UNIVERSIDADE FEDERAL DO PARANÁ

JOÃO GUILHERME SAUER

CONCEPÇÃO OTIMIZADA DE REDES PROFUNDAS DE CONEXÕES ALEATÓRIAS APLICADA À PREVISÃO DE CURTÍSSIMO PRAZO DE DEMANDA DE ENERGIA ELÉTRICA

CURITIBA
2021

JOÃO GUILHERME SAUER

CONCEPÇÃO OTIMIZADA DE REDES PROFUNDAS DE CONEXÕES ALEATÓRIAS APLICADA À PREVISÃO DE CURTÍSSIMO PRAZO DE DEMANDA DE ENERGIA ELÉTRICA

Tese apresentada ao programa de Pós-Graduação em Engenharia Elétrica, Setor de Tecnologia, da Universidade Federal do Paraná como requisito a obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Prof. Leandro dos Santos Coelho

CURITIBA
2021

CATALOGAÇÃO NA FONTE

S255c Sauer, João Guilherme

Concepção otimizada de redes profundas de conexões aleatórias aplicada à previsão de curtíssimo prazo de demanda de energia elétrica [recurso eletrônico]/ João Guilherme Sauer – Curitiba: UFPR, 2021.

Tese (Doutorado) apresentada ao programa de Pós-graduação em Engenharia Elétrica, da Universidade Federal do Paraná como requisito a obtenção do título de Doutor em Engenharia Elétrica.

Orientador: Prof. Leandro dos Santos Coelho

1. Redes neurais (Computação). 3. Otimização matemática. 3. Engenharia elétrica. I. Coelho, Leandro dos Santos. II. Título. III. Universidade Federal do Paraná. CDD 620

Bibliotecária: Vilma Machado CRB9/1563

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **JOÃO GUILHERME SAUER** intitulada: **CONCEPÇÃO OTIMIZADA DE REDES PROFUNDAS DE CONEXÕES ALEATÓRIAS APLICADA À PREVISÃO DE CURTÍSSIMO PRAZO DE DEMANDA DE ENERGIA ELÉTRICA**, sob orientação do Prof. Dr. LEANDRO DOS SANTOS COELHO, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 27 de Julho de 2021.

Assinatura Eletrônica
03/08/2021 17:49:48.0
LEANDRO DOS SANTOS COELHO
Presidente da Banca Examinadora

Assinatura Eletrônica
27/07/2021 15:41:37.0
GIDEON VILLAR LEANDRO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
28/07/2021 09:34:35.0
ROBERTO ZANETTI FREIRE
Avaliador Externo (PONTIFICA UNIVERSIDADE CATóLICA DO PARANá)

Assinatura Eletrônica
27/07/2021 15:45:16.0
ALCEU DE SOUZA BRITTO JR
Avaliador Externo (PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ)

"Só sei que nada sei"

Sócrates (470 a.C. - 399 a.C.)

AGRADECIMENTOS

Agradeço a Deus.

A minha esposa que esteve todo o tempo ao meu lado, me motivando para não desistir no meio do caminho.

Aos meus pais, por sempre me darem o suporte mesmo estando longe fisicamente.

Ao Prof. Leandro dos Santos Coelho, por sempre acreditar que eu poderia percorrer este caminho.

A todos os docentes do departamento de Engenharia Elétrica da UFPR.

Aos amigos e colegas.

RESUMO

A contribuição principal desta tese é a de propor um projeto otimizado de redes profundas de conexões aleatórias (do inglês *deep random vector functional link neural network*, DRVFL) por meio de duas abordagens de otimização a evolução diferencial e a otimização Bayesiana. O projeto otimizado da DRVFL foi aplicado na previsão de demanda de energia elétrica em curtíssimo prazo, ou seja, com horizonte de previsão de um passo à frente. Foram utilizados dois estudos de caso, o primeiro sendo o consumo por hora de energia da região Sul do Brasil durante o mês de janeiro de 2017, e o outro o consumo de energia diário durante o ano de 1990 na Polônia. Além disso, esta tese aborda o uso de uma pré-análise de suas características por meio do seu comportamento sazonal e o uso do método de eliminação de características recursivo (do inglês *recursive feature elimination,* RFE) para a remoção de características menos significantes. A seguir aplicou-se além de DRVFL, diferentes modelos de previsão tais como modelo auto-regressivo integrado de médias móveis, modelo auto-regressivo integrado de médias móveis sazonal, rede neural convolucional, florestas aleatórias, florestas aleatórias quantílicas, rede neural artificial *feedforward*, regressão por vetores de suporte, rede neural com estados de eco, aprendizado de máquina extremo, memória de longo e curto prazos, mínimos quadrados em batelada, máquina aumentada de gradiente simplificado e aumento de gradiente extremo. Como métrica de desempenho das previsões foi adotada a raiz de erro quadrático médio logarítmico (do inglês *root mean squared log error*, RMSLE). No contexto de uma base de comparação, avaliou-se também os modelos de previsão usando as características originais, sem o uso de RFE. Os resultados obtidos pelo modelo DRVFL mostram sua viabilidade e potencialidades na previsão de séries temporais de demanda de energia elétrica em curtíssimo prazo, e que o uso de uma abordagem de extração das características melhora as previsões em termos de variância, precisão e custo computacional em relação ao uso de todas as características originais quanto ao RMSLE.

Palavras-chave: aprendizado profundo*,* redes profundas de conexões aleatórias*,* meta-heurísticas de otimização, otimização Bayesiana. evolução diferencial, redes neurais artificiais, previsão de séries temporais, previsão de demanda de energia elétrica.

ABSTRACT

The contribution of this thesis is an optimized project of deep random vector functional link neural network (DRVFL), through two different approaches for its optimization, known as differential evolution and Bayesian optimization, applied in problems of forecast demand of electric energy in a very short period using only the regression of its historic series for one step ahead. It was used two cases of study, being the first related to the consume of energy per hour in the south region of Brazil during the January month of 2017 and the other one is the diary consumption of energy during the year of 1999 in Poland. This work approaches the usage of a pre-analyses of its features through its seasonal behaviour and the usage of the methods of recursive feature elimination for the removal of its features less important. In the next step, it was applied beside DRVFL other models for forecast, including: auto-regressive integrated moving average, seasonal auto-regressive integrated moving average, convolutional neural network, random forest, quantile random forest, feed forward neural network, support vector regression, echo state neural network, extreme machine learning, long short-term memory, ordinary least square, light gradient boosting machine and extreme gradient boosting, using as performance metric the root mean squared log error (RMSLE). In an extra step, the same algorithms would be again used, however with all the original features. The obtained results using DRVFL showed satisfactory results for both datasets, reaching a better accuracy in the forecast when compared using the performance metric RMSLE, where the extraction of only the best features allowed even a better result when compared with the usage of all the original ones.

Keywords: deep learning, deep random vector functional link, optimization meta-heuristics, Bayesian optimization, differential evolution, artificial neural network, time series forecasting, electric energy demand forecasting.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANN	*Artificial Neural Network*
AR	*Auto Regressive*
ARIMA	*Auto Regressive Integrated Moving Average*
ARMA	*Auto Regressive Moving Average*
BDL	*Bayesian Deep Learning*
ELM	*Extreme Learning Machine*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
DE	*Differential Evolution*
DL	*Deep Learning*
DNN	*Deep Neural Network*
DRVFL	*Deep Random Vector Functional Link*
ESN	*Echo State Network*
FFNN	*Feed Forward Neural Network*
FT	*Fourier Transform*
GA	*Genetic Algorithm*
GP	*Gaussian Process*
GPU	*Graphical Processing Unit*
LIGHTGBM	*Light Gradient Boosting Machine*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MLOps	*Machine Learning Operations*
MSE	*Mean Square Error*
NN	*Neural Network*
OLS	*Ordinary Least Square*
QRF	*Quantile Random Forest*
RF	*Random Forest*
RFE	*Recursive Feature Elimination*
RMSE	*Root Mean Square Error*
RMSLE	*Root Mean Squared Log Error*
RNN	*Recurrent Neural Network*
RVFL	*Random Vector Functional Link*

SARIMA	*Seasonal Auto Regressive Integrated Moving Average*
SNN	*Shallow Neural Network*
SVM	*Support Vector Machine*
SVR	*Support Vector Regression*
XGBOOST	*Extreme Gradient Boosting*

SUMÁRIO

1 INTRODUÇÃO

O setor de energia elétrica é essencial para a economia de um país. Nas últimas décadas a privatização e a desregulamentação deste setor permitiu investimentos acentuados na previsão de demanda para o planejamento de produção e consumo. No Brasil, o setor teve sua restruturação nas últimas duas décadas, permitindo a livre negociação da compra e venda de energia o que permitiu uma forma independente e autônoma de crescimento vertical nas atividades de geração, transmissão e distribuição de energia elétrica. Por meio de informações coletadas no passado e armazenadas na forma de séries históricas, os setores de planejamento de concessionárias passaram a analisá-las para então se obter conhecimento ou padrões necessários para tentar prever comportamentos dinâmicos no futuro. Em termos gerais, pode-se dizer que séries temporais englobam informações que foram baseadas em um conjunto de observações ou medidas coletadas em um determinado intervalo de tempo, estas podem ter seus valores dependentes um dos outros, permitindo assim uma base que pode ser compreendida no estudo de métodos de análise e/ou previsão (Hyndman e Athanasopoulos, 2018).

Conforme mencionado em Montgomery *et al.* (2015), pode-se classificar a previsão de séries temporais como de curtíssimo-prazo, curto-prazo, médio-prazo ou longo-prazo. A séries temporais de curtíssimo prazo são eventos que envolvem a predição somente de pequenos períodos no futuro, como de alguns minutos até uma hora. Para a previsão de curto prazo estima-se uma faixa de algumas horas até uma semana a frente. Enquanto a previsão de médio-prazo pode ser classificada como previsões que envolve alguns meses a frente, sendo que previsões de longo-prazo seriam anos a frente, podendo alcançar até dez anos à frente.

Em previsões de curtíssimo prazo, é essencial para a confiabilidade e eficiência da operação do setor elétrico, permitindo analisar possíveis distorções nos próximos períodos, sendo assim uma previsão de qualidade para valores futuros de demanda de energia elétrica. Este tipo de previsão permite uma melhoria procedimentos intrínsecos aos processos de distribuição de energia, permitindo sanar desperdícios, escassez e/ou a má alocação de recursos, tal como uma alteração na configuração em sistemas de distribuição de energia elétrica (Jana *et al.*, 2020). Outro exemplo também poderia ser o consumo em demanda de energia em áreas estratégicas da economia incluindo o consumo de energia em estufas na China e sua

emissão de gases poluentes (Jiang *et al.*, 2021), o impacto de modificações na legislação para o consumo de energia para prédios na Coreia do Sul (Kim *et al.*, 2021), a correlação entre o consumo de energia elétrica e o crescimento na Itália (Magazzino *et al.*, 2021), o impacto de uma pandemia no consumo de energia (Qarnain *et al.*, 2020), entre outros. Com o foco de otimizar e automatizar tais tarefas pode-se minimizar os problemas de se ter uma previsão em excesso (do inglês *overforecast*) ou por falta (do inglês *underforecast*). Para tal, várias pesquisas têm sido sugeridas, com considerável expectativa de diminuição dos custos e aumento de benefícios (Lim e Zohren 2021), (Liu *et al.*, 2021), (Sezer *et al.*, 2020) e (Singh e Dhiman, 2021).

Uma das áreas de estudo com pesquisas relevantes é o aprendizado de máquina (do inglês *machine learning*, ML), que pode ser definida como um conjunto de algoritmos e métodos que permitem um computador aprender um determinado comportamento ou padrão automaticamente, a partir de exemplos ou observações anteriores.

Dada a diversidade dos problemas de séries temporais vários estudos usando aprendizado de máquina emergiram, sendo que vários estudos comparando os modelos clássicos de previsão e o uso de redes neurais artificiais têm sido apresentados na literatura, para citar Shard e Patil (1992) que compararam os resultados de previsão de séries temporais do modelo ARIMA com redes neurais artificiais ou Hill *et al.* (1997) que compararam uma rede neural com outros modelos estatísticos clássicos no uso de previsão de séries temporais e tomadas de decisões. Mais recentemente, Lim e Zohren (2020) apresentaram uma revisão da literatura em termos de aprendizado de máquina para aplicações de previsão de séries temporais e sugerem tendências do uso de abordagens mais aprimoradas de previsão. Pode-se mencionar que a utilização de modelos de aprendizado de máquina apresentou resultados competitivos em diferentes tipos de séries temporais e isso contribuiu para que contribuições fossem propostas, a citar o projeto de métodos não supervisionados para o ajuste de valores de hiperparâmetros (Zhang *et al.*, 2021) e a detecção de valores atípicos em séries temporais usando a treinamento por métodos de retropropagação de erro (Vishwakarma e Elsawah, 2020).

