

UNIVERSIDADE DE SANTA CRUZ DO SUL  
DEPARTAMENTO DE ENGENHARIAS, ARQUITETURA E COMPUTAÇÃO  
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Maximiliano Meyer

**DESENVOLVIMENTO DE UMA FERRAMENTA DE IDENTIFICAÇÃO  
DE FRAUDES EM TRANSAÇÕES COM CARTÕES DE CRÉDITO COM  
USO DE *MACHINE LEARNING***

Santa Cruz do Sul  
2021

Maximiliano Meyer

**DESENVOLVIMENTO DE UMA FERRAMENTA DE IDENTIFICAÇÃO  
DE FRAUDES EM TRANSAÇÕES COM CARTÕES DE CRÉDITO COM  
USO DE *MACHINE LEARNING***

Trabalho de Conclusão apresentado ao Curso de Ciência da Computação do Departamento de Engenharias, Arquitetura e Computação da Universidade de Santa Cruz do Sul, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me. Ivan Luís Süptitz

Santa Cruz do Sul

2021

*Ao meu pai, que mesmo sem um estudo formal, foi o  
homem mais inteligente e admirável que conheci.*

*Parte da falta de humanidade de um computador é que,  
uma vez programado com competência e funcionando  
perfeitamente, ele é completamente honesto.*

...

*Eu não temo os computadores, temo a falta deles.*

*Isaac Asimov*

## RESUMO

Desde a popularização da internet a compra online tem se tornado um dos métodos preferidos daqueles que possuem acesso à rede, sobretudo em tempos de pandemia, período no qual o crescimento das vendas nesta modalidade disparou. Em meio a todas estas compras, o cartão de crédito se configura como o método mais utilizado para pagamentos. Hoje, são mais de 2,8 bilhões de cartões de crédito ativos no mundo que, juntos, movimentaram mais de 4,28 trilhões de dólares, em 2020. Com tanto dinheiro circulando de maneira digital, as tentativas de fraude envolvendo cartão de crédito crescem na mesma velocidade do aumento das transações. Para combater os fraudadores muitas são as técnicas utilizadas pelas instituições financeiras, com destaque aos algoritmos de *machine learning*. Assim, esta pesquisa buscou identificar o melhor algoritmo de *machine learning* para desenvolver uma solução capaz de auxiliar na identificação de transações falsas de cartão de crédito que pudesse ser implementada em ambientes reais. O algoritmo que apresentou os melhores resultados foi o *Extreme Gradient Boosting*, que utiliza uma técnica conhecida como *ensemble*, que consiste em combinar diversos algoritmos de predição menos poderosos, para no final gerar um resultado mais completo e acurado. Com a implementação deste algoritmo e da técnica de balanceamento de dados chegou-se a uma taxa de revocação de mais de 85% no *dataset* utilizado.

**Palavras chaves:** Machine learning, Fraude, E-commerce, Cartão de crédito

## LISTA DE FIGURAS

Figura 1 - O Diagrama de Venn da Data Science.....	22
Figura 2 - Exemplo do algoritmo K-nearest neighbors aplicado a um classificador de cores. 27	27
Figura 3 - Clássico exemplo do dataset Íris resolvido com Árvore de decisão.....	29
Figura 4 - Representação do funcionamento de um algoritmo de boosting aplicado a 3 árvores de decisão .....	31
Figura 5 - Representação da separação em grupos efetuada pelo Cross Validation .....	33
Figura 6 – Representação de uma Matriz de confusão .....	34
Figura 7 - Dataset de exemplo para compreensão das métricas de análise .....	35
Figura 8 - Matriz de confusão do exemplo proposto.....	35
Figura 9 - Procedimentos metodológicos .....	49
Figura 10 - Funcionamento da solução proposta dividida em etapas.....	52
Figura 11 - Configuração inicial do projeto .....	56
Figura 12 - Processo de remoção de colunas correlacionadas.....	57
Figura 13 - Processo de remoção de colunas não-numéricas .....	58
Figura 14 - Quantidade de registros NaN no dataset.....	58
Figura 15 - Processo de preenchimento dos valores NaN.....	59
Figura 16 – Criação do dataset após aplicação do pré-processamento.....	59
Figura 17 - Pipeline executado na etapa de pré-processamento dos dados .....	60
Figura 18 - Resultados do Random Forest com os valores padrão e distribuição 70/30.....	61
Figura 19 - Melhores resultados obtidos com o Random Forest.....	62
Figura 20 - Testes executados com o GridSearch CV .....	63
Figura 21 - Resultado do cruzamento realizado pelo Grid Search CV .....	64
Figura 22 - Valores otimizados de acordo com o Grid Search CV .....	65
Figura 23 - Resultados da classificação após otimização dos parâmetros .....	67
Figura 24 - Resultados do treinamento após utilização do Balanced Bagging Classifier .....	68
Figura 25 - Código utilizado para exportar como imagem uma das árvores de decisão criadas .....	69

Figura 26 - Comparação entre os métodos citados. Acima a aplicação da média, abaixo a utilização do zero.....	71
Figura 27 - Utilização da lógica Try, Except para carregamento dos arquivos .....	74
Figura 28 - Arquivo de configuração padrão da aplicação.....	75
Figura 29 - Parâmetros que serão testados utilizando o Grid Search CV e seus respectivos valores.....	76
Figura 30 – Função capaz de preparar rapidamente os registros para classificação .....	78
Figura 31 – Tempo necessário para executar 10 folds sobre um dataset de 200 mil registros.	79
Figura 32 - Estrutura inicial da aplicação .....	81
Figura 33 - Menu principal da aplicação .....	81
Figura 34- Etapa de pré-processamento concluída.....	82
Figura 35 - Resultado do treinamento do modelo utilizando a configuração padrão.....	83
Figura 36 - Tempo necessário para executar o Grid Search CV com 20 folds .....	85
Figura 37 - Resultado do treinamento do modelo utilizando a configuração otimizada.....	86
Figura 38 - Classificação final dos arquivos da pasta raiz .....	87
Figura 39 - Processo de comparação dos valores originais e preditos e o resultado da classificação.....	88
Figura 40 – Melhora após calibração do parâmetro de <i>max_depth</i> .....	89
Figura 41 - Tempo necessário para classificar pouco mais de 40 mil transações durante o teste .....	89

## **LISTA DE TABELAS**

Tabela 1 - Bibliometria quantitativa.....	43
Tabela 2 - Quadro comparativo das técnicas levantadas na bibliografia qualitativa.....	47
Tabela 3 - Valores finais utilizados para treinamento e classificação.....	90

## **LISTA DE ABREVIATURAS**

BBC	Balanced Bagging Classifier
CCM	Coefficiente da Correlação de Mathews
CSV	Comma Separated Value
ETL	Extraction Transform and Load
IDE	Integrated Development Environment
KNN	K-Nearest Neighbors
ML	Machine Learning
RF	Random Forest
XGB	Extreme Gradient Boosting

# SUMÁRIO

<b>1</b>	<b>Introdução .....</b>	<b>12</b>
1.1	Justificativa .....	14
1.2	Objetivos .....	16
1.3	Organização do texto .....	16
<b>2</b>	<b>Fundamentação Teórica .....</b>	<b>18</b>
2.1	Cartão de crédito e transação online .....	18
2.2	Fraude digital .....	20
2.3	<i>Data Science</i> .....	21
2.3.1	<i>Balanced Bagging Classifier</i> .....	24
2.4	<i>Machine Learning</i> .....	24
2.4.1	Naive Bayes .....	26
2.4.2	<i>K-nearest Neighbors</i> .....	27
2.4.3	Regressão Logística .....	28
2.4.4	Árvores de Decisões .....	28
2.4.5	<i>Random Forest</i> .....	30
2.4.6	<i>Extreme Gradient Boosting</i> .....	30
2.4.7	<i>Grid Search CV</i> .....	32
2.5	Métricas para análise .....	33
2.5.1	Matriz de confusão .....	33
2.5.2	Precisão .....	35
2.5.3	Revocação .....	36
2.5.4	Precisão x Revocação .....	37
2.5.5	<i>F1 Score</i> .....	37
2.5.6	Coefficiente de Correlação de Matthews .....	38
2.6	Considerações .....	38
<b>3</b>	<b>Metodologia.....</b>	<b>40</b>
3.1	Caracterização da pesquisa .....	40
3.2	Base de dados utilizada .....	40
3.3	Bibliometria Quantitativa.....	43
3.4	Bibliometria Qualitativa .....	43
3.4.1	<i>Credit card fraud detection using Machine Learning Techniques: A Comparative Analysis</i> 44	
3.4.2	<i>Random Forest for Credit card fraud detection</i> .....	45

3.4.3	<i>Credit card fraud detection using Machine Learning Algorithms</i> .....	45
3.4.4	<i>Credit card fraud detection using Pipeling and Ensemble Learning</i> .....	46
3.5	Quadro comparativo .....	47
3.6	Procedimentos Metodológicos .....	49
<b>4</b>	<b>Solução Desenvolvida</b> .....	<b>51</b>
4.1	Visão geral do sistema .....	51
4.2	Tecnologias utilizadas .....	52
4.2.1	Python .....	53
4.2.2	Scikit-Learn .....	53
4.2.3	Imbalanced Learn .....	53
4.2.4	Pandas .....	54
4.2.5	Matplotlib .....	54
4.2.6	Joblib .....	54
4.2.7	Pycharm .....	55
4.2.8	Google Colab .....	55
4.3	Aplicação do <i>Machine Learning</i> .....	56
4.3.1	Pré-processamento dos dados .....	56
4.4	Testes com o algoritmo <i>Random Forest</i> .....	60
4.5	Testes com Extreme Gradient Boosting .....	62
4.6	Criação do aplicação distribuível e seu funcionamento .....	71
<b>5</b>	<b>Testes e validação</b> .....	<b>80</b>
5.1	Testes .....	80
5.2	Análise dos resultados .....	87
<b>7</b>	<b>Conclusão</b> .....	<b>91</b>
	<b>REFERÊNCIAS</b> .....	<b>94</b>
	<b>APÊNDICE A – Árvore de decisão criada pelo algoritmo extreme gradient boosting</b> .....	<b>100</b>

## 1 Introdução

À medida que a sociedade se tornou mais conectada e dependente da Internet, as transações online passaram a fazer parte da vida das pessoas como algo normal, solidificando-se como um dos meios de consumo preferidos e aquele que mais ganha adeptos em tempos de pandemia. No entanto, da mesma maneira que métodos de pagamento e facilidades são criadas para atender a demanda dos consumidores, novas táticas de fraude surgem na mesma velocidade, senão mais depressa (CASTELLS, 2003).

Nesse contexto, o cartão de crédito, ferramenta de pagamento majoritária nas transações do tipo, aparece como o principal alvo para as fraudes (GERALDO e MAINARDES, 2017). Roubo de identidade, sites falsos para capturar os dados, quebra dos algoritmos de geração de cartões e códigos de verificação são alguns dos meios que os criminosos usam para consumir transações fraudulentas e lesar clientes e instituições financeiras. Porém, a mesma tecnologia que propicia um ambiente favorável às fraudes, também fornece as ferramentas que podem ajudar a combatê-las se empregadas de maneira correta.

Com o aumento a uma taxa exponencial no volume de dados que são produzidos de forma *online*, um novo campo surgiu recentemente no escopo da ciência da computação: a Ciência de Dados ou *Data Science*, que, através da análise de grandes quantidades de informação, vem, desde então, auxiliando no processo de tomada de decisão, nas mais variadas situações, da biologia ao mercado de ações. Foi com o surgimento desta nova área que, segundo Rautenberg e Carmo (2019), quantidades gigantescas de dados heterogêneos e complexos passaram a ser catalogados, organizados, interpretados e transformados em ações e ideias de grande valor agregado.

Como a interdisciplinaridade está intrinsecamente ligada à proposta da ciência de dados, o profissional da área utiliza diversas ferramentas e tecnologias para chegar ao resultado esperado, desde modelos matemáticos e estatísticos até os mais recentes algoritmos já de *machine learning* (CONWAY, 2013).

Os algoritmos, contudo, não são, nem de longe, tão recentes quanto a ciência de dados, na verdade, eles são o pilar da computação. Conceituados como uma sequência de instruções ordenadas que geram uma saída esperada, os algoritmos, há tempos, já são utilizados para realizar tarefas rotineiras e repetitivas, ou seja, aquelas que podem facilmente ser executadas

por uma máquina, desde operações matemáticas até movimentos de um robô instalado em uma linha de produção.

No entanto, mesmo com os avanços constantes em hardware e software uma série de problemas permaneceram impossíveis de serem resolvidos algoritmicamente. Como, por exemplo: dirigir um veículo, jogar jogos complexos como Go<sup>1</sup> ou movimentar-se por um ambiente lotado sem esbarrar em nada (ALPAYDIN, 2020).

Tais ações eram impróprias para os computadores até o surgimento dos algoritmos de *machine learning*, momento em que a ciência da computação entrou em uma nova fase de possibilidades e pôde ser capaz de replicar em algoritmos as tarefas até então exclusivamente “humanas”. Dentre as áreas que mais se beneficiaram com esta nova fase está a própria ciência de dados.

O conceito de *machine learning* reside no fato de que, através de um modelo generalista com incontáveis parâmetros e valores, o código possa “aprender” e aperfeiçoar sua técnica, cada vez mais, à medida em que recebe novas entradas de dados. Aprender, nesta situação, nada mais é do que ajustar as funções que fazem a análise dos dados de treinamento e que aprimora as funções de ativação. Logo, quanto maior for a quantidade de dados apresentados ao modelo, mais confiável serão os resultados de seu treinamento e avaliação (ALPAYDIN, 2020). Nesse sentido, a ciência de dados é o campo ideal para os algoritmos de *machine learning*.

Aplicando o conceito de aprendizagem de máquina especificamente à detecção de fraude, os dados de treinamento utilizados são centenas de milhares de registros de compras online efetuadas através do cartão de crédito todos os dias. Com dezenas de parâmetros para cada transação, é possível “ensinar” à máquina padrões comportamentais capazes de criar um perfil de compra para os usuários. É através destes padrões que identificar-se-á as transações que se assemelham àquelas já classificadas como fraudulentas, logo, potenciais tentativas de fraude. Com essa informação faz-se o bloqueio das mesmas antes que sejam processadas e possam gerar prejuízos.

---

<sup>1</sup> Go é um jogo de tabuleiro chinês criado há mais de 2.500 anos, considerado o mais difícil de ser dominado por uma inteligência artificial. Sua complexidade – muito maior do que o xadrez – se deve ao tabuleiro de 19 x 19 e a quantidade absurda de movimentos que precisam ser calculados. O número de jogadas possíveis no Go é de  $2.1 \times 10^{170}$  enquanto que o número de átomos existentes em todo o universo é de “apenas”  $1 \times 10^{80}$  (LEE, 2018). A primeira vez que um computador ganhou de um jogador profissional de Go foi em 2015.

É neste contexto que a ciência de dados e os algoritmos de *machine learning* têm contribuído para a segurança das transações virtuais, tornando-as cada vez mais resistentes a novas tentativas e novos tipos de golpe (SHUKUR e KURNAZ, 2019).

## 1.1 Justificativa

Manter-se imune às fraudes pode parecer uma tarefa árdua no contexto da Internet tendo em vista a velocidade com que novas técnicas maliciosas surgem. No entanto, com o avanço da tecnologia e com algoritmos cada vez mais precisos e inteligentes, medidas de prevenção igualmente dinâmicas e de rápida implementação, evolução e adaptação surgem para combater os fraudadores. Dentre esses algoritmos estão as técnicas de *machine learning*, que podem, literalmente, aprender com casos de fraudes anteriores para identificar tentativas futuras e, assim, impedi-las de serem efetuadas.

Segundo o Relatório Mundial de Pagamentos do ano de 2020, publicado pela empresa de consultoria francesa Capgemini, as transações efetuadas sem o uso de dinheiro físico, ou seja, efetuadas através de cartões de crédito/débito, transferências interbancárias e moedas digitais, cresceram, em média, 12,5% ao ano desde 2014, alcançando a marca de mais de 620 bilhões de transações em 2018. A mesma pesquisa prevê ainda que em 2023 o mundo ultrapassará a marca de 1 trilhão de transações digitais por ano<sup>2</sup> (WORLD PAYMENTS REPORT, pág. 23 - 25, 2020).

Quanto ao uso individualizado por país, o Brasil fica na quarta colocação em números absolutos de transações do tipo, perdendo apenas para Estados Unidos, China e Rússia. Os bons números brasileiros são decorrentes de medidas inovadoras do Banco Central que possibilitaram a criação e popularização de *fintechs*<sup>3</sup> como Nubank, Banco Inter e C6 que facilitaram o acesso ao cartão de crédito para pessoas que até então estavam fora do sistema bancário. Segundo pesquisa do SPC Brasil – Sistema de Proteção ao Crédito do Brasil, 40%

---

<sup>2</sup> Importante salientar que todas as projeções de crescimento deste capítulo foram baseadas em um mundo pré-pandêmico. Sabe-se que após a criação das regras de *lockdown* as vendas online tiveram um aumento significativo fazendo com que o cartão de crédito fosse mais utilizado do que nunca.

<sup>3</sup> Popularmente conhecidas como bancos digitais, as *Fintechs*, abreviação de *Financial Technology* (tecnologia financeira), são as startups ou empresas que têm como atividade a venda de produtos ou serviços financeiros totalmente digitais. A brasileira Nubank é a sétima maior startup do mundo, com um valor estimado em mais de 30 bilhões de dólares.

dos consumidores utilizaram cartão de crédito de alguma fintech entre 2018 e 2019 (SPC BRASIL, 2020a).

Por conta disto, o Brasil opera mais transações sem dinheiro do que países mais ricos, mais modernos e mais tecnológicos, como Coreia do Sul e Japão. Ainda assim, somos apenas o 24º na lista de transações por habitante, ou seja, há um mercado enorme a ser explorado – para o bem ou para o mal. Ainda sobre os números brasileiros, foram mais de 31 bilhões de transações do tipo em 2017, contra 28,9 no ano anterior – um crescimento de 7,2% no período (WORLD PAYMENTS REPORT, pág. 32 - 34, 2019).

Embora o crescimento anual de movimentações deste tipo já não tenha fôlego para assegurar uma taxa na casa dos 40 ou até mesmo 50%<sup>4</sup>, de todos os países analisados pela pesquisa, o Brasil é um dos quatro únicos nos quais a própria circulação de dinheiro físico mostra uma tendência de queda (WORLD PAYMENTS REPORT, pág. 38). Mas os números podem voltar a subir no país. Como anunciado no primeiro semestre de 2020, pelo Banco Central, até outubro de 2021 deve estar concluída a implementação do sistema público de informações bancárias. Com esta regulamentação os bancos deverão compartilhar entre si informações sobre seus clientes, permitindo que diferentes instituições possam oferecer serviços personalizados e mais em conta para correntistas de outras bandeiras. Com isso espera-se gerar mais concorrência e melhores condições aos clientes, além de, é claro, novas oportunidades para golpes e fraudes (PAGBRASIL, 2020).

Questão essa que os bancos já se preparam para lidar. Voltando rapidamente à pesquisa da empresa francesa Capgemini (2019, pág. 21), a segurança e o combate a fraudes foi citada como a segunda prioridade para os bancos dentre as iniciativas de transformação digital em 2019, perdendo apenas para os investimentos em *compliance*<sup>5</sup>.

Assim, considerando a importância e, sobretudo, a necessidade de um ambiente virtual mais seguro em um cenário de constante crescimento nas transações online, esta pesquisa busca responder a seguinte pergunta: De que maneira a *Data Science* e os algoritmos de *machine*

---

<sup>4</sup> Esta taxa vertiginosa ainda se mantém em alguns locais onde se verifica dois cenários distintos: quando um país possui uma população que ultrapassa a marca de 1 bilhão de pessoas e há muitos consumidores sendo, constantemente, introduzidos no comércio digital – como a Índia, que viu as transações digitais crescerem 51% em 2019; ou quando o país está em seu primeiro ou segundo ano de pagamentos digitais, como a Rússia que estreou recentemente o Mir, sistema doméstico de pagamentos sem dinheiro, e cresceu 42% também em 2019 (WORLD PAYMENTS REPORT, pág. 25, 2020).

<sup>5</sup> *Compliance* é o setor da instituição que trabalha para garantir que seja cumprida a legislação pertinente a fim de evitar processos judiciais e a aplicação de multas e sanções por parte do governo e órgãos reguladores. O *compliance* trabalha para evitar desvios tanto de funcionários como de clientes.

*learning* podem auxiliar no combate a um número cada vez maior de transações falsas nas compras online por meio de cartão de crédito?

Por conta disso, formulou-se o seguinte tema de pesquisa: A criação de uma solução que, através de algoritmos de *Machine Learning*, possa identificar fraudes em transações com cartões de crédito em tempo real.

## 1.2 Objetivos

Como objetivo geral, esta pesquisa busca desenvolver uma solução distribuível baseada nas técnicas de *machine learning* que seja capaz de identificar transações online fraudulentas de cartão de crédito em tempo real.

Para isso, pretende-se atingir os seguintes objetivos específicos:

- Estudar diferentes algoritmos de *Machine Learning*, utilizados em trabalhos relacionados, para identificar a melhor abordagem para o caso proposto.
- Modelar e desenvolver um sistema de previsão que possa ser aplicado não só academicamente como, também, em ambientes reais e que seja capaz de analisar os padrões comportamentais das transações e, a partir desta análise, calcular a probabilidade de uma operação de compra ser falsa.
- Desenvolver uma aplicação que possa ser distribuída e implementada em ambientes reais para uso com diferentes bases de dados.
- Realizar testes e validar o modelo desenvolvido utilizando uma base de dados real.
- Analisar o desempenho da solução implementada com diferentes métricas pertinentes ao escopo do projeto.

## 1.3 Organização do texto

O trabalho está organizado nos seguintes capítulos: O capítulo 2 aborda a fundamentação teórica dos temas pertinentes à pesquisa; o capítulo 3 descreve os procedimentos metodológicos definidos para este trabalho; o capítulo 4 contém a proposta de desenvolvimento da ferramenta e as tecnologias utilizadas; e o capítulo 5 apresenta a solução

desenvolvida e o capítulo 6 versa sobre os testes e as validações efetuadas e o capítulo 7 apresenta a conclusão da pesquisa desenvolvida.

## 2 Fundamentação Teórica

O referencial teórico desta pesquisa está dividido em 4 conceitos básicos: Cartão de crédito e transação online, fraude digital, *data science* e *machine learning*.

Esta última categoria será aprofundada sob a ótica de sete outros conceitos, a saber: Naive Bayes, K-Nearest Neighbors, Regressão Logística, Árvore de Decisões, Random Forest, Extreme Gradient Boosting e Grid Search CV. Estes são os algoritmos e métodos de tratamento de dados mais mencionados na bibliografia consultada.

### 2.1 Cartão de crédito e transação online

Segundo o Banco Central, no início de 2019 o Brasil tinha mais de 185 milhões de cartões de crédito ativos, uma média de 0,87 cartão para cada habitante (BACEN, 2020). São mais de 52 milhões de brasileiros com acesso à tecnologia, dos quais, quase a metade (47%) faz, pelo menos, uma transação com ele por mês (SPC BRASIL, 2020a). Embora não seja um número tão expressivo quanto o da população canadense, na qual mais de 82% dos habitantes com mais de 15 anos possui ao menos 1 cartão de crédito, os números brasileiros não devem ser ignorados (THE WORLD BANK, 2020).

Quando se analisa a nível mundial a quantidade de cartões em circulação é ainda mais impressionante: São 2,8 bilhões de cartões em atividade em 2020 – mais de 1 bilhão somente nos Estados Unidos, onde cada habitante tem, em média, 4 cartões e 14% tem, pelo menos, 10 cartões cada. Em 2017 os pagamentos com cartão superaram pela primeira vez os pagamentos com dinheiro, foram 13,2 contra 13,1 bilhões de transações, respectivamente. Um dos motivos é que, com o passar dos anos, o cartão de crédito está sendo utilizado para compras com valor médio cada vez mais baixo. No Reino Unido a média deste tipo de transação era de 66 Euros em 2008, caindo para 53 Euros 10 anos depois (SPEND JOURNAL, 2020).

A popularização dos cartões pode ser atribuída a uma série de motivos, como a possibilidade de diluir altos valores em pagamentos menores, comprar em qualquer lugar do mundo, seja em viagens ou através do comércio eletrônico, efetuar uma compra mesmo sem possuir dinheiro físico naquele momento e, por motivos de segurança. Há de se notar que a proteção se estende desde as compras físicas, principalmente após a introdução do cartão com

chip magnético, até as transações online, que registram taxas inferiores a 1% de fraude. Outro ponto importante em relação à segurança das compras com cartão de crédito é que o fato de portar dinheiro físico é inversamente proporcional à popularização dos cartões, diminuindo assim o roubo em áreas de insegurança (DHANKHAD; MOHAMMED; FAR, 2018).

Em alguns países, como os Estados Unidos - líder mundial na emissão de cartões, possuir um deles significa também ter um registro público junto às instituições de crédito sobre seu perfil pagador, ajudando no chamado *score de crédito*. De acordo com o *score* os bancos oferecem, para cada usuário, taxas de crédito mais ou menos atrativas em financiamentos de casas, carros, empréstimos e outros serviços (KORNEGAY; SCHWAB; DE ALMEIDA, 2009).

Embora tenhamos salientado que o cartão de crédito também se mostra um grande aliado das compras tradicionais, o principal responsável pelos números impressionantes acerca de seu uso é, sem dúvida, o comércio online que, nas últimas 2 décadas, vem mostrando uma taxa de crescimento superior as de qualquer outro setor. De acordo com a Associação Brasileira de Comércio Eletrônico, de 2003 a 2018 a categoria cresceu, em média, 33,18% ao ano no país (ABCOMM, 2019). A título de comparação, o Produto Interno Bruto (PIB) brasileiro teve um crescimento bruto de 2,36% no período (IPEADATA, 2021). Em 2019 foram mais de 19 bilhões de dólares em ordens de compra via comércio eletrônico, tanto nacionais quanto internacionais originadas do Brasil. Estes números colocam o país na 10ª posição dos maiores mercados de *e-commerce* do mundo (SHOPIFY, 2019).

