abordagem Prática

Da mesma forma que foi feito para a modelagem dos dados METAR, nesta etapa também utilizou-se a ferramenta integrada *Jupyter Notebook* (JUPYTER, 2022), desde a parte analítica inicial até a implementação final do modelo.

3.6.2.2.1 Análise Exploratória dos Dados

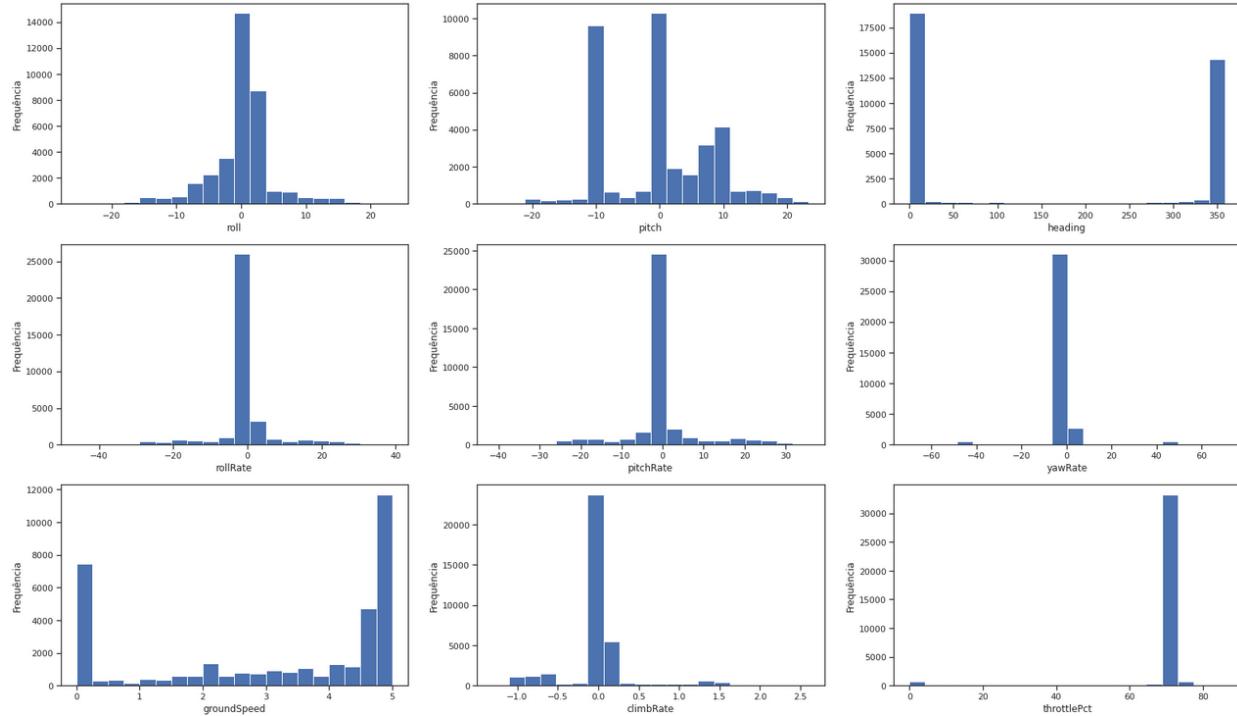
Nesta etapa foi realizada uma interpretação estatística da nossa base afim de compreender a distribuição dos dados brutos. Os resultados estão dispostos na Tabela 5.

Tabela 5 – Interpretação Estatística dos Dados

Medida	heading	rollRate	pitchRate	yawRate	groundSpeed	throttlePct
total de registros	36000	36000	36000	36000	36000	36000
média	157.15	0.02	0.21	0.04	3.15	68.79
desvio padrão	173.67	7.38	8.34	10.64	1.94	11.67
mínimo	0.00	-46.10	-41.40	-69.70	0.00	0.00
máximo	359.00	39.00	35.60	70.40	5.00	86.00
percentil 25%	0.00	-0.30	-0.40	-0.10	1.50	71.00
percentil 50%	1.00	0.00	0.00	0.00	4.10	71.00
percentil 75%	359.00	0.40	0.30	0.10	4.80	71.00

fonte: Produção do Próprio Autor.

Figura 31 – Histograma de frequência para as principais variáveis de voo



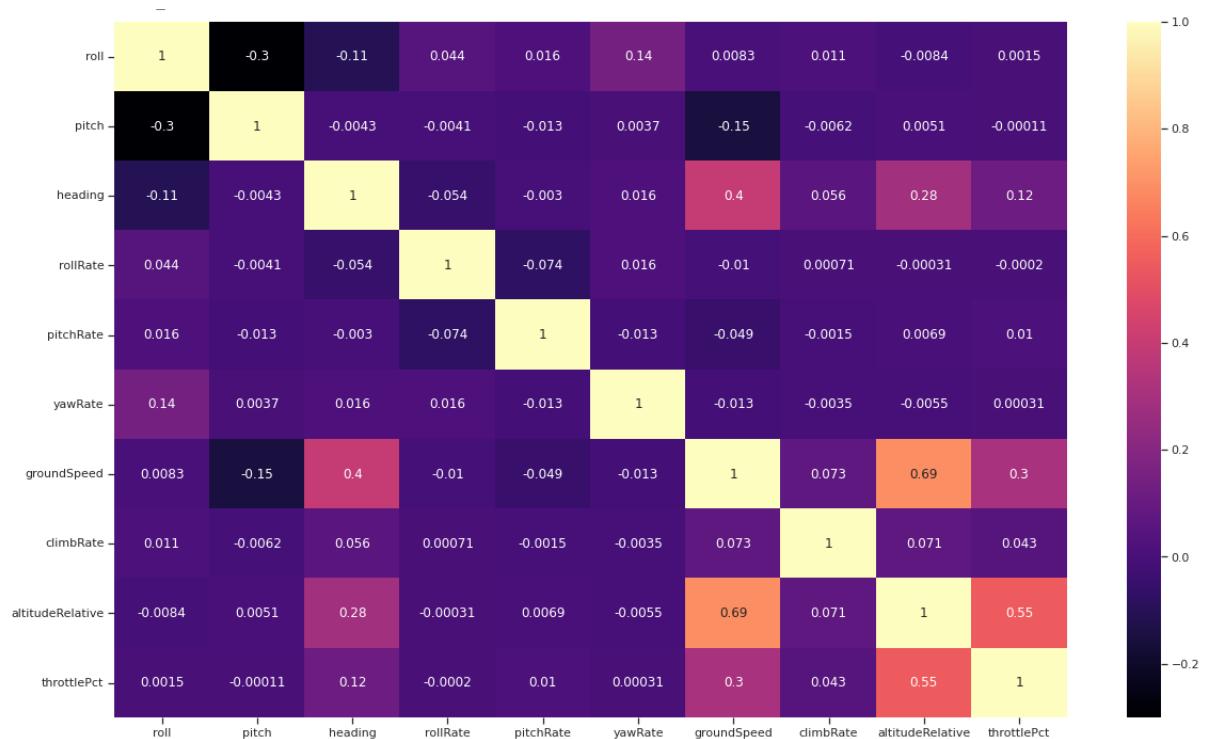
Fonte: Produção do Próprio Autor.

Os dados dispostos na Tabela 5, juntamente com a interpretação gráfica trazida pela Figura 31, nos permite afirmar que:

1. No geral, apesar de não termos uma tendência de distribuição do tipo Gaussiana Normal (com exceção de *roll*), a maioria das variáveis possuem um bom comportamento e dentro do esperado - isso sugere, num primeiro momento, uma boa qualidade dos dados apresentados.
2. Com relação à *outliers*, não verificaram-se grandes discrepâncias, no entanto pode-se notar lacunas nos domínios de algumas variáveis (como por exemplo, *throttlePct*).

Em seguida, foi verificada a correlação numérica de Pearson (σ) para algumas dessas variáveis apresentadas anteriormente. O resultado é apresentado na Figura 32.

Figura 32 – Matriz de correlação numérica (Pearson)



Fonte: Produção do Próprio Autor.

Da matriz apresentada anteriormente, podemos verificar que existe pouca correlação entre as variáveis. Algumas delas apresentaram maior colinearidade (como por exemplo, relações entre *rollRate*, *pitchRate*, *groundSpeed* e *altitudeRelative*), que pode ser explicada pelo fato de que os movimentos axiais da aeronave em questão são acoplados e que para a segurança da missão durante o voo é preferível manter maiores velocidades em altitudes mais elevadas.