Como um aprendizado de máquina pode ter várias informações e procedimentos que podem afetar a modelagem de seus sistemas, um novo conceito, denominado operações de aprendizado de máquina (do inglês *machine learning operations*, MLOps) foi proposto pela comunidade científica para se tentar resolvê-los

(Talagala, 2018). MLOps é um conjunto de práticas e *softwares* que são usados de forma a facilitar a criação, reprodução, aprimoramento e controle de como um modelo de aprendizado de máquina pode ser melhor operacionalizado. Na Figura 1.1 é apresentado um diagrama do ciclo de vida de um sistema de aprendizado de máquina, que é comumente observado em representações de MLOps. Inicialmente, na fase de manipulação de dados no ciclo de MLOps, tem-se uma análise das séries temporais, permitindo a identificação de possíveis características, a coleta de dados, ou mesmo uma possível manipulação para se melhorias quanto ao uso dela, e finalmente a definição de como a série temporal será armazenada para facilitar seu posterior processamento. A seguir, na fase de aprendizado de máquina, se faz a exploração e a extração dos dados para obter-se o máximo possível de dados relacionados aos requerimentos e métricas de desempenho, partindo-se então para treinamento, validação e teste do mesmo para se obter um modelo eficiente de aprendizado de máquina. Em seguida, na fase de desenvolvimento, decide-se como tal modelo(s) de aprendizado de máquina será usado, programado, empacotado e testado antes de ser enviado para o ambiente de pré-produção que pode ser distribuído para as pessoas responsáveis pelo lançamento em produção, configuração, monitoramento e validação. Finalmente, com o sistema em produção, a fase de coleta de propostas de melhoria, coleta maior de dados e novos requerimentos geram informações que podem iniciar um novo ciclo de projeto baseado em MLOps.

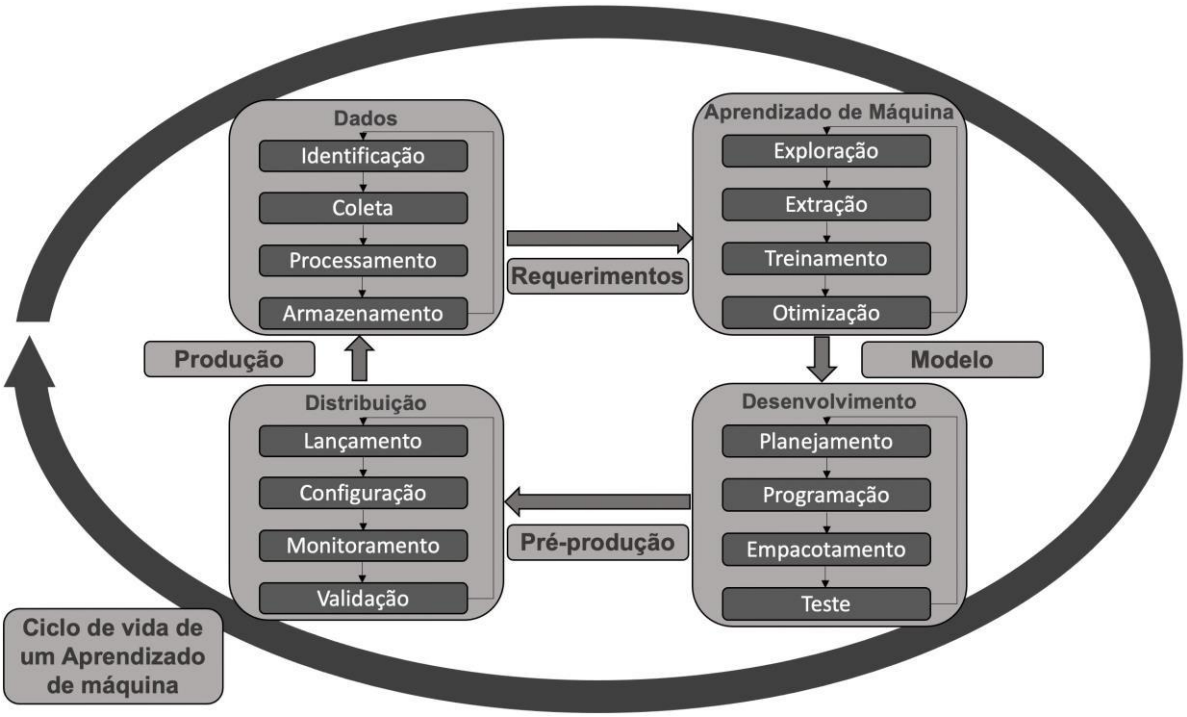

Figura 1.1: Ciclo de vida de um aprendizado de máquina

Um ramo do aprendizado de máquina é o aprendizado profundo que é um conjunto de algoritmos que tentam modelar abstrações de alto nível de dados usando várias camadas de processamento, compostas de várias transformações lineares e/ou não-lineares.

Com o aumento da popularidade e possibilidades de aplicação de abordagens de aprendizado profundo, alguns focos de pesquisa têm sido apresentados na literatura recente. Por exemplo, Tsitsiklis e Van Roy (1997) propuseram o aprendizado profundo por reforço onde o sistema interage com o ambiente e usa a realimentação dos resultados de saída do sistema para receber “recompensas” por seus atos e assim tenta obter ações que lhe deem recompensas maiores. Alguns exemplos estão presentes em robótica (Kober e Peters, 2012), operações de entrega em sistemas de inventário (Kara e Dogan, 2018) e predições de investimentos (Zhang *et al.*, 2020). No caso do aprendizado profundo por reforço, a abordagem de realimentação do sistema possibilitou avanços em diversas áreas relacionadas. Na robótica, Bee (2013) apresentou fundamentos de aprendizado para robôs quanto a jogar *Air Hockey*. Neste contexto, Koç *et al.* (2018) apresentaram um sistema para o jogo de tênis de mesa onde é possível analisar o jogo usando um sistema que contém apenas imagens como dados de entrada. Cabe mencionar que empresas de tecnologia, tal como o Google, por meio da sua empresa subsidiária denominada DeepMind, fazem pesquisas na área, a citar um sistema de aprendizado não-supervisionado com foco em jogos clássicos do sistema Atari (Mnih *et al.*, 2015) ou a Microsoft, que por meio de seu time de pesquisadores também fazem pesquisas, como por exemplo uma análise do uso de aprendizado profundo no uso de iterações para uma conversa informal com humanos (Gao *et al*., 2020).

Outra importante abordagem de aprendizado foi proposta por Huang *et al.* (2006) é a máquina de aprendizado extremo (do inglês, *extreme learning machine*, ELM). Esta abordagem usa uma rede neural artificial *feedforward* onde os pesos dos neurônios presente(s) na(s) camada(s) oculta(s) não são modificados durante o processo de treinamento.

No projeto clássico de uma rede neural artificial *feedforward* os valores dos pesos iniciais da rede são gerados aleatoriamente com distribuição uniforme e após utiliza-se um método de otimização para treinamento, onde os clássicos são o método de retropropagação do erro, quase-Newton e Levenberg-Marquardt. No caso da ELM, o treinamento é substituído pela resolução desde uma matriz inversa na camada de

saída de rede neural *feedforward* por meio dos métodos dos mínimos quadrados, eliminando-se o processo iterativo de treinamento baseado em otimização. Desde então, sugestões têm sido propostas tais como (i) a máquina de aprendizado extremo incremental (Huang *et al.,* 2006), onde os neurônios (também denominados nós) da camada oculta são escolhidos de forma sequencial, conforme seu desempenho em termos de métrica(s) de desempenho, (ii) a máquina de aprendizado extremo evolutivo, que aplica uma abordagem de algoritmo evolutivo denominada evolução diferencial para otimizar a estrutura do modelo de ELM (Zhu *et al.,* 2005), (iii) ELM com o uso do meta-heurística de otimização da área de inteligência de enxames denominada lobo cinzento (Cheng, Feng e Niu, 2020), e (iv) ELM com projeto focado em aprendizado não supervisionado e aplicado em controle estatístico na área de manufatura de uma determinada empresa (Viharos e Jakab, 2020).

Quanto a abordagens de redes neurais artificiais com foco em aprendizado não-iterativo, outro método relevante foi sugerido por Pao *et al.* (1994), as redes de conexões aleatórias (do inglês *random vector functional link neural network*, RVFL), foi usado para predizer os valores dos preços do barril de petróleo, sendo então reavaliado por Zhang *et al.* (2020) que usando a mesma base de dados sugeriu o uso de RVFL combinado com métodos de decomposição de séries temporais. Algumas pesquisas mencionam que a RVFL apresenta resultados promissores em termos de desempenho e tempo de execução para problemas de previsão de séries temporais e classificação de dados, conforme sugerido em Ren *et al.* (2016), Zhang e Suganthan (2017), Henrique e Luz (2018), Tang *et al.* (2018), Zhang *et al.* (2019) e Rakesh e Suganthan (2020).

Katuwal *et al.* (2019) sugeriram a inclusão de abordagens de aprendizado profundo no projeto da RVFL, denominando de redes profundas de conexões aleatórias (do inglês *deep random vector functional link neural network*, DRVFL), onde usa-se camadas ocultas (intermediárias) sobrepostas e conectadas entre si e o aprendizado profundo na forma de um aprendizado por comitê (do inglês *ensemble*) (EDRVFL) que permite o treinamento de uma rede DRVFL única, modificando os pesos de saída para que sejam transformados em pequenos modelos que podem ser treinados separadamente e calculando-se a média de seus valores para se obter o valor do peso de saída final. Na área de estabilidade de sistemas de energia elétrica tem-se a contribuição do modelo proposto por Zhang *et al.* (2021) no qual adota o uso de DRVFL. Wang *et al.* (2021) aplicaram DRVFL no diagnóstico de transformadores

de energia e fizeram um estudo sobre o melhoramento de seu uso, analisando os nós da camada escondida e sugerindo a remoção daqueles da DRVFL que não estão contribuindo para o sistema. Há também a proposta de Alalimi *et al.* (2021) em que um otimizador de busca esférico para os valores de hiperparâmetros para a predição da produção de energia na China foi proposto.

Na fase de treinamento do aprendizado profundo da DRVFL, os valores de seus neurônios e os pesos de suas conexões são escolhidos aleatoriamente com distribuição uniforme ou mesmo pré-definidos pelo projetista. Assim, pode-se empregar algoritmos construtivos baseados em meta-heurísticas de otimização para ajustar os valores iniciais dos hiperparâmetros de projeto da DVRL durante o processo de aprendizado. Tem-se por exemplo o uso da otimização em uma rede neural artificial para auxiliar no gerenciamento de energias solares (Moayedi e Mosavi, 2021), ou ainda uma combinação de meta-heurísticas como otimizadores clássicos de uma mesma rede neural artificial, gerando uma diminuição de aproximadamente 13% nos custos de operação de sistemas de energia (Ikeda e Nagai, 2021), ou ainda o uso de otimizadores de enxame de partículas e algoritmos genéticos para otimizar os modelos de previsão de carga em sistemas elétricos (Bouktif *et al.,* 2021).

Por meio da análise da qualidade dos dados de entrada, pode-se verificar as características ou propriedades especificas do fenômeno a ser observado, nota-se que algumas delas podem ter maior ou menor importância para o sistema como um todo, fazendo com que possam ser removidas caso não estejam ajudando o sistema no seu aprendizado ou que estejam sobre-ajustando (do inglês *overfitting*) o modelo adotado (Bishop, 2006). Para isso, métodos de seleção de características (do inglês *features*) em abordagens de aprendizado de máquina em problemas de previsão de séries temporais têm sido propostos, tal como eliminação recursiva de características (do inglês *recursive feature elimination*, RFE) (Dadebayev *et al.*, 2021), o algoritmo Boruta, que usa a classificação de florestas aleatórias para fazer sua eliminação (Liaw e Wiener, 2002) ou o algoritmo de seleção de características sequencial (Pudil *et al.*, 1994.

1.1 Justificativa

De acordo Wen *et al.* (2020), o uso de aprendizado de máquina na previsão de séries temporais tem apresentado resultados promissores nos últimos anos. No

caso da área de demanda de energia elétrica, o uso de redes neurais artificiais é uma tendência ascendente, conforme mencionado por Pagani *et al.* (2018)

Um dos métodos recentemente propostos na literatura, o DRVFL, tem mostrado resultados competitivos quando comparado com a abordagem denominada ELM, conforme apresentado por Del Ser (2021). O DRVFL, tendo uma estrutura similar com ELM, tem como característica que os valores de entrada e os valores de saída estão diretamente conectados, enquanto o ELM acaba omitindo tais conexões. Apesar de simples o treinamento não-iterativo da DRVFL, os valores iniciais dos parâmetros da camada oculta gerados de forma aleatória podem causar instabilidade ao sistema. Por isso, frequentemente o ajuste de seus parâmetros pode ser otimizado por meio de uma função ativação onde os valores das variáveis da decisão são os parâmetros do DRVFL.

Em problemas de otimização, uma meta-heurística é um método heurístico com propósito de ser mais facilmente adaptado para diferentes problemas de otimização. As meta-heurísticas podem fornecer soluções próximas das ótimas em um tempo computacional razoável para resolver problemas complexos. No entanto, a meta-heurística não garante que a solução seja a melhor possível e não define quão próximas estão as soluções obtidas da solução ótima (Amoroso, 2020).