Se analisado em escala global, em 2019, o volume de vendas online chegou a mais de 3,5 trilhões de dólares, tendo a China como maior mercado (SHOPIFY, 2019). Necessário salientar que todos esses números são pré-Covid-19. Com a pandemia e a necessidade das pessoas ficarem em casa, o *e-commerce* tem se mostrado como um dos únicos setores que está com números positivos. No Brasil o crescimento das vendas do primeiro semestre de 2020 foi de 47% em relação ao mesmo período do ano passado. Em valores foram 38,8 bilhões de reais faturados no mercado digital apenas nos seis primeiros meses do ano (EXAME, 2020).

Em levantamento feito após o fechamento do ano de 2020 confirmou-se o cenário de otimismo: mais de 4,28 trilhões de dólares transacionados no mundo através do comércio eletrônico; um crescimento de 27,6% em relação ao ano anterior (PAGSEGURO, 2021). Somente no Brasil, mais de 10 milhões de consumidores aproveitaram a situação atípica de pandemia para efetuarem sua primeira compra online (VIANA, 2021) enquanto que os clientes

habituais do comércio eletrônico não só compraram mais, como também passaram a comprar produtos que, até então, só eram adquiridos após o contato físico com o bem, como os móveis (JORNAL DO COMÉRCIO, 2021).

De acordo com o World Payments Report o número de consumidores que fazem mais da metade de todas as suas compras mensais de forma online quase dobrou desde o início da pandemia, e mais: a previsão é que essa transição do convencional para o *e-commerce* irá continuar mesmo após o vírus ser controlado. Não surpreendentemente, dentre esses consumidores, o cartão de crédito é o meio mais utilizado, sendo a primeira escolha para mais de 62% deste público (WORLD PAYMENTS REPORT, 2020, pág. 26 – 27).

Dessa forma, não é de se espantar que as transações online como um todo atraiam tantos fraudadores.

## 2.2 Fraude digital

Fraude pode ser definida como uma ação criminosa com o intento de obter vantagem indevida, na maioria das vezes, financeira.

Com um ecossistema bancário cada vez mais conectado, realizando mais operações e movimentando mais dinheiro do que nunca é natural que os fraudadores ataquem de modo pesado o meio. Estima-se que a perda de dinheiro para os fraudadores aumente a uma taxa de dois dígitos a cada ano (XUAN *et al.*, 2018, pág 1).

Em uma conta rápida tem-se noção do tamanho do “mercado”. Considerando que 7 em cada 10 dólares negociados online sejam enviados através de uma operação de cartão de crédito<sup>6</sup>, aproximadamente 3 trilhões de dólares são transacionados todo ano através deste meio de pagamento. Se, deste montante, os fraudadores conseguirem se apropriar de uma minúscula parte correspondente a 0,1%, o valor desviado nas transações fraudulentas será de cerca de 3 bilhões de dólares ao ano.

O número real, porém, é bem maior. Segundo a consultoria Nilson, as fraudes envolvendo pagamentos com cartão (seja ele de crédito, débito ou pré-pago) chegaram a 27,85 bilhões de dólares em 2018, um aumento de 16,2% em relação ao ano anterior (THE NILSON

---

<sup>6</sup> Não se encontrou um número exato para a porcentagem de compras online nas quais o pagamento é feito via cartão de crédito. Porém, como o boleto bancário é algo exclusivamente brasileiro pode-se considerar 70% como um número bastante conservador.

REPORT, 2019). Importante notar que neste valor não estão inclusos os prejuízos secundários, como despesas com as operações dos comerciantes, suporte, envios e manuseios, estornos, custos de investigação, entre outros.

De uma série de métodos possíveis para as fraudes com cartão de crédito, as mais populares entre os fraudadores são aquelas chamadas de *card-not-present fraud*, ou fraude de cartão não presente. Como o nome antecipa, este tipo de transação é aquela em que o cartão físico não é utilizado, como as compras online. Embora as transações deste tipo tenham representado menos de 15% de todas as operações com cartão no mundo, as transações de cartão não presente correspondem a 54% das fraudes ocorridas em 2018, de acordo com o mesmo relatório da Nilson. Em 2018 foram 6,4 bilhões de dólares em prejuízos, de acordo com a empresa de segurança digital ClearSale (2018).

No Brasil o cenário não é diferente. Em uma pesquisa produzida pela Confederação Nacional dos Dirigentes Lojistas e pelo Serviço de Proteção ao Crédito, 7,8 milhões de brasileiros foram vítimas de alguma forma de fraude financeira no período de 12 meses a começar em setembro de 2017. O problema mais comum é em relação aos cartões de crédito, reportado por 41% dos lesados (SPC BRASIL, 2018b)

Em meio a esse cenário de perdas bilionárias a saída é atuar com prevenção e análise de operações em tempo real para diferenciar as operações fraudulentas antes delas serem concretizadas. Com o volume de negociações na casa dos milhares por segundo, qualquer tentativa de avaliação e detecção humana é impossível. A resposta para este problema vem com o surgimento e a popularização da *data science* e dos métodos de detecção de fraude, que têm evoluído a passos largos nos últimos anos.

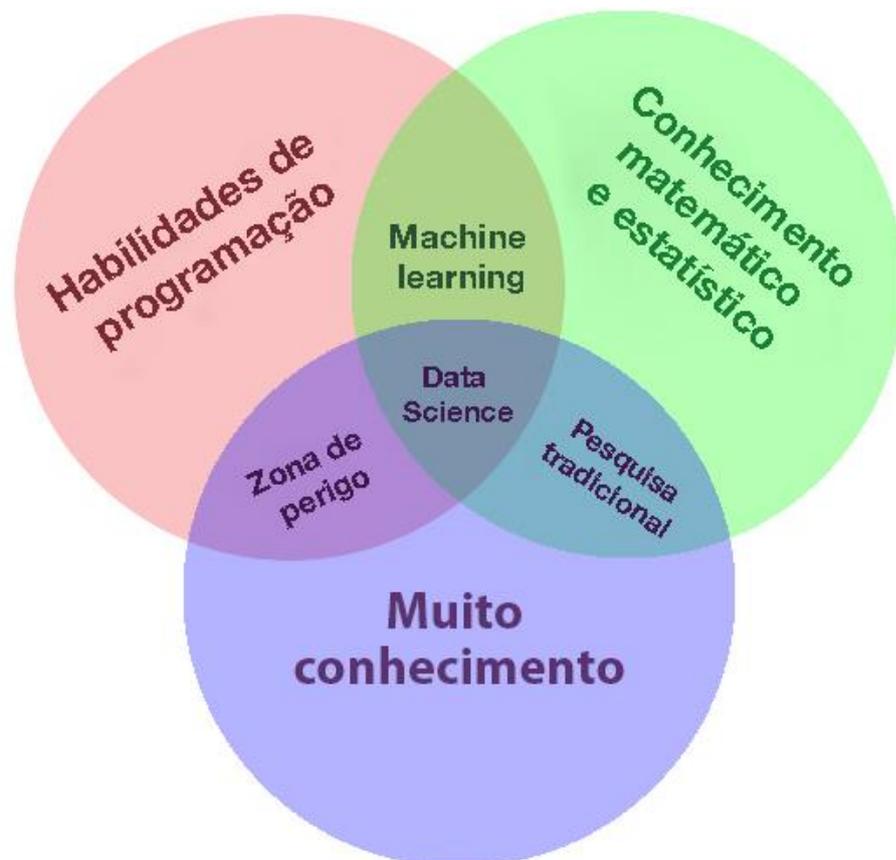
### **2.3 Data Science**

Nos últimos anos a *data science* tem se tornado um dos ramos mais populares das ciências exatas. O termo foi utilizado pela primeira vez em 1960 para se referir ao processamento de dados, quando a computação ainda não era um campo científico bem delimitado. No início ela era sinônimo de estatística, tanto que um dos estatísticos vivos mais famosos, Jeff Wu, perguntou no fim dos anos 90 se a estatística não deveria passar a ser chamada, justamente, de *data science* (GIBERT *et al.*, 2018, pág. 2).

O mundo fora da academia começou a explorar a área nos anos 90 quando a indústria encontrou nela uma excelente oportunidade para alavancar os negócios. A transição se deu em um momento que a internet começava a se popularizar nos Estados Unidos e as companhias, que tinham cada vez mais dados à sua disposição, perceberam que estavam desperdiçando uma quantidade gigantesca de informação que, com as ferramentas certas, poderia ser transformada em conhecimento para agregar valor aos seus negócios.

Multidisciplinar desde o seu início, Drew Conway popularizou em seu famoso diagrama de Venn (CONWAY, 2013) o que seria o resumo da *data science* moderna (Figura 1). Para ele, o campo é composto de três pilares: habilidades de programação, conhecimento matemático e estatístico e técnicas do conhecimento científico.

Figura 1 - O Diagrama de Venn da *Data Science*



Fonte: Traduzido de Conway (2013)

Entrando em detalhes sobre as zonas de intersecção descritas por Conway (2013):

- **Machine Learning:** Na convergência de habilidades de programação e conhecimento matemático e estatístico está o *machine learning*. Conway ressalta que para se aventurar na área não é necessário um doutorado em estatística ou ciência da computação, porém é necessário entender a língua dos hackers (o grupo “habilidades de programação” é referenciado como *hacking skills*, no original – ou habilidades de hacker) e saber o básico dos números e como interpretá-los.
- **Pesquisa Tradicional:** Quem passa a maior parte do tempo buscando conhecimento acadêmico e experiência com os métodos estatísticos, muito provavelmente, irá ter carência de novas tecnologias. Nesse caso é provável que fique restrito à pesquisa tradicional e acadêmica.
- **Zona de Perigo:** De acordo com Conway, estas são as pessoas que sabem demais ao ponto de serem perigosas. Pessoas que sabem obter e estruturar dados, porém, não conseguem compreender seus resultados acabam criando, seja por desconhecimento ou maldade, análises legítimas, mas sem qualquer entendimento de como foi que se chegou lá, e aí é que está o perigo. Por sorte, são necessários erros grotescos e propositalmente para que se possa adquirir tanto conhecimento sem que se aprenda matemática e estatística no decorrer da jornada. Assim, a zona de perigo acaba sendo pouco povoada, mas os que ali estão têm um potencial enorme para danos.

Assim, é inegável que, desde a metade dos anos 2000, a área goze de imensa popularidade, ao ponto da prestigiada *Harvard Business Review* (2012) apontar o cientista de dados como “o profissional mais sexy do século 21”. Caffo (2015), no entanto, citado por Gibert *et al.* (2018, pág. 6), é pragmático ao afirmar que a onda pode passar caso o cenário foque mais em “*data*” e não em tanto em “*science*”.

À medida em que mais dados têm sido produzidos e máquinas mais potentes são desenvolvidas, os conceitos e aplicações de *data science* têm sido estendidos a praticamente todos os ramos da ciência e da indústria. Agarwal e Dhar, mencionados por Gibert *et al.* (2018, pág. 2), apontam que a disponibilidade de *datasets*<sup>7</sup> cada vez maiores e mais complexos em conjunto com máquinas com poder computacional capaz não só para verificar hipóteses, mas

---

<sup>7</sup> Tradicionalmente como são chamados os conjuntos de dados utilizados como insumos básicos na área de *data science*.

também para analisar os dados e propor novas teorias está proporcionando uma “explosão de oportunidades para a pesquisa científica”.

É neste contexto que novas ferramentas começam a trabalhar paralelamente com a *data science* para a produção de melhores resultados. O melhor exemplo talvez seja, justamente, os algoritmos de *machine learning*.

Mas antes de adentrar neste tema é necessário fazer uma breve explicação sobre uma técnica que é fundamental para o desenvolvimento de pesquisas deste tipo.

### **2.3.1 *Balanced Bagging Classifier***

*Datasets* de transações de cartão de crédito são, devido à natureza de sua aplicação, desbalanceadas desde a sua origem, atingindo, com frequência 95% ou mais de registros pertencendo à classe de transações reais e o restante à classe de transações que apresentam fraudes. Assim, muitos algoritmos de *Machine Learning* (ML), principalmente os de viés classificatório, irão performar abaixo do esperado devido à escassez de transações fraudulentas e a consequente distorção no treinamento causada pela falta de registros desta classe.

Portanto, ao trabalhar com conjunto de dados deste tipo é inevitável a aplicação de alguma técnica que seja capaz de minimizar os problemas causados por tamanha disparidade. Na literatura o caminho mais seguro para este cenário são as técnicas de *resampling*, ou seja, técnicas de balanceamento da quantidade de registros entre as classes, seja criando dados sintéticos (chamado *oversampling*) ou redução de registros da classe dominante (chamado *undersampling*) no momento do treinamento do modelo para fins de diminuir a diferença entre elas e incrementar os resultados (KUMAR, 2020).

Neste trabalho será utilizada uma ferramenta chamada *Balanced Bagging Classifier* (BBC) incluída na biblioteca *Imbalanced Learn*, um pacote de soluções voltado a *datasets* desbalanceados (IMBALANCED-LEARN, 2021).

## **2.4 *Machine Learning***

Assim como a *data science*, o *machine learning* não é tão recente quanto se possa pensar. O termo foi cunhado por Arthur Lee Samuel, em 1959, ao conceituar o novo campo no

qual os computadores teriam a habilidade de aprender sem terem sido explicitamente programados para determinada tarefa (SAMUEL, 1959). Em sua pesquisa, ele ensinou uma máquina a jogar Damas ao mostrar a ela diversos registros de jogos que, ao serem analisados em conjunto, revelaram padrões que levavam à vitória ou à derrota. Em outras palavras, para citar aquela que talvez seja a definição mais famosa sobre *machine learning*:

Diz-se que um programa de computador aprende com a experiência E com respeito a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho nas tarefas em T, conforme medido por P, melhora com a experiência E (MITCHELL, 1997, Pág. 2).

As primeiras interações entre a *data science* e o *machine learning* começaram no início dos anos 80, sendo que, em 1985, Douglas Fischer e Bill Gale fundaram a *Artificial Intelligence and Statistics Society*. A ideia era facilitar a cooperação entre pesquisadores das áreas de inteligência artificial e estatística (GIBERT *et al.*, 2018, pág.2).

Classicamente, os algoritmos de *machine learning* podem ser classificados como tendo 3 tipos de aprendizado:

- A) **Aprendizado supervisionado:** No aprendizado supervisionado os dados de treino fornecidos ao algoritmo já estão classificados em relação à saída esperada. Quando uma nova amostra não conhecida pelo sistema é apresentada, a mesma será rotulada como uma das classes previamente apresentadas ao classificador baseada em critérios de similaridade. Dessa forma deve haver uma relação entre a entrada e a saída de dados (CARCILLO *et al.*, 2019, pág.1).
- B) **Aprendizado não supervisionado:** No aprendizado não supervisionado os dados fornecidos não possuem classificação sobre como a saída deve se parecer. Neste cenário busca-se encontrar os *outliers*, ou seja, os valores que saem dos padrões e que, portanto, podem indicar um valor fora do grupo de análise (CARCILLO *et al.*, 2019, pág.2). Outra possibilidade interessante do aprendizado não supervisionado é que ele permite separar os registros em clusters, gerando agrupamentos e mostrando padrões até então desconhecidos.
- C) **Aprendizado por reforço:** Diferentemente dos métodos anteriores, este tipo de aprendizado é direcionado a objetivos específicos, cujo treinamento reforça padrões determinados. Inspirada pelo comportamento psicológico, o aprendizado por reforço visa recompensar as melhores “ideias” durante o treinamento a fim de otimizar o resultado final (CORONATO *et al.*, 2020, pág.1).

Igualmente beneficiada com os avanços em hardware e poder computacional, os algoritmos de *machine learning* cresceram em complexidade e capacidade nas últimas décadas. Atualmente eles são utilizados nas mais variadas áreas da ciência, sendo empregados na resolução de todos os tipos de problemas possíveis, com ênfase naqueles que utilizam quantidades gigantescas de dados.

Na pesquisa bibliográfica realizada para esta pesquisa, e citada nas seções seguintes, levantou-se as técnicas de ML mais utilizadas na detecção de fraudes com cartão de crédito.

### 2.4.1 Naive Bayes

Naive Bayes é um dos mais populares algoritmos de *machine learning*. Ele classifica os novos registros através de uma abordagem probabilística baseando-se na premissa de que todos os atributos são independentes entre si e que a análise do conjunto irá indicar a qual classe pertence o exemplo (CHEN *et al.*, 2019, pág. 2).

Um exemplo bastante difundido na literatura para explicar o algoritmo *Naive Bayes* é a sua utilização para classificação de e-mails como sendo spam ou não-spam. Neste exemplo, imagine que 10 e-mails deverão ser classificados quanto à sua natureza. O primeiro passo é separar todas as palavras do e-mail: “Olá”, “amigo”, “bom”, “dia”, “dinheiro” e assim por diante. No próximo passo, cada uma dessas palavras será “pesada” de acordo com a base de dados supervisionada, por exemplo, pela lógica, “dinheiro” terá mais chances de fazer parte das palavras que denotam uma mensagem de spam do que a palavra “amigo”. Após treinado o modelo, cada nova mensagem recebida será analisada pelo classificador e, com base em suas palavras, terá determinada a probabilidade de ser spam ou não.

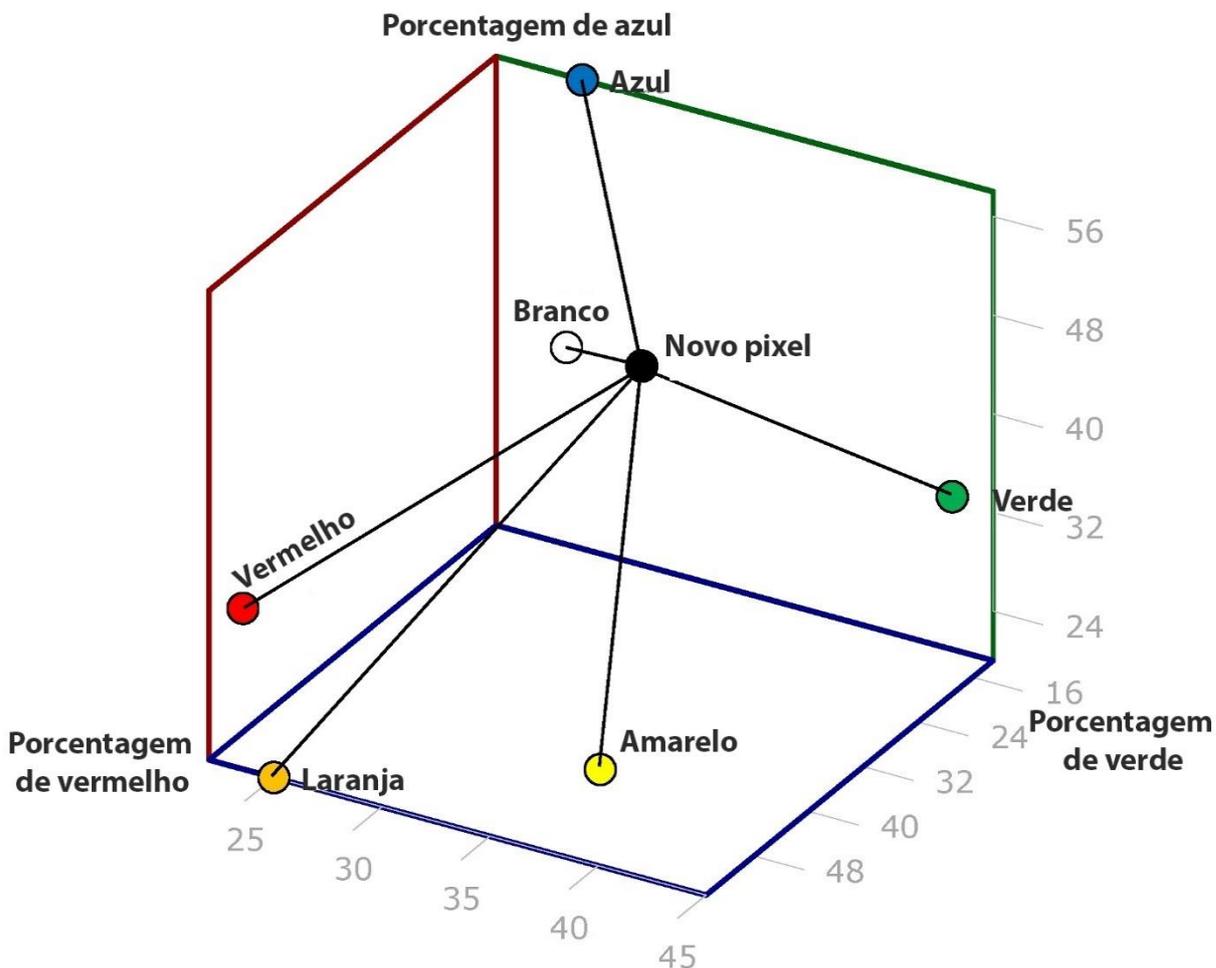
A grande crítica para este algoritmo é que ele não leva em conta a interrelação entre os atributos analisados. Se o objeto forem frutas e não e-mails, por exemplo, o algoritmo não irá considerar “maçã”, “redonda”, “vermelha” e “10 cm de diâmetro” como havendo uma relação entre si, tratando cada um dos termos independentemente. O nome do algoritmo (*naive* significa “ingênuo” em inglês) vem justamente do fato dos atributos de não serem inter-relacionados no momento da análise (BECKER, 2019).

### 2.4.2 K-nearest Neighbors

Diferentemente do algoritmo Naive Bayes, o *K-Nearest Neighbors* (KNN) tem como principal característica a similaridade entre os registros como tomada de decisão para a classificação. O algoritmo parte do pressuposto de que dados similares, ou seja, de uma mesma classe, tendem a estar próximos uns dos outros quando colocados em um plano de dispersão de dados (ZAREAPOOR, 2019, pág. 3).

Em um exemplo prático, extraído de José (2018), imaginemos um classificador de cores de bolinhas. Neste modelo fictício, diversas cores estão classificadas com base em seus valores RGB, ou seja, sua composição de vermelho (*Red*), verde (*Green*) e azul (*Blue*) e dispersas em um plano em três dimensões, tal qual ilustra a Figura 2.

Figura 2 - Exemplo do algoritmo K-nearest neighbors aplicado a um classificador de cores.



Fonte: Traduzido de José (2018)

Quando uma nova bolinha, de cor desconhecida, for submetida à classificação, o modelo irá analisar sua composição RGB e determinar a qual grupo pertence, de acordo com a similaridade dos valores já conhecidos.

Esta classificação por similaridade é feita através de um cálculo de distância, que pode ser distância Euclidiana, distância de Manhattan, distância de Minkowski, entre outras, que gera como saída o quão próximo aquela nova entrada (bolinha de cor desconhecida) está para cada uma das classes. Com base neste valor e em “K”, que delimita os limites de distância mínima para os grupos e que dá nome ao algoritmo, define-se a qual conjunto pertence o novo valor.

### 2.4.3 Regressão Logística

A regressão logística é um algoritmo que tem como objetivo prever a probabilidade de uma variável categórica encaixar-se em uma lógica discretizada de verdadeiro ou falso através de um conjunto de variáveis<sup>8</sup>. A partir do modelo gerado com esta técnica, e dada uma observação aleatória, é possível calcular a probabilidade de um determinado evento ocorrer contra a probabilidade de um evento não ocorrer, o peso de uma variável dependente sobre o conjunto de variáveis fornecido, entre outras aplicações (GONZALEZ, 2018, pág. 15 - 16).

Uma das principais vantagens da classificação por regressão logística é que os *outliers* deixam de ser um problema, já que a classificação foca na região de fronteira (BAGGA *et al.*, 2020, pág. 3).

### 2.4.4 Árvores de Decisões

Uma árvore de decisão é um algoritmo que, com base em um conjunto de dados, traça um mapa das possíveis decisões a serem tomadas formando um caminho de decisões cascadeadas baseado nas probabilidades de algum evento ocorrer.

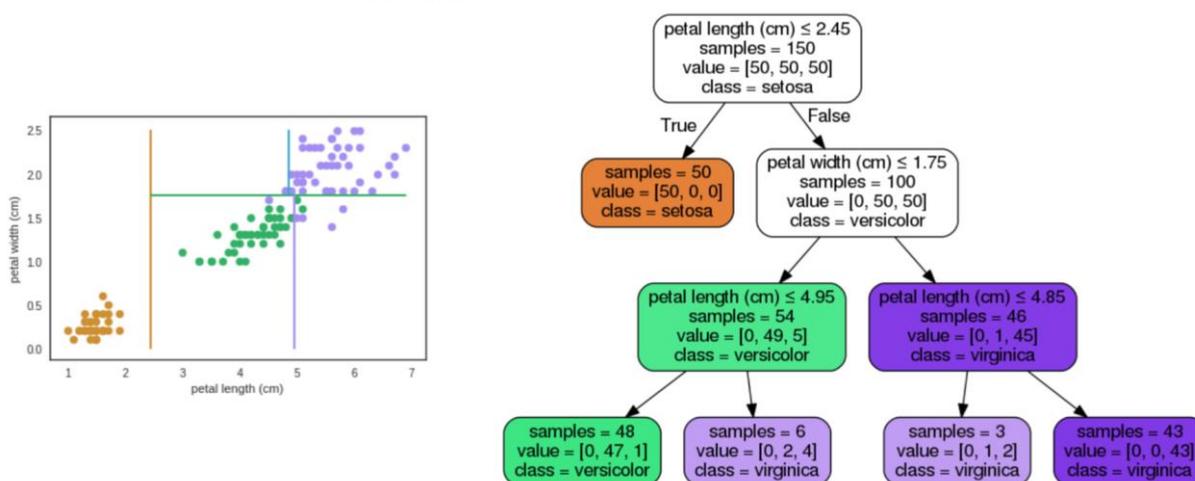
---

<sup>8</sup> Ao menos a regressão logística binomial que faz classificações binárias e é o tipo utilizado para classificações de fraude. Porém, existe também a regressão logística ordinal, que trabalha com classificações em categorias ordenadas (se o atendimento em uma loja foi ruim, médio ou bom, por exemplo) e a regressão logística multinomial, na qual os objetos são classificados em categorias que não tem relação entre si (o animal da foto é um cachorro, gato ou elefante, por exemplo) (GATEFY, 2021).

