UNIVERSIDADE FEDERAL DE OURO PRETO

CHURNAS: UMA BUSCA DE ARQUITETURA NEURAL PARA PREVISÃO DE DESLIGAMENTO DE CLIENTES

Marcus Daniel de Almeida

Orientadores: Gladston Juliano Prates Moreira
Eduardo José da Silva Luz

Ouro Preto
Julho de 2023

UNIVERSIDADE FEDERAL DE OURO PRETO

UFOP

decom
departamento de computação

CHURNAS: UMA BUSCA DE ARQUITETURA NEURAL PARA PREVISÃO DE DESLIGAMENTO DE CLIENTES

Marcus Daniel de Almeida

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, da Universidade Federal de Ouro Preto, como parte dos requisitos necessários à obtenção do título de Mestre em Ciência da Computação.

Orientadores: Gladston Juliano Prates Moreira
Eduardo José da Silva Luz

Ouro Preto
Julho de 2023

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

A447c Almeida, Marcus Daniel de.
CHURNAS [manuscrito]: uma Busca de Arquitetura Neural para previsão de desligamento de clientes. / Marcus Daniel de Almeida. - 2023.
58 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Gladston Juliano Prates Moreira.
Coorientador: Prof. Dr. Eduardo José da Silva Luz.
Dissertação (Mestrado Acadêmico). Universidade Federal de Ouro Preto. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Redes neurais (Computação). 2. Serviços financeiros. 3. Algoritmos Genéticos. I. Moreira, Gladston Juliano Prates. II. Luz, Eduardo José da Silva. III. Universidade Federal de Ouro Preto. IV. Título.

CDU 004

Bibliotecário(a) Responsável: Luciana De Oliveira - SIAPE: 1.937.800

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO
PROGRAMA DE POS-GRADUACAO EM CIENCIA DA COMPUTACAO

FOLHA DE APROVAÇÃO

Marcus Daniel de Almeida

ChurNAS: uma busca de arquitetura neural para previsão de desligamento de clientes

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 10 de abril de 2023.

Membros da banca

Prof. Dr. Gladston Juliano Prates Moreira - Orientador - Universidade Federal de Ouro Preto
Prof. Dr. Eduardo José da Silva Luz - Co-Orientador - Universidade Federal de Ouro Preto
Prof. Dr. Ivair Ramos Silva - Universidade Federal de Ouro Preto
Prof. Dr. Ivan Reinaldo Meneghini - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais

Prof. Dr. Gladston Juliano Prates Moreira, orientador do trabalho, aprovou a versão final e autorizou seu depósito no Repositório Institucional da UFOP em 03/07/2023

Documento assinado eletronicamente por **Gladston Juliano Prates Moreira**, **COORDENADOR(A) DE CURSO DE PÓS-GRADUACÃO EM CIÊNCIA DA COMPUTAÇÃO**, em 06/07/2023, às 17:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0551504** e o código CRC **5BBA9F28**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3559-1641 - www.ufop.br

"Nós só podemos ver um pouco do
futuro, mas o suficiente para
perceber que há muito a fazer."
Alan Turing

Resumo da Dissertação apresentada à UFOP como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CHURNAS: UMA BUSCA DE ARQUITETURA NEURAL PARA PREVISÃO DE DESLIGAMENTO DE CLIENTES

Marcus Daniel de Almeida

Julho/2023

Orientadores: Gladston Juliano Prates Moreira
Eduardo José da Silva Luz

Programa: Ciência da Computação

A Predição de desligamento de clientes PDC (do inglês *Customer Churn Prediction*) é fundamental para a gestão eficiente de clientes, uma vez que permite a otimização da lucratividade por meio de estratégias de marketing informadas e campanhas de retenção. Nesse contexto, o presente estudo propõe uma nova abordagem, chamada ChurNAS, baseada em algoritmo genético para a busca de arquiteturas neurais (Neural Architecture Search - NAS) em problemas de PDC na indústria de serviços financeiros. Ao contrário dos modelos tradicionais, como regressão logística e árvores de decisão, as redes neurais profundas apresentam maior versatilidade para modelagem de dados complexos. No entanto, a busca pela arquitetura ideal em redes neurais profundas é um desafio devido à sua alta complexidade. Os resultados demonstram que a abordagem ChurNAS encontrou modelos com desempenho superior aos classificadores tradicionais ajustados por otimização de hiperparâmetros. A abordagem proposta obteve uma acurácia de 88,6%, em comparação com 82,54% do XG-Boost e 82,49% do Floresta Aleatória. Além disso, alcançou uma sensibilidade de 58,89%, enquanto o XG-Boost e o Floresta Aleatória apresentaram 57,1% e 57,81%, respectivamente. Quanto à precisão, a abordagem ChurNAS obteve 39,41%, superando o XG-Boost (26,96%) e o Floresta Aleatória (26,17%). Adicionalmente, o estudo examinou o impacto da quantidade de dados e da capacidade do modelo, enfatizando a importância de considerar a natureza temporal das transações financeiras ao utilizar redes neurais para PDC. Em suma, este trabalho destaca o potencial da abordagem ChurNAS para solucionar problemas de PDC no setor de serviços financeiros e melhorar a eficiência do gerenciamento de clientes.

Abstract of Dissertation presented to UFOP as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CHURNAS: A NEURAL ARCHITECTURE SEARCH FOR CUSTOMER CHURN PREDICTION

Marcus Daniel de Almeida

July/2023

Advisors: Gladston Juliano Prates Moreira
Eduardo José da Silva Luz

Department: Computer Science

Churn Prediction (CCP) iis essential for efficient customer management, as it allows for profitability optimization through informed marketing strategies and retention campaigns. In this context, this study proposes a new approach, called ChurNAS, based on a genetic algorithm for neural architecture search in CCP problems in the financial services industry. Unlike traditional models, such as logistic regression and decision trees, deep neural networks offer greater versatility for modeling complex data. However, searching for the ideal architecture in deep neural networks is a challenge due to their high complexity. The results demonstrate that the ChurNAS approach found models with better performance than traditional classifiers adjusted by hyperparameter optimization. The proposed approach achieved an accuracy of 88.6%, compared to 82.54% for XG-Boost and 82.49% for Random Forest. In addition, it reached a sensitivity of 58.89%, while XG-Boost and Random Forest presented 57.1% and 57.81%, respectively. As for accuracy, the ChurNAS approach obtained 39.41%, surpassing XG-Boost (26.96%) and Random Forest (26.17%). Additionally, the study examined the impact of data quantity and model capacity, emphasizing the importance of considering the temporal nature of financial transactions when using neural networks for CCP. Overall, this work highlights the potential of the ChurNAS approach to solve CCP problems in the financial services industry and improve customer management efficiency.

Agradecimentos

Gostaria de expressar minha imensa gratidão à minha família, em especial à minha mãe, Terezinha, pelo amor e dedicação investidos em minha educação.

Ao meu pai, Daniel (*in memoriam*), por ser sempre uma inspiração para mim. Também ao meu padrasto, Carlo, pelo apoio ao longo desta jornada.

Agradeço especialmente à minha amiga e colega de curso e projeto, Andressa. Sua presença e apoio foram de valor inestimável, tornando essa jornada mais leve e prazerosa.

Gostaria de estender minha gratidão às minhas colegas de projeto, Mariana e Luísa, pela valiosa troca de conhecimentos durante o desenvolvimento desta pesquisa.

Expresso minha gratidão à Efí S.A pela oportunidade de realizar esta pesquisa e pelo financiamento deste trabalho. Além disso, destaco o suporte fundamental para o sucesso deste projeto dos colaboradores Wellington Ferreira, Marcos Alvarenga e Marcos Nicolau.

Gostaria de expressar meu mais profundo agradecimento ao meu orientador, Gladston, pela oportunidade, confiança, tempo dedicado, sábios conselhos e orientação recebida. Também estendo meu agradecimento ao meu coorientador, Eduardo, pela inestimável contribuição ao trabalho e pela paciência e dedicação com que me auxiliou em todo o processo de aprendizado.

Por fim, desejo expressar minha gratidão à Universidade Federal de Ouro Preto (UFOP) e a todos os indivíduos que, direta ou indiretamente, contribuíram para o êxito desta pesquisa e para a minha formação acadêmica.

Lista de Figuras

2.1 Representação da predição de nova instância no floresta aleatória. Fonte: O Autor. 8
2.2 Representação da predição de nova instância no XG-boost. Fonte: O Autor. 9
2.3 Nas figuras são representadas as convoluções causais e não causais. Fonte: O Autor. 12
2.4 Representação de uma convolução em uma entrada com mais de uma sequência. Fonte: O Autor. 12
2.5 Representação das convoluções dilatadas. As dilatações são representadas pelo espaçamento entre os elementos visualizados pelos filtros. Os filtros, por sua vez, são indicados pelos segmentos que unem os elementos em cada linha. Na figura, o filtro tem tamanho igual a três e d indica o fator de dilatação. Fonte: O autor. 18
2.6 Representação do bloco residual. Fonte: O Autor. 19
2.7 Representação de uma célula LSTM. Fonte: O Autor. 20
2.8 Representação de uma célula LSTM. Fonte: O Autor. 20
2.9 Representação da nova rede de memória e portal da camada de entrada. Fonte: O Autor. 21
2.10 Representação do portal de saída. Fonte: O Autor. 22
2.11 A figura representa de forma abstrata os métodos utilizados na busca de arquitetura neural. Inicialmente, uma estratégia de busca é utilizada para selecionar uma arquitetura dentro de um espaço de busca predefinido. Em seguida, essa arquitetura é avaliada pela estratégia de estimativa de desempenho, que retorna uma estimativa do desempenho para a estratégia de busca. Esse processo é repetido recursivamente até que uma saída seja gerada, ou seja, o processo finalizado. Fonte: O Autor. 23
2.12 Ilustração de um algoritmo genético convencional. Fonte: O Autor. . 25

4.1 Separação dos históricos de registros: as 26 semanas mais antigas geraram os atributos e as 26 semanas (mais recentes) reservadas para a rotulagem dos dados. Fonte: O Autor. 29
4.2 Representação de um deslizamento. Fonte: O Autor. 29
4.3 Na Figura (a) é apresentada a versão tabular do conjunto de dados e na Figura (b) a versão sequencial. Fonte: O Autor. 30

5.1 Fluxo metodológico utilizado neste trabalho. Fonte: O Autor. 32
5.2 Fluxograma do espaço de busca. Fonte: O Autor. 35
5.3 Representação da visão macro de uma arquitetura codificada. Na figura, atenção m1 é o mecanismo *channel attention* a ser aplicado antes do bloco TCNN ou CNN e atenção m2 é o mecanismo de atenção a ser aplicado antes do bloco LSTM. Fonte: O Autor. 37
5.4 Representação da bloco DNN com uma e duas células. Fonte: O Autor. 37
5.5 Exemplo decodificação de elementos em um bloco DNN. Fonte: O Autor. 38
5.6 Exemplo de indivíduo codificado. Na figura, para uma melhor visualização, os hiperparâmetros dos blocos não ativados pelo indivíduo (CNN e LSTM) foram omitidos. Fonte: O Autor. 39
5.7 Exemplo arquitetura correspondente ao indivíduo apresentado na Figura 5.6. Fonte: O Autor. 40
5.8 Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 10% dos dados. Fonte: O Autor. 43
5.9 Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 20% dos dados. Fonte: O Autor. 44
5.10 Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 30% dos dados. Fonte: O Autor. 44
5.11 Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com *dataset* de busca com 40% dos dados. Fonte: O Autor. 45

6.1 Nos gráficos são apresentados as curvas para os 5 *k-folds* e em destaque a curva média para os três modelos. No gráfico (b) o valor entre parenteses representa a área sobre a curva ROC média do modelo. Fonte: O Autor. 48
6.2 Curva de F-Score por modelo encontrados em diferentes particionamentos do conjunto de dados. Fonte: O Autor. 49

6.3 Gráfico de relação entre *F-Score* e parâmetros treináveis. Fonte: O Autor. 50

Sumário

Lista de Figuras

1 Introdução **1**
- 1.1 Questão de Pesquisa . . . 3
- 1.2 Objetivo Geral e Objetivos Específicos . . . 3
- 1.3 Publicação . . . 3
- 1.4 Organização do Trabalho . . . 4

2 Referencial Teórico **5**
- 2.1 A predição de desligamento de clientes . . . 5
- 2.2 Modelagem Temporal . . . 6
- 2.3 Classificadores . . . 6
  - 2.3.1 Floresta Aleatória . . . 7
  - 2.3.2 *Extreme Gradient Boosting* . . . 9
  - 2.3.3 Redes Neurais Convolucionais Unidimensionais . . . 10
  - 2.3.4 Redes Convolucionais Temporais . . . 17
  - 2.3.5 Redes de Memória de Longo Prazo . . . 19
- 2.4 Busca de Arquitetura Neural . . . 22
- 2.5 Algoritmo Genético . . . 23

3 Trabalhos Relacionados **26**

4 Definição do Conjunto de Dados **28**
- 4.1 Extração de atributos . . . 29
- 4.2 Rotulagem dos Dados . . . 30

5 Metodologia **32**
- 5.1 Abordagem Proposta (ChurNAS) . . . 32
  - 5.1.1 Configuração do AG . . . 32
  - 5.1.2 Espaço de busca adotado . . . 34
  - 5.1.3 Codificação das arquiteturas . . . 35
  - 5.1.4 Decodificação das arquiteturas . . . 37

5.2 Metodologia Experimental 38
5.2.1 Implementação dos Modelos 38
5.2.2 Pré-processamento dos dados 39
5.2.3 Estratégia para contornar o desbalanceamento dos conjuntos de dados 39
5.2.4 Particionamento do conjunto de dados para NAS e HPO . . . 40
5.2.5 Métricas de desempenho 40
5.2.6 Busca exaustiva de hiperparâmetros 41
5.2.7 ChurNAS busca 41
5.2.8 Treinamento dos Modelos 43

6 Resultados, Discussões **46**
6.1 Resultados 46
6.2 Discussões 48

7 Considerações Finais **51**
7.1 Conclusões 51
7.2 Trabalhos Futuros 52

Referências Bibliográficas **53**

Capítulo 1

Introdução

Em um cenário atual de alta competitividade no mercado financeiro, o gerenciamento de clientes nas empresas torna-se cada vez mais desafiador. Nesse contexto, conquistar novos clientes pode ter alto custo devido o investimento necessário em campanhas de marketing. Tendo isso, a retenção de clientes já conquistados é importante para manter a lucratividade das empresas. O processo de antecipar e mitigar a evasão de clientes pode ser realizado através da previsão de desligamento de clientes (PDC) que visa detectar os clientes propensos a mudar o provedor de serviço[16, 22]. No entanto, a PDC no serviço financeiro é um processo complexo, uma vez que, os clientes utilizam os serviços de maneiras diferentes [3]. O desligamento de clientes, traduzido aqui do termo em inglês *Customer Churn*, é definido como a perda ou saída de clientes da base de clientes de uma empresa.

Na PDC o conjunto de dados deve expressar o padrão de uso dos serviços pelos clientes, sendo assim, o processo de engenharia de atributos é fundamental. A extração de atributos, geralmente, contempla informações sobre a conta do cliente e estatísticas sobre a utilização dos serviços. As informações comportamentais podem ser atributos de recência, frequência e atributos monetários (RFM)[37]. Tais atributos, geralmente, são estáticos. Todavia, transações financeiras podem ser sequenciadas no tempo, possibilitando a extração de atributos variável no tempo. Tal abordagem resulta em dados temporalmente sensíveis em que a ordem dos dados importa. Adicionalmente, quando disponível, dados não estruturados como, por exemplo, dados textuais, podem ser incorporados para melhorar os resultados [9].

Dentre os métodos preditivos mais utilizados estão a regressão logística [41], árvores de decisão [33], máquinas de vetor de suporte [12] e métodos *ensemble* [14]. Recentemente, modelos de *Deep Learning* têm se mostrado alternativas competitivas e promissoras [1], especialmente em dados temporalmente sensíveis, como a rede recorrente LSTM (*Long-Short Term Memory*) e a rede convolucional TCNN (*Temporal Convolutional Neural Networks*) [1, 2, 37]. Em um trabalho anterior [2], foi proposta uma abordagem temporal para extração de atributos e um modelo baseado em TCNN.

Nos resultados obtidos, o modelo proposto superou os classificadores populares na literatura, como Floresta Aleatória e *Extreme Gradient Boosting* (XG-Boost), em um estudo de caso realizado em um conjunto de dados disponibilizado por um provedor de serviços financeiros.

Um motivo para os métodos consolidados na literatura, como a Floresta Aleatória e o XG-Boost, apresentarem alto desempenho para a PDC pode estar na facilidade de ajustar os hiperparâmetros. Os hiperparâmetros são variáveis que controlam o processo de treinamento de modelos de aprendizado de máquina. Algoritmos como o GridSearchCV, disponível na biblioteca Python *scikit-learn*, simplificam a otimização de hiperparâmetros (*hyperparameter optimization* - HPO) para selecionar o melhor modelo para essas técnicas. No entanto, quando se trata de redes neurais artificiais, especialmente as baseadas em *deep learning*, a otimização de hiperparâmetros (HPO) depende principalmente da arquitetura do modelo. A arquitetura de um modelo refere-se à organização e interconexão das unidades de processamento, como neurônios e camadas, que compõem a rede neural. Encontrar arquiteturas eficientes é um desafio complexo. Nesses casos, uma abordagem alternativa para aprimorar esses modelos é a utilização de métodos de busca de arquitetura neural, conhecidos como *Neural Architecture Search* (NAS) [54].

A NAS tem uma grande sobreposição com o HPO, entretanto, são técnicas substancialmente diferentes. Um problema HPO otimiza uma coleção de hiperparâmetros contínuos e categóricos, enquanto a NAS é especializada na otimização da topologia da arquitetura, a qual é essencialmente mais complexa. Muitas abordagens de NAS foram exploradas como aprendizado por reforço [62], otimização Bayesiana [7], técnicas NAS baseadas em compartilhamento de peso [35] e algoritmos evolucionários (AE) [45]. Neste trabalho, tem-se foco em AE, mais especificamente Algoritmos Genéticos (GA), pois esses tem vantagens como: robustez, otimização conjunta com hiperparâmetros, adaptação facilitada para novos problemas, conjunto de dados e espaços de pesquisa, fácil implementação e utilização, flexibilidade, simplicidade conceitual e resultados competitivos [17, 38].

Nesta pesquisa, é realizada uma investigação da busca de arquitetura neural baseada em algoritmos genéticos para otimizar um modelo de PDC, o que constitui a sua principal contribuição. A abordagem desenvolvida, denominada ChurNAS, contempla um amplo espaço de busca que inclui mecanismos de atenção, operações e camadas, levando em consideração ou não a característica temporal dos atributos. No entanto, o trabalho não se limita a isso. Também é apresentada uma abordagem temporal de extração de atributos em dados reais fornecidos por um provedor de serviços financeiros. Além disso, é realizada uma investigação sobre a influência da quantidade de dados utilizados durante a busca de arquitetura neural na capacidade do modelo e no seu desempenho final. Acredita-se que essas contribuições são

relevantes para o campo da aprendizagem de máquina e têm o potencial de abrir novos caminhos para pesquisas futuras na resolução desse problema.

1.1 Questão de Pesquisa

No contexto de previsão de desligamento de clientes no setor financeiro, considerando um conjunto de dados de informações sobre o padrão de uso dos serviços de clientes, surgem as seguintes questões: (i) os métodos de busca de arquitetura neural podem selecionar arquiteturas mais eficientes para modelos de aprendizagem profunda na previsão de desligamento de clientes no serviço financeiro? e (ii) o tamanho da amostra de dados utilizados na busca de arquitetura neural influencia na arquitetura selecionada e, consequentemente, no desempenho do modelo?