As meta-heurísticas de otimização podem então ser adotadas para a resolução de problemas de otimização global conforme mencionado em Daniel *et al.* (2020), Kashif *et al.* (2019) e Dokeroglu *et al.* (2019). Com isso, pode-se buscar um conjunto de parâmetros mais apropriado para o projeto da DRVFL que métodos de ajuste por tentativa e erro. A comunidade científica já propôs otimizações para o modelo RVFL, como o uso do otimizador de busca *Hunger Games* (AbuShanab *et al.*, 2021) ou ainda o uso de DE para a otimização, como sugerido por Zhou *et al.,* (2020) o uso de otimizadores para o modelo DRVFL não foi ainda analisado pela comunidade científica.

Uma dessas meta-heurísticas é a evolução diferencial (DE), proposta por Storn e Price (1997) que permite a resolução de problemas de otimização não lineares e que permite a presença de funções objetivo não diferenciáveis. Por ter poucos parâmetros de ajuste e por ter uma boa convergência de dados, é muito usado em competições, como a IEEE CEC que foi analisado por Molina *et al.*, (2017), sendo então um dos otimizador escolhidos nesta tese. Outra opção promissora a ser considerada por esta tese é a otimização Bayesiana, proposta inicialmente por

Močkus (1975), mostra-se competitiva com outros métodos propostos mais recentemente na literatura do tema por fazer o uso de uma função de estimação de probabilidade distribuída, gerando resultados promissores que tiveram uma performance melhores que outros algoritmos genéticos quando se comparando a métrica de desempenho RMSE (Pelikan *et al.,* 1999).

A métrica de desempenho usada foi a RMSLE, por proporcionar uma avaliação dos erros de modo relativo, ou seja, os valores de erros continuam o mesmo não importando que a escala de erro seja aumentada.

Ao longo do estudo desta tese, notou-se também que uma pré-análise dos dados, por meio de uma geração de características ou variáveis preditoras, pode representar um passo importante para se preparar o modelo de previsão para lidar com perfis de comportamento complexo nas séries temporais. Por exemplo, com a extração apenas das características que estejam correlacionadas com a saída (variável a ser prevista), é possível diminuir a dependência do sistema para um determinado tipo de dado, causando *overfitting*.

1.2 Objetivos

O objetivo geral desta tese é propor dois sistemas: um combinando DRVFL e algoritmo de otimização DE e outro combinando o DRVFL com o algoritmo de otimização Bayesiano para a previsão de demanda de energia elétrica de séries temporais a curtíssimo prazo, ou seja, com horizonte de previsão de um passo à frente. Neste contexto, é concebida uma pré-seleção das características da série temporal usando o método de extração de características RFE, de tal forma que os seus resultados apresentem uma previsão com melhor acurácia ao ser comparada com outros 13 modelos de algoritmos de aprendizado de máquina em termos da medida de desempenho RMSLE.

1.2.1 Objetivos Específicos

Os objetivos específicos desta tese são os seguintes:

- Propor um modelo combinado de DRVFL e evolução diferencial e um outro modelo combinado de DRVFL e Bayesiana que podem realizar predições acuradas em termos do RMSLE para a previsão de demanda do consumo de energia elétrica;

- Comparar o modelo otimizado DRVFL proposto com modelos de previsão denominados auto-regressivo integrado de médias móveis, auto-regressivo integrado de médias móveis sazonal, rede neural convolucional, florestas aleatórias, florestas aleatórias quantílicas, rede neural *feedforward*, regressão por vetores de suporte, rede neural com estado de eco, aprendizado de máquina extremo, memória longa a curto prazo, mínimos quadrados, máquina aumentada de gradiente simplificado e aumento de gradiente extremo por meio da métrica RMSLE para os caso de previsão de um passo à frente;
- Apresentar uma comparação dos modelos de previsão previamente analisados usando otimização Bayesiana para ajuste de hiperparâmetros;
- Realizar uma análise dos resultados decorrentes do uso do algoritmo de extração de características RFE nos modelos apresentados, aliados a otimização dos algoritmos através do uso de otimização por DE e otimização Bayesiana.

1.3 Contribuições

Seguindo-se os objetivos mencionados, esta tese apresenta contribuições, estas listadas a seguir:

- Propor um novo modelo de previsão combinando DVRF com métodos de otimização incluindo DE e otimização Bayesiana e este aplicado na previsão de séries temporais;
- Extrair as informações relacionadas a frequência da série temporal e analisar sua importância para o treinamento nos modelos de aprendizado de máquina;
- Comparar diferentes modelos de aprendizado de máquina otimizados por algoritmos de otimização em dois estudos de casos relacionados a previsão de séries temporais de demanda de energia elétrica.

1.4 Organização da Tese

O restante desta tese está organizado da seguinte forma:

- Capítulo 2 – Revisão da Literatura. Neste capítulo são apresentados os principais conceitos associados ao tema, seguindo uma ordem

cronológica dos estudos relacionados a previsão da demanda de energia elétrica e séries temporais.

- Capítulo 3 – Fundamentos de Aprendizado. Neste capítulo são discutidos os métodos usados na área de demanda do consumo de energia a curtíssimo prazo. Primeiramente, fundamentos do aprendizado de máquina são mencionados, seguido por uma explicação sobre modelos clássicos de ML, e terminando com uma explicação sobre os modelos de redes neurais artificiais mais usados na área de demanda do consumo de energia.
- Capítulo 4 – Experimentos. Neste capítulo a metodologia usada no desenvolvimento da nova abordagem e como a comparação entre os diferentes será detalhada. Também é explicado neste capítulo a pré-análise dos dados para avaliação dos mesmos quanto usados como variáveis previsoras. Ao final do capítulo, a análise dos resultados de um estudo comparativo entre modelos é abordada.
- Capítulo 5 – Considerações Finais. Neste capítulo são apresentadas as conclusões finais, verificando se os objetivos propostos foram alcançados ou não e é apresentado um levantamento de possíveis desdobramentos para futuras pesquisas em áreas relacionadas ao tema desta tese.

2 REVISÃO DA LITERATURA

A revisão da literatura está dividida primeiramente em uma breve explicação sobre séries temporais, focando nos tipos e usos delas na área de demanda de energia elétrica. Após, modelos clássicos para previsão de séries temporais são abordados. E finalizando o capítulo, uma análise detalhada sobre o uso de aprendizado de máquina para serem aplicados nas previsões de demanda de energia elétrica é apresentada.

As séries temporais podem ser usadas como suporte na configuração e solução de problemas com diferentes focos de pesquisa, tais como a predição de valores futuros, classificação de séries temporais, agrupamento de séries temporais e a detecção de anomalias em séries temporais. Cabe mencionar que geralmente execução de tais tarefas envolve o projeto de modelos e/ou algoritmos. A Figura 2.1 mostra as possibilidades de uso de uma série temporal, sendo que no caso de detecção de anomalias, é possível de se ocorrer tanto em problemas de agrupamento ou de classificação de séries temporais.

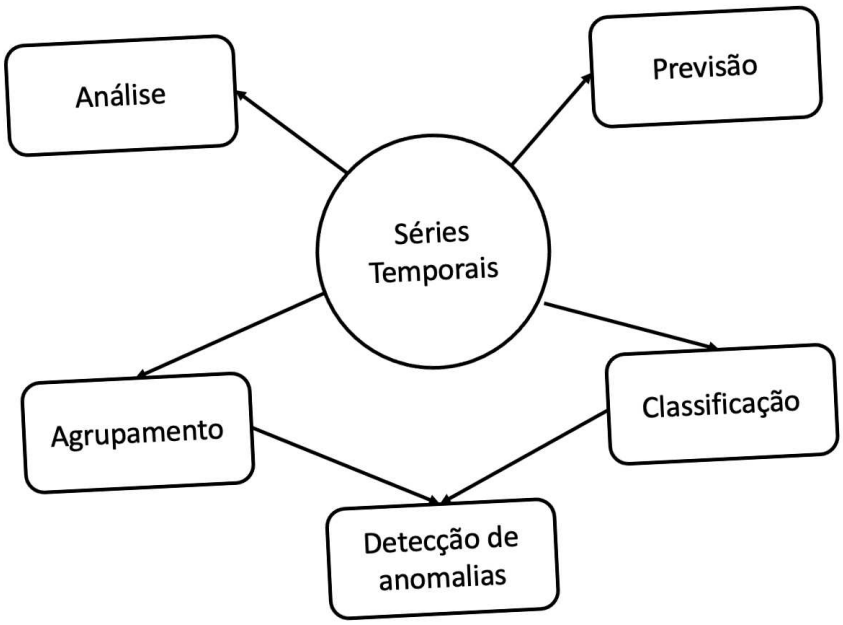

Figura 2.1: Possibilidades de uso de uma série temporal

Um dos primeiros modelos clássicos usado em séries temporais é o modelo de regressão linear baseado na solução de um problema de mínimos quadrados ordinário (do inglês *ordinary least squares*, OLS), que permite a obtenção de uma

relação linear entre os valores da(s) variável(is) preditoras e os valores de previsão (Lakshmi *et al.,* 2021).

Outro modelo sugerido na literatura clássica de análise e previsão de séries temporais é o auto-regressivo integrado de médias móveis (do inglês *auto-regressive integrated moving average*, ARIMA), que pode ser estimado seguindo-se a abordagem de Box e Jenkins que permite obter valores futuros usando-se uma função linear (Box *et al.*, 1994). Para se solucionar o problema do modelo ARIMA para séries que contém componente sazonal, uma modificação do modelo permite o uso de variáveis para se representar por exemplo sazonalidades (Farsi *et al.,* 2021).

Seguindo-se ainda os modelos clássicos de séries temporais, tem-se os modelos de regressão dinâmica, que usam informação quanto a autocorrelação dos erros, pois os ruídos da série não podem ser ignorados, para que eles não afetem os valores preditos (Temkeng *et al.,* 2021).

Outro modelo estatístico usado é o modelo auto-regressivo de média móvel, que é caracterizado por uma formulação para uma função que combina as ideias dos modelos auto-regressivos com modelos de médias móveis em uma forma compacta (ou parcimoniosa) de modelo onde o número de parâmetros usados seja mínimo (Moon *et al.,* 2021).

Existe outras propostas importantes em séries temporais, tais como processos Gaussianos para predições (Ghasemi *et al.,* 2021) e modelos de suavização exponencial (Smyl, 2020).

Algumas variantes propostas na literatura tal como o método de *bagging* (do inglês *bootstrap aggregating*), ou uma forma de aprendizado de comitê (do inglês *ensemble learning*) proposto por Breiman (1996), que permitem agregar múltiplas versões de um modelo de previsão, sendo que cada modelo na forma de árvore de regressão é treinado individualmente e combinados usando um processo de mediana.

Em termos de redes neurais artificiais, existem várias abordagens clássicas “rasas” baseadas em perceptron multicamadas e redes neurais funções de base radial. Em contrapartida, o aprendizado profundo é geralmente baseado em uma arquitetura de rede neural artificial, por exemplo uma rede neural convolucional que contém três ou mais camadas escondidas (Skansi, 2018; Lim e Zohren, 2020).

A popularidade de aprendizado de máquina pode ser verificada na Figura 2.2, que mostra o número de publicações que utilizam o modelo de aprendizado profundo em suas propostas. A pesquisa foi concluída em junho de 2021 usando a *software*

Dimensions (Williams, 2018) que adotou como principais fonte de pesquisa as bases de dados do IEEE, ArXiv, Springer e Elsevier, compreendendo os anos de 2013 a 2021. As palavras chaves foram "*deep learning* + *time series*". Em 2013 houve a primeira publicação referente ao assunto, sendo seguida por dois anos consecutivos sem ter nenhuma outra publicação. Nos 4 anos seguintes, ainda é possível notar que menos de 50 publicações ocorreram por ano. Porém, de 2019 em diante, passou a barreira de 100 publicações e no ano seguinte teve um aumento de aproximadamente 50%, E no ano de 2021 foram verificadas 92 publicações.

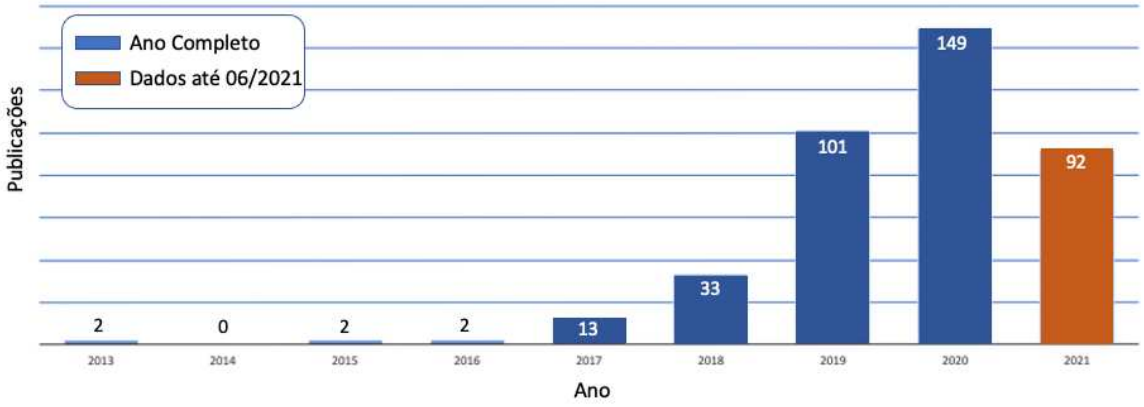

Figura 2.2: Resumo de publicações referentes a aprendizado profundo entre 2007 e 2021.

