UNIVERSIDADE FEDERAL DE PERNAMBUCO

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Eletrônica e Sistemas

Graduação em Engenharia Eletrônica

Gabriel Victor Marques de Oliveira Vital

Modelos de Aprendizado de Máquina para Estimar a Qualidade de Transmissão em Redes Ópticas

Recife

2024

Gabriel Victor Marques de Oliveira Vital

Modelos de Aprendizado de Máquina para Estimar a Qualidade de Transmissão em Redes Ópticas

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Orientador(a): Prof. Leonardo Didier Coelho, Dr.

Recife
2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Vital, Gabriel Victor Marques de Oliveira.
Modelos de aprendizado de máquina para estimar a qualidade de transmissão em redes ópticas / Gabriel Victor Marques de Oliveira Vital. - Recife, 2024.
157 : il., tab.

Orientador(a): Leonardo Didier Coelho
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2024.
Inclui referências, apêndices, anexos.

1. Comunicações ópticas. 2. Redes ópticas. 3. Aprendizado de máquina. 4. Modelo de ruído não linear. 5. Qualidade de transmissão. I. Coelho, Leonardo Didier. (Orientação). II. Título.

600 CDD (22.ed.)

Gabriel Victor Marques de Oliveira Vital

Modelos de Aprendizado de Máquina para Estimar a Qualidade de Transmissão em Redes Ópticas

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Aprovado em: 21/03/2024

Banca Examinadora

Documento assinado digitalmente
gov.br
LEONARDO DIDIER COELHO
Data: 06/07/2024 17:04:24-0300
Verifique em https://validar.iti.gov.br

Prof. Leonardo Didier Coelho, Dr.
Universidade Federal de Pernambuco

Documento assinado digitalmente
gov.br
JOAQUIM FERREIRA MARTINS FILHO
Data: 06/07/2024 17:14:55-0300
Verifique em https://validar.iti.gov.br

Prof. Joaquim F. Martins Filho, Ph.D.
Universidade Federal de Pernambuco

A todos aqueles que, mesmo diante de inúmeros desafios, não desistiram e persistem avançando.

Agradecimentos

A meus familiares e amigos mais próximos por todos os momentos de estudo, apoio, diversão e aprendizado que compartilhamos ao longo desses anos.

A jornada foi desafiadora, mas ter vocês ao meu lado tornou o caminho mais leve, interessante, suportável e feliz. Sempre guardarei esses momentos com carinho e com boas lembranças de um período em que aprendi lições para toda a vida.

Agradeço ao meu orientador, professor Dr. Leonardo Didier, por todo apoio e ajuda, pois seu vasto conhecimento transmitido com maestria foram fundamentais para o desenvolvimento deste trabalho e esta conquista.

Agradeço também a todos os professores e mestres dos quais fui aluno, pois tiveram uma contribuição relevante nesta jornada. Em especial, gostaria de agradecer aos professores Ricardo Campello, Gilson Jeronimo, Sidney Marlon, Hermano Cabral, João Marcelo, Guilherme Melo e Ricardo Bortolotti.

“O que sabemos é uma gota; o que ignoramos é um oceano.”

Isaac Newton

Resumo do Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica(Eng.)

Modelos de Aprendizado de Máquina para Estimar a Qualidade de Transmissão em Redes Ópticas

Gabriel Victor Marques de Oliveira Vital

Este trabalho se refere ao uso de modelos de aprendizado de máquina para estimar a qualidade de transmissão em redes ópticas. O trabalho aborda o uso de modelos e técnicas de aprendizado de máquina para estimar a qualidade de transmissão (QoT) em sistemas de comunicação por fibras ópticas. O objetivo é analisar a eficácia de três modelos de aprendizado de máquina quando usados para estimar a QoT de um caminho óptico ainda não estabelecido. Isso pode otimizar o provisionamento de caminhos ópticos, descartando conexões com baixa QoT e melhorando o desempenho geral da rede. O estudo aborda a complexidade das redes ópticas, incluindo a composição dos caminhos ópticos, com fibras ópticas, amplificadores, receptores ópticos, transmissores ópticos, multiplexação por divisão de comprimento de onda (WDM). Apesar dos desafios de modelar interferências não lineares, o uso de algoritmos de aprendizado de máquina apresentam-se como uma abordagem promissora para aprimorar a eficiência e confiabilidade das redes ópticas.

Palavras-chave: Comunicações ópticas; Redes ópticas; Aprendizado de máquina; Modelo de ruído não linear; Qualidade de transmissão.

Abstract of Course Conclusion Work, presented to Departament of Eletronic and Systems, as a partial fulfillment of the requirements for the degree of Bachelor of Electronic Engineering(Eng.)

Machine Learning Models for Estimating Transmission Quality in Optical Networks

Gabriel Victor Marques de Oliveira Vital

This work refers to the use of machine learning models to estimate the quality of transmission in optical networks. The work addresses the use of machine learning models and techniques to estimate the Quality of Transmission (QoT) in optical fiber communication systems. The goal is to analyze the effectiveness of three machine learning models when used to estimate the QoT of an as-yet-unestablished optical path. This can optimize the provisioning of optical paths by discarding connections with low QoT and improving the overall network performance. The study addresses the complexity of optical networks, including the composition of optical paths with optical fibers, amplifiers, optical receivers, optical transmitters, and wavelength division multiplexing (WDM). Despite the challenges of modeling nonlinear interference, the use of machine learning algorithms presents itself as a promising approach to enhance the efficiency and reliability of optical networks.

Keywords: Optical communications; Optical networks; Machine Learning; Nonlinear noise model; Transmission quality.

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Lista de Abreviações

QoT	*Quality of transmission*
OSNR	*Optical signal to noise ratio*
$\text{OSNR}_{\text{dB}_{\text{NL}}}$	*Optical signal to noise ratio with NLIN interference*
$\text{NLIN}/\text{NLIN}_{\text{Power}}$	*Power of nonlinear interference noise*
BER	*Bit error rate*
EDFA	*Erbium Doped Fiber Amplifier*
LED	*Light Emitting Diode*
QPSK	*Quadrature Phase Shift Keying*
QAM	*Quadrature Amplitude Modulation*
ASE	*Amplified Spontaneous Emission Noise*
SSMF	*Standard Single Mode Fiber*
NZDSF	*Non-zero dispersion-shifted fiber*
PSCF	*Pure Silica Core Fiber*
ML	*Machine Learning*
KNN	*K-Nearest Neighbors*
SVM	*Support Vector Machines*
SVR	*Support Vector Regression*
SVM-RBF	*Support Vector Regression with RBF kernel*
SVM-POLY	*Support Vector Regression with Polynomial kernel*
ANN	*Artificial Neural Networks*
UFPE	*Universidade Federal de Pernambuco*
R^2	*R-Squared*
MAE	*Mean Absolute Error*
MSE	*Mean Squared Error*
RMSE	*Root Mean Squared Error*

Lista de Símbolos

Símbolo	Parâmetro	Unidade
E	Campo Elétrico	V/m
H	Campo Magnético	T
n	Índice de refração	
λ	Comprimento de onda	nm
R^2	Coeficiente de determinação	
α	Coeficiente de atenuação	dB/km
D	Parametro de dispersão	ps/nm/km
ω	Frequência angular	rad/s
β''	Dispersão cromática de determinação	ps^2/km

Sumário

Capítulo 1

Introdução

A crescente complexidade das redes ópticas projetadas para atender a uma infinidade de serviços está gerando grandes quantidades de informação. Esses dados podem ser utilizados de forma inteligente para melhorar o desempenho da rede. Entre as diversas ferramentas matemáticas, o aprendizado de máquina é considerado como uma das abordagens mais promissoras para realizar a análise dos dados (Morais e Pedro, 2018). Em redes ópticas, o provisionamento de um caminho óptico exige o cálculo da qualidade de transmissão (Quality of Transmission - QoT), que pode ser determinada por métricas como a relação sinal-ruído óptica (OSNR) e a taxa de erro de bit (BER). Normalmente isso acarreta a execução de modelos de desempenho computacionalmente intensivos (Essiambre et al., 2010), que podem ser demorados e assim impactar serviços de tempo crítico como, por exemplo, a restauração de um canal óptico. Alternativamente, pode-se desenvolver uma ferramenta que estima a QoT de caminhos ópticos antes de sua implementação na rede com base em algoritmos de aprendizado de máquina (Morais e Pedro, 2018). Essa ferramenta será desenvolvida nesse trabalho e pode ajudar no roteamento e na atribuição de comprimento de onda, descartando, assim, conexões com baixo QoT.

Um caminho óptico consiste em um canal, ou comprimento de onda, entre dois nós de uma rede que é roteado através de vários nós intermediários. Todos os enlaces do caminho óptico possuem basicamente fibras ópticas (Agrawal, 2005a; Agrawal, 2005b), ROADMs (Essiambre et al., 2010; Agrawal, 2005b) e amplifi-

cadores ópticos EDFA (Erbium Doped Fiber Amplifier) (Essiambre et al., 2010; Desurvire et al., 2002) para compensar as perdas na fibra óptica. A informação é transmitida entre os nós através da multiplexação por divisão de comprimento de onda (Wavelength-division Multiplex, ou WDM), onde cada canal (comprimento de onda) ocupa uma banda no espectro, possui uma taxa de transmissão, formato de modulação e potência específicas. Os amplificadores ópticos adicionam ao sinal transmitido ruído, conhecido como ruído ASE (Amplified Spontaneous Emisision Noise). Além do ruído ASE, a interferência não linear entre os canais ópticos devido ao efeito Kerr limita a capacidade do canal (Essiambre et al., 2010; Temprana et al., 2015). Essa interferência pode ser modelada como um ruído Gaussiano (Poggiolini e Jiang, 2017; de A. Barboza et al., 2016) o que facilita a estimação do seu impacto no desempenho do sistema. Existem diversos algoritmos para a compensação dos efeitos lineares e não lineares (Napoli et al., 2014; Bayvel et al., 2013), no entanto, até o momento, não há um algoritmo capaz de compensar completamente os efeitos não lineares.

1.1 Justificativa

No contexto atual, a crescente complexidade e demanda por redes de transporte de dados confiáveis e de alta velocidade impulsionam a necessidade de soluções inovadores e eficientes. As novas gerações de redes móveis, serviços de stream e outras aplicações de alta largura de banda exigem uma comunicação com baixa latência e transmissão confiável. Nesse sentido, as redes ópticas WDM se apresentam como uma resposta viável para atender a essas demandas. As redes ópticas WDM permitem a transmissão simultânea de múltiplos sinais ópticos em diferentes comprimentos de onda, resultando em um aumento significativo da capacidade de transmissão de dados. No entanto, para garantir a qualidade de transmissão (QoT) em redes WDM, é necessário avaliar e prever a degradação do sinal óptico causada por diversos fatores, como atenuação, dispersão cromática, interferências e ruídos. Uma das métricas mais importantes para avaliar a QoT é a Relação Sinal-Ruído Óptico (OSNR).

A previsão precisa da OSNR é crucial para a implantação e operação eficiente de redes ópticas WDM. Atualmente, os métodos tradicionais de previsão da OSNR são computacionalmente custosos e demorados, o que dificulta sua aplicação em tempo real. Nesse contexto, o uso de técnicas avançadas, como as técnicas de aprendizado de máquina (regressores, árvores, máquina de vetores de suporte e redes neurais artificiais), tornam-se uma alternativa promissora.

1.2 Objetivo Geral

O objetivo desse trabalho é analisar a eficácia de três modelos de aprendizado de máquina quando usados para estimar a QoT de um caminho óptico ainda não estabelecido.

1.2.1 Objetivos específicos

Os objetivos específicos deste projeto são:

- Geração de dados sintéticos usando o modelo de ruído de interferência não linear;
- Desenvolvimento dos modelos de aprendizado de máquina: K-vizinhos mais próximos (K-Nearest Neighbors – KNN), máquinas de vetores de suporte (Support Vector Machines – SVM) e redes neurais artificiais (Artificial Neural Networks – ANN);
- Implementação desses modelos em forma de algoritmos e fórmulas com a ajuda de bibliotecas, softwares matemáticos ou linguagem de programação;
- Classificação dos caminhos ópticos usando os algoritmos de aprendizagem de máquina e comparar os três modelos de aprendizado de máquina e validar a ferramenta de QoT.

1.3 Organização do TCC

O conteúdo deste TCC está dividido em sete capítulos e um apêndice. As referências encontram-se nas páginas finais. A seguir, um resumo dos capítulos seguintes do TCC.

Capítulo 2. Fundamentação teórica: Tem como objetivo mostrar os princípios físicos e matemáticos de funcionamento por trás do sistema como um todo, procurando também descrever individualmente cada um dos elementos que compõem um sistema de comunicação óptica e das técnicas de aprendizado de maquina (ML) que serão utilizadas na estimação da qualidade de transmissão dos sistemas de comunicação óptica. São descritos as fibras ópticas e tipos de fibra, amplificadores ópticos, transmissores ópticos receptores ópticos e seus princípios de funcionamento e fenômenos ópticos associados, princípios de comunicação por fibras como formatos de modulação, desempenho de sistemas de comunicação óptica, conceitos fundamentais de ML e métricas de avaliação de desempenho de modelos de regressão.

Capítulo 3. Metodologia: Neste capitulo é desenvolvido os procedimentos para a execução do trabalho, desde a geração dos dados à comparação entre os resultados dos modelos de *machine learning* implementados. Inicialmente é obtido uma grande quantidade de dados através do modelo NLIN, que será visto na subseção 2.9. Estes dados são apresentados, detalhados e pré-processados nas respectivas subseções 3.2, 3.3. Com os dados pré-processados é realizado a implementação dos modelos de *machine learning* propostos para estimar a qualidade de transmissão por meio da ralação sinal-ruído, e também da potência do ruído não-linear.

Capítulo 4. Resultados: Neste capítulo, são apresentados os resultados por meio de metricas de avaliação de algoritmos de regressão, também é realizado uma comparação direta entre os algoritmos indicando o melhor custo beneficio em função do tempo de implementação, treinamento e estimação.

Capítulo 5. Conclusões e perspectivas futuras: Neste capítulo, busca-se chegar a uma conclusão do estudo, mostrando uma síntese dos objetivos do trabalho, a comparação com os resultados obtidos e dificuldades encontradas na sua realização, além da possível continuação do desenvolvimento do sistema experimental proposto.

Apêndice A. Neste capítulo é disponibilizado os algoritmos que foram utilizados no desenvolvimento deste trabalho.

Capítulo 2

Fundamentação Teórica

Neste capitulo é apresentado os conceitos, princípios de funcionamento e componentes dos sistemas de comunicação por fibras ópticas. Em seguida é apresentado o modelo NLIN de ruído de interferência não linear bem como os conceitos fundamentais e técnicas de implementação e estimação de desempenho dos modelos de aprendizado de máquina.

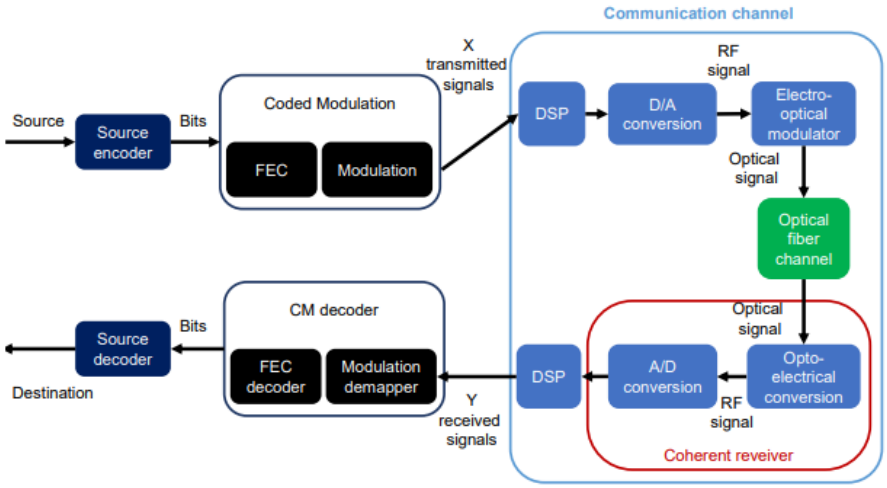
2.1 Sistema de Comunicação Óptica

2.1.1 Visão geral dos sistemas de comunicação óptica atuais.

Os sistemas de comunicação óptica têm desempenhado um papel fundamental na revolução das telecomunicações, proporcionando uma infraestrutura eficiente para a transmissão de dados em alta velocidade (Agrawal, 2021).

A visão geral do sistema de comunicação digital através de um canal de fibra óptica é apresentada na Figura 2.1.

Figura 2.1: Diagrama de um sistema coerente de comunicação por fibra óptica.



Fonte: Ye, 2023

A transmissão de informação na fibra acontece por meio do processo na qual a informação passa por um processo de codificação inicial por meio de um codificador de fonte, resultando em uma sequência de bits. Em seguida, essa sequência é transformada em uma série de símbolos $X = \{x_1, x_2, ..., x_N\}$ através de um codificador de modulação, que incorpora uma correção de erro para aumentar a robustez e confiabilidade da transmissão. Os símbolos X, juntamente com a constelação escolhida, são então transmitidos pelo canal de comunicação óptica. Para otimizar a transmissão, uma etapa de processamento digital de sinais (DSP) é realizada, envolvendo operações como *pulseshaping* e compensação digital. Em seguida, os símbolos X são convertidos em um sinal de radiofrequência (RF) por meio de um conversor digital-analógico (ADC) de alta resolução. Esse sinal de RF é usado para gerar o sinal óptico por meio de um modulador eletro-óptico e um laser. O sinal óptico resultante é então transmitido pela fibra óptica. Na extremidade receptora, o sinal óptico recebido é convertido novamente em RF por um receptor óptico coerente e, em seguida, em um sinal digital por um conversor analógico-digital (ADC). O receptor coerente permite a utilização de formatos de modulação avançados, como o formato de modulação de amplitude de quadratura (QAM), que codifica os dados de informação explorando os domínios de amplitude e fase do sinal óptico. Técnicas de DSP são aplicadas para compensar efeitos decorrentes da propagação na fibra

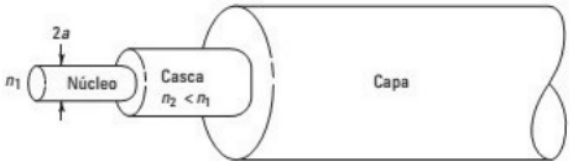
óptica e reconstruir os símbolos Y recebidos. O *demapper*, como a primeira etapa do decodificador de modulação, recupera as informações de bits da constelação dos símbolos recebidos. Em seguida, o decodificador FEC corrige eventuais erros usando relações de redundância com os bits para melhorar o desempenho. Por fim, os bits recuperados são descompactados pela decodificação do código-fonte inverso e encaminhados para o destino final (Ye, 2023).

2.2 Fibra Óptica

Uma fibra óptica é um guia de ondas dielétrico que opera nas chamadas frequências ópticas. Esse guia de ondas possui, normalmente, uma foma cilíndrica. Ela confina a energia eletromagnética na forma de luz dentro de sua estrutura e guia a luz em uma direção paralela ao seu eixo. As propriedades de transmissão de um guia de ondas óptico são ditadas por suas características estruturais, as quais têm efeito importante na determinação de como um sinal óptico é afetado ao propagar-se ao longo da fibra. A estrutura estabelece a capacidade de transportar informação e também influencia a resposta do guia de ondas a pertubações (Keiser, 2010).

A estrutura mais comumente aceita para guias de onda óptico é o cilindro dielétrico sólido de raio a e índice de refração n_1, como mostrado na Figura 2.2. Esse cilindro é conhecido como o núcleo da fibra. O núcleo é revestido por um material dielétrico sólido, conhecido como casca, que possui um índice de refração n_2 menor que o índice de refração do núcleo. A casca serve para reduzir a perda por dispersão resultante das descontinuidades da superfície do núcleo, aumentar a resistência mecânica da fibra e proteger o núcleo da absorção de contaminantes.

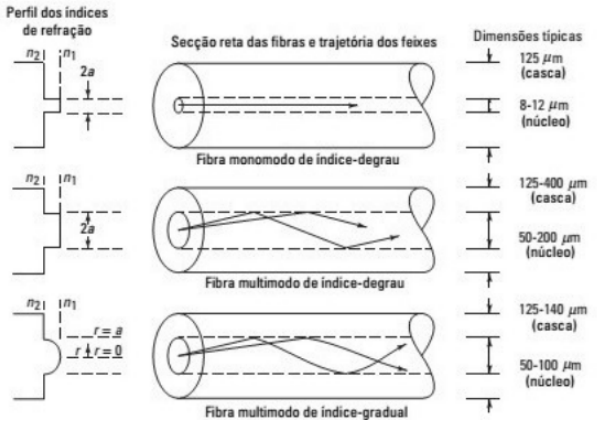
Figura 2.2: Esquema estrutural de uma fibra de sílica convencional.



Fonte: Keiser, 2010

As fibras ópticas mais comuns são compostas por um núcleo de sílica de alta pureza (SiO_2) revestido por uma casca de vidro, embora também sejam amplamente utilizadas fibras com núcleo de material plástico e casca plástica. Além disso, a maioria das fibras são encapsuladas em material plástico elástico resistentes à abrasão. Este material adiciona dureza à fibra, isolando-a mecanicamente. As variações na composição do núcleo resultam em dois tipos de fibras: as de índice-degrau, em que o índice de refração do núcleo é uniforme com uma mudança brusca na interface com a casca, e as de índice-gradual, em que o índice de refração varia radialmente a partir do centro da fibra (Keiser, 2010), a Figura 2.3 ilustra os tipos de fibras comentados .

Figura 2.3: Comparação entre as fibras ópticas convencionais monomodo e multimodo com índice-degrau e índice-gradual.



Fonte: Keiser, 2010

As fibras ainda podem ser divididas em dois grupos:

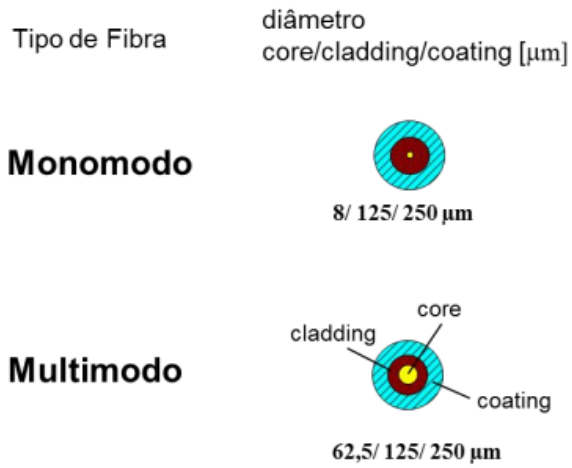
2.2.1 Monomodo (*Single-mode*)

Este tipo de fibra óptica é projetado para permitir que apenas um modo da luz viaje através do núcleo da fibra de cada vez. Ele é usado para transmitir sinais a longas distâncias com mínima atenuação (perda de sinal) e dispersão. A fibra monomodo é comumente usada em aplicações de longa distância, como em redes de telecomunicações e em sistemas de comunicação de alta velocidade.

2.2.2 Multimodo (*Multi-mode*)

Neste tipo de fibra óptica, o núcleo é maior, permitindo que vários modos da luz viajem através dele. Isso resulta em maior dispersão modal e atenuação de sinal em comparação com a fibra monomodo. A fibra multimodo é frequentemente utilizada em aplicações de curta distância, como em redes locais (LANs), sistemas de comunicação de campus e em aplicações de transmissão de dados em geral.
A Figura 2.4 trás uma comparação entre as dimensões das fibras monomodo e multimodo na qual é possível verificar que o diâmetro do núcleo da fibra multimodo é cerca de 7.8 vezes maior que o núcleo da fibra monomodo.

Figura 2.4: Comparação entre as fibras monomodo e multimodo.



Fonte: Leonardo Didier, 2021

2.2.3 Dispersão em fibras ópticas

A dispersão é o fenômeno pelo qual diferentes componentes de um sinal se propagam no núcleo da fibra, com velocidades distintas. Na maior parte dos casos, a dispersão limita a taxa de dados de um sinal digital ao espalhar os pulsos do sinal ao longo do tempo. Fisicamente, esse fenômeno significa que, em um sinal formado por vários pulsos, o alargamento de um pulso é ampliado de tal forma que há uma sobreposição com os pulsos vizinhos, tornando-os indistinguíveis na entrada do receptor. Esse efeito de sobreposição de pulsos é conhecido como interferência intersimbólica

(ISI - Intersymbol Interferency), a ISI causa uma distorção nominal na qual eleva a taxa de erro de bit (BER) (Filho, 2017). A dispersão de um sinal ocorre devido a fatores como o atraso intermodal, dispersão intramodal (ou cromática), dispersão do modo de polarização e efeitos de dispersão de ordem superior (Keiser, 2010), sendo os principais a dispersão cromática (CD - Chromatic Dispersion) e a dispersão de modo de polarização (PMD - Polarization Mode dispersion) (Filho, 2017).

Dispersão cromática

A dispersão cromática, também conhecida como dispersão intramodal, é uma ocorrência presente em todos os tipos de fibras ópticas. Ela ocorre devido à largura de linha espectral finita das fontes ópticas, resultando em diferenças de atraso de propagação entre os diferentes componentes espectrais do sinal transmitido. Isso leva ao alargamento de cada modo transmitido e, consequentemente, à dispersão intramodal. Essas diferenças de atraso podem ser causadas pelas propriedades dispersivas do material do guia de onda (dispersão do material) e pelos efeitos de orientação dentro da estrutura da fibra (dispersão do guia de onda) (Senior e Jamro, 2009).

Em (Filho, 2017) é introduzido a representação matemática para a dispersão cromática, na qual $\beta(\omega)$ é expandida em série de Taylor em torno da frequência central β_0, de forma que:

$$\beta(\omega) = n(\omega)\frac{\omega}{c} = \beta_0 + \beta_1(\omega - \omega_0) + \frac{1}{2}\beta_2(\omega - \omega_0)^2 + \frac{1}{6}\beta_2(\omega - \omega_0)^3 + ..., \qquad (2.1)$$

em que $n(\omega)$ é o índice de refração dependente da frequência ω e c é a velocidade da luz no vácuo. Os coeficientes $\beta_1, \beta_2, ...$ são dados por

$$\beta_m = \left[\frac{d^m\beta(\omega)}{d\omega^m}\right]_{\omega=\omega_0}, com \quad m = 0, 1, 2, 3, ..., \qquad (2.2)$$

Desta forma, o parâmetro β_1 está relacionado com a velocidade de grupo de um pulso enquanto que β_2 se relaciona com a variação da velocidade de grupo com a frequência (Filho, 2017). Utilizando (2.2), β_1 e β_2 podem ser expressados por:

$$\beta_1 = \frac{d\beta(\omega)}{d\omega} = \frac{1}{C}\left[n + \frac{\omega dn}{\omega}\right] = \frac{n_g}{c} = \frac{1}{v_g}, \tag{2.3}$$

$$\beta_2 = \frac{d^2\beta(\omega)}{d\omega^2} = \frac{d\beta_1}{d\omega} = \frac{d}{d\omega}\left[\frac{1}{v_g}\right] = -\frac{1}{v_g^2}\frac{dv_g}{d\omega}, \tag{2.4}$$

em que v_g é a velocidade de grupo. Outra forma de expressar a dispersão de velocidade de grupo em fibras é o parâmetro de dispersão D dado por:

$$D = \frac{d}{\lambda}\left(\frac{1}{v_g}\right) = -\frac{2\pi c}{\lambda}\beta_2 \tag{2.5}$$

Ao final do equacionamento da dispersão em (Filho, 2017) é demonstrado em termos numéricos o efeito da dispersão na prática.

2.2.4 Dispersão de modo de polarização

A dispersão do modo de polarização (PMD) é uma causa de alargamento de pulso originada pela birrefringência da fibra, podendo representar um fator limitante em comunicações por fibra óptica em taxas de transmissão elevadas. É um fenômeno aleatório influenciado por fatores intrínsecos, como a geometria não circular do núcleo da fibra e tensões residuais no material de vidro próximo ao núcleo, e por fatores extrínsecos, como tensões resultantes de carregamento mecânico, flexão ou torção da fibra. Em fibras ópticas reais, esses fatores levam a variações na velocidade de grupo conforme o estado de polarização (Senior e Jamro, 2009). Apesar de se chamarem fibras monomodo, elas suportam dois modos ortogonais de propagação distinguidos pelo seu estado de polarização. O alargamento dos pulsos podem ser estimados a partir do atraso de tempo, T, entre os dois componentes de polarização durante a propagação do pulso. Para uma fibra de comprimento L, $\Delta\tau$, em (Filho, 2017), (Keiser, 2010), o alargamento dos pulsos é dado por:

$$\Delta\tau_{PMD} = \left|\frac{L}{v_{gx}} - \frac{L}{v_{gy}}\right| = L\left|\frac{v_{gy} - v_{gx}}{v_{gx}v_{gy}}\right| \tag{2.6}$$

Vale ressaltar que, ao contrário da dispersão cromática, que é um fenômeno relativamente estável ao longo de uma fibra, a PMD varia aleatoriamente. A principal razão para isso é que as perturbações que causam os efeitos de birrefringência varia com a temperatura e dinâmica das tensões (Keiser, 2010). Outra forma de escrever a equação de (2.6) é:

$$\Delta\tau_{PMD} = D_{PMD}\sqrt{L}, \tag{2.7}$$

em que D_{PMD}, que é medido em $ps/\sqrt{km}$ e é o parâmetro médio de PMD.

2.2.5 Perdas em fibras ópticas

A atenuação do sinal luminoso que se propaga ao longo de uma fibra é uma consideração importante no projeto de um sistema de comunicações ópticas. Os fenômenos físicos que compõem a atenuação na fibra são a absorção, a dispersão e as perdas radiativas por absorção de energia óptica. Os principais fenômenos físicos que resultam no processo de atenuação são: absorção e espalhamento Rayleigh (Keiser, 2010). Como os receptores ópticos necessitam de certa quantidade mínima de energia para recuperar o sinal com precisão, a atenuação se torna um limitador em transmissões de longas distâncias (Filho, 2017). A atenuação na fibra é dada, em (Keiser, 2010), por:

$$\alpha(dB/Km) = \frac{10}{z} log_{10}\left[\frac{P_{out}}{P_{in}}\right] \approx 4,343\alpha(1/Km), \tag{2.8}$$

em que z é a distância percorrida pelo sinal na fibra em km, P_{in} é a potência de entrada na fibra e P_{out} é a potência de saída da fibra. As fibras comerciais possuem aproximadamente uma atenuação de 0,2 dB/km.

2.2.6 Efeitos não-lineares

Os efeitos não-lineares são fenômenos que estão diretamente relacionados com o nível de potência de lançamento do sinal propagado no núcleo da fibra. Como os

receptores ópticos necessitam de certa quantidade mínima de energia para recuperar o sinal com precisão, a atenuação se torna um limitador em transmissões de longas distâncias (Filho, 2017). Os principais efeitos não-lineares são: espalhamento estimulado Raman (SRS), espalhamento de Brillouin (SBS), automodulação de fase, modulação de fase cruzada e mistura de quatro ondas. Os efeitos não lineares são fracos em baixas potências, mas podem se tornar muito mais fortes em altas intensidades ópticas (Senior e Jamro, 2009).

Os efeitos não-lineares têm origem distintas. Os SRS e SBS emergem devido à interação de ondas de luz com as vibrações moleculares e acústicas no meio de sílica. Respectivamente, os SPM, XPM e FWM aparecem devido à dependência do índice de refração com a intensidade do campo elétrico aplicado, que por sua vez é proporcional ao quadrado da amplitude do campo, também referendado na literatura como efeito Kerr. A interação não-linear está diretamente relacionada à distância percorrida pelo sinal e à área da seção transversal da fibra. Quanto maior o comprimento do link, maior será a interação do sinal com o meio, resultando em uma maior intensidade dos efeitos não-lineares. No entanto, à medida que o sinal se propaga ao longo do link, sua potência diminui devido à atenuação da fibra consequentemente reduzindo a intensidade dos efeitos não-lineares (Filho, 2017).
Um modelo simplificado que assume que a potência do sinal permanece constante ao longo de um comprimento efetivo determinado, L_e, demonstrou ser bastante eficaz para compreender o efeito das não-linearidades definido, em (Filho, 2017), por:

$$L_e = \frac{1 - e^{-\alpha L}}{\alpha}, \tag{2.9}$$

em que $L >> \frac{1}{\alpha}$.

Em (Filho, 2017), ainda é discutido sobre os efeitos não-lineares que dependem da área do núcleo da fibra óptica, na qual é equacionada a A_{eff}. Também é discutido sobre o um parâmetro importante que é um coeficiente não-linear que resulta do efeito Kerr, na qual é responsável por produzir uma modulação de fase no sinal propagado induzido pela portadora [(Keiser, 2010), (Filho, 2017)].

Espalhamento estimulado Raman (SRS)

O espalhamento de Raman estimulado é uma interação entre as ondas de luz e os modos de vibração das moléculas de sílica. Se um fóton de energia hv_1 incide sobre uma molécula que possui uma frequência vibratória v_m, a molécula pode absorver alguma energia do fóton. Nessa iteração o fóton é espalhado, atingindo assim uma frequência menor que v_2 e uma energia correspondente menor hv_2. Esse processo gera luz espalhada em um comprimento de onda maior que a da luz incidente.

O espalhamento Raman estimulado é uma interação entre as ondas de luz e os modos de vibração das moléculas de sílica. Se um fóton de energia hv_1 incide sobre uma molécula que possui uma frequência vibratória v_m, a molécula pode absorver alguma energia do fóton. Nessa interação, o fóton é espalhado, atingindo assim uma frequência menor que v_2 e uma energia correspondente hv_2. Esse processo gera luz espalhada em um comprimento de onda maior que o da luz incidente (Keiser, 2010).

Em (Keiser, 2010),(Filho, 2017) e (Senior e Jamro, 2009) os autores discutem em detalhes sobre o fenômeno referido.

Espalhamento estimulado Brillouin (SBS)

O espalhamento Brillouin estimulado ocorre quando um sinal óptico intenso gera uma onda acústica, provocando variações no índice de refração. Essas variações fazem com que as ondas de luz se espalhem na direção oposta, em direção ao transmissor. A luz retrodifundida resultante experimenta ganho a partir dos sinais de propagação originais, resultando no esgotamento da potência do sinal. Além disso, a frequência da luz espalhada sofre um desvio Doppler dada, em (Keiser, 2010), por:

$$V_B = 2n\frac{V_s}{\lambda}, \tag{2.10}$$

Em que n é o índice de refração e V_s é a velocidade do som no matéria.

Para os demais efeitos não-lineares listados em 2.2.6 são discutidos mais profundamente pelos respectivos autores em ,(Keiser, 2010)(Filho, 2017) e (Senior e Jamro, 2009).

2.2.7 Ruído em sistemas de comunicação óptica

Nesta seção, será abordado o ruído da emissão espontânea amplificada (ASE - Amplified Spontaneous Emission). Em (Filho, 2017) os ruído de interferência não-linear (NLIN - Nonlinear Interference Noise) é discuido de maneira mais aprofundada pelo autor.

O principal ruído gerado em um amplificador óptico é conhecido como ruído da emissão espontânea amplificada (ASE). Quando os átomos da fibra dopada retornam aos seus estados de energia mais baixos, eles emitem fótons espontaneamente (Keiser, 2010).