Portanto, o problema de pesquisa consiste na criação de modelos de PDC baseados em redes neurais profundas. Nesse contexto, espera-se investigar o desempenho desses modelos e compará-los com classificadores populares descritos na literatura em um conjunto de dados reais fornecido por um provedor de serviços financeiros.

1.2 Objetivo Geral e Objetivos Específicos

O objetivo deste trabalho é investigar se os algoritmos de busca de arquiteturas neurais são capazes de encontrar modelos de previsão de desligamento de clientes superiores aos modelos baseados em técnicas tradicionais ajustados por otimização de hiperparâmetros. A fim de realizar essa investigação, serão utilizados dados reais fornecidos por um provedor de serviços financeiros.

Os objetivos específicos são: (i) desenhar, desenvolver e fazer a curadoria de um conjunto de dados adequado aos sistemas de aprendizagem de máquina a serem investigados, de modo que, esse expresse os diferentes padrões de uso dos serviços dos clientes; (ii) a investigação de algoritmos genéticos como método de busca para otimizar a arquitetura neural e os hiperparâmetros para o problema de previsão de desligamento de clientes; (iii) a comparação do desempenho dos modelos encontrados por busca de arquitetura neural com os classificadores ajustados por busca exaustiva de hiperparâmetros (XG-Boost e a Floresta Aleatória); (iv) e a investigação da influência da quantidade de dados utilizados na busca de arquitetura neural na convergência ou não em regiões sub-ótimas de arquiteturas.

1.3 Publicação

Este trabalho resultou na publicação do artigo *""A Temporal Approach to Customer Churn Prediction: A Case Study for Financial Services"* [2] no XIX Encontro

Nacional de Inteligência Artificial e Computacional (2022), promovido pela Sociedade Brasileira de Computação.

1.4 Organização do Trabalho

A dissertação está organizada em sete capítulos. O Capítulo 2 apresenta os conhecimentos básicos utilizados neste trabalho, enquanto o Capítulo 3 aborda os trabalhos relacionados. No Capítulo 4, define o conjunto de dados, enquanto o Capítulo 5 apresenta a metodologia desenvolvida e a Metodologia Experimental. O Capítulo 6 traz os resultados obtidos, bem como discussões e possíveis trabalhos futuros. Finalmente, o Capítulo 7 conclui a dissertação.

Capítulo 2

Referencial Teórico

Este capítulo tem como objetivo apresentar os conceitos necessários para compreender o problema a ser resolvido e o funcionamento das técnicas e algoritmos empregados neste trabalho. Ele está organizado da seguinte forma: a seção 2.1 descreve a importância da previsão de desligamento de clientes; a seção 2.2 detalha o processo de modelagem de sequência realizado por modelos especializados em dados temporais; na seção 2.3 são apresentadas as técnicas utilizadas neste trabalho, incluindo floresta aleatória, XG-boost, redes neurais convolucionais unidimensionais, redes neurais convolucionais temporais e redes de memória de longo prazo; a seção 2.4 discute os conceitos de busca de arquitetura neural e, por fim, na seção 2.5 são apresentados os conceitos de algoritmos genéticos.

2.1 A predição de desligamento de clientes

Em um mercado cada vez mais competitivo, as empresas enfrentam altos custos para atrair novos clientes, tornando essencial um trabalho voltado para a retenção daqueles já conquistados. A fidelização dos clientes é vantajosa devido ao menor custo para atendê-los e menor probabilidade de desligamento a longo prazo [16].

A predição de desligamento de clientes (PDC) se concentra em dois tipos de clientes desligados: aqueles totalmente desligados, que interrompem completamente o relacionamento com a empresa, e os parcialmente desligados, que diminuem a frequência de uso dos serviços ou deixam de usar um ou mais serviços, mantendo um relacionamento reduzido com a empresa. O desafio da PDC é identificar ambos os tipos de clientes antes do desligamento total ou parcial [16].

Em [3], são definidos três tipos de desligamento: (i)voluntário, quando os clientes rompem com a empresa e contratam o serviço do concorrente; (ii) não voluntário, quando é a empresa quem rompe o serviço com o cliente; (iii) e desligamentos silenciosos, que se referem a clientes que diminuem a utilização dos serviços e podem vir a se desligar em um futuro próximo. Os tipos (i) e (ii) têm taxas de

desligamento facilmente calculadas, mas o desafio está em prever os clientes do tipo (iii), que precisam ser identificados antes de apresentar um comportamento explícito de desengajamento para que possam ser direcionados para uma campanha de retenção.

A retenção de clientes é particularmente importante para empresas que já atingiram o pico de conquista de novos clientes, pois adquirir novos clientes pode se tornar um processo que demanda muito esforço e dinheiro. No entanto, o gerenciamento de desligamento de cliente não deve focar em todos os clientes conquistados, uma vez que nem todos valem o custo de uma campanha de retenção, podendo resultar em desperdício de recursos. Por isso, esforços devem ser direcionados para manter clientes conquistados e valiosos como a maior prioridade [21].

2.2 Modelagem Temporal

Como a construção dos atributos deste trabalho segue uma abordagem sequencial, esses dados podem ser submetidos a técnica de modelagem de sequência. Por exemplo, dado uma entrada sequencial $x_i, x_{i+1}, x_{i+2}, \ldots, x_T$, onde a informação em x_i com $i < T$, representa um valor no passado, quando se considera o período de tempo T. Se um modelo quer predizer uma saída futura (y_t) com base no histórico de dados do passado ($x_0, ..., x_t$), ele deve, idealmente, levar em consideração a ordem em que os dados são amostrados, ou seja, o fator temporal. De maneira formal, um modelo sequencial é uma função $f : \mathcal{X}^{T+1} \to \mathcal{Y}^{T+1}$ que gera o mapeamento

$$\hat{y}_T = f(x_0, ..., x_T) \tag{2.1}$$

se essa satisfaz a restrição causal de que y_t depende apenas de $x_0, ..., x_t$ e não de nenhuma entrada subsequente $x_{t+1}, ..., x_T$.

2.3 Classificadores

Neste trabalho, utilizamos uma abordagem temporal para a construção do conjunto de dados, na qual os atributos são extraídos sequencialmente ao longo do tempo. Entretanto, duas versões foram desenvolvidas: uma sequencial e outra tabular. A versão sequencial foi criada para utilizar modelos baseados em redes convolucionais unidimensionais e redes de memória de longo prazo, que são especializados em modelagem de sequência. Por outro lado, na versão tabular, cada elemento de cada sequência é tratado como um atributo independente, tornando possível o treinamento de modelos de floresta aleatória e XG-boost, que não realizam modelagem de sequência. O funcionamento dos modelos utilizados neste trabalho são apresentados aqui.

2.3.1 Floresta Aleatória

A técnica floresta aleatória (FA) é um algoritmo do tipo *ensemble* introduzido por [8] que se baseia em Árvores de Decisão. O objetivo da FA é construir um modelo que possa generalizar bem os dados, evitando o *overfitting*, que ocorre quando o modelo se ajusta muito bem aos dados de treinamento, mas não consegue generalizar para novos dados. Para isso, a FA constrói um conjunto de árvores de decisão, onde cada árvore é construída a partir de uma amostra aleatória, com reposição, no conjunto de dados de treinamento. Em cada nó da árvore, é selecionado um subconjunto aleatório de atributos para reduzir a correlação entre as árvores e melhorar a generalização do modelo.

Em [8], cada árvore é construída da seguinte forma:

- Amostrando aleatoriamente (com reposição) N casos, se o conjunto de de dados treinamento possuir N casos. Esta amostra fornece o conjunto de dados de treinamento para o crescimento da árvore.
- Para M atributos de entrada, um número $m \ll M$ de atributos é especificado. A melhor divisão m de atributos selecionados aleatoriamente de M é usada para dividir o nó.
- Cada árvore cresce até a maior extensão possível (sem poda).

Para realizar a predição ou classificação de uma nova instância, a FA utiliza um sistema de votação entre as árvores de decisão que compõem o modelo. Esse sistema consiste em cada árvore gerar uma saída (voto) para a classe da instância, e a classe escolhida por mais árvores é a saída final do modelo para essa instância (conforme ilustrado na Figura 2.1).

Formalmente, a classe final prevista pela FA:

$$\hat{y}(x) = \textit{classe majoritária}\{\hat{y}_m(x)\}_1^M, \tag{2.2}$$

em que $\hat{y}_m(x)$ é a classe predita pela m-ésima árvore para a instância x e M é a quantidade de árvores. Essa abordagem de votação permite que a FA contorne problemas relacionados a ruídos e a possíveis *overfitting*.

A FA é conhecida por sua capacidade de convergência, o que a torna menos propensa a problemas de *overfitting*. Para ilustrar essa característica, é possível recorrer aos fundamentos teóricos apresentados por BREIMAN [8].

Para calcular o erro de generalização da Função de Aproximação (FA), simbolizado por PE^*, utiliza-se a seguinte fórmula:

$$PE^* = P_{X,Y}(mg(X,Y) < 0). \tag{2.3}$$

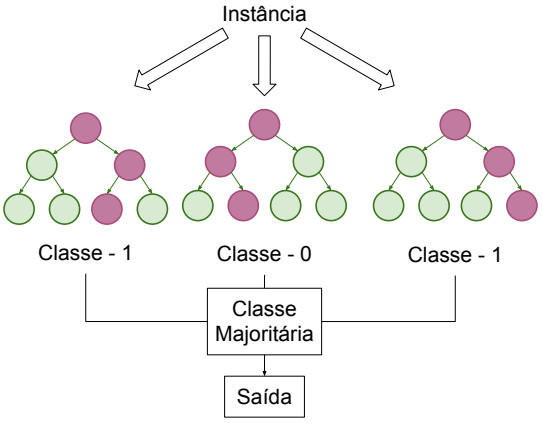


Figura 2.1: Representação da predição de nova instância no floresta aleatória. Fonte: O Autor.

Nessa equação, X representa o vetor preditor e Y a classificação. O termo $mg(X, Y)$ indica a diferença entre a média de votos para a classe correta e a média de votos para qualquer outra classe.

A equação 2.3 define a probabilidade condicional de erro de classificação, PE^*. Em outras palavras, ela indica a chance de ocorrer uma classificação incorreta, tendo em vista a distribuição conjunta entre o vetor preditor X e a classificação Y.

A função *Margin* é diretamente proporcional à confiança na classificação e é formalizada da seguinte forma:

$$mg(X, Y) = av_k I(h_k(X) = Y) - max_{(j \neq Y)} av_k I(h_k(X) = j), \tag{2.4}$$

em que $I(.)$ é a função do indicador e $av_k I(h_k(X) = Y)$ significa a média da função indicadora I que verifica se a saída $h_k(X)$ do k-ésimo classificador da floresta aleatória é igual à classe Y do exemplo de entrada X.

A força (ou *strength*) S da FA é dada em termos do valor esperado da função *Margin* como:

$$S = E_{X,Y}(mg(X, Y)). \tag{2.5}$$

Por fim, o erro de generalização da Função de Aproximação (FA) é limitado superiormente por uma função que leva em conta a correlação média entre os classificadores de base e sua(s) força(s) média(s). Denote por ρ o valor médio da correlação. Nesse caso, um limite superior para o erro de generalização pode ser expresso por:

$$PE^* \leq \rho(1 - s^2)/s^2. \tag{2.6}$$

Essa equação estabelece que o erro de generalização não pode exceder o valor $\rho(1 - s^2)/s^2$, em que s^2 representa a variância das forças dos classificadores de base. Logo, a precisão da FA depende da força das árvores e correlação entre elas, e como

consequência, o *overfitting* é evitado quando as árvores selecionadas são precisas e suficientemente independentes. Mais informações sobre o algoritmo podem ser encontradas em [8].

2.3.2 *Extreme Gradient Boosting*

O Extreme Gradient Booster (XG-boost) desenvolvido por CHEN e GUESTRIN [10] é uma extensão dos modelos de *Gradient Boosting* com maior poder e velocidade computacional. A técnica *boosting* aplicada a essa categoria de algoritmos é baseada na agregação recursiva de novos modelos para ajustar os erros realizados pelos modelos existentes. Novos modelos são adicionados até que não haja nenhuma melhoria perceptível.

Formalmente, um modelo *Boosting* de árvore usa K funções aditivas para prever a saída (Ver Figura 2.2)

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}, \tag{2.7}$$

em que $\mathcal{F}$ é o espaço das árvores de regressão, cada f_k corresponde a uma estrutura de árvore independente, $f_k(x_i)$ é árvore resultante de k e y^i é o valor predito da i-ésima instância x_i.

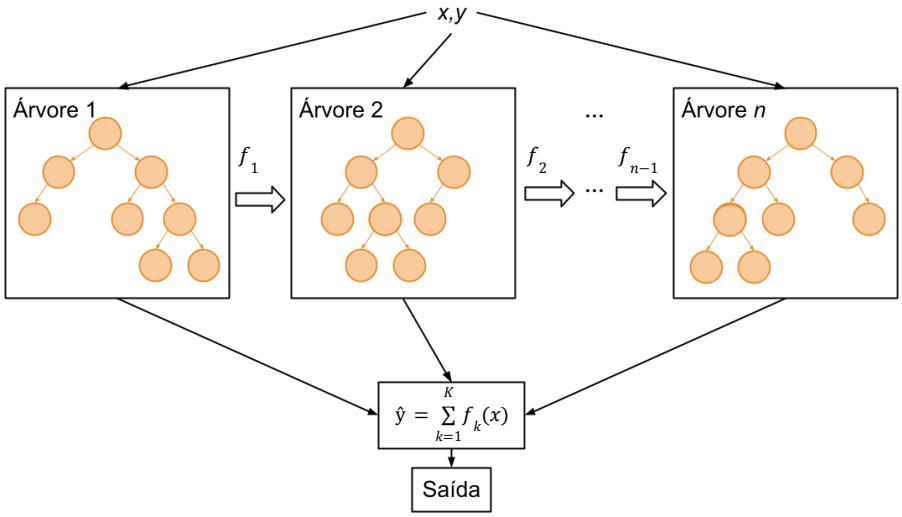


Figura 2.2: Representação da predição de nova instância no XG-boost. Fonte: O Autor.

O XG-Boost usa a seguinte função objetivo para desenvolver o número de árvores:

$$Obj(\theta) = L(\theta) + \Omega(\theta), \tag{2.8}$$

em que $L(\theta) = \ell(y_i, \hat{y}_i)$ é a função de perda que mede a discrepância entre as previsões do modelo e as saídas verdadeiras, $\hat{y}_i$ é a predição e y_i é classe verdadeira e o $\Omega(\theta)$ é

calculado como $\Omega(\theta) = \sum_{k=1}^{K} \Omega(f_k)$ e define um termo de regularização para controlar a complexidade do modelo a fim de evitar o *overfitting*.

No XG-boost, o modelo é otimizado (ou treinado) de forma aditiva. O procedimento é iniciado com uma previsão constante $\hat{y}_i^{(t)}$, e então, uma nova árvore f que minimiza a função de perda é pesquisada e adicionada a cada iteração. A i-ésima instância na t-ésima iteração é representada por

$$\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i). \tag{2.9}$$

Tendo isso, o objetivo pode ser representado da seguinte maneira:

$$Obj^t = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t). \tag{2.10}$$

Adicionalmente, a aproximação de segunda ordem pode ser usada para otimizar rapidamente o objetivo no cenário geral:

$$Obj^t = \sum_{i=1}^{n} \left(\ell(y_i, \hat{y}_i^{t-1} + g_i f_t(x_i)) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t), \tag{2.11}$$

em que g_i e h_i são estatísticas de gradiente de primeira e segunda ordem na função de perda, respectivamente.

O XG-boost é uma técnica com alto desempenho de classificação que combina eficiência e flexibilidade. Mais detalhes sobre a implementação do algoritmo podem ser encontrados em [10].

2.3.3 Redes Neurais Convolucionais Unidimensionais

As Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN), apresentada por LECUNN *et al.* [32], são redes populares para a solução de problemas envolvendo imagens e vídeos. Essas redes utilizam filtros de 2 dimensões em suas operações de convolução. Recentemente, um novo tipo de CNN foi apresentada que difere da tradicional ao trabalhar com filtros de uma dimensão, a CNN 1D. As CNNs 1D apresentam desempenho competitivo com modelos com menor custo computacional [29].

Em uma rede convolucional 1D, a entrada consiste geralmente em um vetor ou em múltiplos vetores. A partir disso, são executadas operações matemáticas denominadas convolução. Essa convolução é aplicada por meio de filtros que deslizam sobre o vetor (ou vetores), permitindo a extração de mapas de características.

Ao implementar uma CNN, é comum usar convolução não causal em que a saída no tempo t depende da entrada futura. Dada a entrada da camada de convolução de

comprimento T, representada por x, e o filtro de comprimento k, representado por h, o filtro é deslocado s posições (número de passos ou (*strides*)) após cada convolução [52]. Então a convolução não causal entre x e h para o passo s é definida como

$$y(t) = \begin{cases} \sum_{i=0}^{k} x(t+i)h(i), & \text{se } t = 0 \\ \sum_{i=0}^{k} x(t+i+(s-1))h(i), & \text{caso contrário.} \end{cases} \tag{2.12}$$

A convolução não causal é representada na Figura 2.3b. A saída das convoluções dado um tamanho de filtro maior que um gera vetores de saída menores que o de entrada. Para contornar esse problema pode-se aplicar preenchimento de zero, adicionando elementos vazios a sequência.

Na convolução não causal, sem preenchimento de zero, o comprimento da saída é:

$$o = \frac{(n-k)}{s} + 1, \tag{2.13}$$

em que n é comprimento da entrada, k é tamanho do filtro e s é tamanho do passo do filtro.

Para convoluções não causais, é utilizado o preenchimento do lado esquerdo e direito da sequência [52]. Com preenchimento de zero, o comprimento da saída:

$$o = \frac{(n+2p-k)}{s} + 1, \tag{2.14}$$

em que p é tamanho do preenchimento.

No entanto, nada impede a utilização de convolução causal. Nesse tipo de operação não se pode ter informação do futuro na operação. Dessa forma, deve-se garantir que para uma saída y no tempo t, apenas elementos no tempo t ou anterior podem fazer parte da operação de convolução [6, 52]. Então a convolução não causal entre x e h para o passo s é definida como

$$y(t) = \begin{cases} \sum_{i=0}^{k} x(t-i)h(i), & \text{se } t = 0 \\ \sum_{i=0}^{k} x(t-i+(s-1))h(i), & \text{caso contrário.} \end{cases} \tag{2.15}$$

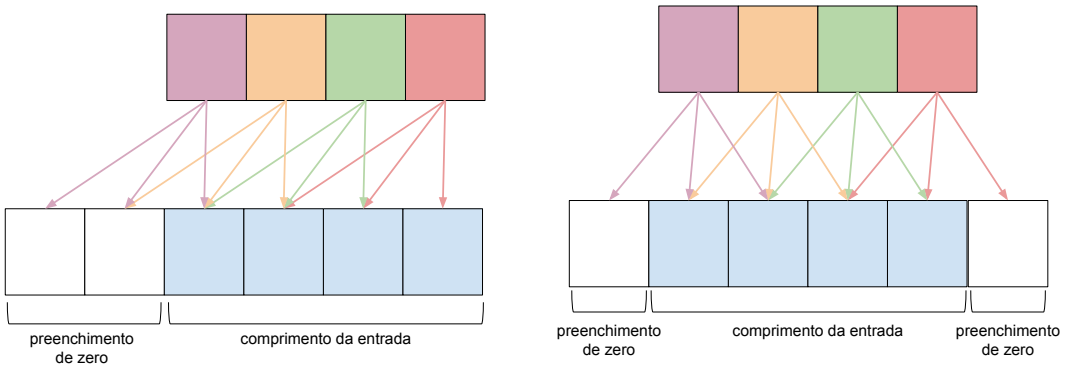
Para convoluções causais, é utilizado o preenchimento apenas do lado esquerdo. Deste modo, a saída em cada posição depende apenas das entradas anteriores na sequência, mantendo a convolução causal. A convolução causal é representada na Figura 2.3a.