Na Figura 2.3 adotou-se o aplicativo VOSviewer (Van Eck e Waltman, 2013) que usou a rede de conexões entre os campos de estudos relacionados a palavra-chave "*machine* + *learning* + *time series*". No total foram encontradas 8480 publicações, sendo que 801 campos de estudos foram citados mais de 10 vezes. Tais campos de estudo foram representados com cores diferentes conforme os grupos em que se encontravam, sendo que o tamanho do nó representa a quantidade de ocorrências dos campos de estudo e a linha entre dois nós representa uma conexão entre os campos estudados. Quanto mais grossa a linha, maior o número de conexões. O maior grupo, como esperado é na área de exatas, tais como ciência da computação, inteligência artificial e aprendizado de máquina. Porém, as áreas de pesquisas são estendidas para outras áreas, como o setor de energia solar, energias renováveis, energia eólica, aprendizado de máquina, computação em nuvem e outros setores. E é interessante notar a conexões entre todos esses setores, mostrando que áreas diferentes estão conectadas entre si, mostrando que os resultados obtidos em uma área de estudo podem ser usados em outras áreas.

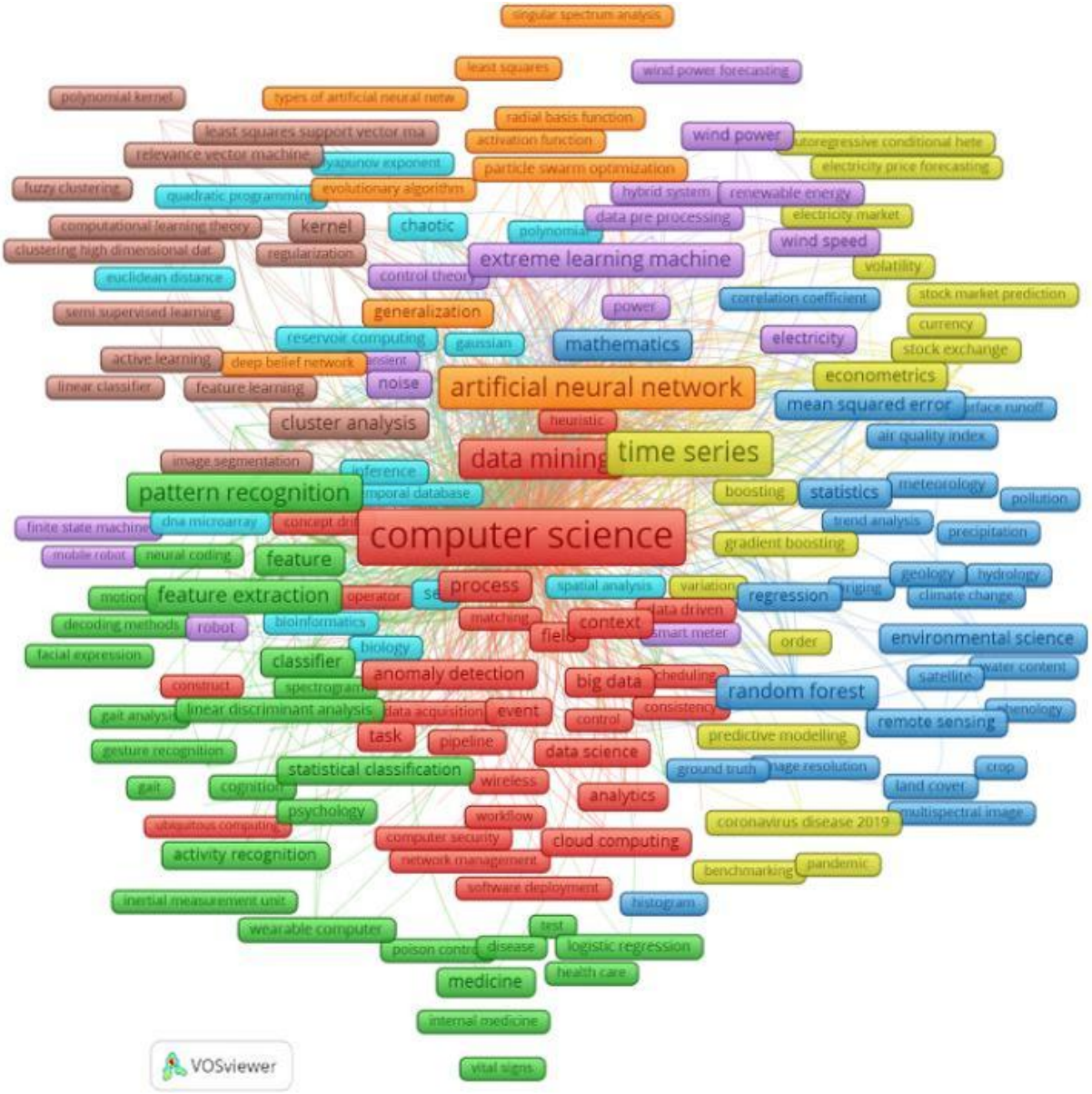

Figura 2.3: Rede de campos de estudo em aprendizado de máquina

Entre os dados coletados, é possível de se notar propostas de análise de diferentes modelos que são combinados, para se verificar se é possível uma melhor acurácia nas previsões. A citar por exemplo Gurnani *et al.* (2017) que propuseram uma comparação entre vários modelos, tais como o ARIMA, regressão por vetores de suporte (do inglês *support vector regression*, SVR), aumento de gradiente extremo (do inglês *extreme gradient boosting*, XGBoost) e rede neural auto-regressiva (do inglês *auto-regressive neural network*) em uma aplicação de previsão de vendas.

Quando se considera períodos de previsão de curtíssimo prazo (do inglês *short-term),* Shahid *et al.* (2021) sugerem o uso de uma decomposição de modo empírico com o modelo memória longa a curto prazo (do inglês *long short-term*

memory, LSTM) justamente com o uso do XGBoost para previsão de energia gerada em parques eólicos da Europa.

Ao se aplicar modelos híbrido, que combinam dois ou mais modelos de aprendizado de máquina, acaba-se gerando um sistema que pode se beneficiar em termos da obtenção de melhores resultados.

Várias propostas têm abordado o uso de modelos híbridos de previsão, que podem ser classificados em três diferentes estruturas: paralelo, em séries e paralelo em série (Hajirahimi e Khashei, 2019). Na Figura 2.4 é ilustrada uma abordagem de modelo híbrido em paralelo.

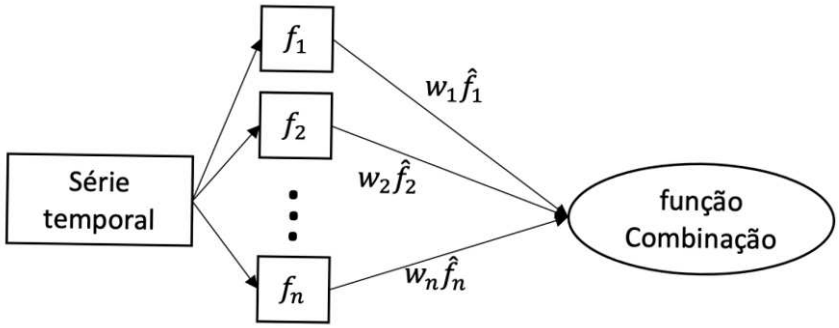

Figura 2.4: Representação de um modelo híbrido em paralelo

Os dados de série temporal são recebidos em separado por cada um dos modelos representado pela letra f e tomam uma decisão isolada, que passa então para uma função de combinação que pode usar diferentes sistemas de peso w para se tomar a decisão do valor final, sendo que cada modelo teria o seu próprio peso, conforme proposto por Wei *et al.* (2019) no uso de previsão de uso diário do gás natural na China e na Grécia.

Na Figura 2.5 tem-se ilustrada a representação de modelos híbridos em série.

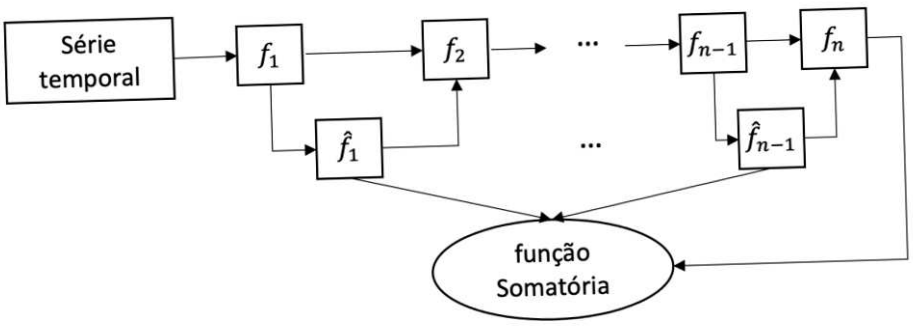

Figura 2.5: Representação de um modelo híbrido em série

Os dados de saída de um modelo f_{x-1} podem ser usados como dados de entrada em outro modelo f_x, e como o valor $\hat{f}_{x-1}$ usado na função somatória, conforme proposto primeiramente por Zhang (2003) que propôs uma decomposição da série temporal em partes lineares e não lineares que foram então processadas respectivamente por ARIMA e uma rede neural perceptron multicamadas. No entanto, em modelos híbridos paralelo em série, como sugerido pelo nome, é uma mistura entre os modelos em série e em paralelo, para se extrair a vantagem de ambas as estruturas, conforme mencionado por Khashei e Bijari (2010) que projetou múltiplos modelos de ARIMA para diferentes séries temporais, conforme apresentado na Figura 2.6.

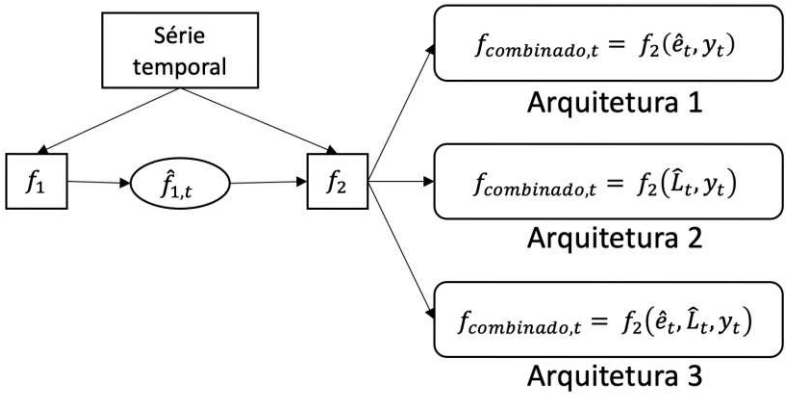

Figura 2.6: Representação de um modelo híbrido em paralelo em série

Neste caso, a série temporal é passada para os modelos f_1 e f_2 como em um modelo híbrido em paralelo, onde ambos modelos simultaneamente recebem os dados, porém, em um tempo t, os resultados $\hat{f}_{1,t}$ de f_1, que consiste em uma parte residual de erro $\hat{e}_t$ e um valor previsto $\hat{L}_{1,t}$ são também passados em série como dados de entrada para o outro modelo. Consequentemente, dependendo dos valores de erro de previsão $\hat{e}_t$, $\hat{L}_{1,t}$, passados como valores do primeiro modelo, combinado com o valor original de entrada y_t, pode-se ter uma das seguintes arquiteturas:

- Arquitetura 1:

$$f_{combinado,\,t} = f_2(\hat{e}_t,\, y_t), \qquad t = 1,2,\ldots,T \tag{2.1}$$

onde f_2 é a função não linear que é determinada por um modelo inteligente. Foi proposta primeiramente por Khashei e Bijari (2010) que usaram um modelo ARIMA para f_1 e rede neural artificial para f_2.

- Arquitetura 2:

$$f_{combinado,\,t} = f_2(\hat{L}_t,\, y_t), \qquad t = 1,2,\dots,T \tag{2.2}$$

onde $\hat{L}_t$ é o valor previsto obtido do primeiro modelo que é um tipo de modelo estatístico.

- Arquitetura 3:

$$f_{combinado,\,t} = f_2(\hat{e}_t, \hat{L}_t,\, y_t),\ t = 1,2,\dots,T \tag{2.3}$$

onde ambos os valores $\hat{L}_t$ e $\hat{e}_t$ são valores obtidos do primeiro modelo. Essa arquitetura foi proposta por Khashei e Bijari (2011), onde um modelo ARIMA e uma rede neural artificial foram novamente usados como f_1 e f_2, respectivamente.

Na área de análise de anomalias, que aplica modelos de aprendizado de máquina para se detectar anomalias em séries temporais que podem causar ruídos em uma análise de previsão, Braei e Wagner (2020) apresentaram uma revisão de modelos estatísticos, aprendizado de máquina e aprendizado profundo, analisando a sua eficiência em diferentes conjuntos de dados que servem de referência para o estudo de análise de anomalias.