O ruído ASE pode ser descrito como um fluxo de pulsos aleatórios que estão distribuídos infinitamente por todo o meio de amplificação. Esse processo aleatório é definido por um espectro de potência de ruído que é constante em todas as frequências. A densidade espectral de potência (PSD - *Power Spectral Density*) do ruído ASE, G_{ASE}, é dada por

$$G_{ASE} = hvn_{esp}[G(f) - 1] = \frac{P_{ASE}}{B_0}, \tag{2.11}$$

em que P_{ASE} é a potência do ruído ASE em um estado de polarização em uma largura de banda óptica B_0, $G(f)$ é o ganho do amplificador em função da frequência do sinal, h é a constante de Plank, v é a frequência do sinal óptico e n_{esp} é o fator de emissão espontânea ou de inversão de população definida, em (Keiser, 2010), como:

$$n_{esp} = \frac{n_1}{n_2 - n_1}, \tag{2.12}$$

em que n_1 e n_2 são densidades fracionadas ou populações de átomos em um estado inferior 1 e em um estado superior 2, respectivamente. Assim, n_{esp} indica o qual completa é a inversão de população entre dois níveis de energia (Keiser, 2010).

2.2.8 Principais tipos de fibra comerciais.

Extensas pesquisas foram realizadas desde a descoberta, por Charles Kao e George Hockham em 1966, da sílica como o material de escolha para fibras ópticas. A atenuação das primeiras fibras, próximas de 20 dB/km em 1970, foi reduzida para menos de 0,2 dB/km na atualidade (Filho, 2017).

Quadro 1: Principais fibras comerciais aplicadas em sistemas de comunicações ópticas.

Norma ITU	**Fibra**	**Característica**	$\lambda\,[nm]$
G.652	SSFM	Comprimento de onda de dispersão nula em torno de 1310 nm, originalmente otimizadas para utilização na região de comprimento de onda de 1310 nm, mas também utilizadas na região de 1550 nm.	1310, 1550
G.654	PSCF	Valor absoluto do coeficiente de dispersão cromática não-nulo em toda a faixa de comprimento de onda de 1530 a 1565 nm.	1530-165
G.655	LEAF/NZDSF	Comprimento de onda de dispersão zero em torno de 1300 nm, com perdas minimizadas no comprimento de onda de corte deslocado em torno da região de 1550 nm	1300,1550

Os principais tipos de fibras aplicadas em sistemas de comunicações ópticas são: fibra de núcleo de sílica puro (PSCF - *Pure-Silica-Core Fiber*), fibra de área efetiva larga (LEAF - *Large Effective Area Fiber*), fibra de dispersão deslocada não-nula (NZDSF - *Nonzero Dispersion Shifted Fiber*), fibra monomodo padrão (SSMF - *Standard Single Mode Fiber*), fibra de múltiplos núcleos (MCF - *Multicore Fiber*) e fibras com poucos modos (FMFs - *Few-Mode Fibers*). As características para cada uma dessas fibras é descrita pela União Internacional das Telecomunicações (ITU.T - *International elecommunication Union*), o Quadro 1 trás um resumo com a norma ITU que serve de recomendação para as fibras, as características e o comprimento de onda de maior eficiência de operação. A Tabela 2.1, exibe os valores dos coeficientes de atenuação, parâmetro de dispersão, coeficientes de não linearidade respectivamente.

Tabela 2.1: Parâmetros dos tipos de fibra

Fibra	$\alpha[dB/km]$	**D** [ps/nm/km]	$\gamma[1/W/km]$
SSMF	0.2	16.7	1.3
NZDSF	0.22	3.8	1.5
PSCF	0.17	20.1	0.8

2.3 Amplificador Óptico

A principal função dos amplificadores ópticos é compensar a atenuação óptica que ocorre nos canais ópticos devido à propagação do sinal nas fibras e nos componentes passivos. A perda gradual de intensidade do sinal pode resultar em um sinal tão enfraquecido que se torna indetectável. Portanto, é necessário restaurar a intensidade do sinal. O parâmetro mais crucial do amplificador óptico é o seu ganho, que é essencialmente definido, em (Filho, 2017), por:

$$G \triangleq \frac{P_{out}}{P_{in}}, \tag{2.13}$$

Em que P_{out} e P_{in} são as potências de saída e de entrada do sinal, respectivamente. No entanto a intensidade de radiação em um fóton varia exponencialmente com a distância percorrida na cavidade de um laser (Keiser, 2010). O ganho óptico é influenciado pelo comprimento de onda do sinal incidente, as intensidades do sinal e do bombeamento, o comprimento de onda do bombeamento (no caso do bombeamento óptico), a fibra hospedeira (fibra óptica dopada), e outros parâmetros relevantes. Uma modelagem inicial do ganho do meio pode ser realizada por um sistema com dois níveis de energia amplamente dispersos (com mesma energia de transição), cuja descrição é aproximada pela função lorentziana como em (Filho, 2017).

$$G_\omega = \frac{g_0}{1 + (\omega - \omega_0)^2 T_2^2 + \frac{P}{P_{sat}}}, \tag{2.14}$$

em que g_0 é o ganho máximo do amplificador, determinado pela intensidade do bombeamento, ω é a frequência óptica do sinal incidente, ω_0 é a frequência da transição atômica e P é a potência óptica do sinal a ser amplificado, P_{sat} é a potência de saturação do sinal, a qual é determinada por parâmetros do meio responsável pela geração do ganho, tais como o tempo de fluorescência e a seção de choque da transição. O parâmetro T_2, chamado de tempo de relaxação de dipolo, geralmente é extremamente curto (0,1 ps a 1 ns) (Filho, 2017).

Ao considerarmos que $P << P_{sat}$, temos que $\frac{P}{P_{sat}} \cong 0$, logo a equação (2.14) pode ser aproximada para:

$$G_\omega = \frac{g_0}{1 + (\omega - \omega_0)^2 T_2^2} \tag{2.15}$$

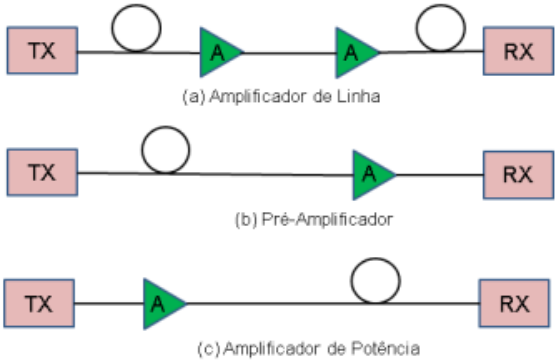
Por (2.15) podemos concluir que o ganho máximo ocorre quando a frequência do sinal coincide com a frequência de transição atômica. Considerando a frequência do sinal exatamente sintonizada na frequência de transição atômica, ou seja $\omega = \omega_0$ tem-se a partir de 2.14

$$G_\omega = \frac{g_0}{1 + \frac{P}{P_{sat}}}, \tag{2.16}$$

Por (2.16), verifica-se que para uma determinada operação de bombeamento, o ganho do amplificador tende a diminuir à medida que a potência óptica do sinal na entrada do amplificador aumenta. Os amplificadores ópticos são capazes de fornecer ganho para múltiplos canais em sistemas de multiplexação por divisão de comprimento de onda (WDM). (Filho, 2017).

Ainda em (Filho, 2017) é discutido sobre uma classificação dos amplificadores dependendo da aplicação. A Figura 2.5 ilustra os tipos discutidos.

Figura 2.5: Classes de amplificadores



Fonte: H. S. CARVALHO, 2006

Os principais tipos de amplificadores ópticos descritos na literatura incluem o amplificador de fibra Raman (FRA - *Fiber Raman Amplifier*), o amplificador de fibra dopada com érbio (EDFA - *Erbium Doped Fibre Amplifier*), o amplificador em

guia de onda dopado com érbio (EDWA - *Erbium Doped Waveguide Amplifier*) e o amplificador óptico semicondutor (SOA - *Semiconductor Optical Amplifier*). Nos sistemas WDM, os amplificadores FRA e EDFA são particularmente notáveis.

2.3.1 Amplificadores EDFA

O EDFA é o amplificador óptico mais utilizado devido à sua compatibilidade com a transmissão em fibras, eficiência energética e baixo custo. Utiliza uma fibra óptica dopada com érbio como meio de ganho. O seu processo de amplificação implica em baixa amplificação de ruído numa banda de comprimento de onda de 1530nm a 1565nm com uma largura de banda total em torno de 35nm, a chamada banda C (Agrell et al., 2016). A grande vantagem dos EDFA é que eles são capazes de amplificar simultaneamente muitos canais WDM. Um aspecto importante dos amplificadores ópticos é o ruído devido à emissão espontânea amplificada (ASE - *Amplified Spontaneous Emission*), além da emissão estimulada que gera ganho. Este ganho médio também produz emissão espontânea, o que dá origem ao espectro da ASE do amplificador. O ruído de ASE limita a relação sinal-ruído óptica (OSNR - *Optical Signal-to-Noise Ratio*), principalmente quando amplificadores são posicionados em "cascata", sendo quantificada na figura de ruído (NF - *Noise Figure*) no amplificador. (Filho, 2017)

2.3.2 Amplificadores Raman

Os amplificadores Raman exploram o fenômeno do espalhamento Raman estimulado (SRS - *Stimulated Raman Scattering*)em fibras de sílica para amplificar sinais ópticos. Durante o processo de SRS, um fóton de bombeio transfere energia para criar um fóton de menor energia na frequência do sinal, com a energia restante absorvida como vibrações moleculares. Esses amplificadores oferecem baixos ganhos por unidade de comprimento em comparação com os amplificadores de fibra dopada com érbio (EDFA), requerendo fibras longas para operação eficaz. Ganhos mais elevados são obtidos utilizando fibras com áreas efetivas menores e baixas perdas. O

bombeamento Raman pode ser feito tanto na mesma direção do sinal (copropagante) quanto na direção oposta (contrapropagante), porém, a necessidade de lasers de alta potência para estimular o SRS representa um desafio devido aos custos associados (Filho, 2017).

2.4 Transmissor óptico

O objetivo de um transmissor óptico é converter sinais elétricos em sinais ópticos, em forma de luz, para serem transmitidos em fibras ópticas. Este processo é realizado por meio de componentes como lasers ou diodos emissores de luz (LEDs), que convertem os sinais elétricos em pulsos de luz que são então transmitidos pela fibra óptica. Esses sinais ópticos podem percorrer longas distâncias com alta velocidade e baixa perda de sinal, sendo utilizados em diversas aplicações de comunicação, como redes de telecomunicações, internet de alta velocidade e transmissão de dados em geral.

2.4.1 Diodo emissor de luz (LED)

Em termos simples o LED (do inglês *Light Emitting Diode*) é um tipo de dispositivo semicondutor que emite luz quando polarizado diretamente na junção p-n. O funcionamento do LED é baseado em uma forma especial de eletroluminescência, produzida pela injeção de portadores de carga em uma junção p-n. Quando uma junção p-n é polarizada diretamente, buracos no lado p e elétrons no lado n se movem em direções opostas em direção à região de depleção. Os buracos injetados no lado n se recombinam com elétrons que estão chegando na região de depleção, enquanto os elétrons injetados no lado p se recombinam com buracos. Dessa forma, todos os elétrons e buracos que participam da corrente se recombinam nas proximidades da região de depleção, em uma camada de espessura L_p no lado p e L_n no lado n. Se o semicondutor da junção tem um gap indireto, como Si ou Ge, a recombinação produz fônons, além de fótons e, portanto, calor. Isso torna a emissão de luz muito

ineficiente em junções p-n feitas com semicondutores de gap indireto. No entanto, se o semicondutor tem um gap direto, a recombinação de cada par elétron-buraco resulta na emissão de um fóton (Rezende, 2022).

Os fótons emitidos podem pertencer à faixa infravermelha, visível ou ultravioleta, dependendo do material semicondutor utilizado. Essa variação no comprimento de onda está diretamente relacionada ao intervalo de energia, conhecido como gap de energia, do material semicondutor. A potência óptica emitida por um LED é diretamente proporcional à corrente que flui no circuito. Em termos simples, aumentar a corrente no circuito elétrico apenas resulta em maior emissão espontânea, sem chance de gerar emissão estimulada devido à baixa refletividade. Devido a essa limitação, os LEDs não podem produzir potências ópticas tão altas quanto os lasers (Filho, 2017).

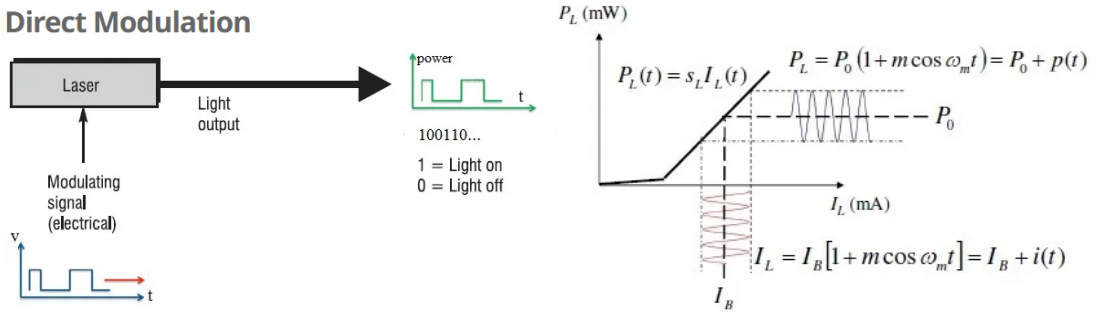
2.4.2 Laser

O conceito do laser foi inventado por Schawlow e Townes em 1958, sendo testado pela primeira vez por Maiman em 1960 usando uma haste de rubi. Seguindo este conceito, surgiram uma grande variedade de lasers, entre esses estão: o laser hélio-neon (He-Ne - Hélio- Neon), o laser de dióxido de carbono (CO2 - Carbon dioxide), o laser de "corante" e o laser semicondutor. O laser semicondutor é o mais importante em se tratando de comunicações ópticas, sendo aplicados quase que exclusivamente para comunicações em fibras ópticas. Desenvolvidos na década de 1980, operam na faixa de comprimento de onda de 1,3 μm a 1,6μm e utilizam em sua composição um composto quaternário de índium-gálio-arseneto-fosfeto ($In_{1-x}Ga_xAs_yP_{1-y}$ - *Indium-gallium-arsenide-phosphide*) (Filho, 2017). Ainda em (Filho, 2017), também é discutido os principais *lasers* semicondutores e sobre a degradação da sensibilidade na qual é introduzida uma equação que descreve como a largura espectral varia em função da potência de saída.

2.4.3 Modulação direta e externa

A inserção de dados em sinais de luz é denominada modulação óptica, sendo geralmente executada de duas formas: modulação direta ou através de um modulador externo. O esquema mais comum de modulação direta é o *On-Off Keying*, onde o fluxo de luz é ligado ou desligado para representar os bits de dados. Na modulação direta, a corrente de acionamento no laser semicondutor é ajustada acima de um limite para o bit 1 e abaixo desse limite para o bit 0, e a proporção das potências de saída dos bits 1 e 0 é chamada de razão de extinção, a Figura 2.6 em (a), é mostrado um diagrama da modulação direta. Em (b), é mostrado a curva característica entre entrada elétrica vs saída óptica. Embora seja simples e econômica, a modulação direta aumenta a largura de linha do laser, resultando em *chirp*, o que limita o desempenho do sistema a taxas de dados acima de 5 Gbit/s (Filho, 2017).

Figura 2.6: Em (a), temos o diagrama de bloco para a modulação direta em que o diodo laser usado como modulação direta. Em (b), temos a característica de entrada elétrica vs saída óptica para o método de modulação direta.

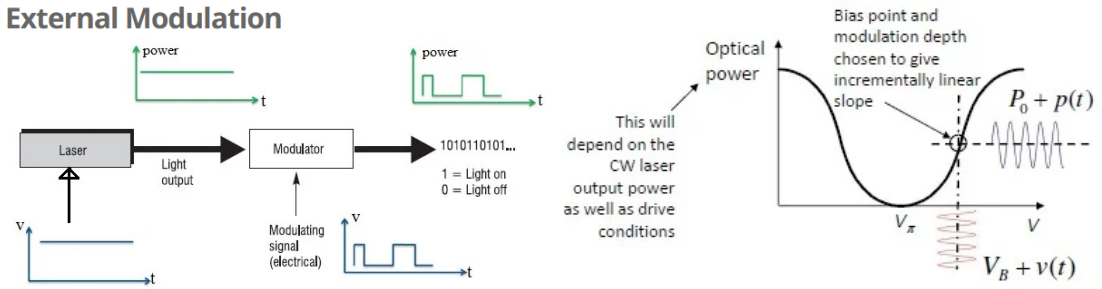


(a) (b)

Fonte: RF *Wireless World*

Em sistemas de comunicação de alta taxa, é preferível empregar um modulador externo. Os dois principais tipos de moduladores externos usados em sistemas de comunicação óptica são o modulador de eletroabsorção (EAM - Modulador Eletro-absorvente) e o modulador Mach-Zehnder eletro-óptico de fase (MZM - Modulador Mach-Zehnder). A Figura 2.7, ilustra o diagrama de blocos da modulação externa (Filho, 2017).

Figura 2.7: Em (a), temos o diagrama de blocos para a modulação externa na qual o diodo laser emite o sinal e o bloco do modulador é responsável por codificar o sinal óptico em sinal digital. Em (b), temos a característica de entrada elétrica vs saída óptica para o método de modulação direta.



(a) (b)

Fonte: RF *Wireless World*

O Quadro 2 trás de forma resumida as vantagens e desvantagens discutidas pelo autor.

Quadro 2: Vantagens e desvantagens dos moduladores externos EAM e MZR

Modulador	Vantanges	Desvantagens
EAM	1. Pode ser integrado sobre o mesmo substrato do laser de realimentação distribuída (DFB). 2. Altera as propriedades de transmissão do material para permitir modulação direta. 3. Funciona tanto como modulador de intensidade quanto de fase.	1. Limitado pela eficiência de modulação e largura de banda. 2. Pode exigir maior voltagem para operar.
MZR	1. Permite modulação de fase e intensidade. 2. Alta eficiência de modulação e largura de banda. 3. Pode alcançar razão de extinção superior a 20 dB.	1. Requer design cuidadoso para garantir eficiência máxima. 2. Exige voltagem específica (V_π) para produzir mudança de fase desejada. 3. Mais complexo em comparação com o EAM.

Ainda em (Filho, 2017) é discutido sobre o funcionamento dos tipos de moduladores externo.

2.5 Receptor Óptico

Um receptor óptico é constituído por um fotodetector, um amplificador e um circuito de processamento de sinal. O objetivo do receptor óptico é transformar o sinal óptico enviado pelo transmissor óptico através da fibra óptica, na forma de luz, em um sinal elétrico, recuperando os dados transmitidos de forma confiável (Keiser, 2010). O sinal óptico é convertido em sinal elétrico por meio do fotodetector p-i-n ou avalanche (seção (a) da Figura 2.8), que gera uma corrente elétrica proporcional à potência óptica incidente. O amplificador de front-end (receptor óptico constituído de um fotodiodo seguido de um pré-amplicador) aumenta a potência do sinal elétrico

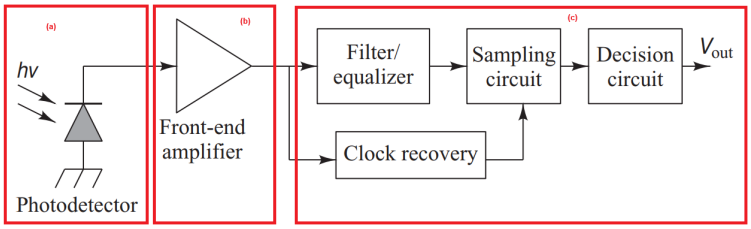
gerado para um nível utilizável (seção (b) da Figura 2.8). O amplificador front-end é seguido por um circuito de decisão que estima os dados a partir da saída do amplificador front-end (seção (c) Figura 2.8).

Uma característica importante do receptor óptico é o ruído gerado pelo fotodetector. Os receptores ópticos convertem a potência óptica incidente, P_{in}, em corrente elétrica através de um fotodiodo. A relação, $I_p = R \cdot P_{\text{in}}$, onde a fotocorrente I_p é diretamente proporcional à potência óptica incidente P_{in}, assume que essa conversão é livre de ruído. No entanto, este não é o caso mesmo para um receptor perfeito. Existem dois mecanismos fundamentais de ruído: o ruído de disparo e ruído térmico. Esses ruídos levam à flutuações na corrente, mesmo quando o sinal óptico incidente tem uma potência constante. A relação $I_p = R \cdot P_{\text{in}}$ ainda é válida se I_p for interpretado como a corrente média. E o ruído elétrico induzido pelas flutuações de corrente e este afeta o desempenho do receptor. (Filho, 2017).

Os receptores ópticos podem ser classificados em dois tipos: receptores ópticos com detecção direta e receptores ópticos coerentes. Nos receptores ópticos com detecção direta a recuperação do sinal transmitido é realizado de maneira direta a partir da detecção do sinal por um fotodetector.

Os receptores ópticos coerentes combinam o sinal óptico de entrada de forma coerente com um laser de onda contínua (CW - *Continuous Wave*), antes do detector. O CW fornece a fase de referência necessária para a detecção coerente do sinal óptico recebido. A mistura do sinal recebido com a referência do laser permite a extração de informações sobre a amplitude e a fase do sinal, possibilitando a detecção de modulações de fase, como a modulação de fase diferencial (DPSK), ou modulações de amplitude e fase, como a modulação de amplitude e fase coerente (QAM coerente).

Figura 2.8: Seções básicas de um receptor óptico



Fonte: Keiser, G, 2010. Adaptado

Em (Filho, 2017) e (Faruk e Savory, 2017), o processo de transmissão e detecção coerente são abordados desde a introdução à modelagem matemática e a explicação de conceitos fundamentais essenciais para o entendimento do processo.

2.5.1 Fotodetector

O sinal elétrico captado pelo fotodetector incorpora três fontes distintas de ruído: o ruído térmico, o ruído de disparo e, em sistemas amplificados, o ruído ASE. O ruído térmico é uma presença constante em todos os sistemas de comunicação que operam em temperaturas acima do zero absoluto (0 K). Originado pela agitação térmica das cargas, ele segue um processo aleatório gaussiano, sendo independente da frequência dentro da banda de transmissão e, portanto, é denominado ruído branco. Sua magnitude aumenta conforme a largura de banda elétrica do receptor cresce (Ramaswami et al., 2010).

A conversão do sinal óptico em corrente elétrica ocorre pela detecção de fótons no fotodetector. No entanto, a chegada desses fótons no detector é um processo aleatório, modelado por uma distribuição de Poisson, caracterizando o ruído de disparo. Isso resulta em flutuações na corrente independentes da frequência. A variância do ruído de disparo é diretamente proporcional à largura de banda elétrica do receptor, e seu impacto na recepção do sinal é também considerável. Entre esses fenômenos de ruído, o ASE é predominante nos sistemas ópticos modernos, ocorrendo nos amplificadores ópticos e nos lasers. Caracterizado pela geração de fótons com fase e polarização aleatórias devido à emissão espontânea, ele é especialmente

limitante em situações de pré-amplificação no receptor ou de amplificação em cascata ao longo do enlace, onde o ruído se acumula em cada estágio de amplificação.

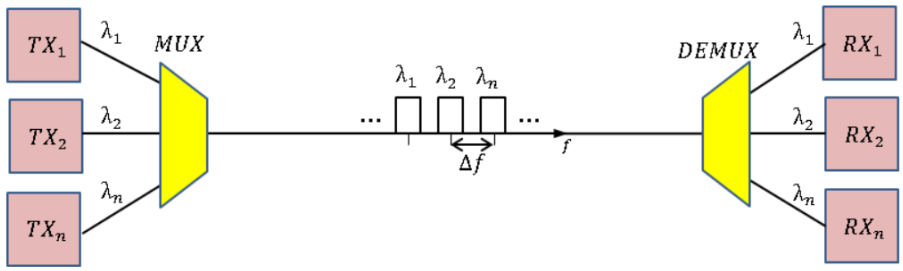
2.6 Multiplexadores e demultiplexadores

As primeiras conexões ópticas eram ponto a ponto, utilizando fibras ópticas com uma única fonte de luz para transmitir informações entre dois pontos. Com o surgimento das redes ópticas WDM, os multiplexadores e demultiplexadores tornaram-se indispensáveis nos sistemas WDM.

A função do multiplexador é combinar múltiplos sinais ópticos provenientes de diversas fibras em uma única fibra. Por sua vez, o demultiplexador recebe todos os sinais combinados em uma única fibra, separa esses canais e os encaminha para cada fibra ou fotodetector correspondente a cada sinal.

Os multiplexadores e demultiplexadores ópticos podem ser categorizados como componentes passivos ou ativos. Uma representação esquemática de um sistema WDM utilizando multiplexador e demultiplexador é apresentada na Figura 2.9. A letra grega λ (lambda) representa os n sinais ópticos que entram no multiplexador e os sinais ópticos correspondentes que saem do demultiplexador.

Figura 2.9: Diagrama de blocos de um sistema WDM utilizando multiplexador e demultiplexador



Fonte: Senior, J. M., 2009

A principal consideração na aplicação de multiplexadores e demultiplexadores está relacionada aos espaçamentos entre os canais. Quanto mais compacto for o sistema WDM, mais próximos estarão os canais, aumentando o risco de interferência entre eles.

2.7 Formatos de Modulação

Os avanços nos formatos de modulação e nos esquemas de multiplexação têm impulsionado um notável aumento na capacidade das fibras ópticas monomodo. Inicialmente, os sistemas comerciais predominavam com modulação binária *on-off keying*, atingindo uma taxa de 10 Gb/s por canal de comprimento de onda, e um espaçamento de frequência entre canais de 50 GHz ou 100 GHz. Contudo, os produtos comerciais atuais oferecem taxas de bits de até 200 Gb/s por canal de comprimento de onda, utilizando larguras de banda de 37,5 GHz ou até mesmo mais estreitas (Filho, 2017).

2.7.1 Modulação QPSK

Na modulação QPSK, uma forma de onda senoidal é variada em fase, mantendo constante a amplitude e a frequência. O termo quadratura indica que existem quatro fases possíveis. A equação da forma de onda QPSK é dada, como em (Filho, 2017), pela expressão geral:

$$s_i(t) = Acos[\omega_c t + \phi_0 + \phi_i(t)], \tag{2.17}$$

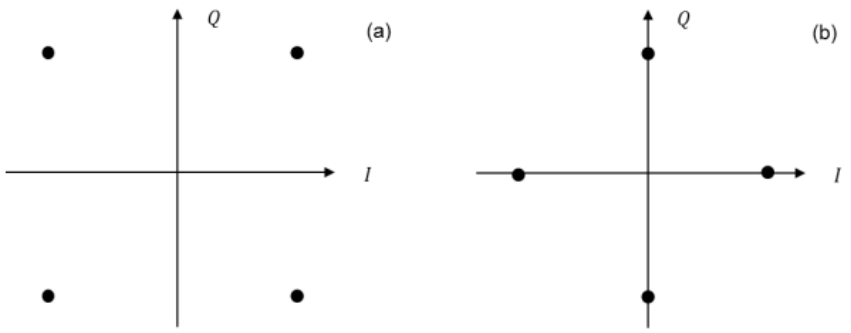
em que $A = \sqrt{\frac{2E_S}{T_S}}$, s_i é a forma de onda do sinal PSK para a fase i, t é o tempo, A é a amplitude, ω_c é a frequência da portadora em radiano por segundo ($\omega_c = 2\pi f_c$), ϕ_0 é o ângulo de fase de referência, ϕ_i é a fase i, com i = 1, 2, 3 e 4. A fase instantânea $\phi_i(t)$ possui valores discretos iguais a $\phi_0 + \frac{2\pi i}{4}$. Substituindo os valores em 2.17

$$S_{QPSK} = \sqrt{\frac{2E_S}{T_S}} cos\left[2\pi f_c + (i-1)\frac{\pi}{2}\right], \tag{2.18}$$

em que E_S é a energia do símbolo e T_S é a duração do símbolo e é igual a duas vezes o período do bit (Filho, 2017). Ainda em (Filho, 2017) é realizada uma manipulação algébrica utilizando identidades trigonométricas para checar a equação:

$$S_{QPSK} = \left\{ \sqrt{E_S} cos\left[(i-1)\frac{\pi}{2}\right]\phi_1(t) - \sqrt{E_S} sin\left[(i-1)\frac{\pi}{2}\right]\phi_2(t) \right\}, i = 1, 2, 3\, e\, 4. \quad (2.19)$$

Figura 2.10: Constelação QPSK: (a) ($\phi = \frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4} e \frac{7\pi}{4}$); ($b$)$\phi = 0, \frac{\pi}{2}, \pi e \frac{3\pi}{2}$)



Fonte: LABVOLT, 2016

Em que $\phi_1(t) = \frac{2}{T_S} cos(2\pi f_c t)$, $\phi_2 = \frac{2}{T_S} sin(2\pi f_c t)$ e $0 \leq t \leq T$ para o QPSK. A Figura 2.10 ilustra a costelação para a modulação QPSK.

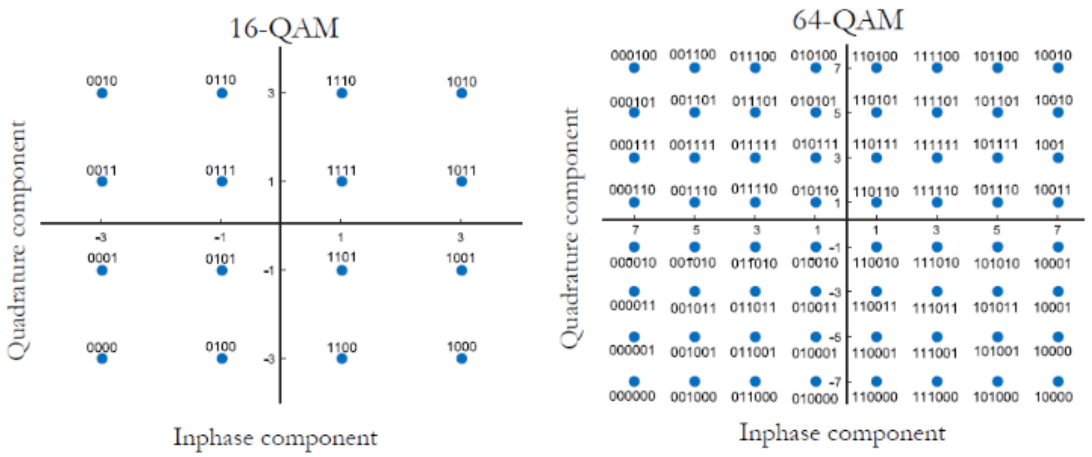
2.7.2 Modulação QAM

Na modulação PSK M-ária, a amplitude do sinal transmitido é mantida constante, resultando em uma constelação circular. No entanto, ao permitir que a amplitude varie com a fase, obtemos um novo esquema de modulação conhecido como QAM (Filho, 2017). Os tipos mais comuns são a QAM-16, a QAM-64 e a QAM-256. A QAM-16 utiliza 16 níveis de amplitude e fase, permitindo transmitir até 4 bits por símbolo. Já a QAM-64 utiliza 64 níveis, possibilitando transmitir até 6 bits por símbolo. Por fim, a QAM-256 utiliza 256 níveis, permitindo transmitir até 8 bits por símbolo (Besnoff e Ricketts, 2015). A forma geral de um sinal QAM M-ária pode ser definida como:

$$S_i = \sqrt{\frac{2E_{min}}{T_S}} a_i cos(2\pi f_c t) + \sqrt{\frac{2E_{min}}{T_S}} b_i sin(2\pi f_c t), \quad (2.20)$$

em que $0 \leq t \leq T$, $i = 1, 2, ..., M$, E_{min} é a energia do sinal com menor amplitude, e a_1 e b_1 são um par de inteiros independentes escolhidos de acordo com a localização do ponto de sinal particular. A modulação QAM M-ary não tem energia constante por símbolo, nem tem distância constante entre possíveis estados de símbolos (Filho, 2017). Na Figura 2.11, (a) e (b) ilustram o diagrama de constelação das modulações PM-16QAM e PM-64QAM.

Figura 2.11: Diagrama de constelação das modulações PM-16QAM (a) e PM-64QAM (b).



(a) (b)

Fonte: *Gaussian Wave*

2.8 Desempenho de sistemas de comunicação óptica

A medição do desempenho de uma rede óptica é fundamental para garantir sua eficiência, confiabilidade e qualidade de serviço. Com o aumento da demanda por largura de banda e confiabilidade nas comunicações ópticas, torna-se essencial empregar métodos precisos e abrangentes para avaliar o desempenho dessas redes. A avaliação do desempenho de uma rede óptica envolve a análise de várias métricas-chave, incluindo a taxa de erro de bit (BER), a relação sinal-ruído óptica (OSNR/$OSNR_{dB_{NL}}$), a potência óptica recebida (ROP), a eficiência espectral, a perda de inserção, o tempo de latência e outros parâmetros relevantes. Este processo

de medição não apenas fornece informações sobre a qualidade do serviço oferecido pela rede, mas também ajuda na identificação e resolução de problemas, na otimização do desempenho e na implementação de melhorias contínuas. Nesta seção, exploraremos as $OSNR_{dB_{NL}}$.

2.8.1 Relação sinal-ruído óptico.

A relação sinal-ruído óptica (OSNR - Optical Signal-to-Noise Ratio) é definida como a proporção entre a potência média do sinal óptico e a potência do ruído. Em sistemas de transmissão óptica WDM coerentes, nos quais são empregados amplificadores EDFA para compensar as perdas nas fibras ópticas, essa relação é especificamente a razão entre a potência por canal e a potência total do ruído ASE acumulado [(Keiser, 2010), (Filho, 2017)], dada por:

$$OSNR = \frac{P_{ch}}{P_{ASE}}, \tag{2.21}$$

em que P_{ch} é a potência média de entrada por canal e P_{ASE} é a potência acumulada do ruído ASE. Em (Filho, 2017) a equação (2.21) é expressa em dB, como:

$$OSNR_{dB} = 58 - NF - \alpha L - 10log_{10}N_s, \tag{2.22}$$

em que αL são as perdas nas fibra, N_s é o número de *spans* e NF é a figura de ruído que quantifica o ruído ASE dos amplificadores (Keiser, 2010), (Filho, 2017). Em (Filho, 2017)(Agrawal, 2010), a NF é expressa como

$$NF = 2n_{esp}\left(1 - \frac{1}{G}\right) + \frac{1}{G}, \tag{2.23}$$

para $G >> 1$, a equação em (2.23) pode ser aproximada para

$$NF \approx 2n_{esp} \tag{2.24}$$

No cenário há um regime não-linear, regime no qual os efeitos não-lineares afetam

a propagação do sinal na fibra, a OSNR é determinada pela proporção entre a potência por canal e a soma da potência ASE com a potência NLIN (Filho, 2017). Então a OSNR é expressa como

$$OSNR_{NL} = \frac{P_{ch}}{P_{ASE} + P_{NLIN}}, \tag{2.25}$$

em que P_{NLIN} é a potência acumulado do ruído causado por efeitos não-lineares na propagação do sinal na fibra.

2.9 Modelo NLIN

O desempenho de um sistema óptico coerente é comumente avaliado por meio da razão óptica sinal-ruído (OSNR). O OSNR está relacionado à relação sinal/ruído (SNR) por meio da largura de banda de ruído B_N e taxa de símbolos R_S, sendo o mesmo quando $B_N = R_S$ (Poggiolini et al., 2014). O desempenho de um sistema óptico é basicamente limitado pelo ruído de emissão espontânea amplificada (ASE) e ruído de interferência não linear (NLIN) (da Silva et al., 2017). Assim, a SNR de um determinado canal óptico que trata distorções não lineares como ruído pode ser modelada como em (Poggiolini et al., 2014):

$$SNR = \frac{P}{\sigma^2_{ASE} + \sigma^2_{NLIN}}, \tag{2.26}$$

em que P denota a potência do canal em teste, σ^2_{ASE} é a variância do ruído devido ao ASE e σ^2_{NLIN} é a variância do ruído de interferência não linear (NLIN). Neste caso, ASE e NLI são assumidos como aditivos e independentes. Assumindo um enlace óptico com N amplificadores, como EDFAs, com ganho G e fator de ruído F, a potência ASE é dada por (Poggiolini et al., 2014):

$$\sigma^2_{ASE} = NhvFGB_n, \tag{2.27}$$

em que h é a constante de Planck, v é a frequência do canal e B_N é a largura

de banda do ruído. Neste trabalho assumiremos sempre que B_N=R_S, como em (da Silva et al., 2017).