3.7 MÓDULO 1: PREVISÃO DE DADOS METAR

3.7.1 Criação das Matrizes Adequadas ao Modelo

Para a criação das matrizes do modelo, isto é, variáveis de entrada (X) e variáveis resposta (y), foi inicialmente feito uma tratativa de separação de linhas pares e ímpares do conjunto de dados inicial. Esse critério foi fundamental para compreender registros "do passado" (referentes às linhas pares) e registros "futuros", referentes às linhas ímpares do nosso modelo. O principal motivo para ter-se adotado esse padrão foi referente ao modo como o algoritmo viria a funcionar: com registros METAR de hora em hora, devemos ter como *input* do modelo todos os principais dados climáticos informados naquele instante para que possamos prever os dados para a próxima hora (*output*). Isso se torna muito auto-explicativo quando pensamos, por exemplo, no planejamento de missão de um voo de drone - uma vez que a previsão com 1h de antecedência pode ser fundamental para uma possível mudança de cenário ou até mesmo a escolha em adiar ou postergar a missão caso seja necessário.

- **input:** contém todas as variáveis do conjunto de dados inicial, incluindo as variáveis extraídas pelas transformações de *Feature Engineering*. Sua representação simplificada pode ser entendida como sendo: $X = [\text{'tmpf'}, \text{'dwpf'}, \text{'reh'}, \text{'drct'}, \text{'alti'}, \text{'skyc1'}, \text{'month'}, \text{'seasonflag'}, \text{'dayofweek'}, \text{'frcode'}]$. Neste caso, *month* representa o mês, *'seasonflag'* representa um atributo referente à estação do ano, *'dayofweek'* representa o dia da semana e *'frcode'* representa à regra de voo aplicada no momento, sendo estas quatro últimas extraídas do processo de *Feature Engineering*.
- **output:** contém apenas as variáveis numéricas que impactam diretamente no desempenho do voo. Sua representação simplificada pode ser entendida como sendo: $y = [\text{'tmpf'}, \text{'reh'}, \text{'sknt'}, \text{'alti'}]$.

Após criadas as matrizes principais do modelo, passou-se para a etapa de separação das variáveis em treino e teste. Essa é uma parte importante do processo, pois o ajuste adequado dessas amostras pode gerar resultados bem satisfatórios, ou ainda gerar problemas de *overfitting* quando não separados da maneira correta. Desta forma, foi estabelecido um conjunto amostral de 25% dos dados para teste, sendo os outros 75% de X e y representando os dados de treino, isto é, os dados necessários para o ajuste do modelo.

3.7.2 Seleção do Modelo

Como foi abordado na seção anterior, nossa variável resposta do modelo é estritamente numérica - isto é, desejamos prever dados climáticos essencialmente numéricos num primeiro momento. Desta forma, foi necessário adotar um algoritmo preditivo para ajustar nossas matrizes e chegar nas variáveis climáticas de interesse. Foram escolhidos alguns mais comumente utilizados e posteriormente avaliados

seus desempenhos (Tabela 6). Toda a implementação numérica foi realizada com as ferramentas nativas da biblioteca *Scikit-Learn* (LEARN, 2022).

Tabela 6 – Valores de R^2 encontrados para cada uma das variáveis em questão

Algoritmo	<i>tmpf</i>	<i>relh</i>	<i>sknt</i>	<i>alti</i>	média de R^2
Regressão Linear	0.912356	8.826107	51.270046	21.206183	20.553673
LARS Lasso	0.912356	8.826107	51.270046	21.206183	20.553673
SVM	0.901640	9.013409	51.465525	21.207635	20.647052
Random Forest	0.914206	8.862739	51.266211	21.206219	20.562344
Gradient Boosting	0.919527	8.826254	51.277854	21.206124	20.557440

fonte: Produção do Próprio Autor.

Por análise da Tabela 6, foi adotado o algoritmo de *Gradient Boosting* como sendo o mais adequado para o problema. Apesar de não ter sido o mais performático por análise da média dos R^2 obtidos, essa escolha se deve por três fatores:

- maior valor de R^2 encontrado para a variável *tmpf*.
- maior quantidade de opções de hiperparâmetros a serem ajustados, o que torna esse algoritmo muito versátil e passível de ser otimizado.
- maior quantidade de exemplos práticos na literatura, permitindo portanto se basear em modelos previamente implementados para o refinamento adequado.

3.7.3 Ajuste do Modelo Através dos Hiperparâmetros

Após verificado que o algoritmo de *Gradient Boosting* seria o melhor dentre os possíveis candidatos, foi utilizada a técnica de *Randomized Search* do próprio *Scikit-Learn* (LEARN, 2022). Essa técnica consiste na análise combinatória e iterativa de diferentes hiperparâmetros previamente setados pelo usuário e na obtenção das métricas de desempenho do modelo de acordo com a combinação entre elas. A Figura 33 apresenta um trecho de código extraído nessa etapa.

O ajuste foi realizado como apresentado na Figura 33, sendo os melhores hiperparâmetros encontrados:

- **n_estimators:** 700
- **min_samples_split:** 2
- **min_samples_leaf:** 2
- **min_impurity_decrease:** 0
- **max_leaf_nodes:** 10
- **max_depth:** 3
- **learning_rate:** 0.5

Figura 33 – Exemplo da *Randomized Search* utilizada no processo

```

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.multioutput import MultiOutputRegressor
from sklearn.model_selection import RandomizedSearchCV

model = MultiOutputRegressor(GradientBoostingRegressor(learning_rate=0.1, n_estimators=100, subsample=1.0,
                                                       criterion='friedman_mse', min_samples_split=2,
                                                       min_samples_leaf=1,
                                                       min_weight_fraction_leaf=0.0, max_depth=3,
                                                       min_impurity_decrease=0.0, init=None, random_state=42,
                                                       max_features=None,
                                                       alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False,
                                                       validation_fraction=0.1, n_iter_no_change=None, tol=0.0001,
                                                       ccp_alpha=0.0))

hyperparameters = dict(estimator_learning_rate=[0.05, 0.1, 0.2, 0.5, 0.9],
                       estimator_n_estimators=[20, 50, 100, 200, 300, 500, 700, 1000],
                       estimator_criterion=['friedman_mse', 'mse'], estimator_min_samples_split=[2, 4, 7, 10],
                       estimator_max_depth=[3, 5, 10, 15, 20, 30], estimator_min_samples_leaf=[1, 2, 3, 5, 8, 10],
                       estimator_min_impurity_decrease=[0, 0.2, 0.4, 0.6, 0.8],
                       estimator_max_leaf_nodes=[5, 10, 20, 30, 50, 100, 300])

randomized_search = RandomizedSearchCV(model, hyperparameters, random_state=42, n_iter=5, scoring=None,
                                         n_jobs=2, refit=True, cv=5, verbose=True,
                                         pre_dispatch='2*n_jobs', error_score='raise', return_train_score=True)

hyperparameters_tuning = randomized_search.fit(X_train, y_train)
print('Best Parameters = {}'.format(hyperparameters_tuning.best_params_))

Fitting 5 folds for each of 5 candidates, totalling 25 fits
Best Parameters = {'estimator_n_estimators': 700, 'estimator_min_samples_split': 2, 'estimator_min_samples_leaf': 2, 'estimator_min_impurity_decrease': 0, 'estimator_max_leaf_nodes': 10, 'estimator_max_depth': 3, 'estimator_learning_rate': 0.5, 'estimator_criterion': 'friedman_mse'}

```

Fonte: Produção do Próprio Autor.

- **criterion:** *friedman_mse*

Apesar de serem os possíveis hiperparâmetros otimizados, foi feito um segundo ajuste e verificou-se que para um *learning rate* de 0.01 e *max leaf nodes* de 20 os resultados se mostraram mais satisfatórios.