Em outra área de estudo, tem-se uma pré-análise dos dados de entrada, antes mesmo de serem processados por um modelo, Barandas *et al.* (2020) sugeriram uma biblioteca, denominada *time series feature extraction library*, que pode analisar uma série temporal, separá-la em pequenos pedaços ao longo do tempo, aonde cada bloco de informação é então denominada de janela, e então retornar uma matriz aonde cada linha representa uma janela e cada coluna sendo uma característica obtida para aquela janela em especifico, sendo que pode gerar mais de sessenta tipos de características por janela. Em caminho similar tem-se então (Zhang *et al.*, 2021) que sugerem o uso de OLS para a seleção de características usando uma classificação linear.

Quando se implementa tais métodos que dependem de dados de entrada para alinharem seus pesos internos dos nós e conexões, pode-se ter um problema quando se tem séries temporais com muitas redundâncias que podem afetar o resultado final, tem-se então o efeito denominado “maldição da dimensionalidade” (Wang *et al.*, 2020). O RVFL pode ser uma abordagem eficiente para ajustar os pesos durante a fase de treinamento, sendo possível escolher aleatoriamente um valor e manter o

durante todo a fase de treinamento e que os dados de entrada devem ser usados diretamente nos dados de saída (Wu *et al.,* 2020). Uma importante contribuição para o modelo foi realizada por Katuwal *et al.* (2019), que notaram ser possível criar múltiplas camadas ocultas, ao invés de apenas uma camada singular, sendo que todas são conectadas entre si e, como no RVFL, os pesos das camadas ocultas são aleatoriamente gerados e não mudam ao longo da fase de treinamento.

Outra proposta similar com RVFL foi sugerida por Huang *et al.* (2006), que remove a necessidade de se ter os dados de entrada sendo usados na camada de saída, que foi chamada então de aprendizado de máquina extremo (do inglês *extreme learning machine,* ELM), que usaram um modelo de rede neural com apenas uma camada oculta e que não requer uma abordagem de treinamento que utiliza retropropagação do sinal de erro para ajustar os pesos dos neurônios e suas conexões. Tal estratégia permitiu obter resultados promissores em problemas de classificação, incluindo aplicações complexas, sendo mais rápida que muitos modelos clássicos de aprendizado de máquina.

A ELM tem encontrado aplicações bem-sucedidas em várias áreas, tais como na análise de resistência em estruturas de concretos (Shariati *et al.*, 2019), no reconhecimento de expressão facial (Adegun e Vadapali, 2020) e na modelagem de motores turbojet (Zhao *et al.*, 2018).

Aprimorando a abordagem de Huang *et al*. (2006), Zou *et al.* (2016) sugeriram um sistema híbrido de ELM com treinamento por método da retropropagação do erro. Usando uma estratégia híbrida de ELM com redes neurais *feedfoward*, Han *et al.* (2014) conseguiram melhorar em aproximadamente 20% os resultados em uma aplicação de reconhecimento de emoções quando comparado com outras estratégias consideradas mais promissoras nesta área de pesquisa.

Sugerindo uma estratégia similar, Eshtay *et al.* (2018) obtiveram melhores promissores em termos de acurácia dos resultados, estabilidade contra os ruídos dos dados de entrada, complexidade e tempo de execução em problemas de diagnóstico médico usando um sistema híbrido de ELM e otimização baseada em colônia de abelhas.

Para se obter modelos com uma melhor flexibilidade e melhor capacidade de análise, vários projetos começaram a propor modelos híbridos, sendo muitas vezes que o aprendizado de máquina é usado como a parte de predição e um modelo de

meta-heurística acaba sendo usado para a parte de otimização do sistema (Sina *et al.,* 2019).

O clássico artigo Yao (1999) menciona algoritmos de redes neurais artificiais combinados a meta-heurísticas de otimização com o intuito de evoluir o sistema em três diferentes níveis: otimizar o peso das conexões entre suas camadas, adaptar as topologias da arquitetura para diferentes tarefas e determinar mudança nas regras de aprendizado, como por exemplo realizar a mudança das regras de quando os pesos das conexões devem ser atualizados.

A Figura 2.7 mostra a correlação entre os campos de estudos relacionadas a otimização por evolução diferencial, observadas em junho de 2020 para o período de 2007 a 2020. A pesquisa foi realizada usando o *software* VosViewer e a base de dados da *Microsoft Academy*, usando a palavra-chave "*differential + evolution + optimization*" No total foram encontradas 8544 publicações, que foram filtradas em campos de estudo com ao menos 10 correlações entre si, totalizando 966 publicações. Cada nó representa o campo de estudo e quanto maior o nó, maior é o número de correlações entre as publicações sobre o mesmo campo de estudo. As cores representam o ano que a publicação foi publicada, onde cores mais escuras são publicações mais velhas e cores mais claras, publicações mais recentes. Verifica-se que a predominância das cores mais claras mostra que o algoritmo tem sido adotado em pesquisas recentes. Nota-se também que as áreas de estudo relacionadas a energia, como consumo de energia e energia solar contém publicações recentes.

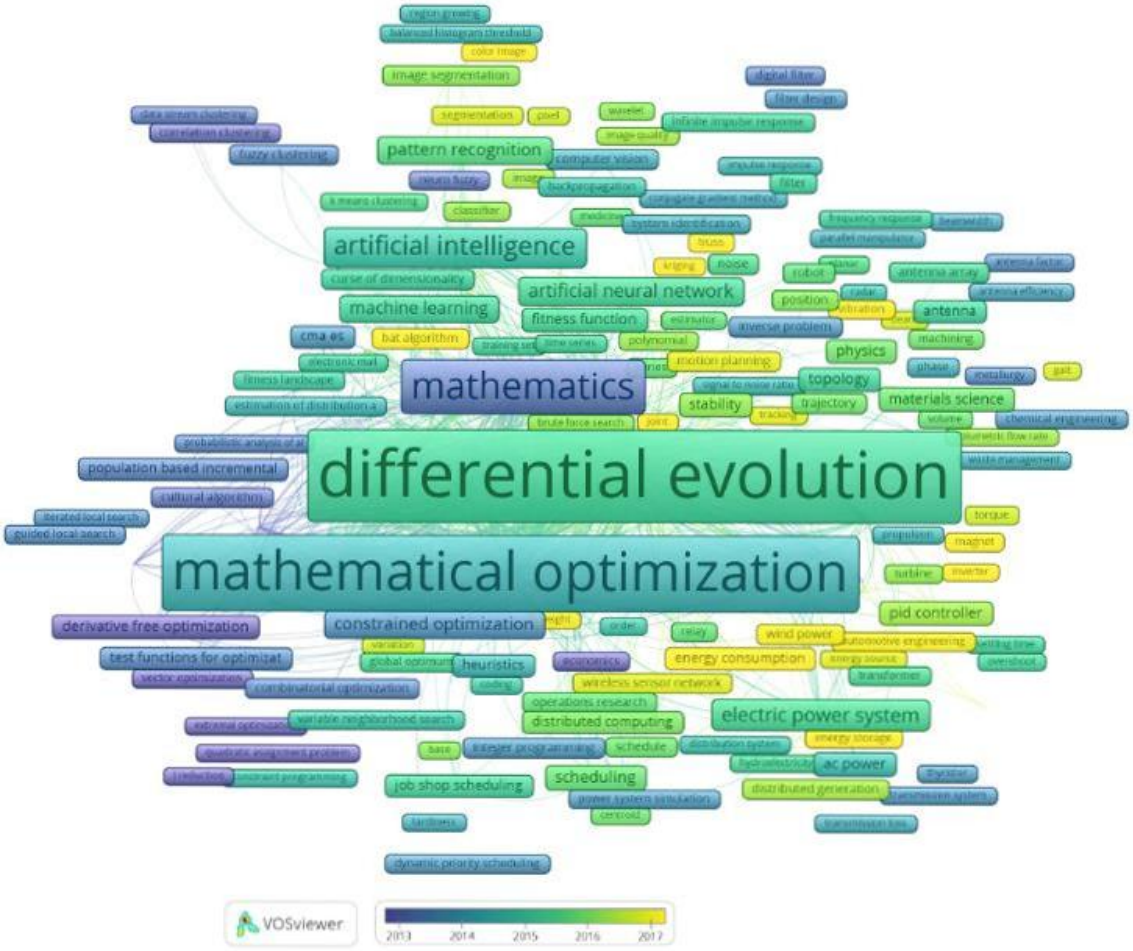

Figura 2.7: Grafo de correlação de publicações sobre evolução diferencial entre 2007 e 2020.

Reusando novamente as mesmas configurações, porém com a palavra-chave "*Bayesian* + *optimization*", obteve-se 5759 publicações, sendo que 874 campos de estudo tiveram mais de 10 publicações encontradas. Na Figura 2.8 é mostrado o resultado obtido. Neste caso, o uso de optimização de hiperparâmetros mostra como sendo uma área de estudo recente, uma vez que é representada na cor amarela.

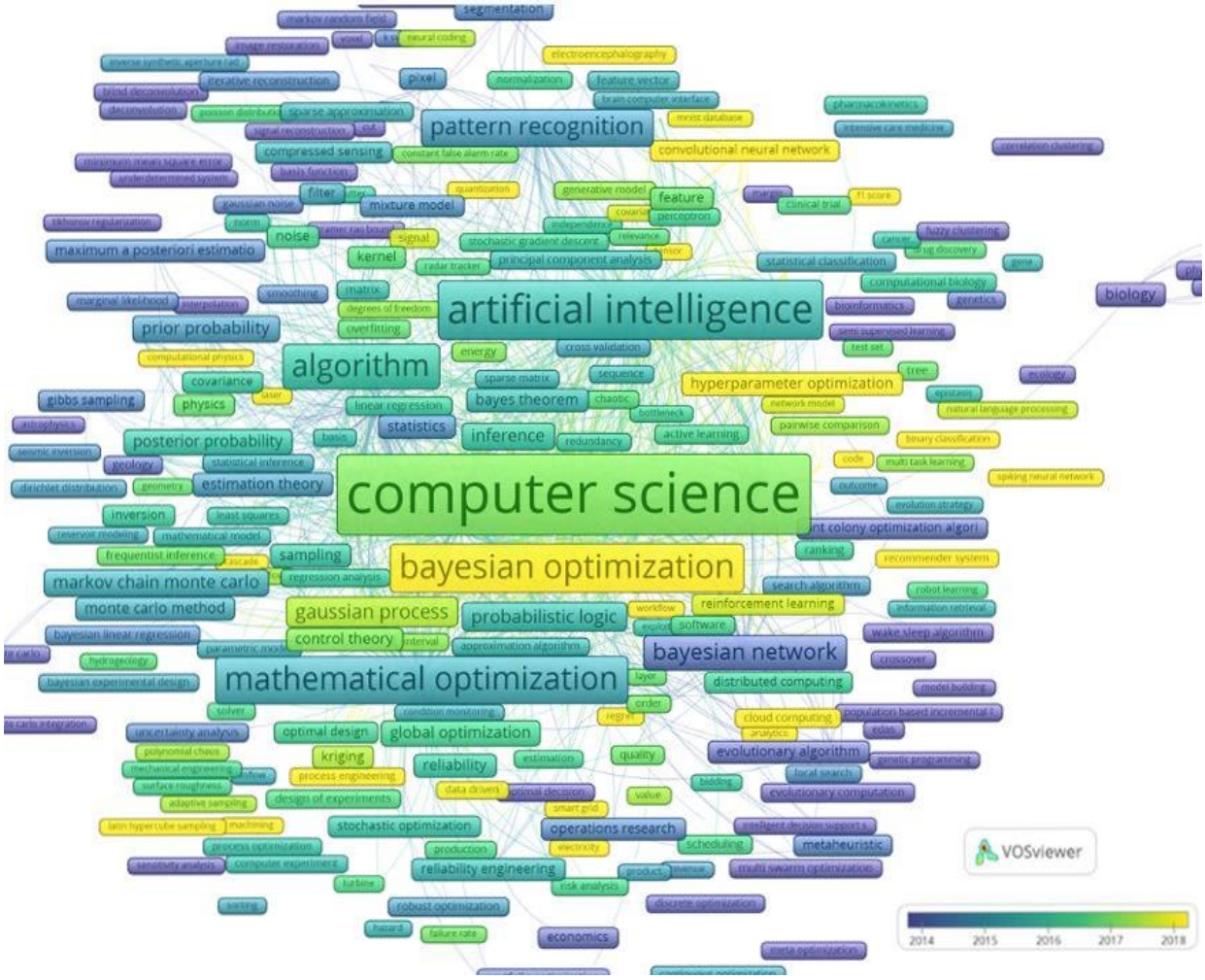

Figura 2.8: Grafo de correlação de publicações sobre otimização Bayesiana entre 2007 e 2020

Tem-se como um exemplo de tais publicações uma abordagem sugerida por Karaboga e Akay (2007) de um sistema híbrido usando uma meta-heurística de otimização denominada otimização por colônia de abelhas que foi usada para otimizar uma rede neural *feedforward*.

Por outro lado, o algoritmo de otimização por enxame de partículas, proposto por Kennedy e Eberhart (1995) tenta reproduzir o comportamento social e cooperativo bio-inspirado de populações em um espaço de pesquisa e foi integrado ao algoritmo de aprendizado profundo por Garro *et al.* (2009).