Assumindo a transmissão de canais N_{ch} WDM com potências de lançamento iguais a P, a variância NLIN em um canal em teste pode ser obtida a partir do modelo NLIN para dois canais, reescrito aqui para qualquer número de canais:

$$\sigma^2_{NLIN} = P^3 \sum_{1}^{N_{ch}} \left\{ \chi_1 + \chi_2 \left(\frac{\left\langle |b|^4 \right\rangle}{\left\langle |b|^2 \right\rangle^2} - 2 \right) \right\} + P^3 \chi_3 \left(\frac{\left\langle |b|^6 \right\rangle}{\left\langle |b|^3 \right\rangle^2} - 9 \frac{\left\langle |b|^4 \right\rangle}{\left\langle |b|^2 \right\rangle^2} + 12 \right), \quad (2.28)$$

em que b representa um ponto da constelação de polarização única do canal de interferência, e os colchetes angulados denotam média estatística sobre os pontos da constelação de dados. Em (da Silva et al., 2017), os termos χ_1, χ_2e χ_3 são derivados como em (Dar et al., 2013) e (Carena et al., 2014).

Esses coeficientes são funções da forma de onda do pulso transmitido e dos parâmetros das fibras. A função com os parâmetros da fibra é comum aos três coeficientes χ_1, χ_2 e χ_3 e, para um enlace óptico homogêneo, é dada, em (Dar et al., 2013), por:

$$\rho_{lmn} = \frac{\gamma}{(MT)^{3/2}} e^{-i(\omega_l + \omega_m - \omega_n)t} \times \frac{1 - e^{i\theta N L_s}}{1 - e^{i\theta L_s}} \frac{1 - e^{i\alpha L_s} e^{i\theta L_s}}{\alpha - i\theta} \quad (2.29)$$

Em que $\theta = \beta''(\omega_m – \omega_n)(\omega_l – \Omega – \omega_n)$, o espaçamento entre canais WDM é assumido como $\Omega = q2\pi/T$, $q = 0, 1, 2, ...$, $\omega_n = n2\pi/MT$, T é a duração do símbolo, M é o período dos símbolos transmitidos, L_s é o comprimento de um único *span* e N é o número total de *spans* no sistema. Os parâmetros α, β'' e γ são os coeficientes de atenuação, dispersão e não-linear da fibra óptica, respectivamente. O cálculo dos coeficientes χ_1, χ_2 e χ_3 requer a soma de até cinco índices. Um código (escrito em Matlab) calcula esses coeficientes usando o método de integração de Monte Carlo e está disponibilizado em (Filho, 2017). Ele faz uma modificação no código fonte de (Dar et al., 2014a) para generalizar o algoritmo para links heterogêneos.

2.10 Introdução à Aprendizagem de máquina (ML -*Machine Learning*)

Em 1959, o cientista da computação e pioneiro em aprendizado de máquina Arthur Samuel definiu o aprendizado de máquina como o "campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados" (Samuel, 1959). O livro de Tom Mitchell de 1997 sobre aprendizado de máquina definiu aprendizado de máquina como "o estudo de algoritmos de computador que permite que programas de computador melhorem automaticamente através da experiência". Ele definiu a aprendizagem da seguinte forma: "Diz-se que um programa de computador aprende com a experiência E em relação a alguma classe de tarefas T e mede o desempenho P, se seu desempenho em tarefas em T, medido por P, melhora com a experiência E" (Mitchell-Hill, 1997). O aprendizado de máquina (ML) é um ramo da inteligência artificial (IA) que permite que computadores e máquinas aprendam com as informações existentes e apliquem esse aprendizado para executar outras tarefas semelhantes. Sem programação explícita, a máquina aprende com os dados. A máquina capta ou aprende padrões, tendências ou recursos essenciais de dados anteriores e faz uma previsão em novos dados.

Como dito em (Ye, 2023), o aprendizado de máquina tem ganhado atenção nos últimos anos na comunidade de comunicação óptica devido à sua poderosa capacidade de aprender com dados usando algoritmos e estatísticas computacionais. Os algoritmos de aprendizagem de máquina são subdivididos em três categorias: aprendizado supervisionada, aprendizado não supervisionado e aprendizado por reforço. Os problemas de aprendizado de máquina podem ser subdivididos em quatro categorias: regressão, classificação, agrupamento e aprendizado por reforço. A Figura 2.12 ilustra as sub-divisões do aprendizado de máquina, além de listar tipos de problemas que cada sub-área pode lidar.

Figura 2.12: Sub-áreas de aprendizagem de máquina com exemplos de problemas de cada sub-área.

Fonte: Redes de saúde - Aprendizado de Máquina (Machine Learning)

2.10.1 Aprendizado supervisionado

O Aprendizado Supervisionado é uma técnica de aprendizado de máquina na qual um algoritmo é treinado utilizando um conjunto de dados rotulados. Nesse tipo de aprendizado, o objetivo é mapear os dados de entrada para os rótulos correspondentes, a fim de que o algoritmo seja capaz de prever corretamente os rótulos de novos exemplos não rotulados (Russell e Norvig, 2020). Cada exemplo de treinamento tem uma ou mais entradas e a saída desejada ou rótulo. Através da otimização iterativa de uma função objetivo, algoritmos de aprendizagem supervisionada aprendem uma função que pode ser usada para prever a saída associada a novas entradas (Mohri e Afshin Rostamizadeh, 2018). Os tipos de algoritmos de aprendizagem supervisionada incluem **aprendizagem ativa**, **classificação** e **regressão** (Alpaydin, 2014).

- Os algoritmos de classificação são utilizados quando as saídas são restritas a um conjunto limitado de valores.

- Os algoritmos de regressão são usados quando as saídas podem variar em

qualquer valor numérico dentro de um intervalo.

- Os algoritmos de aprendizagem ativa são uma abordagem especializada dentro do aprendizado supervisionado. Eles funcionam de forma semelhante aos algoritmos de classificação e regressão, mas com uma diferença fundamental: em vez de receber um conjunto fixo de dados de treinamento, o modelo de aprendizado ativo interage com um usuário para selecionar os exemplos mais informativos e relevantes para o treinamento do modelo.

2.10.2 Aprendizado não supervisionado

O aprendizado não supervisionado é um método em que os modelos são fornecidos com dados não rotulados para explorar e entender diferentes padrões e estruturas dos dados. O modelo então categoriza os dados em alguns grupos de acordo com as semelhanças e dissimilaridades nos padrões e estruturas de dados. A aprendizagem não supervisionada é fundamental para obter observações perspicazes a partir de padrões subjacentes. Além disso, esse método de aprendizado de máquina não requer entrada manual tediosa do usuário para rotular o conjunto de dados. Os tipos de algoritmos de aprendizado não supervisionado incluem *clustering*, redução de dimensionalidade e aprendizagem de associação(Hossain, 2024).

- Em *clustering* os padrões e estruturas de dados são explorados em busca de semelhanças e diferenças. Esse conhecimento é então utilizado para agrupar os dados em diversos conjuntos, chamados de *clusters*, de modo que os dados dentro de cada grupo sejam mais similares entre si do que com os de outros grupos. Esse processo de agrupamento é conhecido como clustering.

- A redução de dimensionalidade é um processo usado para reduzir o número de variáveis aleatórias sob consideração, mantendo o máximo de informações possível.

- O aprendizado por associação é uma técnica amplamente utilizada em diversas áreas, incluindo análise de cesta de mercado, bioinformática, mineração de uso

da web e detecção de intrusão. Essa técnica explora padrões e relacionamentos nos dados a partir do histórico de transações. Enquanto a filtragem colaborativa se concentra em identificar itens semelhantes para um único usuário, a regra de associação analisa os dados de transações de todos os usuários como um grupo, permitindo recomendar itens semelhantes em sites de comércio eletrônico.

2.10.3 Aprendizado por reforço

Os agentes de aprendizagem interagem com um ambiente dinâmico e aprendem a realizar ações que maximizam uma recompensa cumulativa. O aprendizado é orientado pela retroalimentação recebida do ambiente, incentivando o agente a descobrir uma política de ação ótima. O aprendizado por reforço tem aplicações no campo da robótica, carros autônomos, saúde, automação industrial, processamento de linguagem natural, finanças comerciais e controle de movimento de robôs (Hossain, 2024).

- O aprendizado por reforço se concentra em resolver problemas que envolvem tomar decisões sequenciais para maximizar uma recompensa cumulativa ao longo do tempo.

Um problema famoso no domínio de RL, é o problema do bandido multi armado, também conhecido como o problema do bandido armado. Esse problema representa muito bem o dilema Exploração versus Explotação na RL. Suponha que você more em uma cidade grande com muitos restaurantes. Você visitou alguns dos restaurantes, mas não todos. De todos os restaurantes que você visitou, você obtém a maior satisfação de alguns deles. Para alcançar a máxima satisfação, você pode ir a um restaurante que você já gosta ou visitar um novo com a esperança de que eles possam fazer melhor.(Hossain, 2024)

2.10.4 Algoritmos de aprendizado de máquina

Os algoritmos utilizados nesse trabalho estão dentro do campo do aprendizado supervisionado.

K-Nearest Neighbors (KNN)

O *K-nearest neighbor* (KNN) é um dos algoritmos de aprendizado supervisionado. Ele pode ser usado para resolver problemas de regressão e classificação. Dois adjetivos comuns do algoritmo KNN são que ele não é paramétrico (ou seja, não faz suposições subjacentes sobre a distribuição de dados) e um aprendiz preguiçoso (ou seja, não aprende imediatamente com o conjunto de dados de treinamento). A métrica mais comum utilizada para a avaliação de uma nova amostra no KNN é a distância euclidiana.

A distância euclidiana é medida a partir de pontos vizinhos "k"mais próximos ao ponto de dados que deve ser classificado/estimado. Por exemplo, entre os "K"pontos vizinhos mais próximos, se a maioria dos pontos vizinhos mais próximos pertencer à Classe A, então o ponto de dados é classificado como classe A. Para a regressão o valor atribuído a amostra será a média (ou qualquer outra métrica escolhida) entre os "K"vizinhos mais próximos (Hossain, 2024).

A distância euclidiana entre uma amostra de treino e teste com n atributos pode ser definida em (Dokmanic et al., 2015), como:

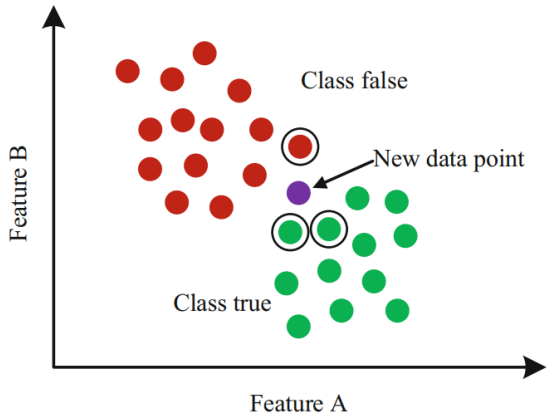
Seja P_n a n-ésima amostra conjunto de treino com n atributos, tal que $P_n = (a_1, a_2, ..., a_n)$ e Q a amostra a ser testada com n atributos, tal que $\mathrm{Q} = (t_1, t_2, ..., t_n)$, então a distância euclidiana pode ser calculada como:

$$D = \sqrt{(t_1 - a_1)^2 + (t_2 - a_2)^2 + ... + (t_n - a_n)^2} = \sqrt{\sum_{i=1}^{n} (t_i - a_i)^2} \tag{2.30}$$

O cálculo da distância é feito para todas as amostras e depois é selecionada as amostras que possuem o menor valor de distância. Em seguida é realizado uma

votação majoritária com os K vizinhos mais próximos, na qual o a classe atribuída à amostra de teste é a classe da maioria para a classificação, enquanto que para a regressão o valor estimado atribuído para a amostra de teste é a média dos valores dos K vizinhos mais próximos.

Figura 2.13: Ilustração do algoritmo KNN, com k=3, a amostra de teste (em roxo) será classificada como classe verde, pois a classe verde possui duas amostras mais próximas da amostra de teste enquanto a classe vermelha só possuí uma amostra mais próxima da amostra de teste.

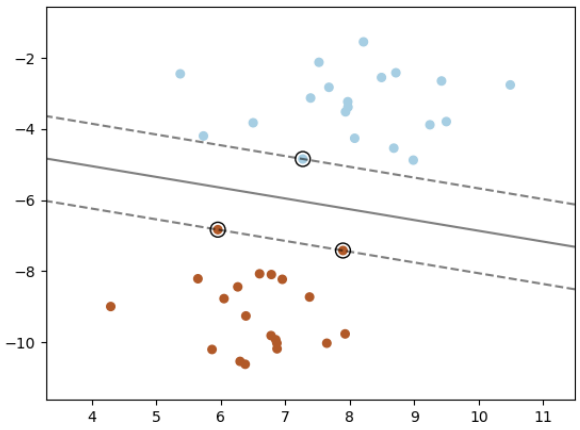


Fonte: Hossain, 2024

Support Vector Machine **(SVM)**

O *Support Vector Machine*, ou Máquina de Vetores de Suporte em português, é um algoritmo de aprendizado de máquina supervisionado. O SVM constrói um hiperplano ou conjunto de hiperplanos em um espaço de alta ou infinita dimensionalidade, que pode ser utilizado para classificação, regressão ou outras tarefas. Intuitivamente, uma boa separação é alcançada pelo hiperplano que possui a maior distância para os pontos de dados de treinamento mais próximos de qualquer classe (chamada de margem funcional), visto que, em geral, quanto maior a margem, menor o erro de generalização do classificador. A Figura 2.14 mostra a decisão para um problema linearmente separável, com três amostras no limites de margem, chamados de “vetores de suporte” (Hearst et al., 1998).

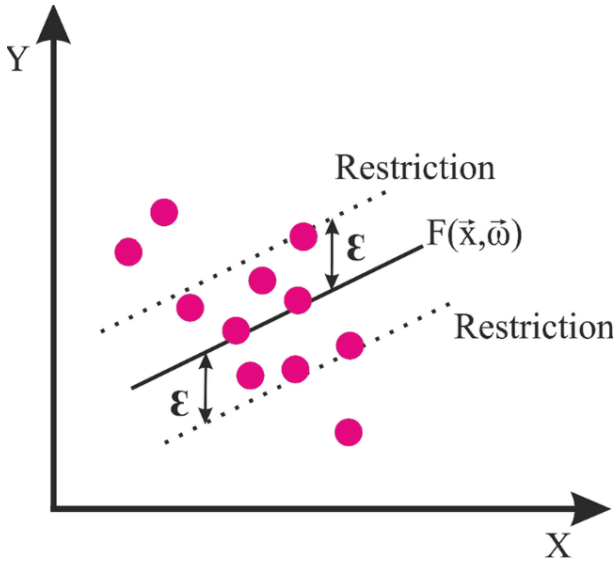
Figura 2.14: Ilustração de um problema linearmente separável em que é mostrado um limites de margem para a separação.



Fonte: Hearst et al., 1998

***Support Vector Machine Regression* (SVR)** O *Support Vector Regression* é aplicado a problemas de regressão, semelhante ao SVM lineares (2.10.4). O SVR busca encontrar um hiperplano que melhor se ajuste aos dados em um espaço contínuo. Isso é alcançado por meio do mapeamento das variáveis de entrada para um espaço de características de alta dimensão e da busca pelo hiperplano que maximize a margem entre ele e os pontos de dados mais próximos, ao mesmo tempo em que minimiza o erro de previsão. O SVR é capaz de lidar com relações não lineares usando funções de *kernel* para mapear os dados para espaços de dimensão superior, tornando-o uma ferramenta valiosa para tarefas de regressão com relações complexas entre as variáveis de entrada e a variável de destino.

Figura 2.15: Visão esquemática do *Support Vector Regression* (SVR).



Fonte: *Research Gate*

Em [(Hearst et al., 1998), (Chang e Lin, 2001)], o SVR é modelado pelo seguinte problema:

Dado os vetores de treinamento $x_i \in \mathbb{R}^p$, $i = 1, ..., n$ e a vetor $y \in \mathbb{R}^n$ em que C,$\varepsilon > 0$, ε-SVR resolve o seguinte problema primário:

$$\begin{aligned} &\min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ &\text{subject to } y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ &\qquad w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ &\qquad \zeta_i, \zeta_i^* \geq 0, i = 1, ..., n \end{aligned} \tag{2.31}$$

As amostras que estão a pelo menos um ε de distância são penalizadas, contribuindo para o objetivo por ζ_i ou ζ_i^*, dependendo se suas estimações estão acima ou abaixo do cilindro de raio ε.

O problema secundário dado por:

$$\min_{\alpha,\alpha^*} \frac{1}{2}(\alpha - \alpha^*)^T Q(\alpha - \alpha^*) + \varepsilon e^T(\alpha + \alpha^*) - y^T(\alpha - \alpha^*)$$
$$\text{subject to } e^T(\alpha - \alpha^*) = 0 \tag{2.32}$$
$$0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, ..., n$$

em que ϵ um vetor com todos elementos iguais a um, Q é uma matriz positiva semi-definida de ordem $n \times n$, $Q_{ij} \equiv K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ é uma função kernel (2.10.4). Aqui, os vetores de treinamento são implicitamente mapeados para um espaço dimensional mais alto (possivelmente infinito) pela função ϕ.

Após a resolução de (2.32), a estimação é dada por:

$$\sum_{i \in SV} (\alpha_i - \alpha_i^*) K(x_i, x) + b \tag{2.33}$$

Kernel A função *kernel* é usada para mapear os dados de entrada em um espaço de características de dimensão superior, permitindo assim lidar com relações não lineares entre as variáveis de entrada e a variável de destino.

Os tipos de *kernel* comumente usados no SVR incluem:

- Linear: Realiza um mapeamento linear dos dados para um espaço de características de dimensão superior.

$$K(x, y) = \gamma x^T y + c \tag{2.34}$$

- Polinomial: Mapeia os dados para um espaço de características de dimensão superior usando funções polinomiais, permitindo modelar relações não lineares mais complexas. Para polinômios de grau d, o *kernel* polinomial é definido, em (Padierna et al., 2018), como:

$$K(x, y) = (\gamma x^T y + c)^d \tag{2.35}$$

- Função de Base Radial (RBF): O *kernel* RBF é frequentemente usado devido à sua capacidade de capturar relações não lineares complexas. Ele mapeia os dados para um espaço de características de dimensão infinita usando uma função de base radial. O *kernel* RBF é definido, em (Thurnhofer-Hemsi et al., 2020), como:

$$K(x,y) = exp\Big(\frac{-|x-y|^2}{2\sigma^2}\Big) = exp\Big(-\gamma|x-y|^2\Big) \tag{2.36}$$

  em que, $\gamma = \frac{1}{2\alpha}$, $|x-y|^2$ pode ser entendido como a distância euclidiana entre x e y e γ é um parâmetro livre.

- Sigmoid: Mapeia os dados para um espaço de características de dimensão superior usando uma função sigmoide, que é útil para problemas de classificação binária, mas pode não ser tão comum em problemas de regressão. O *kernel Sigmoid* é definido, em (Lin e Lin, 2005), como:

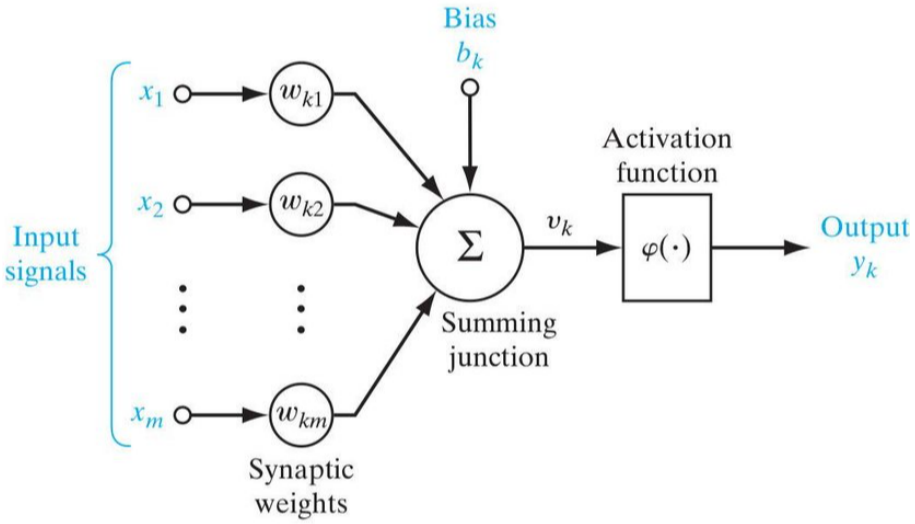
$$K(x,y) = tanh\Big(ax^T y + r\Big), \tag{2.37}$$

  em que a é um parâmetro de dimensão dos dados de entrada e r é o parâmetro de deslocamento que controla o limiar de mapeamento.

Artificial Neural Network (ANN)

As redes neurais artificiais (ANNs) são modelos de aprendizado de máquina inspirados na estrutura e no funcionamento do cérebro humano. A representação geral de um neurônio artificial simples é mostrada na Figura 2.16. As ANNs são popularmente conhecidas como redes *feed-forward* (retro-propagação). Uma RNA geralmente consiste de muitos neurônios, empilhados juntos em camadas específicas. Cada neurônio recebe um conjunto de entradas, realiza operações matemáticas nelas e produz uma saída que é transmitida para outros neurônios na rede. Um neurônio pode receber uma ou várias entradas e gera uma saída. Os pesos determinam a força da influência que um neurônio da camada anterior terá na camada posterior.

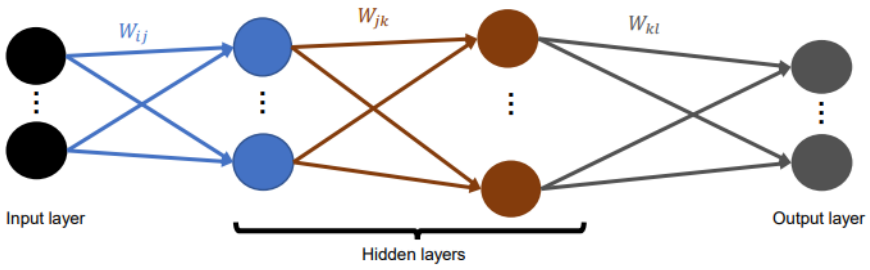
Figura 2.16: Modelo de neurônio artificial. Na qual, x_n são as entradas, w_{k_m} são os pesos sinápticos, b_k representa um ruído e φ representa uma função de ativação que é aplicado em v_k



Fonte: *Research Gate*

As camadas compostas por neurônios podem estar totalmente conectadas ou parcialmente conectadas. Uma ANN padrão deve ter uma camada de entrada, uma ou várias camadas ocultas e uma camada de saída.

Figura 2.17: Arquitetura de uma rede neural MLP generica, com duas camadas ocultas.



Fonte: Ye, 2023

A Figura 2.17 mostra um MLP totalmente conectado com uma camada de entrada, duas camadas ocultas e uma camada de saída. A saída da camada l pode ser calculada como em (Ye, 2023):

$$a^{(l)} = \varphi\left(z^{(l)}\right), \tag{2.38}$$

com

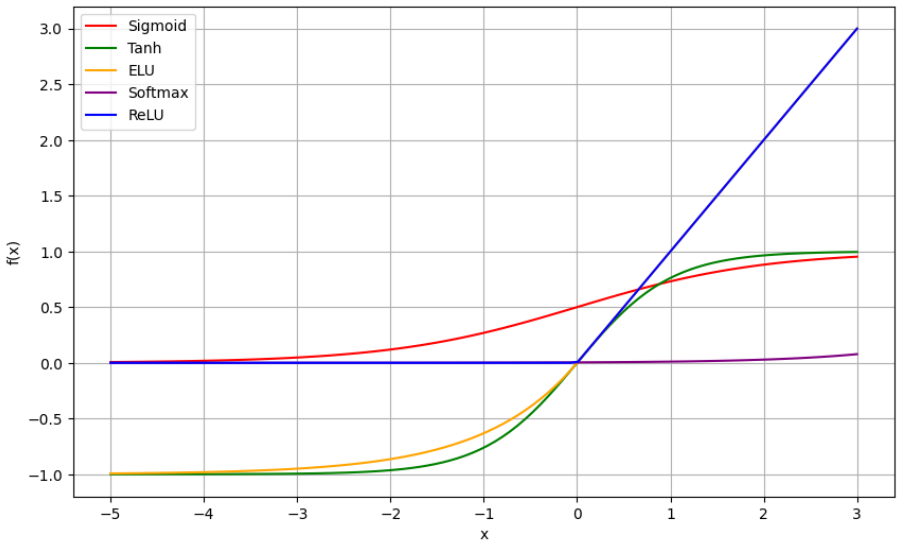
$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{2.39}$$

$z^{(l)}$ é a soma ponderada das entradas, $W^{(l)}$ os pesos, $b^{(l)}$ o viés e $\varphi(.)$ a função de ativação dos neurônios da camada l. A primeira camada é a camada de entrada então $a^{(0)} = x$ e a última camada, indexada como N_l, então $a^{(N_l)} = y$.

Funções de ativação A função de ativação em uma rede neural é responsável por introduzir não linearidades nas saídas dos neurônios, permitindo que a rede aprenda e represente relações complexas nos dados. Ela é aplicada após a combinação linear das entradas e pesos em cada neurônio e determina se o neurônio deve ”disparar”ou não, ou seja, se deve produzir uma saída ativada ou não ativada.

A Figura 2.18 ilustram a variação das funções de ativação de acordo com a entrada x.

Figura 2.18: Curvas das funções de ativação



Fonte: O autor

As funções de ativação mais comuns usadas em redes neurais incluem:

- Função de Ativação Linear Retificada (ReLU): A ReLU é uma função de ativação simples que retorna zero para valores negativos e o próprio valor para valores positivos. Ela é amplamente utilizada devido à sua simplicidade

e eficácia. A ativação ReLU é definida como:

$$f(x) = \begin{cases} x, & \text{se } x > 0 \\ 0, & \text{caso contrário} \end{cases} \tag{2.40}$$

- Sigmoid: A função sigmoid mapeia valores reais para o intervalo entre 0 e 1. Ela é usada em camadas de saída binárias ou para problemas em que a saída precisa ser interpretada como uma probabilidade. A ativação Sigmoid é definida como:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.41}$$

- Tangente Hiperbólica (tanh): A função tanh é semelhante à sigmoid, mas mapeia valores reais para o intervalo entre -1 e 1. Ela é comumente usada em camadas ocultas de redes neurais. A ativação tangente hiperbólica é definida como:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.42}$$

- Unidade Linear Exponencial (ELU): A ELU é uma alternativa à ReLU que produz um gradiente suave para valores negativos. Ela ajuda a evitar o problema de "neurônios mortos"durante o treinamento. A ativação ELU é definida como:

$$f(x) = \begin{cases} x, & \text{se } x > 0 \\ \alpha(e^{x} - 1), & \text{se } x \leq 0 \end{cases} \tag{2.43}$$

- Função de Ativação Softmax: A função softmax é usada na camada de saída de redes neurais para problemas de classificação multiclasse. Ela normaliza as saídas para que a soma de todas as saídas seja igual a 1, o que permite

interpretá-las como probabilidades. A ativação Softmax é definida como:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.44}$$

Etapa de treinamento O processo de treinamento de modelos de aprendizado de máquina envolve a escolha do modelo desejado e a inicialização dos parâmetros antes do treinamento com os dados. Os parâmetros, como pesos e vieses em redes neurais, são normalmente inicializados utilizando métodos como Xavier ou Kaiming. Os dados são divididos em conjuntos de treinamento, validação e teste, com o conjunto de treinamento usado para atualizar os parâmetros do modelo. A retropropagação, um algoritmo proposto em 1986, é usada para treinar redes neurais, com duas etapas: passagem para frente e passagem para trás. Durante a passagem para frente, as amostras de dados são alimentadas na rede para gerar previsões, e durante a passagem para trás, os gradientes são calculados e usados para atualizar os parâmetros da rede. Isso melhora gradualmente o desempenho do modelo (Ye, 2023).

O método de retro-propagação, amplamente utilizado para treinar redes neurais feed-forward, foi proposto em 1986 (Rumelhart et al., 1986). Esse algoritmo consiste em duas etapas: a passagem para frente e a passagem para trás. Durante a passagem para frente, cada amostra de dados de treinamento é alimentada na rede, passando por todas as camadas ocultas até chegar à camada de saída, que produz uma previsão. Em seguida, é calculada uma medida de perda para avaliar o desvio entre a previsão e o valor alvo, conhecido como verdade do solo, usando uma função de custo, como o erro quadrático médio (MSE) cuja definição é feita em 2.11.3 ou o erro médio absoluto (MAE) cuja definição é feita em 2.11.2. Na passagem para trás, os gradientes são calculados e usados para atualizar os parâmetros da rede, começando da camada de saída e retrocedendo, o que gradualmente melhora o desempenho da rede (Ye, 2023).

Diversos algoritmos de otimização são utilizados para atualizar os parâmetros durante a etapa de retro-propagação com o objetivo de minimizar a função de custo. Um dos mais básicos é a descida do gradiente, que envolve calcular os gradientes da função de perda em relação aos parâmetros e ajustar os parâmetros na direção oposta aos gradientes para reduzir a perda. As equações (2.45) a (2.46) representam os passos fundamentais desse algoritmo de atualização, onde $^{(l)}$ e $^{(L)}$ correspondem, respectivamente, ao erro na l-ésima camada e na camada de saída. Esses erros são calculados utilizando a regra da cadeia.

$$\delta^{(L)} = \nabla_{z^{(L)}} J = \nabla_{a^{(L)}} J \odot \varphi'(z^{(L)}) \tag{2.45}$$

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot \varphi'(z^{(l)}) \tag{2.46}$$

em que $\nabla_{z^{(L)}} J$ representa o gradiente da perda J em relação à soma ponderada $z^{(L)}$, φ' é a derivada da função de ativação e $\odot$ denota a multiplicação elementar. Ao combinar as equações (2.46) e (2.45), podemos calcular o erro $\delta^{(l)}$ para qualquer camada na rede. Começamos computando $\delta^{(L)}$, seguido por $\delta^{(L-1)}$, $\delta^{(L-2)}$ e assim por diante, retrocedendo através da rede, originando o termo de *backpropagation* (Ye, 2023). Além disso, os gradientes da perda em relação aos pesos e vieses são computados da seguinte maneira:

$$\nabla_{W^{(l)}} J = \nabla_{z^{(l)}} J \cdot {a^{(l-1)}}^T = \delta^{(l)} \cdot {a^{(l-1)}}^T \tag{2.47}$$

$$\nabla_{b^{(l)}} J = \nabla_{z^{(l)}} J = \delta^{(l)} \tag{2.48}$$

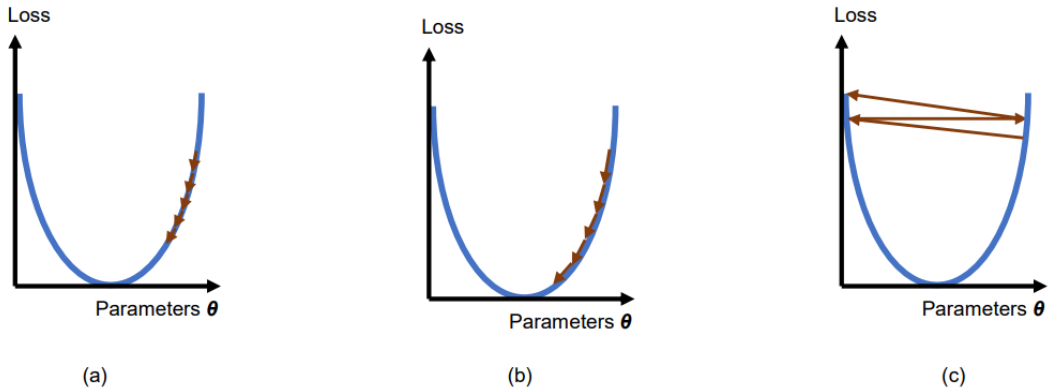
Os pesos e vieses são atualizados com um importante hiperparâmetro: o tamanho do passo η, chamado de taxa de aprendizado (LR - Learning rate), as equações (2.49) e (2.50) descrevem como os pesos e vieses são atualizados iterativamente.

$$W^{(l)} \leftarrow W^{(l)} - \eta \cdot \nabla_{W^{(l)}} J \tag{2.49}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \cdot \nabla_{b^{(l)}} J \tag{2.50}$$

A Figura 2.20 ilustra o impacto do LR durante as etapas de descida do gradiente.

Figura 2.19: (a) o LR é muito baixo, por isso requer mais passos antes de atingir o ponto mínimo; (b) aprendizagem eficiente com uma LR ótima; (c) grandes LR levam a oscilações.



Fonte: Ye, 2023

A escolha adequada do LR é crucial para o desempenho e a convergência eficiente do algoritmo de otimização, como o Gradiente Descendente.As implicações de um LR muito baixo, alto e normal são:

- **Taxa de Aprendizado Muito Baixa**:
  - Convergência Lenta
  - Susceptibilidade a Mínimos Locais
- **Taxa de Aprendizado Muito Alta**:
  - Oscilações e Divergência
  - Instabilidade Numérica
- **Taxa de Aprendizado Normal**:
  - Convergência Eficiente

  - Robustez
  - Equilíbrio entre Velocidade e Precisão

Além do algoritmo de descida de gradiente, existem vários outros algoritmos de otimização para atualização dos hiper parâmetros de um modelo de ANN. (Chollet, 2015) e (Abadi et al., 2015) implementam diversos algoritmos. Alguns desses algoritmos são:

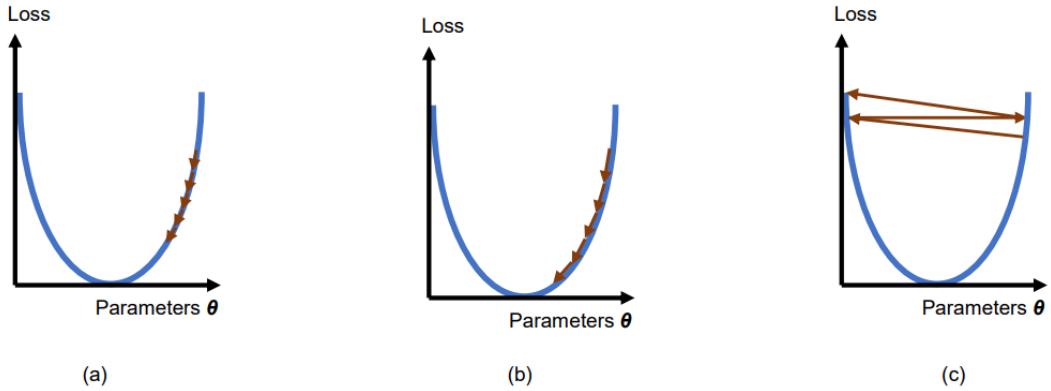
- ***Gradiente Descendente Estocástico (SGD)***: O Gradiente Descendente Estocástico é uma versão estocástica do gradiente descendente, onde os parâmetros do modelo são atualizados com base no gradiente calculado em um único exemplo de treinamento por vez. É eficiente para grandes conjuntos de dados, mas pode ser mais ruidoso que métodos mais avançados.
- **Gradiente Descendente com Momento (Momentum)**: O Gradiente Descendente com Momento adiciona uma componente de momento às atualizações dos parâmetros. Isso ajuda a acelerar o processo de treinamento e a evitar oscilações indesejadas, especialmente em terrenos com curvatura variável.
- **AdaGrad (*Adaptive Gradient Algorithm*)**: O AdaGrad adapta a taxa de aprendizado para cada parâmetro com base na frequência com que esse parâmetro é atualizado durante o treinamento. É eficaz para ajustar a taxa de aprendizado automaticamente, mas pode diminuir muito rapidamente, o que pode prejudicar o treinamento.
- **RMSprop (Root Mean Square Propagation)**: O RMSprop é uma variação do AdaGrad que mantém uma média móvel da magnitude dos gradientes recentes. Isso ajuda a suavizar a direção das atualizações dos parâmetros e a melhorar a convergência em direção ao mínimo global.
- **Adam (*Adaptive Moment Estimation*)**: O Adam combina os conceitos de momento e RMSprop. Além de adaptar a taxa de aprendizado para cada

parâmetro, ele utiliza momentos de primeira e segunda ordem dos gradientes para ajustar a direção e a magnitude das atualizações dos parâmetros de forma adaptativa.