3.7.4 Resposta do Modelo

3.7.4.1 Resposta Geral

Uma vez adotado o modelo e refinados os hiperparâmetros, a próxima etapa foi retreinar o algoritmo utilizando os dados de treino (X_{train}, y_{train}) e avaliar a resposta final nos dados de teste (X_{test}, y_{test}). Desta forma, obtiveram-se as seguintes métricas ajustadas para a matriz resposta do modelo (y):

Tabela 7 – Métricas finais obtidas na matriz resposta do modelo

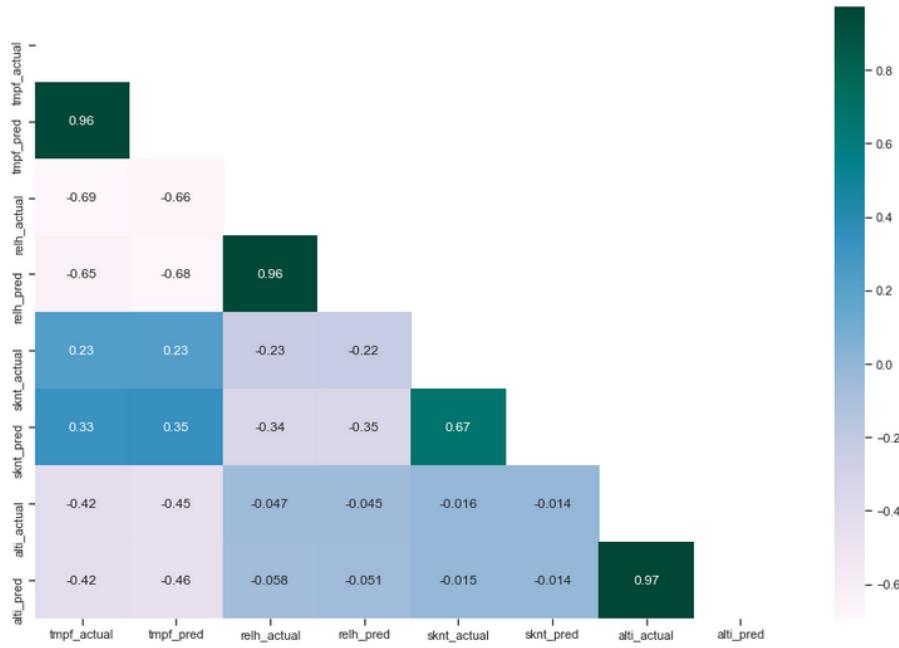
Métrica	tmpf	rehl	sknt	alti
MAE	1.984	4.764	1.617	0.016
RMSE	7.742	42.051	4.720	0.001
R^2	0.917	0.915	0.454	0.947

fonte: Produção do Próprio Autor.

A seguir veremos uma correlação entre os valores esperados e os valores previstos para as variáveis da matriz y , isto é, um comparativo na escala de Pearson para *tmpf*, *rehl*, *sknt* e *alti*.

A Figura 34 nos mostra que dentre as variáveis previstas em y apenas uma delas não apresentou forte colinearidade - que no caso foi a velocidade do vento (*sknt*). Isso é um ponto positivo para nossa matriz resposta, pois indica que nossas variáveis previstas estão se comportando linearmente dentro do esperado quando comparadas ao padrão tido como esperado.

Figura 34 – Matriz de correlação entre variáveis esperadas e variáveis previstas pelo algoritmo



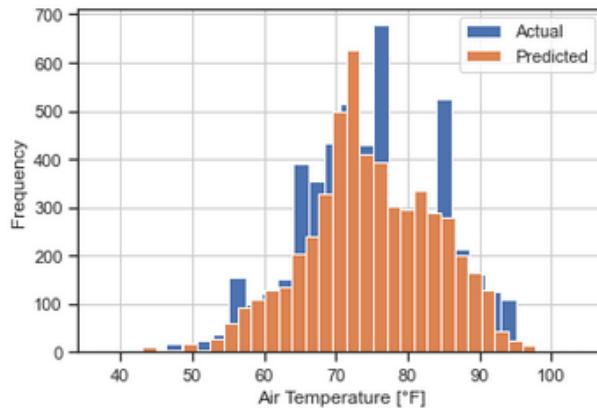
Fonte: Produção do Próprio Autor.

3.7.4.2 Respostas Isoladas por Variável

3.7.4.2.1 Temperatura do Ar

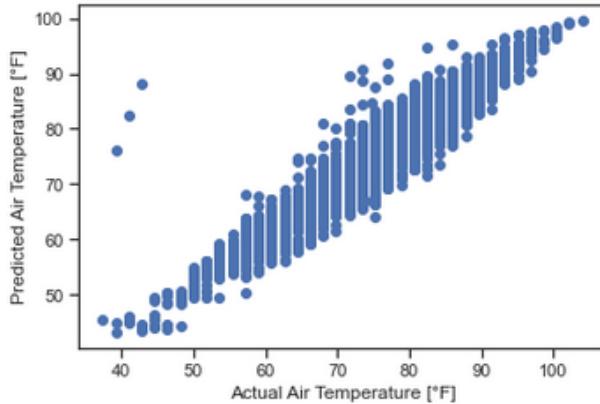
Nesta seção apresentam-se os resultados para a temperatura do ar, prevista em °F.

Figura 35 – Histograma para a temperatura do ar esperada e temperatura do ar prevista



Fonte: Produção do Próprio Autor.

Figura 36 – Dispersão: temperatura do ar esperada vs. temperatura do ar prevista



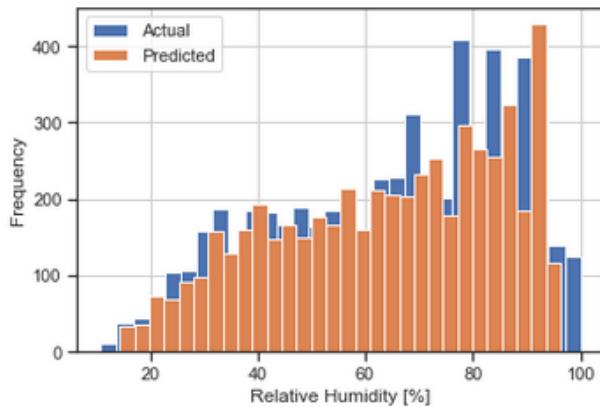
Fonte: Produção do Próprio Autor.

A Figura 35 mostra o histograma para a temperatura do ar esperada ou real (*actual*) sobreposto pela temperatura prevista ou obtida (*predicted*). Podemos perceber que, salvos alguns pontos isolados de maior concentração de frequência para a variável esperada, temos um comportamento muito próximo entre elas. Além disso, a Figura 36 evidencia que a dispersão dessas variáveis ocorre de forma muito similar, num comportamento praticamente linear (ou seja, se tirarmos os valores médios obtidos entre os pontos da Figura 36, teremos uma reta que pode ser modelada através de uma função de primeiro grau e que intercepta esses pontos médios obtidos).

3.7.4.2.2 Umidade Relativa do Ar

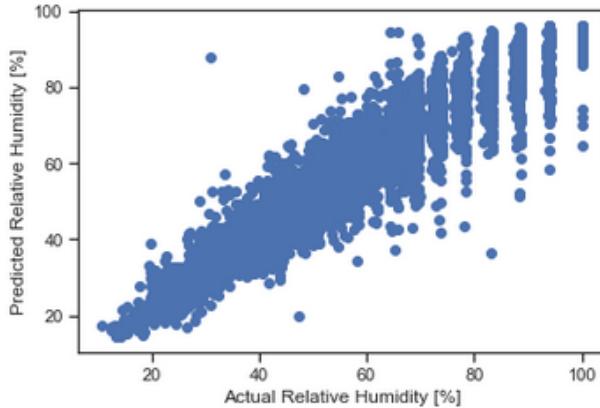
Nesta seção apresentam-se os resultados para a umidade relativa do ar, prevista em %.

Figura 37 – Histograma para a umidade relativa do ar esperada e umidade relativa do ar prevista



Fonte: Produção do Próprio Autor.

Figura 38 – Dispersão: umidade relativa do ar esperada vs. umidade relativa do ar prevista



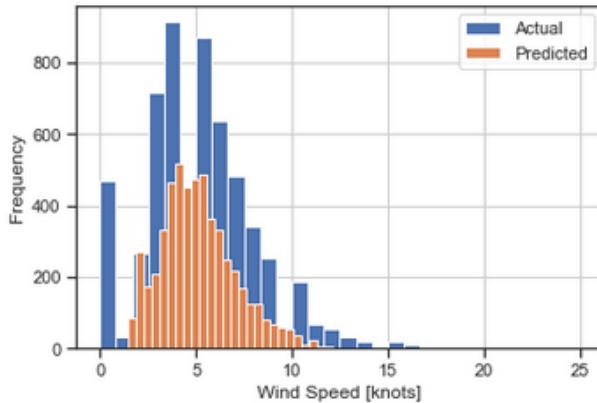
Fonte: Produção do Próprio Autor.

A Figura 37 mostra um histograma de frequência parcialmente mais divergente entre as variáveis prevista e esperada quando comparado ao mesmo gráfico apresentado para a temperatura do ar. Temos na dispersão (Figura 38) um comportamento mais linear entre os primeiros pontos (20%-60%), com posterior divergência ou aleatoriedade dos pontos no intervalo subsequente.