Wang *et al.* (2018) sugeriram um sistema híbrido entre otimização por enxame de partículas e um sistema de aprendizado adaptativo. Também sugerido tem-se um sistema híbrido de algoritmo de busca gravitacional com otimização por enxame de partículas e aprendizado profundo para otimização da função objetivo.

Por fim, salienta-se que na base de dados da *Microsoft Academic*, que é uma base de dados para referencia cruzada de publicações, ao se fazer uma busca pelo texto (sem o uso de parênteses) “DRVFL + *differential* + *evolution*” ou “DRVFL + *Bayesian*” na data de junho de 2021, não foi possível encontrar publicações referentes ao texto, sendo que diferentes combinações também foram testadas, juntamente com possíveis traduções do texto para o português

3 FUNDAMENTOS DE SÉRIES TEMPORAIS

Formalmente, uma série temporal pode ser definida como a realização de um processo estocástico, que é um processo aleatório que se desenvolve cronologicamente (Giusti, 2017). A definição de um processo estocástico de tempo discreto pode ser definida como um conjunto contável de variáveis aleatórias $X = \{X_1, X_2, \dots\}$, tal que $X_t : \Omega \rightarrow \mathbb{R}$ descreve os possíveis estados do processo ao longo do tempo (Giusti, 2017). Os fenômenos contínuos são representados como processos estocásticos de tempo discreto por meio de um procedimento denominado amostragem. O número de amostras tomadas durante o processo de amostragem é denominado taxa de amostragem da série temporal, que pode ser definida como f_s. Sua duração então pode ser definida como o produto da taxa de amostragem pelo número de observações amostradas $f_s \cdot n^1$. Por exemplo, seja $S = (S_1, S_2, \dots, S_n)$ uma série temporal amostrada a uma taxa de f_s, pode-se dizer que o tamanho, a dimensionalidade ou a cardinalidade de S é o número de observações $|S| = n$ e a sua duração é $n \cdot f_s$.

Como apresentado por Amaral (2020), os modelos temporais assumem que um evento futuro tem uma relação com eventos passados e por isso podem ser representados por meio de modelos criados a partir de dados passados. Um dos modelos tradicionais para expressar tais eventos seria o modelo de decomposição multiplicativa, que tem normalmente a sua decomposição expressada por:

$$Y_t = f(T_t, S_t, e_t) \qquad (3.1)$$

onde o resultado da série no tempo t é representado por Y_t, que tem os componentes de tendência, sazonalidade e residual(erro) no tempo t representados como T_t, S_t e e_t respectivamente.

Já a parte multiplicativa do modelo pode ser expressa como:

$$Y_t = T_t \times S_t \times e_t \qquad (3.2)$$

No caso do componente de tendência, é mostrado um indicativo de como as mudanças ocorreram no longo prazo, sendo que o componente de sazonalidade

representa um padrão de repetição ou um ciclo que possam existir na série. E a componente de erro representa o ruído aleatório que pode existir na série temporal.

O modelo de decomposição temporal é um dos modelos clássicos para previsão de série temporal que consiste em decompor o modelo que o descreve. O cálculo usado para tal decomposição pode ser dividido em etapas:

- Primeiramente, é usado uma média móvel MA para se poder definir os índices de sazionalidade. A média móvel é uma janela de valores de entrada deslocados no tempo. Com isso é possível suavizar os dados, removendo a aleatoriedade que possa existir no mesmo. A média móvel pode ser representada por:

$$MA(k) = \frac{1}{k}\sum_{t=1}^{k} Y_t \qquad (3.3)$$

  onde se tem a média móvel utilizando k amostras, que irá obter a média de m valores observados antes do valor atual até m valores após.

- Na etapa seguinte, calcula-se o fator de sazonalidade SF, que pode ser obtido por meio da relação entre os valores reais da série temporal no tempo t e o valor dessazonalidado resultante da média móvel centralizada, tal que:

$$SF_t = \frac{Y_t}{MA_t} \qquad (3.4)$$

  onde cada fator de sazonalidade calculado está associado a um ponto da série temporal. Através da média dos fatores de sazonalidade referentes ao mesmo tempo, remove-se qualquer componente residual, originando-se o índice sazonal S_t que irá possuir m resultados, referentes ao tempo da janela temporal utilizada na média móvel.

- Na próxima etapa, os dados dessazonalizados são calculados usando a relação entre o valor real da série temporal no tempo t e o índice sazonal S_t dado por:

$$d = \frac{Y_t}{S_t} \qquad (3.5)$$

- E, na etapa final, utiliza-se o método de mínimos quadrados para obter a regressão linear, tal que:

$$m = \frac{n\sum(xy) - \sum x \sum y}{n\sum(x^2) - (\sum x)^2} \quad (3.6)$$

$$b = \frac{\sum y - m\sum x}{n} \quad (3.7)$$

$$T_t = m \times t \times b \quad (3.8)$$

onde m é o coeficiente angular da reta, b é o coeficiente linear da reta, n é o número de amostras, x é a contagem da amostra, y é o valor da amostra no tempo x é T_té a previsão através da regressão no tempo t.

Ao final dessas etapas, resta apenas predizer os valores futuros usando a seguinte expressão:

$$previsão = T_t \times S_t \quad (3.9)$$

onde $previsão$ representa os valores preditos.

3.1 Características de uma Série Temporal

Uma série temporal pode abranger mais de uma característica, o que pode dificultar a escolha de um modelo mais apropriado para se prever o comportamento futuro da série. Neste contexto, cabe mencionar que as principais características podem ser obtidas tais como sazonalidade, tendência, observações aberrantes (anomalias ou *outliers*), heterocedasticidade e não-linearidade. A seguir são mencionados alguns fundamentos de tais características.

- **Sazonalidade**: A sazonalidade é a repetição de um padrão em um período fixo, conhecido também como ciclo. A sazonalidade em uma série temporal pode torná-la em um tipo não-estacionária. Levando-se em conta tal casos, alguns modelos consideram que a série tem componente sazonal e tem em suas configurações parâmetros que permitem adicionar o período de tal sazonalidade. Este seria o caso do modelo auto-regressivo integrado de médias móveis sazonal, (do inglês s*easonal auto-regressive integrated moving average*, SARIMA) proposto por Brockwell e Davis (1991).

- **Tendência**: Seria a tendência para onde a série temporal está se direcionando ao longo da sua trajetória. Ao se ter uma componente de tendência na série temporal, pode-se ter um modelo não-estacionário, que pode afetar a previsão por meio de um modelo. Neste caso, alguns métodos podem ser aplicados para se transformar a série em estacionária, tal como por exemplo o de uma diferenciação que consiste em subtrair os valores consecutivos da série $s_{1,}s_{2,}$ ... $s_{n,}$ tal que:

$$s_t^{nova} = s_t - s_{t-1} \quad (3.10)$$

onde *t* é o tempo, s_t é o componente de tendência, s_{t-1} é o componente de tendência anterior e s_t^{nova} é o novo valor estacionário no tempo *t*. Desta forma é gerada uma nova série temporal. Porém, a variância pode ainda sofrer incrementos no decorrer do tempo, que podem ser estabilizadas usando uma transformação Box-Cox (Box *et al.*, 1994);

- **Observações aberrantes**: Se presentes, as distorções observadas na série que estão fora dos padrões verificados anteriormente na série podem ser consideradas aberrantes ou mesmo atípicas. Tais observações podem ocorrer por amostras observadas não estavam sendo esperadas, conhecidos também como anomalias ou *outliers*, ou uma mudança de nível da série temporal na forma de presença de *drift*. Em casos de *outliers*, pode se tentar automaticamente detectá-los, usando por exemplo um modelo auto-regressivo de médias móveis (ARMA) Gaussiano, conforme sugerido por Ljung (1993);
- **Heterocedasticidade**: Quando uma série temporal não apresenta uma variância constante para todas os regimes da série temporal, pode-se então dizer que existe um comportamento heterocedástico. Tais casos podem ser abordados por meio de modelos auto-regressivos com heterocedasticidade condicional (Ker e Tolhurst, 2019).
- **Não-linearidade:** Qualquer série temporal que não apresente um comportamento linear entre a relação de dependência entre os valores da série temporal pode ser considerada como uma série não-linear. Um modelo clássico usado com certa frequência para tais tipos de séries é o clássico modelo ARMA.

3.2 Modelo Auto-Regressivo Integrado de Médias Móveis

ARIMA é um modelo generalizado de outro modelo chamado ARMA (do inglês a*uto regression moving average*) que foi proposto por Box e Jenkins (1976). O modelo original, ARMA, é um dos modelos mais comuns para modelagem de séries temporais. Tal modelo parte da premissa de que as variações formadas ao longo do tempo são randômicas, ou seja, um conjunto de variáveis aleatórias que existem ao longo do tempo *t*.

O modelo ARMA é então uma combinação de um modelo de auto-regressão (AR) com um modelo com valores médios de sua movimentação. Tais modelos então podem ser diretamente aplicados aos dados estacionários.

No caso de dados não-estacionários, é usado uma variação do modelo original, denominado ARIMA, aonde o (I) da sigla se refere a integração de sazonalidade, que é a ordem de diferenciação envolvida (Zhang *et al.*, 2020). Normalmente, o seu modelo pode ser expresso como ARIMA (*p, d, q*) e o seu modelo matemático pode ser então representado por:

$$\phi_p(B)\phi_P(B^S)\nabla^d\nabla_S^D Y_t = \theta_q(B)_Q(B^S)_{e_t} \quad (3.11)$$

onde as expressões da equação 3.11 podem ser definidas como:

$$\begin{cases} \phi_p(B) = \left(1 - \phi_p B - \phi_p B^2 - \cdots - \phi_p B^p\right) \\ \phi_P(B^s) = (1 - \phi_s B^s - \phi_{2s} B^{2s} - \cdots - \phi_{Ps} B^{Ps}) \\ \theta_q(B) = \left(1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q\right) \\ {}_Q(B^S) = \left(1 - {}_s(B^S) - {}_{2S}(B^{2S}) - \cdots - {}_{Qs}(B^{Qs})\right) \\ B^k Y_t = Y_{t-k} \\ \nabla^d = (1 - B)^d \\ \nabla_S^D = (1 - B^S)^D \end{cases} \quad (3.12)$$

onde $\phi_p(B)$ e $\theta_q(B)$ representam respectivamente os operadores não-sazonais AR e MA de ordem p e q. ∇^d é o operador de diferença que faz a série ser estacionária e d é o valor da diferença. Similarmente, $\phi_P(B)$, ${}_Q(B)$, P, Q e ∇_S^D são os símbolos dos elementos da sazonalidade e B é o operador de defasagem (do inglês *backshift*).

3.2.1 Auto-Regressivo Integrado de Médias Móveis Sazonal

O SARIMA (ou também chamado como S*easonal* ARIMA) é um modelo que tem como base o modelo ARIMA, mas, modificado para que seja usado em casos de dados com sazonalidade (Valipour, 2015).

Seu modelo pode ser representado por $SARIMA(p,d,q)\ x\ (P,D,Q)$, aonde p,d,q e P,D,Q são valores não-negativos que referem respectivamente a ordem polinomial das partes auto-regressivo (AR), integrado (I), e média móvel (MA) dos componentes não-sazonais e sazonais (Vagropoulos *et al.*, 2016). Seu modelo é descrito matematicamente como:

$$\varphi_p(B)\Phi_P(B^S)\nabla^d\nabla_S^D y_t = \theta_q(B)\Theta_Q(B^S)\varepsilon_t \qquad (3.13)$$

onde

y_t é a variável de previsão;

$\varphi_p(B)$ é polinômio normal AR de ordem p;

$\theta_q(B)$ é polinômio normal MA de ordem q;

$\Phi_P(B^S)$ é polinômio sazonal AR de ordem P;

$\Theta_Q(B^S)$ é polinômio sazonal MA de ordem Q.

O operador ∇^d elimina as diferenças não-estacionárias que são não-sazionais enquanto o operador ∇_S^D elimina as diferenças não-estacionárias sazionais. O operador defasagem, representado por B, opera observando y_t ao movê-lo um passo a frente no tempo, ou seja, $B^k(y_t) = y_{t-k}$). O termo ε_t segue um processo de ruído branco e s define of período sazional. Os polinomiais e seus operadores são definidos como:

$$\begin{cases} \varphi_p(B) = \sum_{i=1}^{p} \varphi_i B^i \\ \theta_q(B) = \sum_{i=1}^{q} \theta_i B^i \\ \nabla^d = (1 - B^S) \\ \Phi_P(B^S) = \sum_{i=1}^{P} \Phi_i B^{s,i} \\ \Phi_P(B^S) = \sum_{i=1}^{Q} \Theta_i B^{s,i} \\ \nabla_S^D = (1 - B^S)^D \end{cases} \quad (3.14)$$

3.3 Regressão por Vetores de Suporte

A regressão por vetores de suporte (do inglês *support vector regression*, SVR), é um método muito usado em modelagem de previsão séries temporais, principalmente por sua habilidade em construir modelos regressores não-lineares (Karmy e Maldonado, 2019).