- **Adadelta**: O Adadelta é uma variação do RMSprop que adapta a taxa de aprendizado ao longo do tempo, considerando uma janela deslizante das atualizações anteriores dos parâmetros. Isso ajuda a controlar a taxa de aprendizado de forma mais eficaz e a lidar com o problema de diminuição rápida da taxa de aprendizado.

- **Nadam (*Nesterov-accelerated Adaptive Moment Estimation*)**: O Nadam é uma variação do Adam que incorpora o método de Nesterov de momento em sua formulação. Isso proporciona uma convergência mais rápida em comparação com o Adam, especialmente em terrenos irregulares.

Figura 2.20: O modelo treinado (a) superajusta o conjunto de dados; b) Se encaixe bem; c) Insuficiente o conjunto de dados.

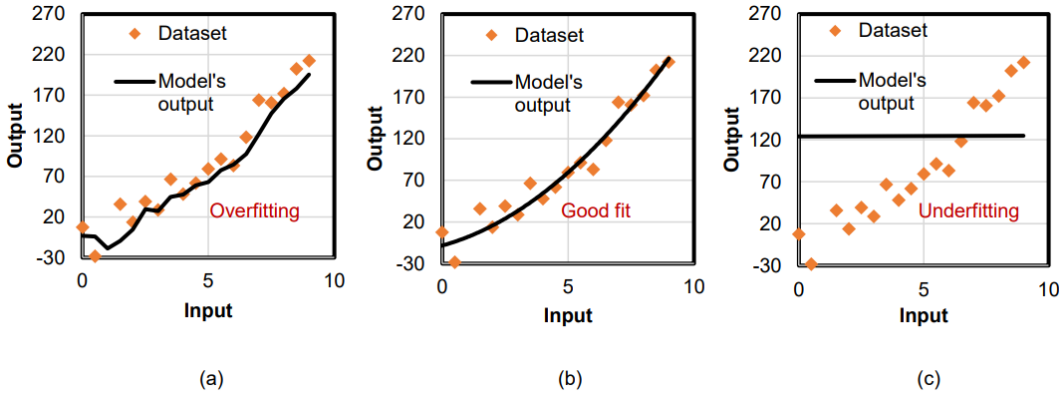


Fonte: Ye, 2023

Durante a etapa de *backpropagation*, podemos atualizar os parâmetros utilizando todo o conjunto de dados ou lotes do conjunto de dados. Um lote é um subconjunto do conjunto de dados usado em uma iteração. O processamento em lote oferece várias vantagens: para conjuntos de dados grandes, processar o conjunto inteiro pode ser computacionalmente caro, lento, e demandar uma grande quantidade de memória. Além disso, os mini lotes podem aproveitar ao máximo a computação paralela, especialmente ao utilizar hardware moderno, como GPUs e TPUs.

O embaralhamento aleatório antes de cada época de treinamento é benéfico, pois permite que o conjunto de dados seja misturado e dividido para reduzir viés e possíveis desequilíbrios de dados em lotes. A passagem para frente seguida pela etapa de atualização para trás sobre todo o conjunto de dados de treinamento é chamada de época. Como o conjunto de dados disponível nunca é de tamanho infinito, é necessário prestar atenção a dois problemas comuns, *overfitting* e *underfitting*. O *overfitting* leva a um modelo treinado que se ajusta muito bem aos dados de treinamento, mas tem um desempenho ruim com novos dados. A Figura 2.21 ilustra três cenários possíveis para treinar um modelo com o conjunto de dados fornecido. Quando o modelo se ajusta demais, o desempenho será ruim em dados de treinamento e dados não vistos, mas se o modelo se ajustar demais, o desempenho será muito bom em relação ao conjunto de dados de treinamento, mas não dará a melhor previsão para dados invisíveis.

Figura 2.21: O modelo treinado (a) superajusta o conjunto de dados; b) Se encaixa bem; c) Insuficiente o conjunto de dados.



Fonte: Ye, 2023

Para evitar o *overfitting*, podemos usar o conjunto de dados de validação para interromper o treinamento logo após atingir o ponto ideal, ou seja, o número de épocas que garantem os menores erros no conjunto de dados de validação e treinamento. Esse método é chamado de parada precoce. Como mostrado na Figura 2.21, após um certo número de épocas, se continuarmos treinando o modelo com o mesmo conjunto de dados, a perda ainda diminuirá para o conjunto de treinamento, mas se tornará plana ou mesmo começará a aumentar para o conjunto de dados de

validação (Schmidt, 2023).

Outra forma de evitar o *overfitting* é utilizar um método de regularização, adicionando um termo de regularização na função de custo ou utilizando outras técnicas como o abandono, em que os neurônios e as conexões da rede são selecionados aleatoriamente para serem ignorados com uma determinada probabilidade, durante uma época da fase de treinamento. Isso ajuda a evitar o problema de co-adaptação e torna a rede mais robusta.

Por outro lado, o *underfitting* ocorre quando o modelo não aprendeu corretamente o mapeamento entre a entrada e a saída. Essa situação leva a um baixo desempenho de previsão tanto em dados de treinamento quanto em novos dados. Isso pode acontecer, por exemplo, quando o modelo de ML não é complexo o suficiente para resolver o problema ou quando os hiperparâmetros não são otimizados, ou ainda quando o conjunto de dados usado é inconsistente, o comportamento do *underffiting* pode ser observado em (c) na Figura 2.21.

A maneira mais simples de evitar o *underfitting* é aumento a complexidade do modelo, ajustar os hiperparâmetros de forma adequada, escolher uma função de custo apropriada, aplicar técnicas de regularização e, se necessário, aumentar o volume de dados de treinamento. Ao seguir essas estratégias, é possível desenvolver modelos mais robustos e capazes de generalizar bem para novos dados (Bashir et al., 2020).

Os hiper parâmetros de uma rede neural são parâmetros cujos valores não são aprendidos durante o treinamento do modelo, ao contrário dos pesos e vieses. Em vez disso, eles são definidos antes do treinamento e afetam diretamente o comportamento e o desempenho do modelo de uma rede neural. Alguns exemplos comuns de hiper parâmetros de redes neurais incluem:

- **Taxa de Aprendizado (*Learning Rate*)**: Determina a magnitude das atualizações dos pesos durante o treinamento.
- **Número de Épocas (*Number of Epochs*)**: O número de vezes que todo o conjunto de treinamento é passado pela rede neural durante o treinamento.

- **Tamanho do Lote (*Batch Size*)**: O número de exemplos de treinamento processados simultaneamente pela rede neural em cada iteração durante o treinamento.
- **Arquitetura da Rede**: Inclui o número de camadas, o número de neurônios em cada camada, a função de ativação em cada neurônio, entre outros.
- **Função de Ativação**: Define como os neurônios nas camadas ocultas produzem uma saída a partir da soma ponderada das entradas.
- **Inicialização dos Pesos**: Define como os pesos da rede neural são inicializados antes do treinamento.
- **Regularização**: Ajuda a evitar o sobreajuste durante o treinamento, introduzindo penalidades nos pesos da rede.
- **Otimizador**: Define o algoritmo de otimização usado para atualizar os pesos da rede durante o treinamento.

2.11 Métricas de avaliação de modelos de regressão

Nesta seção vamos considerar a seguinte condição enunciada abaixo para introduzir as equações das métricas de avaliação.

Seja X conjunto de dados tem n valores rotulados como $Y_{true} = y_1, ..., y_n$, cada um associado a um valor estimado pelos modelos $Y_{pred} = (\hat{y}_1, ..., \hat{y}_n)$.

2.11.1 R-Quadrado (R^2, R-Dois ou Coeficiente de determinação)

O coeficiente de determinação, representado por R^2 ou r^2 e pronunciado como "R ao quadrado", indica a proporção da variação na variável dependente que pode ser explicada pelas variáveis independentes. Ele oferece uma medida de quão bem

o modelo reproduz os resultados observados, com base na fração da variação total dos resultados explicada pelo modelo. Existem diversas definições de R^2 que, ocasionalmente, são equivalentes. Um exemplo é a regressão linear simples, onde r^2 é utilizado em vez de R^2. Quando apenas um intercepto é incluído, r^2 é simplesmente o quadrado do coeficiente de correlação da amostra (ou seja, r) entre os resultados observados e os valores preditores observados. Se regressores adicionais são incluídos, R^2 é o quadrado do coeficiente de correlação múltipla. Em ambos os casos, o coeficiente de determinação normalmente varia de 0 a 1 (van Ginkel, 2019). O R^2 é definido como:

$$R^2 := 1 - \frac{SS_{res}}{SS_{total}}, \tag{2.51}$$

em que SS_{res} é definido como:

$$SS_{res} := \sum_{i}^{n} (y_i - \hat{y})^2, \tag{2.52}$$

e o SS_{total} é definido como:

$$SS_{total} := \sum_{i}^{n} (y_i - \bar{y})^2, \tag{2.53}$$

em que $\bar{y}$ representa a média do conjunto.

2.11.2 Erro Médio Absoluto (MAE - do inglês *Mean Absoluto Error*)

O Erro Médio Absoluto (MAE), também conhecido como Mean Absolute Error em inglês, é uma medida de avaliação comum em problemas de regressão. Ele mede a média das diferenças absolutas entre os valores previstos por um modelo e os valores reais. Em termos simples, o MAE indica quão próximo as previsões do modelo estão dos valores reais em média. Quanto menor o valor do MAE, melhor o desempenho do modelo em fazer previsões precisas. O MAE é fácil de interpretar,

pois está na mesma escala das unidades de medição dos dados, e é menos sensível a valores discrepantes em comparação com outras medidas de erro, como o erro quadrático médio (MSE). O MAE pode ser definido como em (Wang e Lu, 2018).

$$MAE := \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.54}$$

2.11.3 Erro Quadrático Médio (MSE - do inglês *Mean Squared Error*)

O Erro Quadrático Médio (MSE), ou Mean Squared Error em inglês, é uma medida comum de avaliação em problemas de regressão. Ele calcula a média dos quadrados das diferenças entre os valores previstos por um modelo e os valores reais. Em essência, o MSE quantifica o quanto as previsões de um modelo diferem dos valores reais ao quadrado, penalizando mais fortemente as grandes discrepâncias. Quanto menor o valor do MSE, melhor o desempenho do modelo em fazer previsões precisas. O MSE é amplamente utilizado devido à sua propriedade matemática conveniente e à capacidade de refletir tanto a precisão quanto a dispersão das previsões. No entanto, ele pode ser sensível a valores extremos (outliers) nos dados, pois os quadrados dessas diferenças ampliam seu impacto no cálculo do erro médio. O MAE pode ser definido como em (Botchkarev, 2018).

$$MSE := \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.55}$$

2.11.4 Raiz do Erro Quadrático Médio (RMSE - do inglês, *Root Mean Squared Error*)

O RMSE é uma medida comum de avaliação em problemas de regressão semelhante ao MSE. No entanto, o RMSE tem a vantagem de fornecer uma interpretação

direta na mesma unidade de medida dos dados originais. Ele calcula a raiz quadrada do MSE, o que significa que ele representa a média das diferenças entre os valores previstos por um modelo e os valores reais, mas em uma escala mais intuitiva. O RMSE é útil para fornecer uma medida de desempenho que é facilmente compreendida em termos do erro médio absoluto entre as previsões e os valores reais. Assim como o MSE, quanto menor o valor do RMSE, melhor o desempenho do modelo em fazer previsões precisas. Ele é especialmente útil quando os dados têm unidades de medida específicas e a interpretação direta do erro é importante. O RMSE é pode ser definido como:

$$RMSE := \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.56}$$

2.11.5 Validação cruzada

É essencial garantir que os modelos de ML sejam capazes de lidar com qualquer parte do conjunto de dados que, teoricamente, representaria a realidade do contexto na qual o modelo foi concebido, embora, como sabemos, teoria e prática nem sempre estejam alinhadas. Para evitar problemas de generalização, a validação cruzada é indispensável ao lidar com modelos de previsão.

A técnica de validação cruzada é uma abordagem fundamental na avaliação da capacidade de generalização de um modelo, especialmente em problemas de previsão. O conceito central das técnicas de validação cruzada é o particionamento do conjunto de dados em subconjuntos mutuamente exclusivos, e posteriormente, o uso de alguns destes subconjuntos para a estimação dos parâmetros do modelo (dados de treinamento), sendo os subconjuntos restantes (dados de validação ou de teste) empregados na validação do modelo. Existem várias maneiras de realizar essa divisão dos dados, sendo as três mais comuns: holdout, k-fold e leave-one-out (Isaksson et al., 2008).

Método Holdout No método Holdout, os dados são divididos em dois grupos exclusivos: um para treinamento (para estimar os parâmetros) e outro para teste (para validação). A divisão pode ser igual ou desigual, sendo comum usar 2/3 dos dados para treinamento e 1/3 para teste. Após essa separação, o modelo é estimado e, em seguida, testado com os dados de teste para calcular o erro de predição. Essa abordagem é preferível quando há uma grande quantidade de dados disponíveis. No entanto, se o conjunto de dados for pequeno, a variabilidade no erro de predição pode ser significativa.

Método K-fold O método k-fold consiste em dividir o conjunto total de dados em k subconjuntos mutuamente exclusivos de tamanho igual. Em cada iteração, um desses subconjuntos é designado como conjunto de teste, enquanto os k-1 restantes são usados para treinamento e estimativa de parâmetros. Essa divisão é repetida k vezes, alternando circularmente o subconjunto de teste. Ao final das iterações, a acurácia do modelo é calculada com base nos erros encontrados, fornecendo uma medida mais confiável da capacidade do modelo de representar o processo gerador dos dados.

Método leave-one-out O método leave-one-out é uma variação do k-fold, onde k é igual ao número total de dados N. Neste método, é realizado um cálculo de erro para cada dado individualmente. Embora forneça uma análise completa da variação do modelo em relação aos dados, essa abordagem é computacionalmente custosa e geralmente é recomendada quando há poucos dados disponíveis.

Capítulo 3

Desenvolvimento

Este capítulo é desenvolvido a partir das seguintes etapas:

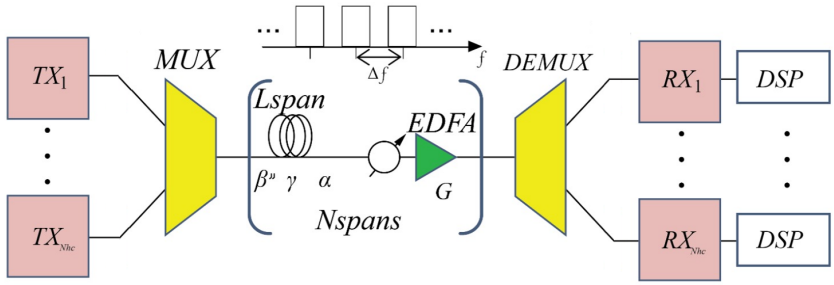
- Geração dos dados sintéticos utilizando o modelo NLIN;
- Introdução ao *dataset*;
- Pré-processamento dos dados;
- Implementação dos modelos;
- Definição das métricas de avaliação.

3.1 Geração de dados sintéticos utilizando o modelo NLIN

O *dataset* é gerado a partir do modelo NLIN descrito na subseção 2.9 utilizando a implementação do modelo em MATLAB disponibilizado em (da Silva et al., 2017) (Anexo A), considerando o sistema WDM ilustrado na Figura 3.1, em que é composto por um bloco contendo os transmissores TX_i e um multiplexador, um bloco contendo um demultiplexador os receptores RX_i, e entre ambos os blocos tem-se conjunto de N *spans*, na qual cada *span* é composto por fibra, atenuador óptico variável (VOA) e um amplificador, conforme ilustrado na Figura 3.1.

As configurações para as simulações com o modelo são as mesmas que em (Filho, 2017) com algumas modificações, pois para gerar um grande volume de dados é preciso fazer com que algumas variáveis variassem em um conjunto de valores. Essas simulações foram utilizadas para computar a BER, BER_{NL}, OSNR, $OSNR_{dB_{NL}}$, $NLIN$ e a potência ótima do canal (PCh_{Opt}), além das características específicas da rede considerada, como por exemplo o comprimento do spam, número de canais, número de *spans*, potência de entrada, modulação dos canais, parâmetros da fibra considerada, entre outros.

Figura 3.1: Configuração do Sistema de Comunicação Óptica



Fonte: FILHO, M. J. C., 2017. Adaptado

As configurações básicas para a simulação descrita em (Filho, 2017) foram adaptadas para que correspondesse ao que é utilizado neste trabalho. O ruído ASE gerado pelos amplificadores ópticos é adicionado ao sinal ao longo do enlace, incluindo o efeito do ruído de fase não-linear. Para garantir um número maior de possibilidades são utilizados três fibras para a geração dos dados, cujos parâmetros estão dispostos no Quadro 1. Os enlaces ópticos testados foram considerados com fibras de mesmo tipo, transparentes e sem compensação de dispersão ao longo do enlace. Nos trechos, o VOA (atenuador óptico variável) é ajustado para 10 dB e sua função é simular as perdas dos nós e limitar a BER em 10^{-4} dB.

Foram utilizados amplificadores EDFA tendo seu ganho configurado para compensar as perdas totais do trecho e do VOA. A figura de ruído dos amplificadores EDFA é configurada para 6 dB. Foram utilizados $N spans$, cada um com L km de comprimento de fibra, com $N_{spans} = 5, 6, ..., 15; L = 80, 90, ..., 120$. O número

de canais considerados é N_{ch}, com $N_{ch} = 5, 7, ..., 15$, na qual é considerado que a modulação dos canais eram iguais, podendo ser DP-QPSK - 112 Gbit/s (ou 28 GBaud/s), DP-16QAM - 224 Gbit/s (ou 28 GBaud/s) ou DP-64QAM - 336 Gbit/s (ou 28 GBaud/s), ou seja, caso a simulação seja feita para a modulação DP-QPSK, temos que 112 Gbit/s WDM foram transmitidos com pulsos de Nyquist com espaçamento entre canais, Δf, de 50 GHz. O número ímpar de canais é escolhido com o objetivo de isolar o canal central, que será o canal avaliado. Cada canal é configurado com a mesma potência de lançamento e os N_{ch} canais foram simulados com a potência de lançamento por canal (P_{ch}) variando de -3 a 5 dBm. Essas configurações foram aplicadas para as as fibras SSMF, NZDSF e PSCF cujos parâmetros estão dispostos na Tabela 1. O ruído de fase do laser é ajustado para zero e os filtros gaussianos de 4ª ordem com largura de banda de 42 GHz no Tx e Rx foram utilizados. O número de pontos de integração de Monte-Carlo é definido como 1.000.000 para garantir que o erro relativo fosse bem inferior a 1%.

A combinação entre os possíveis valores de L, N_{spans}, N_{ch}, formato de modulação, tipo de fibra, comprimento por *span*, potência de lançamento por canal resultou em um conjuto de dados contendo cinquenta mil quatrocentos e oitenta e oito (50488) amostras.

3.2 Introdução ao *dataset*

A partir da configuração de simulação discutida em (3.1) é gerado um grande volume de dados. O dados do *dataset* são compostos por 50488 amostras e 25 atributos, dos quais 6 são referentes atributos de saída. Os atributos são subdivididos em, atributos de entrada constantes, zerados e atributos categóricos.

3.2.1 Atributos de entrada

Os atributos listados abaixo são constantes em todo *dataset* e podem possivelmente servir como entrada para os algoritmos de ML:

- **lightspeed** - Velocidade da luz com valor de 0.299792458×10^9 m/s;
- **lambda** - Comprimento de onda de referência com valor de 1,55 μm;
- **ChSpacing** - Espaçamento do canal com valor de 50 GHz;
- **BaudRate** - Taxa de baud com valor de 28 GHz;
- **Spans_DeltaPdBIntLeft** - Variação da potência de entrada dos canais à direita do canal central, com valor 0 dBm;
- **Spans_DeltaPdBIntRight** - Variação da potência de entrada dos canais à esquerda do canal central, com valor 0 dBm;
- **Spans_SpanLossdB** - Perda do Span com valor 0 dB;

Os atributos listados abaixo foram variados a fim de gerar uma maior quantidade e variedade de dados, para posteriormente serem utilizados como entradas para os algoritmos de ML.

- **Spans_Fn** - Figura de ruído com valor de 16 dB.
- **DispPar** - Parâmetro de dispersão;
- **Spans_gamma** - Coeficiente de não-linearidade;
- **Spans_alpha** - Coeficiente de atenuação;
- **Spans_beta2** - Coeficiente de dispersão, dependente do DisPar.
- **NumCh** - Número de canais, variando de 5 até 15 com passo 2;
- **NumSpans** - Número de Spans, variando de 5 até 15 com passo 1;

- **Spans_L** - Comprimento do Span em km, variando de 80 até 120 com passo 10 km;
- **Spans_PdBmCh** - Potência média de entrada dos canais em dBm, variando de -3 até 5 com passo 0,5 dBm;
- **Spans_ModFormatCh** - Modulação do canal avaliado (central);
- **Spans_ModFormatIntLeft** - Modulação dos canais à esquerda do canal central;
- **Spans_ModFormatIntRight** - Modulação dos canais à direita do canal central.

Parâmetros da fibra, DispPar, γ, α estão dispostos na Tabela 2.1, ressaltando que cada parâmetro da fibra está diretamente aos demais. O parâmetro β_2 é calculado em função de DispPar. As modulações consideradas variam entre DP-QPSK, DP-16QAM e DP-64QAM, sem a combinação delas entre os canais.

3.2.2 Atributos de saída

Os atributos listados abaixo são resultados do modelo NLIN e posteriormente podem possivelmente servir como atributo *target* (de saída) para os algoritmos de ML:

- **PChOptdBm** - Potência ótima da rede em dBm.
- **NLIN_Power** - Potência do ruído com interferência não-linear em dBm.
- **OSNRdB** - Relação Sinal-Ruído em dB.
- **OSNRdB_NL** - Relação Sinal-Ruído com interferência não-linear em dB.
- **BER** - Taxa de erro de bits.
- **BER_NL** - Taxa de erro de bits com interferência não-linear.

3.3 Pré-processamento dos dados

O pré-processamento de dados é uma etapa fundamental em muitas aplicações de análise de dados e aprendizado de máquina. Refere-se ao conjunto de técnicas e operações aplicadas aos dados brutos antes de serem utilizados em modelos analíticos ou algoritmos de aprendizado de máquina. Existem 3 principais passos envolvidos neste processo: limpeza de dados, transformação de dados e redução de dados.

3.3.1 Transformação de atributo por meio da criação de novos atributos

A transformação de dados é uma etapa essencial no pré-processamento de dados, que visa modificar a estrutura ou distribuição dos dados brutos de forma a torná-los mais adequados para análise, modelagem estatística e treinamento de modelos de IA. Por exemplo a transformação de um atributo categórico em um atributo número.

Neste trabalho o formato da modulação dos canais, central, à esquerda do canal central e à direita do canal central foram utilizados transformados de variáveis categóricas (texto) para valores numéricos. O mapeamento do formato de modulação é realizado através da função:

$$f(x) = \begin{cases} 1/4, & \text{se } x = \text{“PM-QPSK”} \\ 1/8, & \text{se } x = \text{“PM-16QAM”} \\ 1/12, & \text{se } x = \text{“PM-64QAM”} \end{cases} \tag{3.1}$$

Este mapeamento faz com que seja possível levar em conta o impacto da modulação dos canais no treinamento dos modelos de machine learning.

Na implementação do modelo SVR com *kernel* polinomial é realizada a normalização dos dados. A normalização aplicada aos dados consiste em subtrair a média dos dados e dividir pela variância. A função que realiza essa normalização é dada por:

$$z_i = \frac{x_i - \mu}{\sigma^2}, \tag{3.2}$$

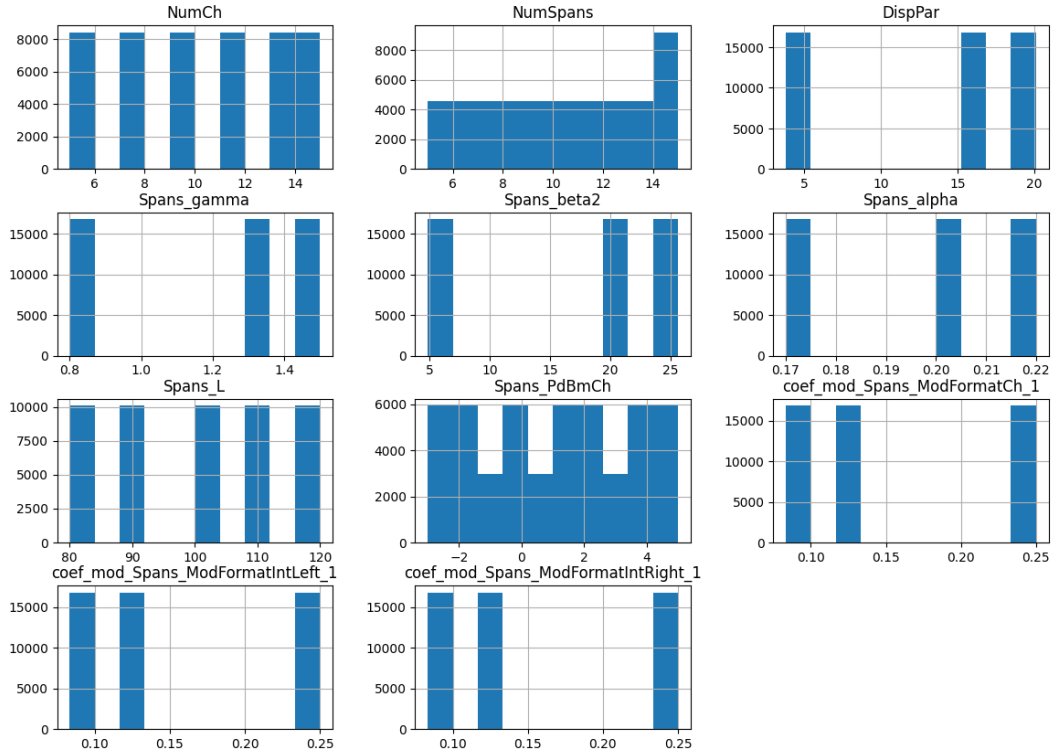
em que x_i é o i-ésimo elemento do conjunto de dados, μ é a média, σ^2 é a variância e z_i corresponde ao valor da amostra após a normalização do atributo em questão.

3.3.2 Redução de dados

A redução dos dados se deu por meio da seleção dos atributos. Inicialmente o conjunto de dados é composto por um total de 50488 amostras, com cada amostra tendo de 25 atributos. Os atributos removidos do conjunto de dados foram os atributos constantes em todas as amostras, atributos com valor 0 em todas as amostras, restando somente 15 atributos, sendo 11 atributos de entrada e 4 possíveis saídas foram selecionados.

A Figura 3.2, ilustra a distribuição dos atributos selecionados por sua faixa de variação para os atributos de entrada que foram selecionados para treinamento e avaliação dos modelos de machine learning.

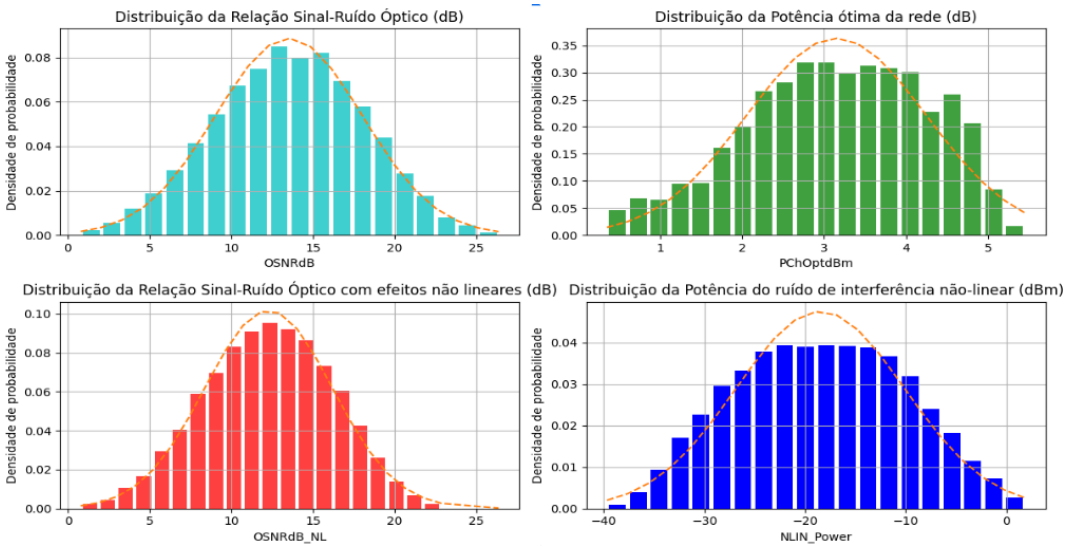
Figura 3.2: Distribuição dos atributos de entrada selecionados



Fonte: O autor

A Figura 3.3, ilustra a distribuição dos atributos de saída selecionados, tais saídas podem servir de objetivo de estimação dos algoritmos de ML.

Figura 3.3: Distribuição dos atributos de saída.



Fonte: O autor

3.3.3 Divisão dados

A divisão dos dados é uma etapa importante no processo de implementação de algoritmos de ML, pois envolve a separação do conjunto de dados disponível em diferentes conjuntos que serão usados para treinamento, teste do modelo.

A divisão dos dados é feita utilizando o método Holdout, ver 2.11.5, na qual 30% dos dados é separado para teste dos modelos e 70% dos dados é separado para o treinamento dos modelos. Este processo é realizado utilizando a função ***train_-tast_split*** do pacote ***model_selection*** da biblioteca ***scikit-learn*** do ***Python***, na qual disponibiliza uma série de implementações desde modelos, calculo de métricas e utilitários, como essa função que separa os dados em treino e teste (Pedregosa et al., 2011).

O conjunto de treinamento é usado para treinar o modelo. Ele contém exemplos de entrada (atributos) e as saídas correspondentes (rótulos) que o modelo deve aprender a prever. O modelo ajusta seus parâmetros com base nos dados deste conjunto.

O conjunto de teste é usado para avaliar o desempenho final do modelo após o treinamento ser concluído. Ele contém exemplos que o modelo não viu durante o treinamento. Avaliar o modelo com dados não vistos ajuda a verificar se ele tem um desempenho bem para novos dados e fornece uma estimativa mais realista do desempenho do modelo.

3.4 Implementação dos modelos de aprendizado de máquina

Nesta subseção trataremos da implementação dos algoritmos *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM)/*Support Vector Regression* (SVR) e *Artificial Neural Network* para regressão. Os algoritmos KNN e SVR foram implementados em *Python* utilizando a biblioteca *Scikit-learn*. Os modelos de KNN são implementados pelo *Scikit-learn* no pacote *neighbors* na qual tem uma variedade de

implementações, no caso deste trabalho a implementação utilizada é a ***KNeighborsRegressor***, pois o escopo de estimação/predição são problemas de regressão. Para o SVM a implementação é feita utilizando o pacote **svm**, pacote que implementa uma série de algoritmos de SVM, a implementação utilizada neste trabalho é o SVR, na qual é a versão para problemas de regressão do SVM. O modelo de rede neural artificial é implementado utilizando a biblioteca Tensorflow (Abadi et al., 2015) por meio da *Application Programming Interface* (API) Keras (Chollet, 2015), o Keras é uma API de alto nível do TensorFlow para criar e treinar modelos de aprendizado profundo. Ele é usado para prototipagem rápida, pesquisa avançada e produção (Chollet, 2015).

As bibliotecas necessárias para a implementação dos algoritmos, desde o pré-processamento dos dados até a visualização dos resultados são:

- **Pandas**: O Pandas é uma biblioteca de código aberto usada para manipulação e análise de dados em *Python*. Criado por Wes McKinney em 2008, sendo amplamente utilizado na comunidade de ciência de dados e análise (Harris et al., 2020).

- **NumPy**: NumPy abreviação de *Numerical Python*, é uma poderosa biblioteca do *Python* para computação numérica. é criado em 2005 por Travis Oliphant, a NumPy se tornou uma das mais importantes bibliotecas do ecossistema de computação científica em *Python* (Harris et al., 2020).

- **Matplotlib**: Matplotlib é uma biblioteca em *Python* para criação de gráficos e visualizações de dados de alta qualidade. Foi criada por John D. Hunter em 2003 e se tornou uma das bibliotecas mais utilizadas para visualização de dados em *Python* (Hunter, 2007).

- **Scikit-learn**: A scikit-learn (originalmente scikits.learn) é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python. Ela inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, gradient bo-

osting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy (Pedregosa et al., 2011).

- **Keras**: O Keras é uma biblioteca de rede neural de código aberto escrita em Python. Ele é capaz de rodar em cima de TensorFlow, Microsoft Cognitive Toolkit, R, Theano, ou PlaidML. Projetado para permitir experimentação rápida com redes neurais profundas, ele se concentra em ser fácil de usar, modular e extensível (Chollet, 2015).

- **Tensorflow**: TensorFlow é uma biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. É um sistema para criação e treinamento de redes neurais para detectar e decifrar padrões e correlações, análogo (mas não igual) à forma como humanos aprendem e raciocinam. O Tensorflow é usado tanto para a pesquisa quanto produção no Google, e está aos poucos substituindo seu antecessor de código proprietário, DistBelief. TensorFlow foi desenvolvido pela equipe Google Brain para uso interno na empresa. Foi lançado sob a licença de código aberto Apache 2.0 em 9 de novembro de 2015 (Abadi et al., 2015).

3.4.1 Implementação do algoritmo KNN

Para implementar os modelos de KNN, é necessário a importação do pacote **sklearn.neighbors** que contém a classe KNeighborsRegressor. A instância do KNeighborsRegressor pode receber uma grande quantidade de parâmetros, a lista de parâmetros de entrada pode ser acessada em sklearn.neighbors.KNeighborsRegressor, porém neste trabalho para a implementação dos modelos o parâmetro utilizado é:

- n_neighbors: Número de vizinhos considerados (Pedregosa et al., 2011).