3.7.4.2.3 Velocidade do Ar

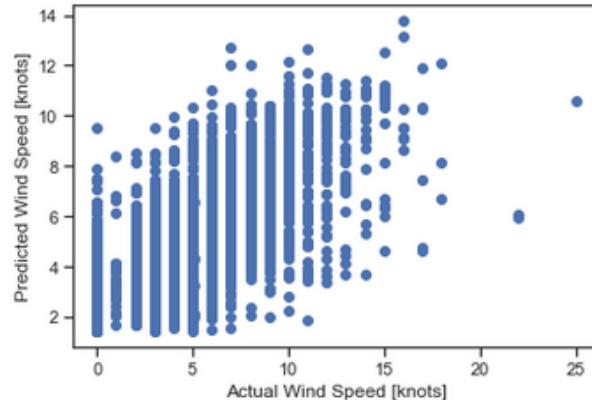
Nesta seção apresentam-se os resultados para a velocidade do ar, prevista em nós (ou *knots*).

Figura 39 – Histograma para a velocidade do ar esperada e velocidade do ar prevista



Fonte: Produção do Próprio Autor.

Figura 40 – Dispersão: velocidade do ar esperada vs. velocidade do ar prevista



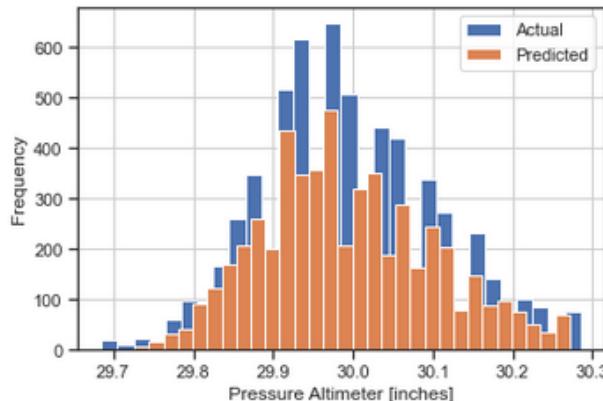
Fonte: Produção do Próprio Autor.

Podemos observar que, diferentemente do que foi apresentado para as variáveis anteriores, a velocidade do ar se mostra bem menos expressiva. Isso é bastante factível, pois a correlação obtida de $\sigma \approx 0.67$ na Figura 34 já indicou uma baixa colinearidade entre as variáveis esperada e prevista. Além disso, a dispersão observada na Figura 40 revela uma aleatoriedade bastante presente.

3.7.4.2.4 Pressão Barométrica

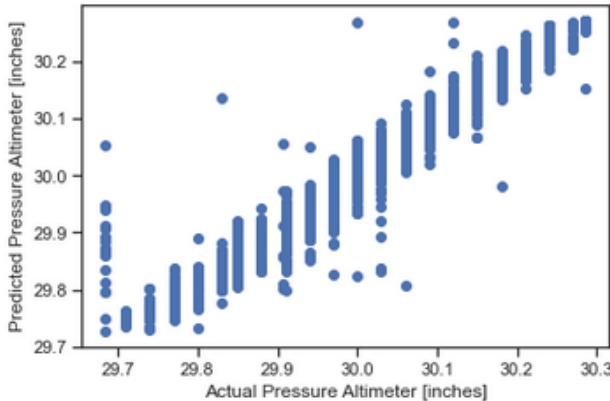
Nesta seção apresentam-se os resultados para a pressão barométrica do ar, prevista em $mmHg$.

Figura 41 – Histograma para a pressão barométrica do ar esperada e pressão barométrica do ar prevista



Fonte: Produção do Próprio Autor.

Figura 42 – Dispersão: pressão barométrica do ar esperada vs. pressão barométrica do ar prevista



Fonte: Produção do Próprio Autor.

No geral, a análise da pressão nos permite observar esta variável com o melhor resultado esperado (não só pela maior colinearidade apresentada na Figura 34, mas também pelo histograma da Figura 41). Além disso, podemos observar, novamente, o comportamento praticamente linear na dispersão desses dados (Figura 42).

3.7.4.3 Análise Distributiva dos Resíduos

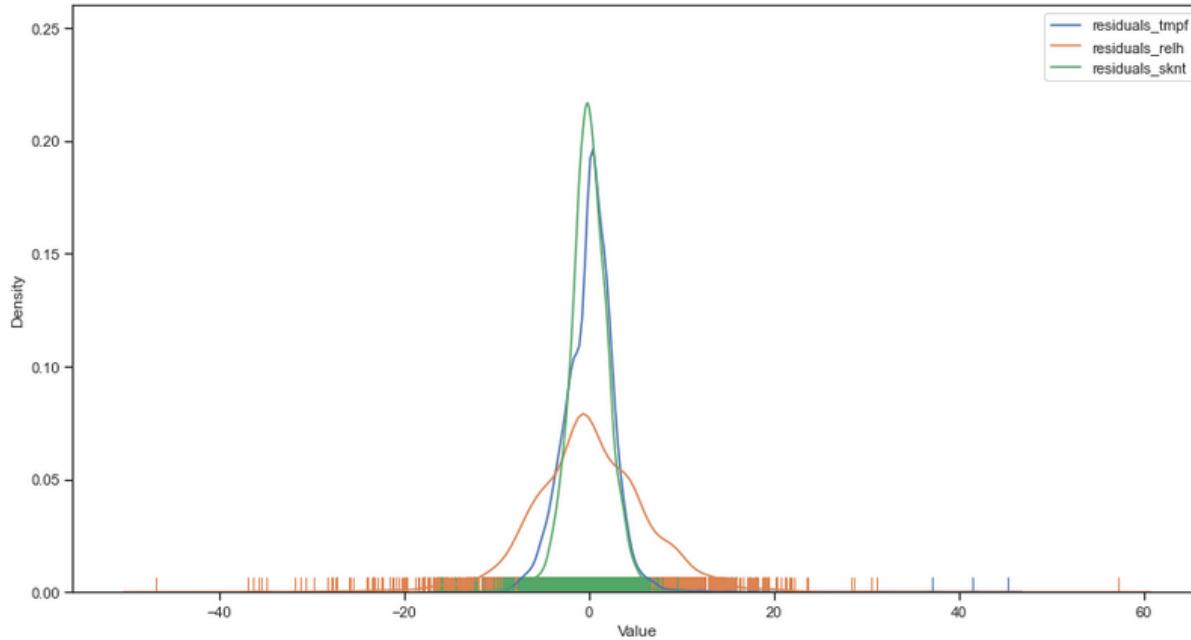
Avaliadas as métricas obtidas para cada uma das variáveis isoladas, podemos então entender como se comporta a distribuição residual (ou dos erros) para cada uma delas em conjunto. Os resíduos são apresentados como sendo:

$$\delta y = y_{true} - y_{pred} \quad (1)$$

em que y_{pred} é o valor previsto, y_{true} é o valor esperado ou real, e δy é o erro calculado ou resíduo a partir da diferença entre eles.

Desta forma foi traçada a curva de densidade populacional para as variáveis *tmpf*, *relh* e *sknt*, uma vez que a pressão barométrica apresentou a melhor resposta do modelo (sendo pouco suscetível a erros de previsão).

Figura 43 – Curvas de distribuição de densidade dos erros para algumas variáveis previstas em y



Fonte: Produção do Próprio Autor.

Analizando, por fim, a Figura 43, podemos observar que apesar dessas três variáveis apresentarem erros de previsão, suas distribuições estão praticamente centradas em 0, com a maior densidade populacional neste ponto. Isso evidencia que, no geral, o modelo mais acerta do que erra. Além disso, essas distribuições estão muito próximas de uma Gaussiana Normal, excluindo possíveis anomalias de distribuição destes dados.

3.8 MÓDULO 2: IDENTIFICAÇÃO DE PADRÕES DE ANOMALIA

3.8.1 Criação das Matrizes Adequadas ao Modelo

Para a criação das matrizes do modelo, foi feita uma tratativa de normalização ou escalonamento dos dados, que pode ser representada numericamente pela equação:

$$x'_{i,j} = \frac{x_{i,j} - \bar{x}_j}{\sigma(x_j)} \quad (2)$$

Nessa equação, \bar{x}_j e $\sigma(x_j)$ são, respectivamente, o valor da estimativa para a média e o desvio padrão da característica j . Essa tratativa tem o objetivo de centralizar todos os dados de uma dada variável em torno de sua média, fazendo com que seus novos valores sejam assumidos no domínio [-1,1].