A árvore começa através do nó raiz e vai se ramificando em novos caminhos embasada na análise de eventos consolidados, até que uma nova divisão já não faça diferença estatística na distribuição e no cálculo da probabilidade (SAHIN e DUMAN, 2011, pág. 3).

Quando um novo registro é submetido a uma árvore para classificação ele avança do topo em direção às folhas seguindo um caminho de distribuição entre os nodos de acordo com os valores de suas variáveis até encontrar seu o seu valor estatístico, tal qual ilustra a Figura 3.

Figura 3 - Clássico exemplo do dataset Íris resolvido com Árvore de decisão



Fonte: Campos (2017)

De fácil compreensão devido à sua visualização intuitiva distribuída em níveis, por conseguir trabalhar com problemas multi rotulados, por conseguir lidar nativamente com valores faltantes, categóricos e numéricos, por normalmente não exigir a normalização dos dados, entre outras características, a árvore de decisão é algoritmo muito popular para problemas de classificação. Sua maior desvantagem reside na sensibilidade a novas entradas, já que a adição de um único novo registro de treinamento pode fazer com que toda a estrutura da árvore inteira seja alterada.

Para contornar este problema foi criado a variação da árvore de decisão tradicional: o Random Forest.

### 2.4.5 *Random Forest*

Similar às árvores de decisão, o *Random Forest* (RF) conta com uma particularidade que faz dele um algoritmo completamente distinto: neste modelo não é criada apenas uma única árvore com uma única distribuição, mas sim várias árvores de decisão menores, cada qual com seus caminhos únicos que podem ser, ou não, bastante diferentes entre si (XUAN, 2018, pág. 2).

Através de uma seleção aleatória de amostras do *dataset* de treino – e das variáveis que serão utilizadas para o cálculo das ramificações – o RF cria  $n$  árvores de decisão que seguem a mesma lógica do algoritmo anterior: começa com um nó raiz e vai se distribuindo em nós folhas até serem exauridas as possibilidades. A quantidade de árvores criadas é definida de acordo com as características do problema, sobre a qual deve-se analisar o tipo de dados a serem trabalhados, o volume do *dataset*, o problema que se deseja resolver, entre outros pontos.

Como o RF possui  $n$  possibilidade de classificação, quando um novo registro é submetido à validação, ele irá percorrer todas as árvores criadas, obtendo um valor de saída para cada uma. O resultado final será então a média de todas as saídas individuais destas árvores (XUAN, 2018). Assim, além de possuir um grau de confiança maior nos resultados, possui um algoritmo menos custoso em capacidade computacional já que, ao invés do consumo necessário para a criação de uma única e dispendiosa árvore, há  $n$  árvores menores que podem ser criadas com o auxílio de processamento paralelo.

### 2.4.6 *Extreme Gradient Boosting*

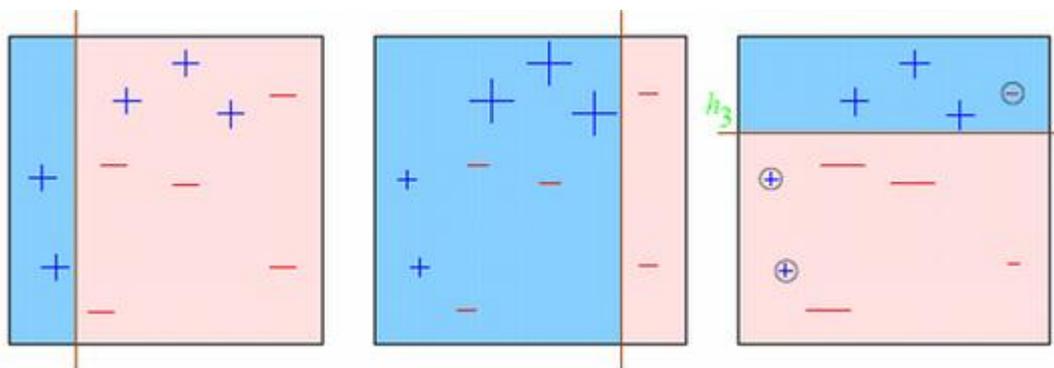
A principal característica das técnicas classificadas como algoritmos de *boosting* é o fato dos mesmos utilizarem o princípio de *ensemble* em suas abordagens, ou seja, a combinação de diferentes e menos poderosos algoritmos de ML para a criação de uma predição mais completa e acertada no final do processo. Na prática os algoritmos deste tipo funcionam como árvores de decisão sequenciais já que o valor que foi predito em  $n$  irá ser levado em conta para a predição em  $n+1$ .

Aplicando ao exemplo da predição de fraude: se a primeira árvore concluir que determinada transação é uma transação falsa, mas o rótulo a classificar como legítima, esta transação irá para a árvore seguinte, criada aleatoriamente com novas colunas e, portanto,

composta de uma nova lógica de classificação, com pesos e valores diferentes. Se nesta nova árvore a transação for classificada corretamente, o modelo aprendeu a lidar com esta variação nos dados, senão, ela passa para uma terceira árvore na qual o modelo irá novamente tentar entender suas particularidades, e assim por diante. Dessa maneira o modelo consegue aprender melhor com os registros que está errando e ficar mais acurado do que quando treinado de maneira convencional (ESTATIDADOS, 2019).

A Figura 4 sintetiza este processo. Nela é possível ver que a primeira árvore classifica corretamente os 2 sinais azuis à esquerda (posicionados na região de mesma cor do quadrado) e os 5 sinais vermelhos (posicionados na respectiva seção de sua cor), errando a classificação dos demais sinais azuis. Ao passar para a segunda árvore, os pesos das classificações corretas anteriormente são atenuados em relação aos pesos dos que foram erroneamente classificados, representado por um sinal gráfico menor e maior, respectivamente. Realiza-se uma nova rodada de classificação com base nestes novos pesos atribuídos e assim sucessivamente. No final do processo, os resultados das médias é que será a saída do algoritmo.

Figura 4 - Representação do funcionamento de um algoritmo de *boosting* aplicado a 3 árvores de decisão



Fonte: Jain (2016)

Além da expressiva melhoria nos resultados desde o primeiro uso, uma das principais vantagens dos algoritmos deste tipo é a quantidade de parâmetros que podem ser otimizados durante o processo de treinamento. Nesta pesquisa utilizou-se o *Extreme Gradient Boosting* (XGB) que possui mais de 80 variações em sua configuração e que podem – e devem – ser alterados em busca da combinação que apresente os melhores resultados. Entre estas otimizações estão o controle de nós, a profundidade da árvore, a taxa de aprendizagem, entre outros parâmetros que serão detalhados no capítulo 5, referente aos testes executados.

Além da questão da otimização de parâmetros o XGB possui outros pontos que fazem dele uma excelente escolha para problemas de classificação, por exemplo: técnicas nativas de resistência a *overfitting*<sup>9</sup>, capacidade nativa de lidar com valores faltantes, processamento paralelo, poda automática da árvore quando ocorrer um cenário de perda  $n$  níveis abaixo, entre outros pontos. Atualmente o XGB goza de uma grande popularidade, tanto pelos motivos elencados acima como por ter sido a ferramenta vencedora em várias competições online no Kaggle<sup>10</sup>.

No entanto, o vasto leque de personalização oferecido por seus parâmetros também pode ser considerado – senão um ponto negativo – um ponto de atenção, já que não há uma lista de quais parâmetros devem ser explorados e quais devem ser mantidos em sua configuração padrão. A solução é efetuar várias rodadas de combinações e, através da tentativa e erro, encontrar o melhor cenário para o problema. Porém, há ferramentas que podem auxiliar.

#### 2.4.7 *Grid Search CV*

O *Grid Search CV* é um módulo da biblioteca *Scikit Learn*<sup>11</sup> utilizado para automatizar o processo de otimização de parâmetros que o XGB faz necessário. Além de criar tantas simulações quanto forem necessárias através de um cruzamento de dados, a ferramenta também é capaz de avaliar o desempenho de cada um destes arranjos utilizando uma métrica definida pelo usuário.

Para utilizá-lo basta definir quais serão os parâmetros testados, quais serão os valores possíveis para cada cenário e o número de *Cross Validations* (CV) a serem executados. Também chamado de *folds*, do inglês “dobras”, este valor representa o número de divisões em que o *dataset* será particionado. A técnica consiste em gerar  $n$  recortes de mesmo tamanho com o objetivo de alcançar uma maior variedade de cenários de treinamento e teste em uma simulação. Por exemplo: Se utilizarmos um conjunto de dados com 5 mil registros e definirmos

---

<sup>9</sup> *Overfitting* é o termo usado para descrever um modelo que foi treinado com tamanha exatidão ao ponto de ficar “viciado” nos dados de treinamento e não conseguir corresponder nos dados de teste.

<sup>10</sup> Kaggle é uma comunidade voltada para cientistas de dados e entusiastas de *machine learning* que foi fundada em 2010 e adquirida pelo Google em 2017, quando já contava com mais de 1 milhão de usuários registrados. No site são disponibilizados milhares de *datasets* para estudo, cursos, competições promovidas por empresas que buscam soluções para problemas reais, entre outras funcionalidades.

<sup>11</sup> Descrita no subcapítulo 4.2.2.

o valor de CV como sendo 5, então este *dataset* será dividido em 5 partes de 1000 registros cada. Na sequência estes 5 conjuntos de dados serão submetidos a treinamento e validação alternando qual conjunto está sendo utilizado como teste até que todas as combinações sejam executadas, conforme ilustra a Figura 5. A precisão final do modelo é então calculada através de um cálculo apropriado (UYEKITA, 2019).

Figura 5 - Representação da separação em grupos efetuada pelo *Cross Validation*

### Cross Validation em 5 grupos

Combinação 1	Teste	Treino	Treino	Treino	Treino
Combinação 2	Treino	Teste	Treino	Treino	Treino
Combinação 3	Treino	Treino	Teste	Treino	Treino
Combinação 4	Treino	Treino	Treino	Teste	Treino
Combinação 5	Treino	Treino	Treino	Treino	Teste

Fonte: Autores (2021)

Embora a implementação do *Grid Search CV* seja bastante simples, há de se ter cuidado para não extrapolar na quantidade de valores e parâmetros a serem testados e cair em um gargalo de horas de execução a cada rodada de testes. Essa questão será melhor aprofundada no seção 4.3 do capítulo 4, que abordará os testes executados com a ferramenta.

## 2.5 Métricas para análise

Tão importante quanto um modelo executando de maneira satisfatória é um modelo devidamente avaliado para que se possa extrair dos resultados os *insights* mais proveitosos à pesquisa. Assim, através de levantamento bibliográfico identificaram-se as métricas mais utilizadas no contexto desta pesquisa.

### 2.5.1 Matriz de confusão

Embora não seja um método de avaliação dos resultados por si só, a matriz de confusão é o conceito basilar para a aplicação e o entendimento de todas as métricas que serão utilizadas.

A matriz de confusão consiste em uma tabela com 4 valores:

- **Verdadeiro Positivo:** Quando o modelo previu corretamente a classe positiva.
- **Falso Positivo:** Quando o modelo previu positivo, mas o correto seria negativo.
- **Verdadeiro Negativo:** Quando o modelo previu corretamente a classe como negativa.
- **Falso Negativo:** Quando o modelo previu negativo, mas o correto seria positivo (MULLER; GUIDO, 2016, pág 266).

Graficamente, a matriz de confusão é representada da seguinte maneira, como ilustra a Figura 6:

Figura 6 – Representação de uma Matriz de confusão

		Valor verdadeiro (Valor real)	
		positivos	negativos
Valor Previsto (saída do algoritmo)	positivos	<b>VP</b> Verdadeiro Positivo	<b>FP</b> Falso Positivo
	negativos	<b>FN</b> Falso negativo	<b>VN</b> Verdadeiro Negativo

Fonte: Autores (2021)

Ilustrando a aplicação de uma tabela e confusão, temos uma situação hipotética na qual um determinado algoritmo foi treinado para prever se, baseado em  $n$  variáveis, uma mulher está grávida ou não. Os dados reais e a previsão do algoritmo estão exemplificados na Figura 7, onde 0 corresponde a não-grávida e 1 corresponde a grávida:

Figura 7 - Dataset de exemplo para compreensão das métricas de análise

Dados reais = [0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0]  
 Previsão = [0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0]

Fonte: Autores (2021)

Com estes dados monta-se a matriz de confusão, conforme a Figura 8:

Figura 8 - Matriz de confusão do exemplo proposto

		Valor verdadeiro (Valor real)	
		grávida	não-grávida
Valor Previsto (saída do algoritmo)	grávida	<b>VP</b> <b>6</b> Verdadeiro Positivo	<b>FP</b> <b>3</b> Falso Positivo
	não-grávida	<b>FN</b> <b>3</b> Falso negativo	<b>VN</b> <b>8</b> Verdadeiro Negativo

Fonte: Autores (2021)

Com este cenário em mente será mais fácil compreender os parâmetros utilizados para avaliação descritos nas próximas seções.

### 2.5.2 Precisão

Nesta métrica calcula-se quantas previsões foram feitas de maneira correta dentre todas as que foram classificadas, independente da classe, no exemplo acima, se grávida ou não-grávida (MULLER; GUIDO, 2016, pág. 270).

Tornando mais concreto: Das classificações que nosso modelo fictício apresentou sobre estar grávida, a precisão é de 66% já que das 9 conclusões feitas como tal, 6 estão corretas.

Porém, se analisarmos a precisão considerando os resultados sobre a classe não-grávida, nossa precisão é de 72%, pois dos 11 valores classificados como não grávida, 8 estão corretos.

$$Precisão = \frac{VP}{VP + FP}$$

Onde VP são os valores verdadeiros positivo e FP os falsos positivos.

A precisão é importante quando o objetivo é eliminar os falsos positivos. Se em nosso cenário fictício muitas mulheres forem classificadas como grávidas, mesmo não estando, por exemplo, todas aquelas que foram erroneamente indicadas como tal passariam pela carga psicológica e emocional de pensarem estar gestando e depois descobrirem ter sido vítimas de um alarme falso (NICHOLSON, 2021). Neste exemplo a precisão seria de grande ajuda.

### 2.5.3 Revocação

Com a revocação – ou *recall* – avalia-se a porcentagem dos valores pertencentes a uma determinada classe que foram identificados dentre todos as possibilidades do *dataset* (MULLER; GUIDO, 2016, pág. 108), ou seja, busca diminuir os falsos negativos.

Assim, a revocação é muito importante no contexto das fraudes bancárias. Em um cenário real, cada transação fraudulenta classificada como verdadeira pelo modelo desenvolvido irá causar prejuízos à instituição financeira, logo, mitigar os falsos negativos é essencial para um modelo deste tipo (SHUNG, 2018).

Por exemplo, a revocação em relação a estar grávida, no nosso exemplo, foi de 66%, já que dos 9 possíveis resultados corretos para a classe, nosso algoritmo fictício identificou 6. Quando analisado sobre a classe não-grávida, a métrica é de 72% já que foram identificados 8 dos 11 possíveis resultados positivos da classe.

$$Revocação = \frac{VP}{VP + FN}$$

Onde VP são os valores verdadeiros positivo e FN são os falsos negativos.

## 2.5.4 Precisão x Revocação

Normalmente existe uma relação inversa entre precisão e revocação, na qual, ao aumentar um valor, diminui-se o outro. Em mais um exemplo fictício para fins de melhor compreensão do conceito, imagine a situação de um médico que precisa remover o tumor cerebral de seu paciente. Se ele remover apenas as células tumorais e não remover as células cancerígenas, o tumor pode voltar a se desenvolver a partir das células remanescentes, porém, ele não pode remover células de maneira displicente para garantir a remoção total do tumor, senão pode prejudicar a função cerebral do paciente.

Caso ele decida agir com mais ousadia, para garantir que todas as células cancerígenas tenham sido removidas, aumentará a sua revocação (pegando mais células de câncer), porém, diminuirá sua precisão (pegando células saudáveis no entorno das células cancerígenas). Por outro lado, se ele for conservador e remover apenas aquelas células que tem certeza serem malignas estará aumentando sua precisão, mas diminuindo a revocação (QAZ, 2021).

Por conta desta relação de antagonismo entre as métricas, muitas vezes, as medições de precisão e *recall* não são discutidas isoladamente, mas comparadas a um valor fixo da outra medida ou então combinadas em um único cálculo.

Assim, mesmo que precisão e revocação sejam métricas bastante utilizadas em pesquisas com *datasets* desbalanceados, como será melhor detalhado na seção 3.4 do capítulo 3, algumas métricas mais específicas podem ser utilizadas para uma melhor compreensão dos dados, como as que serão detalhadas a seguir.

## 2.5.5 F1 Score

Muito utilizado em *datasets* deste tipo, o *f1 score*, também chamado apenas de *f score*, é uma métrica que combina os resultados de precisão e revocação para calcular a exatidão do modelo. Em termos gerais, o *F1 Score* calcula a média harmônica entre os valores de precisão e revocação (NICHOLSON, 2021).

Desta forma, quanto menos falsos positivos e menos falsos negativos nas métricas anteriores, melhor será o resultado do *F1 Score*, sendo que 0 representa um modelo totalmente ineficaz e 1 um modelo perfeito.

$$F1\ Score = 2 * \frac{Precisão * Revocação}{Precisão + Revocação}$$

### 2.5.6 Coeficiente de Correlação de Matthews

O Coeficiente de Correlação de Matthews (CCM) é mais uma métrica a ser empregada na interpretação dos resultados mesmo quando as classes do *dataset* são desbalanceadas. Nesta métrica os valores assumidos ficam entre 1 (resultado perfeito) e -1 (pior resultado possível) (SHMUELI, 2019).

Segundo Chicco e Jurman (2020) o CCM é ainda melhor do que o F1 Score pois adiciona ao cálculo uma relação de equilíbrio entre as categorias da matriz de confusão.

$$CCM = \frac{VP * VN - FP * FN}{\sqrt{(VP + FP) * (VP + FN) * (VN + FP) * (VN + FN)}}$$

Onde VP são os valores verdadeiros positivo, VN os verdadeiros negativo, FP os falsos positivos e FN os falsos negativos.

## 2.6 Considerações

Com base nos dados apresentados neste capítulo conclui-se que o comércio eletrônico e as transações online estão passando por um momento de, até então, inimaginável crescimento, devido à pandemia do Coronavírus, tanto em número de vendas e valores transacionados como em confiança e adesão de novos consumidores, adeptos ao comércio tradicional. Contudo, conclui-se também que este cenário acaba por ser atrativo também para os fraudadores, que se aproveitam do mesmo momento para incrementar seus ganhos.

Hoje, uma das ferramentas mais poderosas para ajudar no combate às fraudes online de cartão de crédito é, sem dúvidas, a utilização dos algoritmos de *machine learning*. Através destas técnicas é possível explorar todo o poder computacional moderno e treinar uma máquina, através da análise de uma extensa base de dados, para que ela seja capaz de identificar padrões

comportamentais associados aos hábitos dos consumidores. Após “aprender” quais são os padrões associados às transações fraudulentas, o modelo pode calcular a probabilidade de qualquer transação ser uma tentativa de golpe e bloqueá-la em tempo real.

Um ponto a ser ressaltado nas considerações sobre a fundamentação teórica, que talvez chame a atenção de qualquer um que já tenha trabalhado com *data science*, é a falta da métrica Acurácia, que nesta pesquisa foi deixada de lado pelo fato de que, ao utilizarmos um conjunto de dados altamente desbalanceado, a métrica acabaria por levar a conclusões enganosas na análise dos resultados. Como será descrito na seção 3.2, do capítulo 3, nosso *dataset* possui mais de 96,5% dos registros classificados como transações reais. Desta forma, se desenvolvermos uma aplicação que classifique todas as transações como verdadeiras – incluindo aquelas que são fraudulentas – teremos uma acurácia de 96,5%. Ao olharmos para este valor, poderíamos dizer que o modelo está performando muito bem, quase atingindo a perfeição, porém, ao analisarmos um pouco mais a fundo veríamos que a aplicação desenvolvida seria completamente inútil.

O motivo é simples: Para a instituição financeira, o foco de uma ferramenta de detecção de fraudes é, rigorosamente, identificar as que possam vir a ser fraudulentas, ou seja, que podem vir a se tornar um prejuízo para a instituição caso sejam efetivadas e, portanto, precisam ser bloqueadas. Assim, um modelo com impressionantes 96,5% de acurácia, mas que falha em identificar, justamente, os 3,5% para o qual foi desenvolvido, seria um modelo péssimo para a situação, embora com números que aparentam um modelo quase perfeito.

A partir destes algoritmos, em especial o *Extreme Gradient Boosting*, esta pesquisa, desenvolveu uma ferramenta que tenha a capacidade de ser treinada para captar padrões comportamentais, baseada em grandes volumes de dados. Na sequência, através destes padrões, a solução é capaz de analisar transações em tempo real e determinar a probabilidade dela ser fraudulenta ou legítima.

### 3 Metodologia

Esta seção descreve a metodologia e os procedimentos metodológicos definidos para a pesquisa.

#### 3.1 Caracterização da pesquisa

A pesquisa é exploratória e descritiva, sendo constituída, inicialmente, de um levantamento do *dataset*, levantamento bibliográfico sobre as diferentes técnicas de *machine learning* que podem ser empregadas na detecção de fraudes, técnicas de pré-processamento de dados características e este tipo de pesquisa e as métricas ideais para aferimento dos resultados obtidos.

A natureza da pesquisa é tanto quantitativa como qualitativa. Quantitativa, pois diversas métricas matemáticas foram utilizadas para análise de dados, padrões, comportamentos e taxas de acerto da solução desenvolvida. Resultados e análises estatísticas também foram produzidas, direcionando ainda mais a pesquisa para o aspecto quantitativo. Qualitativa já que com base nos dados brutos e nos resultados, diversas análises mais subjetivas e menos exatas foram realizadas durante a pesquisa.

Também foi realizada bibliometria quantitativa e qualitativa para levantamento e síntese de trabalhos relacionados ao tema proposto.

#### 3.2 Base de dados utilizada

Para a parte prática desta pesquisa (treinamento e validação dos resultados) foi utilizado um *dataset* público<sup>12</sup>, com dados reais coletados da plataforma de pagamentos digitais Vesta<sup>13</sup>,

---

<sup>12</sup> Disponível em <https://www.kaggle.com/c/ieee-fraud-detection/overview>

<sup>13</sup> Vesta foi a primeira empresa no mundo a oferecer transações totalmente digitais para companhias de telecomunicação. Fundada em 1995, nos Estados Unidos, a Vesta já completou mais de 1.1 bilhão de operações que somaram mais de 90 bilhões de dólares em valores transacionados.

que foi disponibilizado no já mencionado *Kaggle* em uma competição promovida pela *IEEE Computational Intelligence Society*<sup>14</sup>.

O *dataset* é formado por 5 arquivos *.csv* (*comma separated value*), dos quais, apenas 1 foi utilizado nesta pesquisa.

Nomeado como *train Transaction*, este arquivo contém mais de 590 mil registros de compra e venda efetuados com cartão de crédito rotulados como fraudulentos ou não fraudulentos. O arquivo é composto de 394 colunas, as quais são descritas da seguinte maneira<sup>15</sup>:

*TransactionDT*: Registra o momento da compra, porém, não é um *timestamp*. O segredo para entender esta coluna é atentar ao valor do primeiro registro, que marca 86400. Aparentemente, este valor representa a unidade de tempo em segundos já que 86400 segundos é exatamente a duração de um dia ( $60 * 60 * 24$ ). Assim, o momento em que as transações ocorreram vão sendo registradas com base na quantidade de segundos transcorridos a partir do valor inicial. Por exemplo, a segunda transação da base de dados tem como valor de *TransactionDT* o valor 86401, ou seja, ocorreu 1 segundo após o primeiro registro, já a terceira transação tem como valor 86469, ou seja, ocorreu 69 segundos após o primeiro registro e assim sucessivamente. Como o maior valor para esta coluna é 15811131, percebe-se que as transações distribuem-se em um período de 6 meses, já que  $15811131 / 86400 = 183$  ou, em outras palavras: total de segundos / segundos compreendidos em 1 dia = dias transcorridos.

*TransactionAMT*: Valor da transação expresso em dólares americanos (USD). Alguns registros contam com 3 casas decimais após a vírgula, provavelmente por se tratar de uma compra em moeda estrangeira que fora convertida para dólares através da multiplicação pela taxa de câmbio.

*ProductCD*: Código do produto transacionado no registro. Pode ser tanto um produto tangível como um serviço.

*card1* – *card6*: Colunas relacionadas ao cartão que fora utilizado para efetuar o pagamento. Consta nestas colunas, por exemplo, o tipo de cartão (Visa, Mastercard, American Express, etc.), tipo do cartão (crédito ou débito)

---

<sup>14</sup> A organização, composta por profissionais engenheiros eletricitas e engenheiros eletrônicos, foca na teoria, desenvolvimento e aplicação de paradigmas computacionais com ênfase em redes neurais, algoritmos genéticos, lógica fuzzy, entre outros conceitos correlatos (IEEE, 2016).

<sup>15</sup> A descrição das colunas e seus valores é um trabalho conjunto entre a própria Vesta no lançamento da competição, como dos usuários da comunidade *Kaggle*, conforme pode ser visto em <https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203>

*addr1 – addr2*: Relativos ao endereço do comprador, onde *addr1* é relativa à região do comprador e *addr2* é relativo ao país. Ambos os valores são relacionados ao valor que consta no cadastro do cartão de crédito.

*dist1 – dist2*: Distância entre endereço de cobrança, endereço de correspondência, código postal, ip do dispositivo que efetuou a compra, código de área do telefone, etc. Não é possível saber quais distâncias estão sendo comparadas

*P\_emaildomain – R\_emaildomain*: Serviço de e-mail do comprador e vendedor, respectivamente. Ex: Gmail.com, Yahoo.com, Hotmail.com, etc.