Para os restantes dos parâmetros é utilizado o valor padrão da instância da classe. As etapas de implementação, supondo que os dados já foram divididos entre treino e teste como na subseção 3.3.3, de um modelo são:

- **Instância da Classe:**

  - Inicialmente deve-se criar um objeto knn *regressor* instanciando a classe `KNeighborsRegressor`.
  - Deve-se escolher o número de vizinhos (k) e outros hiper parâmetros conforme necessário.
  - Exemplo:

```
from sklearn.neighbors import KNeighborsRegressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
```

- **Treinamento do Modelo:**
  - O treinamento do modelo é feito utilizando o método `fit` passando como argumento o conjunto de dados do treinamento (atributos de entrada, rótulo).
  - Exemplo:

```
knn_regressor.fit(X_train, y_train)
```

    em que X_train é um *array* multi-dimensional, ou um DataFrame e y_train representa a saída desejada, neste trabalho as saídas utilizadas foram $OSNR_{dB_{NL}}$ ou $NLIN_{Power}$.

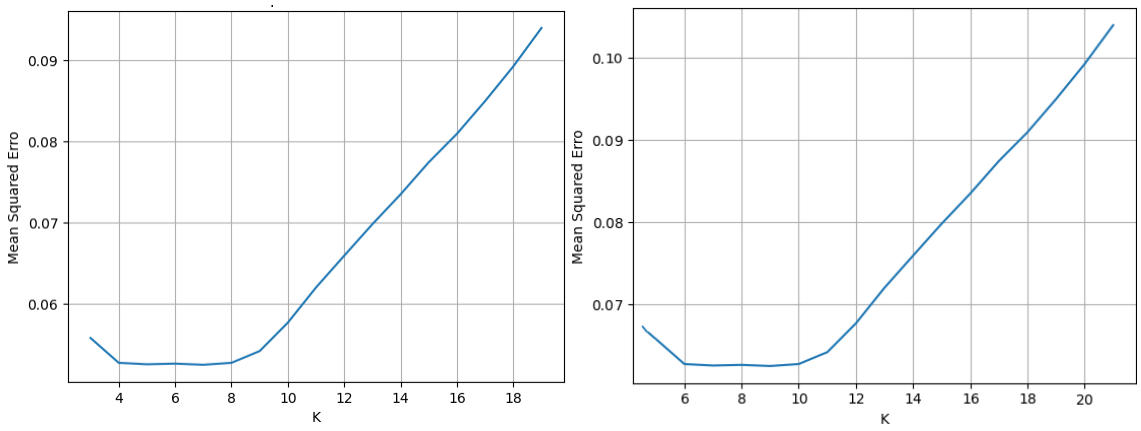
- **Estimação:**
  - A estimação dos valores de saída é feita utilizando o modelo treinado e o método `predict`, que deve receber como parâmetro o conjunto de dados de entrada.
  - Exemplo:

```
y_pred = knn_regressor.predict(X_test)
```

O exemplo utiliza os dados teste pare fazer a predição, que posteriormente é utilizado para avaliação dos modelos.

Foram realizadas sucessivas simulações com o valor de K variando de 2 a 22 para identificar qual seria o melhor valor de K para estimar a $OSNRdb_{NL}$ e o $NLIN_{Power}$, para ambos os modelos é computado o erro quadrático médio (MSE), o resultado é plotado e a Figura 3.4 ilustra a curva.

Figura 3.4: Em (a) temos o MSE vs K para o KNN-$OSNRdB_{NL}$ e em (b) temos MSE vs K para o modelo KNN-$NLIN_{Power}$



(a) (b)

Fonte: O autor

Observa-se então que um bom valor para K para ambos os modelos podem ser valores entre 4 e 8 para ambos os modelos. Os modelos de KNN para estimar a $OSNRdB_{NL}$ e o $NLIN_{Power}$ foram implementados utilizando k = 5.

3.4.2 Implementação do algoritmo SVR

Para implementar os modelos de SVR, é necessário a importação do pacote **sklearn.svm** que contém a classe SVR. A instância do SVR pode receber uma grande quantidade de parâmetros, a lista de parâmetros de entrada pode ser acessada em sklearn.svm.SVR, porém neste trabalho para a implementação dos modelos os parâmetro utilizados foram:

- *kernel*: Especifica o tipo de kernel a ser usado no algoritmo (Pedregosa et al., 2011). Os tipos de *kernels* utilizados foram descritos na subseção 2.10.4.

- *degree*: Grau da função *kernel* polinomial (poly). Deve ser não-negativo e é ignorado por todos os outros *kernels*.
- coef0: Termo independente na função kernel. Só é significativo para os *kernels* poly e sigmoid (Pedregosa et al., 2011).
- epsilon: Épsilon no modelo épsilon-SVR. Especifica o tubo épsilon dentro do qual nenhuma penalidade está associada na função de perda de treinamento com pontos previstos a uma distância épsilon do real valor. Deve ser não-negativo (Pedregosa et al., 2011).

Kernel RBF

As etapas de implementação, supondo que os dados já foram divididos entre treino e teste como na subseção 3.3.3, de um modelo de SVR para o kernel RBF são:

- **Instância da Classe:**
  - Inicialmente deve-se criar um objeto de *regressor* SVR usando a classe `SVR` do pacote `svm`.
  - Os hiperparâmetros são escolhidos conforme necessário. O tipo de kernel utilizado é o RBF 2.10.4, enquanto que os demais parâmetros, como por exemplo de regularização (C) e o parâmetro de largura de banda do kernel (*gamma*) foram utilizados com o valor padrão.
  - Exemplo:

```
from sklearn.svm import SVR
svr_regressor = SVR(
                    kernel='rbf',
                    C=1.0,
                    gamma='scale'
)
```

- **Treinamento do Modelo:**
  - O treinamento do modelo SVR é realizado a partir do método `fit`, método que recebe como parâmetro os dados de treino (atributos de entrada, rótulo).
  - Exemplo:

```
svr_regressor.fit(X_train, y_train)
```

    em que X_train é um *array* multi-dimensional ou um DataFrame e y_train representa a saída desejada, neste trabalho as saídas utilizadas foram $OSNR_{dB_{NL}}$ ou $NLIN_{Power}$.

- **Previsão:**
  - As estimações são feitas com o método `predict`, que recebe como parâmetro o conjunto de dados de que corresponde as características (atributos) utilizados como entrada no treinamento.
  - Exemplo:

```
y_pred = svr_regressor.predict(X_test)
```

    este exemplo realiza a estimação para os dados teste par, que posteriormente é utilizado para avaliação do modelo.

Kernel Polinomial

As etapas de implementação, supondo que os dados já foram divididos entre treino e teste como na subseção 3.3.3, de um modelo de SVR para o kernel Polinomial são:

- **Instância da Classe:**

- Inicialmente deve-se criar um objeto de regressor SVR usando a classe `SVR` do pacote `svm`.
- Os hiperparâmetros são escolhidos conforme necessário. O tipo de kernel utilizado é o Poly 2.10.4, o grau do polinômio (*degree*) com valor de 8 e o coeficiente independente (coef0) com valor de $1,25$ e épsilon com valor de $0,01$, enquanto que os demais parâmetros, como por exemplo de regularização (C) e o parâmetro de largura de banda do kernel (*gamma*) foram utilizados com o valor padrão.
- Exemplo:

```
from sklearn.svm import SVR
svr_regressor = SVR(
                    kernel='poly',
                    C=1,
                    epsilon=0,01,
                    degree=8,
                    gamma='scale',
                    coef0=1,25
)
```

- **Treinamento do Modelo:**
  - O treinamento do modelo SVR é realizado a partir do método `fit`, método que recebe como parâmetro os dados de treino (atributos de entrada, rótulo).
  - Exemplo:

```
svr_regressor.fit(X_train, y_train)
```

em que X_train é um *array* multi-dimensional ou um DataFrame e y_train

representa a saída desejada, neste trabalho as saídas utilizadas foram $OSNR_{dB_{NL}}$ ou $NLIN_{Power}$.

- **Previsão:**
  - As estimações são feitas com o método `predict`, que recebe como parâmetro o conjunto de dados de que corresponde as características (atributos) utilizadas como entrada no treinamento.
  - Exemplo:

```
y_pred = svr_regressor.predict(X_test)
```

este exemplo realiza a estimação para os dados teste para que posteriormente seja utilizado para avaliação do modelo.

Os modelos de SVR foram implementados com os parâmetros $kernel = rbf$; $C = 1$ e $gamma =$ '$scale$' para o *kernel* RBF, $kernel = poly$; $C = 1$; $epsilon = 0,01$; $degree = 8$; $gamma = scale$ e $coef0 = 1,25$ para o *kernel* Polinomial, respectivamente.

3.4.3 Implementação dos modelos de ANN

Para implementar os modelos de ANN, foi escolhido a biblioteca TensorFlow (Abadi et al., 2015) com a API Keras (Chollet, 2015). Primeiramente é necessário importar as bibliotecas do TensorFlow para criar e treinar o modelo neural. Essas bibliotecas incluem o *Sequential* para criar o modelo sequencial, *Dense* para adicionar camadas densas (totalmente conectadas), AdamW para definir os parâmetros do otimizador AdamW. A seguir é exibido como foi implementado os modelos de ANNs para a realização deste trabalho.

- **Definição da Arquitetura da Rede Neural:**
  - A arquitetura da rede neural é escolhida empiricamente como 4 camadas, na qual, a primeira é a camada de entrada (a camada de entrada não

é explicitamente definida no exemplo), as duas camadas seguinte são as camadas ocultas e a terceira é a camada de saída. A rede neural é implementada usando o modelo sequencial do Keras e as camadas densas.

- As funções de ativação das camadas de entrada e escondida foram ReLu e Sigmoid. Essas funções são descritas em 2.10.4 e o número de neurônios em cada camada é escolhido conforme necessário.
- Exemplo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import AdamW


model = Sequential([
    Dense(
        64, activation='relu',
        input_shape=train_shape
    ),
    Dense(64, activation='sigmoid'),
    Dense(1)
])
```

- **Treinamento do Modelo:**
  - O modelo é compilado para que as configurações sejam aplicadas no objeto. A compilação é feita pelo `método` compile que pode receber vários parâmetros, ver tf.keras.Sequential.
  - O modelo é treinado usando o método `fit`, especificando o número de épocas, tamanho do lote (*batch size*), e a proporção de validação.
  - O *callback EarlyStopping* é utilizado para parar o treinamento prematuramente se não houver melhorias na perda de validação.

– Exemplo:

```
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=25
)
optimizer = AdamW(learning_rate=0.001)


model.compile(
            optimizer=optimizer,
            loss='mse',
            metrics=['mae', 'mse']
)
history = model.fit(X_train, y_train,
                    epochs=1000,
                    batch_size=64,
                    validation_split=0.3,
                    callbacks=[early_stop]
)
```

A variável *history* armazena os dados de treinamento, que podem servir para plotar a evolução do erro ao longo das épocas.

- **Estimação dos Valores:**
  - Use o modelo treinado para fazer previsões sobre o conjunto de teste com o método `predict`.
  - Exemplo:

```
y_pred = model.predict(X_test)
```

Foram realizadas múltiplas simulações para a ANN modificando os otimizadores,

a fim de encontrar o otimizador que tivesse os melhores resultados para as métricas. A Tabela 3.1 ilustra os resultados dos otimizadores.

Tabela 3.1: Desempenho dos otimizadores

Otimizador	R^2	MSE	MAE
Adam	0,998	0,025	0,160
SGD	0,000	15,152	3,893
RMSprop	0,995	0,067	0,258
AdamW	0,999	0,002	0,045
Adadelta	0,743	3,420	1,849
Adagrad	0,967	0,468	0,684
Adamax	0,999	0,009	0,093
Nadam	0,998	0,031	0,176
Ftrl	0,961	0,550	0,741

A partir dos resultados para os otimizadores é possível identificar que o AdamW teve o melhor desempenho. O que indica que ele teve um desempenho melhor em minimizar o erro durante o treinamento em comparação com os outros otimizadores. Os modelos de ANN são implementados de acordo com a arquitetura do exemplo em 3.4.3.

3.5 Avaliação do desempenho

A avaliação do desempenho dos modelos é realizada usando métricas adequadas para problemas de regressão, como erro médio quadrático (MSE), erro médio absoluto (MAE), raiz do erro médio quadrático (RMSE) e coeficiente de determinação (R^2), a subseção 2.11.5 detalha cada uma dessas métricas.

Avaliação do Modelo:

- A avaliação do desempenho dos modelos é realizada usando métricas adequadas para problemas de regressão, como erro médio quadrático (MSE), erro

médio absoluto (MAE), raiz do erro médio quadrático (RMSE) e coeficiente de determinação (R^2), a subseção 2.11 detalha cada uma dessas métricas.

- Exemplo:

```
from sklearn import metrics


MSE = metrics.mean_squared_error(y_test, y_pred)
MAE = metrics.mean_absolute_error(y_test, y_pred)
RMSE = metrics.root_mean_squared_error(y_test, y_pred)
R² = metrics.r2_score(y_test, y_pred)
```

A avaliação dos modelos também é realizada utilizando a validação cruzada k-fold, descrita em 2.11.5, a implementação do k-fold é feita utilizando a biblioteca (Pedregosa et al., 2011) com as funções que calculam as métricas, da classe Kfold, na qual é implementado no pacote *sklearn.model_selection*. Os passos para realizar a avaliação do modelo são

- Divisão dos Dados:
  - A primeira etapa da implementação do k-fold é escolher um valor para k para que o conjunto de dados seja dividido em k partes iguais, ou "folds".
  - Cada fold é representado por um subconjunto dos dados.
  - Por exemplo, se temos 1000 amostras e escolhemos $k = 5$, cada fold terá 200 amostras.
- Treinamento e Teste:
  - Em cada iteração, um fold é retido como conjunto de teste, enquanto os k-1 folds restantes são usados como conjunto de treinamento, como é discutido em 2.11.5.

  - Isso é repetido k vezes, de forma que cada fold seja utilizado somente uma vez como conjunto de teste exatamente uma vez.

- Avaliação dos modelos:

  - A avaliação do desempenho dos modelos é realizada usando métricas adequadas para problemas de regressão, como erro médio quadrático (MSE), erro médio absoluto (MAE), raiz do erro médio quadrático (RMSE) e coeficiente de determinação (R^2), a subseção 2.11 detalha cada uma dessas métricas.

  - Exemplo:

```
from sklearn import metrics


MSE = metrics.mean_squared_error(y_test, y_pred)
MAE = metrics.mean_absolute_error(y_test, y_pred)
RMSE = metrics.root_mean_squared_error(y_test, y_pred)
R² = metrics.r2_score(y_test, y_pred)
```

- Média das métricas:

  - Ao final das k iterações, as métricas de desempenho registradas em cada fold são agregadas para fornecer uma estimativa final do desempenho do modelo.

  - Isso pode incluir a média, mediana, desvio padrão ou outras medidas estatísticas das métricas de desempenho em cada fold.

Neste trabalho as métricas utilizadas para avaliação dos modelos são descritas em 2.11.

Capítulo 4

Resultados

Neste tópico são apresentados os resultados da estimação da qualidade de transmissão em um sistema ponto-a-ponto, por meio dos algoritmos de aprendizado de máquina, utilizando a base de dados sintética gerada por meio do modelo EGN de Propagação não-linear.

4.1 Resultados dos modelos de aprendizagem de máquina

Este trabalho se concentrou em avaliar a estimação da relação sinal-ruído óptica considerando efeitos não lineares ($OSNR_{dB_{NL}}$) e potência do ruído não linear ($NLIN_{Power}$) considerando *link* ponto-a-ponto.

As estimações da $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ são realizadas utilizando os algoritmos *K-Nearest Neighbors* (KNN), *Support Vector Regression* (SVR) e *Artificial Neural Network* (ANN). O desempenho dos modelos é avaliado por meio das métricas R^2, MSE, RMSE e MAE, detalhados na seção 2.11. Também é realizada a avaliação do desempenho em relação à generalização utilizando um conjunto de validação, a fim de determinarmos se o modelo é capaz de generalizar para qualquer combinação de entrada, mantendo um erro em relação ao valor real próximo de zero.

4.1.1 *K-Nearest Neighbors* (KNN)

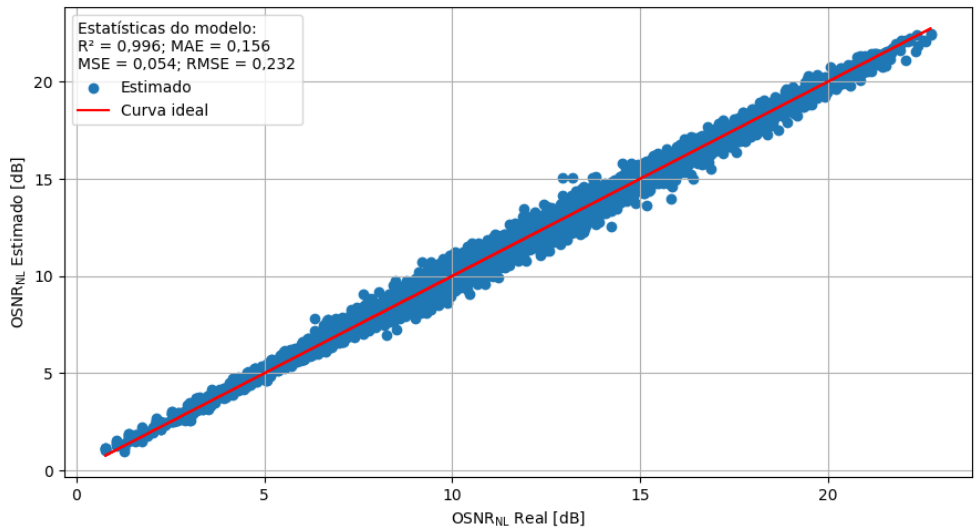
Ao avaliar os resultados da implementação dos modelos de KNN, observamos que o desempenho é satisfatório para ambas as saídas consideradas. Os modelos de $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ obtiveram um R^2 de $99,6\%$ e $99,3\%$ respectivamente, indicando uma boa capacidade de explicar a variabilidade nos dados, ou seja, ambos os modelos tendem a se ajustarem bem às amostras de teste (2.11.1). Além disso, as métricas de erros MAE, MSE e RMSE foram menores que $0,5$, demonstrando uma boa precisão na estimação. Como pode ser observado na Tabela 4.1.

Tabela 4.1: Resultados obtidos pela validação cruzada para os modelos KNN para o conjunto de teste que possui cerca de 30% dos dados do *DataSet*.

Entrada	Saída	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,996	0,156	0,054	0,232
$N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$NLIN_{Power}$ [dB]		0,993	0,570	0,513	0,716

O R^2 da $OSNR_{dB_{NL}}$ pode ser visualizado graficamente na Figura 4.1, na qual a distribuição das estimações em função do valor real tende a se aproximarem da curva ideal.

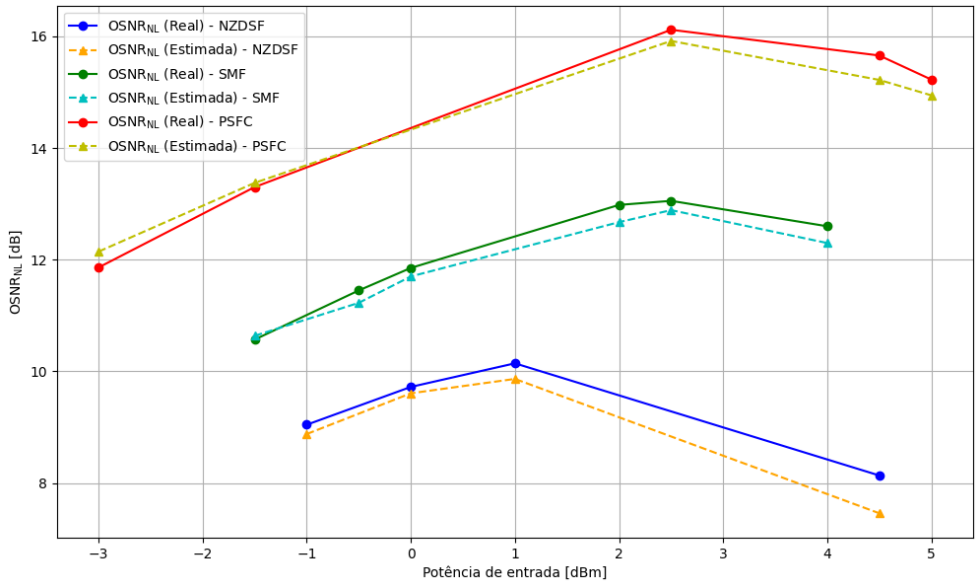
Figura 4.1: $OSNR_{dB_{NL}}$ real versus $OSNR_{dB_{NL}}$ estimado. Algoritmo KNN, resultados do conjunto de teste.



Fonte: O autor

O impacto de erro para a $OSNR_{dB_{NL}}$ da tabela Tabela 4.1 pode ser visualizado graficamente na Figura 4.2, onde é possível verificar a proximidade dos valores estimados e reais.

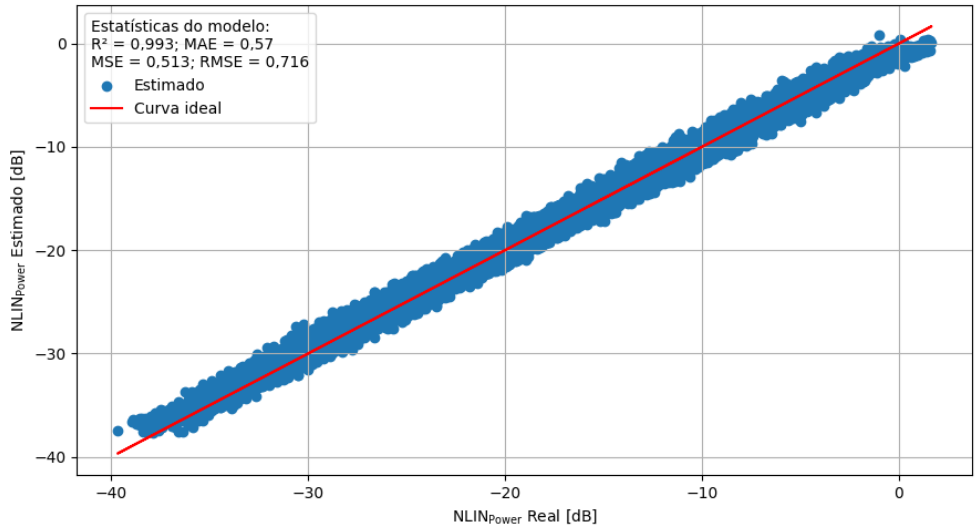
Figura 4.2: $OSNR_{\mathrm{dB,NL}}$ versus potência de entrada. Algoritmo KNN, resultados do conjunto de teste. Parâmetros fixados: $L_{\mathrm{Span}} = 90\,\mathrm{km}$, $N_{\mathrm{Ch,Span}} = 15$, $N_{\mathrm{Spans}} = 15$ e Modulação: QPSK cujo $coef_{\mathrm{Mod}} = \frac{1}{4}$.



Fonte: O autor

O R^2 da $NLIN_{Power}$ pode ser visualizado graficamente na Figura 4.3. Ao analisarmos o gráfico, notamos que a distribuição das estimações em função do valor real tende a se aproximarem da curva ideal, porém, com um grau de espalhamento maior devido a um valor de erro mais expressivo para algumas amostras.

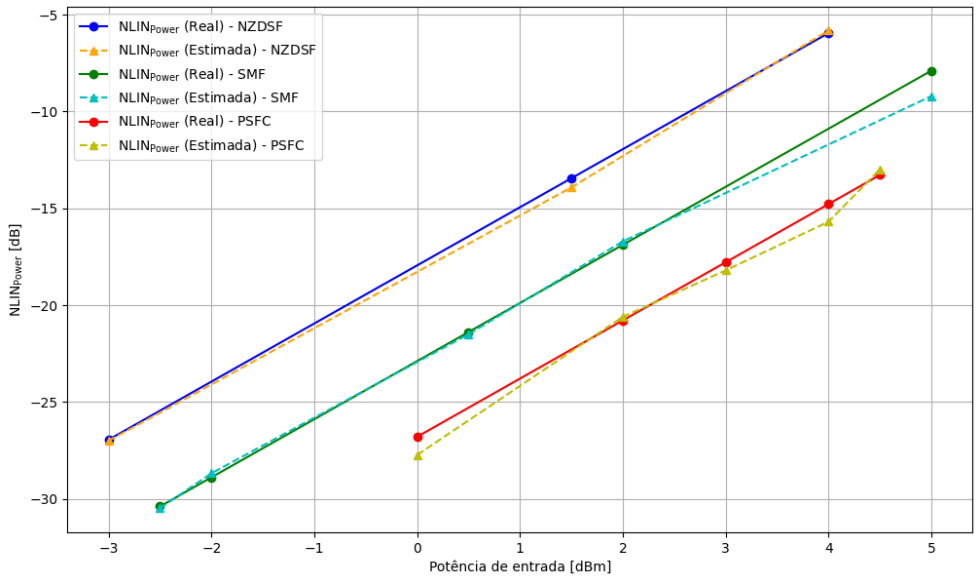
Figura 4.3: $NLIN_{Power}$ real versus $NLIN_{Power}$ estimado. Algoritmo KNN, resultados do conjunto de teste.



Fonte: O autor

A influência dos valores das métricas de erro para a $NLIN_{Power}$ observadas na Tabela 4.1, podem ser visualizadas graficamente na Figura 4.4, em que é possível verificar a proximidade dos valores estimados e reais.

Figura 4.4: $NLIN_{Power}$ versus potência de entrada. Algoritmo KNN, resultados do conjunto de teste. Parâmetros fixados: $L_{\text{Span}} = 120$ km, $N_{\text{Ch\_Span}} = 11$, $N_{\text{Spans}} = 7$ e Modulação: 16QAM cujo $\text{coef}_{\text{Mod}} = \frac{1}{8}$.



Fonte: O autor

Apesar dos resultados dos modelos de estimação da $OSNR_{dB_{NL}}$ e do $NLIN_{Power}$

se mostrarem promissores para os dados de teste, ao realizarmos a estimação da $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ para o conjunto de validação, ligeiramente diferente dos conjuntos de treino e teste, observou-se uma queda no desempenho dos modelos.

Os resultado de R^2 para os modelos de estimação da $OSNR_{dB_{NL}}$ e do $NLIN_{Power}$ do conjunto de validação foram de $74,2\%$ e $87,\%$ respectivamente, indicando uma degradação na explicabilidade dos dados pelos modelos.

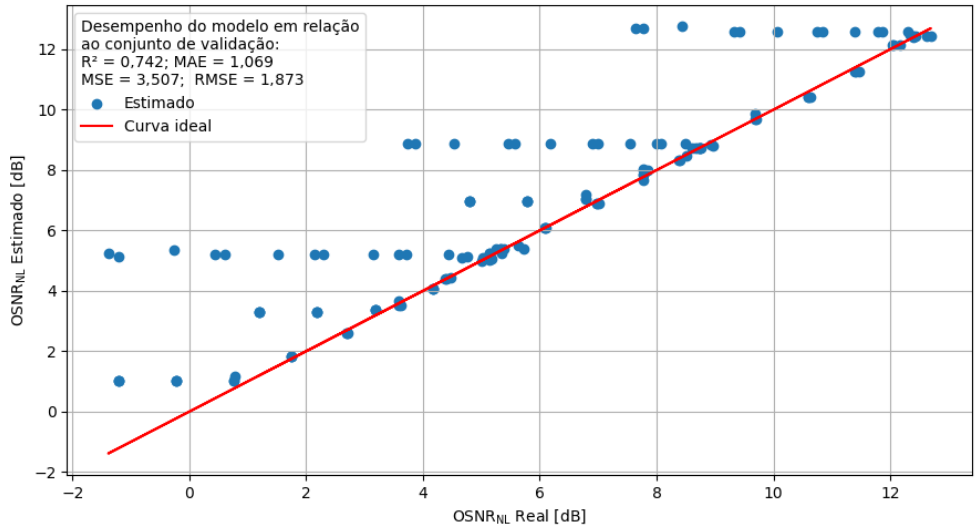
As métricas de MAE, MSE e RMSE alcançaram valores significativamente altos, com o MSE atingindo picos de $3,507$ e $23,12$ para os modelos de estimação de $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ respectivamente. Conforme evidenciado na Tabela 4.2.

Tabela 4.2: Resultados obtidos para o conjunto para validação. Algoritmo KNN.

Entrada	Saída	Fonte de Dados	Desempenho			
			R²	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,742	1,069	3,507	1,873
$N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$NLIN_{Power}$ [dB]		0,873	3,060	23,120	4,808

O resultado da queda do valor do R^2 para o modelo de estimação da $OSNR_{dB_{NL}}$ pode ser observado visualmente gráfico da Figura 4.5, na qual é possível verificar um maior espalhamento das estimações.

Figura 4.5: $OSNR_{dB_{NL}}$ estimado vs $OSNR_{dB_{NL}}$ real. Algoritmo KNN, resultados do conjunto de validação.

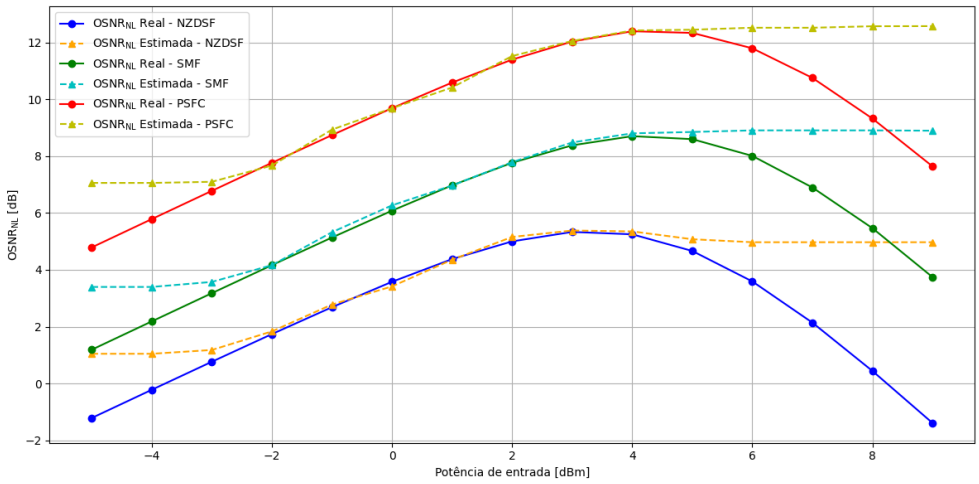


Fonte: O autor

O valor de erro obtido para o modelo de estimação da $OSNR_{dB_{NL}}$ do conjunto

de validação observadas na Tabela 4.1, podem ser visualizadas graficamente na Figura 4.2, onde é possível verificar que, para o intervalo de **potência de entrada** $[-3, 5]$, o erro de estimação da $OSNR_{dB_{NL}}$ é extremamente baixo. No entanto, nos intervalos $[-5, -3[$ e $]5, 9]$, o modelo começa a demonstrar erros mais consideráveis em comparação com os valores reais. Além disso, observa-se que os valores estimados se tornam praticamente constantes para potências de entrada menores que -4 [dBm] e maiores que 5 [dBm], enquanto a curva real continua em uma tendência de queda.

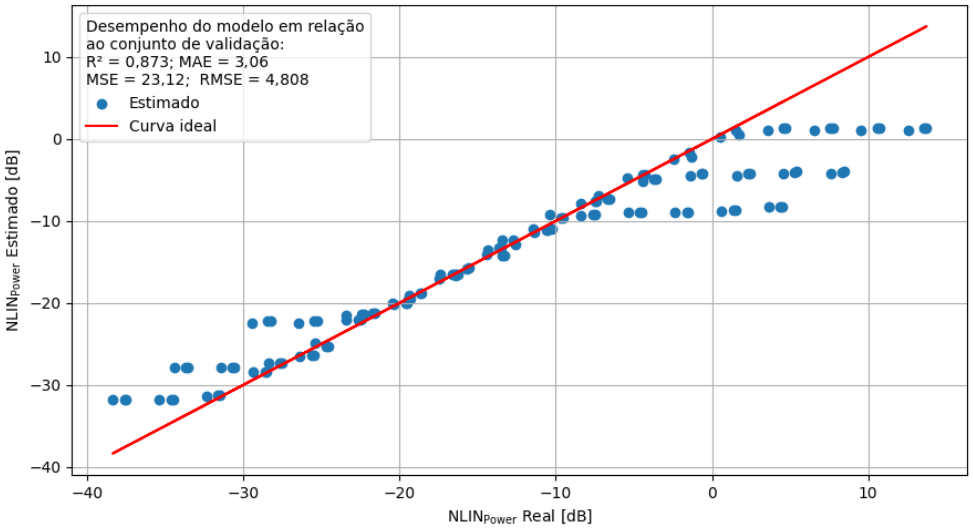
Figura 4.6: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo KNN, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulação : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

O resultado da queda do valor do R^2 para o modelo de estimação da $NLIN_{Power}$ do conjunto de validação pode ser observado visualmente no gráfico da Figura 4.7, na qual é possível verificar que uma parte dos valores estão espalhados, porém, a maior parte continua tendendo a se agrupar em torno da curva ideal.

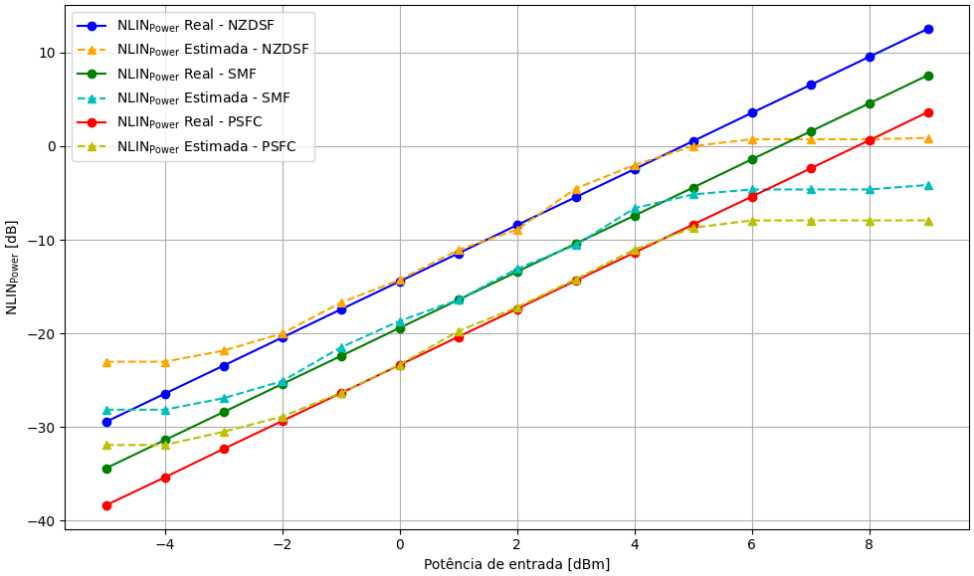
Figura 4.7: $NLIN_{Power}$ real vs $NLIN_{Power}$ estimado. Algoritmo KNN, resultados do conjunto de validação.



Fonte: O autor

O valor de erro obtido para o modelo de estimação da $NLIN_{Power}$ do conjunto de validação observadas na Tabela 4.2, podem ser visualizadas graficamente na Figura 4.8, em que verifica-se que, para o intervalo de **potência de entrada** $[-3, 5]$, o erro de estimação da $OSNR_{dB_{NL}}$ é extremamente baixo. No entanto, o modelo de estimação do $NLIN_{Power}$ se comporta como o modelo de estimação da $OSNR_{dB_{NL}}$, na qual as estimações realizadas para valores de potência fora do intervalo [-3, 5] tendem a apresentar um erro muito maior em relação ao valor real. Além disso, observa-se que os valores estimados se tornam praticamente constantes para potências de entrada fora do intervalo [-3, 5], enquanto que a curva dos valores reais continua em uma tendência de queda.

Figura 4.8: $NLIN_{Power}$ versus potência de entrada. Algoritmo KNN, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulação : QPSK$ cujo $coef_{Mod} = \frac{1}{4}$.



Fonte: O autor

4.1.2 *Support Vector Regression* (SVR)

Neste subtópico iremos tratar da análise dos resultados e avaliação do desempenho do algoritmo *Support Vector Regression*, implementado utilizando dois tipos diferentes de funções *Kernel*. O ***Radial Basis Function*(RBF)** e **Polynomial (Poly)**, definidos em 2.10.4.

***Kernel* RBF**

Ao avaliar os resultados da implementação dos modelos de SVR com *kernel* RBF observamos que, o desempenho é satisfatório para ambos modelos. Os modelos de $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ obtiveram um R^2 de $95,1\%$ e $99,7\%$ respectivamente, indicando uma boa capacidade de explicar a variabilidade nos dados, ou seja, ambos os modelos tendem a se ajustarem bem as amostras de teste (2.11.1). Além disso, as métricas de erros MAE, MSE e RMSE foram menores que 1, demonstrando uma boa precisão na estimação. Como pode ser observado na Tabela 4.3.

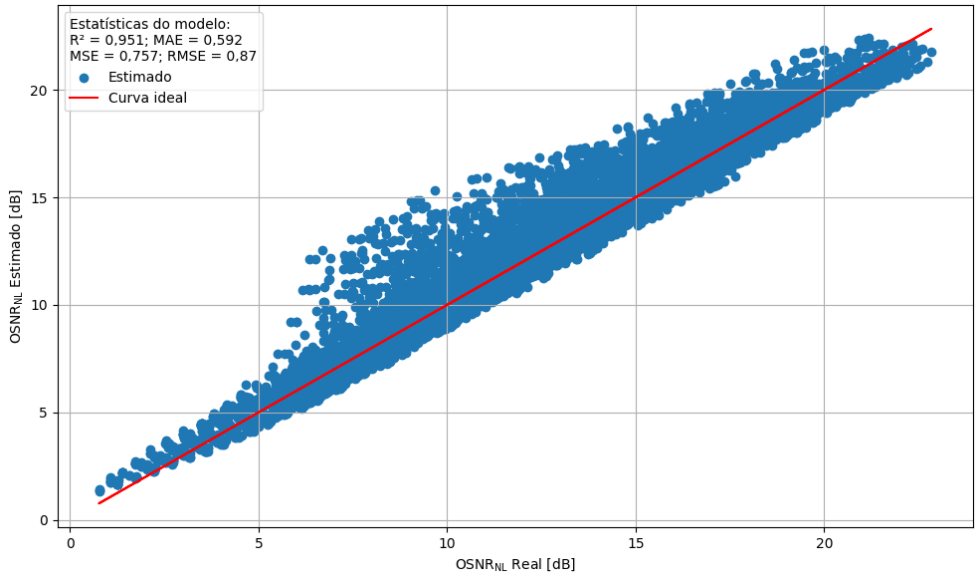
O R^2 da $OSNR_{dB_{NL}}$ pode ser visualizado graficamente na Figura 4.9 na qual a

Tabela 4.3: Resultados da validação cruzada para os modelos SVR *kernel* RBF.

Entrada	Saída	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span} $N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,951	0,592	0,757	0,870
	$NLIN_{Power}$ [dB]		0,997	0,583	0,520	0,721

distribuição das estimações em função do valor real está bastante espalhada, porém com um certo agrupamento em torno da curva ideal.

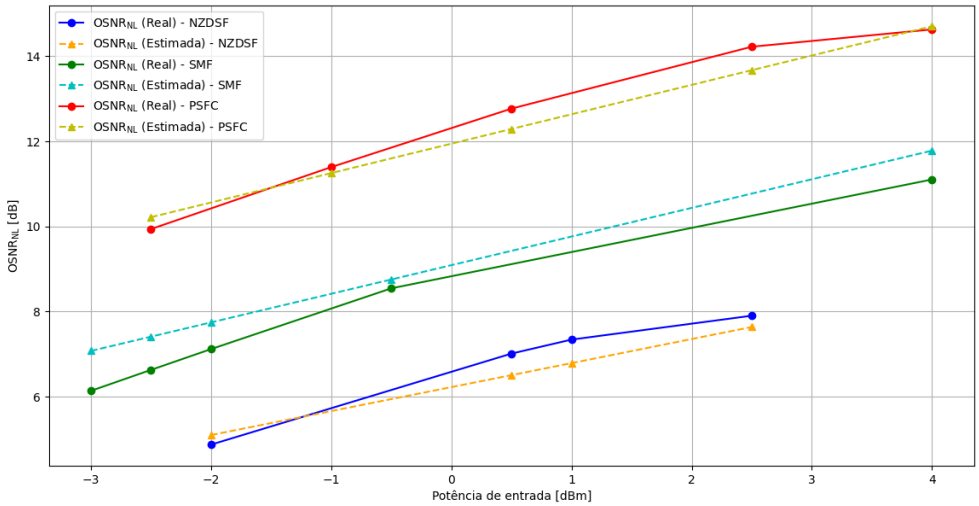
Figura 4.9: $OSNR_{dB_{NL}}$ estimado vs $OSNR_{dB_{NL}}$ real - Algoritmo SVR *Kernel* RBF, resultados do conjunto de teste



Fonte: O autor

O impacto do erro para a $OSNR_{dB_{NL}}$ observados na Tabela 4.3 podem ser visualizadas graficamente na Figura 4.10, na qual é possível verificar que as estimações estão bem próximos dos valores reais.

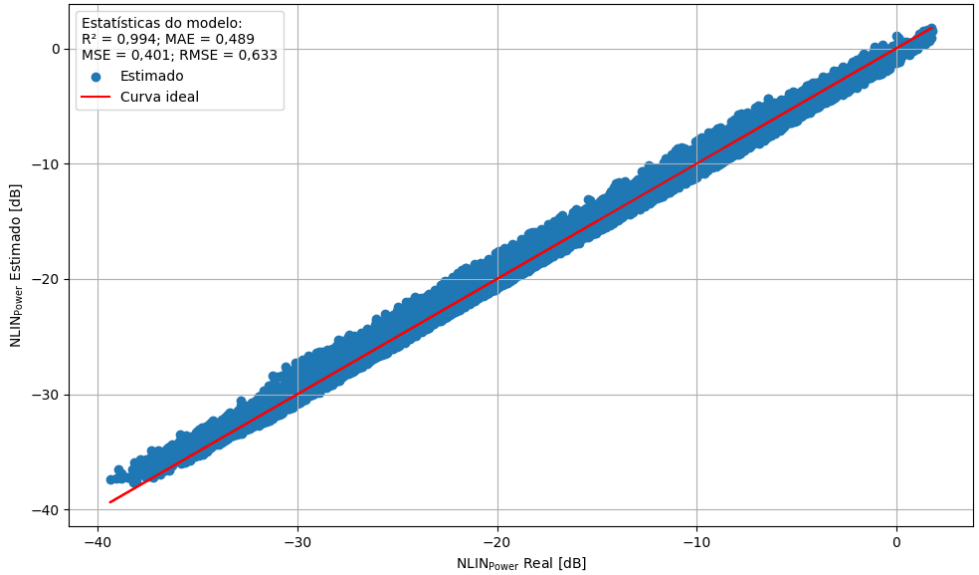
Figura 4.10: $OSNR_{dB_{NL}}$ versus potência. Algoritmo SVR *kernel* RBF, resultados do conjunto de teste. Parâmetros das amostras: $L_{Span} = 110$ [km], $N_{Ch_{Span}} = 13$, $N_{Spans} = 12$ e $Modulação : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

O R^2 da $NLIN_{Power}$ pode ser visualizado graficamente na Figura 4.11. Ao analisarmos o gráfico notamos que a distribuição das estimações em função do valor real tende a se aproximarem da curva ideal.

Figura 4.11: $NLIN_{Power}$ estimado vs $NLIN_{Power}$ real - Algoritmo SVR *Kernel* RBF, resultados do conjunto de teste

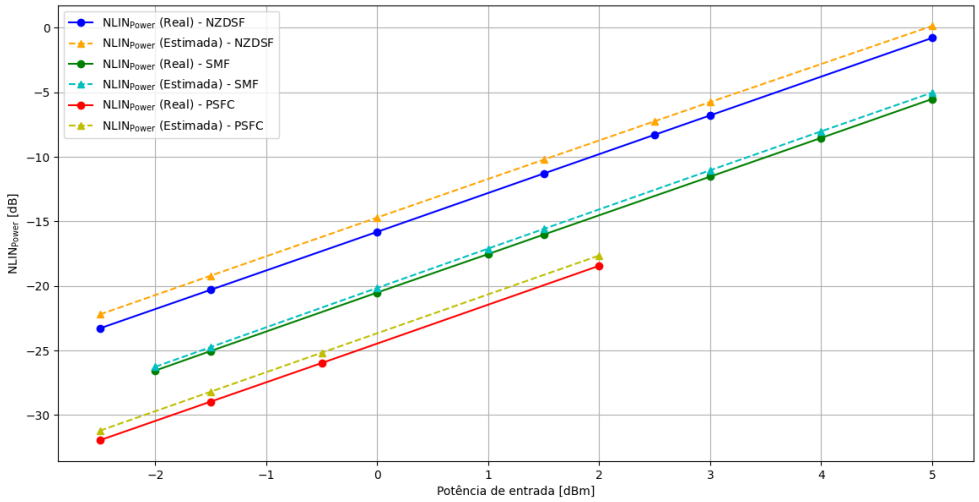


Fonte: O autor

O impacto das métricas de erro do modelo de SVR para a $NLIN_{Power}$ pode ser observadas na Tabela 4.3 podem ser visualizadas graficamente na Figura 4.12, em

que é possível verificar a proximidade dos valores estimados e reais.

Figura 4.12: $NLIN_{Power}$ versus potência de entrada. Algoritmo SVR *kernel* RBF, resultados do conjunto de teste. Parâmetros das amostras: $L_{Span} = 100$ [km], $N_{Ch_{Span}} = 11$, $N_{Spans} = 13$ e $Modulação : QPSK$ cujo $coef_{Mod} = \frac{1}{4}$.



Fonte: O autor

Apesar dos resultados dos modelos se mostrarem bastante promissores para os dados de treinamento, conforme evidenciado na Tabela 4.3 e nos gráficos das Figura 4.9 à Figura 4.12, é realizada a estimação da $OSNR_{dB_{NL}}$ e do $NLIN_{Power}$ utilizando o conjunto de dados de validação para verificar a generalização do modelo. As métricas foram calculadas e estão apresentadas na Tabela 4.4.

Tabela 4.4: Resultados dos modelos SVM - *Kernel* RBF, para os dados de validação

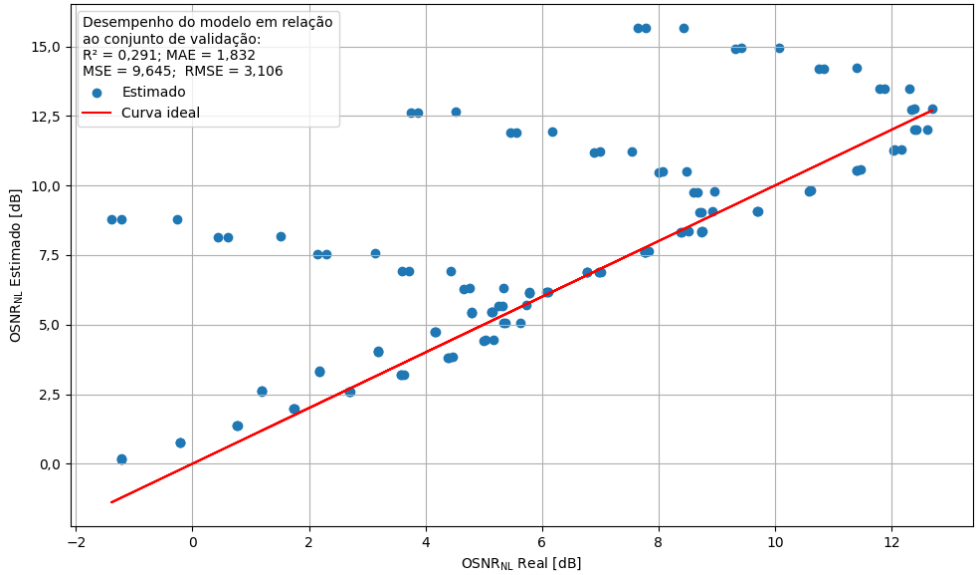
Entrada	Saída	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span} $N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,291	1,832	9,654	3,106
	$NLIN_{Power}$ [dB]		0,994	0,489	0,401	0,633

Os resultados obtidos para a $OSNR_{dB_{NL}}$ foram extremamente desanimadores, com um valor de R^2 de apenas $29,1\%$, indicando que o modelo não consegue se ajustar ao conjunto de dados de validação para generalização, conforme ilustrado na Figura 4.13. Além disso, o MAE, MSE e RMSE apresentaram valores elevados, sugerindo uma precisão com uma margem de erro considerável, o que consequentemente resulta em estimativas insatisfatórias, conforme mostrado na Figura 4.14.

O R^2 o modelo de SVM para a estimação da $OSNR_{dB_{NL}}$ pode ser visualizado graficamente na Figura 4.13 na qual a distribuição das estimações em função do

valor real está bastante espalhada além de várias amostras estarem distantes da curva ideal.

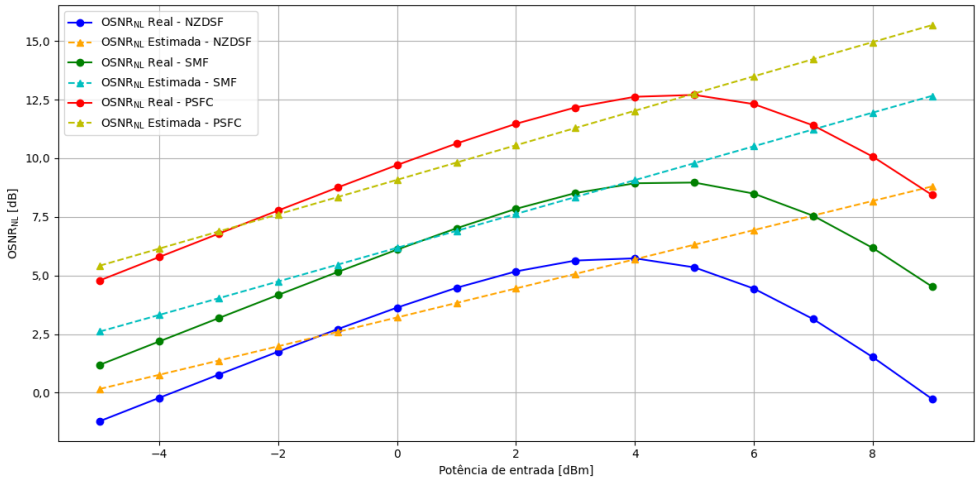
Figura 4.13: $OSNR_{dB_{NL}}$ real versus $OSNR_{dB_{NL}}$ estimado. Algoritmo SVR *kernel* RBF, resultados do conjunto de validação.



Fonte: O autor

O impacto dos valores das métricas de erro para o modelo SVR para estimação da $OSNR_{dB_{NL}}$ do conjunto de validação observadas na Tabela 4.4 podem ser visualizadas graficamente na Figura 4.2, onde é possível verificar, que, para o intervalo de **potência de entrada** $[-3, 5]$, o erro de estimação da $OSNR_{dB_{NL}}$ é extremamente baixo. No entanto, nos intervalos $[-5, -3[$ e $]5, 9]$, o modelo começa a demonstrar erros mais consideráveis em comparação com os valores reais. Além disso nota-se que os valores estimados continuam em uma tendência de crescimento linear em função da potência de entrada enquanto que a curva real continua com uma tendência de queda.

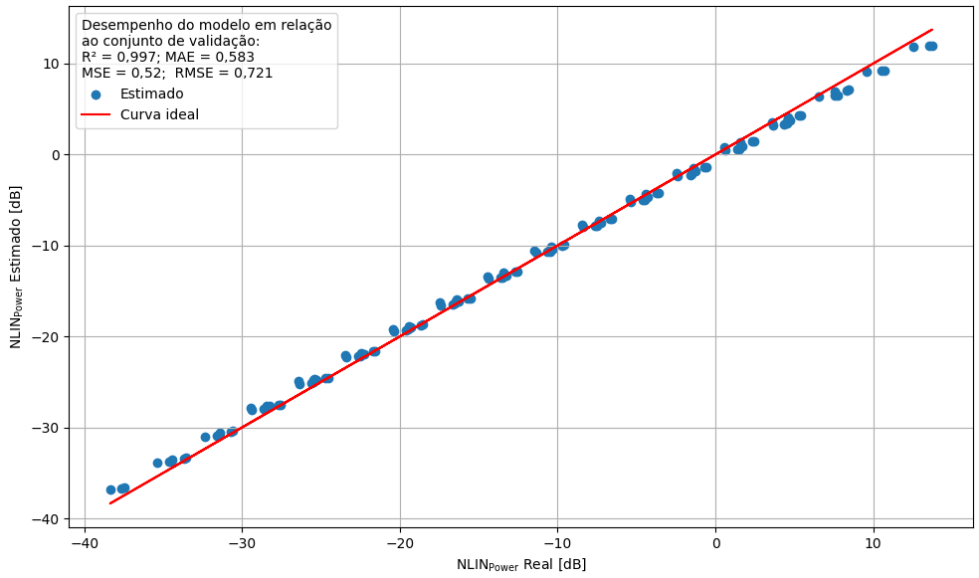
Figura 4.14: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo SVR *kernel* RBF, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 110$ [km], $N_{Ch_{Span}} = 13$, $N_{Spans} = 12$ e $Modulação : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

O resultado para a $NLIN_{Power}$ é bastante satisfatório, alcançando um valor de R^2 de 99,4%, o que indica que o modelo consegue ajustar-se bem ao conjunto de dados de validação para generalização, como mostrado na Figura 4.15, observa-se que es estimações estão bastante agrupadas e muito próximos da curva ideal.

Figura 4.15: $NLIN_{Power}$ estimado vs $NLIN_{Power}$ real. Algoritmo SVR *Kernel* RBF, resultados do conjunto de validação

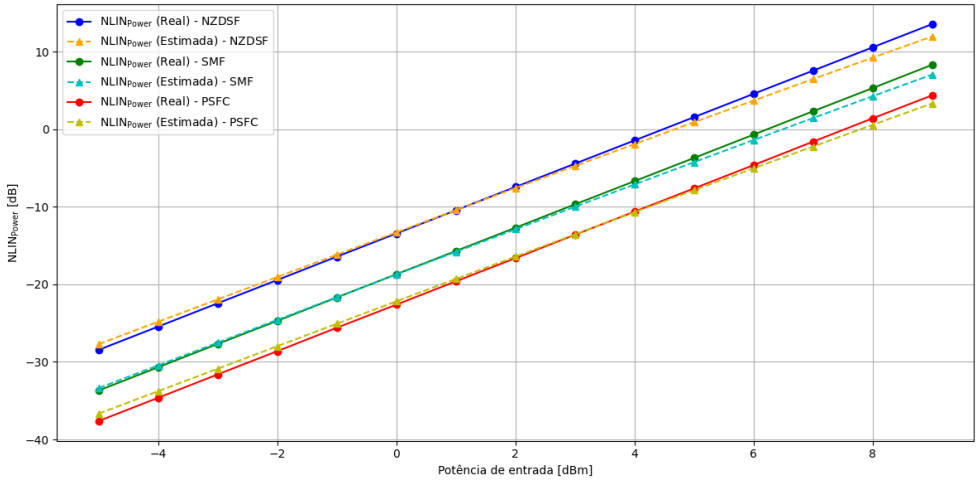


Fonte: O autor

O gráfico da Figura 4.16 ilustra como o modelo SVR de estimação do $NLIN_{Power}$

se comporta para o conjunto de validação, é possível verificar que as estimações do modelo tendem a se aproximar do valor real, resultando na quase sobreposição das curvas estimadas e real.

Figura 4.16: $NLIN_{Power}$ versus potência de entrada. Algoritmo SVR *kernel* RBF, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulação : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

Em resumo, embora o modelo SVR para a estimação da $OSNR_{dB_{NL}}$ possa apresentar resultados satisfatórios em dados semelhantes aos do conjunto de treinamento e teste, sua capacidade de generalização para amostras ligeiramente diferentes é bastante limitada, essa limitação pode ter sido causada devido a um *overfitting* na qual o modelo se superajusta aos dados de treino e teste e quando é confrontado com dados diferentes resulta em estimações péssimas. Por outro lado, o modelo SVR para a estimação do $NLIN_{Power}$ demonstrou resultados excelentes, evidenciando sua capacidade de ajustar-se para estimar o $NLIN_{Power}$ mesmo para amostras muito diferentes dos dados de treinamento.

Kernel Polinomial

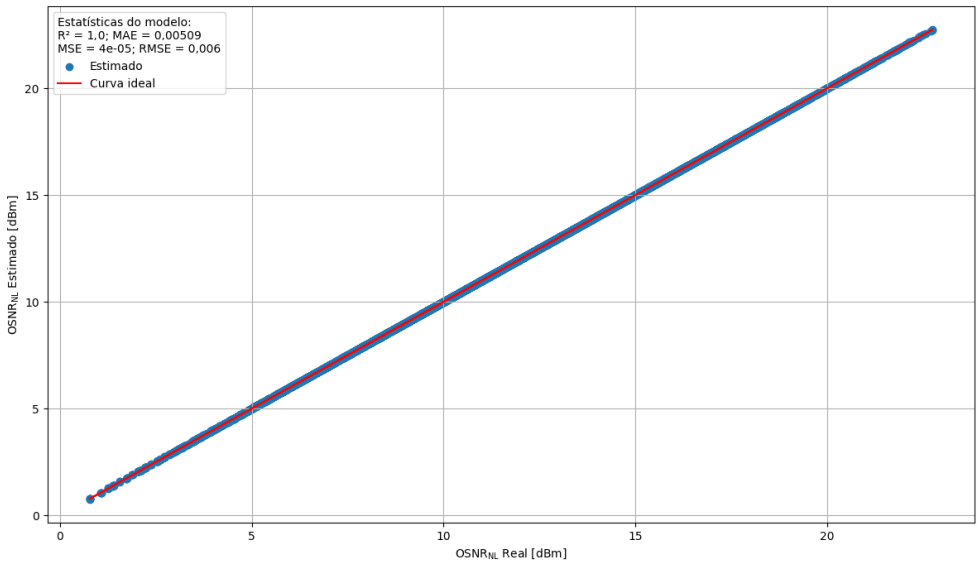
Ao avaliar os resultados da implementação dos modelos de SVR com *kernel* polinomial de grau 8 observamos que, o desempenho é satisfatório para ambos modelos. Os modelos de $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ obtiveram um R^2 de 100% em ambos os modelos, indicando uma boa capacidade de explicar a variabilidade nos dados, ou seja, ambos os modelos tendem a se ajustarem bem às amostras de teste (2.11.1). Além disso, as métricas de erros MAE, RMSE estão com ordem de grandeza de 10^{-3} e o MSE está com ordem de grandeza de 10^{-5}, o que indica uma boa precisão na estimação. Como pode ser observado na Tabela 4.5.

Tabela 4.5: Resultados dos modelos SVR - Kernel Poly com grau 8 para o conjunto de teste. Modelos treinados com os dados normalizados, resultados do conjunto de teste.

Entrada	Saída	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span} $N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$OSNR_{dB_{NL}}$ [dB]	Sintético	1	0,005	3,6e-5	0,006
	$NLIN_{Power}$ [dB]		1	0,007	9e-5	0,009

O R^2 da $OSNR_{dB_{NL}}$ pode ser visualizado graficamente na Figura 4.17 na qual a distribuição das estimações em função do valor real está bastante espalhada, porém com um certo agrupamento em torno da curva ideal.

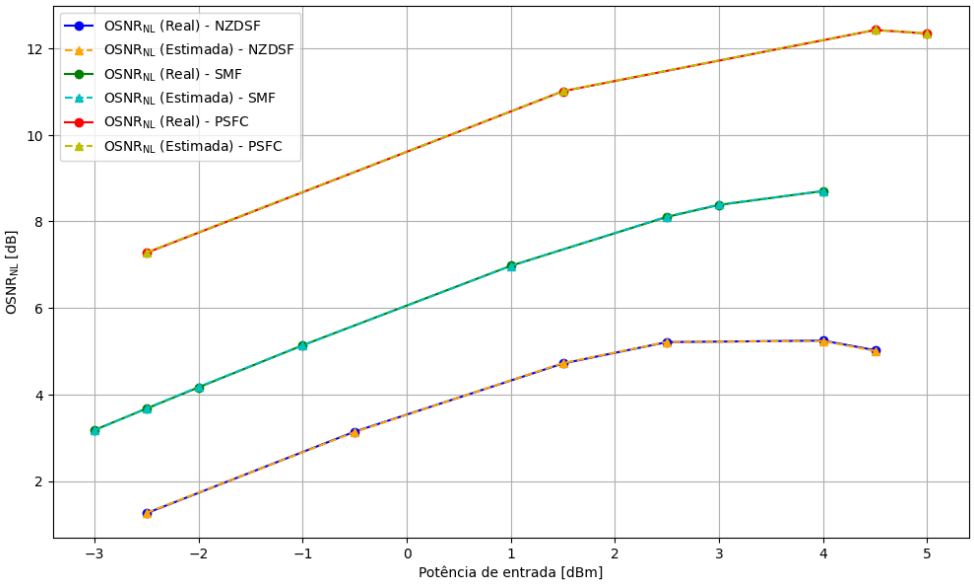
Figura 4.17: $OSNR_{dB_{NL}}$ estimado vs $OSNR_{dB_{NL}}$ real - Algoritmo SVR *Kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de teste.



Fonte: O autor

O impacto do erro no modelo de SVR para a $OSNR_{dB_{NL}}$ disposto na Tabela 4.5 pode ser visualizado graficamente na Figura 4.18, na qual é possível verificar que as estimações estão bem próximas dos valores reais.

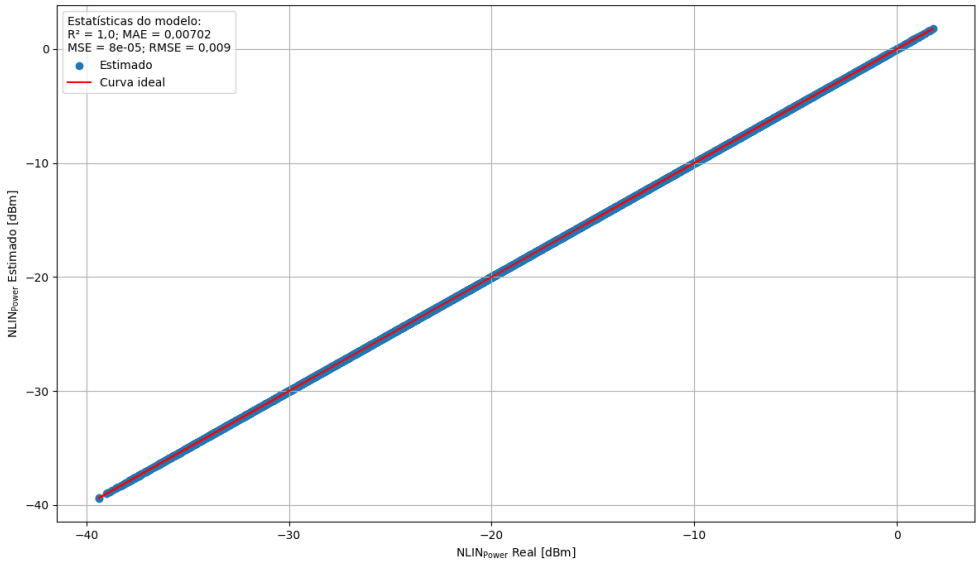
Figura 4.18: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo SVR *kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de teste. Parâmetros das amostras: $L_{Span} = 110$ [km], $N_{Ch_{Span}} = 13$, $N_{Spans} = 12$ e $Modulação$: $64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

O R^2 da $NLIN_{Power}$ pode ser visualizado graficamente na Figura 4.19. Ao analisarmos o gráfico notamos que a distribuição das estimações em função do valor real tende a se aproximarem da curva ideal.

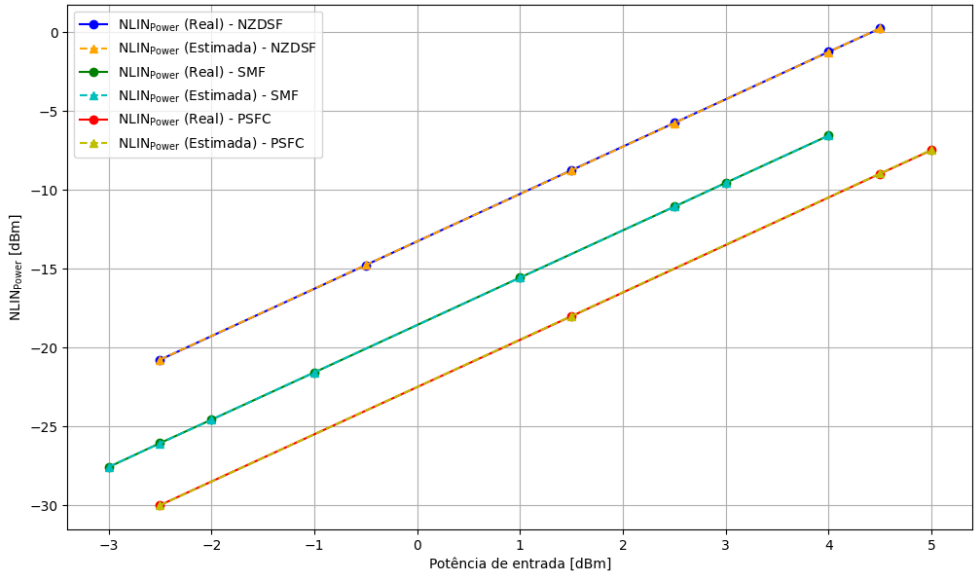
Figura 4.19: $NLIN_{Power}$ estimado vs $NLIN_{Power}$ real - Algoritmo SVR *Kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de teste.



Fonte: O autor

O impacto do erro no modelo de SVR para a $NLIN_{Power}$ dispostos na Tabela 4.5 pode ser visualizado graficamente na Figura 4.20, em que é possível verificar a proximidade dos valores estimados e reais.

Figura 4.20: $NLIN_{Power}$ versus potência de entrada. Algoritmo SVR *kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de teste. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulação : QPSK$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

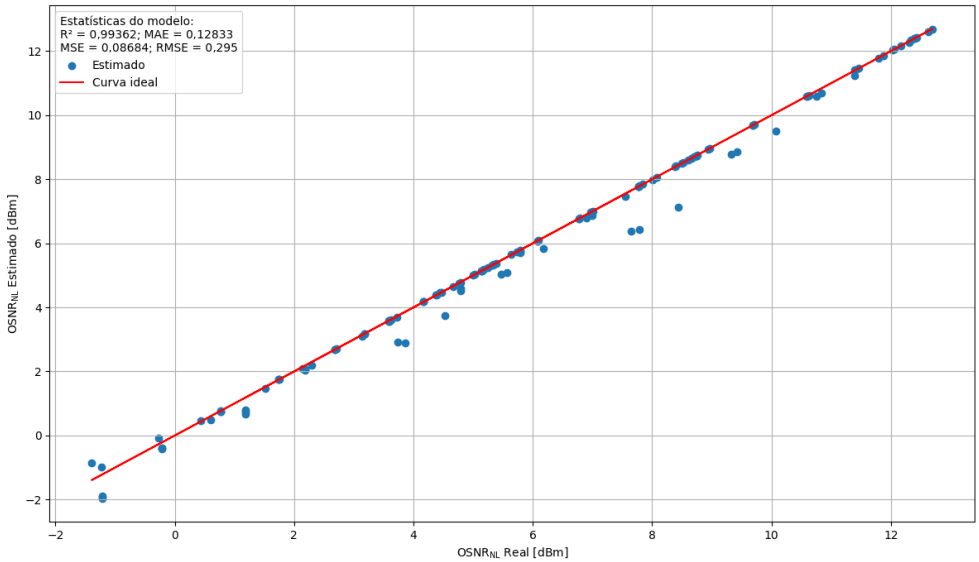
Apesar dos resultados dos modelos se mostrarem bastante promissores para os dados de treinamento, conforme evidenciado na Tabela 4.5 e nos gráficos das Figura 4.17 à Figura 4.20, é realizada a estimação da $OSNR_{dB_{NL}}$ e do $NLIN_{Power}$ utilizando o conjunto de dados de validação para verificar a generalização do modelo. As métricas foram calculadas e estão apresentadas na Tabela 4.6. Na qual é possível verificar que, apesar dos resultados terem sidos ligeiramente piores quando comparados com os resultados do conjunto de teste eles ainda continuam promissores.

Tabela 4.6: Resultados para os dados de validação para os modelos de SVR *kernel* Polinomial com grau 8. Modelos treinados treinados com dados normalizados.

Saída	Entrada	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,994	0,128	0,087	0,295
$N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$NLIN_{Power}$ [dB]		0,999	0,049	0,016	0,127

Os resultados são promissores, como pode ser visto na Tabela 4.6. Os valores das métricas para a $OSNR_{dB_{NL}}$ demonstram que o modelo ainda é capaz de fazer boas estimativas com alta precisão e explicabilidade dos dados. Essas conclusões podem ser visualizadas graficamente nas Figura 4.21 e Figura 4.22. O alto valor de R^2 indica que os dados conseguem ser explicados pelo modelo, fazendo com que as estimativas se concentrem próximos à linha vermelha, linha que indica a curva ideal na qual $Predição = Estimação$.

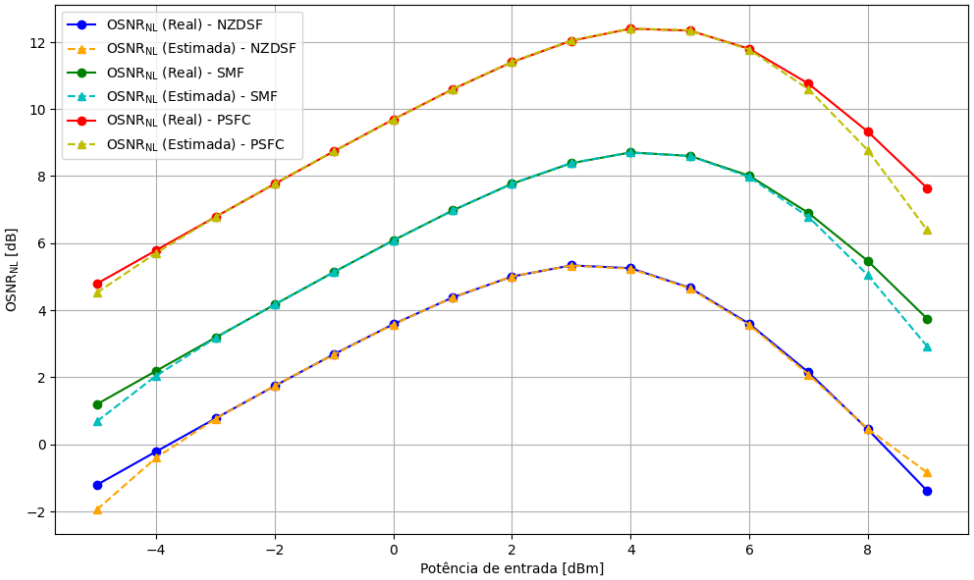
Figura 4.21: $OSNR_{dB_{NL}}$ real versus $OSNR_{dB_{NL}}$ estimado. Algoritmo SVR *Kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de validação.



Fonte: O autor

Na Figura 4.22 é possível observar que em todo o intervalo o modelo consegue acompanhar a curva real, indicando baixíssimos erros no final do intervalo, enquanto que dentro do intervalo de [-3, 5] é possível observar que o erro entre o valor estimado e real é praticamente nulo, este fato acontece porque o modelo é treinado para um intervalo de potência de entrada de -3 à 5 dBm.

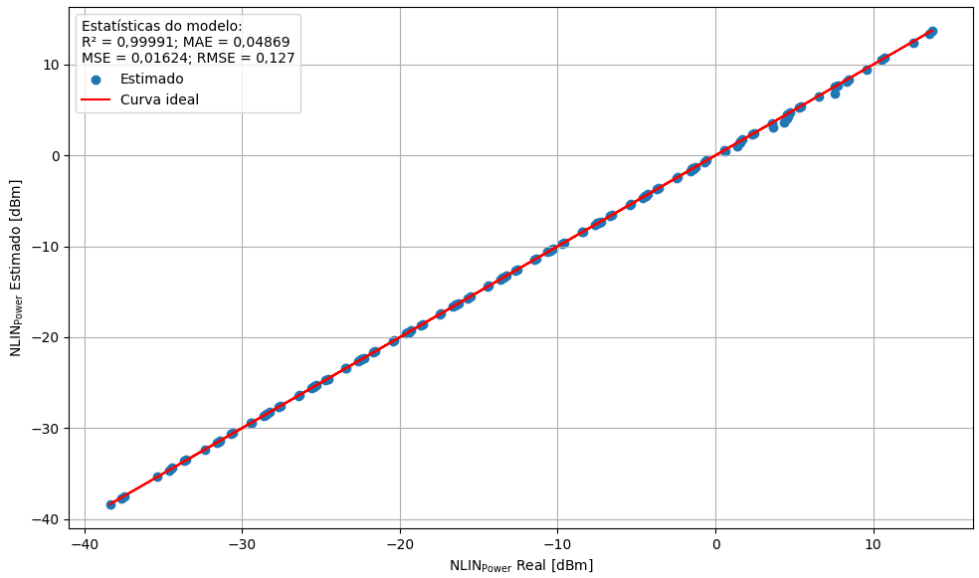
Figura 4.22: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo SVR *kernel* Polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulacão : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

O resultado para a $NLIN_{Power}$ também é satisfatório, pois os valores das métricas para a $NLIN_{Power}$, apesar de terem sidos ligeiramente piores quando comparado com os resultados do conjunto de treinamento o modelo ainda é capaz de fazer boas estimativas com alta precisão e explicabilidade dos dados. Essas conclusões podem ser visualizadas graficamente nas Figura 4.23 e Figura 4.24. O valor de R^2 continua alto o que indica modelo pode explicar a variabilidade dos dados, fazendo com que as estimativas se concentrem próximos à linha vermelha, linha que indica a curva ideal

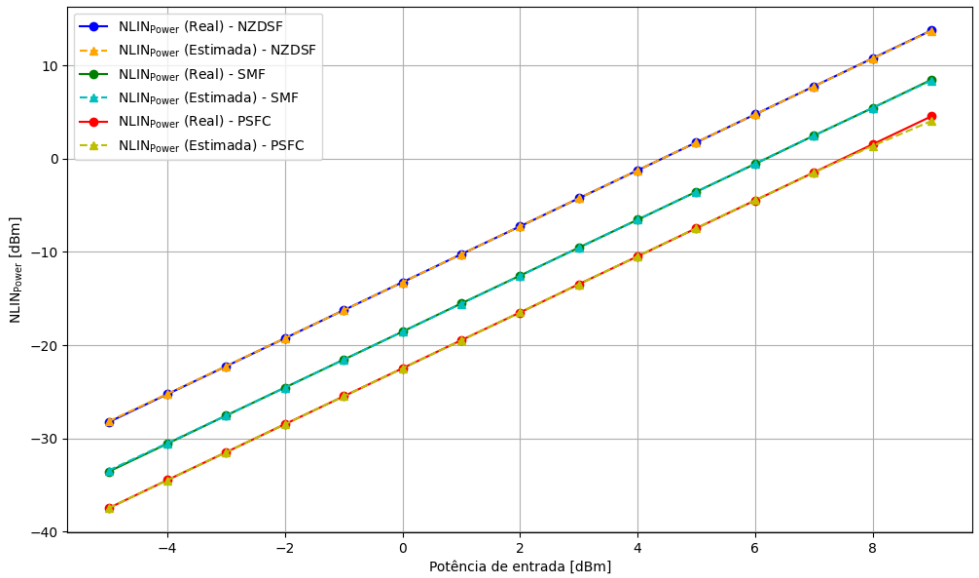
Figura 4.23: $NLIN_{Power}$ estimado vs $NLIN_{Power}$ real. Algoritmo SVR *kernel* polinomial com grau 8. Modelo treinado com dados normalizados, resultados do conjunto de validação.



Fonte: O autor

O gráfico da Figura 4.16 ilustra como o modelo SVR de estimação do $NLIN_{Power}$ se comporta para o conjunto de validação, é possível verificar que as estimações do modelo tendem a se aproximar do valor real, resultando na quase sobreposição das curvas estimadas e real.

Figura 4.24: $NLIN_{\text{Power}}$ versus potência de entrada. Algoritmo SVR *kernel* polinomial com grau 8 Modelo treinado com dados normalizados, resultados do conjunto de validação. Parâmetros das amostras: $L_{\text{Span}} = 120\,\text{km}$, $N_{\text{Ch,Span}} = 15$, $N_{\text{Spans}} = 15$ e Modulação: 64QAM cujo $coef_{\text{Mod}} = \frac{1}{12}$.



Fonte: O autor

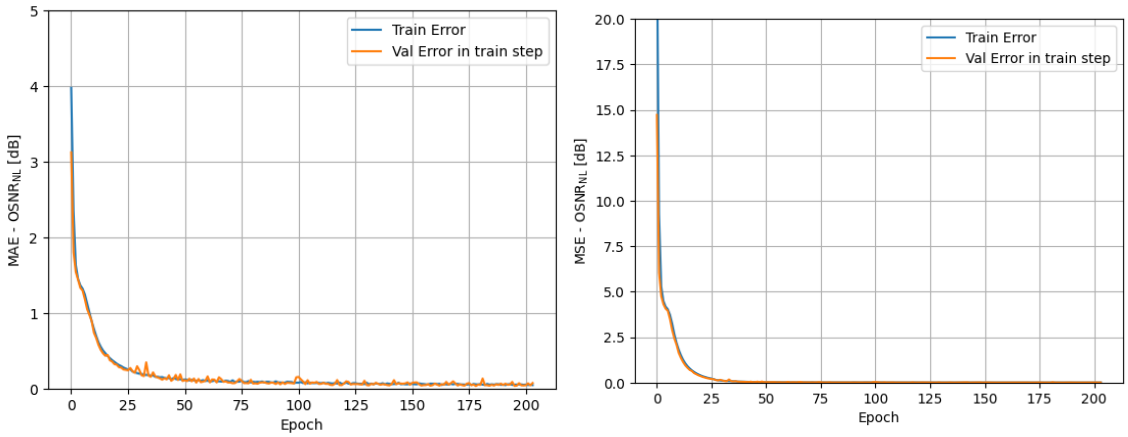
Em resumo, os modelo SVR implementados com o *kernel* polinomial demonstra um resultado excelente quando comparado com os outros modelos, pois tanto para o conjunto de testes quanto para o conjunto de treinamento os valores estimados eram suficientemente próximos dos valores reais, o que resulta num baixo erro e num R^2 alto.

4.1.3 *Artificial Neural Network* (ANN)

Neste subtópico iremos tratar da análise dos resultados e avaliação do desempenho do algoritmo *Artificial Neural Network*.

O modelo implementado segue a arquitetura descrita em 3.4.3, na qual podemos verificar a evolução do MAE e MSE de treino e validação da etapa de treinamento da ANN implementada para estimar a $OSNR_{dB_{NL}}$ como na Figura 4.25.

Figura 4.25: Em (a) evolução do MAE por época, em (b) evolução do MSE por época para do modelo de ANN para estimação da $OSNR_{dB_{NL}}$

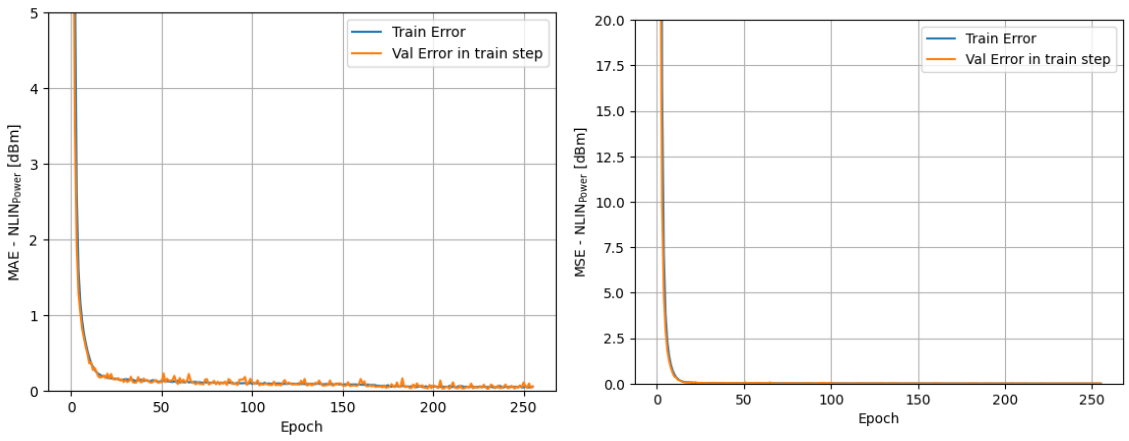


(a) (b)

Fonte: O autor

A Figura 4.26 exibe a evolução do MAE e MSE de treino e validação da etapa de treinamento da ANN implementada para estimar a $NLIN_{Power}$.

Figura 4.26: Em (a) evolução do MAE por época, em (b) evolução do MSE por época para do modelo de ANN para estimação da $NLIN_{Power}$



(a) (b)

Fonte: O autor

Pelos gráficos observa-se que, por volta da época 200 os algoritmos não conseguem mais diminuir o MSE e MAE, com isso o *callback* EarlyStopping, do pacote keras.callbacks da biblioteca tensorflow é utilizado para que o treinamento seja interrompido a fim de prevenir um superajuste dos modelos.

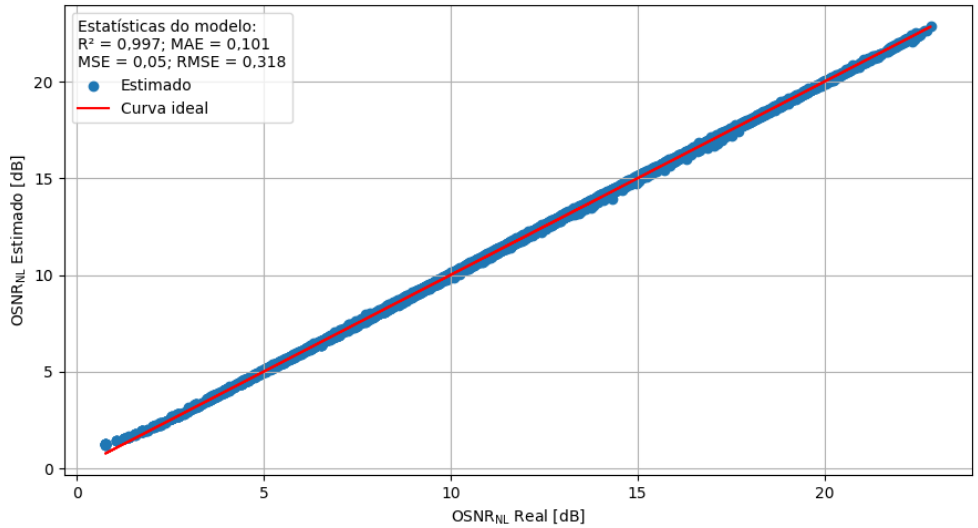
Ao avaliar os resultados da implementação dos modelos de ANN para a estimação da $OSNR_{dB_{NL}}$ e $NLIN_{Power}$ (Tabela 4.7), observamos que, o desempenho é satisfatório para ambas as saídas consideradas.

Tabela 4.7: Resultados da validação cruzada para modelos de ANN para o conjunto de testes.

Saída	Entrada	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,997	0,101	0,050	0,224
$N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$NLIN_{Power}$ [dB]		1	0,052	0,005	0,071

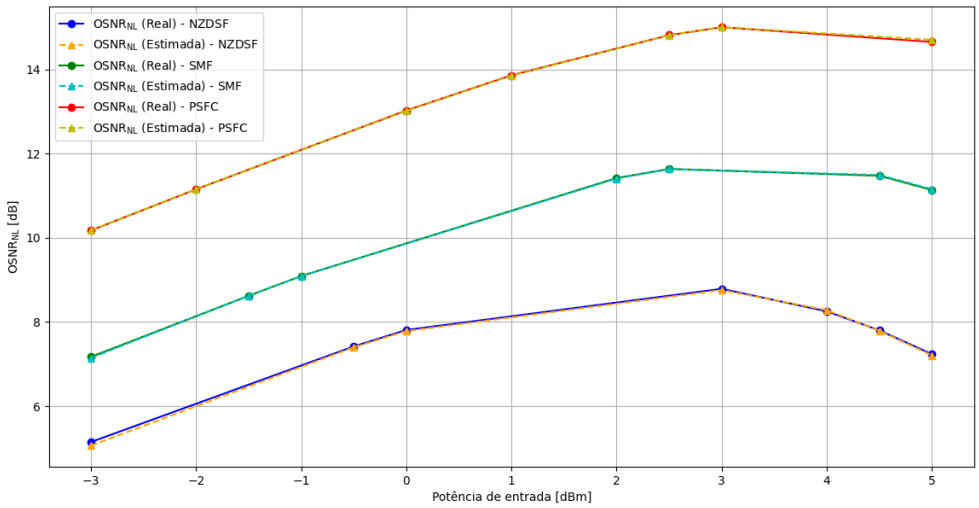
Em ambos os modelos valor obtido para o R^2 foram superiores a 99% para o conjunto de teste, indicando uma boa capacidade do modelo em explicar a variabilidade nos dados (ver Figura 4.27 e Figura 4.29). Além disso, as métricas de erros MAE, MSE e RMSE estão com valores abaixo de 0,3 demonstrando uma alta precisão na estimação, tais resultados podem ser observados nas Figura 4.28 e Figura 4.30.