Da modelagem adotada para a construção de um identificador de anomalias, foram estabelecidos os seguintes critérios para entrada e saída de dados:

- **input:** contem as variáveis, normalizadas, referentes às taxas temporais de variação de controle, isto é, $X = [rollRate, pitchRate, yawRate]$.

- **output:** contem uma resposta binária para as três variáveis lidas num dado instante, isto é, assume valor 0 quando não é identificado um padrão anômalo e 1 caso contrário.

Após criadas as matrizes principais, passou-se para a etapa de separação das mesmas em conjunto de dados de treino e conjunto de dados de teste. A relação estabelecida resultou num conjunto amostral de 25% dos dados para teste, sendo os outros 75% para ajuste do modelo.

3.8.2 Configuração do Modelo

Feitas as tratativas numéricas adequadas e selecionadas as matrizes, o próximo passo consistiu em arquitetar a rede neural para receber devidamente os dados e identificar possíveis padrões de anomalia durante o voo. As células adotadas dentro das camadas ocultas foram do tipo LSTM bidirecional e uma célula de ativação linear na saída. A Figura 44 apresenta um trecho do código extraído nessa etapa.

Figura 44 – Trecho do código implementado para construir a rede

```
#Building the model
model = Sequential()

model.add(Bidirectional(LSTM(units=180, input_dim=X_train.shape[-1], return_sequences=False)))
model.add(Dropout(0.2))

model.add(Dense(units=1))
model.add(Activation('linear'))

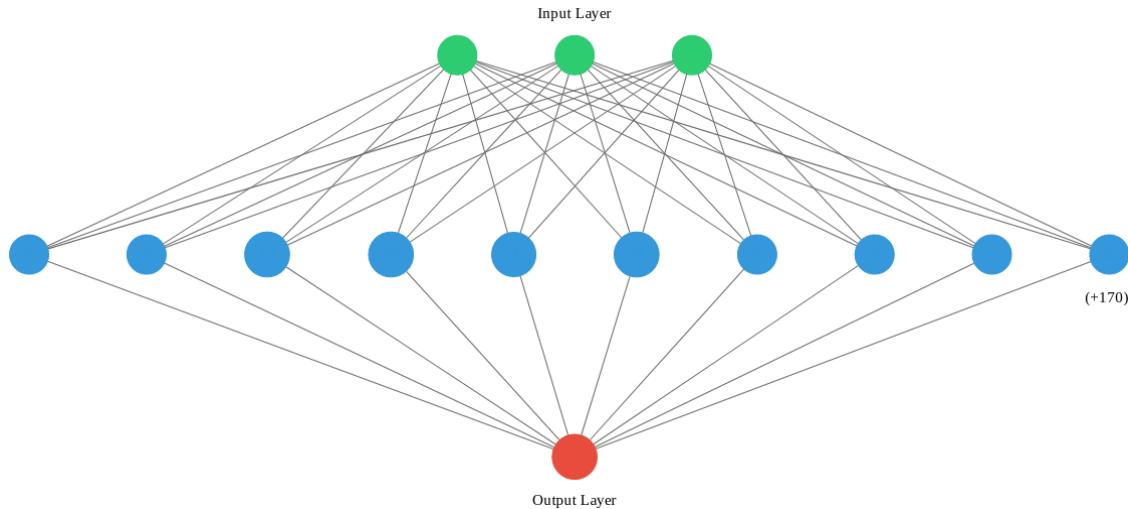
start = time.time()
model.compile(loss='mse', optimizer='rmsprop')
print('compilation time : {}'.format(time.time() - start))

compilation time : 0.010078668594360352
```

Fonte: Produção do Próprio Autor.

A rede neural devidamente construída pode ser visualizada a partir da Figura 45. Ela apresenta a estrutura adotada, sendo 3 células de entrada para captura das variáveis de *input* (taxas de *roll*, *pitch* e *yaw*), 180 células LSTM bidirecional na camada oculta com ativação do tipo *ReLU* (representando a janela de tempo utilizada para registro de 1 voo completo) e, por fim, uma única célula na camada de saída (ativação linear) para obter a resposta binária esperada.

Figura 45 – Exemplo da arquitetura da rede neural utilizada no processo



Fonte: Produção do Próprio Autor.

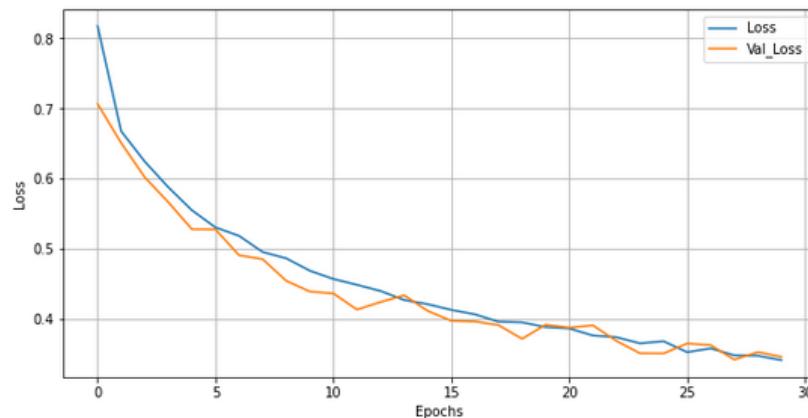
O ajuste foi realizado de acordo com os hiperparâmetros dispostos a seguir:

- **batch_size:** 18000 (adozido como sendo 50% dos dados, ou seja, utilizou-se o critério de *mini-batch*).
- **epochs:** 30.
- **validation_split:** 0.1 (representa o percentual de dados reservado para avaliação da performance do modelo em cada *Epoch*).

3.8.3 Resposta do Modelo

Durante o treinamento do modelo, foi obtida a curva de perda representada pela Figura 46.

Figura 46 – Decaimento da perda em função do número de Batch



Fonte: Produção do Próprio Autor.

Nela, podemos notar que *val_loss* é o valor da função de custo para os dados de validação cruzada, enquanto *loss* é o valor da função de custo para os dados de treinamento. A curva de perda da função custo ficou bem mais comportada, enquanto a perda pela validação cruzada teve uma grande variação de aumento e redução de valor durante as passagens pelos *Epochs*. Apesar disso, ao final ambas tiveram uma queda global, o que representa tendência de convergirem para um número de *Epochs* ideal.

No caso deste trabalho em específico não foi realizado o aumento para verificar a otimização dos hiperparâmetros devido ao alto custo computacional envolvido (não só pela volumetria dos dados, mas também pela arquitetura da rede como um todo). A resposta para identificação de padrões anômalos pode ser verificada na Tabela 8:

Tabela 8 – Métricas finais obtidas na matriz resposta do modelo

Valor Obtido	Classificação	Quantidade	Relação Percentual
0	Padrão Normal	33800	93.89%
1	Padrão Anômalo	2200	6.11%

fonte: Produção do Próprio Autor.

3.9 MÓDULO 3: CLASSIFICANDO ANOMALIAS EM SUBNÍVEIS PARA TOMADA DE DECISÃO

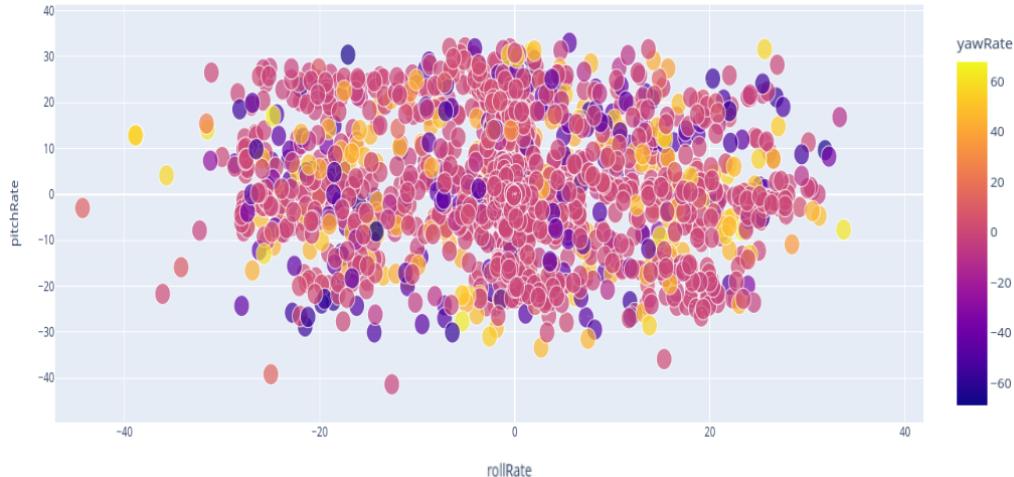
3.9.1 Criação das Matrizes Adequadas ao Modelo