Como mencionado por Quan *et al.* (2020), ao se resolver um problema não-linear, a função SVR mapeia o problema de regressão não-linear para o espaço de maior latitude, para que então possa um hiperplano otimizado para os pontos separados da função tal que

$$\max\left[-\frac{1}{2}\sum_{i=1}^{k}\sum_{i=j}^{k}(a_i - a_i^*)(a_j - a_j^*)K(X_i, X_j) - \sum_{i=1}^{k}(a_i - a_i^*)\varepsilon + \sum_{i=1}^{k}(a_i - a_i^*)Y_i\right] \quad (3.15)$$

sendo que

$$\begin{cases} \sum_{i=1}^{k}(a_i - a_i^*) = 0, \\ 0 \le a_i, a_i^* \le \frac{C}{l}, \\ i = 1, 2, \ldots, l \end{cases} \quad (3.16)$$

Tem-se ainda X_i que é os dados da amostra; l é o tamanho do exemplo; C é o coeficiente de erro; ε excede o tamanho da amostra de erro; $K(X_i, X_j)$ é a função *kernel*. Com isso, se tem-se como $a = \left[a_1, a_1^*, \ldots, a_l, a_j^*\right]^T$ e a função para a regressão SVR seria é dada por:

$$f(x) = \sum_{i=1}^{k} (a_i - a_i^*) K(X_i, X_j) + b^* \qquad (3.17)$$

desde que o resultado de $(a_i - a_i^*)$ não seja igual a zero, os valores correspondentes das amostras de X_i são os valores de suporte para o problema.

3.4 Regressão linear usando Mínimos Quadrados Ordinários

Os mínimos quadrados ordinários (do inglês o*rdinary least square*, OLS) como é usado para obter modelos de regressão linear. Conforme detalhado por Pohlman e Leitner (2003), o modelo de regressão linear baseia-se na relação entre as variáveis dependentes e a coleção de variáveis independentes. Os valores das variáveis dependentes são como uma combinação linear de variáveis independentes mais o termo de erro:

$$\boldsymbol{Y} = \beta_0 + \beta_1 \boldsymbol{X}_1 + \beta_2 \boldsymbol{X}_2 + \cdots + \beta_k \boldsymbol{X}_k + \varepsilon \qquad (3.18)$$

onde β_0 são os coeficientes de regressão, $\boldsymbol{X}_s$ são os vetores de colunas para os valores independentes e ε é vetor de predição de erros. O modelo é linear no parâmetro β, mas pode ser usado para ajustar as relações não-lineares entre $\boldsymbol{X}_s$ e $\boldsymbol{Y}$. Os coeficientes de regressão estão interpretados como as mudanças esperadas do valor de $\boldsymbol{Y}$ associado com um aumento unitário na variável independente, sendo que os valores das outras variáveis independentes continuem constante. Os erros são assumidos a ser uma distribuição normal com a expectativa de ser entre zero e a uma variável comum.

4 FUNDAMENTOS DE APRENDIZADO DE MÁQUINA

Aprendizado é uma importante atividade relacionada aos seres vivos, que não podem executar algo apropriadamente sem antes aprender como fazer tal tarefa (Groumpos, 2016). Transpondo-se a mesma ideia para o mundo virtual, pode-se então definir que o aprendizado de máquina é uma área da inteligência artificial relacionada ao desenvolvimento de técnicas e métodos capazes de dotar um sistema computacional da habilidade de aprender (Meng *et al.,* 2020).

Matematicamente, pode-se representar aprendizado de máquina pela expressão

$$f = x \rightarrow y \qquad (4.1)$$

onde x são os dados de entrada e y são os dados de saída. Ao se configurar problemas de séries temporais, onde tem-se um espaço contínuo, como um conjunto de números no domínio dos reais $\mathbb{R}$, tem-se um problema de regressão. Neste caso, pode-se imaginar que para os dados de entrada passados, espera-se uma previsão para os dados futuros. Pode-se ainda definir como dados de treinamento os valores que se previamente conhece os valores futuros e que podem ser usados para "ensinar" o sistema para o que se espera dele e, como dados de testes, os valores que não se conhece ainda e que gostaria que o sistema previsse.

Uma das definições clássicas para aprendizado de máquina foi proposta por Mitchell (1997): "Um programa de computador é considerado que aprendeu de uma experiência $\boldsymbol{E}$ em respeito a alguma tarefa $\boldsymbol{T}$ e medido pelo seu desempenho $\boldsymbol{P}$ seu desempenho para a tarefa $\boldsymbol{T}$**,** medido por $\boldsymbol{P}$**,** melhorou sua experiência $\boldsymbol{E}$."

O nome em inglês para aprendizado de máquina foi proposto em Samuel (1959) onde definiu-se o aprendizado de máquina como: *"campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados".*

Porém, no início da abordagem nos anos 1980, a maioria dos sistemas ditos "inteligentes" eram os sistemas especialistas. No entanto, na década de 1990, é possível notar que tais sistemas passam por uma revolução com a adoção de modelos estatísticos mais aprimorados e mineração dos dados.

Na década de 2000, começou-se novamente uma nova evolução no projeto de abordagens de aprendizado de máquina com a geração de computadores com placas de vídeo e processadores de alto desempenho, e com um custo para armazenamento de dados acentualmente menor. Isto permitiu a coleta e

armazenamento de maiores blocos de informações que poderiam ser usados por modelos de aprendizado de máquina durante a sua fase de treinamento, ajudando a popularizar o uso dela em diferentes áreas de estudo (Panesar, 2019).

Os modelos de aprendizado de máquina podem ser classificados conforme o aprendizado a ser realizado, sendo que isso pode ser determinado por fatores como os dados de entrada, uma descrição detalhada pode ser encontrada na seção 3.2 a seguir.

4.1 Abordagens de Aprendizado de Máquina

Baseado no tipo de dados de entrada, os algoritmos de aprendizado de máquina podem ser divididos em vários tipos: supervisionado, não supervisionado, semi supervisionado e por reforço. Uma descrição detalhada é apresentada nas subseções a seguir.

4.1.1 Aprendizado Supervisionado

O aprendizado supervisionado é uma das formas de aprendizado de máquina mais usadas (Lecun *et al.*, 2015). Nesse tipo de aprendizado, o algoritmo aprende a mapear os dados de entrada e saída de tal forma que para futuros casos, ele sabe determinar a saída prevista conforme a entrada passada.

Formalmente, os dados estão disponíveis em pares:

$$(x^{(i)}, y^{(i)}) \tag{4.2}$$

onde $x^{(i)} \in \mathbb{R}^n$ é um vetor de entrada; e $y^{(i)}$ é o seu correspondente de saída.

Quando as amostras $x^{(i)}$ de entradas tem igual dimensão, pode-se referir a eles como características, o conjunto de características podem ser expresso pelo vetor:

$$X = \left[x^{(0)}, x^{(1)}, \ldots, x^{(k)}\right]^T \tag{4.3}$$

e as amostras de saída têm representação dada por:

$$Y = \left[y^{(0)}, y^{(1)}, \ldots, y^{(k)}\right]^T \tag{4.4}$$

ou apenas um vetor y se os items de saída $y^{(0)}$ são unidimensionais.

Em um sistema supervisionado a meta é conseguir aproximar a função f de tal forma que se consiga obter:

$$f(x) = y \tag{4.5}$$

Com isso, pode-se corresponder que o aprendizado para se chegar a uma função seria uma distribuição de probabilidade *p* condicional do tipo: $p(y|x)$.

Tem-se então dois possíveis modelos para os tipos de tarefas:

- **Classificação de dados:** Quando as amostras de saída são uma parte discreta de $y^{(i)} \in \mathbb{Z}^m$.
- **Regressão:** Quando as amostras de saída são valores contínuos ($y^{(i)} \in \mathbb{R}^m$). Por exemplo, um sistema que tenta prever os valores futuros para uma série temporal, tal como a demanda de energia de uma determinada região.

4.1.2 Aprendizado Não Supervisionado

Neste tipo de aprendizado, nenhum rótulo é previamente informado, é apenas disponibilizado ao sistema os dados de entrada. O objetivo então torna-se em obter internamente as conexões existentes entre esses dados. Ao contrário do aprendizado supervisionado, o sistema não tem como saber quais são os possíveis valores de saída. Este tipo de aprendizado pode então ser comparado ao mais próximo do aprendizado humano. Na Figura 4.1 pode-se notar a diferença entre o aprendizado supervisionado e o não supervisionado. No aprendizado supervisionado cada um dos elementos está previamente rotulado. Porém no aprendizado não supervisionado, tem-se a segmentação dos dados conforme suas similaridades.

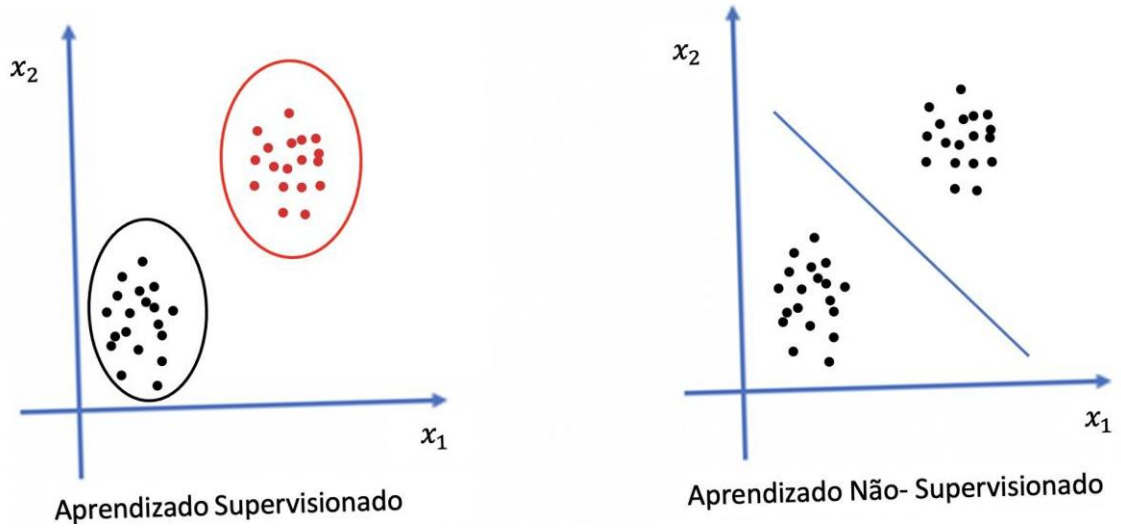

Figura 4.1: Diferença entre aprendizados supervisionado e não supervisionado

Conforme mencionado em Hinton *et al.* (1995) a base de dados normalmente é obtida no formato de $x^{(i)}$ e seu objetivo é modelar a probabilidade de distribuição de dados $p(x)$.

O tipo de tarefas em que o aprendizado não supervisionado pode ser usado são variados e alguns exemplos seriam:

- Análise de grupos (agrupamento de dados);
- Redução de dimensionalidade em mineração de dados;
- Projeto de modelos generativos, onde o sistema automaticamente aprende padrões que existem nos dados de entrada de uma maneira que podem ser usados para gerar valores de saída que parecem ser valores encontrados nos dados originais.

4.1.3 Aprendizado Semi Supervisionado

Neste tipo de aprendizado adota-se uma combinação entre os métodos supervisionados e não supervisionados, com o intuito de obter resultados melhores durante o aprendizado. Ou seja, ter um sistema que usa o aprendizado não supervisionado para extrair resultados de seu aprendizado para usá-los como forma de aprendizado em um sistema de aprendizado supervisionado.

Na área de séries temporais, é pode ser utilizado, por exemplo, um sistema de classificação de dados, que aprende a estrutura/complexidade “oculta” de uma série temporal que não tem seus dados rotulados (Fan *et al.,* 2021).

4.1.4 Aprendizado por Reforço

Neste tipo de aprendizado existe um agente que é capaz de aprender com as consequências de suas ações “de maneira autônoma”, o que significa que ao invés de ser explicitamente ensinado com dados de entradas e seus marcadores de saída, seu aprendizado é consequência de experiências passadas (Kaelbling *et al.*, 1996).

O sinal que o agente recebe é uma recompensa na forma de um número, que demonstra o sucesso da ação tomada. Com isso, o agente procura aprender a tomar ações que maximizem a recompensa com o passar do tempo. Na Figura 4.2 é ilustrado o ciclo do processo de aprendizado por reforço.

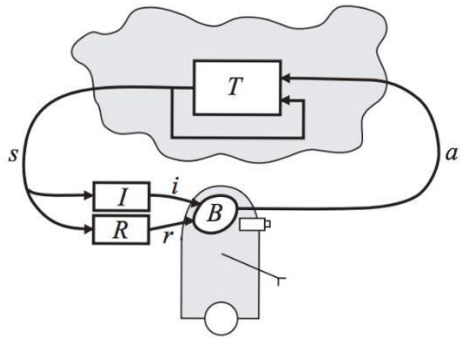

Figura 4.2: Modelo de aprendizado por reforço (Kaelbling *et al.*, 1996)

Em cada interação, o agente B recebe do interpretador I um conjunto de entradas i que identificam o estado atual s do ambiente T. O agente então escolhe uma ação a que será o sinal de saída. A ação modifica o estado do ambiente T e o valor do seu estado de transição é comunicado para o agente por meio do um sinal de reforço escalar r que foi previamente processado pelo sistema de recompensas R. O comportamento do agente, B, deve escolher ações que tendem a aumentar os valores das recompensas r ao longo de sua execução.