*C1 – C14*: Colunas para contagens diversas, como, por exemplo, quantos endereços válidos estão associados a este cartão? Quantos números de telefone? Quantos endereços de e-mail? Quantos endereços IPs? Esta contagens referem-se tanto aos compradores quanto aos vendedores.

*D1 – D15*: Intervalos de tempo para questões variadas, como por exemplo, quantos dias se passaram entre esta compra e a última compra efetuada pelo usuário?

*M1 – M9*: Correspondências encontradas entre o nome atribuído ao cartão e o endereço registrado do mesmo, por exemplo.

*V1 – V339*: *Features* anonimizadas criadas pelo time de engenharia de dados da Vesta, que incluem, por exemplo, classificações, contagens e outros relacionamentos entre entidades.

Um ponto importante a ser levado em consideração ao explorar este conjunto de dados é que *datasets* de transações bancárias reais possuam grande parte das colunas e dados anonimizados. Ou seja, colunas que poderiam ser usadas para chegar à identidade do verdadeiro cliente possuem uma máscara numérica capaz de transformar dados sensíveis em dados não-relacionáveis aos usuários reais. Todas as 339 colunas criadas pela Vesta, citada anteriormente, por exemplo, são anonimizadas.

Outro ponto digno de nota é quanto às colunas de variáveis categóricas, ou seja, variáveis que assumem um número limitado de valores possíveis, como, por exemplo, um suposto *dataset* com a coluna “UF”, onde as possibilidades variam entre os 26 estados e o distrito federal. As colunas de variáveis categóricas são: *ProductCD*, *card1 – card6*, *addr1 – addr2*, *P\_emaildomain*, *R\_emaildomain*, *M1-M9*.

### 3.3 Bibliometria Quantitativa

Para assegurar a importância da pesquisa foi realizado um levantamento quantitativo que confirmou os indícios de que a produção na área está em alta. Os dados da Tabela 1 são referentes a uma busca contendo os termos “*fraud detection*”, “*credit card*” e a conjunção dos termos “*machine learning*”, “*data science*” e “*data analytics*”. Os resultados foram limitados a artigos com no máximo 5 anos de publicação. Foram consultadas as bases de dados Scielo.org; Web of Science e Science Direct, encontrando-se, respectivamente, nenhum, 40 e 223 artigos, quando dos termos utilizados em conjunto.

Tabela 1 - Bibliometria quantitativa

<i>Scielo.org, Web of Science e Science Direct</i>	<i>“data science” OR “machine learning” OR “data analytics”</i>			<i>“Fraud detection”</i>			<i>“credit card”</i>		
	<i>Scielo. org</i>	<i>Web of Science</i>	<i>Science Direct</i>	<i>Scielo. org</i>	<i>Web of Science</i>	<i>Science Direct</i>	<i>Scielo. org</i>	<i>Web of Science</i>	<i>Science Direct</i>
<i>“data science” OR “machine learning” OR “data analytics”</i>	69	80.569	86.361	0	158	829	0	158	868
<i>“Fraud detection”</i>	-	-	-	26	604	1.249	1	101	276
<i>“credit card”</i>	-	-	-	-	-	-	14	729	3.227
<i>“credit card” AND “fraud detection” AND (“data science” OR “machine learning” OR “data analytics”)</i>						<i>Scielo.org: 0 Web of Science: 40 Science Direct: 223</i>			

Fonte: Pesquisa realizada em 19 de junho de 2021

### 3.4 Bibliometria Qualitativa

Como a quantidade de artigos encontrados através da combinação de termos foi bastante volumosa, alguns poucos foram selecionados para serem lidos e resenhados. Aqui, cita-se

aqueles que, por conta do título, mais proveitosos a esta pesquisa pareciam ser. Os critérios para esta seleção foram os algoritmos empregados para classificação citados na seção 2.4.1 a 2.4.5.

### **3.4.1 *Credit card fraud detection using Machine Learning Techniques: A Comparative Analysis***

No artigo apresentado por Awoyemi *et al.*, (2017) os autores analisam os resultados obtidos na detecção de fraude após aplicarem os algoritmos Naive Bayes, *K-Nearest Neighbors* e Regressão Logística. Os autores atacam inicialmente o problema do desbalanceamento dos dados. Em uma acurada revisão bibliográfica eles relataram os benefícios de fazer uma distribuição artificial dos dados fraudulentos/legítimos em 50:50; também foram apresentados os resultados de uma distribuição 1:99 e 10:90. Outro ponto abordado que merece destaque é a análise sobre os diferentes tipos de variáveis que podem ser consideradas para fazer uma análise de cartão de crédito. A título de análise os autores dividem as variáveis em 5 grupos: a) Variáveis estatísticas regionais; b) variáveis estatísticas do vendedor; c) variáveis estatísticas de quantidade com base temporal; d) variáveis estatísticas de valor, também baseadas temporalmente e; e) todas elas em conjunto. Através de sua revisão bibliográfica são citados estudos que utilizam de 16 a 30 variáveis do mesmo dataset para chegar aos seus resultados. Como a pesquisa se propõe a ser um estudo comparativo, uma das seções se dedica a analisar outros artigos e, qual o melhor resultado em cada ocasião, como eles contornam o problema de desbalanceamento de dados, entre outros aspectos. Os autores também apresentam os respectivos pontos negativos encontrados nas pesquisas e nos métodos aplicados. No desenvolvimento da sua solução eles fizeram um pré-tratamento dos dados em uma técnica híbrida de *undersampling* e *oversampling*, criando 2 datasets com, respectivamente 10:90 e 34:66 de transações falsas/verdadeiras. Para analisar os resultados foram utilizadas as métricas acurácia, sensibilidade, especificidade, precisão, correlação de Matthews e taxa balanceada de classificação. De todas as 10 métricas analisadas somente regressão logística não apresentou melhorias quando passou de uma distribuição 10:90 para uma distribuição 34:66. O KNN registrou 0 falsos positivos para os 2 tipos de distribuição sendo o melhor em quase todas as métricas com exceção de acurácia na distribuição 10:90. Os autores conseguiram, com isso, resultados melhores do que os trabalhos comparados.

### **3.4.2 Random Forest for Credit card fraud detection**

No artigo apresentado por Xuan *et al.*, (2018) é explorada uma base de dados com mais de 30 milhões de registros de compra com cartão de crédito registrados entre os meses de novembro de 2016 a janeiro de 2017. A database foi fornecida por uma companhia de e-commerce chinesa e possui, apenas, 0,27% de transações marcadas como fraudulentas. Na pesquisa são utilizadas 2 variações do método *Random Forest*: método *Random-tree-based* e método *Cart*. De forma simplificada, o método funciona como um Random Forest que utiliza o cálculo da distância de Manhattan para traçar os caminhos dos nodos. Já no segundo método os dados são distribuídos de acordo com o atributo que tem o menor nível Gini de impureza. Os autores alertam os prós e contra de ambos os métodos. Nos testes os autores observaram ser o primeiro algoritmo mais rápido para computar os valores do centro, porém, mais lento para efetuar os cálculos que definem o caminho a serem seguidos nos nodos; já o segundo método mostrou-se mais lento para computar o índice de impureza dos atributos, no entanto, mais veloz para processar a distribuição dos dados. As métricas analisadas foram acurácia, precisão, *recall* e *F1-measure*. O segundo método se mostrou mais eficiente em todos os quesitos, com exceção de precisão. Necessário salientar que, antes de efetuarem os testes, foram feitas 10 diferentes distribuições dos dados que variaram de uma proporção de 1 para 1 entre transações verdadeiras/fraudulentas até 10 para 1. Segundo os autores a proporção de 5:1 obteve melhores resultados quando trabalhada em conjunto com o segundo método de Random Forest. Os autores lembram que foram testados também os algoritmos Naive Bayes, Redes Neurais, *Support Vector Machine*, entre outros, mas que, devido aos resultados ruins, nem mesmo foram citados na pesquisa.

### **3.4.3 Credit card fraud detection using Machine Learning Algorithms**

No artigo apresentado por Dornadula e Geetha (2019) os autores apresentam um novo método de detecção de fraudes para dados de transação em tempo real. O método criado analisa as transações anteriores dos clientes e monta um perfil de usuário com seus comportamentos de compra. Os clientes são, então, separados em clusters de acordo com o montante de suas

transações. O próximo passo é utilizar a estratégia das janelas deslizantes<sup>16</sup> para agregar as transações feitas pelos clientes em diferentes grupos e, assim, extrair o padrão de consumo desses grupos. Finalmente, diferentes classificadores são treinados com estes grupos separadamente. O classificador com os melhores números é escolhido para ser usado como preditor de fraudes. Os algoritmos empregados pelos autores foram *Random Forest*, Árvore de Decisão, Regressão Logística, *Support Vector Machine*, Floresta de Isolamento e *Local Outlier Factor* em uma rodada com e sem a aplicação da técnica SMOTE<sup>17</sup>. Como métodos de avaliação eles utilizaram Acurácia, Precisão e Coeficiente da Correlação de Matthews. Os melhores resultados ficaram por conta do *Random Forest*.

#### 3.4.4 *Credit card fraud detection using Pipeling and Ensemble Learning*

No artigo apresentado por Bagga *et al.* (2020) os autores fazem uma revisão dos principais algoritmos de *machine learning* aplicados no contexto da detecção de fraudes em transações de cartão de crédito. Os algoritmos empregados pelos autores foram Regressão Logística, Naive Bayes, *Random Forest*, *k-Nearest Neighbors*, Perceptron multicamada, Ada Boost e Análise Discriminatória de Quadrante. Para que fossem melhorados os resultados de acurácia foram, previamente, aplicadas duas técnicas no *dataset*: *Pipelining* e *Ensemble Learning*. O primeiro deles refere-se à aplicação de uma série de transformações e a aplicação de um classificador no final do processo. Desta maneira é possível o cruzamento de diversos processos com diferentes parâmetros. Já o segundo método refere-se à combinação de vários estimadores (foram utilizados 15 deles) que são desenvolvidos com um método de aprendizagem específico para melhorar aquele estimador individual. Neste trabalho foi utilizado o classificador *Bagging*, que aplica os estimadores em subsets randômicos do *dataset* original e então aplica o método da Agregação para realizar as predições individuais. Para medir os resultados foram utilizados Acurácia, Precisão, *F1 score* e Revocação.

---

<sup>16</sup> Do inglês *Sliding Window Strategy*, esta técnica cria um subconjunto (uma janela) a partir de um array, no qual, a cada iteração, 1 posição é deslocada, criando um novo recorte de análise (TREUTLER, 2020).

<sup>17</sup> Acrônimo para *Synthetic Minority Oversampling Technique* (Técnica de sobreamostragem de minoria sintética, em tradução literal) é uma técnica aplicada em datasets nos quais há uma grande diferença de registros entre as classes. Para compensar esta disparidade o SMOTE cria instâncias da classe menos representada a partir dos registros existentes da própria classe (EL-SAYED et al., 2015).

### 3.5 Quadro comparativo

A Tabela 2 apresenta um quadro comparativo das principais características dos trabalhos selecionados e descritos na bibliografia qualitativa. Os critérios de comparação são os algoritmos de *machine learning* utilizados e os respectivos resultados obtidos.

Tabela 2 - Quadro comparativo das técnicas levantadas na bibliografia qualitativa

Artigo	Algoritmos utilizados	Resultados obtidos <sup>18</sup>	Observação
<b>Awoyemi et al., (2017)</b>	Naive Bayes	Acurácia: 0,9769 Precisão: 0,9786 CCM: 0,9478	Os autores utilizaram a distribuição 70:30 para treino/validação. Os testes foram feitos com um arranjo fixo de 2 registros legítimos para cada registro fraudulento.
	K-Nearest Neighbors	Acurácia: 0,9792 Precisão: 1 CCM: 0,9535	
	Regressão Logística	Acurácia: 0,5486 Precisão: 0,3836 CCM: 0,1080	Foi utilizado 3 como valor de K nos testes com o algoritmo KNN.
<b>Xuan et al., (2018)</b>	Random Forest	Acurácia: 0,9867 Precisão: 0,3268 Revocação: 0,5962	Foi utilizada a distribuição 70:30 para treino/validação e dois métodos diferentes de Random Forest foram aplicados. Os resultados ao lado referem-se ao método Random Forest CART, que mostrou melhores resultados
<b>Dornadula e Geetha (2019)</b>	Regressão Logística	Acurácia: 0,9718 Precisão: 0,9831 CCM: 0,9438	Os autores não citam o recorte utilizado de treino/validação.
	Random Forest	Acurácia: 0,9998 Precisão: 0,9996 CCM: 0,9996	

<sup>18</sup> Importante ressaltar que, com exceção de Xuan *et al.* e da presente pesquisa, os demais trabalhos utilizaram a mesma base de dados. O *dataset* em questão compreende dados oriundos de companhias de cartão de crédito europeias com cerca de 285 mil registros e já foi exaustivamente explorado.

	Árvore de decisão	Acurácia: 0,9708 Precisão: 0,9814 CCM: 0,9420	Os resultados exibidos aqui foram obtidos após aplicação da técnica SMOTE.
<b>Bagga et al., (2020)</b>	Naive Bayes	Acurácia: 0,996 Precisão: 0,26 Revocação: 0,85 <i>F1 Score</i> : 0,40	Foi utilizada a distribuição 70:30 para treino/validação. Os autores não informam o valor dado para K nos testes com o algoritmo KNN.
	K-Nearest Neighbors	Acurácia: 0,944 Precisão: 0,02 Revocação: 0,55 <i>F1 Score</i> : 0,03	
	Regressão Logística	Acurácia: 0,982 Precisão: 0,07 Revocação: 0,91 <i>F1 Score</i> : 0,14	
	Random Forest	Acurácia: 0,997 Precisão: 0,31 Revocação: 0,89 <i>F1 Score</i> : 0,46	
<b>Esta pesquisa</b>	Random Forest	Revocação: 0,89 CCM: 0,46	Após diferentes testes foi utilizada a divisão 80:20 como recorte nesta pesquisa, pois os resultados foram mais precisos.
	Extreme Gradient Boosting	Revocação: 0,87 CCM: 0,29	

Fonte: Autores (2020)

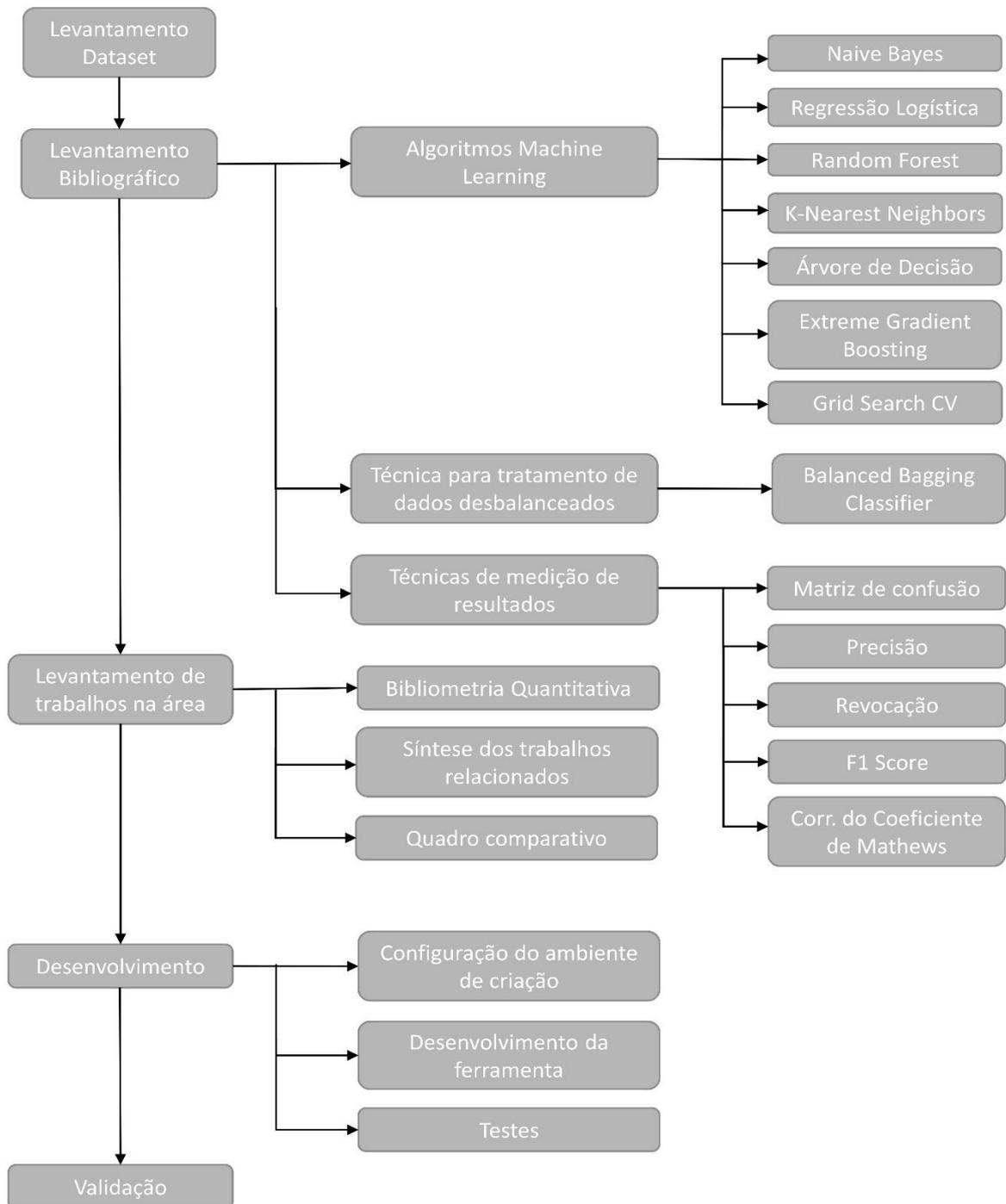
A partir dos trabalhos relevantes selecionados para a leitura e revisão crítica, foi possível observar os principais algoritmos de ML utilizados para a identificação de transações falsas com cartão de crédito, com destaque para o algoritmo *Random Forest*, o mais citado dentre os analisados.

Nota-se entretanto, que esta pesquisa não pôde obter os mesmos resultados com o referido algoritmo – os motivos serão discutidos no capítulo 5 deste trabalho – e uma nova alternativa foi empregada, esta sim entregando resultados satisfatórios.

### 3.6 Procedimentos Metodológicos

A Figura 9 exibe os procedimentos metodológicos do desenvolvimento deste trabalho.

Figura 9 - Procedimentos metodológicos



Fonte: Autores (2021)

Na primeira parte deste trabalho, foi realizado o levantamento do *dataset*, o conjunto de dados que seria utilizado para análise, treinamento e validação dos resultados. A base de dados selecionada é proveniente do site Kaggle e foi fornecida pela *IEEE Computational Intelligence Society*. Nela constam mais de 590 mil registros de transações reais rotuladas como fraudulentas ou não fraudulentas fornecidos pela Vesta, uma empresa americana de pagamentos digitais.

Na sequência fez-se a pesquisa bibliográfica para aprofundar o conhecimento nos assuntos relacionados ao tema de pesquisa. Os tópicos explorados foram os algoritmos de *machine learning* como Regressão Logística, *Random Forest*, entre outros; conceitos e metodologias para a medição dos resultados, como precisão e revocação, bem como a ferramenta *Balanced Bagging Classifier* para tratamento de dados desbalanceados. Também foi feito o levantamento dos trabalhos relacionados, iniciando pela bibliometria quantitativa, onde analisamos o número de trabalhos publicados nos últimos cinco anos que compartilham das mesmas áreas de estudo deste trabalho. Já na bibliometria qualitativa, o passo seguinte, foram sintetizados trabalhos relacionados e comparados em relação às técnicas utilizadas e resultados obtidos.

Na segunda parte do trabalho, foi desenvolvida a ferramenta capaz de ser treinada para reconhecer potenciais transações fraudulentas de cartão de crédito utilizando ML. A partir da ferramenta desenvolvida e do subsequente treinamento, foram realizados os testes de validação dos resultados.

## 4 Solução Desenvolvida

Diversos estudos já se dedicaram a fazer a mesma análise proposta neste trabalho. Assim, soluções neste campo não são difíceis de serem encontradas. No entanto, a grande maioria se apresenta como não implementável em um ambiente de produção. O que se encontra, no tocante ao tema, são as chamadas soluções caixa-preta, ou seja, soluções que não explicam os métodos criados ao ponto de permitirem a replicabilidade do estudo em uma base de dados diferente. Quais as configurações finas dos algoritmos? Quais as variáveis analisadas? Qual o *pipeline* proposto? Quais as técnicas de pré-processamento que foram aplicadas?

Por melhores que sejam os resultados obtidos, como os de Bagga *et al.*, (2020), é preciso entender as métricas com ótimos resultados e o passo a passo para replicação e para implementação em sistemas reais.

Assim, propôs-se criar, nesta pesquisa, um modelo que busca não só os melhores resultados possíveis, mas também ser o mais transparente sobre as etapas do processo, permitindo a replicabilidade.

Para isso propõe-se uma documentação detalhada e a disponibilização de um script pré-configurado para replicação desta pesquisa a qualquer base de dados supervisionada de transações de cartão de crédito.

### 4.1 Visão geral do sistema

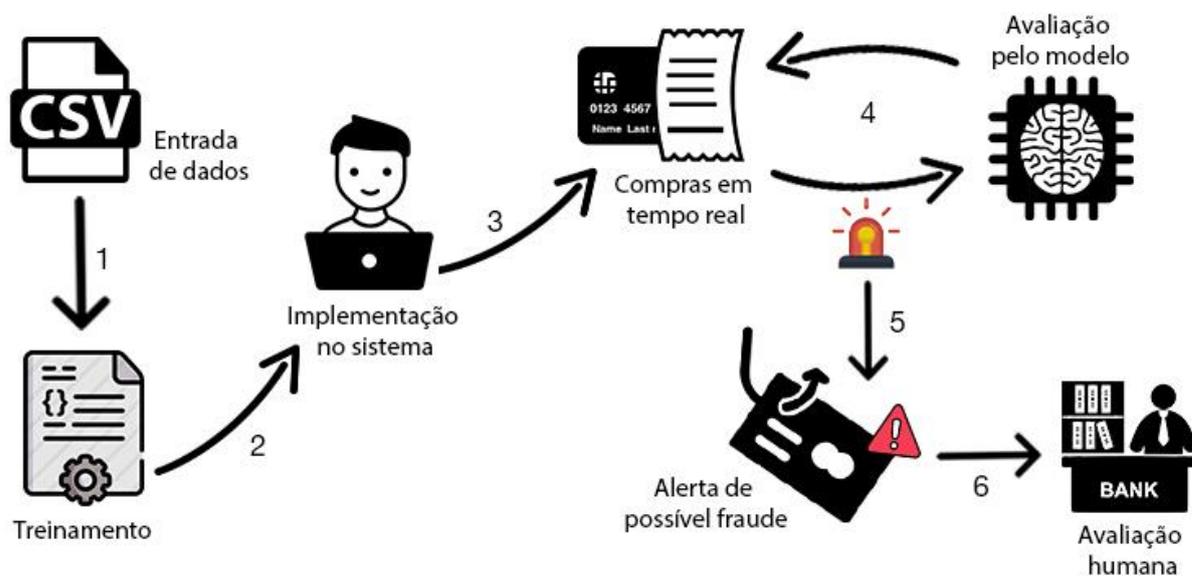
A solução foi idealizada na criação de um modelo de *machine learning* eficiente, simples de ser implementado em um ambiente real e de fácil configuração.

O processo funciona da seguinte maneira: Os dados de entrada a serem carregados no script para fins de treinamento devem estar no formato *Comma Separated Values* (CSV). O script que treinará o modelo terá configurações pré-definidas e selecionadas de acordo com os melhores resultados obtidos nos testes, no entanto, estes valores poderão ser alterados, caso desejado, por meio de um arquivo de configurações.

Após o modelo ter sido treinado, novas transações poderão ser submetidas para avaliação e classificação das probabilidades de ser uma transação fraudulenta.

Ressalta-se que o uso desta ferramenta em um ambiente real não é de fácil simulação, já que o processo de cancelamento de compras potencialmente falsas segue regras específicas de acordo com as políticas da instituição em questão. Imagina-se, para fins didáticos desta pesquisa, que após ser implementado o modelo rodará na nuvem, de onde fará a análise das operações de compra instantaneamente. Quando alguma delas for sinalizada como uma possível fraude, será feito o bloqueio da mesma e emitido um alerta a ser conferido por um especialista humano, de acordo com os protocolos necessários. Após implementado, o modelo deverá passar por treinamentos periódicos para incorporação de novos dados e aprendizagem/aperfeiçoamento de padrões de compra. Este processo é representado na Figura 10.

Figura 10 - Funcionamento da solução proposta dividida em etapas



Fonte: Autores (2020)

## 4.2 Tecnologias utilizadas

Para o desenvolvimento da solução foi necessário a pesquisa de viabilidade e a identificação de uma série de tecnologias que pudessem viabilizar os objetivos propostos. Dentre as possibilidades encontradas, utilizou-se:

### 4.2.1 Python

É uma linguagem de programação de alto nível, orientada a objetos, de tipagem dinâmica e forte. Por ser uma linguagem fácil de aprender, escalável e mais rápida tanto para execução como para desenvolvimento de aplicações, foi “adotada” pela comunidade como a linguagem padrão para ciência de dados. Atualmente está na versão 3.9.5 (PYTHON.ORG, 2021).

Outro ponto que contribuiu para a predominância do Python na área são as diversas bibliotecas existentes, que atendem as mais variadas necessidades, como algumas que foram utilizadas neste projeto.

### 4.2.2 Scikit-Learn

É uma das bibliotecas mais utilizadas nos projetos de *machine learning* devido sua completude em oferecer as ferramentas básicas que estes projetos necessitam. A biblioteca possui vários algoritmos de classificação, regressão e agrupamento como Random Forest, K Nearest Neighbor, Support Vector Machine e tantos outros.