Diferentemente do que ocorria com os demais algoritmos, e como foi abordado na parte teórica, algoritmos de clusterização são do tipo não-supervisionados. Isso significa que não teremos uma matriz de entrada (X) e uma na saída tida como *target* do nosso modelo (y). Aqui temos simplesmente um conjunto de dados de entrada para o algoritmo, e ele mesmo será capaz (de acordo com a implementação realizada) de entender o comportamento dos nossos dados e dizer quais serão os agrupamentos encontrados. Desta forma, foi adotado como critério para a entrada de dados todas as variáveis de voo descritas na seção anterior, incluindo - no entanto - uma coluna adicional referente à resposta binária da rede. Isso significa que para o treinamento e validação do modelo, utilizou-se o *output* da rede concatenada na matriz geral das variáveis de voo. Isso pode ser representado como sendo:

- **input:** contem todas as variáveis de voo incluindo a coluna *anomaly*, que representa a resposta binária (0,1) para o conjunto de *rollRate*, *pitchRate*, *yawRate* interpretados como sendo ou não um padrão anômalo num dado instante.

A próxima etapa, após a criação da matriz de entrada, foi uma breve análise exploratória dos nossos novos dados, porém menos voltada para uma visão estatística (como feito nas seções anteriores) e mais voltada para o entendimento de possíveis agrupamentos de dados pré-existentes.

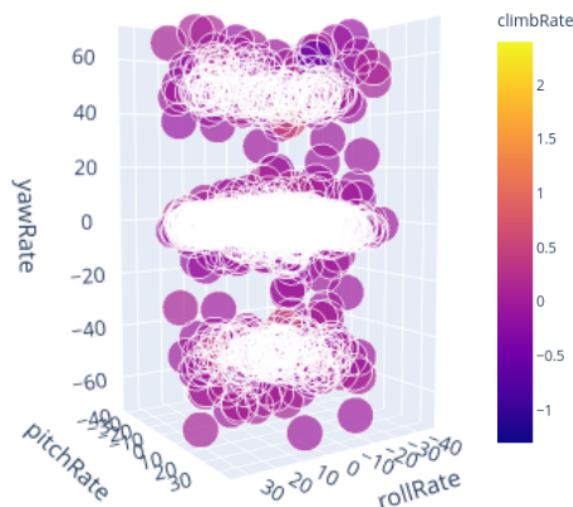
Figura 47 – Gráfico de dispersão 2D para as taxas temporais de roll, pitch e yaw



Fonte: Produção do Próprio Autor.

A Figura 47 não nos revela um agrupamento previamente definido para essa leitura de dispersão. No entanto, a escolha em representá-la é justificada pelo fato de serem as 3 variáveis necessárias para uma identificação binária de anomalia.

Figura 48 – Gráfico de dispersão 3D para as taxas temporais de roll, pitch e yaw mapeadas pela taxa temporal de subida



Fonte: Produção do Próprio Autor.

A Figura 48, no entanto, sugere um possível agrupamento de dados. Nela podemos notar (de forma não tão intuitiva) 3 agrupamentos distintos no domínio, representados pela densidade de dados no espaço mostrado.

3.9.2 Configuração do Modelo

Nesta seção foi adotado o modelo de *k-Means* através da transformação de escalonamento (ou normalização) dos dados, da mesma forma que ocorreu na seção anterior. A Figura 49 apresenta um trecho do código extraído nessa etapa.

Figura 49 – Trecho do código implementado

```

1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import MinMaxScaler
3 import plotly.graph_objects as go
4 X=df.anomaly
5 scaler = MinMaxScaler()
6 scaler.fit(X)
7 X=scaler.transform(X)
8 inertia = []
9 for i in range(1,11):
10     kmeans = KMeans(
11         n_clusters=i, init="k-means++",
12         n_init=10,
13         tol=1e-04, random_state=42
14     )
15     kmeans.fit(X)
16     inertia.append(kmeans.inertia_)
17 print(f'inertia: {inertia}')

```

inertia: [965.59128554924]
inertia: [965.59128554924, 518.1353133861223]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356, 350.5788179396271]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356, 350.5788179396271, 328.3147459673535]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356, 350.5788179396271, 328.3147459673535, 313.59550675383605]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356, 350.5788179396271, 328.3147459673535, 313.59550675383605, 292.84768548967736]
inertia: [965.59128554924, 518.1353133861223, 456.4315293974395, 399.33598317720214, 371.76571687890356, 350.5788179396271, 328.3147459673535, 313.59550675383605, 292.84768548967736, 280.0968100639065]

Fonte: Produção do Próprio Autor.

Os hiperparâmetros adotados foram:

- **n_clusters:** 3. Neste caso, escolheu-se 3 como sendo o número de *clusters* justamente por se desejar agrupar em 3 níveis distintos de anomalia, como abordado nas primeiras seções deste trabalho.
- **n_init:** 10. Número de vezes que o algoritmo é executado com *seeds* de centróides diferentes.
- **tol:** 1×10^{-4} . Tolerância relativa em relação à norma de Frobenius da diferença nos centros de cluster de duas iterações consecutivas para declarar a convergência.
- **random_state:** 42. É apenas uma *seed* utilizada para que possamos garantir que os resultados obtidos possam ser reproduzidos.

3.9.3 Respostas do Modelo

Após feitos os ajustes, as variáveis de entrada foram lidas e interpretadas pelo algoritmo, resultando em novas *labels* ou agrupamentos esperados para o nível de anomalia. Neste caso, criou-se um rótulo que identificaria o nível em específico, assumindo 0 para níveis leves, 1 para níveis moderados e 2 para níveis críticos. Parte do resultado é apresentado na Figura 50:

Figura 50 – Exemplo de tabela gerada como resposta contendo a coluna de labels criadas

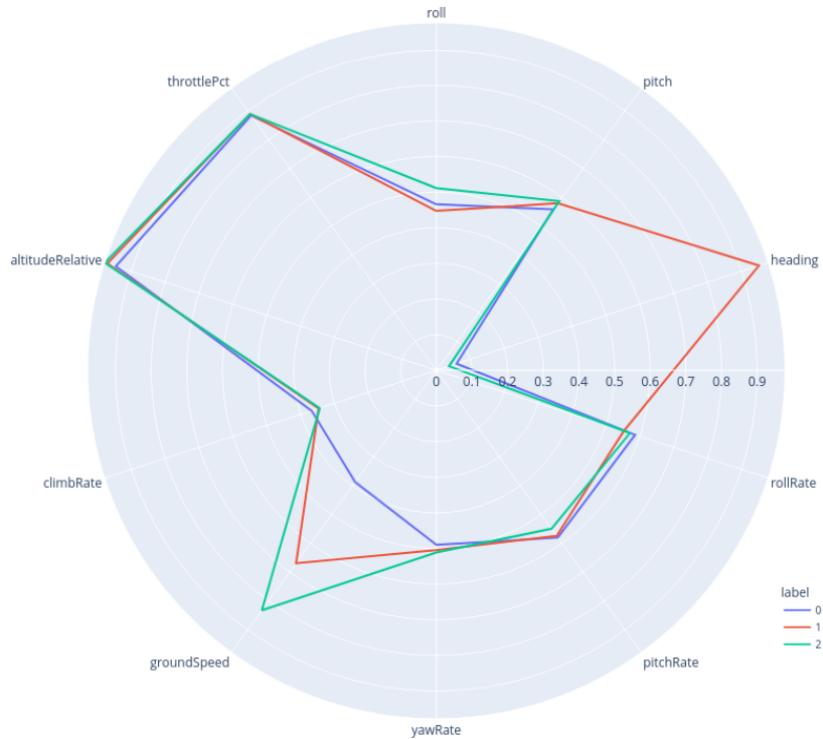
	roll	pitch	heading	rollRate	pitchRate	yawRate	groundSpeed	climbRate	altitudeRelative	throttlePct	label	
0	0.479029	0.731557	0.0	0.538462	0.591398	0.504392		0.74	0.346154	0.980392	0.9125	2
1	0.443709	0.844262	1.0	0.556410	0.641129	0.504392		0.14	0.384615	0.973856	0.9000	1
2	0.743929	0.504098	0.0	0.852564	0.813172	0.496340		0.42	0.346154	0.980392	0.9000	0
3	0.514349	0.528689	1.0	0.583333	0.522849	0.504392		0.84	0.346154	0.980392	0.8750	1
4	0.582781	0.573770	0.0	0.520513	0.491935	0.503660		0.40	0.346154	0.980392	0.8750	0
...	
2195	0.476821	0.872951	0.0	0.598718	0.254032	0.496340		0.90	0.346154	0.980392	0.9000	2
2196	0.472406	0.752049	1.0	0.564103	0.559140	0.502928		0.98	0.346154	0.980392	0.8875	1
2197	0.145695	0.573770	0.0	0.373077	0.501344	0.500000		0.54	0.346154	0.980392	0.9000	0
2198	0.322296	0.692623	1.0	0.566667	0.553763	0.502928		0.98	0.346154	0.980392	0.8875	1
2199	0.514349	0.506148	0.0	0.566667	0.552419	0.503660		0.42	0.346154	0.980392	0.8875	0

2200 rows × 11 columns

Fonte: Produção do Próprio Autor.