Figura 4.27: $OSNR_{dB_{NL}}$ estimado vs $OSNR_{dB_{NL}}$. real. Algoritmo ANN, resultados do conjunto de teste.



Fonte: O autor

Figura 4.28: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo ANN, resultados do conjunto de teste. Parâmetros fixados: $L_{Span} = 110$ [km], $N_{Ch_{Span}} = 11$, $N_{Spans} = 15$ e $Modulação$: $QPSK$ cujo $coef_{Mod} = \frac{1}{4}$.



Fonte: O autor

Figura 4.29: $NLIN_{Power}$ estimado vs $NLIN_{Power}$. real. Algoritmo ANN, resultados para os dados de teste.

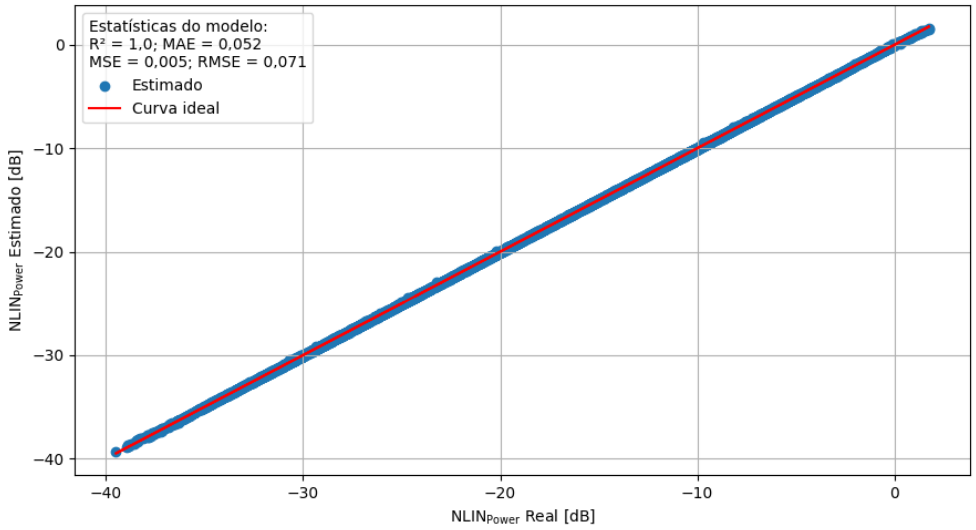
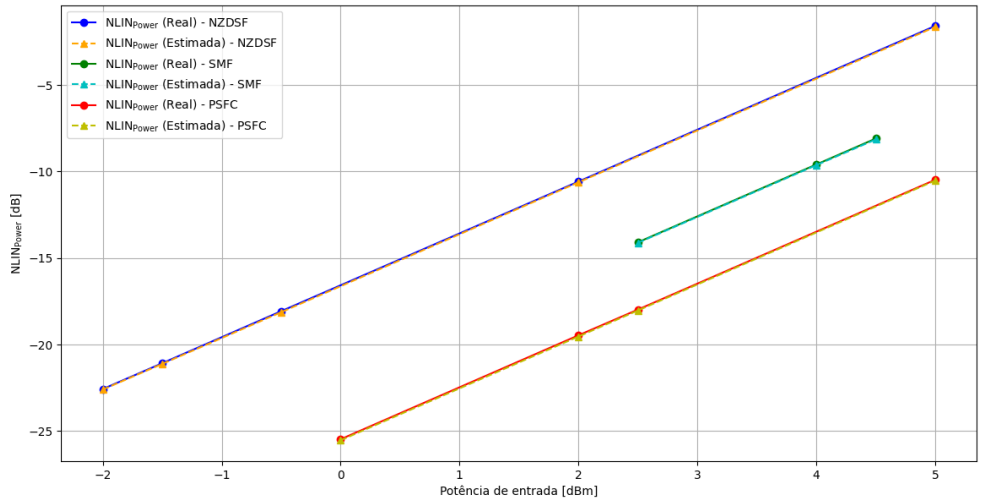


Figura 4.30: $NLIN_{\text{Power}}$ versus potência de entrada. Algoritmo ANN, resultados do conjunto de teste. Parâmetros fixados: $L_{\text{Span}} = 110\,\text{km}$, $N_{\text{Ch,Span}} = 11$, $N_{\text{Spans}} = 9$ e Modulação: 16QAM cujo $coef_{\text{Mod}} = \frac{1}{8}$.



Fonte: O autor

Tabela 4.8: Resultados do conjunto de validação para os modelos de ANN.

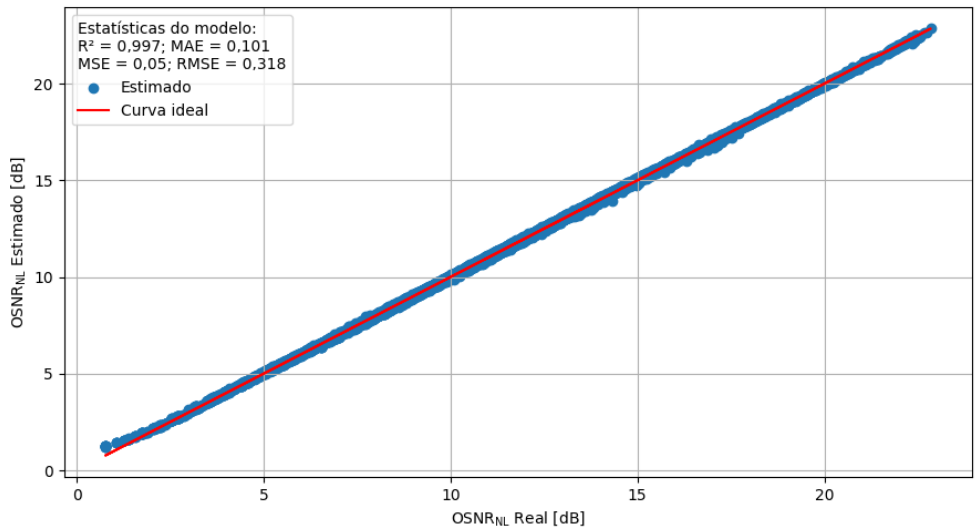
Entrada	Saída	Fonte de Dados	Desempenho			
			R^2	MAE	MSE	RMSE
D, γ, β_2, α, $P_{Ch_{In}}$, L_{Span}	$OSNR_{dB_{NL}}$ [dB]	Sintético	0,965	0,380	0,477	0,691
$N_{Ch_{Span}}$, $coef_{Mod}$, N_{Spans}	$NLIN_{Power}$ [dB]		0,972	1,020	5,09	2,256

Os resultados obtidos para conjunto de validação se mostram bastante promis-

sores. Verifica-se que o R^2 para ambos os modelos é muito próximo do valor obtido para o conjunto de testes, como pode ser observado na Tabela 4.8. Em relação às métricas de erro nota-se que os valores aumentaram consideravelmente.

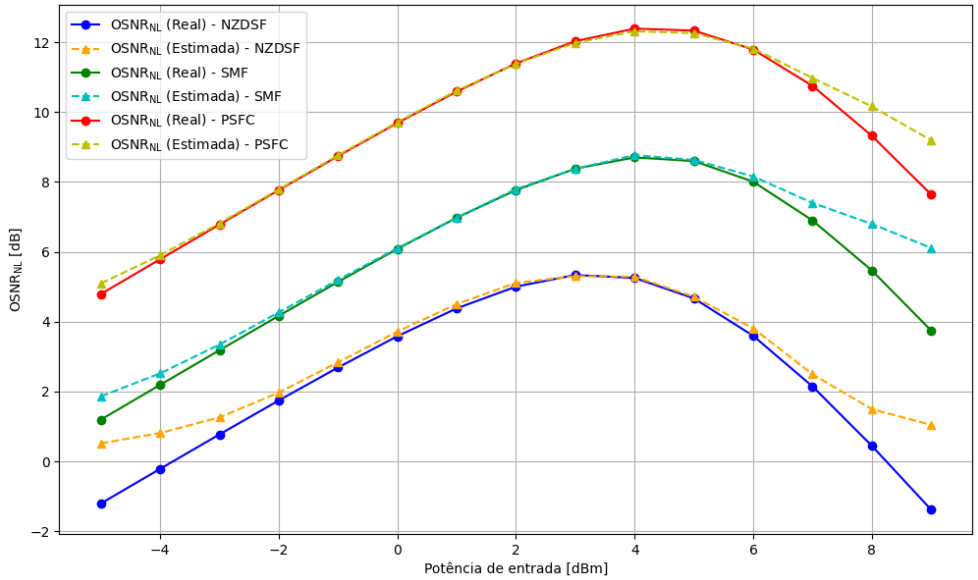
O modelo da $OSNR_{dB_{NL}}$ demonstrou uma boa capacidade de explicar a variação dos dados, conforme mostrado na Figura 4.31, e uma boa precisão. Porém, para os valores de potência de entrada muito menores que -5 [dBm] ou muito maiores que 9 [dBm], o erro entre o valor estimado e o valor real começa a ficar mais expressivo, fazendo com que o modelo tenha um desempenho ruim para valores de potência de entrada muito distantes do intervalo [-5, 9]. Uma forma de melhorar esse resultado seria aumentando a quantidade de amostras mais representativas e também aumentando a complexidade da rede neural, adicionando mais camadas e mais neurônios.

Figura 4.31: $OSNR_{dB_{NL}}$ estimado vs $OSNR_{dB_{NL}}$ real. Algoritmo ANN, resultados do conjunto de validação.



Fonte: O autor

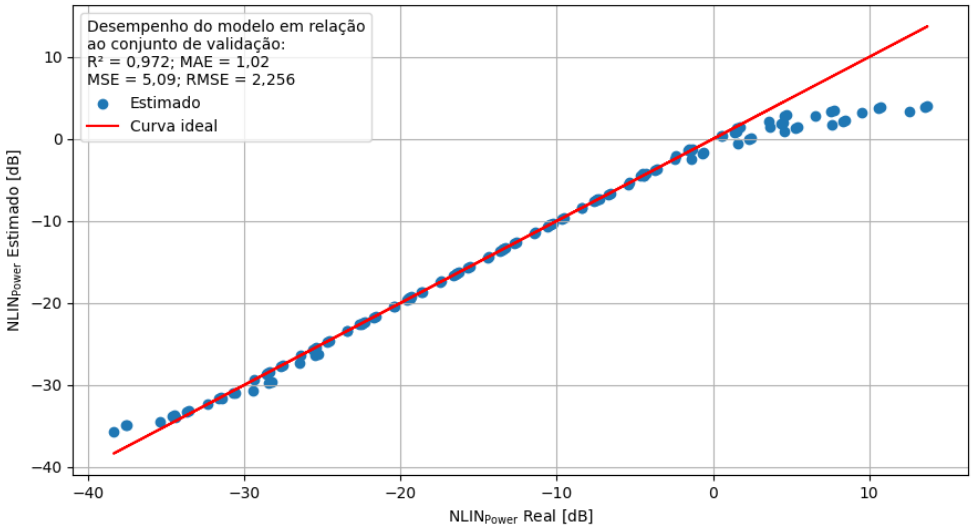
Figura 4.32: $OSNR_{dB_{NL}}$ versus potência de entrada. Algoritmo ANN, resultados do conjunto de validação. Parâmetros das amostras: $L_{Span} = 120$ [km], $N_{Ch_{Span}} = 15$, $N_{Spans} = 15$ e $Modulação : 64QAM$ cujo $coef_{Mod} = \frac{1}{12}$.



Fonte: O autor

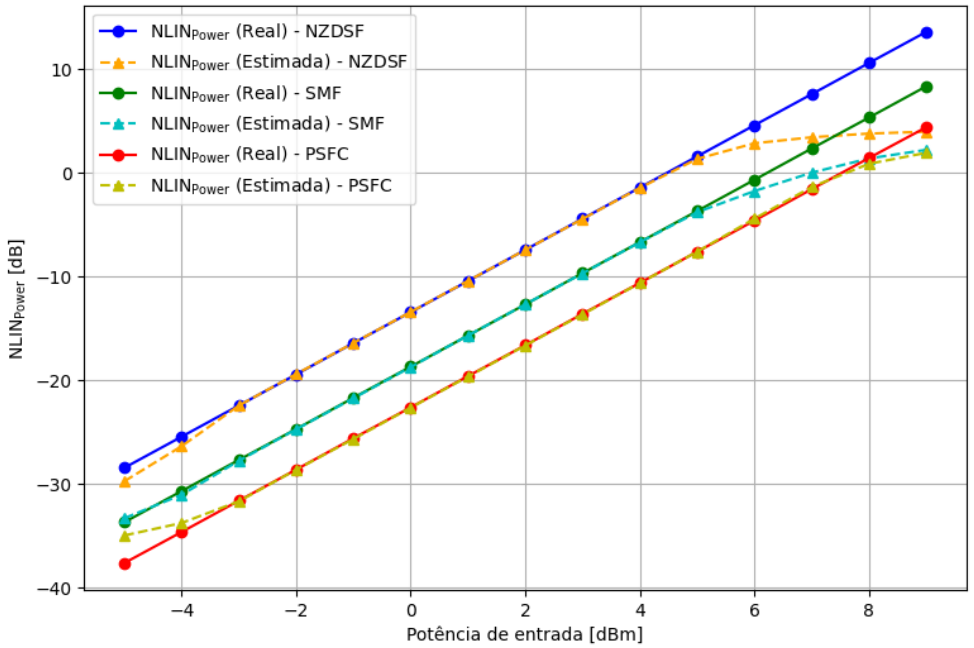
O modelo do $NLIN_{Power}$ teve bons resultados para o R^2, ver Figura 4.33 e um resultado ruim para as métricas de erros. Ao verificar o gráfico da Figura 4.34 nota-se que há uma tendência da estimação de se aproximar de um valor a medida que à potência de entrada aumenta. Esse padrão indica que este modelo de ANN implementado para a estimação do $NLIN_{Power}$ não consegue realizar boas estimativas para valores de potência de entrada muito maiores que 5 [dBm].

Figura 4.33: $NLIN_{Power}$ estimado vs $NLIN_{Power}$ real. Algoritmo ANN, resultados do conjunto de validação.



Fonte: O autor

Figura 4.34: $NLIN_{\text{Power}}$ versus potência de entrada. Algoritmo ANN, resultados do conjunto de validação. Parâmetros das amostras: $L_{\text{Span}} = 120\,\text{km}$, $N_{\text{Ch,Span}} = 15$, $N_{\text{Spans}} = 15$ e Modulação: 16QAM cujo $coef_{\text{Mod}} = \frac{1}{8}$.



Fonte: O autor

Em suma, embora os resultados obtidos para o $NLIN_{Power}$ tenham deixado a desejar no conjunto de validação da generalização, o modelo mostrou-se bastante promissor para a estimação quando os dados de entrada possuem valores próximos

aos dos dados de treinamento. Essa degradação do desempenho pode ser explicada por um *overfitting* ou pelo número reduzido de amostras. Por outro lado, o modelo de ANN para a estimação do $OSNR_{dBNL}$ teve um desempenho excelente, conseguindo se aproximar da curva formada pelos valores reais do $OSNRdB_{NL}$.

4.2 Comparativo entre os resultados dos modelos

A Tabela 4.9 apresenta os resultados dos algoritmos de ML para estimar a $OSNR_{dB_{NL}}$, incluindo o tempo de treinamento, o tempo de estimação/predição e o erro médio quadrático (MSE). Observa-se que, embora o KNN tenha tempos de treinamento e predição extremamente baixos, o MSE associado é muito alto em comparação com os algoritmos ANN e SVR-Poly. Além disso, a análise do KNN em relação a um conjunto de validação revela sua limitação em termos de generalização, resultando em estimativas ruins à medida que as amostras diferem dos dados de treinamento e teste. O SVR-RBF, apesar de ter tempos de treinamento e predição baixos, possui o maior MSE associado entre os quatro algoritmos. Sua análise em relação ao conjunto de validação sugere uma limitação semelhante ao KNN, indicando um problema de generalização. No caso do SVR-POLY, os tempos de treinamento são significativamente maiores do que os dos demais algoritmos, sendo cerca de 94,3 vezes maior que o segundo mais demorado. Apesar disso, os resultados são muito promissores, com o menor MSE associado de todos os algoritmos. Em relação ao conjunto de validação, o SVR-POLY se destaca, pois todas as métricas foram boas e a curva estimada acompanhou a curva real com um erro muito baixo. Por fim, o ANN apresentou um desempenho excelente, com tempos de treinamento e predição baixos, e um MSE associado também baixo. Os resultados do conjunto de validação mostraram que as estimativas feitas pelo ANN conseguiram se aproximar da curva real. No entanto, é possível melhorar ainda mais os resultados para o ANN modificando a rede, tornando-a mais complexa e otimizando os hiper parâmetros.

Tabela 4.9: Comparação entre os tempos de treinamento e tempo de predição dos algoritmos implementados para a estimação da $OSNR_{dB_{NL}}$. Quantidade de amostras no conjunto de treinamento 35341, número de amostras do conjunto de testes 15147.

Algoritmo	**Tempo de treinamento [s]**	**Tempo de predição [s]**	**MSE (Validação) [dBm]**
KNN	0,5	1,2	3,507
SVR-RBF	105	36	9,654
SVR-POLY	28308	43	0,087
ANN	300	22	0,477

Os resultados apresentados na Tabela 4.10 mostram desempenho dos diferentes algoritmos de aprendizado de máquina (ML) na tarefa de estimativa da $OSNR_{dB_{NL}}$. O KNN exibe o menor tempo de estimativa, apenas $7,9x10^{-5}$ segundos, porém, os valores de MAE e MSE são relativamente altos, indicando possíveis limitações na precisão das previsões. O SVR-RBF, apesar de apresentar um tempo de estimativa ligeiramente maior do que o KNN, ainda mantém uma performance inferior, com valores de MAE e MSE ainda maiores, tanto para o conjunto de teste quanto para o de validação. Por outro lado, o SVR-POLY destaca-se com tempos de execução semelhantes aos do SVR-RBF, mas com valores de MAE e MSE notavelmente mais baixos, indicando uma melhor precisão na estimativa dos valores reais. Surpreendentemente, o algoritmo ANN, apesar de um tempo de estimativa semelhante ao do SVR-RBF e SVR-POLY, apresenta valores de MAE e MSE baixos para o conjunto de teste, sugerindo um bom desempenho inicial. No entanto, para o conjunto de validação, tanto o MAE quanto o MSE são mais altos, indicando possíveis dificuldades em generalizar para novos dados não vistos durante o treinamento.

Tabela 4.10: Comparação entre o modelo NLIN e os modelos de ML para estimar o $OSNR_{dB_{NL}}$.

Algoritmo	**Tempo para estimar uma amostra [s]**	**MAE (Teste) [dBm]**	**MAE (Validação) [dBm]**	**MSE (Teste) [dBm]**	**MSE (Validação) [dBm]**
Modelo NLIN	4,20	-	-	-	-
KNN	7,90e-5	0,156	1,069	0,054	3,507
SVR-RBF	2,37e-3	0,592	1,832	0,757	9,654
SVR-POLY	2,83e-3	0,005	0,128	3,6e-5	0,087
ANN	1,45e-3	0,101	0.380	0,050	0,477

Em suma, os resultados indicam que o SVR-POLY é o algoritmo mais promissor para esta tarefa de estimação, apresentando uma boa precisão nas previsões, tanto para os dados de teste quanto para os de validação.

A Tabela 4.11 apresenta os resultados dos algoritmos de *Machine Learning* para estimar a $NLIN_{Power}$, incluindo o tempo de treinamento, o tempo de estimação/predição e o erro médio quadrático (MSE). Analogamente ao resultado anterior, para o KNN, observa-se que os tempos de treinamento e estimação são muito baixos. No entanto, o MSE associado é significativamente alto, cerca de 6,6 vezes maior do que o algoritmo anterior. Além disso, a análise do KNN em relação ao conjunto de validação mostra sua limitação, resultando em estimativas ruins à medida que as amostras divergem dos dados de treinamento e teste. O SVR-RBF obteve um erro associado mais baixo para o $NLIN_{Power}$, mantendo praticamente o mesmo tempo de treinamento e teste. A análise em relação ao conjunto de validação indica que o algoritmo consegue fazer excelentes estimativas para amostras nunca antes vistas, evidenciando uma ótima capacidade de generalização. No caso do SVR-POLY, os tempos de treinamento são muito maiores que os dos demais algoritmos, cerca de 245,3 vezes maior que o segundo mais demorado. Apesar disso, os resultados são promissores, com o MSE associado sendo o menor de todos, com ordem de grandeza de 10^{-5}. Para o conjunto de validação, o SVR-POLY demonstrou ser altamente promissor, com todas as métricas apresentando bons resultados e a curva estimada seguindo de perto a curva real, com um erro muito baixo. Por fim, o desempenho do ANN é ruim, apesar do baixo tempo de treinamento e predição. Apresentou um MSE associado alto, e os resultados do conjunto de validação mostraram que as estimativas falharam em se aproximar da curva real. Melhorias podem ser buscadas modificando a rede para torná-la mais complexa e otimizando os hiper parâmetros.

Tabela 4.11: Comparação entre os tempos de treinamento e tempo de predição dos algoritmos implementados para a estimação da $NLIN_{Power}$. Quantidade de amostras no conjunto de treinamento 35341, número de amostras do conjunto de testes 15147.

Algoritmo	**Tempo de treinamento [s]**	**Tempo de predição [s]**	**MSE (Validação) [dBm]**
KNN	0,33	1	23,120
SVR-RBF	95	33	0,401
SVR-POLY	82179	47	9e-5
ANN	335	20	5,09

Os resultados apresentados na Tabela 4.12 revelam uma análise comparativa dos diferentes modelos de estimativa do $NLIN_{Power}$, destacando métricas importantes para avaliar o desempenho de cada algoritmo. O KNN demonstrou uma eficiência computacional notável, com um tempo mínimo para estimar uma amostra de 6,6e-5 segundos. No entanto, os resultados de MAE e MSE para o conjunto de teste foram relativamente altos, indicando uma dificuldade em fazer previsões precisas. Essa dificuldade é ainda mais evidente no conjunto de validação, onde os valores de MAE e MSE foram consideravelmente maiores, sugerindo uma falta de capacidade de generalização do modelo. Em contraste, o SVR-RBF apresentou um tempo mínimo para estimar uma amostra um pouco maior, de 2,18e-3 segundos. No entanto, os resultados para as métricas de erro foram mais promissores, com valores de MAE e MSE menores tanto para o conjunto de teste quanto para o conjunto de validação, indicando uma capacidade melhor de generalização em comparação com o KNN. O SVR-POLY (Polynomial) mostrou resultados ainda mais impressionantes, com um tempo mínimo para estimar uma amostra ligeiramente maior que o RBF, mas com desempenho muito superior. Os valores extremamente baixos de MAE e MSE para ambos os conjuntos de dados sugerem uma precisão excepcional na previsão, com uma capacidade excelente de generalização, conforme evidenciado pelos resultados do conjunto de validação. Finalmente, o ANN apresentou um tempo mínimo para estimar uma amostra relativamente baixo, de 1,32e-3 segundos. Os resultados do conjunto de teste foram satisfatórios, com baixos valores de MAE e MSE, indicando uma boa precisão na previsão dos dados de teste. No entanto, a capacidade de generalização do modelo é questionável, como indicado pelos valores significativamente maiores de MAE e MSE para o conjunto de validação, destacando uma falha em generalizar efetivamente para novos dados.

Tabela 4.12: Comparação entre o modelo NLIN e os modelos de ML para estimar o $NLIN_{Power}$.

Algoritmo	Tempo mínimo para estimar uma amostra [s]	MAE (Teste) [dBm]	MAE (Validação) [dBm]	MSE (Teste) [dBm]	MSE (Validação) [dBm]
Modelo NLIN	4,20	-	-	-	-
KNN	6,6e-5	0,570	3,060	0,513	23,120
SVR-RBF	2,18e-3	0,583	0,489	0,520	0,401
SVR-POLY	3,10e-3	0,007	0,049	0,016	9e-5
ANN	1.32e-3	0,052	1,020	0,005	5,09

O KNN não conseguiu se destacar em nenhum algoritmo, devido ao seu alto erro e à baixa capacidade de generalização. No geral o SVR com *kernel* POLY se destaca como o modelo com o melhor desempenho geral em cenários que não levam o tempo de treinamento em questão. Já o modelo de ANN tem um excelente desempenho na estimação da $OSNR_{dB_{NL}}$ e quando levamos em consideração o tempo de treinamento o modelo de ANN para estimar a $OSNR_{dB_{NL}}$ tem o melhor desempenho, estimação e valores das métricas de erro e R^2. Apesar do custo-beneficio excelente do ANN desempenho do modelo pode ser melhorado realizando uma otimização dos hiper parâmetros, adicionando mais complexidade à rede aumentando o número de camadas intermediárias entre outros ajustes do modelo.

Capítulo 5

Considerações Finais

Neste capítulo é apresentado as conclusões baseadas nos resultados do capítulo anterior e as perspectivas futuras a respeito deste tema.

5.1 Conclusão

O desempenho e resultados obtidos das implementações dos algoritmos estão em linha com as expectativas, especialmente para os modelos KNN e ANN. O modelo ANN para a $OSNR_{dB_{NL}}$ demonstrou um desempenho muito bom tanto para o conjunto de testes quanto para o de validação, apesar de sua arquitetura relativamente simples, composta por apenas três camadas. Por outro lado, o ANN obteve resultados muito aquém do esperado para estimação do $NLIN_{Power}$, visto que os erros associados ao modelo cresceram significativamente. Isso é evidenciado pelos resultados apresentados nas Tabelas 4.7 e 4.8. O KNN obteve um bom desempenho para o conjunto de testes para das saídas $OSNR_{dB_{NL}}$ e $NLIN_{Power}$, porém, os resultados para o conjunto de validação demonstram que o algoritmo tem uma grande limitação, principalmente para amostras muito diferentes das do conjunto de treinamento. Esses resultados foram evidenciados nas Tabelas 4.1 e 4.2.

Quanto aos algoritmos SVR-RBF e SVR-POLY, esperava-se que ambos fornecessem boas estimativas das saídas $OSNR_{dB_{NL}}$ e $NLIN_{Power}$, por serem algoritmos mais complexos com grande capacidade de resolver problemas não lineares. Embora

o SVR-RBF tenha demonstrado um bom desempenho na estimação do $NLIN_{Power}$, os resultados para a $OSNR_{dB_{NL}}$ ficaram aquém do esperado, como evidenciado nas Tabelas 4.3 e 4.4. Em contrapartida, o modelo SVR-POLY apresentou resultados excepcionais para ambas saídas, conforme mostrado nas Tabelas 4.5 e 4.6. No entanto, um ponto negativo do modelo SVR-POLY é o longo tempo de treinamento, que chegou a aproximadamente 8 horas, dificultando a realização de múltiplas simulações para encontrar os melhores parâmetros.

Com bases nos resultados apresentados, pode-se concluir que os modelos de ANN e SVR-POLY são indicados para estimar a $OSNR_{dB_{NL}}$ de novos caminhos ópticos. Já para estimar o $NLIN_{Power}$ os modelos SVR-RBF e SVR-POLY demonstraram excelentes resultados e são indicados para esta finalidade.

5.2 Dificuldades Encontradas

Entre os principais desafios encontrados, além das dificuldades teóricas relacionadas aos sistemas de comunicação por fibras ópticas, destacam-se também as dificuldades na geração de dados sintéticos para os modelos de aprendizado de máquina devido ao NLIN, ao tempo necessário para gerar um grande volume de dados, as simulações envolvendo o SVR-POLY dado ao tempo exigido por simulação.

5.3 Trabalhos Futuros

As perspectivas futuras para continuidade deste trabalho incluem várias direções de pesquisa e aprimoramentos, tais como:

- Explorar técnicas avançadas de otimização de hiper parâmetros para os modelos de ANN, a fim de melhorar ainda mais o desempenho e a capacidade de generalização;
- Investigar arquiteturas de redes neurais mais complexas, adicionando mais camadas e mais neurônios nas camadas intermediárias ou implementando redes

profundas ou redes neurais convolucionais (CNNs), para melhorar a capacidade de capturar relações mais complexas nos dados e melhorar a precisão das previsões;

- Investigar outros algoritmos de aprendizado de máquina, como redes neurais recorrentes (RNNs), modelos baseados em árvores (por exemplo, *Random Forests*) ou métodos ensemble, para avaliar se proporcionam resultados ainda melhores em termos de precisão e generalização;

- Explorar conjuntos de dados mais diversificados e abrangentes, capturando uma gama mais ampla de condições de rede e cenários operacionais, a fim de garantir que os modelos sejam robustos o suficiente para lidar com diferentes situações;

- Testar e validar os modelos desenvolvidos em ambientes de rede reais para verificar sua eficácia e aplicabilidade e considerar a integração com sistemas de gerenciamento de redes ópticas existentes;

Referências

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., e Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agrawal, G. P. (2005a). *Lightwave Technology: Components and Devices*. John Wiley & Sons, Inc.

Agrawal, G. P. (2005b). *Lightwave Technology: Telecommunication Systems*. John Wiley & Sons, Inc.

Agrawal, G. P. (2010). *Fiber-Optic Communication Systems*. Wiley-Interscience.

Agrawal, G. P. (2021). *Fiber-Optic Communication Systems.* John Wiley & Sons, Inc.

Agrell, E. et al. (2016). Roadmap of optical communications. *Journal of Optics*, 18(6):063002.

Alpaydin, E. (2014). *Introduction to Machine Learning*. The MIT Press.

Bashir, D., Montanez, G. D., Sehra, S., Segura, P. S., e Lauw, J. (2020). An information-theoretic perspective on overfitting and underfitting.

Bayvel, P., Behrens, C., e Millar, D. S. (2013). Digital signal processing (DSP) and its application in optical communication systems. In *Optical Fiber Telecommunications VIB*.

Besnoff, J. e Ricketts, D. (2015). Quadrature amplitude modulated (qam) communication link for near and mid-range rfid systems. In *2015 IEEE International Conference on RFID (RFID)*, páginas 151–157.

Botchkarev, A. (2018). Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*. Interdisciplinary Journal of Information, Knowledge, and Management, 2019, 14, 45-79. https://doi.org/10.28945/4184.

Carena, A. et al. (2014). Electronic dispersion pre-compensation in pm-qpsk systems over mixed-fiber links. In *Optical Communication (ECOC), 2014 European Conference on*. IEEE.

Chang, C.-C. e Lin, C.-J. (2001). A library for support vector machines.

Chollet, F. (2015). keras. `https://github.com/fchollet/keras`.

da Silva, M. J., Correia-Filho, M. J., Coelho, L. D., e Martins Filho, J. F. (2017). Impact of the fiber type arrangement on bidirectional mixed-fiber optical links. In *2017 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference (IMOC)*, páginas 1–5.

Dar, R., Feder, M., Mecozzi, A., e Shtaif, M. (2013). Properties of nonlinear noise in long, dispersion-uncompensated fiber links. *Opt. Express*, 21:25685–25699.

Dar, R., Feder, M., Mecozzi, A., e Shtaif, M. (2014a). Accumulation of nonlinear interference noise in fiber-optic systems. *Optics Express*, 22(12):14199–14211.

Dar, R., Feder, M., Mecozzi, A., e Shtaif, M. (2014b). Accumulation of nonlinear interference noise in fiber-optic systems. *Optics express*, 22(12):14199–14211.

de A. Barboza, E., Bastos-Filho, C. J. A., Filho, J. F. M., da Silva, M. J., Coelho, L. D., de Moura, U. C., e de Oliveira, J. R. F. (2016). Local and global approaches for the adaptive control of a cascade of amplifiers. *Photonic Network Communications*, 1:1–14.

Desurvire, E., Bayart, D., Desthieux, B., e Bigo, S. (2002). *Erbium Doped Fiber Amplifiers: Device and System Developments*. John Wiley & Sons.

Dokmanic, I., Parhizkar, R., Ranieri, J., e Vetterli, M. (2015). Euclidean distance matrices: Essential theory, algorithms and applications. *arXiv preprint arXiv:1502.07541*. 17 pages, 12 figures, to appear in IEEE Signal Processing Magazine - change of title in the last revision.

Essiambre, R., Kramer, G., Winzer, P. J., Foschini, G. J., e Goebel, B. (2010). Capacity limits of optical fiber networks. *IEEE Journal of Lightwave Technology*, 28(4):662–701.

Faruk, M. S. e Savory, S. J. (2017). Digital signal processing for coherent transceivers employing multilevel formats. *Journal of Lightwave Technology*, 35(5):1125.

Filho, M. J. C. (2017). Desempenho de sistemas de transmissão óptica com diferentes tipos de fibra.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., e Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hearst, M., Dumais, S., Osuna, E., Platt, J., e Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28.

Hossain, E. (2024). *Machine Learning Crash Course for Engineer*. Springer.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Isaksson, A., Wallman, M., Goransson, H., e Gustafsson, M. G. (2008). Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, 29(14):1960–1965.

Keiser, G. (2010). *Optical Fiber Communications*. McGraw-Hill Education.

Lin, H.-T. e Lin, C.-J. (2005). A study on sigmoid kernels for svm and the training ofnon-psd kernels by smo-type methods.

Mitchell-Hill, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mohri, M. e Afshin Rostamizadeh, A. T. (2018). *Foundations of Machine Learning*. The MIT Press.

Morais, R. M. e Pedro, J. (2018). Machine learning models for estimating quality of transmission in dwdm networks. *IEEE/OSA Journal of Optical Communications and Networking*, 10(10):D84–D99.

Napoli, A., Maalej, Z., Sleiffer, V. A. J. M., Kuschnerov, M., Rafique, D., Timmers, E., Spinnler, B., Rahman, T., Coelho, L. D., e Hanik, N. (2014). Reduced complexity digital back-propagation methods for optical communication systems. *IEEE Journal of Lightwave Technology*, 32(7):1351–1362.

Padierna, L., Carpio, M., Rojas-Domínguez, A., Puga, H., e Fraire, H. (2018). A novel formulation of orthogonal polynomial kernel functions for svm classifiers: The gegenbauer family. *Pattern Recognition*, 84:211–225.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., e Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Poggiolini, P., Bosco, G., Carena, A., Curri, V., Jiang, Y., e Forghieri, F. (2014). The gn-model of fiber non-linear propagation and its applications. *Journal of Lightwave Technology*, 32(4):694–721.

Poggiolini, P. e Jiang, Y. (2017). Recent advances in the modeling of the impact of nonlinear fiber propagation effects on uncompensated coherent transmission systems. *IEEE Journal of Lightwave Technology*, 35(3):458–480.

Ramaswami, R., Sivarajan, K. N., e Sasaki, G. H. (2010). *Optical Networks: A Practical Perspective*. Morgan Kaufmann Publishers, 3rd. edição.

Rezende, S. M. (2022). *Introduction to Electronic Materials and Devices*, volume 3. Springer.

Rumelhart, D. E., Hinton, G. E., e Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Russell, S. J. e Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. The MIT Press.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, (3):210–229.

Schmidt, J. (2023). Testing for overfitting.

Senior, J. M. e Jamro, M. Y. (2009). *Optical fiber communications : principles and practice*. Financial Times/Prentice Hall.

Temprana, E. et al. (2015). Overcoming kerr-induced capacity limit in optical fiber transmission. *Science*, 348(6242):1445–1448.

Thurnhofer-Hemsi, K., López-Rubio, E., Molina-Cabello, M. A., e Najarian, K. (2020). Radial basis function kernel optimization for support vector machine classifiers. *IEEE Transactions on Neural Networks and Learning Systems*.

van Ginkel, J. R. (2019). Significance tests and estimates for r^2 for multiple regression in multiply imputed datasets: A cautionary note on earlier findings, and alternative solutions. *Multivariate Behavioral Research*, 54(4):514–529.

Wang, W. e Lu, Y. (2018). Analysis of the mean absolute error (mae) and the root mean square error (rmse) in assessing rounding model. *IOP Conference Series: Materials Science and Engineering*, 324:012049.

Ye, X. (2023). *Applications of Artificial Intelligence to Control and Analyze the Performance of Fiber Optic Transmission Systems.* PhD thesis, Institut Polytechnique de Paris, Paris.

Apêndice A

As implementações do pré-processamento, modelos de ML em Python e visualização dos dados estão disponíveis no seguinte repositório do github: eQoT_optical_network

Anexo A

Neste anexo é exibido o código utilizado no desenvolvimento do projeto.

O cálculo dos coeficientes χ_1 χ_2 e χ_3 requer a soma de até cinco índices. Um código (escrito em Matlab) que computa esses coeficientes usando método de integração Monte Carlo foi fornecido em (Dar et al., 2014b). O código fornecido é: .