Na convolução causal, com preenchimento de zero, o comprimento da saída:

$$o = \frac{(n+p-k)}{s} + 1, \tag{2.16}$$

em que n é o comprimento da entrada, k é o tamanho do filtro, s é o tamanho do

passo do filtro e p e comprimento do preenchimento.



(a) convoluções causais (b) convoluções não causais

Figura 2.3: Nas figuras são representadas as convoluções causais e não causais. Fonte: O Autor.

Quando há múltiplas sequências de entrada, o processo descrito acima é repetido para cada uma delas, com um filtro diferente a cada vez. Os vetores de saída intermediários são somados para gerar o vetor de saída final. Isso é análogo a uma convolução 2D com filtro 2D (com dimensões correspondentes ao tamanho do filtro e número de atributos). Entretanto, ainda é considerado uma convolução 1D, pois a janela de convolução desliza somente ao longo de um eixo, conforme exemplificado na Figura 2.4.

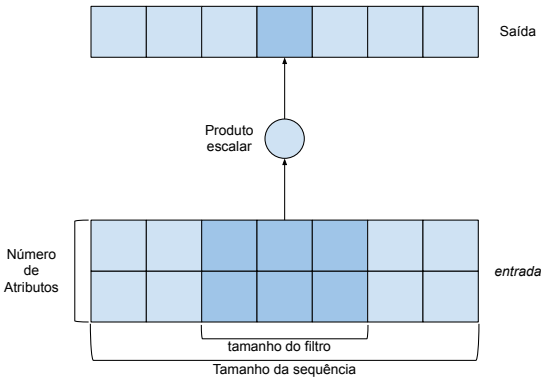


Figura 2.4: Representação de uma convolução em uma entrada com mais de uma sequência. Fonte: O Autor.

As camadas de convolução são frequentemente usadas para extrair novas características dos dados. No entanto, para construir um modelo completo baseado em convoluções, é necessário adicionar camadas e operações adicionais. As camadas mais comuns incluem *Pooling*, de ativação *Dropout*, *Flatten*, *Normalização* e camadas totalmente conectadas (ou camadas densas).

Nas próximas seções, serão apresentados detalhes sobre como cada uma dessas camadas é usada em um modelo de CNN 1D, o que ajudará a entender como as camadas podem ser combinadas para criar um modelo baseado em CNN mais eficiente.

Pooling

As camadas *Pooling* são utilizada para reduzir a dimensionalidade de um ou vários mapeamentos enquanto destaca os recursos importantes. As técnicas mais populares são *Max-Pooling* e *Global average Pooling*:

- ***Max-pooling***. Essa camada é utilizada para extrair o valor máximo em uma janela e esta janela é deslizada sobre a entrada com um passo após cada operação de *pooling*. A operação pode ser expressada como:

$$MaxPool(x_i) = \max_{j=i}^{i+S-1} x_j, \tag{2.17}$$

  em que x_i é o valor na posição i do vetor de entrada, S é o tamanho do filtro de *pooling*. Geralmente, essa técnica é empregada após a camada de convolução, ajudando também na redução de *overfitting* [28].

- ***Global average pooling***. Essa camada é utilizada para extrair a média de cada mapa de característica para gerar um vetor resultante que alimentará a camada a próxima camada. A operação GAP pode ser expressada como:

$$\text{GAP}(x_i) = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{2.18}$$

  em que x_i é o valor na posição i do vetor de entrada e N é o número total de elementos no vetor. Esse tipo de camada é adicionada ao fim do bloco convolucional substituindo uma camada totalmente conectada. O apelo dessa camada é que ela não utiliza parâmetros treináveis auxiliando na redução de *overffitng* [34].

Ativação

As camadas de ativação possuem uma função de ativação $f(\cdot)$ que decide se uma informação contida na rede é relevante ou não. A função de ativação é a transformação não linear realizada ao longo do sinal recebido. Esta saída transformada é então enviada para a próxima camada como entrada. Existem várias funções de ativação com diferentes transformações não-lineares. Algumas das mais populares em modelos

CNN são: *Rectified Linear Unit* (ReLU)[40], *Gaussian Error Linear Unit* (Gelu) [23] e *Exponential Linear Unit* (Elu) [13].

Na ativação ReLU, a saída y é dada por:

$$y = f(x) = \max(0, x), \tag{2.19}$$

em que x é a entrada. A função $\max(0, x)$ define que, se x é negativo, a saída será zero e, caso contrário, a saída será igual à entrada [50].

Já na ativação Gelu, a saída y é dada por:

$$y = f(x) = x\Phi(x), \tag{2.20}$$

em que $\Phi(x)$ é a função de distribuição cumulativa gaussiana padrão, definida como:

$$\Phi(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]. \tag{2.21}$$

A função GELU tem a vantagem de ser mais rápida do que outras funções de ativação como a ReLU em determinadas arquiteturas de rede [23].

Finalmente, a ativação ELU é definida por:

$$y = f(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha(e^x - 1) & \text{se } x \leq 0, \end{cases} \tag{2.22}$$

em que α é uma constante positiva. Ao contrário de ReLU, a função ELU também tem um valor negativo que faz com que a média das ativações se aproxime de zero [27].

Dropout

O *Dropout* é uma técnica de regularização usada em redes neurais convolucionais (CNNs) para prevenir overfitting. Durante o treinamento, o *Dropout* desativa aleatoriamente um conjunto de neurônios em uma camada com uma probabilidade pré-definida. Isso ajuda a rede a aprender representações redundantes em vários subconjuntos de neurônios, melhorando a generalização do modelo para dados de teste [31].

Ao implementar um modelo de rede neural convolucional 1D, a técnica de regularização *Dropout* pode ser adicionada após as camadas convolucionais ou densas, com o objetivo de aplicar *Dropout* em toda a camada. A ativação de cada neurônio de entrada x_j na passagem para frente é multiplicada por um fator aleatório α_i:

$$y_i = \alpha_i x_i, \text{ onde } \alpha_i \sim Bernoulli(p). \tag{2.23}$$

em que α_i uma variável aleatória que segue uma distribuição Bernoulli com parâmetro p que controla a participação de x_i na passagem para frente. A probabilidade de manter x_i ativo durante o treinamento é definida como p, e a probabilidade de desativá-lo é $(1-p)$. Portanto, α_i é uma variável aleatória que assume valores 0 ou 1, com probabilidade $(1-p)$ e p. Dessa forma, o *Dropout* impõe uma restrição aleatória nos neurônios de entrada, forçando o modelo a aprender caraterísticas mais robustas que não dependem da presença de neurônios específicos.

Flatten

A camada *Flatten* é utilizada em redes neurais para transformar um tensor multi-dimensional em um vetor unidimensional, a fim de ser direcionado como entrada para uma camada totalmente conectada. Essa camada é comumente utilizada após camadas de convolução em redes convolucionais, em que a saída dessas camadas pode ter uma profundidade maior que um.

Por exemplo, em uma rede convolucional que recebe um tensor de entrada x com dimensão $(batch\_size, steps, input\_dim)$, em que $batch\_size$ é o tamanho da amostra de instâncias, $steps$ é o número de passos (comprimento da sequência) e $inputs\_dim$ é o número de atributos (ou canais), a camada *Flatten* transforma a entrada x em um vetor unidimensional, alterando a forma do tensor x sem alterar seus valores.

Normalization

As camadas de normalização são técnicas utilizadas em redes neurais para melhorar o desempenho do treinamento e evitar problemas como o dissipação ou explosão do gradiente [5]. Problemas de dissipação e explosão de gradiente podem ocorrer durante o treinamento da rede neural, levando a uma convergência lenta ou mesmo a um colapso do treinamento. A dissipação do gradiente ocorre quando o gradiente diminui rapidamente à medida que se propaga pela rede, enquanto a explosão do gradiente ocorre quando o gradiente cresce exponencialmente. Existem várias camadas de normalização, incluindo *Batch Normalization* [25], *Weight Normalization* [48] e *Layer Normalization* [5].

- ***Batch Normalization***. Essa abordagem é utilizada para normalizar a entrada em um mini-lote para cada camada de uma rede neural, tornando o treinamento mais estável e rápido. É uma técnica popular de normalização amplamente utilizada em aplicações de *deep learning*. Ela adiciona camadas que definem a média e a variância de cada ativação como zero e um, respectivamente, e normaliza as entradas em lote com base em parâmetros treináveis. Isso é aplicado antes da não-linearidade da camada anterior para melhorar a estabilidade da rede durante o treinamento [49].

Dado um mini-lote (ou *mini-batch*) de entradas $\mathbf{x} = \mathbf{x}_1, ..., \mathbf{x}_m$, o *Batch Normalization* transforma cada entrada $\mathbf{x}_i$ em $\mathbf{y}_i$ da seguinte forma:

$$\mathbf{y}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma} + \epsilon} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}, \tag{2.24}$$

em que $\boldsymbol{\mu}$ e $\boldsymbol{\sigma}$ são a média e o desvio padrão do mini-lote, respectivamente, ϵ é uma pequena constante para evitar divisão por zero, $\odot$ denota a multiplicação elemento a elemento, e $\boldsymbol{\gamma}$ e $\boldsymbol{\beta}$ são parâmetros de escala e deslocamento que podem ser aprendidos, respectivamente [25].

- ***Weight Normalization***. Essa técnica é utilizada para reparametrizar os vetores de peso em uma rede neural, acelerando a convergência da otimização estocástica do gradiente descendente. Ela padroniza a distribuição das entradas para cada entrada, resultando em uma norma de saída aproximadamente igual à norma de entrada. O objetivo é reduzir a variação da distribuição de entrada para cada camada e resolver parcialmente o problema de desaparecimento e explosão de gradientes [19, 48]. Os vetores de peso gerados são expressos em termos dos novos parâmetros usando:

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|}\mathbf{v}, \tag{2.25}$$

em que $\mathbf{v}$ é o vetor de parâmetro, g é um escalar e $\|\mathbf{v}\|$ denota a norma euclidiana de $\mathbf{v}$. Esta reparametrização tem o objetivo de fixar a norma euclidiana de $\mathbf{w}$, de modo que, $\|\mathbf{w}\| = g$, independente dos parâmetros $\mathbf{v}$.

- ***Layer Normalization***. A normalização de camada é uma técnica que estimativa diretamente as estatísticas de normalização das entradas somadas para os neurônios dentro de uma camada oculta. Isso garante que a normalização não introduza nenhuma nova dependência entre os casos de treinamento. Além disso, na normalização de camada, todos os neurônios de uma camada têm a mesma distribuição em relação aos atributos de entrada, o que também ajuda a evitar instabilidades e inconsistências nos dados [5].

  A normalização de camada é realizada de forma semelhante à técnica de *Batch Normalization*. No entanto, em vez de normalizar as entradas para cada mini-lote, a normalização de camada normaliza as entradas para todas as unidades ocultas de uma camada. Para a entrada $\hat{x}$, que é uma representação vetorial das entradas somadas para os neurônios de uma camada, a normalização é realizada da seguinte maneira:

$$y_i = \frac{\hat{x}_i - \mu}{\sigma + \epsilon} \odot \gamma + \beta, \tag{2.26}$$

em que μ e σ são a média e o desvio padrão de $\hat{x}$, respectivamente.

Camadas Totalmente Conectadas

As camadas totalmente conectadas se tornaram populares graças a autores como [30], e desde então têm sido amplamente utilizadas em conjunto com CNNs, especialmente para tarefas de classificação ou predição. Cada neurônio em uma camada totalmente conectada está conectado a todos os neurônios da camada anterior, permitindo que a rede neural forneça um caminho de decisão para as classes de resposta com base no aprendizado adquirido pelos filtros nas camadas convolucionais. Normalmente, a camada totalmente conectada é usada no final da rede para fazer uma combinação não linear de recursos selecionados globalmente de todas as camadas anteriores para a classificação de dados [44].

Formalmente,a saída de uma camada totalmente conectada (ou camada densa) em uma rede neural pode ser descrita da seguinte forma:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

em que $\mathbf{x}$ é o vetor de entrada da camada, $\mathbf{W}$ é a matriz de pesos que conecta cada neurônio da camada de entrada a cada neurônio da camada de saída, $\mathbf{b}$ é o vetor de *bias* (ou viés), e σ é a função de ativação aplicada a cada elemento do vetor resultante $\mathbf{W}\mathbf{x} + \mathbf{b}$.

Em problemas de classificação binária, é comum adicionar uma função sigmoide $\sigma(z) = \frac{1}{1+e^{-z}}$ à saída da camada totalmente conectada para gerar a saída do modelo. A função sigmoide retorna valores no intervalo aberto (0, 1).

2.3.4 Redes Convolucionais Temporais

As Redes Neurais Convolucionais Temporais (TCNN) não são uma arquitetura de rede convolucional nova, mas sim um termo que descreve um conjunto de arquiteturas [6]. Essas redes são capazes de fazer previsões com base em um passado distante, usando camadas convolucionais dilatadas e causais. Segundo BAI *et al.* [6], duas características definem uma rede como TCNN: (i) suas convoluções devem ser causais e (ii) a arquitetura deve ser capaz de processar sequências de entrada de qualquer tamanho e mapeá-las para uma sequência de saída de mesmo tamanho.

Para que uma convolução seja causal, é necessário que ela não possua informação do futuro na operação, como ilustrado na Figura 2.3a. Ao utilizar uma rede de convolução tradicional para uma tarefa de previsão, é limitado o campo receptivo das operações de convolução. Para aproveitar melhor o comprimento do dado de entrada, as TCNNs frequentemente utilizam convoluções dilatadas [6]. As convoluções dilatadas são equivalentes a introduzir um distanciamento fixo entre os elementos

da sequência que são enxergados pelo filtro, conforme ilustrado na Figura 2.5. Formalmente, para uma sequência 1D entrada x e um filtro $f : \{0, ..., k-1\}$, a operação de convolução dilatada F no elemento s da sequência é da forma:

$$F(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{2.27}$$

em que d é o fator de dilatação, k é o tamanho do filtro e $s - d \cdot i$ representa a direção do passado.

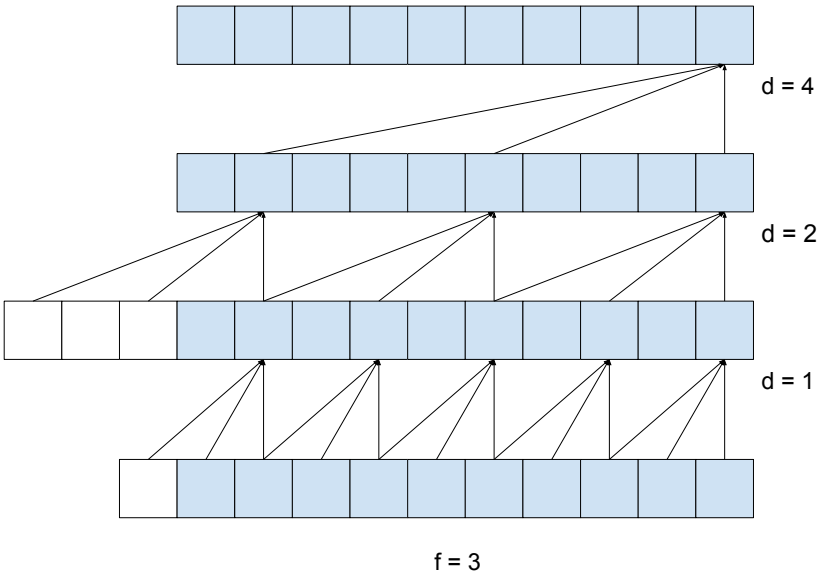


Figura 2.5: Representação das convoluções dilatadas. As dilatações são representadas pelo espaçamento entre os elementos visualizados pelos filtros. Os filtros, por sua vez, são indicados pelos segmentos que unem os elementos em cada linha. Na figura, o filtro tem tamanho igual a três e d indica o fator de dilatação. Fonte: O autor.

As Redes Neurais Convolucionais Temporais (TCNNs) utilizam conexões residuais para melhorar o desempenho da rede. Essas conexões são formadas adicionando a saída de uma operação de convolução 1x1 à saída das camadas convolucionais dilatadas, criando um bloco residual, conforme ilustrado na Figura 2.6. O uso de conexões residuais é benéfico para aliviar o problema de degradação da rede em arquiteturas mais profundas, permitindo que a informação flua diretamente através das camadas. A adição da operação de convolução 1x1 antes da adição, juntamente com o *padding* causal, garante que a sequência de entrada e saída tenha o mesmo tamanho, satisfazendo o segundo critério [6].

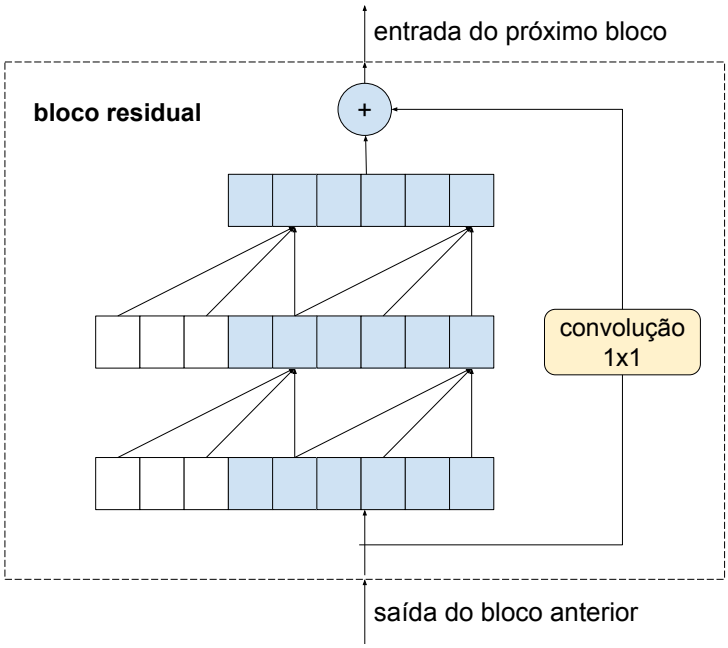


Figura 2.6: Representação do bloco residual. Fonte: O Autor.

2.3.5 Redes de Memória de Longo Prazo

As Redes de Memória de Longo Prazo, também conhecidas como LSTM (*long short-term memory*), foram introduzidas por HOCHREITER e SCHMIDHUBER [24] como uma arquitetura específica para lidar com dados sequenciais. Essas redes foram projetadas para evitar problemas de dependência de longo prazo e são otimizadas através do cálculo de pesos conectados, evitando a explosão e o dissipação de gradientes. Além disso, elas contam com *auto-loops*, que atuam como caminhos pelos quais os gradientes podem fluir por períodos mais longos [42, 60].

A LSTM é composta por células de memória que são conectadas em sucessivas camadas, conforme ilustrado na Figura 2.7. Cada camada possui portais, conhecidos como *gates*, que gerenciam a memória e controlam a entrada e saída de informações no estado da célula.