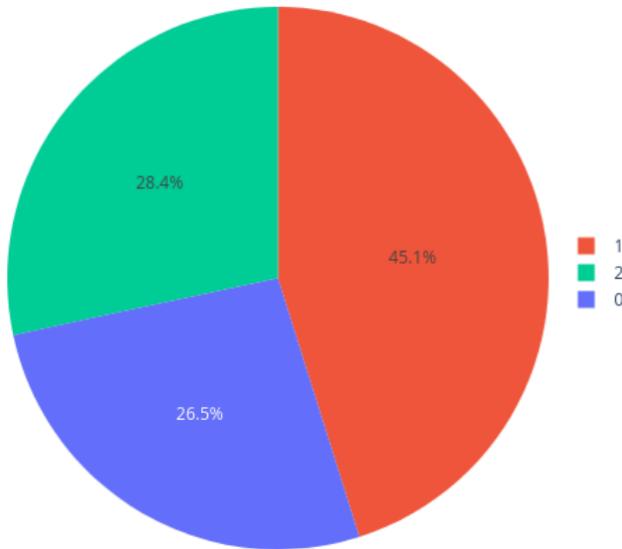
Podemos, ainda, estender essa análise de forma mais intuitiva. Para isso, foram gerados dois gráficos dispostos a seguir:

Figura 51 – Gráfico de radar mostrando a clusterização de anomalias com base nas variáveis de voo



Fonte: Produção do Próprio Autor.

Figura 52 – Gráfico setores mostrando a relação percentual obtida na clusterização



Fonte: Produção do Próprio Autor.

A Figura 51 nos dá uma ideia de densidade das variáveis com base nos subníveis criados, isso é, quais das variáveis estão mais relacionadas com uma *label* em específico. Podemos notar que *heading* apresenta maior percentual na categoria 1, enquanto na categoria 2 a variável *groundSpeed* é a que mais impacta nesse agrupamento. Por fim, a categoria 0 foi mais bem representada pela variável *altitudeRelative*. Além disso, a Figura 52 mostra que quase metade dos nossos dados iniciais foram agrupados como sendo do tipo 1 (ou anomalia moderada), ao passo que anomalias leves (categoria 0) e anomalias críticas (categoria 2) estão mais bem distribuídas, tendo uma massa de dados bem próximas uma da outra.

3.10 RESPOSTA CAÓTICA DO SISTEMA: GERANDO RUÍDOS NA BASE DE DADOS

Após construída toda a arquitetura do sistema (Figura 22), a última etapa consistiu em criar uma execução do programa de fácil interpretação pelo usuário. Deste modo, foi feita a leitura do conjunto de dados de voo extraídos da base original e verificado o comportamento do sistema através do terminal.

Figura 53 – Exemplo de execução do sistema através de um terminal

```

.....
Reading data: /
* rollRate:-69.035dg/sec
* pitchRate:-70.614dg/sec
* yawRate:225.696dg/sec

Found Anomalous Pattern!

>> Starting clustering...
* roll:57.722dg
* pitch:-50.405dg
* heading:54.0dg
* rollRate:-69.035dg/s
* pitchRate:-70.614dg/s
* yawRate:225.696dg/s
* groundSpeed:19.871m/s
* climbRate:1.426m/s
* altitudeRelative:15.0m
* throttlePct:71.0%

>> Main Cluster (k-Means):[1]
>> Individuals:
    roll      pitch heading  rollRate pitchRate  yawRate groundSpeed climbRate altitudeRelative throttlePct
0  critical   critical     ok  critical  critical  critical  critical  critical          ok          ok

Weighted Cluster: Critical Level

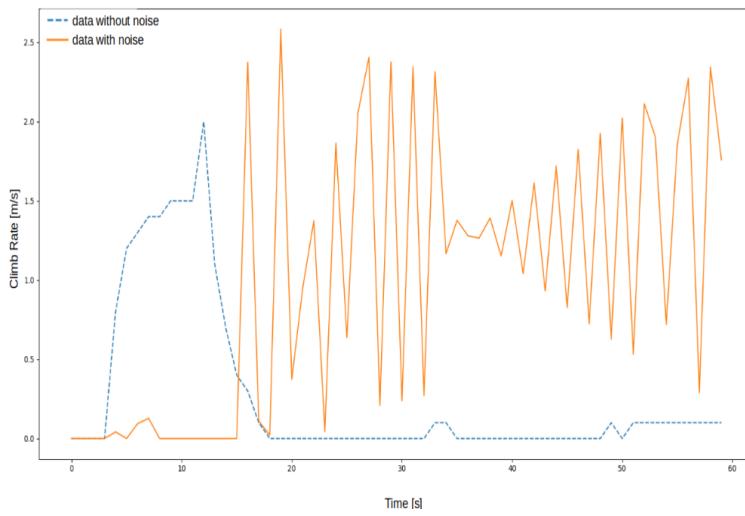
```

Fonte: Produção do Próprio Autor.

Nesta etapa também foi gerado um ruído nos dados afim de se verificar a estabilidade do sistema frente a comportamentos caóticos da aeronave. Os ruídos gerados foram baseados na equação de rajada do tipo 1-cosseno (WRIGHT; COOPER, 2008), porém de forma adaptada. A equação original é apresentada a seguir:

$$w_g(x_g) = \frac{w_{g0}}{2} \left(1 - \cos \frac{2\pi x_g}{L_g} \right), \quad 0 \leq x_g \leq L_g \quad (3)$$

Figura 54 – Exemplo de ruído gerado na taxa de subida da aeronave



Fonte: Produção do Próprio Autor.

O tipo de resposta esperada nessa geração de ruído é apresentado na Figura 54. Após aplicado o ruído na base de dados inicial, o sistema manteve a detecção de anomalias e a clusterização, sobretudo nos subdomínios nos quais essa resposta caótica (como na Figura 54) se manteve mais presente. Isso indica que a arquitetura se mantém estável, mesmo em situações mais extremas e menos prováveis de ocorrer.

3.11 TRABALHOS FUTUROS

Como fomento de futuros trabalhos a serem realizados para a continuação deste projeto, será necessário num primeiro momento a captura de dados de voo a partir de uma aeronave real (e não de simulações, como ocorreu anteriormente), retreino dos modelos e validação da estabilidade do sistema durante o voo.

A aeronave a ser utilizada para embarcar o sistema será o drone quadricóptero DJI Tello. Todas as informações referentes a esse modelo encontram-se no site: <<https://www.ryzerobotics.com/tello>>. A Figura 55 apresenta o modelo adotado como protótipo de testes futuros.

Figura 55 – Drone DJI Tello usado como protótipo para testes futuros



Fonte: Produção do Próprio Autor.

A aeronave em questão (Figura 55) possuí um protocolo de comunicação via *wifi*, o que permite uma versatilidade de implementação e utilização de módulos personalizáveis pelo usuário. A ideia para os próximos passos é criar uma missão autônoma para a aeronave e fazer a leitura das variáveis de voo em tempo real (de forma remota), ao passo que a arquitetura geral deste trabalho esteja funcional e executando em segundo plano, podendo abortar diretamente o voo caso seja necessário. Toda a documentação técnica encontra-se no próprio site do fabricante, incluindo exemplos de como executar missões de forma autônoma para o voo desta aeronave.

4 CONCLUSÃO

Por meio deste trabalho foi possível realizar um estudo com base na coleta de dados climáticos disponíveis em APIs de terceiros e dados gerados por simulação de voo. Em ambas as bases foi aplicado conceitos gerais de estatística pela análise descritiva dos dados, tratamento e manipulação dos mesmos através das técnicas discutidas anteriormente, bem como a geração de novas variáveis e obtenção de *insights* pela etapa de *Feature Engineering*.