```
function main()

function [NLIN_Power,OSNRdB_NL,BER_NL,OSNRdB,BER] =
    NonlinearPerformance(NumCh,ChSpacing,Spans,BaudRate)

% Nch - number of channels inclunding teh channel of interest
% ChSpacing - Channel spacing in GHz between the channels
% Spans - Structure containing the parameters of each span
%   Spans().gamma
%   Spans().beta2
%   Spans().alpha
%   Spans().L
%   Spans().PdBmCh
%   Spans().ModFormatCh
%   Spans().DeltaPdBIntLeft
%   Spans().ModFormatIntLeft
%   Spans().DeltaPdBIntRight
%   Spans().ModFormatIntRight
%   Spans().Fn
% BaudRate
```

```
20
21 N = 1000000;    % Number of integration points in algorithm [14].
       Should
22 % be set such that the relative error is desirably small.
23 T = 1000/BaudRate;
24 planckc    = 6.626068e-34;                % Planck's constant [m^2*
       kg/s]
25 lightspeed = 2.99792458e8;                % Speed of light [m/s]
26 lambda = 1.55e-6;
27
28 % Calculate NLIN
29 NLIN_Power = 0;
30 for k =1:NumCh
31
32     ChSpacing_norm = abs(k-(NumCh+1)/2)*ChSpacing./BaudRate;
33
34     if ChSpacing_norm == 0
35         R = 2*pi*(rand(5, N)-0.5*ones(5, N));
36
37         % X1
38         w0 = R(1,:) - R(2,:) + R(3,:);
39         argInB = (w0 < pi).*(w0 > -pi);
40         arg1 = (R(2,:) - R(3,:)).*(R(2,:) - R(1,:));
41
42         % X2
43         w1 = R(1,:) - R(2,:) + R(3,:);
44         arg2 = (R(2,:) - R(3,:)).*(R(4,:) - R(1,:));
45         IndMF1 = (w1 < pi).*(w1 > -pi);
46
47         % X21
48         w2 = R(4,:) - R(1,:) - R(3,:);
49         arg21 = (R(2,:) - R(4,:)).*(R(2,:) + w2);
50         IndMF2 = (w2 < pi).*(w2 > -pi);
51
52         % X3
```

```
53        w3 = R(1,:) - R(2,:) + R(4,:) + R(3,:) - R(5,:);
54        arg3 = (R(4,:) - R(5,:)).*(R(4,:) - w3);
55        IndMF3 = (w3 < pi).*(w3 > -pi);
56
57        LinkSum_chi1 = 0;
58        LinkSum_chi2 = 0;
59        LinkSum_chi21 = 0;
60        LinkSum_chi3 = 0;
61
62        Lgth = 0;
63        for j = 1:length(Spans)
64            LinkSum_chi1 = LinkSum_chi1 + (10^((Spans(j).PdBmCh-30)
    /10))^(3/2)...
65                *Spans(j).gamma*exp(1i*Spans(j).beta2/T^2*arg1*Lgth
    ).*...
66                (1 - exp(1i*Spans(j).beta2/T^2*arg1*Spans(j).L -
    Spans(j).alpha/10*log(10)*Spans(j).L))...
67                ./(Spans(j).alpha/10*log(10) - 1i*Spans(j).beta2/T
    ^2*arg1);
68            LinkSum_chi2 = LinkSum_chi2 + (10^((Spans(j).PdBmCh-30)
    /10))^(3/2)...
69                *Spans(j).gamma*exp(-1i*Spans(j).beta2/T^2*arg2*
    Lgth).*...
70                (1 - exp(-1i*Spans(j).beta2/T^2*arg2*Spans(j).L -
    Spans(j).alpha/10*log(10)*Spans(j).L))...
71                ./(Spans(j).alpha/10*log(10) + 1i*Spans(j).beta2/T
    ^2*arg2);
72            LinkSum_chi21 = LinkSum_chi21 + (10^((Spans(j).PdBmCh
    -30)/10))^(3/2)...
73                *Spans(j).gamma*exp(-1i*Spans(j).beta2/T^2*arg21*
    Lgth).*...
74                (1 - exp(-1i*Spans(j).beta2/T^2*arg21*Spans(j).L -
    Spans(j).alpha/10*log(10)*Spans(j).L))...
75                ./(Spans(j).alpha/10*log(10) + 1i*Spans(j).beta2/T
    ^2*arg21);
```

```
76          LinkSum_chi3 = LinkSum_chi3 + (10^((Spans(j).PdBmCh-30)
    /10))^(3/2)...
77              *Spans(j).gamma*exp(-1i*Spans(j).beta2/T^2*arg3*
    Lgth).*...
78              (1 - exp(-1i*Spans(j).beta2/T^2*arg3*Spans(j).L -
    Spans(j).alpha/10*log(10)*Spans(j).L))...
79              ./(Spans(j).alpha/10*log(10) + 1i*Spans(j).beta2/T
    ^2*arg3);
80          Lgth = Lgth + Spans(j).L;
81      end
82
83      X0 = abs(sum(LinkSum_chi1.*argInB)/N).^2;
84      X1 = sum(abs(LinkSum_chi1.*argInB).^2)/N;
85      X2 = real(sum(LinkSum_chi2.*LinkSum_chi1.*IndMF1.*argInB))/
    N;
86      X21 = real(sum(LinkSum_chi21.*LinkSum_chi1.*IndMF2.*argInB)
    )/N;
87      X3 = real(sum(LinkSum_chi3.*LinkSum_chi1.*IndMF3.*argInB))/
    N;
88
89      % calculate NLIN
90      switch Spans(1).ModFormatCh
91          case 'DP-QPSK'
92              kur = 1; kur3 = 1;
93          case 'DP-16QAM'
94              kur = 33/25; kur3 = 49/25;
95          case 'DP-64QAM'
96              kur = 29/21; kur3 = 30195/13566;
97      end
98      NLIN_var = (9/8)^2*16/81*(2*X1 + (kur-2)*(4*X2+X21) ...
99          + (kur3-9*kur+12)*X3 - (kur-2)^2*X0 + X1 + (kur-2)*X2);
100
101  else
102      R = 2*pi*(rand(4, N) - 0.5*ones(4, N));
103      w0 = R(1,:) - R(2,:) + R(3,:);
```

```
104         IndMF = (w0 < pi).*(w0 > -pi);
105         arg1 = (R(2,:) - R(3,:)).*(R(2,:) + 2*pi*ChSpacing_norm - R
    (1,:));
106         arg2 = (R(2,:) - R(3,:)).*(R(4,:) + 2*pi*ChSpacing_norm - R
    (1,:));
107         LinkSum_chi1 = 0;
108         LinkSum_chi2 = 0;
109
110         Lgth = 0;
111         NLIN_var = 0;
112         if sign(k-(NumCh+1)/2) > 0
113             ModFormatIntRight = Spans(1).ModFormatIntRight;
114             for j = 1:length(Spans)
115                 Pint = 10^((Spans(j).PdBmCh + Spans(j).
    DeltaPdBIntRight - 30)/10);
116                 LinkSum_chi1 = LinkSum_chi1 + sqrt(10^((Spans(j).
    PdBmCh-30)/10))*Pint...
117                     *Spans(j).gamma*exp(1i*Spans(j).beta2/T^2*arg1*
    Lgth).*...
118                     (1 - exp(1i*Spans(j).beta2/T^2*arg1*Spans(j).L
    - Spans(j).alpha/10*log(10)*Spans(j).L))...
119                     ./(Spans(j).alpha/10*log(10) - 1i*Spans(j).
    beta2/T^2*arg1);
120                 LinkSum_chi2 = LinkSum_chi2 + sqrt(10^((Spans(j).
    PdBmCh-30)/10))*Pint...
121                     *Spans(j).gamma*exp(-1i*Spans(j).beta2/T^2*arg2
    *Lgth).*...
122                     (1 - exp(-1i*Spans(j).beta2/T^2*arg2*Spans(j).L
     - Spans(j).alpha/10*log(10)*Spans(j).L))...
123                     ./(Spans(j).alpha/10*log(10) + 1i*Spans(j).
    beta2/T^2*arg2);
124                 Lgth = Lgth + Spans(j).L;
125                 if j==length(Spans) || ~strcmp(Spans(j+1).
    ModFormatIntRight,ModFormatIntRight)
126                     switch ModFormatIntRight
```

```
127                         case 'DP-QPSK'
128                             kur = 1;
129                         case 'DP-16QAM'
130                             kur = 33/25;
131                         case 'DP-64QAM'
132                             kur = 29/21;
133                     end
134                     chi1 = 4*sum(abs(LinkSum_chi1.*IndMF).^2)/N;
135                     chi2 = 4*real(sum(LinkSum_chi2.*LinkSum_chi1.*
    IndMF))/N;
136                     LinkSum_chi1 = 0;
137                     LinkSum_chi2 = 0;
138                     Lgth = 0;
139                     NLIN_var = NLIN_var + (9/8)^2*16/81*(chi1 + (
    kur-2)*chi2 + 2*chi1/4 + (kur-2)*chi2/4);
140                 end
141             end
142         else
143             ModFormatIntLeft = Spans(1).ModFormatIntLeft;
144             for j = 1:length(Spans)
145                 Pint = 10^((Spans(j).PdBmCh + Spans(j).
    DeltaPdBIntLeft - 30)/10);
146                 LinkSum_chi1 = LinkSum_chi1 + sqrt(10^((Spans(j).
    PdBmCh-30)/10))*Pint...
147                     *Spans(j).gamma*exp(1i*Spans(j).beta2/T^2*arg1*
    Lgth).*...
148                     (1 - exp(1i*Spans(j).beta2/T^2*arg1*Spans(j).L
    - Spans(j).alpha/10*log(10)*Spans(j).L))...
149                     ./(Spans(j).alpha/10*log(10) - 1i*Spans(j).
    beta2/T^2*arg1);
150                 LinkSum_chi2 = LinkSum_chi2 + sqrt(10^((Spans(j).
    PdBmCh-30)/10))*Pint...
151                     *Spans(j).gamma*exp(-1i*Spans(j).beta2/T^2*arg2
    *Lgth).*...
152                     (1 - exp(-1i*Spans(j).beta2/T^2*arg2*Spans(j).L
```

```
    - Spans(j).alpha/10*log(10)*Spans(j).L))...
153                     ./(Spans(j).alpha/10*log(10) + 1i*Spans(j).
    beta2/T^2*arg2);
154                 Lgth = Lgth + Spans(j).L;
155                 if j==length(Spans) || ~strcmp(Spans(j+1).
    ModFormatIntLeft,ModFormatIntLeft)
156                     switch ModFormatIntLeft
157                         case 'DP-QPSK'
158                             kur = 1;
159                         case 'DP-16QAM'
160                             kur = 33/25;
161                         case 'DP-64QAM'
162                             kur = 29/21;
163                     end
164                     chi1 = 4*sum(abs(LinkSum_chi1.*IndMF).^2)/N;
165                     chi2 = 4*real(sum(LinkSum_chi2.*LinkSum_chi1.*
    IndMF))/N;
166                     LinkSum_chi1 = 0;
167                     LinkSum_chi2 = 0;
168                     Lgth = 0;
169                     NLIN_var = NLIN_var + (9/8)^2*16/81*(chi1 + (
    kur-2)*chi2 + 2*chi1/4 + (kur-2)*chi2/4);
170                 end
171             end
172         end
173     end
174     NLIN_Power = NLIN_Power + NLIN_var;
175 end
176
177 ASE_power = 0;
178 for j = 1:length(Spans)
179     if j == length(Spans)
180         GdB = Spans(1).PdBmCh - Spans(j).PdBmCh + Spans(j).alpha*
    Spans(j).L;
181     else
```

```
182         GdB = Spans(j+1).PdBmCh - Spans(j).PdBmCh + Spans(j).alpha*
    Spans(j).L;
183     end
184     ExtraNoise = 10^( (Spans(j).SpanLossdB - GdB)/10);
185     if ExtraNoise < 1
186         ExtraNoise = 1;
187     end
188     ASE_power = ASE_power + BaudRate*1e9*planckc*lightspeed/lambda
    *...
189         Spans(j).Fn*(10^(GdB/10))*ExtraNoise;
190 end
191
192 SNR = 1e-3*10^(Spans(1).PdBmCh/10)/ASE_power;
193 SNR_NL = 1e-3*10^(Spans(1).PdBmCh/10)/(ASE_power + NLIN_Power);
194
195 OSNRdB = 10*log10(BaudRate/12.5*SNR);
196 OSNRdB_NL = 10*log10(BaudRate/12.5*SNR_NL);
197
198 switch Spans(1).ModFormatCh
199     case 'DP-QPSK'
200         BER = 0.5*erfc(sqrt(SNR/2));
201         BER_NL = 0.5*erfc(sqrt(SNR_NL/2));
202     case 'DP-16QAM'
203         BER = 2/log2(16)*( 1 - 1/sqrt(16) )*erfc(sqrt(3*SNR
    /2/(16-1)));
204         BER_NL = 2/log2(16)*( 1 - 1/sqrt(16) )*erfc(sqrt(3*SNR_NL
    /2/(16-1)));
205     case 'DP-64QAM'
206         BER = 2/log2(64)*( 1 - 1/sqrt(64) )*erfc(sqrt(3*SNR
    /2/(64-1)));
207         BER_NL = 2/log2(64)*( 1 - 1/sqrt(64) )*erfc(sqrt(3*SNR_NL
    /2/(64-1)));
208 end
209 end
```

Listing A.1: Implementação modificada para o modelo NLIN modificado

Anexo B

Neste Anexo é exibido o código utilizado no desenvolvimento do projeto.

O código em matlab fornecido em (Filho, 2017) precisou ser ligeiramente modificado para gerar um grande volumes de dados através de *loops* aninhados.

```
% System parameters
clear;
IniSim = clock;
lightspeed = 0.299792458;  % Speed of light [10^9*m/s]
lambda = 1.55; % reference wavelength [um]

%SMF -> DisPar = 16.7, alpha = 0.2, gama = 1.3
%NZDSF Dispar = 3.8, alpha = 0.22, gama = 1.5
%PSCF -> Dispar = 20.1, alpha = 0.17 gama = 0.8

DisPar_list = [16.7,  3.8, 20.1];
alpha_list = [0.2, 0.22, 0.17];
gama_list = [1.3, 1.5, 0.8];
L_list = 80:10:120; %[20, 30, ..., 200]
NumCh_list = 5:2:15; %[3, 5, 7, 11, ..., 15]
NumSpans_list = 5:1:15;
ModFormatCh_list = {'DP-QPSK', 'DP-16QAM', 'DP-64QAM'};

% Channel spacing [GHz]
ChSpacing = 50;
% Baud-rate [GHz]
```

```
22 BaudRate = 28;
23 % Number of channels
24 NumCh = 0;
25 % Number of spans
26 NumSpans = 0;
27 % dispersion parameter [ps/nm/km]
28 DispPar = 0.0;
29 % Nonlinearity coefficient in [1/W/km]
30 Spans.gamma = 0.0;
31  % Dispersion coefficient [ps^2/km]
32 Spans.beta2 = DispPar*lambda^2/2/pi/lightspeed;
33 Spans.alpha = 0.0;    % Fiber loss coefficient [dB/km]
34 Spans.L = 0;         % Span length [km]
35 Spans.PdBmCh = 0;       % Average input power [dBm]
36 Spans.ModFormatCh = 'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP-64QAM'
37 Spans.DeltaPdBIntLeft = 0;
38 Spans.ModFormatIntLeft = 'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP-64
       QAM'
39 Spans.DeltaPdBIntRight = 0;
40 Spans.ModFormatIntRight = 'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP
       -64QAM'
41 Spans.Fn = 10^(16/10); % Noise Figure
42 Spans.SpanLossdB = 0; % Span Loss [dB]
43
44 for idx_L = 1:length(L_list)
45 for idx_NumCh = 1:length(NumCh_list)
46 for idx_NumSpans = 1:length(NumSpans_list)
47 for idx_ModFormatCh = 1:length(ModFormatCh_list)
48 for idx_FiberParam = 1:3
49 initSimIt = clock;
50 % Number of channels
51 NumCh = NumCh_list(idx_NumCh);
52 % Number of spans
53 NumSpans = NumSpans_list(idx_NumSpans);
54 % dispersion parameter [ps/nm/km]
```

```
DispPar = DisPar_list(idx_FiberParam);
% Nonlinearity coefficient in [1/W/km]
Spans.gamma = gama_list(idx_FiberParam);
% Dispersion coefficient [ps^2/km]
Spans.beta2 = DispPar*lambda^2/2/pi/lightspeed;
% Fiber loss coefficient [dB/km]
Spans.alpha = alpha_list(idx_FiberParam);
  % Span length [km]
Spans.L = L_list(idx_L);
% Average input power [dBm]
Spans.PdBmCh = 0;
%'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP-64QAM'
Spans.ModFormatCh = ModFormatCh_list{idx_ModFormatCh};
Spans.DeltaPdBIntLeft = 0;
%'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP-64QAM'
Spans.ModFormatIntLeft = ModFormatCh_list{idx_ModFormatCh};
Spans.DeltaPdBIntRight = 0;
% 'DP-QPSK'; % 'DP-QPSK', 'DP-16QAM', 'DP-64QAM'
Spans.ModFormatIntRight = ModFormatCh_list{idx_ModFormatCh};


Spanss = repmat(Spans,1,NumSpans);


PdBmCh = -3:0.5:5;
BER_NL = ones(1,length(PdBmCh));

for j = 1:length(PdBmCh)
    for jj = 1:NumSpans
        Spanss(jj).PdBmCh = PdBmCh(j);
    end
    [NLIN_Power,OSNRdB_NL,BER_NL(j),OSNRdB,BER] =
   NonlinearPerformance(
    NumCh,ChSpacing,Spanss,BaudRate
    );
```

```
89      PChOptdBm = ((4*PdBmCh(j) - OSNRdB + 10*log10(BaudRate/12.5)
90                      - 10*log10(2*NLIN_Power))/3 - 10
91      );
92      fprintf('NLIN Power=%g dBm | PChOpt=%g dBm |
93          OSNR_NL=%g dB | OSNR=%g dB | BER=%g | BER_NL=%g \n',...
94          10*log10(NLIN_Power/1e-3), PChOptdBm, OSNRdB_NL, OSNRdB,
    BER, BER_NL(j));
95      % Abre o arquivo em modo de adicao ('a' para append)
96      NLIN_Power_aux = 10*log10(NLIN_Power/1e-3);
97      fid = fopen('daba_base_parametros_fiber.csv', 'a');
98      % Verifica se o arquivo foi aberto corretamente
99      if fid == -1
100         error('N o foi poss vel abrir o arquivo para adicao');
101     end
102
103     fprintf(fid, '%.10f,%.2f,%d,%.2f,%.2f,%d,%.2f,%.2f,%.10f,%.2f
    ,%.2f,%s,%d,%s,%d,%s,%.10f,%d,%.10f,%.10f,%.10f,%.10f\n', ...
104         lightspeed, lambda, NumCh, ChSpacing, BaudRate, NumSpans,
    DispPar, Spanss(1).gamma, Spanss(1).beta2, Spanss(1).alpha,
    Spanss(1).L, ...
105         PdBmCh(j), Spanss(1).ModFormatCh, Spanss(1).DeltaPdBIntLeft
    , Spanss(1).ModFormatIntLeft, Spanss(1).DeltaPdBIntRight, Spanss
    (1).ModFormatIntRight, ...
106         Spanss(1).Fn, Spanss(1).SpanLossdB, PChOptdBm,
    NLIN_Power_aux, OSNRdB_NL, BER_NL(j), OSNRdB, BER);
107
108     fclose(fid); % Feche o arquivo
109     disp('Valores dos parametros salvos em "
    daba_base_parametros_fiber.csv" com sucesso.');
110
111 end
112 clear Spanss;
113 EndSimIt = clock;
114 fprintf('Total Simulation Time: %g h ; %g min ; %g sec \n',...
115     floor(etime(EndSimIt,initSimIt)/3600),floor(etime(EndSimIt,
```

```
    initSimIt)/60) ...
116     - floor(etime(EndSimIt,initSimIt)/3600)*60,floor(etime(EndSimIt
    ,initSimIt)) - ...
117     floor(etime(EndSimIt,initSimIt)/60)*60);
118 end
119 end
120 end
121 end
122 end
123 EndSim = clock;
124 fprintf('Total Simulation Time: %g h ; %g min ; %g sec \n',...
125     floor(etime(EndSim,IniSim)/3600),floor(etime(EndSim,IniSim)/60)
    ...
126   -floor(etime(EndSim,IniSim)/3600)*60,floor(etime(EndSim,IniSim))
    - ...
127     floor(etime(EndSim,IniSim)/60)*60);
```

Listing B.1: Implementação modificada para o modelo NLIN modificado