Na primeira operação, a LSTM decide quais informações serão descartadas do estado da célula atual recebido. O primeiro portal, conhecido como *forget gate* ou portal do esquecimento (Figura 2.8), gerencia a entrada da célula e o *hidden state* anterior e tem como finalidade decidir em qual grau a informação contida na célula anterior deve ou não ser mantida [42, 60]. Formalmente, a informação propagada é computada através da equação:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.28}$$

em que f_t é o vetor de ativação do *forget gate* no tempo t, h_{t-1} é o *hidden state* anterior, x_t é o vetor de entrada no tempo t, W_f e b_f são os parâmetros da camada

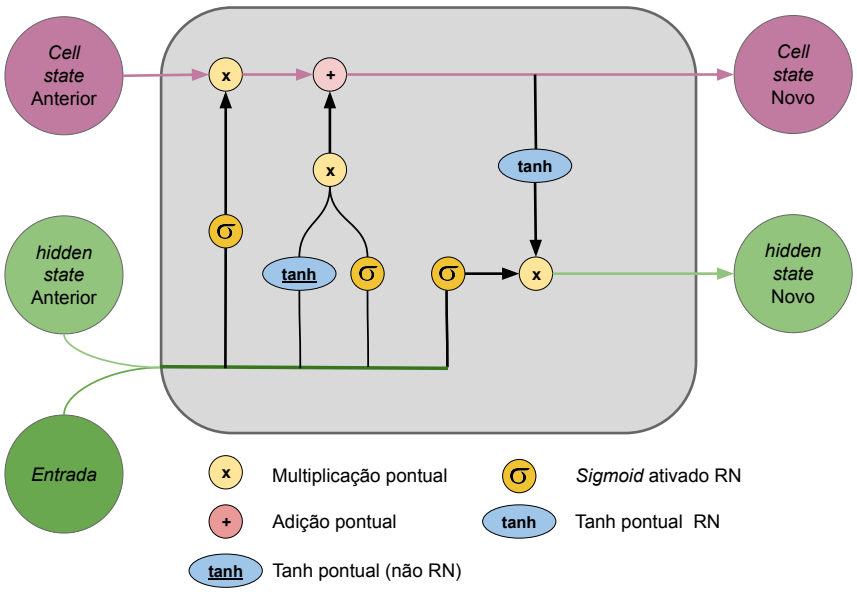


Figura 2.7: Representação de uma célula LSTM. Fonte: O Autor.

do *forget gate* e σ é a função sigmoide. Assim, valores próximos a 0 representam informações a serem esquecidas e, no caso de 1, a serem mantidas.

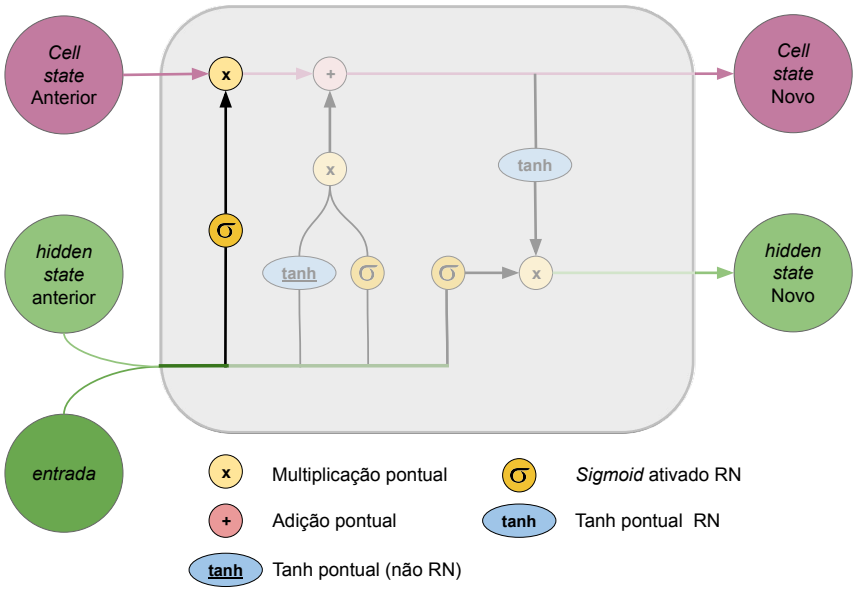


Figura 2.8: Representação de uma célula LSTM. Fonte: O Autor.

Após o portal de esquecimento, tem-se o *input gate layer* ou portal da camada de entrada. Seu papel é decidir, a partir do *hidden state* (estado oculto) anterior e a entrada atual, quais novas informações entrarão no estado da célula. Para isso, são necessárias duas operações, vistas na Figura 2.9: uma camada com função de ativação sigmoidal σ para decidir quais partes do vetor farão parte da saída e uma outra camada com uma função de ativação tangente hiperbólica tanh, que produz um vetor de valores de candidatos. Feito isso, a atualização é feita por uma multiplicação

do valor do primeiro portal e adição do valor do portal de saída ao estado da célula [42, 60]. As equações correspondentes são:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C), C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t, \quad (2.29)$$

em que i_t é o vetor de ativação do *input gate layer* no tempo t, $\tilde{C}_t$ é o vetor de valores de candidatos, C_t é o estado da célula no tempo t, W_i, W_C, b_i, e b_C são parâmetros das camadas correspondentes. O primeiro termo na última equação representa as informações que são mantidas na célula e o segundo termo representa as novas informações que entrarão na célula.

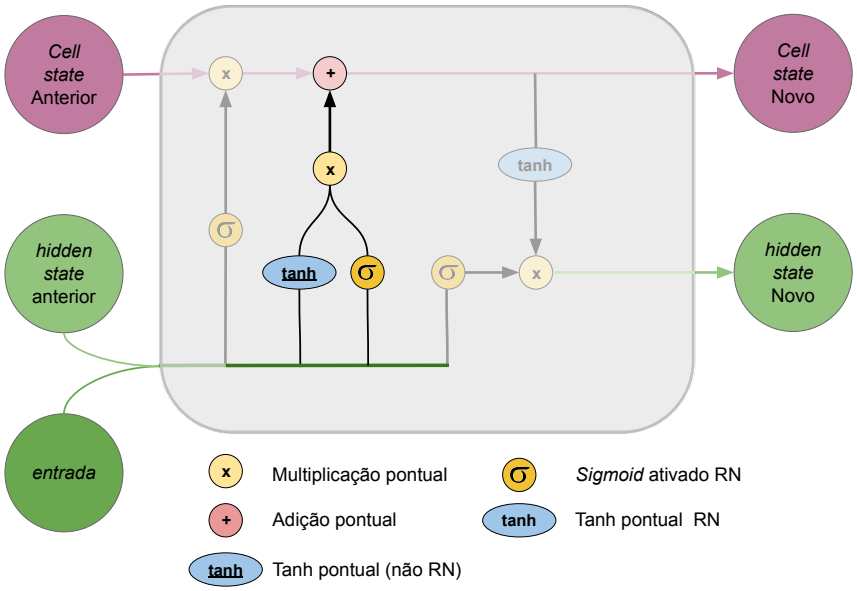


Figura 2.9: Representação da nova rede de memória e portal da camada de entrada. Fonte: O Autor.

No final, o *output gate* ou portal de saída (ver Figura 2.10) decide quais informações do estado da célula serão transmitidas. Dois passos são necessários para produzir a saída da célula: primeiro, uma função sigmoide σ que decide quais partes do vetor farão parte da saída e, logo após, o estado da célula é passado para uma função de ativação tangente hiperbólica tanh(.) (normalizando os valores entre -1 e 1) e então multiplicado pelo resultado da operação do passo anterior e o resultado é o novo estado da célula. As equações correspondentes são:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), h_t = o_t \cdot \tanh(C_t), \quad (2.30)$$

em que o_t é o vetor de ativação do *output gate* no tempo t, h_t é o *hidden state* no tempo t, W_o e b_o são parâmetros da camada correspondente. O primeiro termo na segunda equação representa as informações que são transmitidas e o segundo termo é a normalização do estado da célula.

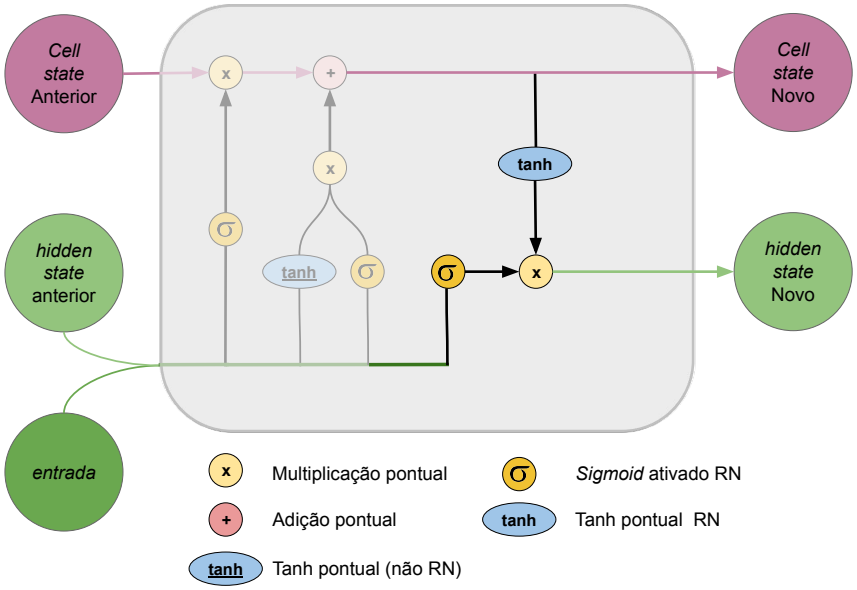


Figura 2.10: Representação do portal de saída. Fonte: O Autor.

2.4 Busca de Arquitetura Neural

A busca por arquiteturas neurais (*Neural Architecture Search* - NAS) substitui o processo manual de encontrar uma melhor arquitetura de redes neurais artificiais por um processo automatizado de otimização no aprendizado de máquina. A NAS tem sido investigada desde o final da década de 1980 [38, 55], mas o interesse no campo aumentou com a aplicação de Aprendizagem por Reforço [62]. Desde então, a NAS tem se tornado cada vez mais reprodutível, inicialmente concentrada em problemas de classificação de imagens e estendendo-se agora ao processamento de linguagem natural [26, 56].

Segundo ELSKEN *et al.* [17], os métodos NAS podem ser classificados de acordo com três dimensões:

1. **Espaço de busca**: define quais arquiteturas podem ser representadas pelo algoritmo. Para isso, é incorporado um conhecimento prévio das propriedades típicas de uma arquitetura adequada para a tarefa a ser realizada. Sendo assim, o espaço de busca é desenhado introduzindo um viés humano, limitando, de certa forma, as possibilidades de saídas do algoritmo.

2. **Estratégia de busca**: tem o intuito de detalhar como explorar o espaço de busca. É desejável que o algoritmo encontre boas arquiteturas rapidamente, mas também evite a convergência prematura para uma região de arquiteturas sub-ótimas.

3. **Estratégia de estimativa de desempenho**: é o processo de estimar o desempenho das arquiteturas. Geralmente, o caminho mais simples é realizar

um treinamento padrão e validação da arquitetura nos dados, o que é computacionalmente caro. Portanto, trabalhos recentes buscam alternativas para essa etapa, buscando reduzir o custo das avaliações de desempenho.

Um ponto importante a ser considerado em NAS é a abordagem dos objetivos. A busca pode ter como foco a otimização de um único objetivo, como a maximização da acurácia, ou de múltiplos objetivos, como a maximização da acurácia e a minimização da perda. Na otimização multi-objetivo, a tarefa se torna ainda mais desafiadora quando os objetivos são conflitantes, como no caso em que se deseja maximizar a acurácia e, ao mesmo tempo, minimizar o tamanho do modelo (por exemplo, a quantidade de parâmetros treináveis) [36, 57].

A Figura 2.11 ilustra os métodos utilizados na busca por arquiteturas neurais.

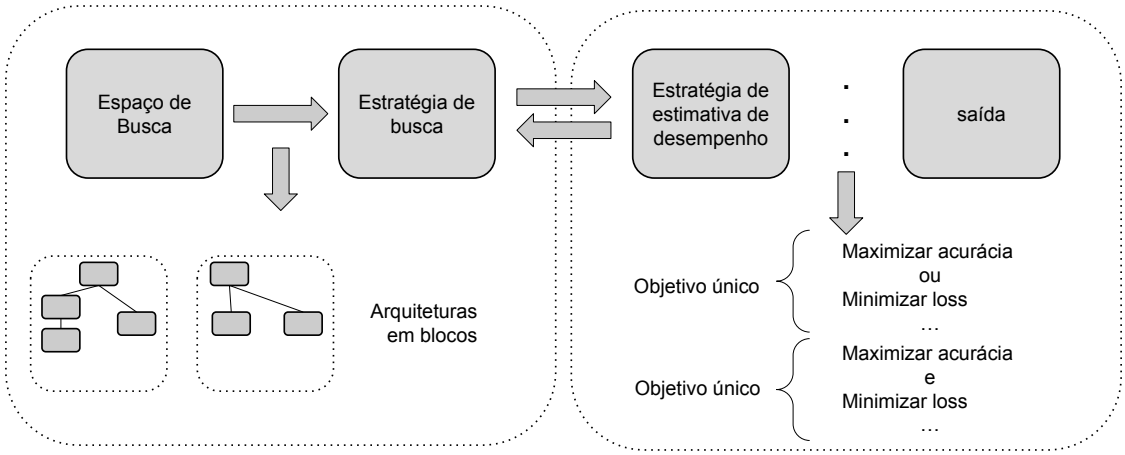


Figura 2.11: A figura representa de forma abstrata os métodos utilizados na busca de arquitetura neural. Inicialmente, uma estratégia de busca é utilizada para selecionar uma arquitetura dentro de um espaço de busca predefinido. Em seguida, essa arquitetura é avaliada pela estratégia de estimativa de desempenho, que retorna uma estimativa do desempenho para a estratégia de busca. Esse processo é repetido recursivamente até que uma saída seja gerada, ou seja, o processo finalizado. Fonte: O Autor.

2.5 Algoritmo Genético

Algoritmo genético (AG) é uma técnica de otimização baseada no processo de seleção natural da evolução biológica [20]. Utilizando uma abordagem de busca heurística, os AGs iterativamente geram soluções candidatas para um problema, avaliam sua aptidão, aplicam os operadores de seleção, cruzamento e mutação para criar novas soluções, e repetem o processo até um critério de parada ser atingindo. Os AGs são amplamente aplicados em problemas de otimização, em áreas como aprendizado de máquina, engenharia, finanças, entre outros [56, 59].

Para estabelecer um paralelo entre os termos de uma situação biológica e uma situação de otimização matemática, é importante entender o significado de cada termo em cada contexto. Na biologia, uma população é um grupo de indivíduos que pertencem à mesma espécie e vivem em uma mesma área geográfica. No contexto de um AG, a população é um conjunto de soluções candidatas representadas por um conjunto de genes que são manipulados para gerar novas soluções. Na biologia, um indivíduo é um organismo único que pertence a uma determinada espécie. No contexto de um AG, um indivíduo é uma solução candidata única, que é representada por um conjunto de genes ou cromossomos. Cada gene é uma parte da solução candidata que é manipulada pelo algoritmo para gerar novas soluções [51].

A seguir, são descritos os operadores utilizados pelos AGs.

- **Seleção**. A seleção é um mecanismo essencial em algoritmos genéticos para escolher os indivíduos a serem usados na etapa de transformação do algoritmo, através dos operadores de cruzamento e mutação. Existem vários métodos de seleção, incluindo seleção por torneio, seleção por roleta, seleção por classificação, entre outros [18].

- **Cruzamento**. O operador de cruzamento (ou *crossover*) do algoritmo genético é responsável por criar novos indivíduos da população a partir de combinações dos genes de dois ou mais indivíduos selecionados. O objetivo do cruzamento é explorar a diversidade genética da população e gerar novas soluções que possam ter características positivas de ambos os pais[15]. Existem vários métodos de cruzamento, e a escolha do método depende do problema e a parametrização utilizada.

- **Mutação**. O operador de mutação do algoritmo genético é responsável por introduzir aleatoriedade na população, a fim de explorar novas áreas do espaço de busca e evitar a convergência prematura para soluções sub-ótimas. A mutação geralmente envolve a alteração aleatória de um ou mais genes de um indivíduo selecionado [39].

Uma representação visual de um algoritmo GA convencional é apresentada na Figura 2.12.

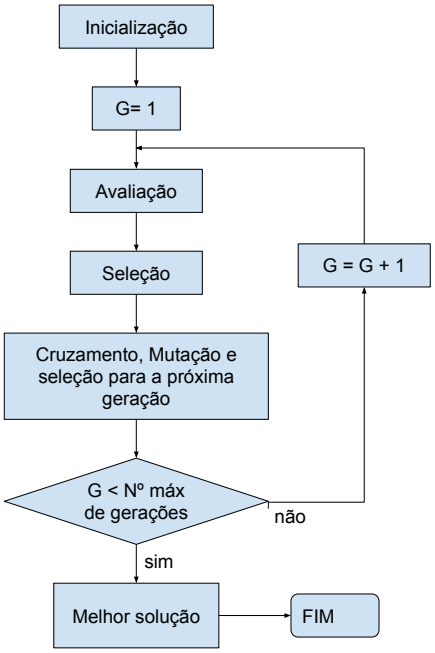


Figura 2.12: Ilustração de um algoritmo genético convencional. Fonte: O Autor.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, serão apresentados estudos relacionados ao tema proposto que utilizam algoritmos inteligentes para busca de arquiteturas neurais e otimização de hiperparâmetros. Na literatura, diversas técnicas têm sido aplicadas para solucionar esse problema. Uma revisão abrangente da literatura sobre o problema de NAS é apresentada em [56].

LIU *et al.* [35] propõe o algoritmo DARTS (*Diferentiable Arquitecture Search*) para busca de arquitetura de redes neurais baseadas em redes convolucionais e recorrentes. O DARTS é uma técnica de NAS *one-shot* que usa um relaxamento contínuo do espaço de busca de arquitetura discreta. No algoritmo, é realizado uma otimização dos hiperparâmetros e pesos da rede por meio do método do gradiente e então induz uma arquitetura final a partir das combinações aprendidas. Nos experimentos realizados, o DARTS superou o estado da arte para o problema de modelagem de linguagem no conjunto de dados *Penn Treebank* com 55.7 de perplexidade e mostrou-se competitivo para a tarefa de classificação de imagens.

SUGANUMA *et al.* [53] propõem uma alternativa de NAS evolutiva para a tarefa de restauração de imagem baseada em *autoencoders* convolucionais. O algoritmo visa otimizar os hiper parâmetros e as conexões das camadas com o objetivo de minimizar a perda padrão ℓ_2. A função de seleção utilizada seleciona os melhores indivíduos entre a população pai e a população filha resultante do processo de mutação. Nos experimentos realizados, a arquitetura encontrada pela abordagem alcançou 33,3 dB de *peak signal-to-noise ratio* (PSNR) no conjunto de dados SVHN, superando métodos estado da arte.

REAL *et al.* [45] propõe um algoritmo NAS evolutivo para criar o classificador de imagens AmoebaNet-A. Neste trabalho foi utilizado um espaço de busca baseado em blocos do tipo Inception. Adicionalmente, foi implementado a seleção por torneio com uma modificação para dar preferência a indivíduos mais recentes, mesmo que esse tenha menor desempenho na avaliação. Nos experimentos, em sua versão com maior número de parâmetros, o AmoebaNET-A atinge a acurácia de 83,9% superando

o estado da arte no momento em que foi publicado. Ainda nos experimentos, os autores defendem que a abordagem genética pode ter menor custo computacional quando comparado com abordagens de aprendizado por reforço.