Com ela é possível extrair valor dos dados e produzir *data science* em questão de minutos, pois dispensa ao usuário o conhecimento da lógica por trás da construção de qualquer um dos algoritmos citados. Assim como o Python, a biblioteca é mantida pela comunidade e sua versão estável é a 0.24.2 (SCIKIT-LEARN, 2021).

### 4.2.3 Imbalanced Learn

Esta biblioteca é uma ramificação da Scikit Learn, da qual compartilha comandos e parâmetros, e é voltada para a aplicação em conjunto de dados altamente desbalanceados. Com ela é possível criar registros para classes com *undersampling*, remover registros em classes com

*oversampling*<sup>19</sup> e efetuar o *resampling*, técnica na qual é utilizada a pequena quantidade de dados disponível de uma classe desbalanceada para estimar um parâmetro populacional.

Criada recentemente, está na versão 0.8 e também é mantida pela comunidade (IMBALANCED LEARN, 2021)..

#### 4.2.4 Pandas

Pandas é uma popular biblioteca Python para manipulação e análise de dados. Ela oferece as estruturas de dados e as operações necessárias para manipulação de tabelas e séries temporais. Além destes pontos, outra vantagem que faz com que esta biblioteca tenha destaque na ciência de dados é que, com ela, torna-se fácil importar e manipular arquivos .CSV. Sua versão estável é 1.2.4 (PANDAS, 2021).

#### 4.2.5 Matplotlib

Matplotlib é a mais famosa biblioteca de criação de gráficos e visualizações de dados em geral para Python. A biblioteca contém inúmeros tipos de gráficos como gráficos de contorno, área, linha, histograma, distribuição, mapas de calor, dispersão, vela, entre tantos outros. Sua versão estável é a 3.4.2 (MATPLOTLIB, 2021).

#### 4.2.6 Joblib

Joblib é uma biblioteca utilizada em Python para persistência de arquivos. Com ela é possível, por exemplo, salvar e carregar um modelo de *machine learning* que fora treinado previamente. Como o treinamento dos modelos costuma demorar horas por conta da quantidade de registros a serem analisados, ter uma maneira de carregar o classificador instantaneamente é

---

<sup>19</sup> Conforme exposto na seção 2.3.1, *oversampling* consiste em criar dados sintéticos para uma classe com poucos registros, enquanto que *undersampling* consiste na redução de registros da classe majoritária. As técnicas são parte do *resampling*, que pode ser caracterizado como o processo de reorganização dos registros das classes.

algo fundamental, sobretudo por conta da necessidade em submeter as transações em tempo real para averiguação. Sua versão atual é a 1.0.1 (JOB LIB, 2021).

#### 4.2.7 Pycharm

Pycharm é um ambiente de desenvolvimento Python criado pela empresa JetBrains, que pode ser utilizado tanto em sua versão gratuita ou versão paga, com ferramentas adicionais.

O software conta com análise de código, um *debugger* gráfico, testes unitários integrados, recursos de desenvolvimento cooperativos e remotos, suporte ao desenvolvimento web através do *framework* Django, Flask e Web2py, entre outras características. Sua versão estável é a 2021.1 (JET BRAINS, 2021).

#### 4.2.8 Google Colab

Para rodar o projeto com eficiência foi utilizada a plataforma Colab, desenvolvida pelo Google, que permite que o código seja rodado na nuvem utilizando recursos do próprio Google, sem depender, portanto, da máquina do usuário. Por permitir que blocos de código sejam adicionados e rodados de maneira independentemente, bem como ter suporte a anotações e formatação de texto, a plataforma se popularizou como ferramenta de ensino e compartilhamento de conhecimento.

No Google Colab é possível, por exemplo, desenvolver um código, explicá-lo de maneira mais amigável do que os tradicionais comentários das IDEs e então compartilhá-lo. Assim, a plataforma despontou entre estudantes e entusiastas de *data science* que a utilizam para distribuir seus notebooks, como são chamados os arquivos criados. Por ser uma aplicação que roda diretamente no navegador não possui uma versão estável a ser citada (COLAB RESEARCH, 2020).

### 4.3 Aplicação do *Machine Learning*

Após analisar os resultados levantados ao fim da bibliometria qualitativa e condensados no quadro comparativo, observou-se que o algoritmo Random Forest apresentava os melhores resultados em diversas métricas e estudos. Portanto, este foi o algoritmo escolhido para iniciar os testes com o *dataset*, porém, antes, há de se realizar algumas etapas.

#### 4.3.1 Pré-processamento dos dados

A tarefa mais básica de todas em uma pesquisa que envolve dados é, sem dúvida, carregá-los no sistema, além de, é claro, importar as bibliotecas que serão utilizadas no decorrer da aplicação. Assim, a base de dados foi enviada ao Google Drive e então carregada para o Colab, conforme ilustra a Figura 11.

Figura 11 - Configuração inicial do projeto

```
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import plot_confusion_matrix, f1_score, recall_score, matthews_corrcoef

drive.mount('/content/gdrive')

data = pd.read_csv('gdrive/MyDrive/Unisc/TCC/train_transaction.csv')
```

Fonte: Autores (2021)

Após a importação das bibliotecas necessárias é pertinente um pré-tratamento dos dados. De valor fundamental para o bom andamento da pesquisa, ações tomadas (ou não tomadas) nesta fase podem, inclusive, determinar se as taxas de acerto do modelo ficarão acima ou abaixo do esperado.

No pré-processamento de dados busca-se encontrar e solucionar problemas comuns de ocorrerem na etapa de ETL<sup>20</sup> antes mesmo de começar a modelagem da solução desenvolvida. Questões como normalização dos valores, busca e tratamento de registros faltantes ou de pontos

---

<sup>20</sup> Do inglês *Extract, Transform and Load*, é a etapa na qual os dados são coletados do banco de dados e transformados em um *dataset* a ser trabalhado.

extremos fora da curva, conhecidos como *outliers*, por exemplo, são apenas algumas das ações a serem tomadas neste ponto do projeto.

A primeira questão a ser considerada, portanto, foi a grande quantidade de colunas, a fim de ganho de desempenho na execução do futuro treinamento. Como descrito no seção 3.2, do capítulo 3, o *dataset* contém mais de 380 colunas neste momento, sendo, a grande maioria – mais de 330, criada pelo time de engenharia de dados da companhia e que, além de não demonstrarem valor explícito para a análise (pois todas elas têm seus valores mascarados), continham, em sua grande maioria, somente valores binários.

Assim, para identificar aquelas que não agregam ao conjunto foi feita a análise das colunas correlacionadas, aquelas que, estatisticamente, eram iguais para o modelo, ou seja, que tinham a mesma capacidade de “ensinar”. Para tal utilizou-se de uma função nativa da biblioteca Pandas chamada *data.corr()* com a seguinte lógica: as colunas que demonstram uma taxa de correlação acima de 80% com alguma outra coluna seria excluída e somente uma seria mantida. A função encontrou 252 colunas nessa condição. O processo pode ser visto na Figura 12.

Figura 12 - Processo de remoção de colunas correlacionadas

```
correlacionadas = set()
matriz_corr = data.corr()

for i in range(len(matriz_corr.columns)):
    for j in range(i):
        if abs(matriz_corr.iloc[i, j]) > 0.8:
            colname = matriz_corr.columns[i]
            if colname[0] == 'V' or colname[0] == 'C' or colname[0] == 'D':
                correlacionadas.add(colname)

dados_sem_correlacoes = data.drop(correlacionadas, axis=1)
dados_sem_correlacoes.to_csv('gdrive/MyDrive/Unisc/TCC/data_corr.csv', index=False)

data = pd.read_csv('gdrive/MyDrive/Unisc/TCC/data_corr.csv')
```

Fonte: Autores (2021)

Após este processo foi salvo um novo recorte do *dataset*, para registrar as etapas e para uma eventual necessidade de carregar os dados neste ponto do desenvolvimento.

O próximo passo foi eliminar as colunas que não fossem do tipo numérico, já que o algoritmo Random Forest não consegue interpretá-las nativamente. Não foi utilizada nenhuma

função específica, apenas a comparação do tipo de dados da coluna. Aquelas que fossem do tipo *Object* seriam removidas. Após esta etapa mais 14 colunas foram excluídas e um novo recorte do *dataset* foi salvo, como pode ser visto no processo mostrado na Figura 13.

Figura 13 - Processo de remoção de colunas não-numéricas

```
nao_numerico = set()

for y in data.columns:
    if(data[y].dtype == object):
        colname = y
        nao_numerico.add(colname)

dados_sem_str = data.drop(nao_numerico, axis=1)
dados_sem_str.to_csv('gdrive/MyDrive/Unisc/TCC/data_sem_str.csv', index=False)

data = pd.read_csv('gdrive/MyDrive/Unisc/TCC/data_sem_str.csv')
```

Fonte: Autores (2021)

O passo seguinte foi lidar com as colunas que continham registros *not a number (NaN)*, ou seja, valores em branco, já que esta também é uma limitação do algoritmo RF. Como pode ser visto na Figura 14, neste momento o *dataset* continha mais de 26 milhões registros classificados como tal.

Figura 14 - Quantidade de registros NaN no dataset

```
data.isna().sum().sum()

26348344
```

Fonte: Autores (2021)

Neste cenário há duas opções: remover o registro que contém o valor *NaN* ou preencher com um valor numérico a ser definido. A primeira opção é sempre arriscada, sobretudo se aplicada a *datasets* com muitas colunas, já que, ao eliminar a linha inteira de um registro corre-se o risco de eliminar informações significativas presentes em outras colunas. E quando considera-se em remover mais de 26 milhões de registros, as chances de remover algo significativo por uma abordagem errada são imensas.

A segunda opção, adotada portanto, oferece diferentes possibilidades, como, por exemplo, preencher o valor *NaN* com algum valor determinado e fixo, preencher com o último valor disponível, preencher com o próximo valor disponível, com uma média dos últimos *n* valores, com a média da coluna, entre outros (CHEN, 2020).

Para encontrar a melhor opção para este cenário específico, foram rodados quatro testes com as configurações padrão do algoritmo RF: Com preenchimento dos valores *NaN* utilizando o 0, que obteve 0,36 de revocação; preenchimento com o próximo valor disponível e preenchimento com valor anterior disponível, nos quais, ambos alcançaram 0,33 de *recall* e o procedimento onde utilizou-se a média dos valores da coluna para preenchimento dos *NaN* da mesma. A taxa de *recall* desse último método foi de 0,40, portanto, o processo de utilização da média, demonstrado na Figura 15, foi o método escolhido para tratamento dos valores *Not a Number*.

Figura 15 - Processo de preenchimento dos valores *NaN*

```
▶ for i in data.columns:  
    data[i].fillna(data[i].mean(), inplace=True)  
  
data.to_csv('gdrive/MyDrive/Unisc/TCC/data_sem_nan.csv', index=False)
```

Fonte: Autores (2021)

Com todas as etapas do pré-processamento concluídas, gerou-se o recorte final do conjunto de dados que seria utilizado para o treinamento do modelo. O processo pode ser visto na Figura 16.

Figura 16 – Criação do dataset após aplicação do pré-processamento

```
▶ data.to_csv('gdrive/MyDrive/Unisc/TCC/pre_processamento_ok.csv', index=False)
```

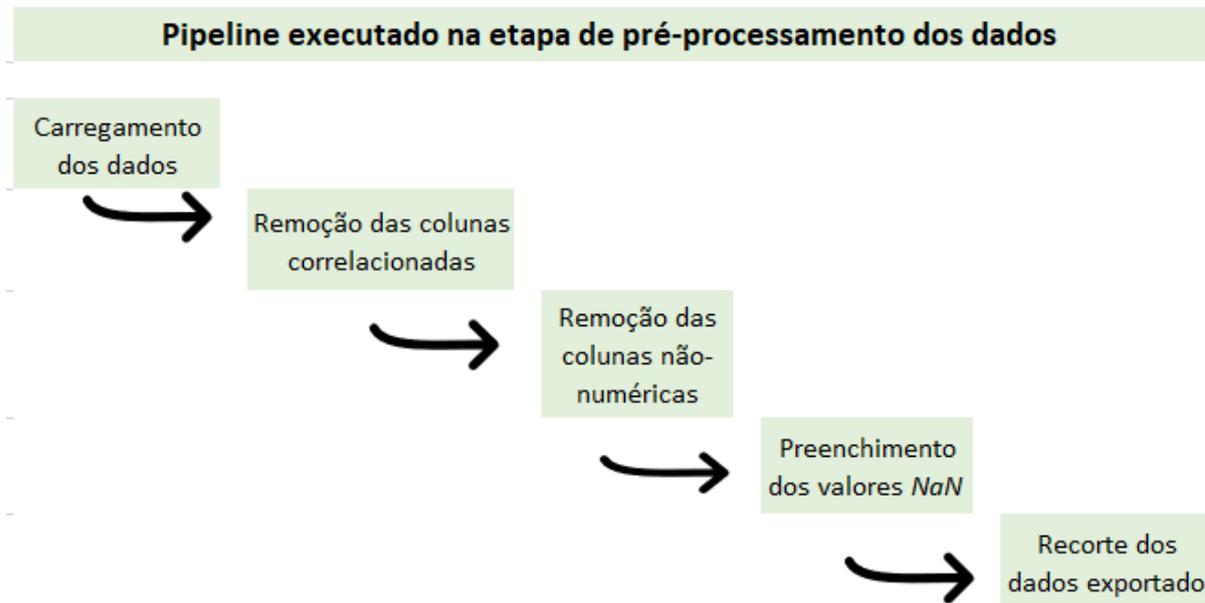
Fonte: Autores (2021)

A Figura 17 sintetiza o *pipeline*<sup>21</sup> proposto e executado.

---

<sup>21</sup> Define-se como *Pipeline* uma sequência de eventos que podem ser executados, inclusive em paralelo para ganhos de desempenho e tempo, a fim de automatizar um processo de transformação dos dados, por exemplo, como no nosso caso.

Figura 17 - Pipeline executado na etapa de pré-processamento dos dados



Fonte: Autores (2021)

#### 4.4 Testes com o algoritmo *Random Forest*

Antes de iniciar a explicação do processo de treinamento com *Random Forest* e os resultados alcançados, é necessário fazer uma observação: Após algumas rodadas de teste verificou-se que o treinamento completo do modelo necessitava de, em média, 14 minutos para analisar os mais de 590 mil registros a cada novo treinamento. Para deixar as rodadas de teste e modelagem mais ágeis foram criados 2 recortes no *dataset*: um com 100 mil e outro com 30 mil registros, ambos com cerca de 2,8% de transações fraudulentas. Embora as imagens e os valores dos testes descritos neste capítulo estejam ilustrados com os números do *dataset* completo, as rodadas de teste foram executadas com o conjunto de 30 mil registros para fins de celeridade.

O primeiro passo para treinar o modelo é decidir qual será a divisão entre os dados de treinamento e os dados de teste. Seguindo a divisão mais utilizada, de acordo com os artigos relacionados na bibliometria qualitativa, os primeiros testes foram feitos com 70% dos dados sendo utilizados para treinamento e 30% para teste. A divisão dos dados foi feita com a ferramenta *train\_test\_split* da biblioteca Scikit Learn, com o parâmetro *stratify* configurado, o que garante que, após a separação, os dados de treino e de teste manterão a mesma divisão entre as classes que é observada no *dataset* íntegro.

Os primeiros resultados, conforme mostra a Figura 18, não foram nem um pouco animadores, identificando apenas 2728 fraudes das 6199 possíveis, alcançando uma taxa modesta de 0,44 de revocação.

Figura 18 - Resultados do Random Forest com os valores padrão e distribuição 70/30



Fonte: Autores (2021)

Após diversas rodadas de testes e configurações alternativas, pouco se conseguiu evoluir nos resultados com o Random Forest<sup>22</sup>. Assim, supõe-se que os resultados apresentados pelas demais pesquisas, citadas na bibliometria qualitativa, só conseguiram obter os excelentes resultados apresentados com Random Forest por trabalharem com um *dataset* mais simples e já otimizado, de apenas 31 colunas.

<sup>22</sup> Como pode ser observado nas Figuras apresentadas, nos testes foi utilizado o parâmetro *random\_state* com valor fixo. Com ele garante-se que a separação nos dados de treinamento/teste e a criação das árvores siga sempre a mesma ordem e divisão de registros. Com isso é possível mensurar se as alterações feitas nos parâmetros a cada rodada de testes estão melhorando ou piorando o modelo, descartando quaisquer possíveis variações nos resultados por conta de eventos randômicos.

Foi observado que ao aplicar o mesmo algoritmo em um *dataset* mais complexo os resultados foram pouco satisfatórios, independentemente das calibrações dos parâmetros que fossem feitas. A pequena melhora, apresentada na Figura 19, foi obtida após a calibração do número de árvores a serem criadas para 500 e de uma nova divisão treinamento/teste de 80/20, respectivamente.

Figura 19 - Melhores resultados obtidos com o Random Forest



Fonte: Autores (2021)

Assim surgiu a necessidade de testar um novo algoritmo que pudesse ser capaz de entregar melhores resultados de detecção de fraude.

#### 4.5 Testes com Extreme Gradient Boosting

Após pesquisar diferentes algoritmos que poderiam atingir melhores resultados do que o RF, aquele que pareceu mais vantajoso em relação aos demais foi o XGB. Embora também

se trate de um algoritmo baseado em árvores de decisão<sup>23</sup>, tal qual o *Random Forest*, o XGB possui diversas melhorias em relação ao Random Forest que contribuíram na sua escolha para uma nova série de testes como o processamento paralelo, otimização de memória, entre outras características descritas com mais detalhes na seção 2.4.6 do capítulo 2.

Mas, a principal vantagem do XGB reside no fato de que o algoritmo é altamente personalizável através dos seus mais de 80 valores configuráveis. Assim, para extrair o máximo da implementação o primeiro passo com o XGB foi implementar uma técnica chamada Grid Search CV, também pertencente à já descrita biblioteca Scikit Learn.

Com esta ferramenta torna-se simples realizar centenas de combinações de parâmetros em busca da melhor configuração possível para o XGB. Na Figura 20, por exemplo, foram definidos 3 valores para cada um dos seguintes parâmetros: *eta*, *n\_estimators* e *min\_child\_weight*<sup>24</sup>, que foram combinados entre si, divididos em 5 grupos, e testados para encontrar a combinação que apresente o melhor resultado de acordo com a métrica de revocação (*recall*)<sup>25</sup>.

Figura 20 - Testes executados com o GridSearch CV

```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV as CV

modelo = XGBClassifier()

parametros = {
    'eta': [0.1, 0.5, 1],
    'n_estimators': [100, 200, 300],
    'min_child_weight': [4, 5, 6],
}

grid = CV(estimator = modelo,
          param_grid = parametros,
          scoring = 'recall',
          cv = 5)

grid.fit(data.drop(['isFraud'], axis = 'columns'), data.isFraud)
```

Fonte: Autores (2021)

<sup>23</sup> Ainda que também possa ser utilizado um motor para criação de modelos baseados em cálculos lineares, o modelo de árvores de decisão é o mais utilizado nos projetos que utilizam o XGB, além de ser o valor padrão da ferramenta (AVATO, 2019).

<sup>24</sup> Estes parâmetros serão explicados em detalhes na sequência.

<sup>25</sup> Uma explicação mais detalhada de como funciona o algoritmo pode ser encontrada no subcapítulo 2.4.7 do capítulo 2.

Após executados os testes é possível imprimir as informações desejadas: qual o melhor resultado e quais os parâmetros que levaram a estes resultados. Para fins de ilustração, na Figura 21 foi adicionado um *dataframe* com os cruzamentos e as respectivas pontuações.

Figura 21 - Resultado do cruzamento realizado pelo Grid Search CV

```
print("Melhor resultado: ", grid.best_score_)
print("Melhores parâmetros: ", grid.best_params_)
pd.DataFrame(grid.cv_results_)[['params', 'rank_test_score', 'mean_test_score']]
```

Melhor resultado: 0.3290697674418605  
Melhores parâmetros: {'eta': 0.1, 'min\_child\_weight': 4, 'n\_estimators': 300}

	params	rank_test_score	mean_test_score
0	{'eta': 0.1, 'min_child_weight': 4, 'n_estimat...	19	0.244186
1	{'eta': 0.1, 'min_child_weight': 4, 'n_estimat...	10	0.305814
2	{'eta': 0.1, 'min_child_weight': 4, 'n_estimat...	1	0.329070
3	{'eta': 0.1, 'min_child_weight': 5, 'n_estimat...	22	0.223256
4	{'eta': 0.1, 'min_child_weight': 5, 'n_estimat...	13	0.289535
5	{'eta': 0.1, 'min_child_weight': 5, 'n_estimat...	4	0.329070
6	{'eta': 0.1, 'min_child_weight': 6, 'n_estimat...	25	0.217442
7	{'eta': 0.1, 'min_child_weight': 6, 'n_estimat...	16	0.288372
8	{'eta': 0.1, 'min_child_weight': 6, 'n_estimat...	7	0.320930
9	{'eta': 0.5, 'min_child_weight': 4, 'n_estimat...	19	0.244186
10	{'eta': 0.5, 'min_child_weight': 4, 'n_estimat...	10	0.305814

Fonte: Autores (2021)

Assim que definidas as melhores variações para os parâmetros e valores inseridos, deve-se recomeçar uma nova rodada de testes. Os valores mais eficientes da última rodada passam a ser os valores centrais enquanto que novos valores marginais são inseridos. Por exemplo: foram utilizados os valores 4, 5 e 6 para o parâmetro *min\_child\_weight*, sendo o 4 o valor que apresentou o melhor resultado. Na próxima rodada, portanto, são testados os valores 3, 4 e 5 para verificar se os valores da borda podem desbancar o valor central como o mais otimizado. O processo é repetido até que não se consiga avançar em ganhos de desempenho e então passa-se para outro grupo de parâmetros, até que o melhor cenário possível seja definido.

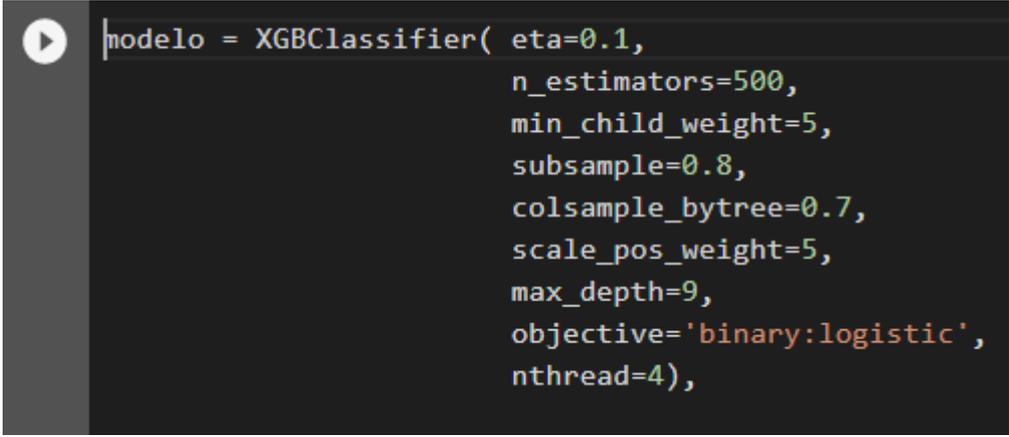
Embora a eficácia e a praticidade da técnica em encontrar a melhor personalização dos parâmetros não possa ser questionada, a questão sobre o gargalo de execução e processamento, mencionada no item 2.4.7 do capítulo 2, merece atenção.

Como o algoritmo faz o cruzamento de todos os parâmetros entre si, ao utilizarmos 3 parâmetros, com 3 valores cada e 5 partições, resulta em um total de 135 simulações ( $3 * 3 * 3 * 5$ ) e um tempo total de execução de mais de 30 minutos, mesmo rodando sobre o já mencionado – e mais enxuto – recorte de 30 mil registros do *dataset* original. Há de se pontuar, é claro, que a simulação citada foi escolhida para ilustrar este exemplo, especialmente, por ser mais custosa. Foram testados os valores de *eta*, parâmetro que representa a taxa de aprendizagem e, quanto menor forem seus valores de teste, maior será a necessidade computacional e, conseqüentemente, o tempo de processamento.

Contudo, mesmo que a simulação da vez não envolva o parâmetro da taxa de aprendizagem, há de se ter cautela ao executar os testes com o Grid Search CV: se apenas 1 novo parâmetro, com o mesmo número de valores, fosse adicionado ao conjunto anterior, por exemplo, se passaria de 135 para 405 simulações ( $3 * 3 * 3 * 3 * 5$ ) e um aumento de, no mínimo, 3 vezes no tempo de execução dos testes.

Após centenas de cruzamentos entre parâmetros e milhares de combinações testadas, a Figura 22 mostra os valores indicados como os mais eficientes para o *dataset*.

Figura 22 - Valores otimizados de acordo com o Grid Search CV

A screenshot of a code editor with a dark background and light text. On the left side, there is a play button icon. The code displayed is a Python function call for an XGBClassifier model with the following parameters: eta=0.1, n\_estimators=500, min\_child\_weight=5, subsample=0.8, colsample\_bytree=0.7, scale\_pos\_weight=5, max\_depth=9, objective='binary:logistic', and nthread=4.

```
modelo = XGBClassifier( eta=0.1,
                        n_estimators=500,
                        min_child_weight=5,
                        subsample=0.8,
                        colsample_bytree=0.7,
                        scale_pos_weight=5,
                        max_depth=9,
                        objective='binary:logistic',
                        nthread=4),
```

Fonte: Autores (2021)

Cada um destes parâmetros possui sua importância na construção do modelo. Eis os respectivos papéis de cada um.

**Eta:** Representa a taxa de aprendizagem, chamada de eta na documentação oficial do XGB. A taxa de aprendizagem é o ajuste aplicado ao algoritmo a cada etapa de iteração enquanto se busca o menor resultado possível para a função de perda. Se muito baixa torna o aprendizado muito lento e, em determinados cenários, inviável, porém, se muito alta impede a convergência do processo de aprendizado.