Após treinados os algoritmos inerentes a cada uma das etapas, foi feita a arquitetura de cada um dos módulos individuais (módulo do pré-voo, módulo de identificação de anomalias e módulo para clusterização final dos dados), o que permitiu a tomada de decisão de ponta-a-ponta, isto é, decisão entre voar ou não pela resposta das variáveis climáticas e a decisão entre abortar ou não uma missão em tempo real dada a resposta dos níveis de anomalia obtidos. Além disso, neste trabalho avaliou-se também a estabilidade do sistema dada uma entrada caótica e verificou-se que a identificação de padrões anômalos ocorreu como esperado.

Por fim, conclui-se que os objetivos foram atingidos, tanto no âmbito da modelagem dos algoritmos, quanto na implementação de um sistema de planejamento de voo autônomo auto-suficiente e passível de ser continuado.

REFERÊNCIAS

- ALLBEE, B. **Hands-On Software Engineering with Python: Move beyond basic programming and construct reliable and efficient software with complex code.** [S.l.]: Packt Publishing Ltd, 2018.
- ALSMADI, M. K. et al. Performance comparison of multi-layer perceptron (back propagation, delta rule and perceptron) algorithms in neural networks. In: IEEE. **2009 IEEE International Advance Computing Conference.** [S.l.], 2009. p. 296–299.
- AN, D.; KIM, N. H.; CHOI, J.-H. Practical options for selecting data-driven or physics-based prognostics algorithms with reviews. **Reliability Engineering & System Safety**, Elsevier, v. 133, p. 223–236, 2015.
- BEN-GAL, I. Outlier detection. In: **Data mining and knowledge discovery handbook.** [S.l.]: Springer, 2005. p. 131–146.
- BENGIO, Y. Practical recommendations for gradient-based training of deep architectures. In: **Neural networks: Tricks of the trade.** [S.l.]: Springer, 2012. p. 437–478.
- BEST, K. L. et al. **How to analyze the cyber threat from drones: Background, analysis frameworks, and analysis tools.** [S.l.], 2020.
- BOX, G. E. et al. **Time series analysis: forecasting and control.** [S.l.]: John Wiley & Sons, 2015.
- BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, n. 2, p. 123–140, 1996.
- BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.
- CHEIN, F. Introdução aos modelos de regressão linear: um passo inicial para compreensão da econometria como uma ferramenta de avaliação de políticas públicas. Escola Nacional de Administração Pública (Enap), 2019.
- CHOLLET, F. **Deep learning with Python.** [S.l.]: Simon and Schuster, 2021.
- DRONECODE. **QGC - QGroundControl - Drone Control.** 2022. Disponível em: <<http://qgroundcontrol.com/>>.
- DU, K.-L.; SWAMY, M. N. **Neural networks and statistical learning.** [S.l.]: Springer Science & Business Media, 2013.
- DŽAKULA, N. B. et al. Convolutional neural network layers and architectures. In: SINGIDUNUM UNIVERSITY. **Sinteza 2019-International Scientific Conference on Information Technology and Data Related Research.** [S.l.], 2019. p. 445–451.
- EPPERSON, J. F. **An introduction to numerical methods and analysis.** [S.l.]: John Wiley & Sons, 2021.
- FENG, J. et al. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. **Physical Review E**, APS, v. 100, n. 3, p. 033308, 2019.
- FRENCH, R. M. The turing test: the first 50 years. **Trends in cognitive sciences**, Elsevier, v. 4, n. 3, p. 115–122, 2000.
- GAZEBO. **Gazebo.** 2022. Disponível em: <<https://gazebosim.org/home>>.

- GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.** [S.l.]: "O'Reilly Media, Inc.", 2022.
- GOSWAMI, B. et al. Abrupt transitions in time series with uncertainties. **Nature communications**, Nature Publishing Group, v. 9, n. 1, p. 1–10, 2018.
- GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In: SPRINGER. **European conference on information retrieval.** [S.l.], 2005. p. 345–359.
- IBM. **What is a Decision Tree?** 2022. Disponível em: <<https://www.ibm.com/topics/decision-trees>>.
- IBM. **What is Random Forest?** 2022. Disponível em: <<https://www.ibm.com/cloud/learn/random-forest>>.
- JAIN, A. K.; DUBES, R. C. **Algorithms for clustering data.** [S.l.]: Prentice-Hall, Inc., 1988.
- JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. **ACM computing surveys (CSUR)**, Acm New York, NY, USA, v. 31, n. 3, p. 264–323, 1999.
- JHA, A. K.; SATHYAMOORTHY, S.; PRAKASH, V. Bird strike damage and analysis of uav's airframe. **Procedia Structural Integrity**, Elsevier, v. 14, p. 416–428, 2019.
- JUPYTER. **Project Jupyter.** 2022. Disponível em: <<https://jupyter.org/>>.
- KOWALECZKO, G.; PIATKOWSKI, L. Estimation of extreme loads of the mi-24 helicopter during maneuvers using simulation method. **Preprints**, 2022.
- LAWRENCE, S.; GILES, C. L. Overfitting and neural networks: conjugate gradient and backpropagation. In: IEEE. **Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium.** [S.l.], 2000. v. 1, p. 114–119.
- LEARN scikit. **scikit-learn: machine learning in Python.** 2022. Disponível em: <<https://scikit-learn.org/stable/>>.
- MESONET, I. E. **IEM: Download ASOS/AWOS/METAR Data.** 2022. Disponível em: <https://mesonet.agron.iastate.edu/request/download.phtml?network=BR__ASOS>.
- MOLICK, E. Establishing moore's law. **IEEE Annals of the History of Computing**, IEEE, v. 28, n. 3, p. 62–75, 2006.
- NETTO, P. O. B. **Grafos: teoria, modelos, algoritmos.** [S.l.]: Editora Blucher, 2003.
- NGUYEN, H. et al. Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. **International Journal of Information Management**, Elsevier, v. 57, p. 102282, 2021.
- PAPERSPACEBLOG. **A Complete Guide to Decision Trees.** 2020. Disponível em: <<https://blog.paperspace.com/decision-trees/>>.
- POPESCU, M.-C. et al. Multilayer perceptron and neural networks. **WSEAS Transactions on Circuits and Systems**, World Scientific and Engineering Academy and Society (WSEAS) Stevens Point . . . , v. 8, n. 7, p. 579–588, 2009.
- QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.
- SADHU, V.; ZONOZ, S.; POMPILI, D. On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification. In: IEEE. **2020 ieee international conference on robotics and automation (icra)**. [S.I.], 2020. p. 5255–5261.
- SCIENCE, T. D. **Clustering: How to Find Hyperparameters using Inertia**. 2021. Disponível em: <<https://towardsdatascience.com/clustering-how-to-find-hyperparameters-using-inertia-b0343c6fe819>>.
- SERVICE, N. W. **AWC - Terminal Aerodrome Forecasts (TAFs)**. 2022. Disponível em: <<https://www.aviationweather.gov/taf/help?page=plot>>.
- SERVICES, A. W. **What is Boosting? Guide to Boosting in Machine Learning**. 2022. Disponível em: <<https://aws.amazon.com/what-is/boosting/>>.
- SHARMA, S.; SHARMA, S.; ATHAIYA, A. Activation functions in neural networks. **towards data science**, v. 6, n. 12, p. 310–316, 2017.
- TENSORFLOW. **API Documentation**. 2022. Disponível em: <https://www.tensorflow.org/api_docs>.
- TING, J.-A.; THEODOROU, E.; SCHAAAL, S. A kalman filter for robust outlier detection. In: IEEE. **2007 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.I.], 2007. p. 1514–1519.
- TWITTER. **Ryan Urbanowicz: New proposed field/term Venn diagram for an upcoming talk**. 2018. Disponível em: <<https://twitter.com/DocUrbs/status/1007375834347376642/photo/1>>.
- UZAIR, M.; JAMIL, N. Effects of hidden layers on the efficiency of neural networks. In: IEEE. **2020 IEEE 23rd international multitopic conference (INMIC)**. [S.I.], 2020. p. 1–6.
- WRIGHT, J. R.; COOPER, J. E. **Introduction to aircraft aeroelasticity and loads**. [S.I.]: John Wiley & Sons, 2008. v. 20.
- ZHAO, Y. et al. Deep residual bidir-lstm for human activity recognition using wearable sensors. **Mathematical Problems in Engineering**, Hindawi, v. 2018, 2018.