LU *et al.* [36] propõe uma abordagem NAS baseada em algoritmos genéticos com dois objetivos: minimizar o erro e a complexidade computacional. No algoritmo, chamado de NSGA-Net, são aplicados procedimentos de *Exploration* e *Exploitation* para gerar combinações de blocos por meio de cruzamento e mutação, e, em seguida, esses blocos são conectados para gerar uma arquitetura final por meio de uma rede Bayesiana. Como base, são utilizados blocos de redes conhecidas, como ResNet e DenseNet. Nos resultados obtidos, o NSGA-Net mostrou-se competitivo em comparação a métodos NAS de última geração para a tarefa de classificação de imagens no conjunto de dados CIFAR-10, com uma taxa de erro de 3,85%. Esse resultado mostra o potencial dos algoritmos evolutivos para o problema de NAS.

ZHANG [61] propôs um algoritmo genético com foco na otimização de hiperparâmetros para melhorar a precisão de modelos baseados em redes neurais profundas na previsão do mercado financeiro. O algoritmo realiza a seleção de combinações de funções de ativação que minimizam a erro médio quadrático (*Mean squared error* - MSE). O algoritmo encontrou modelos com diferentes funções de ativação que superaram os modelos com apenas uma função de ativação, resultando em um MSE de 0.0351 para o conjunto de dados *Dow Jones Industrial Average* e 0.0132 para o *30-Year Treasury Constant Maturity Rate*.

XIE e YUILLE [59] propõe um algoritmo genético que utiliza codificação em string binária para otimizar redes neurais compostas por três segmentos de múltiplas camadas de convolução. O algoritmo é avaliado com base na precisão de classificação e utiliza seleção proporcional de aptidão para garantir a sobrevivência dos indivíduos mais aptos. Testado no conjunto de dados CIFAR10, o método atingiu uma acurácia média de 76.81% e as estruturas aprendidas foram transferíveis para o conjunto de dados ILSVRC2012.

Neste estudo, trabalhamos com um conjunto de dados sequenciais, o que torna as redes especializadas em dados sequenciais, como LSTM e TCNN, uma escolha interessante para o problema. No entanto, também consideramos a utilização de redes que não levam em conta explicitamente a dimensão temporal dos dados, como CNN unidimensional não causal e camadas densas. Acreditamos que algoritmos genéticos são uma abordagem promissora, uma vez que permitem uma busca adaptável no espaço de soluções e possuem um desempenho competitivo. Dessa forma, podemos testar diferentes camadas de redes e operações, bem como combiná-las para criar um modelo final. Além disso, podemos realizar uma otimização de hiperparâmetros simultaneamente à busca da arquitetura.

Capítulo 4

Definição do Conjunto de Dados

Os dados foram fornecidos pela Efí S.A, uma instituição financeira brasileira e disponibilizados em forma de banco de dados relacional. Movimentações de serviços financeiros utilizados pelos clientes foram registrados compreendendo um período de 88 semanas, de agosto de 2019 a outubro de 2021. Ressaltamos que os dados fornecidos não incluem um indicador de desligamento. Respeitando a Lei Geral de Proteção de Dados brasileira, todas as informações foram devidamente anonimizadas e os respectivos valores das transações normalizados.

Neste trabalho, a extração de atributos segue uma abordagem temporal. As informações dos clientes foram extraídas considerando uma janela de de 52 semanas dividida em duas sub-janelas temporais. Os registros das transações relativas à sub-janela temporal das 26 semanas mais antigas foram separados para a extração dos atributos para gerar os atributos do conjunto de dados de treinamento. Os registros das transações das outras 26 semanas (as mais recentes) foram reservados para o processo de rotulagem dos dados, visando aprendizado supervisionado. A escolha do tamanho das sub-janelas foi definida a partir de conversas com especialistas da empresa baseando-se no padrão de uso dos clientes. A janela deve ser grande o suficiente para capturar o comportamento do cliente. Todavia, um tamanho muito grande de janela pode capturar comportamentos de usos no passado que não representem mais o cliente atual. A Figura 4.1 apresenta a separação dos dados.

A janela foi deslizada em passos de uma semana para extrair novas instâncias de clientes em diferentes períodos. Dessa maneira, um cliente pode ser um cliente desligado em um intervalo de observação e não desligado em um intervalo diferente. Um total de 62 deslizamentos foram criados no período disponibilizado pelos dados fornecidos. A Figura 4.2 ilustra o processo de deslizamento.

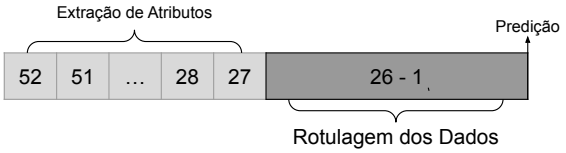


Figura 4.1: Separação dos históricos de registros: as 26 semanas mais antigas geraram os atributos e as 26 semanas (mais recentes) reservadas para a rotulagem dos dados. Fonte: O Autor.

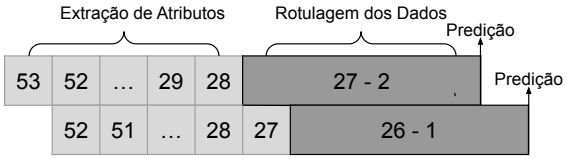


Figura 4.2: Representação de um deslizamento. Fonte: O Autor.

4.1 Extração de atributos

Visando um conjunto de dados mais apropriado para aprendizagem supervisionada, foram extraídos atributos com base nas tabelas do banco de dados relacional (por cliente). A extração dos atributos foi realizada para cada serviço separadamente. Aqui, os atributos foram classificados em quatro categorias: (i) atributos monetários, (ii) atributos de atividade, (iii) atributos de recência e (iv) atributos de frequência. A Tabela 4.1 descreve os atributos utilizados neste trabalho.

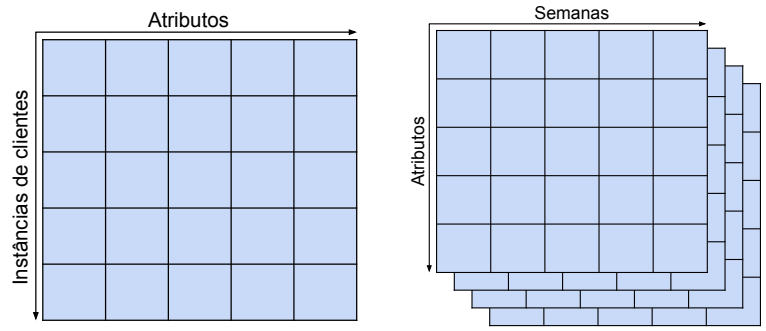
Os seguintes serviços financeiros disponibilizados pela empresa foram selecionados: conta paga, cobranças, Pix enviado e recebido, Transferência Eletrônica Disponível (TED) enviado e recebido.

Tabela 4.1: descrição dos atributos extraídos por categoria.

Categoria	Atributos extraídos
Monetários	Valor máximo, médio e mínimo de cada serviço utilizado pelo cliente
Atividade	Total de transações de cada serviço utilizado pelo cliente, total de cobranças com API, total de cobranças com carnê, total de cobranças com boleto, total de cobranças pagas, total de cobranças não pagas, total de cobranças com cartão e total de cobranças canceladas
Recência	Os dias desde a última transação de cada serviço utilizado pelo cliente
Frequência	O máximo, a média e o mínimo de dias entre transações subsequentes de cada serviço realizado pelo cliente

Os atributos foram gerados de forma sequencial, semana-a-semana. Tem-se, então, versões de um mesmo atributo extraído em períodos diferentes. Um conjunto de dados tabular foi gerado, orientado por cliente, com total de 3757634 instâncias de clientes com 62 tipos de atributos, cada um com 26 versões, totalizando 1612 atributos de histórico de transações (Representado na Figura 4.3a).

A partir do conjunto de dados tabular, foi gerado um conjunto de dados sequencial



(a) Tabular (b) Sequencial

Figura 4.3: Na Figura (a) é apresentada a versão tabular do conjunto de dados e na Figura (b) a versão sequencial. Fonte: O Autor.

para se adequar a entrada de modelos de especializados em modelagem de sequência. Para cada atributo, as 26 versões foram ordenadas para gerar uma lista que representa o atributo. E então, cada instância de cliente é representado por um conjunto de atributos sequenciados que pode ser representado por uma matriz (Ver Figura 4.3b). O conjunto de dados completo é um *tensor* de formato $(batch\_size, steps, input\_dim)$, em que $batch\_size$ é o tamanho da amostra de instâncias, $steps$ é o número de passos (semanas) e $input\_dim$ é o número de atributos (ou canais).

4.2 Rotulagem dos Dados

Uma boa abordagem para previsão de desligamento de clientes é aprender os padrões de comportamento do cliente (transações realizadas) a partir dos dados históricos. Cada cliente possui padrões próprios de uso dos serviços. Neste sentido, definimos o engajamento de cada cliente baseado na redução da frequência de uso dos serviços.

A partir dos registros das últimas 26 semanas (ver Figura 4.1), definimos um limiar para cada cliente, em cada serviço, como sendo o percentil 60 dos dias entre suas transações. Neste trabalho definimos o percentil 60% empiricamente como um valor acima da mediana (uma redução significativa da frequência de uso), pensado na classificação do cliente desligado antes de um comportamento de desengajamento explícito. Formalmente definimos o engajamento E do cliente C_i no serviço s_j, como:

$$E(C_i, s_j) = \begin{cases} 1, & \text{se (dias desde a última transação)} < \text{(limiar)} \\ 0, & \text{caso contrário} \end{cases} \tag{4.1}$$

Se $E(C_i, s_j) = 1$, o cliente C_i usa o serviço s_j como de costume, ou seja, definindo o cliente C_i engajado no serviço s_j. Por outro lado, se $E(C_i, s_j) = 0$, o cliente C_i não usa o serviço s_j como de costume, definindo o cliente C_i desengajado no serviço s_j.

Formalmente, a rotulagem proposta neste trabalho é definida como:

$$R(C_i) = \begin{cases} 1, & \text{se } \sum_{j=1}^{n} E(C_i, s_j) = 0 \\ 0, & \text{se } \sum_{j=1}^{n} E(C_i, s_j) > 0 \end{cases} \tag{4.2}$$

em que n é o número total de serviços selecionados para a base de dados. Assim, se $R(C_i) = 0$ o cliente C_i é rotulado como não desligado, ou seja, está engajado em pelo menos um serviço, e se $R(C_i) = 1$ o cliente está desengajado em todos os serviços e rotulado como desligado.

A intenção dessa abordagem é prever se um cliente vai reduzir o uso dos serviços mesmo antes de apresentar um comportamento de desligamento explícito. Um total de 82645 instâncias clientes desligados foram encontrados na rotulagem.

Capítulo 5

Metodologia

Para a resolução do problema apresentado, foi utilizado o seguinte fluxo metodológico apresentado na Figura 5.1.

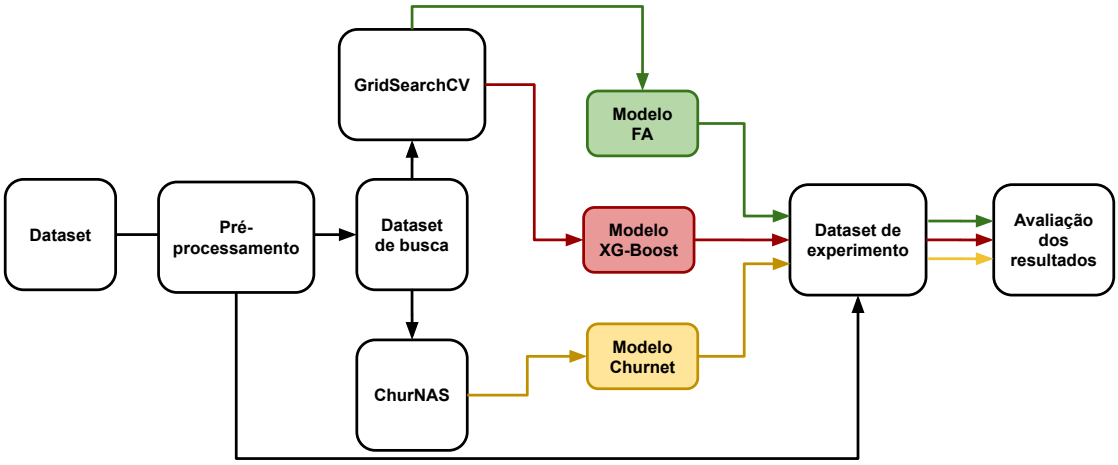


Figura 5.1: Fluxo metodológico utilizado neste trabalho. Fonte: O Autor.

5.1 Abordagem Proposta (ChurNAS)

Propomos neste trabalho, o ChurNAS, uma abordagem de busca de arquitetura neurais usando algoritmo genético para gerar automaticamente um conjunto de arquiteturas que otimizam desempenho na tarefa de predição de desligamento de clientes. O restante desta seção descreve o espaço de busca, o esquema de codificação e decodificação das arquiteturas e os principais componentes do algoritmo em detalhes.

5.1.1 Configuração do AG

Neste trabalho, utilizamos o algoritmo genético [20] para executar a busca de arquitetura de redes neurais para a predição de desligamentos de clientes. Foi

usado a biblioteca Python DEAP (*Distributed Evolutionary Algorithms in Python*) como base para a abordagem proposta. O Algoritmo 1 apresenta um resumo do processo.

O operador de seleção por torneio (do inglês *Tournament Selection*), foi escolhida como método de seleção neste trabalho. Este operador é largamente utilizado por ser simples e eficaz para manter a diversidade na população, enquanto seleciona os melhores indivíduos para reprodução. O processo de seleção por torneio consiste em selecionar aleatoriamente um subconjunto de indivíduos da população e, em seguida, escolher o melhor indivíduo deste subconjunto [18]. A probabilidade de um indivíduo ser selecionado é dada pela seguinte equação:

$$P_i = \frac{s_i}{\sum_{j=1}^{n} s_j}, \tag{5.1}$$

em que P_i é a probabilidade de o indivíduo i ser selecionado, s_i é a pontuação de aptidão/avaliação do indivíduo i e n é o tamanho do torneio.

O operador de cruzamento binário simulado (do inglês *simulated binary*) foi utilizado neste trabalho. Nesse método, dois indivíduos da população pai são selecionados x_1 e x_2. Em seguida, um número aleatório entre 0 e 1, u, é gerado para criar o parâmetro β da seguinte maneira:

$$\beta = \begin{cases} (2u)^{\frac{1}{n+1}}, & \text{se } u \leq 0.5; \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & \text{caso contrário.} \end{cases} \tag{5.2}$$

em que o índice de distribuição n é qualquer número real não negativo. Para grandes valores de n, a probabilidade de criar indivíduos próximos dos pais é maior. Quando n tem valores menores, indivíduos distantes são selecionadas como filhos. E então, o seguinte procedimento é seguido para criar os dois novos indivíduos $x_i^{1,t+1}$ e $x_i^{2,t+1}$ a partir de dois indivíduos pais $x_i^{1,t}$ e $x_i^{2,t}$

$$x_i^{1,t+1} = 0.5[(1+\beta)x_i^{1,t} + (1-\beta)x_i^{2,t}], \tag{5.3}$$

$$x_i^{2,t+1} = 0.5[(1-\beta)x_i^{1,t} + (1+\beta)x_i^{2,t}]. \tag{5.4}$$

O operador de mutação polinomial, também conhecida como *Polynomial Mutation*, foi utilizada neste trabalho. Esse operador, pode ser descrito da seguinte forma:

$$y_i^{1,t+1} = x_i^{1,t+1} + (x_i^u - x_i^l)\delta_i, \tag{5.5}$$

em que, $y_i^{1,t+1}$ é o indivíduo filho e $x_i^{1,t+1}$ é o indivíduo pai, (x_i^u e x_i^l são, respectivamente, os limites superior e inferior do componente pai, e o parâmetro δ_i é calculado

a partir da distribuição de probabilidade polinomial da seguinte forma:

$$\delta = \begin{cases} (2u)^{\frac{1}{n_m+1}}, & \text{se } u \leq 0.5; \\ 1 - [2(1-u)]^{\frac{1}{n_m+1}}, & \text{caso contrário.} \end{cases} \tag{5.6}$$

em que u é um número aleatório uniforme entre (0, 1) e o índice de distribuição de mutação n_m é qualquer número real não negativo.

Algorithm 1 *AG proposto*

```
população ← (população inicial);
AVALIAÇÃO (população);
hallda fama ← (Melhor indivíduo);
for gen ← 1 to ngen do
    Pais ← SELEÇÃOPORTORNEIO(população);
    Filhos ← VARIAÇÃO(Pais);
    AVALIAÇÃO(filhos);
    população ← SELEÇÃODOSMELHORES(população + filhos);
    hallda fama ← (Melhor indivíduo);
end for
return população, hallda fama
```

5.1.2 Espaço de busca adotado

O espaço de busca adotado investiga encontrar uma arquitetura baseada em três tipos de blocos de redes neurais: TCNN, CNN e LSTM. Adicionalmente outros mecanismos e operações auxiliares são implementos como: *channel attention* [58], *global avarage pooling* (GAP), *flatten* e *concatenate*. Ainda, a busca contempla um bloco de rede neural totalmente conectada (*Dense Neural Network* - DNN). No entanto, a única camada estritamente obrigatória é a camada final DNN com a função de ativação *sigmoid* para gerar a predição.

A Figura 5.2 apresenta uma representação visual do espaço de busca. Neste fluxograma, um indivíduo é decodificado em uma arquitetura. O processo começa com a decisão de adicionar ou não um mecanismo de atenção de canal para o primeiro caminho. Em seguida, há três opções disponíveis: utilizar um bloco TCNN, um bloco CNN ou nenhuma das opções. As operações GAP e *flatten* são obrigatórias somente um bloco CNN tiver sido adicionado. No segundo caminho, que é realizado em paralelo ao primeiro, é decidido se será implementado um mecanismo de canal e um bloco LSTM. Após o bloco LSTM, as operações GAP e *flatten* podem ser aplicadas. Essas operações são obrigatórias se for necessário padronizar a saída do bloco LSTM com a do bloco TCNN ou do CNN. Se houver dois modelos em paralelo (TCNN e LSTM ou CNN e LSTM), a operação *concatenate* é acionada para juntar

as saídas. Em seguida, é possível adicionar um bloco DNN. Por fim, é adicionada uma camada DNN + *sigmoid* para finalizar o classificador.

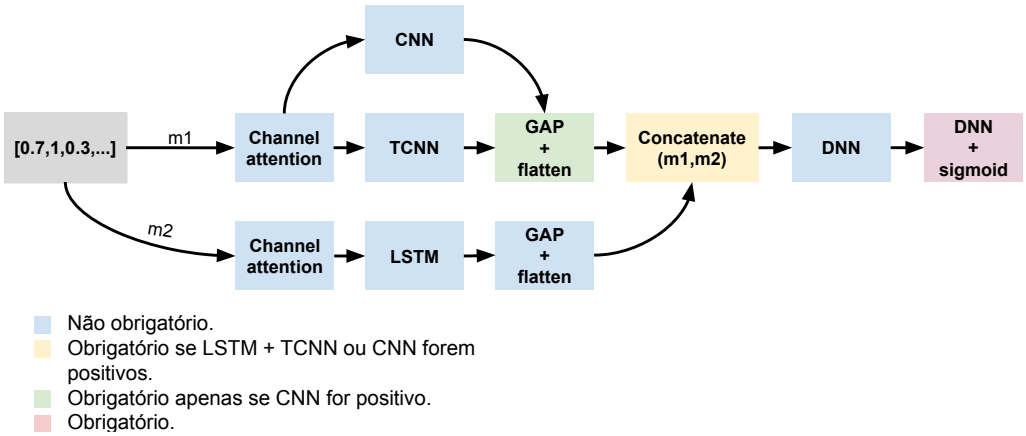


Figura 5.2: Fluxograma do espaço de busca. Fonte: O Autor.