**N\_estimators:** Refere-se ao número de árvores de decisão que serão criadas pelo modelo durante o treinamento.

**Objective:** Tipo da função de perda a ser minimizada. Por conta do problema exigir uma classificação binária – fraude ou não-fraude, foi utilizada a função *binary:logistic* que retorna as probabilidades de uma amostra pertencer a uma ou outra classe.

**Min\_child\_weight:** Define a soma mínima dos pesos necessária para a árvore seguir sendo particionada. Durante o processo de desmembramento, caso uma folha tenha a soma de pesos menor do que o estipulado neste parâmetro então o processo é interrompido. Quanto mais alto este valor, mais conservador será o algoritmo e menos relações superespecíficas serão aprendidas. É, portanto, um parâmetro utilizado para controlar o *overfitting*. Porém, valores excessivamente altos podem resultar em um modelo com características opostas: *underfitting*, ou seja, que não fez ligações o suficiente para aprender com os dados e, por consequência, não apresentará bons resultados.

**Max\_depth:** Representa a profundidade máxima da árvore. Assim como o parâmetro anterior, possui a mesma relação de controle com *overfitting*. Se muito alto vai permitir que o modelo aprenda relações muito específicas.

**Subsample:** Determina a fração de dados de treinamento aleatória que será repassada a cada árvore antes delas aumentarem mais 1 nível. Igualmente aos anteriores, um valor muito baixo pode ocasionar *underfitting* enquanto que valores muito altos podem causar *overfitting*.

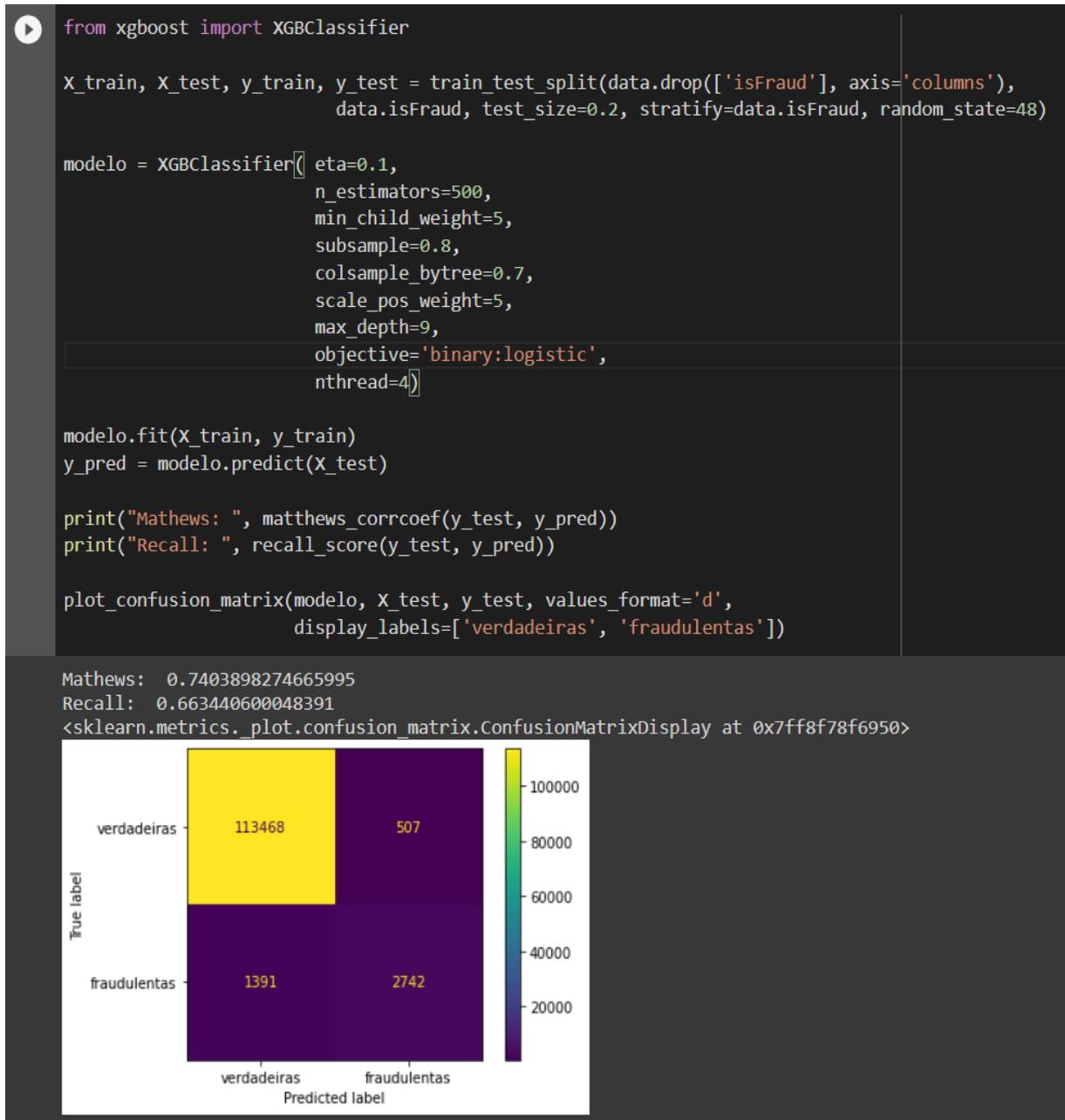
**Colsample\_bytree:** Similar ao anterior, determina a fração de colunas que serão fornecidas aleatoriamente no momento de criação das árvores.

**Scale\_pos\_weight:** Controla o balanço entre pesos positivos e negativos. É recomendado para casos com grande desbalanceamento entre classes.

**Nthread:** Garante que o modelo será criado utilizando processamento paralelo a fim de ganho de desempenho.

Ao rodar o treinamento utilizando os valores recomendados pelo *Grid Search CV* nota-se uma grande melhora nos valores, alcançando 0,66 de revocação, como ilustra a Figura 23.

Figura 23 - Resultados da classificação após otimização dos parâmetros



Fonte: Autores (2021)

No entanto, para que um modelo que se destina a identificar transações fraudulentas seja proveitoso, é necessário incrementar ainda mais os valores de *recall*. Por conta disso implementou-se uma outra técnica chamada de *Balanced Bagging Classifier*.

Incluída na biblioteca Imbalanced Learn, um pacote de soluções voltado a *datasets* desbalanceados, a ferramenta faz o *resampling* para diminuir os problemas causados pela

disparidade entre as classes. *Resampling* é o nome genérico do processo de balanceamento que emprega a tática de *oversampling* (criar registros na classe menos representada) ou *undersampling* (remover registros da classe dominante).

Após a implementação da ferramenta de *resampling* o modelo teve desempenho como o esperado, como mostra a Figura 24.

Figura 24 - Resultados do treinamento após utilização do Balanced Bagging Classifier



Fonte: Autores (2021)

A configuração utilizada com esta ferramenta é bastante simples e apenas foram utilizados dois parâmetros. São eles:

*Sampling\_strategy*: Parâmetro que configura qual estratégia será utilizada para efetuar o *sampling*. Ao utilizar o método “all”, são organizadas ambas as classes. Testes foram feitos com as demais opções para este parâmetro, mas não obtiveram resultados mais satisfatórios.

*N\_jobs*: Assim como no XGB, configura-se este valor para que o algoritmo utilize processamento paralelo. O valor -1 permite a utilização de tantas threads quanto necessárias..

Um ponto que chama a atenção após a aplicação do BBC é que o número de falsos positivos aumentou consideravelmente, passando de pouco mais de 500 para mais de 21 mil, porém, em contrapartida, a identificação de transações falsas aumentou de 2742 para 3728, ou seja, um aumento de 36% no bloqueio de transações falsas que passariam despercebidas e acabariam gerando prejuízos para a instituição financeira. Assim, certamente, este sacrifício do valor da Correlação do Coeficiente de Mathews para gerar ganho em revocação é proveitoso neste cenário.

O Apêndice A mostra a estrutura de uma das 500 árvores de decisão criadas pelo modelo. Para criar a representação da árvore foi utilizada a própria biblioteca Xgboost em conjunto com a biblioteca Matplotlib. O processo é apresentado na Figura 25.

Figura 25 - Código utilizado para exportar como imagem uma das árvores de decisão criadas

```
from xgboost import plot_tree
import matplotlib.pyplot as plt

plot_tree(modelo)
fig = plt.gcf()
fig.set_size_inches(150, 45)
plt.savefig("arvore.png", bbox_inches = 'tight')

plt.show()
```

Fonte: Autores (2021)

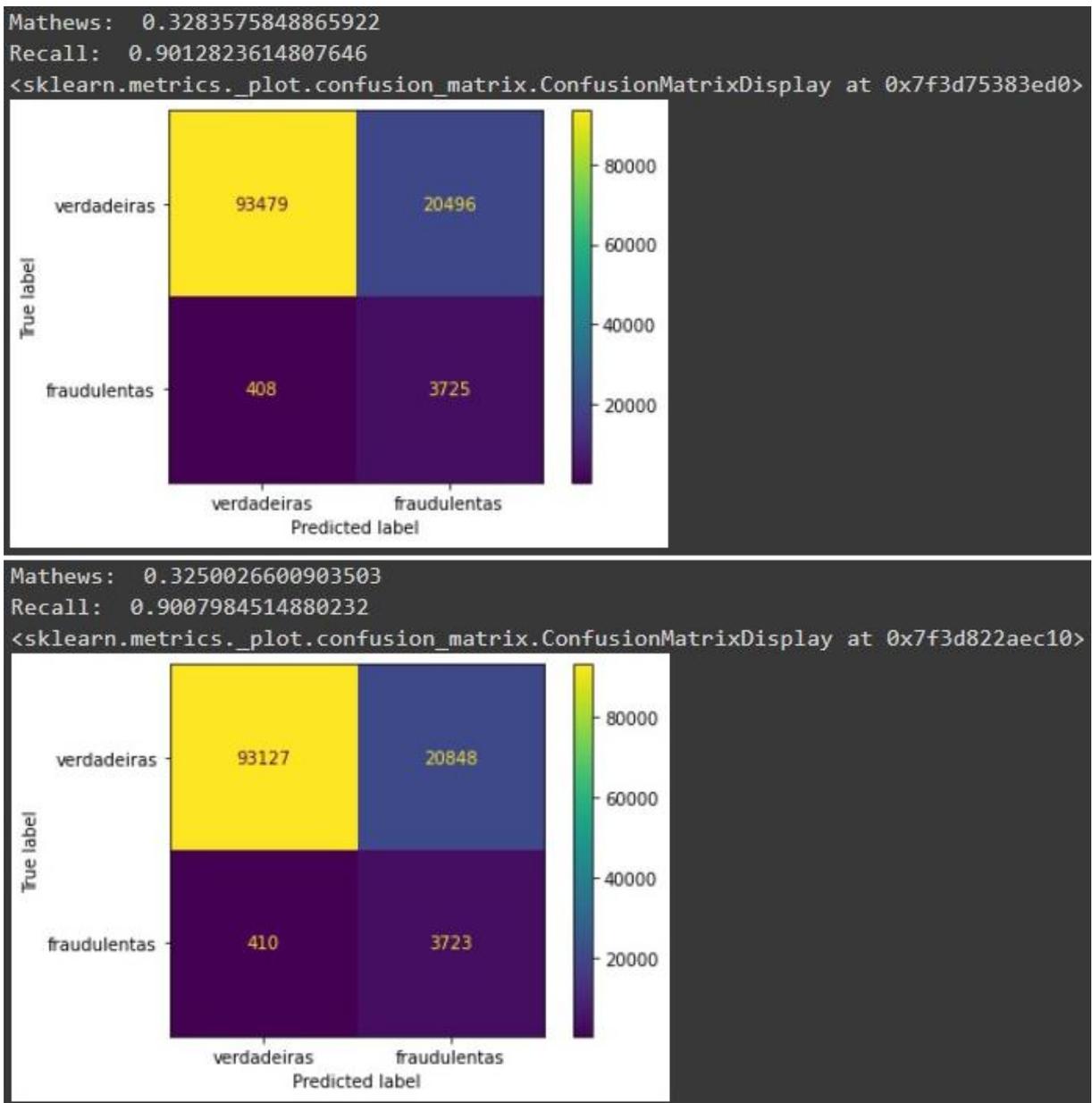
Por fim, após alcançar resultados satisfatórios na classificação, o próximo passo é criar a versão distribuível da aplicação, mas antes, foi necessário realizar uma atualização sobre os valores aplicados na etapa de pré-processamento dos dados.

Se no Random Forest foi utilizado a média para compor os valores *NaN*, com o XGB utilizou-se o valor zero para preenchimento. A explicação é que, embora a utilização do valor médio da coluna também apresentasse um melhor resultado quando utilizado com o algoritmo XGB, o tamanho do arquivo após passar pelo *pipeline* não justificava a manutenção do método mais custoso.

Utilizando o XGB e a opção de média para lidar com os valores *NaN* chegou-se a uma revocação de 0,901, contra 0,9 quando da inserção do zero. A diferença mais significativa, porém, reside no fato de que no primeiro caso a saída é um arquivo .CSV com cerca de ~609 MB, enquanto que a segunda opção resulta em um .CSV de ~303 MB.

Assim, optou-se pelo método que entrega um valor final “inferior” por conta da significativa diferença no tamanho do arquivo final que será utilizado para o treinamento. Acredita-se que valha a pena sacrificar esses milésimos de pontuação – duas transações em um universo de mais de 4.100, como pode-se ver na Figura 26 – por um significativo ganho de eficiência no tempo de treinamento do modelo.

Figura 26 - Comparação entre os métodos citados. Acima a aplicação da média, abaixo a utilização do zero



Fonte: Autores (2021)

#### 4.6 Criação do aplicação distribuível e seu funcionamento

Com o modelo desenvolvido e executando conforme o esperado, o próximo objetivo a ser alcançado nesta pesquisa é a criação de um script que possa ser implementado localmente por quem desejar e que seja funcional com qualquer base de dados a ser utilizada. Como o

Google Colab não tem a capacidade de gerar scripts, nesta etapa do projeto passou-se a utilizar o ambiente de desenvolvimento Pycharm.

A base do código utilizado neste script é a mesma que fora criada, previamente, no Google Colab e explicada nas seções anteriores deste capítulo. A partir desta solução inicial foi implementada uma série de adaptações para adequar os procedimentos à nova dinâmica de execuções locais estruturadas.

Diferentemente do código estruturado e linear dos testes iniciais, para otimizar o código foram criadas 7 funções. São elas:

**def menu():** Uma das funções básicas da aplicação, ela é chamada no início da execução do programa e após o término de qualquer uma das demais etapas. Com isso pode-se dar continuidade às operações de treinamento e validação sem a necessidade de ter que encerrar o script e executá-lo novamente para iniciar uma nova tarefa.

**def carregar():** A outra função básica, é executada sempre que há uma interação do programa com um arquivo local, como *datasets*, arquivos de configuração ou modelos treinados.

**def preProcessamento():** Esta função é responsável por aplicar ao *dataset* o mesmo pré-processamento de dados que foi desenvolvido no Google Colab e já documentado. Ao final de sua execução terão sido gerados dois arquivos: um *.CSV* chamado **pre\_processamento\_concluido**, pronto para passar pela etapa de treinamento e um *.TXT* com a descrição das colunas restantes após a aplicação do processo de análise de dados correlacionados. Este txt terá sua importância explicada nas próximas funções.

**def otimizar():** Caso o usuário não queira utilizar as configurações padrão fornecidas com o script para treinar o seu modelo, poderá utilizar esta função para iterar com o Grid Search CV em busca das melhores opções para o seu *dataset*. No término da execução é gerado um *.TXT* com os novos valores a serem utilizados no treinamento.

**def treinar():** Nesta função é feito o treinamento com base nas configurações padrão ou otimizadas, de acordo com a escolha do usuário. É nesta função que será criada a solução distribuível a ser utilizado posteriormente.

**def padronizar(data):** Esta função é responsável por ajustar os dados de entrada (novas transações a serem classificadas) às mesmas configurações de colunas que o modelo de ML foi configurado após a etapa de pré-processamento. Trata-se de uma função interna chamada pela função classificar e não pode ser chamada diretamente pelo usuário.

**def classificar(modelo):** Finalmente, esta função será a responsável por classificar as transações em busca das fraudulentas. Sua saída é um .CSV baseado no arquivo de testes com uma coluna binária extra que informa se a transação foi classificada como maliciosa (valor 1) ou como legítima (valor 0). Para fins de testes, como veremos no capítulo 6, esta função recebe como entrada um arquivo no formato .CSV contendo milhares de transações a serem classificadas simultaneamente. Porém, como um dos objetivos desta pesquisa é criar uma ferramenta que possa rodar em tempo real, indicando se uma operação deve ser aprovada ou negada, ela pode, facilmente, ser convertida em uma lógica na qual é passado apenas um registro e, em milésimos de segundos, gera-se o resultado quanto à sua veracidade.

Logo que o script é iniciado é feita uma conferência na pasta raiz onde o mesmo está rodando. A partir dos arquivos que forem encontrados na pasta – ou não encontrados na pasta – a aplicação toma caminhos diferentes. Como em um ambiente real a ferramenta será mais utilizada para classificar transações do que para treinar modelos, caso o programa detecte que já exista um modelo treinado e pronto para classificar, o programa oferece ao usuário a opção para que ele comece as classificações imediatamente. Do contrário, o menu principal é invocado e todas as demais opções podem ser acessadas.

Para carregar os arquivos o programa foi construído com uma lógica de *Try, Except* na qual a falha em carregar um arquivo necessário para o seu correto funcionamento irá ocasionar avisos e, nos piores casos, como a falta do arquivo de configuração padrão, o encerramento da aplicação, conforme ilustra a Figura 27.

Figura 27 - Utilização da lógica *Try, Except* para carregamento dos arquivos

```
try:
    arquivo = os.listdir(f"{raiz_projeto}/Dataset")

    print("\nQual arquivo você quer usar nesta operação?\n")
    for i in range(len(arquivo)):
        print(f"{i + 1} - {arquivo[i]}")
    op = int(input("\nResposta: "))

    print("\nCarregando dados.\nIsto pode demorar de acordo com o tamanho do arquivo")
    data = pd.read_csv(raiz_projeto + '/Dataset/' + arquivo[op - 1])
    print("\nArquivo " + arquivo[op - 1] + " carregado com sucesso")

    return data

except:
    print("\nNão foi possível carregar o arquivo\nPor favor verifique se o mesmo encontra-se na"
          " pasta 'Dataset' dentro da raiz do projeto e se possui a extensão .CSV")
    sys.exit()
```

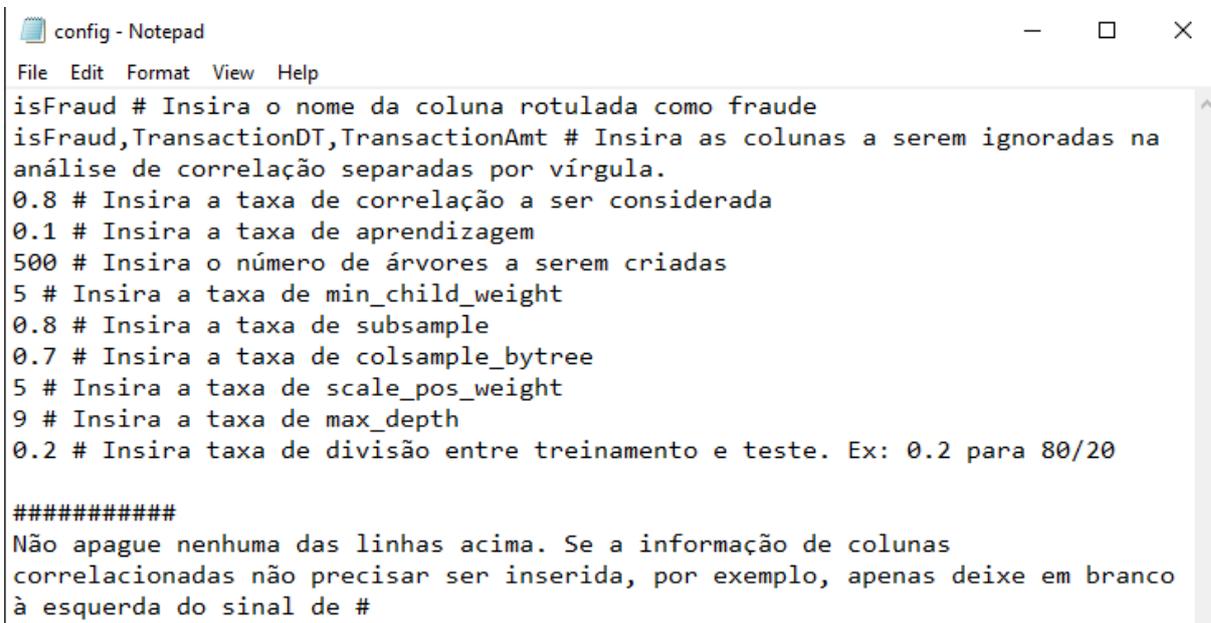
Fonte: Autores (2021)

Entre os arquivos vitais para o funcionamento da aplicação estão o *dataset* no formato .CSV e o arquivo de configuração padrão no formato .TXT. Este arquivo de configuração tem as definições dos melhores parâmetros de acordo com os testes realizados com o Grid Search CV durante a otimização do algoritmo XGB, documentadas na seção 4.3 deste capítulo.

Além dos campos referentes às configurações básicas, o arquivo conta também com um campo para que o usuário indique quais as colunas numéricas que devem ser ignoradas na etapa de análise de correlação e o nome da coluna referente à classificação supervisionada dos dados. Como o programa faz a leitura dos valores linha a linha capturando tudo que está à esquerda do sinal de “#”, é fundamental que nenhuma delas seja apagada, mesmo que desnecessária, como é o caso da linha referente à análise de colunas correlacionadas que, em alguns *datasets*, pode não ser utilizada.

Este arquivo de configuração padrão, portanto, deve ser adequado ao *dataset* da vez, antes da primeira execução. A Figura 28 mostra o arquivo, tal qual se encontra na versão distribuída para download.

Figura 28 - Arquivo de configuração padrão da aplicação



```
config - Notepad
File Edit Format View Help
isFraud # Insira o nome da coluna rotulada como fraude
isFraud,TransactionDT,TransactionAmt # Insira as colunas a serem ignoradas na
análise de correlação separadas por vírgula.
0.8 # Insira a taxa de correlação a ser considerada
0.1 # Insira a taxa de aprendizagem
500 # Insira o número de árvores a serem criadas
5 # Insira a taxa de min_child_weight
0.8 # Insira a taxa de subsample
0.7 # Insira a taxa de colsample_bytree
5 # Insira a taxa de scale_pos_weight
9 # Insira a taxa de max_depth
0.2 # Insira taxa de divisão entre treinamento e teste. Ex: 0.2 para 80/20

#####
Não apague nenhuma das linhas acima. Se a informação de colunas
correlacionadas não precisar ser inserida, por exemplo, apenas deixe em branco
à esquerda do sinal de #
```

Fonte: Autores (2021)

Caso o usuário não queira, no entanto, utilizar as configurações padrão, poderá recorrer à opção que faz a otimização dos valores de acordo com as particularidades da base de dados que estiver sendo utilizada.

Ao selecionar esta opção o programa irá, automaticamente, rodar o algoritmo Grid Search CV para os oito parâmetros definidos na função, variando de 4 até 6 valores para cada um deles. Para garantir a possibilidade real de otimização, a ordem de teste dos parâmetros e dos respectivos valores é escolhida de maneira aleatória através da biblioteca Random.

No início de cada rodada, o usuário é informado dos parâmetros e valores selecionados para o teste. Da mesma forma, no final da etapa são informados os que, em conjunto, tiveram o melhor *score* com base no critério de revocação e que, portanto, serão mantidos para as próximas rodadas.

Para não cair no já referido gargalo de execução deste algoritmo, os testes são feitos com 3 valores para cada um dos 3 parâmetros selecionados a cada rodada. O número de *folds* é definido pelo usuário no início do processo. O treinamento otimizado é encerrado ao restar somente 1 valor para cada parâmetro, ou seja, o valor otimizado. Estes valores são exportados em um arquivo .TXT para a pasta raiz do script e segue a mesma estrutura do arquivo de configuração padrão visto na Figura 28.

O intervalo de valores a serem testados para cada um dos parâmetros foi definido com base na documentação da biblioteca, que indica o melhor *range* para cada campo, conforme Figura 29.

Figura 29 - Parâmetros que serão testados utilizando o Grid Search CV e seus respectivos valores

```
parametros = [['eta', 0.05, 0.1, 0.15, 0.2, 0.25, 0.3],  
              ['n_estimators', 300, 400, 500, 600, 700, 800],  
              ['min_child_weight', 3, 4, 5, 6, 7],  
              ['subsample', 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],  
              ['colsample_bytree', 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],  
              ['scale_pos_weight', 3, 4, 5, 6, 7],  
              ['max_depth', 5, 6, 7, 8, 9, 10],  
              ['split', 0.2, 0.25, 0.3, 0.35]]
```

Fonte: Autores (2021)

É importante destacar que para que os cruzamentos gerem melhores combinações e valores na etapa de configuração otimizada é necessário que seja utilizado um *dataset* que tenha passado pela etapa de tratamento de dados adequada. Para isso, a aplicação oferece a função **preProcessamento()**, que aplica, automaticamente, o *pipeline* de pré-processamento ao conjunto de dados escolhido.

Utilizando um arquivo .CSV de entrada, todas as operações já detalhadas em ocasiões anteriores são empregadas novamente, resultando em um novo arquivo .CSV chamado **pre\_processamento\_concluido.CSV** que deverá ser usado na etapa seguinte: o treinamento.

Mas antes, cita-se outro arquivo de tamanha importância criado durante esta fase: o **colunas.TXT**. Este arquivo é uma lista nominal de todas as colunas que serão usadas para criar o modelo. Sua importância será explicada em breve, ao tratar das atividades de classificação.

Com os arquivos de configuração definidos e com o pré-processamento realizado, o modelo pode, finalmente, ser treinado.