No presente trabalho, os blocos são compostos por células (ou sub-blocos), que por sua vez são formadas por camadas e operações executadas em sequência. A Tabela 5.1 apresenta as células e seus respectivos hiperparâmetros e operações utilizadas. Os mecanismos de atenção de canal e a camada de classificação binária final DNN não possuem hiperparâmetros variáveis. Alguns hiperparâmetros de blocos e operações podem ser variados, e as possibilidades são mostradas na Tabela 5.2. Adicionalmente, o otimizador (*optmizer*) foi variado a partir das seguintes opções: *adam*, *rmsprop*, *adagrad*.

Tabela 5.1: células e seus respectivos hiperparâmetros e operações.

Célula	hiperparâmetros e operações
TCNN	*normalization, nb_filters, kernel_size, activation, skip_connection, return_sequences, dropout, dilations e kernel_initializer.*
CNN	*filters, kernel_size, batch_norm, activation, dropout, max_pooling_activation e max_pooling_size.*
DNN	*units, batch_norm, activation e dropout.*
LSTM	*units, return_sequences e dropout.*

5.1.3 Codificação das arquiteturas

Neste trabalho, além de buscar a arquitetura, visa-se ajustar os hiperparâmetros e o otimizador do modelo. Com isso, as arquiteturas são codificadas para representar a existência ou não de mecanismos de atenção (*channel attention*) e blocos, bem como os hiperparâmetros e o otimizador.

Tabela 5.2: Variações de hiperparâmetros.

hiperparâmetros	Variações
activation (TCNN e CNN)	*relu e gelu.*
activation (DNN)	*relu, gelu e elu.*
dilations (TCNN)	[1,2,4,8,16] e [1,2,4,8,16,32]
kernel_initializer (TCNN)	*glorot_uniform e GlorotNormal.*
normalization (TCNN)	*batch_norm, weight_norm e layer_norm.*
nb_filters (TCNN)	4,5,6,...,64.
skip_connection (TCNN)	boleano: 0,1.
return_sequences (TCNN e LSTM)	boleano: 0,1.
batch_norm (DNN)	boleano: 0,1.
units/filters (CNN e DNN)	4,5,6,...,512.
kernel_size (CNN e TCNN)	1,2,3,4,5.
pool_size (MaxPooling1D)	2,3,4,..,11.
units (LSTM)	4,5,6,...,64.
dropout_rate (TCNN, CNN, DNN, LSTM)	0.0,0.1,0.2,...,0.7.

Inicialmente, uma visão mais abrangente é apresentada na Figura 5.3, que representa uma arquitetura codificada. Na figura, é possível observar que o indivíduo (arquitetura codificada) é composto por dois mecanismos de atenção, quatro blocos de redes neurais e um otimizador. No indivíduo, os mecanismos de atenção são representados por um parâmetro que chamaremos de *Active*, que indica se o mecanismo de atenção será acionado ou não. Já o otimizador é obrigatório e é representado por um único elemento que indica qual otimizador deve ser utilizado. Quanto aos blocos, estes são representados por células. Cada célula é composta por um indicador de existência da célula (*Active*), seguido dos hiperparâmetros referentes a célula definidos na Seção 5.1.2, Tabela 5.1. Para uma melhor visualização da codificação, a Figura 5.4 foi gerada e apresenta um bloco DNN com uma e duas células.

Um indivíduo no AG é representado por um vetor $a = [x_1, , x_2, \ldots, x_n]$ que codifica uma arquitetura. Os n elementos x_i de um indivíduo a são valores reais no intervalo [0,1]. Neste trabalho, um indivíduo possui dois elementos para cada mecanismo de atenção (um para cada mecanismo), dez para o bloco TCNN (indicador *Active* + nove hiperparâmetros), 32 para o bloco CNN ((indicador *Active* + sete hiperparâmetros) multiplicados por quatro células), quatro para o bloco LSTM (indicador *Active* + três hiperparâmetros), dez para o bloco DNN ((indicador *Active* + quatro hiperparâmetros) multiplicados por duas células), e um para o otimizador, totalizando 58 elementos.

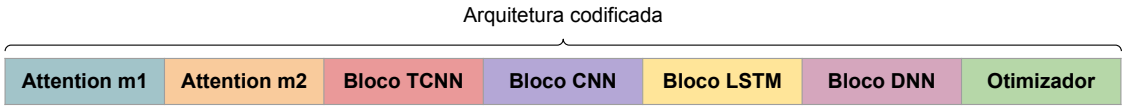


Figura 5.3: Representação da visão macro de uma arquitetura codificada. Na figura, atenção m1 é o mecanismo *channel attention* a ser aplicado antes do bloco TCNN ou CNN e atenção m2 é o mecanismo de atenção a ser aplicado antes do bloco LSTM. Fonte: O Autor.

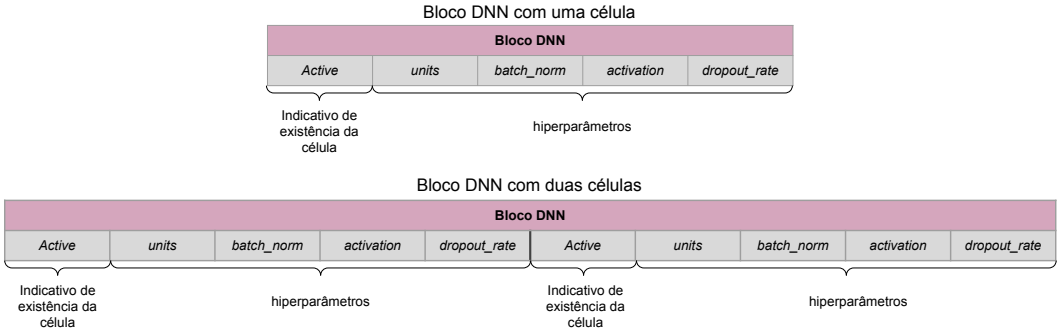


Figura 5.4: Representação da bloco DNN com uma e duas células. Fonte: O Autor.

5.1.4 Decodificação das arquiteturas

Para avaliar as arquiteturas, é necessário decodificar os indivíduos em modelos de redes neurais. Para isso, os elementos x_i do indivíduo a precisam ser decodificados em valores que representam informações sobre a arquitetura. No entanto, os elementos x_i representam parâmetros da arquitetura que variam de maneiras diferentes. Com base nessa variação, duas abordagens foram aplicadas para decodificação de x_i.

A primeira abordagem consiste em lidar com parâmetros booleanos, como o indicador *Active* e o hiperparâmetro *batch_norm* do bloco DNN. Nessa abordagem, os elementos x_i são arredondados para valores 0 ou 1, utilizando a função *round*. Dessa forma, $y_i = 0$ se $x_i < 0.5$ e $y_i = 1$ se $x_i \geq 0.5$.

Já a segunda abordagem é utilizada para lidar com parâmetros numéricos, como o hiperparâmetro número de filtros ($nb\_filters$) em cada camada de convolução do TCNN, e parâmetros categóricos, como as funções de ativação (*activation*). Para os parâmetros numéricos, os elementos x_i são escalados para o intervalo apropriado por meio de uma função de mapeamento que denominamos *map*. Por exemplo, o valor de *dropout_rate* é escalado para um valor de y_i no intervalo apropriado para a rede neural em questão, como $[0, 0.7]$. Já para os parâmetros categóricos, como a *activation*, o processo é semelhante, em que valores próximos de 0 representam índices mais à esquerda na lista de opções e valores próximos de 1 representam opções mais à direita na lista. A Figura 5.5 apresenta um exemplo de decodificação dos elementos para um bloco DNN.

Além disso, para uma compreensão mais abrangente dos processos de codificação e decodificação das arquiteturas, foram criadas duas imagens ilustrativas. A Figura 5.6 mostra a representação de um indivíduo codificado, enquanto a Figura 5.7 apresenta a arquitetura correspondente.

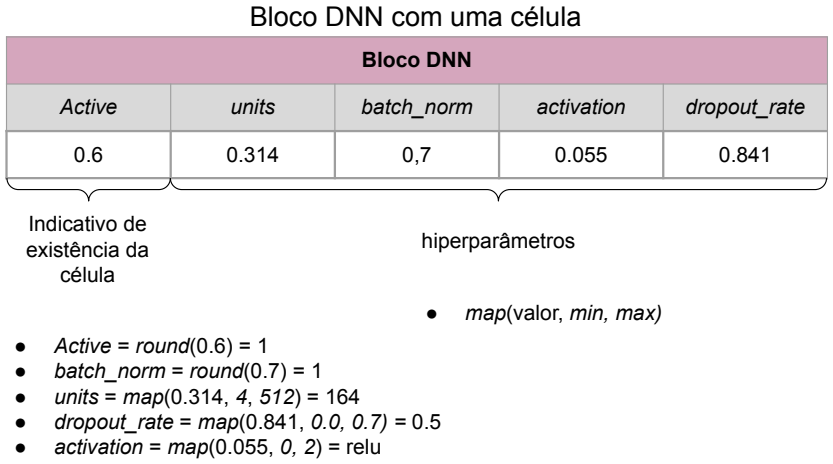


Figura 5.5: Exemplo decodificação de elementos em um bloco DNN. Fonte: O Autor.

5.2 Metodologia Experimental

Nesta seção é descrito a implementação dos modelos, os passos de preparação realizados no conjunto de dados, a estratégia de balanceamento dos dados, as configurações da busca de arquitetura neural e otimização de hiperparâmetros e as métricas de avaliações aplicadas neste trabalho.

5.2.1 Implementação dos Modelos

Os modelos XG-Boost e Floresta Aleatória, considerados estado-da-arte para problemas de predição com dados tabulares, foram selecionados para a comparação com o modelo desenvolvidos aqui neste trabalho. A linguagem de programação *Python* foi utilizada para o desenvolvimento dos algoritmos. Os modelos de redes neurais artificiais foram construídos a partir da API *Keras* da biblioteca *Tensorflow* a e biblioteca *Keras-TCN* [46]. Os modelos Floresta Aleatória foram implementados a partir da biblioteca de aprendizado de máquina *scikit-learn* [43] pelo algoritmo *ensemble RandomForestClassifier*. Já os modelos *XG-Boost* pelo algoritmo *XGBClassifier* da biblioteca *XGBoost*[11] (com *objective binary:logistic* e *eval_metric logloss*).

Atenção m1	Atenção m2
Active	*Active*
0.2	0.16

Bloco TCNN									
Active	*Normalization*	*nb_filters*	*kernel_size*	*activation*	*use_skip_connections*	*return_sequences*	*dropout_rate*	*dilations*	*kernel_initializer*
0.75	0.8	0.48	0.1	0.75	1.0	0.0	0.857	0.5	0.0

Bloco CNN	
Active	...
0.2	...

Bloco LSTM	
Active	...
0.15	...

Bloco DNN									
Active	*units*	*batch_norm*	*activation*	*dropout_rate*	*Active*	*units*	*batch_norm*	*activation*	*dropout_rate*
0.6	0.314	0.7	0.055	0.841	0.88	0.861	1.0	0.14	0.502

Otimizador
0.197

Figura 5.6: Exemplo de indivíduo codificado. Na figura, para uma melhor visualização, os hiperparâmetros dos blocos não ativados pelo indivíduo (CNN e LSTM) foram omitidos. Fonte: O Autor.

5.2.2 Pré-processamento dos dados

O conjunto de dados passou por um processo de pré-processamento. Os valores ausentes foram substituídos por zero, para atributos de atividade e monetários quando o cliente não realizou nenhuma transação serviço naquela semana. Para os atributos de frequência, os valores ausentes foram representados pelo pior caso da janela (sete dias) quando nenhuma transação do serviço foi encontrada naquela semana. Já para os atributos de recência, os valores ausentes da semana mais distante foram substituídos pelo pior caso da janela e o restante foi calculado recursivamente dado o valor da semana anterior mais o pior caso. Os clientes que não usaram nenhum serviço durante o período de observação da rotulagem foram removidos considerados clientes fantasmas dada a abordagem baseada em recência e frequência. Por fim, os dados foram normalizados entre 0 e 1.

5.2.3 Estratégia para contornar o desbalanceamento dos conjuntos de dados

O conjuntos de dados apresentou um alto desbalanceamento das classes, o que é natural. A discrepância na distribuição das classes pode impactar o desempenho

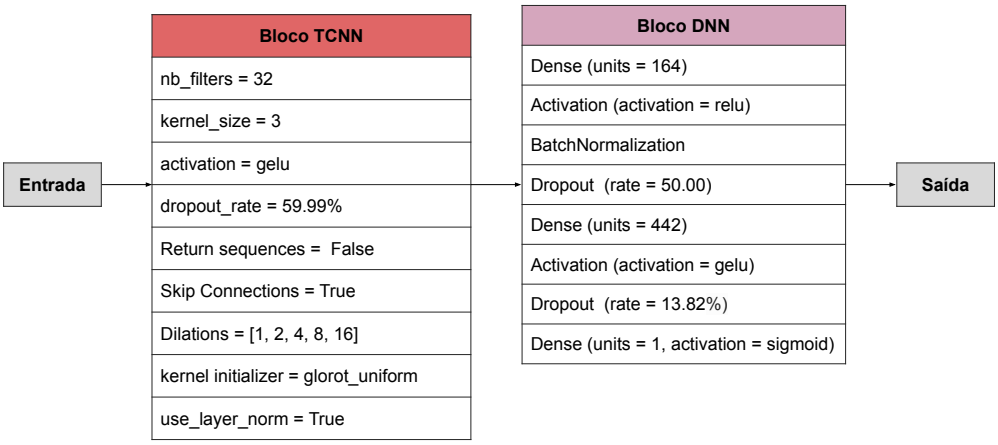


Figura 5.7: Exemplo arquitetura correspondente ao indivíduo apresentado na Figura 5.6. Fonte: O Autor.

dos modelos de machine learning, uma vez que eles têm a tendência de adquirir um maior conhecimento sobre as classes majoritárias, encontrando dificuldades para lidar com as classes minoritárias. Para contornar o desbalanceamento, uma combinações de técnicas foi aplicada para a partição de treinamento dos dados: subamostragem aleatória de 50% das instâncias da classe majoritária, combinada com compensação por meio da técnica *compute_sample_weight* do *scikit-learn* que usa os valores dos rótulos para ajustar pesos inversamente proporcionais às frequências das classes nos dados. No *dataset*, a classe 0 contou com 853683 instâncias enquanto a classe 1 com 82645 instâncias. Após o pré-processamento e subamostragem, o conjunto de dados completo (*dataset* de experimento) contém 936328 instâncias em que 853683 são instâncias de clientes não desligados e 82645 são instâncias de clientes desligados.

5.2.4 Particionamento do conjunto de dados para NAS e HPO

Com o objetivo de investigar a influência da quantidade de dados na busca de arquitetura e otimização de hiperparâmetros, foram extraídas diferentes partições do *dataset* de experimento, denominadas *datasets* de busca. As porcentagens de dados extraídas foram 10%, 20%, 30% e 40%. Para criar os particionamentos, utilizou-se o método de divisão estratificada $StratifiedShuffleSplit$ da biblioteca *scikit-learn*, que realiza as divisões preservando a proporção de amostras para cada classe.

5.2.5 Métricas de desempenho

Métricas de avaliação utilizadas são:

Acurácia:

$$Acc = \frac{(VP + VN)}{VP + FN + VN + FP}$$

Sensibilidade:

$$Se = \frac{VP}{(VP + FN)}$$

Especificidade:

$$Es = \frac{VN}{(FP + VN)}$$

Precisão:

$$Pr = \frac{VP}{(VP + FP)}$$

Taxa de Falso Positivos:

$$TFP = \frac{FP}{VN + FP}$$

F-score:

$$F - score = 2 \times \frac{(Pr \times Se)}{(Pr + Se)}$$

em que FN o número de falsos negativos; FP o número de falsos positivos; VN o número de verdadeiros negativos; VP e; número de verdadeiros positivos.

5.2.6 Busca exaustiva de hiperparâmetros

Os hiperparâmetros dos modelos XG-Boost e Floresta Aleatória foram ajustados usando o algoritmo *GridSearchCV* da biblioteca *scikit-learn*. O objetivo da busca é de obter o maior *F-score* dispondo o *dataset* de busca rotulados, divididos em 80% treino e 20% teste em uma validação cruzada interna de 3 *k-folds*. As variações e os hiperparâmetros para o modelos XG-Boost e Floresta Aleatória são apresentadas na Tabela 5.3.

As configurações de hiperparâmetros e *f-Score* encontrados durante a avaliação, utilizando as técnicas XG-Boost e Floresta Aleatória para os diferentes particionamentos, estão apresentados nas Tabelas 5.4 e 5.5.

5.2.7 ChurNAS busca

No ChurNAS, a população inicial é gerada por amostragem aleatória uniforme, adicionando a arquitetura proposta em [2], bem como três variações da mesma. As probabilidades de cruzamento e mutação são definidas como 0.3 e 0.02, respectivamente. O objetivo da busca é maximizar o *F-Score*, e para isso, os indivíduos são decodificados em arquiteturas e avaliados por uma função de avaliação. A população tem tamanho 30 e o número máximo de gerações é 40. O algoritmo é executado a

Tabela 5.3: Variações de hiperparâmetros para busca exaustiva.

Algoritmo	hiperparâmetros	Variações
XG-Boost	*min_child_weight*	(1, 5, 10)
	gamma	(0.5, 1, 1.5, 2)
	subsample	(0.6, 0.8, 1.0)
	colsample_bytree	(0.6, 0.8, 1.0)
	max_depth	(2,4,6,8, 10)
Floresta Aleatória	*n_estimators*	(200,300,400,500)
	criterion	(*gini, entropy)*
	max_features	(*sqrt*,*log2*, 4,5,6,7,8)
	max_depth	(5,10,15,20,25)
	bootstrap	(*True*, *False*)

Tabela 5.4: Configurações de hiperparâmetros encontrados pelo *GridSearchCV* para a técnica *XG-Boost* nos diferentes particionamentos do conjunto de dados. Na tabela, o *F-Score* é a média da métrica para os *k-folds* nos dados de teste. Fonte: O Autor.

	colsample_bytree	*gamma*	*max_depth*	*min_child_weight*	*subsample*	*F-Score*
10%	0.8	1.5	6	10	1	35.64
20%	0.8	2	6	10	1	35.83
30%	0.8	1	6	10	1	35.92
40%	1	1	6	10	1	36.43

cada 10 gerações e, caso não haja melhora do *F-Score* máximo (considerando três dígitos), a execução é finalizada. Assim, o ChurNAS pode pesquisar um total de 1200 arquiteturas de rede.

Para avaliação, o *dataset* de busca foi dividido em 80% treino e 20% teste. Cada indivíduo é treinado por 50 épocas, *batch_size* de 64 e taxa de aprendizagem de 0.001. Após treinado, o modelo gera as predições para partição de teste dos dados e extrai o valor de *F-Score* que será retornado para o algoritmo genético.

O ChurNAS executou 40 gerações para as partições de 10% e 40% do conjunto de dados, e 20 gerações para a partição de 20%. Já para a partição de 30% do conjunto de dados, foram realizadas 30 gerações. As Figuras 5.8, 5.9, 5.10 e 5.11 apresentam as arquiteturas encontradas, os hiperparâmetros selecionados, o otimizador utilizado, o número de parâmetros treináveis dos modelos e o *f-Score* obtido durante a avaliação do ChurNAS.

Tabela 5.5: Configurações de hiperparâmetros encontrados pelo *GridSearchCV* para a técnica Floresta Aleatória nos diferentes particionamentos do conjunto de dados. Na tabela, o *F-Score* é a média da métrica para os *k-folds* nos dados de teste. Fonte: O Autor.

	bootstrap	*criterion*	*max_depth*	*max_features*	*n_estimators*	*F-Score*
10%	*False*	*entropy*	15	*log2*	500	34.01
20%	*True*	*gini*	15	*log2*	500	34.39
30%	*False*	*gini*	15	*sqrt*	300	34.09
40%	*False*	*gini*	20	*log2*	500	34.55

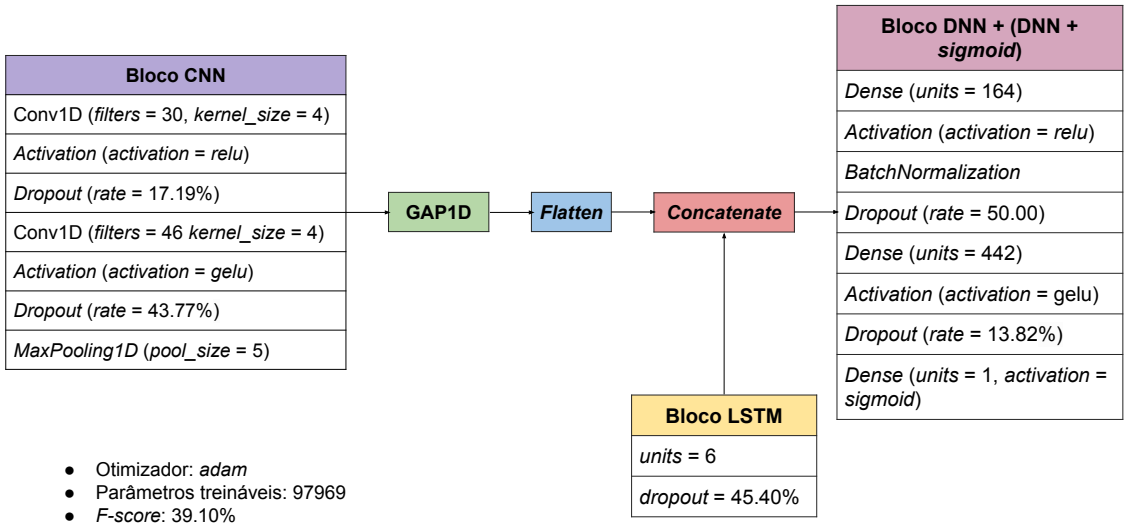


Figura 5.8: Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 10% dos dados. Fonte: O Autor.

5.2.8 Treinamento dos Modelos

Os modelos encontrados pelo ChurNAS e pelo *GridSearchCV* foram treinados usando um esquema de validação cruzada estratificada de 5 *k-folds*. Para cada *k-fold*, o conjunto de dados do experimento foi dividido em 80% para treinamento e 20% para teste. Para os modelos selecionados pelo ChurNAS, foram utilizadas 150 épocas, tamanho de lote (*batch_size*) de 64 e taxa de aprendizagem de 0,001. As métricas avaliadas para todos os modelos são a média das métricas obtidas em todos os *k-folds*.

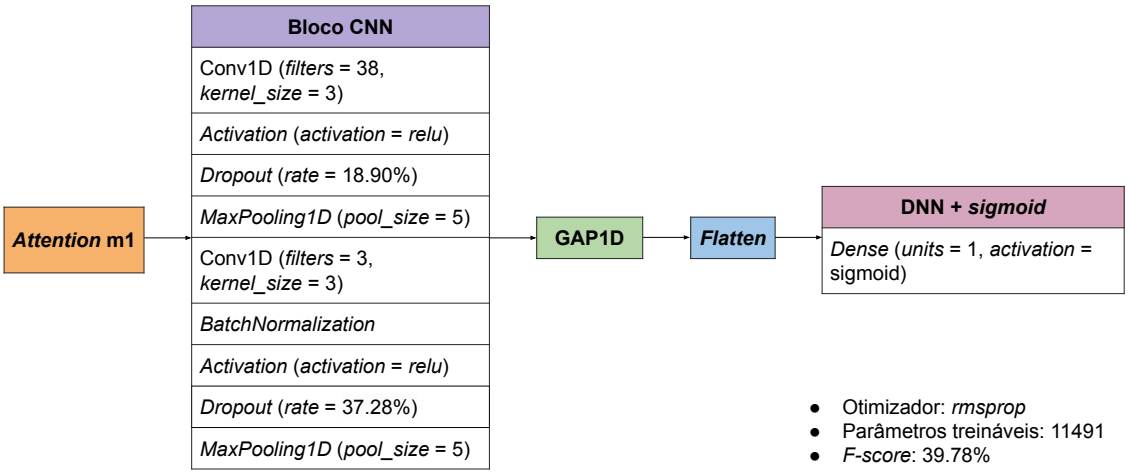


Figura 5.9: Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 20% dos dados. Fonte: O Autor.

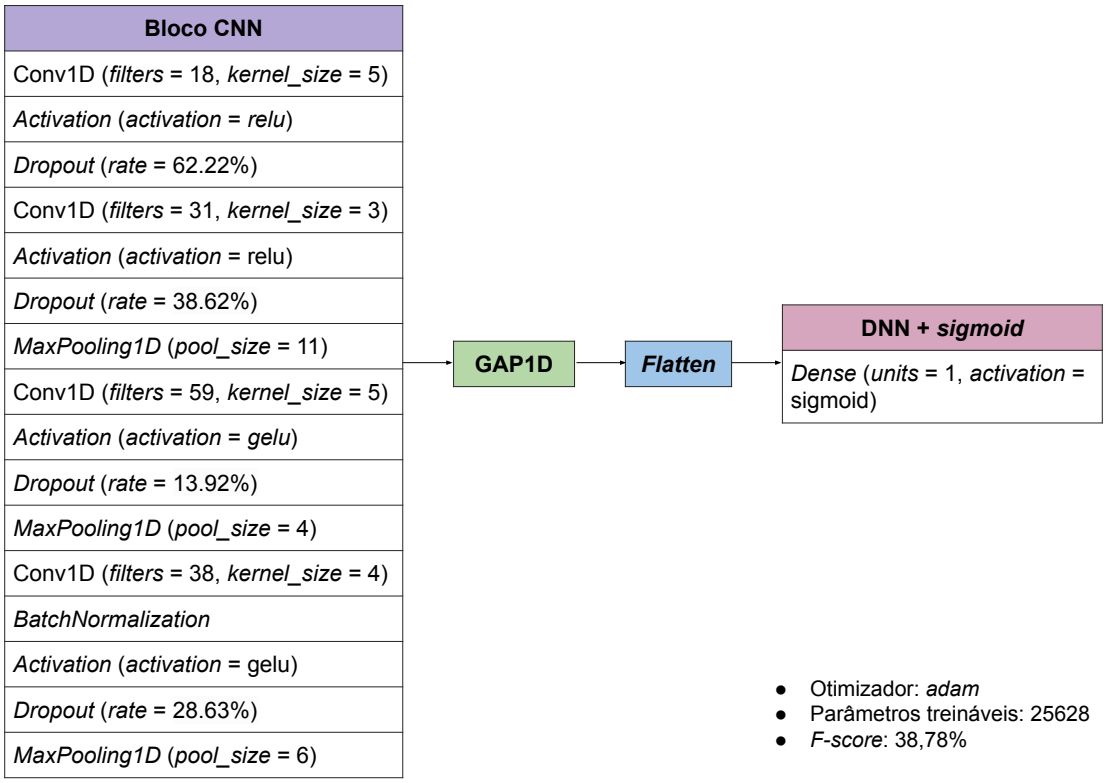


Figura 5.10: Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com o *dataset* de busca com 30% dos dados. Fonte: O Autor.

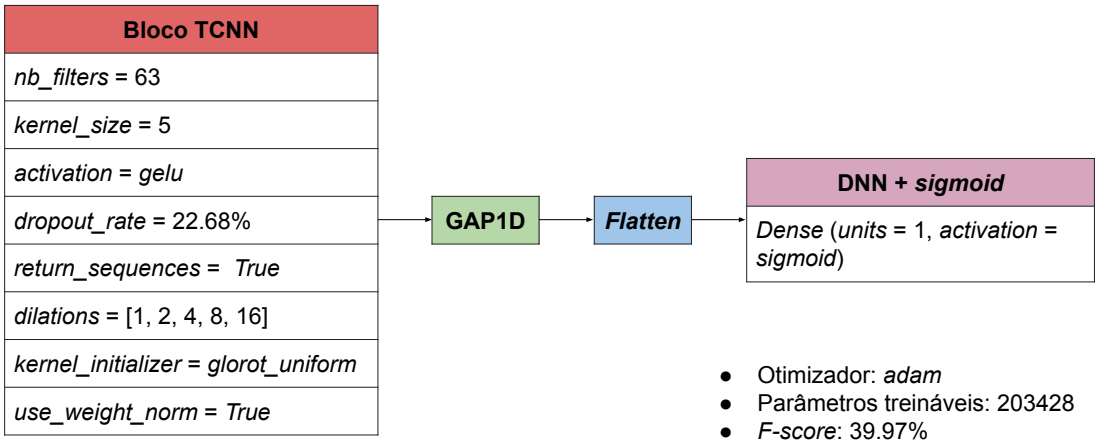


Figura 5.11: Representação da melhor arquitetura e hiperparâmetros encontrados pelo ChurNAS com *dataset* de busca com 40% dos dados. Fonte: O Autor.

Capítulo 6

Resultados, Discussões

As questões de pesquisa deste estudo são: (i) Os métodos de busca de arquitetura neural podem selecionar arquiteturas mais eficientes para modelos de aprendizagem profunda na previsão de desligamento de clientes no serviço financeiro? e (ii) A quantidade de dados utilizados na busca de arquitetura neural influencia na arquitetura selecionada e, consequentemente, no desempenho do modelo? O objetivo dos experimentos realizados é responder a essas perguntas. Para responder à primeira questão, foram gerados a Tabela 6.1 e as Figuras 6.1a, 6.1b e 6.3. Em relação à segunda questão, o gráfico representado na Figura 6.3 fornece informações fundamentais para a resposta.

Este capítulo está organizado da seguinte forma: a seção 6.1 apresenta os resultados do experimento; a seção 6.2 apresenta uma breve discussão dos resultados; e na seção 7.2 são apresentadas sugestões de pesquisas futuras em aberto.

6.1 Resultados

Com o objetivo de verificar o desempenho dos modelos no conjunto de dados completo (*dataset* de experimento), os modelos selecionados por meio da busca exaustiva de hiperparâmetros, conforme descrito na seção 5.2.6, e pela abordagem de busca de arquitetura neural ChurNAS, como descrito na seção 5.2.7, para cada particionamento definido na seção 5.2.4 treinados de acordo com a seção 5.2.8. Um sumário do resultado experimental para os modelos encontrados pode ser visto na Tabela 6.1.

No experimento realizado com uma partição de 10%, o modelo Churnet superou todos os outros classificadores em todas as métricas, com destaque para a sensibilidade, que atingiu 58,89%. Na partição de 20%, o modelo FA obteve destaque na métrica de sensibilidade, atingindo 56,84%, enquanto o modelo Churnet apresentou resultados superiores em todas as outras métricas. Já na partição de 30%, tem-se destaque para a técnica FA na métrica de sensibilidade, alcançando 57,58%, mas novamente o modelo Churnet apresentou resultados superiores nas demais métricas. Finalmente,

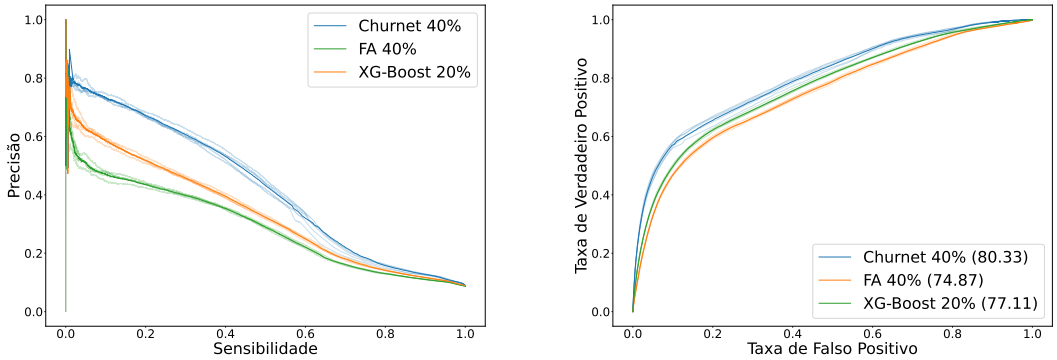
Tabela 6.1: Comparação entre médias das métricas e desvio padrão obtidos, pelos XG-Boost, FA (Floresta Aleatória) e Churnet (modelos encontrados pelo ChurNAS). As porcentagens indicadas em parênteses seguidas do nome do modelo indicam o particionamentos dos dados (porcentagem de dados do *dataset* de busca) utilizadas para realizar a NAS para o churnet e HPO para os outras técnicas.

Modelo	*Acc*	*Se*	*Es*	*TFP*	*Pr*	*F-Score*
FA (10%)	80.02(0.1)	57.81(0.36)	82.17(0.1)	17.83(0.1)	23.89(0.16)	33.81(0.22)
XG-Boost (10%)	82.53(0.15)	57.29(0.31)	84.97(0.16)	15.03(0.16)	26.95(0.24)	36.66(0.25)
Churnet (10%)	83.06(1.27)	**58.89**(2.19)	85.4(1.6)	14.6(1.6)	28.2(1.56)	38.08(0.93)
FA (20%)	80.56(0.1)	56.84(0.42)	82.86(0.14)	17.14(0.14)	24.3(0.11)	34.04(0.15)
XG-Boost (20%)	82.53(0.15)	57.31(0.34)	84.97(0.16)	15.03(0.16)	26.96(0.25)	36.67(0.2)
Churnet (20%)	84.33(1.14)	54.06(2.47)	87.26(1.48)	12.74(1.48)	29.23(1.42)	37.89(0.67)
FA (30%)	80.84(0.17)	57.58(0.36)	83.1(0.21)	16.9(0.21)	24.8(0.19)	34.67(0.4)
XG-Boost (30%)	82.51(0.01)	57.3(0.44)	84.95(0.13)	15.05(0.13)	26.93(0.16)	36.64(0.2)
Churnet (30%)	83.26(1.54)	55.68(3.07)	85.93(1.98)	14.07(1.98)	27.87(1.88)	37.05(0.97)
FA (40%)	82.49(0.15)	54.04(0.48)	85.24(0.16)	14.76(0.16)	26.17(0.27)	35.27(0.32)
XG-Boost (40%)	82.54(0.19)	57.09(0.44)	85.0(0.21)	15.0(0.21)	26.93(0.28)	36.6(0.3)
Churnet (40%)	**88.6**(0.63)	53.55(2.84)	**91.99**(0.95)	**8.01**(0.95)	**39.41**(1.72)	**45.33**(0.51)

na partição de 40%, o XG-Boost se destacou na métrica de sensibilidade, com 57,09%, mas o modelo Churnet apresentou métricas superiores de especificidade (91,99%), precisão (39,41%) e F-Score (45,33%), superando todos os outros modelos em todos os particionamentos.

O modelo encontrado pelo ChurNAS no particionamento 40% supera os FA e XG-Boost em 5 das 6 métricas extraídas considerando todas versões de modelos criadas por HPO. Considerando o melhor modelo de cada técnica, as curvas de *Precisão* $\times$ *Sensibilidade* foram geradas, e indicam o compromisso entre a precisão e a sensibilidade (ver Figura 6.1a). Adicionalmente, foram geradas as curvas ROC (*Receiver Operating Characteristic*), e apresenta a performance entre as taxas de verdadeiro positivo e falso positivo (ver Figura 6.1b). Em ambos os gráficos, o modelo Churnet sobrepõem os outros dois modelos na maioria dos limites.

Respondendo, portanto, à questão (i) foram geradas curvas de *F-Score* para os modelos encontrados pelo ChurNAS e HPO em diferentes conjuntos de dados (ver Figura 6.2). É importante notar que os modelos encontrados pelo ChurNAS apresentaram um desempenho superior às técnicas tradicionais ajustadas pelo HPO. Isso foi confirmado pelo fato de que as curvas dos modelos ChurNAS se sobrepuseram às curvas do XG-Boost e FA em todos os particionamentos do conjuntos de dados. O resultado obtido sugere que as técnicas NAS são eficientes na seleção de arquiteturas adequadas para modelos de aprendizagem profunda na previsão de desligamento de clientes no setor financeiro. Além disso, a abordagem genética para NAS parece ser um caminho promissor para resolver o problema de identificação de clientes propensos a deixar a empresa no setor financeiro.



(a) Curva de Precisão Sensibilidade.

(b) Curva ROC.

Figura 6.1: Nos gráficos são apresentados as curvas para os 5 *k-folds* e em destaque a curva média para os três modelos. No gráfico (b) o valor entre parenteses representa a área sobre a curva ROC média do modelo. Fonte: O Autor.

6.2 Discussões

Para investigar com mais profundidade os modelos encontrados pelo ChurNAS nos diferentes particionamentos do conjunto de dados, criamos um gráfico de barras (ver Figura 6.3) que relaciona a quantidade de parâmetros treináveis com o desempenho dos modelos em termos de *F-Score*. No gráfico, é possível observar que o modelo que implementa um bloco TCNN, o Churnet (40%), possui o maior número de parâmetros treináveis, enquanto o modelo que implementa um bloco LSTM (Churnet 10%) tem um número consideravelmente menor de parâmetros treináveis.

O número de parâmetros treináveis dos modelos selecionados pelo ChurNAS, neste trabalho, está relacionado com o tamanho da amostra utilizada na busca por arquiteturas neurais. Na partição de 10%, a segunda arquitetura com maior capacidade de aprendizado foi selecionada. Para as partições de 20% e 30%, o algoritmo genético selecionou arquiteturas ainda menores. Esses modelos obtiveram um desempenho inferior ao encontrado na partição de 40% do conjunto de dados, indicando uma convergência para regiões sub-ótimas de arquiteturas. Isso pode ser atribuído ao aumento do tamanho da amostra de dados juntamente com a quantidade relativamente pequena de épocas aplicadas na função de avaliação, o que favorece modelos com menor número de parâmetros treináveis a chegar mais próximo do seu pico de aprendizado. No entanto, quando 40% do conjunto de dados foi utilizado na busca, as arquiteturas com menor número de parâmetros treináveis não foram capazes de superar as arquiteturas com maior capacidade de armazenamento de informações dos clientes. Isso destaca a influência da quantidade de dados na escolha da arquitetura e no desempenho do modelo, respondendo, assim, à questão de pesquisa (ii).

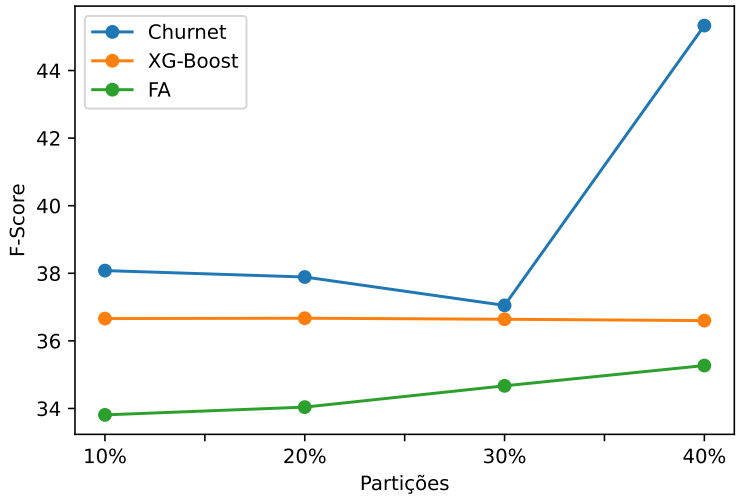


Figura 6.2: Curva de F-Score por modelo encontrados em diferentes particionamentos do conjunto de dados. Fonte: O Autor.