A aplicação utiliza o algoritmo *Extreme Gradient Boosting* configurado com os valores padrão ou otimizados e, em conjunto com o *Balanced Bagging Classifier*, cria o modelo utilizado para classificação das futuras transações. A novidade desta etapa é a utilização da biblioteca *Joblib*, implementada para criar a solução distribuível.

Com este arquivo, gerado no final do processo, o software efetua a classificação das transações a cada vez que for iniciado, sem ter de refazer todas as etapas anteriores novamente. É a mesma biblioteca Joblib que se encarrega de carregar o arquivo salvo localmente.

Após o modelo estar treinado as transações podem ser classificadas. O processo responsável por esta etapa é tão simples quanto o de classificação, consistindo em apenas 3 linhas de código: Uma para carregar os dados, outra para classificar e uma terceira para salvar o arquivo “**classificacoes.CSV**” contendo os resultados.

O ponto mais interessante da função **classificar()**, porém é a função interna **padronizar()**. Sua importância é justificada pela seguinte lógica: o *dataset* – ou registro de transação individual – a ser classificado deve ter a mesma quantidade de colunas que o arquivo que foi utilizado para gerar o modelo. O que parece simples em um teste acadêmico com variações controladas pode ser algo bastante complicado em um ambiente real.

Imagine o cenário cuja instituição financeira extraiu de seu banco de dados um conjunto de informações sobre suas transações, contendo centenas de milhares de registros, distribuídos em 500 colunas. Após realizarem todo o pré-processamento dos dados e remover as colunas desnecessárias, o modelo criado foi configurado com cerca de 250 colunas.

Assim, a cada nova classificação que se pretenda fazer utilizando este modelo, esta instituição financeira teria de aplicar os mesmos passos de pré-processamento para chegar nas mesmas 250 colunas e poder submeter o registro para classificação, afinal, o registro a ser analisado saiu do mesmo banco de dados, com as mesmas 500 colunas originais.

Para que esta etapa não seja um ponto de lentidão no processo, uma análise real deve ser capaz de ser realizada em segundos. Assim, foi implementada a função **padronizar()**. Utilizando o já citado arquivo **colunas.TXT** contendo todas as colunas utilizadas na criação do modelo – e, portanto, necessárias para o treinamento – esta função, chamada internamente pela função **classificar()**, prepara os dados a serem classificados para que não haja incompatibilidade entre eles e o modelo. Com esta função o processo ocorre de maneira muito mais rápida do que se tivesse de ser reproduzido através do *pipeline* convencional que aplicado às etapas iniciais de tratamento de dados. Além de remover as colunas desnecessárias, a função também preenche os valores *NaN*. A função é mostrada na Figura 30.

Figura 30 – Função capaz de preparar rapidamente os registros para classificação

```
def padronizar(data):  
  
    file = open(raiz_projeto + "/colunas.txt", "r")  
    colunas = file.read().splitlines()  
    file.close()  
  
    for i in data.columns:  
        if i not in colunas:  
            data.drop([i], axis='columns', inplace=True)  
        else:  
            data[i].fillna(0, inplace=True)  
  
    return data
```

Fonte: Autores (2021)

Com o arquivo de classificação gerado basta ao usuário fazer a análise que deseja dos resultados, se análise do desempenho, como em um viés acadêmico, nos moldes deste trabalho, ou se utilizando dos resultados para basear a aprovação de transações em um ambiente de produção.

Em aspectos gerais sobre a aplicação, também foi adicionada uma funcionalidade que registra o tempo necessário para cada etapa. Esta funcionalidade se fez necessária para termos a noção não só do tempo consumido durante as execuções, como para projeções do que será necessário para se treinar, modelar e classificar conjuntos de dados mais robustos.

Na Figura 31, pode-se visualizar os resultados da execução da primeira rodada de testes na qual se busca definir as configurações otimizadas a serem utilizadas no futuro treinamento do modelo. Neste exemplo observa-se que foram necessários mais de 25 horas para completar o processo iterando 10 *folds* sobre um recorte do *dataset* com 300 mil registros.

Figura 31 – Tempo necessário para executar 10 folds sobre um dataset de 200 mil registros

```
Round 6
Valores testados: {'eta': [0.25, 0.1, 0.15], 'colsample_bytree': [0.4, 0.5], 'max_depth': [10, 8, 7]}
Melhores parâmetros: {'colsample_bytree': 0.4, 'eta': 0.1, 'max_depth': 7}
Tempo necessário para o round de treinamento: 15405.092 segundos

Round 7
Valores testados: {'max_depth': [5, 7], 'subsample': [0.7, 0.8], 'eta': [0.1, 0.05]}
Melhores parâmetros: {'eta': 0.05, 'max_depth': 5, 'subsample': 0.7}
Tempo necessário para o round de treinamento: 14717.405 segundos

Tempo total: 92624.535 segundos
Arquivo otimizado.txt exportado para a pasta raiz da aplicação
```

Fonte: Autores (2021)

Com essa informação pode-se planejar que, para rodar sobre o conjunto completo de dados do *dataset* (cerca de 600 mil registros), seria necessário dispor deste tempo multiplicado por 2. Se o usuário resolver dobrar para 20 folds, valor indicado pela literatura como de melhor resultado (UYEKITA, 2019), multiplica-se novamente este tempo por 2. A previsão não pode ser tomada como certeza, é claro, mas serve como um ponto inicial para o planejamento.

Considerando que um conjunto de 600 mil registros pode ser interpretado como um *dataset* pequeno – o conjunto de dados utilizado na pesquisa resenhada na seção 3.4.2 do capítulo 3 tinha mais de 30 milhões de registros – conclui-se que a implementação de funções de paralelização será muito bem-vinda para as próximas versões desta aplicação.

No capítulo seguinte foi simulada a utilização da ferramenta como se manipulada por um usuário real. Nele descreveremos como a questão do tempo acabou se tornando o ponto de maior sensibilidade.

## 5 Testes e validação

Este capítulo apresenta as simulações realizadas sobre o conjunto de dados utilizado, juntamente com os resultados atingidos. Para validar os resultados efetuamos o passo a passo que um analista de uma instituição financeira aplicaria em seu trabalho ao executar a ferramenta desenvolvida.

Para executar o script, o arquivo **classificador.py**, mais especificamente, pode-se utilizar qualquer IDE que ofereça suporte a projetos desenvolvidos na linguagem Python, ou, até mesmo, o próprio terminal de comandos do Windows. Independentemente da opção escolhida, é necessário ter o Python instalado na máquina. A aplicação não foi desenvolvida visando ao uso em sistemas operacionais baseados em Linux, Mac OS ou quaisquer outros.

Nos testes apresentados neste capítulo foi utilizada uma máquina equipada com processador Intel Core i7-4720HQ de 4 núcleos e suporte a até 8 threads, placa de vídeo Nvidia GeForce GTX 960M com 2 GB de memória dedicada, 8 GB de memória RAM e SSD instalado. O sistema operacional é Windows 10 e o ambiente de desenvolvimento é o Pycharm.

Neste processo foi utilizado o mesmo *dataset* manipulado ao longo da pesquisa, o qual, para este cenário hipotético, foi dividido em 2 arquivos: **treinamento.CSV**, contendo cerca de 550 mil transações e **classificacao.CSV**, com, aproximadamente, 40 mil transações inéditas para o modelo a fim de gerar uma classificação totalmente isenta. Ressalta-se que os recortes mantiveram taxas de transações falsas similares, sendo 3,51% 3,43%, respectivamente.

Inicialmente, ambos os arquivos contam com a mesma estrutura de colunas, como se houvesse sido extraídas do banco de dados da instituição através do mesmo processo de ETL, como se imagina que ocorra no dia a dia destas corporações.

### 5.1 Testes

O processo inicia-se após o analista efetuar o download e descompactar o arquivo para um local de sua máquina a escolher. Nela o usuário terá uma pasta com os seguintes arquivos: **classificador.py**, **config.txt** e um diretório vazio chamado **Dataset**, conforme ilustra a Figura 32.

Figura 32 - Estrutura inicial da aplicação

Name	Date modified	Type	Size
Dataset	29/06/2021 17:07	File folder	
classificador	29/06/2021 16:38	JetBrains PyCharm	16 KB
config	29/06/2021 17:34	Text Document	1 KB

Fonte: Autores (2021)

A primeira ação a ser tomada é a configuração do ambiente para a execução inicial. Para isso deve-se editar o arquivo **config.txt** adicionando o rótulo da coluna de fraudes e das colunas a serem ignoradas na ação da análise de colunas correlacionadas, além de adicionar o **treinamento.CSV** na pasta **Dataset** para que a aplicação possa carregar o arquivo necessário corretamente. Após realizar estas etapas já é possível iniciar o programa.

Na primeira execução, logicamente, não será exibida ao analista a opção inicial de classificar transações, pois o modelo de *machine learning* ainda não foi criado. Dessa forma, o analista verá em sua tela o menu principal com todas as opções disponíveis, conforme ilustra a Figura 33.

Figura 33 - Menu principal da aplicação

```
C:\Users\mmeyer1\Anaconda3\python.exe C:/Users/mmeyer1/PycharmProjects/Simulação/classificador.py
Selecione uma opção:
1 - Aplicar o pré processamento ao dataset (Escolha esta opção se for a primeira vez que utiliza o mesmo)
2 - Treinar o modelo utilizando a configuração padrão
3 - Treinar o modelo Utilizando a configuração otimizada
4 - Testar combinações de parâmetros e valores e gerar configuração otimizada
5 - Classificar transações usando o modelo treinado
6 - Ajuda
7 - Sobre esta aplicação
Resposta:
```

Fonte: Autores (2021)

Como o próprio console explicita, a primeira medida a ser tomada ao trabalhar com um novo *dataset* é aplicar o pré-processamento aos dados através da opção 1 do menu. O programa pergunta então, tarefa padrão, qual será o arquivo utilizado na etapa escolhida. Como há apenas

1 arquivo na pasta, basta informar o número que o identifica e na listagem o processo começa a ser executado.

Dois arquivos serão exportados ao final da execução deste *pipeline*, minutos depois. O primeiro deles, chamado de **pre\_processamento\_concluido.csv** é o resultado do *dataset* de entrada após passar pelo tratamento inicial dos dados e ter as colunas correlacionadas e não-numéricas removidas e os valores *NaN* preenchidos. Este arquivo será exportado para a pasta **Dataset** e será utilizado em todas as etapas futuras de treinamento e otimização.

O segundo arquivo criado na conclusão da etapa de pré-processamento é o **colunas.TXT**, que será utilizado durante o processo de classificação para garantir que as transações a serem classificadas terão a mesma configurações de colunas que o modelo criado. Sem essa padronização é impossível utilizar, em conjunto, o modelo e os dados a serem classificados, conforme ilustra a Figura 34.

Figura 34- Etapa de pré-processamento concluída

```
Qual arquivo você quer usar nesta operação?

1 - treinamento.csv

Resposta: 1

Carregando dados.
Isto pode demorar de acordo com o tamanho do arquivo

Arquivo treinamento.csv carregado com sucesso

Pré processamento dos dados iniciado

Colunas correlacionadas removidas
Colunas não numéricas removidas
Valores not a number preenchidos

Pré processamento concluído
Arquivo pre_processamento_concluido.csv exportado para a pasta Dataset

Tempo total: 53.156 segundos
```

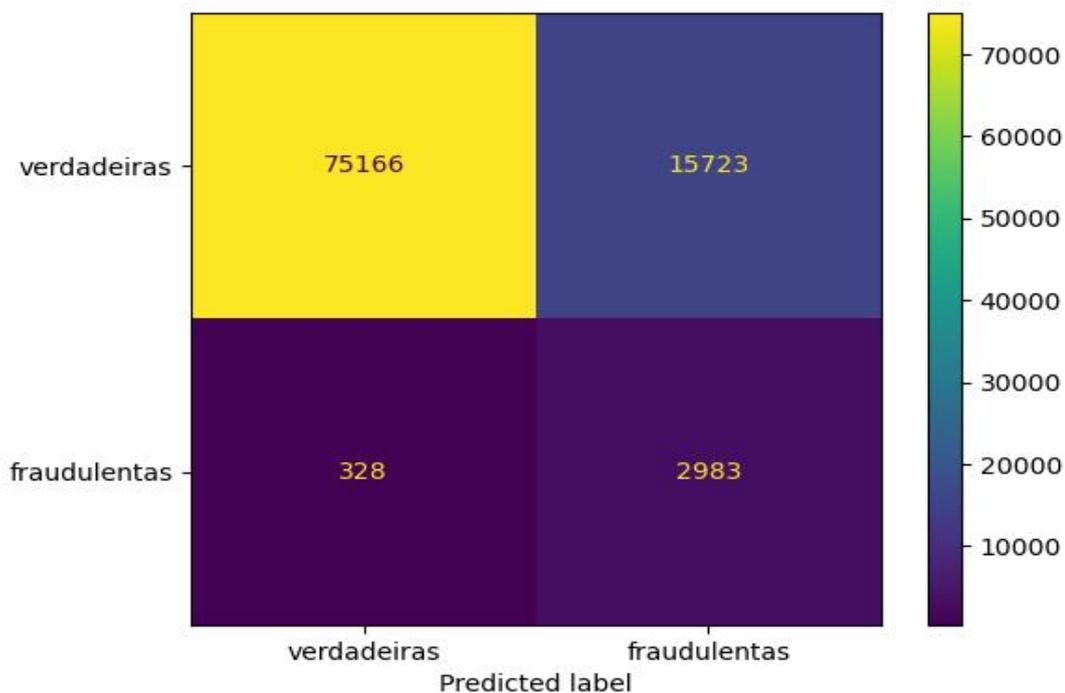
Fonte: Autores (2021)

Em seguida o programa retorna para o menu principal, a partir do qual o analista já pode realizar o treinamento do modelo, contudo usando as configurações padrão apenas, já que os valores otimizados ainda não foram validados.

Assim, ele executa a primeira classificação com os valores padrão para ver como o modelo se comporta. Como dito anteriormente, as etapas de treinamentos devem ser feitas com o arquivo CSV gerado no final da etapa anterior, conforme ilustra a Figura 35.

Figura 35 - Resultado do treinamento do modelo utilizando a configuração padrão

```
Algumas informações sobre o seu dataset:  
  
Transações totais: 471000  
Transações verdadeiras: 454447  
Transações fraudulentas: 16553  
Porcentagem de fraudes: 3.514%  
  
Modelo gerado e exportado para a pasta raiz da aplicação  
  
Seus valores de treinamento:  
Mathews: 0.3360370488270655  
Recall: 0.9009362730292962  
  
Tempo total: 281.308 segundos
```



Fonte: Autores (2021)

Embora tenha obtido números expressivos nos testes com a configuração padrão, o analista resolveu ir mais a fundo e testar a opção que faz a busca automática pelos melhores valores. Utilizando o mesmo arquivo .CSV ele configura como 20 o número de  *folds*, conforme indica a literatura. E aí se mostra o maior gargalo da aplicação.

Mesmo rodando em uma máquina com configurações acima da média, com 4 cores e 8 threads, o tempo necessário para conclusão de todos os cruzamentos possíveis foi de, impressionantes, 66 horas. Os motivos desse tempo de processamento fora da curva são os mesmos já relatados, quando da explicação sobre o funcionamento do  *Grid Search CV*.

Ao fazer com que o algoritmo cruzasse todas as possibilidades oferecidas por 3 parâmetros, com 3 valores cada, em uma divisão de 20  *folds*, o analista fez com que 540 operações de treinamento do modelo fossem executadas. Para tornar a operação ainda mais custosa, cada uma destas 540 operações rodava sobre um  *dataset* de 550 mil registros. Em outras palavras, se cada registro for analisado 1 única vez por treinamento a cada  *fold*, cada rodada fará, aproximadamente, 300 milhões de análises de registros de transações. Destaca-se que todas estas operações são referentes a 1 única rodada de treinamento e que a ferramenta pode executar até 8 rodadas<sup>26</sup>. A Figura 36 mostra as etapas sendo executadas.

Como será relatado na conclusão, o ponto de otimização do tempo de processamento, certamente, poderá ser aprofundado por pesquisas futuras.

---

<sup>26</sup> Ressalta-se que a medida em que as rodadas vão avançando e eliminando os valores menos eficientes, as possibilidades de cruzamento diminuem, logo, diminui-se também o tempo necessário para a execução das últimas etapas. Ao chegar na 8ª e última rodada, por exemplo, somente resta 1 parâmetro com 2 valores para teste.

Figura 36 - Tempo necessário para executar o Grid Search CV com 20 folds

```
Quantos folds por rodada? Valor sugerido mínimo 10: 20

Round 1
  Valores testados: {'subsample': [0.8, 0.4, 0.7], 'eta': [0.1, 0.05, 0.2], 'split': [0.35, 0.2, 0.3]}
  Melhores parâmetros: {'eta': 0.2, 'split': 0.35, 'subsample': 0.7}
  Tempo necessário para o round de treinamento: 42731.105 segundos

Round 2
  Valores testados: {'eta': [0.15, 0.2, 0.3], 'colsample_bytree': [0.7, 0.8, 0.6], 'max_depth': [10, 5, 9]}
  Melhores parâmetros: {'colsample_bytree': 0.8, 'eta': 0.15, 'max_depth': 10}
  Tempo necessário para o round de treinamento: 40083.096 segundos

Round 3
  Valores testados: {'scale_pos_weight': [3, 4, 5], 'eta': [0.15, 0.25], 'split': [0.35, 0.25]}
  Melhores parâmetros: {'eta': 0.15, 'scale_pos_weight': 5, 'split': 0.35}
  Tempo necessário para o round de treinamento: 18211.286 segundos

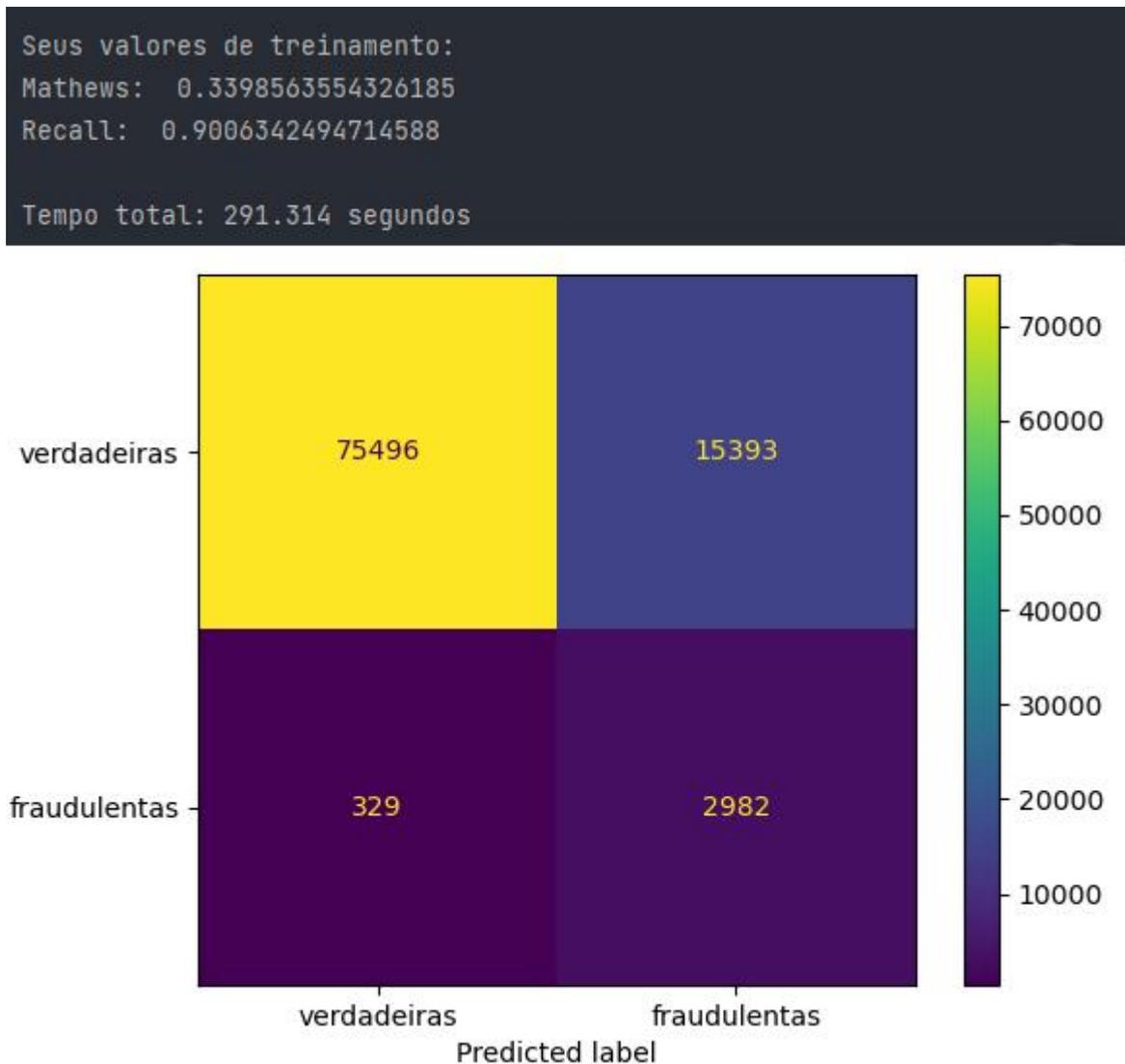
Round 4
  Valores testados: {'subsample': [0.9, 0.7, 0.6], 'min_child_weight': [6, 7, 3], 'max_depth': [7, 8, 6]}
  Melhores parâmetros: {'max_depth': 7, 'min_child_weight': 3, 'subsample': 0.7}
  Tempo necessário para o round de treinamento: 49159.378 segundos

Round 5
  Valores testados: {'min_child_weight': [4, 5, 3], 'subsample': [0.7, 0.5], 'max_depth': [10, 7]}
  Melhores parâmetros: {'max_depth': 7, 'min_child_weight': 3, 'subsample': 0.7}
  Tempo necessário para o round de treinamento: 26199.132 segundos
```

Fonte: Autores (2021)

No final do processo é criado mais um arquivo de texto chamado **otimizado.TXT**, que fornecerá os novos valores para rodar o treinamento do modelo. O próximo passo tomado pelo analista foi justamente executar este novo treinamento em busca de melhores resultados, conforme ilustra a Figura 37.

Figura 37 - Resultado do treinamento do modelo utilizando a configuração otimizada



Fonte: Autores (2021)

Com as duas possibilidades de treinamento disponíveis, o analista opta por aquela que apresenta um melhor resultado, embora com uma melhoria quase imperceptível: a configuração padrão.

Assim, a próxima etapa é realizar a classificação do modelo utilizando novamente o arquivo **pre\_processamento\_concluido.CSV**. No final da execução desta função o modelo de ML capaz de classificar transações finalmente será criado e exportado para a pasta raiz da aplicação.

Assim, da próxima vez que o analista iniciar o programa para classificar novas transações suspeitas – imagina-se que a função de classificação será o uso mais recorrente da ferramenta – poderá, rapidamente, carregar o modelo gerado neste passo e saber se determinada transação tem mais probabilidade de ser falsa ou legítima. A Figura 38 mostra a pasta raiz com todos os arquivos gerados.

Figura 38 - Classificação final dos arquivos da pasta raiz

Name	Date modified	Type	Size
Dataset	29/06/2021 18:36	File folder	
classificador	29/06/2021 18:41	JetBrains PyCharm	16 KB
colunas	29/06/2021 18:36	Text Document	1 KB
config	29/06/2021 17:34	Text Document	1 KB
modelo.pkl	29/06/2021 18:46	PKL File	51.372 KB
otimizado	29/06/2021 00:11	Text Document	1 KB

Fonte: Autores (2021)

A última etapa é extremamente rápida, uma vez que todo o trabalho pesado de pré-processamento e treinamento do modelo já está completo. Para isto basta que o analista selecione a opção correspondente no menu e depois selecione o arquivo a ser classificado, neste caso o **classificacao.CSV**. Após concluído o processo, o resultado será exportado através do arquivo **resultado\_classificacao.CSV**.

Como dito anteriormente, a classificação de um *dataset* com  $n$  registros é um tanto quanto improvável para um ambiente real como o inserido pelo analista deste exemplo, porém, é necessário para que se possa, academicamente, fazer a análise dos resultados.

## 5.2 Análise dos resultados

Com os testes finalizados, finalmente pode ser feita a análise para saber se o treinamento foi bem sucedido.

O melhor resultado nos testes foi com o conjunto de valores referente à configuração padrão, que marcou 0,9 de revocação no treinamento do modelo. Assim, chegou a hora de

analisar como ele se saiu ao classificar os valores inéditos reservados no *dataset classificacao.CSV*.

Para comparar os valores, foi utilizada uma função conhecida desta pesquisa, presente desde os testes iniciais, ainda usando RF, a **recall\_score()** da biblioteca Sklearn.

Figura 39 - Processo de comparação dos valores originais e preditos e o resultado da classificação

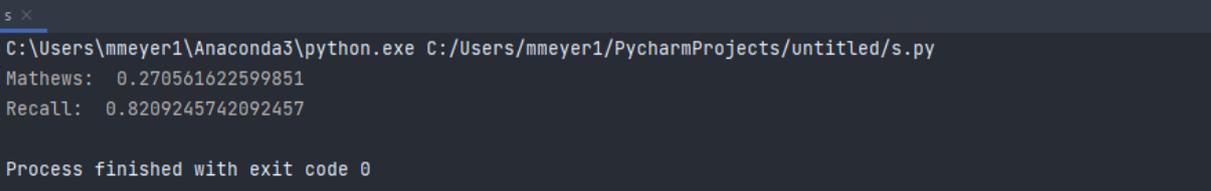
```
import pandas as pd
from sklearn.metrics import recall_score, matthews_corrcoef

gabarito = pd.read_csv('C:/Users/mmeyer1/PycharmProjects/Simulação/Dataset/gabarito.csv')
predicao = pd.read_csv('C:/Users/mmeyer1/PycharmProjects/Simulação/Dataset/resultado_classificacao.csv')

for i in gabarito.columns:
    if i != 'isFraud':
        gabarito.drop([i], axis='columns', inplace=True)

for i in predicao.columns:
    if i != 'isFraud':
        predicao.drop([i], axis='columns', inplace=True)

print("Mathews: ", matthews_corrcoef(gabarito, predicao))
print("Recall: ", recall_score(gabarito, predicao))
```



```
C:\Users\mmeyer1\Anaconda3\python.exe C:/Users/mmeyer1/PycharmProjects/untitled/s.py
Mathews:  0.270561622599851
Recall:  0.8209245742092457

Process finished with exit code 0
```

Fonte: Autores (2021)

Após este processo, observa-se pela Figura 39, que o classificador atingiu a marca de 0.82 de revocação sobre os valores preditos, o que configura 3370 transações corretamente classificadas como fraudulentas, de um universo de 4110 transações possíveis para a categoria. Se considerarmos que este fosse o resultado de uma implementação em ambiente real, o modelo conseguiria impedir, aproximadamente, 8 em cada 10 tentativas de fraude.

Se comparado aos resultados obtidos nas simulações com os dados de treino, observa-se que o modelo final acabou não performando tão bem quanto nas etapas anteriores. Analisando a documentação do XGB, novamente, pode-se inferir que o treinamento tenha incorrido em um possível caso de *overfitting* no modelo.

A explicação se dá por conta da configuração do parâmetro “*max\_depth*” muito elevada. Quanto maior este parâmetro, mais profunda será a árvore e mais especificidades serão aprendidas no treinamento que serão incapazes de serem reproduzidas durante os testes.

Após um ajuste neste parâmetro, os números já melhoraram e se aproximaram do resultado obtido durante o treinamento, conforme pode-se observar na Figura 40.

Figura 40 – Melhora após calibração do parâmetro de *max\_depth*

```
"C:\Users\Maximiliano Meyer\AppData\Local\Programs\Python\Python39\python.exe"  
Mathews: 0.2748045190524931  
Recall: 0.8748419721871049  
  
Process finished with exit code 0
```

Fonte: Autores (2021)

Quanto ao tempo necessário para classificar as transações, como pode ser observado na Figura 41, a aplicação precisou de pouco mais de 22 segundos para analisar mais de 40 mil registros. Se transposta essa medição para uma execução única, imaginando uma transação em tempo real, seria necessário algo como 0.00055 segundo para a decisão do classificador.

Figura 41 - Tempo necessário para classificar pouco mais de 40 mil transações durante o teste

```
Carregando dados.  
Isto pode demorar de acordo com o tamanho do arquivo  
  
Arquivo classificar.csv carregado com sucesso  
  
Processo de classificação iniciado  
  
Arquivo resultado_classificacao.csv exportado para a pasta Dataset  
Tempo total para a classificação: 22.086 segundos  
  
Process finished with exit code 0
```

Fonte: Autores (2021)

Quanto aos resultados serem satisfatórios ou não em um ambiente real, não temos como afirmar em comparação, pois, infelizmente, as instituições financeiras não abrem seus dados

sobre a porcentagem de fraudes consumadas frente às tentativas de fraudes que são bloqueadas. Contudo, identificar 87% das fraudes parece ser um bom ponto de partida para um modelo de ML que tem este objetivo.

*Tabela 3 - Valores finais utilizados para treinamento e classificação*

<b>Parâmetro</b>	<b>Valor</b>
Taxa de correlação	0.8
ETA	0.1
n_estimators	500
min_child_weight	5
subsample	0.8
colsample_bytree	0.7
scale_pos_weight	5
max_depth	7
split	0.2
<b>Valor de revocação obtido:</b>	<b>0.8748</b>

Fonte: Autores (2021)

## 7 Conclusão

Há um bom tempo as compras online deixaram de ser somente um complemento às compras físicas e passaram a fazer parte da rotina das pessoas. Por se tratar de uma transação digital, o método de pagamento mais utilizado é igualmente digital, o cartão de crédito. O número de transações deste tipo, aliás, é um bom modo de analisar o fenômeno do comércio eletrônico e o tamanho do mercado movimentado por ano.

Em 2018, foram registradas mais de 650 bilhões de transações de cartão de crédito no mundo, sendo o Brasil o quarto país que mais contribuiu para este número. Com mais de 34 bilhões de operações registradas naquele ano, ficamos atrás apenas de Estados Unidos, China e Rússia. Quanto aos valores movimentados, somente em 2019 foram mais de 3,5 trilhões de dólares americanos transacionados através da internet.

Embora estes números sejam surpreendentes, eles podem ser considerados modestos e a justificativa é simples: estes números remetem a um cenário pré-Covid-19, ou seja, antes do crescimento inédito que o comércio eletrônico passou a experimentar a partir de março de 2020.

Somente no primeiro ano de pandemia os valores transacionados cresceram mais de 20% a nível mundial, batendo quase 4,3 trilhões de dólares. Quanto ao número de operações, ainda não há dados oficiais, mas, se antes da pandemia acreditava-se que a marca de 1 trilhão de operações em um único ano seria ultrapassada somente em 2023, hoje já se pode imaginar 2022, ou até mesmo 2021.

Além da Covid-19 não dá para esquecer que o comércio eletrônico está em alta ininterrupta há décadas. Os motivos para que a modalidade seja a preferida de muitas pessoas se dá por conta da sua praticidade, da variedade de itens e claro, dos preços baixos. Contudo, há de se ter em mente que este cenário aquecido para os comerciantes e compradores também acaba sendo altamente vantajoso para os criminosos.

Considerando que 7 em cada 10 dólares transacionados online sejam através do cartão de crédito, há aproximadamente 3 trilhões de dólares negociados através dos cartões de plástico. Logo, se os fraudadores conseguirem se apropriar da ínfima parte que corresponde a apenas 0,1%, deste montante, o valor desviado ilicitamente será de cerca de 3 bilhões de dólares ao ano.

O combate a esse tipo de crime se dá, também, através da tecnologia, mais precisamente através da ciência de dados – campo que busca extrair informações valiosas dos dados e

transformá-las em solução para os mais variados tipos de problemas – e dos algoritmos de *machine learning* – técnicas capazes de identificar padrões através dos dados e, com isso, “aprender” a fazer previsões e classificações, por exemplo, em dados inéditos. As técnicas de ML são os principais aliados contra os fraudadores de transações online.

Visando a contribuir com este cenário de combate ao crime virtual, esta pesquisa se propôs a desenvolver uma ferramenta que pudesse ser treinada para aprender como agem os fraudadores e, com isso, ser utilizada em um ambiente real, no qual possa identificar e bloquear as operações com grandes chances de serem uma tentativa de fraude.

Para isso a primeira – e mais complicada – etapa foi a obtenção de um conjunto de dados que fosse público, composto de transações reais e que não tivesse sido utilizado em pesquisas à exaustão. A justificativa para este ponto, à primeira vista restritivo e ineficiente, é que, de todos os *datasets* que foram utilizados nos artigos reunidos na análise bibliográfica desta pesquisa, todos, sem exceção, utilizavam a mesma base. Portanto, para que a presente pesquisa tivesse a chance de contribuir com algo relevante ao campo de estudo, uma base de dados diferenciada era fundamental.

Após encontrar o *dataset* ideal, começou-se a etapa de testes utilizando os algoritmos que tiveram melhor desempenho nas mesmas pesquisas anteriormente citadas. Assim, as primeiras tentativas começaram com o algoritmo Random Forest, unanimidade entre os trabalhos.

No entanto, quando aplicado o algoritmo Random Forest à base de dados utilizada, os resultados ficaram aquém do esperado. Uma possível explicação reside no fato de que esta base utilizada por outros pesquisadores é demasiadamente simples: são apenas 23 colunas e 200 mil registros. Assim, quando o algoritmo foi exposto a um *dataset* mais complexo – 394 colunas e pouco menos de 600 mil registros – os números de seu desempenho não impressionaram.

Após uma intensa pesquisa sobre qual opção poderia ser capaz de entregar melhores resultados, foi encontrado o Extreme Gradient Boosting, um algoritmo que combina diferentes algoritmos de ML menos poderosos para, no final do processo ao somar todas as etapas, obter uma predição mais completa e acertada. Na prática este algoritmo cria  $n$  árvores de decisão sequenciais que levam o valor predito em  $n$  para  $n+1$ .

Após obter-se uma pequena, mas não suficiente, melhora quando da utilização direta deste algoritmo, recorreremos a uma de suas principais características: o ajuste dos mais de 80 valores e parâmetros de configuração possíveis. Por se tratar de um processo lento e maçante,

utilizou-se uma ferramenta capaz de, automaticamente, cruzar todas as opções entre si e retornar os melhores resultados dentre todos.

Com o algoritmo XGB configurado com os parâmetros ideais para este *dataset* notou-se um incremento significativo nos resultados, porém, era preciso um pouco mais para chegar a um nível de eficiência o qual justificar-se-ia utilizar esta ferramenta em cenários reais. Por isso, na sequência foi implementada uma técnica capaz de fazer o *resampling* dos dados e, assim, diminuir o impacto das classes desbalanceadas na pontuação final da classificação.

Após ter um *pipeline* que resulta em taxas satisfatórias de classificação de transações fraudulentas, pode-se investir no último objetivo específico desta pesquisa: o desenvolvimento de uma aplicação que pudesse ser distribuída e utilizada com qualquer *dataset* real.

Desenvolvida em Python, a solução conta com ferramentas que vão desde o pré-processamento dos dados crus até a classificação de registros não rotulados, passando por treinamento do modelo e otimização automática dos parâmetros e valores.

Com isso conclui-se todos os objetivos propostos no início da pesquisa.

Quanto aos resultados, se considerar que trata-se de uma pesquisa inicial, pode-se considerá-los satisfatórios com os valores de revocação atingirem uma marca superior aos 87% de acerto. Em trabalhos futuros, é claro, estes resultados podem ser incrementados.

Outros pontos que podem ser investigados em pesquisas subsequentes está a possibilidade de transformar o script em um software instalável e fechado, sem acesso ao código fonte por motivos de segurança. Outro ponto a ser investigado é a possibilidade de reduzir os tempos de execução das etapas, em especial, aquela que envolve o Grid Search CV para encontrar as melhores configurações para o treinamento. Imagina-se para versões futuras, por exemplo, uma construção de software capaz de distribuir o custo computacional em diferentes máquinas, tirando o máximo de proveito das técnicas de computação distribuída.

Outra questão importante para que a ferramenta desenvolvida possa ser utilizada por mais pessoas é a portabilidade da solução para sistemas operacionais que não Windows, como Mac OS e aqueles baseados na família Linux.

Por fim, é interessante que o trabalho desenvolvido aqui seja colocado à prova com o maior número de bases de dados diferentes possível para, além de ir mitigando eventuais problemas, possa-se avançar em taxas de predição mais acertadas. Um ponto de partida promissor é a própria base de dados da IEEE, que conta com significativo material voltado à área.

## REFERÊNCIAS

- ABCOMM. **Crescimento do e-commerce no Brasil**. Disponível em: < <https://abcomm.org/noticias/crescimento-do-e-commerce-no-brasil/> >. Acesso em: 13 out. 2020
- ALPAYDIN, Ethem. **Introduction to machine learning**. MIT press, 2020.
- AVATO. **XGBOOST: Differences between gbtrees and gblines**. Disponível em: < <https://www.avato-consulting.com/?p=28903&lang=en> >. Acesso em: 08 maio 2021
- AWOYEMI, John O.; ADETUNMBI, Adebayo O.; OLUWADARE, Samuel A. **Credit card fraud detection using machine learning techniques: A comparative analysis**. In: 2017 International Conference on Computing Networking and Informatics (ICCNI). IEEE, 2017. p. 1-9
- BAGGA, Siddhant *et al.*, **Credit Card Fraud Detection using Pipelining and Ensemble Learning**. Procedia Computer Science, v. 173, p. 104-112, 2020.
- BANCO CENTRAL DO BRASIL. **Cartões de Crédito Emitidos**. Disponível em: < <https://dadosabertos.bcb.gov.br/dataset/25147-cartoes-de-credito-emitidos> >. Acesso em 12 out. 2020
- BECKER, Lauro. **Algoritmo de Classificação Naive Bayes**. Disponível em: < <https://www.organicadigital.com/blog/algoritmo-de-classificacao-naive-bayes/> >. Acesso em: 02 maio 2021
- CAMPOS, Raphael. **Árvores de Decisão**. Disponível em: < <https://medium.com/machine-learning-beyond-deep-learning/%C3%A1rvores-de-decis%C3%A3o-3f52f6420b69> >. Acesso em: 09 maio 2021
- CARCILLO, Fabrizio *et al.*, **Combining unsupervised and supervised learning in credit card fraud detection**. In: Information Sciences. 2019
- CASTELLS, Manuel. **A Galáxia da Internet – Reflexões sobre a internet, os negócios e a sociedade**. Rio de Janeiro: Jorge Zahar, 2003.
- CHEN, Bindi. **Working with missing values in Pandas**. Disponível em: < <https://towardsdatascience.com/working-with-missing-values-in-pandas-5da45d16e74> >. Acesso em: 10 jun. 2021
- CHEN, Shenglei *et al.*, **A novel selective naïve Bayes algorithm**. In: Knowledge-Based Systems, v. 192, 2020.
- CHICCO, Davide; JURMAN, Giuseppe. **The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation**.

Disponível em: <https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7>  
>. Acesso em: 24 abr. 2021

COLAB RESEARCH. **Help and Documentation in Python**. Disponível em: <  
<https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/01.01-Help-And-Documentation.ipynb> >. Acesso em: 18 dez. 2020

CONWAY, Drew. **The Data Science Venn Diagram**. Disponível em: <  
<http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram> >. Acesso em: 14 out. 2020

CORONATO, Antonio *et al.*, **Reinforcement Learning for Intelligent Healthcare Applications: A Survey**. In: Artificial Intelligence in Medicine. 2020

CLEARSALE. **Card-not-present Fraud Is Skyrocketing**. Disponível em: <  
<https://www.clear.sale/Infographics/cnp-fraud-is-skyrocketing> >. Acesso em: 14 out. 2020

DHANKHAD, Sahil; MOHAMMED, Emad; FAR, Behrouz. **Supervised machine learning algorithms for credit card fraudulent transaction detection: a comparative study**. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). IEEE, 2018. p. 122-125.

DORNADULA, Vaishnavi Nath; GEETHA, S. **Credit Card Fraud Detection using Machine Learning Algorithms**. Procedia Computer Science, v. 165, p. 631-641, 2019.

EL-SAYED, Asmaa Ahmed et al. **Handling autism imbalanced data using synthetic minority over-sampling technique (SMOTE)**. In: 2015 Third World Conference on Complex Systems (WCCS). IEEE, 2015. p. 1-5.

ESTATIDADOS. **Parte 4 - Árvore de Decisão, Random Forest e Gradient Boosting - Agora ou nunca!** - Prof. Dri. Youtube, 5 jan. de 2019 Disponível em: <  
<https://www.youtube.com/watch?v=noy13V1nTz4> >. Acesso em: 13 maio 2021

EXAME. **Mercado Livre agora tem frota própria de aviões no Brasil** Disponível em: <  
<https://exame.com/negocios/mercado-livre-agora-tem-frota-propria-de-avioes-no-brasil/> >. Acesso em: 07 nov. 2020

GATEFY. **O que é regressão logística e como a utilizamos para classificar e-mails**. Disponível em: < <https://gatefy.com/pt-br/blog/o-que-e-regressao-logistica-e-como-a-utilizamos-para-classificar-e-mails/> >. Acesso em: 02 maio 2021

GERALDO, Graciela Cristina; MAINARDES, Emerson Wagner. **Estudo sobre os fatores que afetam a intenção de compras online**. In Rege – Revista de Gestão, Volume 24, Issue 2, April – June 2017. Pág. 181 – 194 Disponível em  
<https://www.sciencedirect.com/science/article/pii/S1809227617300620?via%3Dihub#bibl0005> >. Acesso em: 21 maio 2020

GIBERT, Karina *et al.*, **Environmental data science**. in Environmental Modelling & Software 106 (2018): 4-12.

GONZALEZ, Leandro de Azevedo. **Regressão Logística e suas Aplicações**. Disponível em: < <https://monografias.ufma.br/jspui/bitstream/123456789/3572/1/LEANDRO-GONZALEZ.pdf> >. Acesso em: 09 maio 2021

HARVARD BUSINESS REVIEW. **Data Scientist: The Sexiest Job of the 21st Century** Disponível em: < <https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century> >. Acesso em: 14 out. 2020

IEEE COMPUTATIONAL INTELLIGENCE SOCIETY. **Scope**. Disponível em: < <https://web.archive.org/web/20160604143046/http://cis.ieee.org/scope.html> >. Acesso em: 13 maio 2021

IMBALANCED-LEARN. **Imbalanced-Learn Documentation**. Disponível em: < <https://imbalanced-learn.org/stable/index.html> >. Acesso em: 13 maio 2021

IPEA DATA. **Produto Interno Bruto (PIB) Real**. Disponível em: < <http://www.ipeadata.gov.br/exibeserie.aspx?serid=38414> >. Acesso em: 16 maio 2021

JAIN, Aarshay. **Complete Machine Learning Guide to Parameter Tuning in Gradient Boosting (GBM) in Python**. Disponível em: < <https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/> >. Acesso em: 13 maio 2021

JETBRAINS. **Pycharm Documentation**. Disponível em: < <https://www.jetbrains.com/help/pycharm/quick-start-guide.html> >. Acesso em: 26 jun. 2021

JOBLIB. **Joblib Documentation**. Disponível em: < <https://joblib.readthedocs.io/en/latest/index.html> >. Acesso em: 26 jun. 2021

JORNAL DO COMÉRCIO. **E-commerce consolida-se para além da pandemia**. Disponível em: < [https://www.jornalcomercio.com/\\_conteudo/editorial/2021/02/780299-e-commerce-consolida-se-para-alem-da-pandemia.html](https://www.jornalcomercio.com/_conteudo/editorial/2021/02/780299-e-commerce-consolida-se-para-alem-da-pandemia.html) >. Acesso em: 12 abr. 2021

JOSÉ, Italo. **KNN (K-Nearest Neighbors) #1**. Disponível em: < <https://medium.com/brasil-ai/knn-k-nearest-neighbors-1-e140c82e9c4e> >. Acesso em: 02 maio 2021

KAGGLE. **IEEE-CIS Fraud Detection**. Disponível em: < <https://www.kaggle.com/c/ieee-fraud-detection/overview> >. Acesso em: 21 abr. 2021

KORNEGAY, Adam T.; SCHWAB, Matthew R.; DE ALMEIDA, Marcos C. **Credit score simulation**. U.S. Patent n. 7,593,891, 22 set. 2009.

KUMAR, Satyam. **7 Over Sampling techniques to handle Imbalanced Data**. Disponível em: < <https://towardsdatascience.com/7-over-sampling-techniques-to-handle-imbalanced-data-ec51c8db349f> >. Acesso em: 13 maio 2021

LEE, Kai-Fu. **AI Superpowers: China, Silicon Valley, and the New World Order**. Boston, Houghton Mifflin Harcourt, 2018

MATPLOTLIB. **Matplotlib Documentation**. Disponível em: < <https://matplotlib.org/stable/gallery/index.html> >. Acesso em 21 jun. 2021

MITCHELL, Tom M. **Machine Learning**. New York, McGraw Hill, 1997

MÜLLER, Andreas C.; GUIDO, Sarah. **Introduction to machine learning with Python: a guide for data scientists**. O'Reilly Media, Inc., 2016.

NICHOLSON, Chris. **Evaluation Metrics for Machine Learning - Accuracy, Precision, Recall, and F1 Defined**. Disponível em: < <https://wiki.pathmind.com/accuracy-precision-recall-f1> >. Acesso em: 24 abr. 2021

PAGBRASIL. **Central Bank of Brazil announces open banking regulations**. Disponível em: < <https://www.pagbrasil.com/news/central-bank-of-brazil-announces-open-banking-regulations/> >. Acesso em: 17 nov. 2020

PAGSEGURO. **2020, um ano histórico para a venda online**. Confira dados e tendências para 2021 Disponível em: < <https://blog.pagseguro.uol.com.br/2020-um-ano-historico-para-a-venda-online/> >. Acesso em: 12 abr. 2021

PANDAS. **Pandas Documentation**. Disponível em: < <https://pandas.pydata.org/docs/> >. Acesso em 18 dez. 2020

POURHABIBI, Tahereh; ONG, Kok-Leong; KAM, Booi H; BOO, Yee Ling. **Fraud detection: A systematic literature review of graph-based anomaly detection approaches**. Disponível em: < <https://www.sciencedirect.com/science/article/pii/S0167923620300580> >. Acessado em: 12 maio 2020

PYTHON. Disponível em: < <https://www.python.org/> >. Acesso em: 18 dez. 2020

QAZ. **Precisão e Recall**. Disponível em: < [https://pt.qaz.wiki/wiki/Precision\\_and\\_recall](https://pt.qaz.wiki/wiki/Precision_and_recall) >. Acesso em 23 mar. 2021

RAUTENBERG, Sandro; CARMO, Paulo Ricardo Viviurka do. **Big Data e Ciência de Dados: complementariedade conceitual no processo de tomada de decisão**. *Brazilian Journal of Information Science* 13.1 (2019): 56-67. Disponível em: < <https://dialnet.unirioja.es/servlet/articulo?codigo=6983493> >. Acessado em: 17 jun. 2020

SAHIN, Y.; DUMAN, E. **Detecting Credit Card Fraud by Decision Trees and Support Vector Machines**. In: Proceedings of the International Multiconference of Engineers and Computer Scientists. vol. I, p. 442–447, 2011

SAMUEL, Arthur Lee. **Some Studies in Machine Learning Using the Game of Checkers.** Disponível em: < <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560> >. Acesso em: 14 out. 2020

SCIKIT-LEARN. **Scikit Documentation.** Disponível em: < <https://scikit-learn.org/stable/> >. Acesso em: 18 dez. 2020

SHOPIFY. **Global Ecommerce Statistics and Trends to Launch Your Business Beyond Borders.** Disponível em: < <https://www.shopify.com/enterprise/global-ecommerce-statistics> >. Acesso em: 13 out. 2020

SHUKUR, Hamzah Ali; SEFER Kurnaz. **Credit Card Fraud Detection using Machine Learning Methodology.** (2019). Disponível em: < <http://www.academia.edu/download/58683754/V8I3201941.pdf> >. Acessado em: 17 jun. 2020

SHUNG, Koo Ping. **Accuracy, Precision, Recall or F1?** Disponível em: < <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9> >. Acesso em 25 abr. 2021

SPC BRASIL. **40% dos consumidores utilizaram cartão de crédito de alguma fintech nos últimos 12 meses, aponta pesquisa CNDL/SPC Brasil.** Disponível em: < <https://www.spcbrasil.org.br/pesquisas/pesquisa/7212> >. Acesso em: 17 nov. 2020a

\_\_\_\_\_. **52 Milhões de brasileiros usam o cartão de crédito como forma de pagamento, diz SPC Brasil.** Disponível em: < [https://www.spcbrasil.org.br/uploads/st\\_imprensa/release\\_cartao\\_de\\_credito.pdf](https://www.spcbrasil.org.br/uploads/st_imprensa/release_cartao_de_credito.pdf) >. Acesso em: 12 out. 2020b

\_\_\_\_\_. **Em 12 meses, quase oito milhões de brasileiros foram vítimas de fraudes, estimam CNDL/SPC Brasil.** Disponível em: < <https://www.spcbrasil.org.br/pesquisas/pesquisa/5546> >. Acesso em: 14 out. 2020c

SPEND JOURNAL. **Credit card statistics 2020.** Disponível em: < <https://blog.spendesk.com/en/credit-card-statistics-2020> >. Acesso em 12 out. 2020

THE NILSON REPORT; **Card Fraud Losses Reach \$27.85 Billion.** Disponível em: < <https://nilsonreport.com/mention/407/1link/> >. Acesso em: 13 out. 2020

THE WORLD BANK. **credit card ownership (% age 15+).** Disponível em: < <https://todata360.worldbank.org/indicators/h83ea0f24?indicator=3360&viz=choropleth&years=2017&compareBy=region> >. Acesso em 12 out. 2020

TREUTLER, Amanda. **The Sliding Window Strategy for Solving Algorithms.** Disponível em: < <https://levelup.gitconnected.com/the-sliding-window-strategy-for-solving-algorithms-34c95c80c506> >. Acesso em: 08 fev. 2021

UYEKITA, Anderson. **Como usar o GridSearchCV**. Disponível em: < <https://andersonuyekita.github.io/notebooks/blog/2019/03/21/como-usar-o-gridsearchcv/> >. Acesso em: 08 jun. 2021

VESTA. **Vesta Company**. Disponível em: < <https://trustvesta.com/> >. Acesso em: 21 mar. 2021

VIANA, Humberto. **Pandemia consolida crescimento de vendas na internet**. Disponível em: < <https://cndl.org.br/varejosa/pandemia-consolida-crescimento-de-vendas-na-internet/> >. Acesso em: 04 mar. 2021

World Payments Report. **WORLD PAYMENTS REPORT 2019**. Disponível em: < <https://worldpaymentsreport.com/wp-content/uploads/sites/5/2019/09/World-Payments-Report-WPR-2019.pdf> >. Acesso em: 2/5/2020

\_\_\_\_\_. **WORLD PAYMENTS REPORT 2020**. Disponível em: < <https://worldpaymentsreport.com/resources/world-payments-report-2020/> >. Acesso em 13 out. 2020

XGBOOST. **XGBoost Documentation**. Disponível em: < <https://xgboost.readthedocs.io/en/latest/index.html> >. Acesso em: 04 mai. 2021

XUAN, Shiyang *et al.*, **Random forest for credit card fraud detection**. In: 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC). IEEE, 2018

ZAREAPOOR, Masoumeh *et al.*, **Application of credit card fraud detection: Based on bagging ensemble classifier**. In: Procedia computer science, v. 48, n. 2015, p. 679-685, 2015.

## APÊNDICE A – Árvore de decisão criada pelo algoritmo extreme gradient boosting