Em relação às arquiteturas, constatou-se que os modelos que incorporaram um bloco TCNN (Churnet (40%)) e LSTM (Churnet (10%)) alcançaram melhores resultados. É válido ressaltar que o modelo com a rede TCNN apresentou desempenho significativamente superior. Isso pode ser atribuído à utilização de camadas convolucionais para extrair características e à arquitetura em cascata, que permite o fluxo de informações entre as camadas nas redes TCNN. Por outro lado, a LSTM é mais adequada para modelar relações temporais complexas em sequências longas, o que não se aplica aos dados utilizados neste estudo. Uma possível explicação adicional está relacionada à capacidade dos modelos. Conforme evidenciado na Figura 6.3, o modelo com uma camada LSTM possui consideravelmente menos parâmetros treináveis em comparação com o modelo com bloco TCNN.

A abordagem proposta neste estudo apresenta algumas limitações que devem ser mencionadas. Embora os métodos de busca de arquitetura neural possam automatizar o processo, ainda existe um viés humano presente na definição manual do espaço de busca, o que pode limitar as possibilidades de arquiteturas representadas. Além disso, a definição empírica das configurações do espaço de busca, como a quantidade máxima de camadas em cada bloco e os hiperparâmetros selecionados e suas variações, pode afetar a escolha dos blocos e o desempenho do modelo. É importante lembrar que as arquiteturas disponíveis também são limitadas pela abordagem proposta neste estudo. Outra limitação a ser considerada é que o algoritmo ChurNAS foi projetado especificamente para conjuntos de dados sequenciais e, portanto, não é adequado para outras configurações de conjuntos de dados, como os tabulares, que são comuns em tarefas de predição de desligamento de clientes.

Por fim, é relevante destacar que o modelo que obteve os melhores resultados é

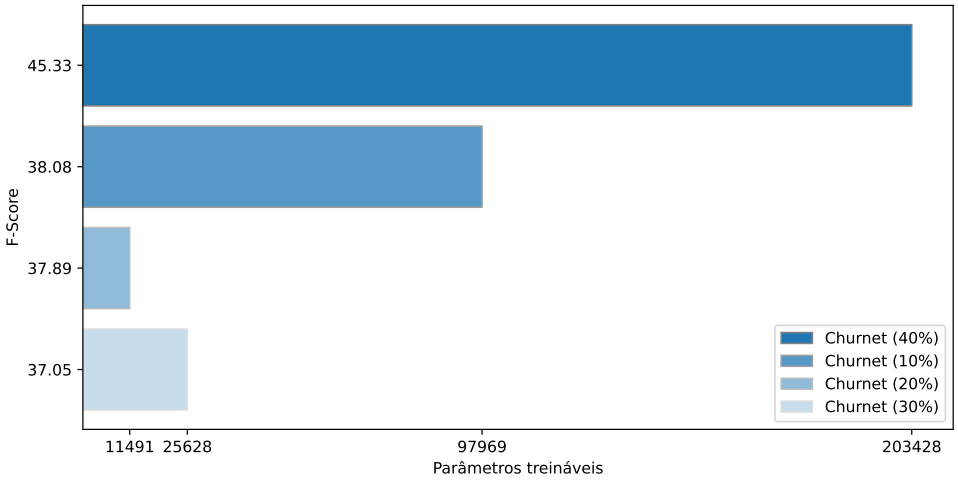


Figura 6.3: Gráfico de relação entre *F-Score* e parâmetros treináveis. Fonte: O Autor.

uma versão especializada de uma rede convolucional. As redes convolucionais são altamente vantajosas para a indústria devido à sua facilidade de paralelização e compatibilidade com aceleradores de hardware, como GPUs e TPUs. Além disso, a manutenção do modelo pode ser facilitada por meio de treinamento online em mini-lotes e transferência de aprendizagem. Isso possibilita a criação de um pipeline automatizado para a atualização constante dos modelos de forma fácil e eficiente.

Capítulo 7

Considerações Finais

7.1 Conclusões

No competitivo mercado das empresas financeiras, prever o desligamento de clientes é essencial para manter os mais valiosos, aumentar os lucros e fornecer serviços competitivos. Este estudo apresenta uma metodologia baseada em séries temporais que permite generalizar o comportamento do cliente em diferentes períodos e uma abordagem personalizada de rotulação que considera o padrão de uso de cada cliente, usando dados reais fornecidos por um provedor de serviços financeiros.

Este trabalho representa uma extensão da pesquisa anterior [2] conduzida pelos autores, na qual foi investigada uma abordagem temporal para a previsão de desligamento de clientes usando um conjunto de dados menos expressivo. Além disso, os autores utilizaram um modelo baseado em TCNN desenvolvido manualmente para gerar as predições e avaliar a metodologia.

No presente estudo, realizamos melhorias no conjunto de dados e na rotulagem com o objetivo de aprimorar a eficiência das previsões. Entretanto, essas alterações tornaram o problema consideravelmente mais complexo. Diante disso, optamos por substituir a busca por uma arquitetura neural manual por uma abordagem automatizada, a qual denominamos ChurNAS.

O ChurNAS é a principal contribuição deste estudo. O modelo Churnet gerado pelo ChurNAS teve um desempenho superior aos métodos tradicionais XG-Boost e Floresta Aleatória com ajuste de hiperparâmetros, com *F-Score* de 45,33%. Além disso, o estudo discute a influência da quantidade de dados e da capacidade dos modelos no desempenho dos mesmos.

O ChurNAS juntamente com a abordagem temporal de extração de atributos oferecem um caminho promissor para o gerenciamento de clientes desligados. Com essa metodologia implementada como uma ferramenta de auxílio na tomada de decisões, as empresas podem tomar medidas proativas para evitar o desligamento

de clientes valiosos e manter sua base de clientes de forma eficaz. Isso pode levar a uma maior satisfação do cliente, fidelização e, em última análise, a uma maior lucratividade para a empresa no futuro.

7.2 Trabalhos Futuros

Uma possível abordagem de trabalho futuro seria explorar mais as possibilidades das redes TCNN e LSTM, uma vez que as camadas neurais que consideram a causalidade dos dados obtiveram melhores resultados do que as redes CNN unidimensionais causais tradicionais. Isso poderia ajudar o modelo a convergir para regiões de redes mais eficientes e reduzir o custo computacional da busca.

Seria ainda mais interessante explorar informações cadastrais da conta do cliente, que poderiam ser utilizadas para gerar novos atributos que comporiam um conjunto de dados auxiliar. Por exemplo, é possível considerar o número total de usuários na conta, a data de aniversário, a profissão informada, se o cliente é estrangeiro, o DDD do celular e o tempo gasto em ligações de atendimento na instituição financeira. Dessa forma, seria possível ampliar o espaço de busca do ChurNAS para agregar camadas de redes paralelas especializadas em classificação de dados tabulares, como a TabNet [4], e combiná-las com as redes TCNN e/ou LSTM para gerar um modelo mais robusto e preciso.

Outro ponto relevante a ser investigado é a aplicação de métodos de Inteligência Artificial Explicável (Explainable Artificial Intelligence XAI) para aprimorar a compreensão dos modelos de aprendizado de máquina e tornar suas saídas mais interpretáveis. Os métodos de XAI possibilitam explicar o conhecimento adquirido pelo modelo e fornecer informações adicionais que podem ser relevantes para a tomada de decisão [47]. Em particular, tais informações podem ser úteis para campanhas de retenção de clientes propensos a desligar. Portanto, o estudo dos métodos de XAI pode agregar informações valiosas para o aprimoramento dessas campanhas.

Por fim, é importante também investigar o desempenho da metodologia utilizada em diferentes cenários, não se restringindo apenas ao problema de prever o desligamento de clientes e dados reais.

Referências Bibliográficas

[1] AHN, D., LEE, D., HOSANAGAR, K., 2020, "Interpretable Deep Learning Approach to Churn Management", *SSRN*, pp. 1–44.

[2] ALMEIDA, M., MOTA, M., SOUZA, W., et al., 2022, "A Temporal Approach to Customer Churn Prediction: A Case Study for Financial Services". In: *Anais do XIX Encontro Nacional de Inteligência Artificial e Computacional*, pp. 83–94.

[3] AMIN, A., KHAN, C., ALI, I., et al., 2014, "Customer Churn Prediction in Telecommunication Industry: With and without Counter-Example". In: *2014 European Network Intelligence Conference*, pp. 134–137. doi: 10.1109/ENIC.2014.29.

[4] ARIK, S. Ö., PFISTER, T., 2021, "Tabnet: Attentive interpretable tabular learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 35, pp. 6679–6687.

[5] BA, J. L., KIROS, J. R., HINTON, G. E., 2016, "Layer normalization", *arXiv preprint arXiv:1607.06450*.

[6] BAI, S., KOLTER, J. Z., KOLTUN, V., 2018, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", *arXiv:1803.01271*.

[7] BERGSTRA, J., YAMINS, D., COX, D., 2013, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures". In: *International conference on machine learning*, pp. 115–123. PMLR.

[8] BREIMAN, L., 2001, "Random forests", *Machine learning*, v. 45, n. 1, pp. 5–32.

[9] CAIGNY, A. D., COUSSEMENT, K., BOCK, K. W. D., et al., 2020, "Incorporating textual information in customer churn prediction models based on a convolutional neural network", *International Journal of Forecasting*, v. 36, n. 4, pp. 1563–1578. ISSN: 0169-2070. doi: https:

//doi.org/10.1016/j.ijforecast.2019.03.029. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169207019301499>.

[10] CHEN, T., GUESTRIN, C., 2016, "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, .

[11] CHEN, T., GUESTRIN, C., 2016, "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, . ACM. ISBN: 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. Disponível em: <http://doi.acm.org/10.1145/2939672.2939785>.

[12] CHEN, Z.-Y., FAN, Z.-P., SUN, M., 2012, "A hierarchical multiple kernel support vector machine for customer churn prediction using longitudinal behavioral data", *European Journal of Operational Research*, v. 223, n. 2, pp. 461–472. ISSN: 0377-2217. doi: https://doi.org/10.1016/j.ejor.2012.06.040. Disponível em: <https://www.sciencedirect.com/science/article/pii/S037722171200495X>.

[13] CLEVERT, D.-A., UNTERTHINER, T., HOCHREITER, S., 2015, "Fast and accurate deep network learning by exponential linear units (elus)", *arXiv preprint arXiv:1511.07289*.

[14] COUSSEMENT, K., DE BOCK, K. W., 2013, "Customer churn prediction in the online gambling industry: The beneficial effect of ensemble learning", *Journal of Business Research*, v. 66, n. 9, pp. 1629–1636. ISSN: 0148-2963. doi: https://doi.org/10.1016/j.jbusres.2012.12.008. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0148296312003530>.

[15] DEB, K., BEYER, H.-G., 2001, "Self-adaptive genetic algorithms with simulated binary crossover", *Evolutionary computation*, v. 9, n. 2, pp. 197–221.

[16] DEVRIENDT, F., BERREVOETS, J., VERBEKE, W., 2021, "Why you should stop predicting customer churn and start using uplift models", *Information Sciences*, v. 548, pp. 497–515. ISSN: 0020-0255. doi: https://doi.org/10.1016/j.ins.2019.12.075. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025519312022>.

[17] ELSKEN, T., METZEN, J. H., HUTTER, F., 2019, "Neural architecture search: A survey", *The Journal of Machine Learning Research*, v. 20, n. 1, pp. 1997–2017.

[18] FANG, Y., LI, J., 2010, "A review of tournament selection in genetic programming". In: *Advances in Computation and Intelligence: 5th International Symposium, ISICA 2010, Wuhan, China, October 22-24, 2010. Proceedings 5*, pp. 181–192. Springer.

[19] GITMAN, I., GINSBURG, B., 2017, "Comparison of batch normalization and weight normalization algorithms for the large-scale image classification", *arXiv preprint arXiv:1709.08145*.

[20] GOLDBERG, D. E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. USA, Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201157675.

[21] HADDEN, J., TIWARI, A., ROY, R., et al., 2007, "Computer assisted customer churn management: State-of-the-art and future trends", *Computers & Operations Research*, v. 34, n. 10, pp. 2902–2917.

[22] HADDEN, J., TIWARI, A., ROY, R., et al., 2007, "Computer assisted customer churn management: State-of-the-art and future trends", *Computers & Operations Research*, v. 34, n. 10, pp. 2902–2917. ISSN: 0305-0548. doi: https://doi.org/10.1016/j.cor.2005.11.007. Disponível em: <`https://www.sciencedirect.com/science/article/pii/S0305054805003503`>.

[23] HENDRYCKS, D., GIMPEL, K., 2016, "Gaussian error linear units (gelus)", *arXiv preprint arXiv:1606.08415*.

[24] HOCHREITER, S., SCHMIDHUBER, J., 1997, "Long short-term memory", *Neural computation*, v. 9, n. 8, pp. 1735–1780.

[25] IOFFE, S., SZEGEDY, C., 2015, "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*, pp. 448–456. pmlr.

[26] JAVAHERIPI, M., SHAH, S., MUKHERJEE, S., et al., 2022, "LiteTransformerSearch: Training-free On-device Search for Efficient Autoregressive Language Models", *arXiv preprint arXiv:2203.02094*.

[27] KILIÇARSLAN, S., KEMAL, A., ÇELIK, M., 2021, "An overview of the activation functions used in deep learning algorithms", *Journal of New Results in Science*, v. 10, n. 3, pp. 75–88.

[28] KIRANYAZ, S., INCE, T., GABBOUJ, M., 2015, "Real-time patient-specific ECG classification by 1-D convolutional neural networks", *IEEE Transactions on Biomedical Engineering*, v. 63, n. 3, pp. 664–675.

[29] KIRANYAZ, S., AVCI, O., ABDELJABER, O., et al., 2021, "1D convolutional neural networks and applications: A survey", *Mechanical systems and signal processing*, v. 151, pp. 107398.

[30] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., 2017, "Imagenet classification with deep convolutional neural networks", *Communications of the ACM*, v. 60, n. 6, pp. 84–90.

[31] LABACH, A., SALEHINEJAD, H., VALAEE, S., 2019, "Survey of dropout methods for deep neural networks", *arXiv preprint arXiv:1904.13310*.

[32] LECUNN, Y., BOTTOU, L., BENGIO, Y., et al., 1998, "c", *Proceedings of the IEEE*, v. 86, n. 11, pp. 2278–2324.

[33] LIMA, E., MUES, C., BAESENS, B., 2009, "Domain knowledge integration in data mining using decision tables: case studies in churn prediction", *Journal of the Operational Research Society*, v. 60, n. 8, pp. 1096–1106. doi: 10.1057/jors.2008.161.

[34] LIN, M., CHEN, Q., YAN, S., 2013, "Network in network", *arXiv preprint arXiv:1312.4400*.

[35] LIU, H., SIMONYAN, K., YANG, Y., 2018, "Darts: Differentiable architecture search", *arXiv preprint arXiv:1806.09055*.

[36] LU, Z., WHALEN, I., BODDETI, V., et al., 2019, "Nsga-net: neural architecture search using multi-objective genetic algorithm". In: *Proceedings of the genetic and evolutionary computation conference*, pp. 419–427.

[37] MENA, C. G., DE CAIGNY, A., COUSSEMENT, K., et al., 2019, "Churn prediction with sequential data and deep neural networks. a comparative analysis", *arXiv preprint arXiv:1909.11114*.

[38] MILLER, G. F., TODD, P. M., HEGDE, S. U., 1989, "Designing Neural Networks using Genetic Algorithms". In: *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 379–384.

[39] MUHURI, P. K., ASHRAF, Z., LOHANI, Q. D., 2017, "Multiobjective reliability redundancy allocation problem with interval type-2 fuzzy uncertainty", *IEEE Transactions on Fuzzy Systems*, v. 26, n. 3, pp. 1339–1355.

[40] NAIR, V., HINTON, G. E., 2010, "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

[41] NESLIN, S. A., GUPTA, S., KAMAKURA, W., et al., 2006, "Defection Detection: Measuring and Understanding the Predictive Accuracy of Customer Churn Models", *Journal of Marketing Research*, v. 43, n. 2, pp. 204–211. doi: 10.1509/jmkr.43.2.204.

[42] OLAH, C., 2018, "Understanding lstm networks, August 2015", *URL https://colah. github. io/posts/2015-08-Understanding-LSTMs*.

[43] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al., 2011, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, v. 12, pp. 2825–2830.

[44] RAWAT, W., WANG, Z., 2017, "Deep convolutional neural networks for image classification: A comprehensive review", *Neural computation*, v. 29, n. 9, pp. 2352–2449.

[45] REAL, E., AGGARWAL, A., HUANG, Y., et al., 2019, "Regularized evolution for image classifier architecture search". In: *Proceedings of the aaai conference on artificial intelligence*, v. 33, pp. 4780–4789.

[46] REMY, P., 2020. "Temporal Convolutional Networks for Keras". `https://github.com/philipperemy/keras-tcn`.

[47] ROJAT, T., PUGET, R., FILLIAT, D., et al., 2021, "Explainable artificial intelligence (xai) on timeseries data: A survey", *arXiv preprint arXiv:2104.00950*.

[48] SALIMANS, T., KINGMA, D. P., 2016, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks", *Advances in neural information processing systems*, v. 29.

[49] SANTURKAR, S., TSIPRAS, D., ILYAS, A., et al., 2018, "How does batch normalization help optimization?" *Advances in neural information processing systems*, v. 31.

[50] SHARMA, S., SHARMA, S., ATHAIYA, A., 2017, "Activation functions in neural networks", *Towards Data Sci*, v. 6, n. 12, pp. 310–316.

[51] SRINIVAS, M., PATNAIK, L. M., 1994, "Genetic algorithms: A survey", *computer*, v. 27, n. 6, pp. 17–26.

[52] SRINIVASAMURTHY, R. S., 2018, *Understanding 1D Convolutional Neural Networks Using Multiclass Time-Varying Signalss*. Tese de Doutorado, Clemson University.

[53] SUGANUMA, M., OZAY, M., OKATANI, T., 2018, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search". In: *International Conference on Machine Learning*, pp. 4771–4780. PMLR.

[54] TAN, M., CHEN, B., PANG, R., et al., 2019, "Mnasnet: Platform-aware neural architecture search for mobile". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828.

[55] TENORIO, M., LEE, W.-T., 1988, "Self organizing neural networks for the identification problem", *Advances in Neural Information Processing Systems*, v. 1.

[56] WHITE, C., SAFARI, M., SUKTHANKER, R., et al., 2023, "Neural Architecture Search: Insights from 1000 Papers", *arXiv preprint arXiv:2301.08727*.

[57] WISTUBA, M., RAWAT, A., PEDAPATI, T., 2019, "A survey on neural architecture search", *arXiv preprint arXiv:1905.01392*.

[58] WOO, S., PARK, J., LEE, J.-Y., et al., 2018. "CBAM: Convolutional Block Attention Module". Disponível em: <`https://arxiv.org/abs/1807.06521`>.

[59] XIE, L., YUILLE, A., 2017, "Genetic cnn". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388.

[60] YU, Y., SI, X., HU, C., et al., 2019, "A review of recurrent neural networks: LSTM cells and network architectures", *Neural computation*, v. 31, n. 7, pp. 1235–1270.

[61] ZHANG, L. M., 2015, "Genetic deep neural networks using different activation functions for financial data mining". In: *2015 IEEE International Conference on Big Data (Big Data)*, pp. 2849–2851. IEEE.

[62] ZOPH, B., LE, Q. V., 2016, "Neural architecture search with reinforcement learning", *arXiv preprint arXiv:1611.01578*.