4.2 Redes Neurais Artificiais e Fundamentos do Perceptron Simples

O perceptron é uma abordagem de rede neural artificial proposta originalmente para classificação binária (Rosenblatt, 1957).

A concepção do perceptron foi inspirada em um modelo de neurônio “natural”, mostrado na Figura 4.3, que é uma célula biológica que processa informações. Ele é composto pelo corpo da célula e dois tipos de ramos, similares aos das árvores: o axônio e os dendritos. O corpo da célula contém um núcleo que contém informações

sobre a hereditariedade dos dados e o plasma que contém os equipamentos moleculares necessários para o neurônio.

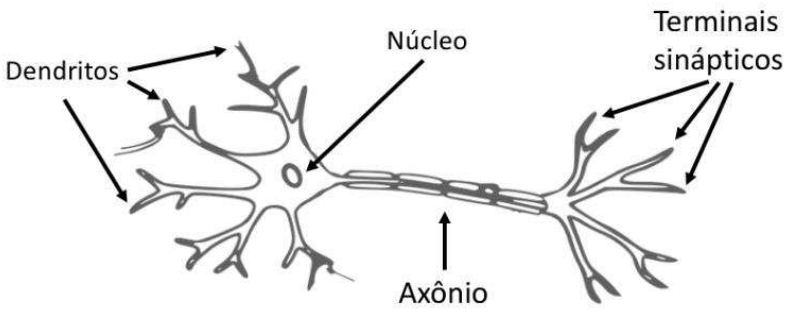

Figura 4.3: Uma representação de um neurônio (Openclipart-Vectors, 2017)

O modo com que um neurônio processa as informações é simples: um sinal, denominado também de impulso, é enviado por outros neurônios que é recebido pelos dendritos (*receivers*) e então gera sinais pelo seu corpo que é repassado pelo axônio (*transmitter*) para outros neurônios (Brunak e Lautrup, 1990).

Um outro modelo, este proposto por Mcculloch e Pitts (1943), usava uma unidade binária com limiar (em inglês *threshold*) como um modelo computacional para um neurônio artificial, conforme mostrado na Figura 4.4.

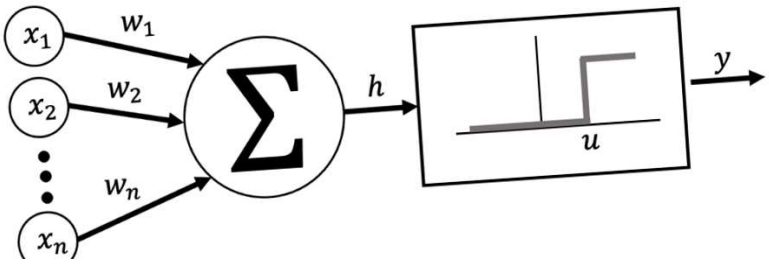

Figura 4.4: Representação de um neurônio artificial

O modelo de neurônio mencionado efetua o cálculo do peso *w* do somatório de n sinais de entrada, representados por um vetor x_j onde $j = 1, 2, \ldots, n$ e gera um somatório h, que se for maior que um determinado limiar, representado por u, gera uma saída y unitária ou zero. Tal modelo de neurônio faz uso do seguinte cálculo

$$y = \theta\left[\sum_{j=1}^{n} w_j x_j - u\right] \qquad (4.6)$$

onde $\theta[\,]$ é a função degrau e w_j é o peso da sinapse associada com seu sinal de entrada *j*. Para simplificar, comumente considera-se o limiar u como sendo um peso

do tipo: $w_0 = -u$ que está conectado diretamente com o neurônio com uma constante do tipo: $x_0 = 1$.

O modelo simples pode ser generalizado para outros tipos de neurônios. Por exemplo, pode-se mudar função ativação de um simples limiar para outros modelos, como por exemplo sigmoide linear ou Gaussiana, conforme mostradas na Figura 4.5.

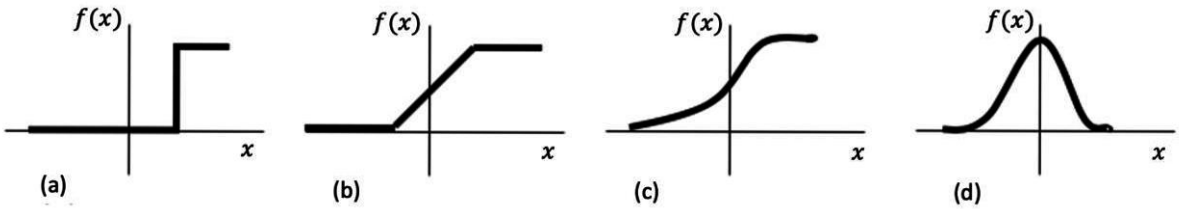

Figura 4.5: Tipos de função ativação: (a) limiar, (b) linear, (c) sigmoide e (d) Gaussiana

A função de ativação *sigmoide* é uma das mais usadas para redes neurais artificiais, pois suas funções que aumentam suavemente ao longo de um período criam um sistema mais restrito e com propriedades assimptóticas (Jain *et al.*, 1996).

Uma função *sigmoide* padrão seria a função logística, definida por:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \qquad (4.7)$$

onde e seria a base dos logaritmos naturais, x_0 o valor de x no ponto médio da curva sigmoide e L seria o valor máximo da curva e k seria a declividade da curva.

4.3 Perceptron Multicamada

Como pode-se notar pelo próprio nome, o perceptron multicamada (do inglês *multi-layer perceptron*, MLP) seria a composição do perceptron simples em várias camadas fazendo com que camadas escondidas sejam usadas. Com isso, em uma rede neural multicamada, têm-se ao menos 3 camadas, conforme é apresentado na Figura 4.6 onde se tem 3 sinais de entrada e 2 sinais de saída.

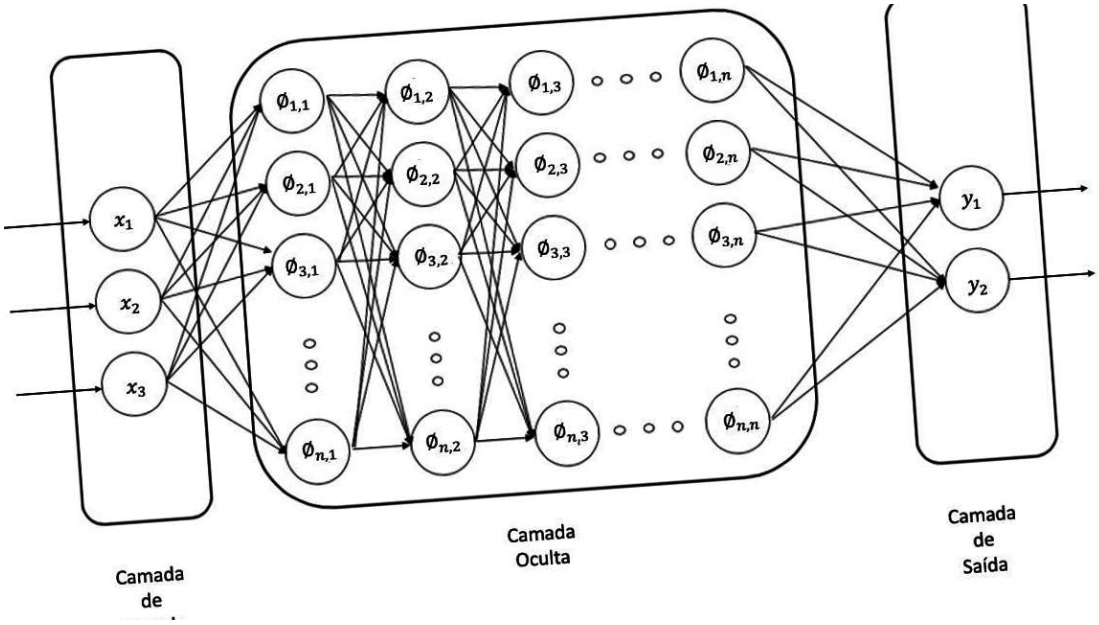

Figura 4.6: Modelo de rede neural multicamada

Neste tipo de rede, o aprendizado supervisionado é usado com uma técnica de correção dos valores dos pesos denominada retropropagação do erro para o treinamento da rede (Rosenblatt, 1961). O algoritmo consiste em dois passos:

- **Um passo à frente:** Onde as ativações são propagadas para frente pela adição de seus valores aos pesos calculados para seus valores de saída;
- **Um passo para trás**: Onde os erros entre os valores atuais e os valores obtidos na camada de saída são enviados para trás por meio de uma rede interligada entre as camadas e os seus pesos e objetivos são recalculados.

Neste tipo de perceptron é comum usar uma função ativação do tipo *sigmoide* $\sigma(w * x + b)$, definidos por:

$$\sigma(z) \equiv \frac{1}{1+e^{-z}} \text{ ou } \frac{1}{1+\left(-\sum_j w_j x_j - b\right)} \quad (4.8)$$

onde $w_1, w_2, \ldots, w_n$ são os pesos das conexões, expressando números reais e $x_1, x_2, \ldots, x_n$ são os valores de entrada passados ao perceptron e b é o *bias* que é usado como um limiar.

Com isso, o algoritmo mencionado tem-se uma forma de adaptar os pesos em cada camada, fazendo com que a função do sinal de erro seja minimizada conforme a saída.

O desempenho pode ser calculado ao se analisar o erro gerado na camada de saída do MLP. Com o algoritmo de treinamento por retropropagação do erro, espera-se que o sistema acabe melhorando durante o processo iterativo medido em épocas, conforme ele recebe respostas sobre o seu desempenho. Entretanto, existem outras técnicas que podem ser usadas para se melhorar os resultados (Lecun ***et al.***, 1998) tais como:

- **Parada antecipada:** Este parâmetro deve ser usado para evitar um sobreajuste. Depois de cada época de treinamento, a função custo é calculada para dados de validação que não foram usados durante a etapa de treinamento. Uma vez que os erros de validação começam a aumentar significantemente mesmo depois que os valores de erro para os dados de treinamento continuam caindo, o sistema acaba parando, pois está ocorrendo problemas devido a presença de sobre ajuste.
- **Diminuição do valor de peso:** Muda-se a função custo $L(x)$ que está sendo minimizada para que a mesma contenha termos adicionais que acabem penalizando os maiores pesos evitando assim um sobreajuste. Para isso, muda-se a função custo com a regularização do termo $L(x) + \lambda\|\theta\|$ onde λ é um hiperparâmetro de projeto. Normalmente, adota-se normas do tipo $L1$ ou $L2$.
- **Adaptação dos valores de aprendizado com erro de validação:** No começo da etapa de treinamento do MLP, começa-se com um valor para treinamento, normalmente com o maior valor possível e caso os valores das épocas sejam passados sem decrementar os valores de erro de validação, os valores de aprendizado são decrementados.
- **Pré-treinamento:** Conforme incrementa-se a quantidade de camadas na camada oculta, o custo computacional aumenta exponencialmente. Por isso, uma abordagem seria fazer um pré-treinamento, camada por camada, com um sistema não supervisionado e usar os valores obtidos como valores iniciais para o treinamento em si. Esta ideia foi baseada na distribuição da probabilidade de entrada $p(x)$ que pode fornecer

alguma informação sobre probabilidade condicional do marcador recebido pelos dados de entrada $p(t|x)$ (Bengio, 2009).

- **Desistência:** é uma técnica introduzida por Hinton (2012) para prevenir sobreajuste entre neurônios. A ideia básica é remover algumas das unidades na camada escondida (com probabilidade p) durante a propagação para frente, enviando 0 valores ao invés de um valor real de saída e então recompensar isso durante os testes dividindo os valores por p. Esta técnica pode ser entendida como modelos de treinamento múltiplos em diferentes porções de dados e prever a média dos modelos para reduzir a variação.

4.3.1 Aprendizado Profundo

Os algoritmos de aprendizado profundo são redes neurais com múltiplas camadas ocultas, que por sua vez podem trabalhar com sistemas supervisionados, não supervisionados ou semi supervisionados e sua principal característica é que a rede tende a aprender conforme os dados de entrada, sendo o oposto aos sistemas de tarefas específicas.

Conforme explicado em Biacnhini e Scarselli (2014), em arquiteturas com apenas uma camada escondida são denominadas como rede neural rasa (do inglês *shallow neural network,* SNN*)*, enquanto arquiteturas com múltiplas camadas escondidas, é denominada rede neural profunda (do inglês *deep neural network,* DNN*)*.

Várias pesquisas têm mencionado as potencialidades e limitações destas arquiteturas. Conforme mencionado em Mhaskar *et al.* (2017) tem-se:

- Em SNN, cada característica da camada escondida é única e, de certa forma, independente. Enquanto em DNN, as camadas, como usam dados de várias outras características das camadas anteriores, acabam sobrepondo-se entre elas. Isso acaba fazendo com que cada unidade seja mais e mais complexa conforme a profundidade da rede é aumentada.
- Em SNN, a sua função de ativação acaba sendo simples de ser implementada enquanto em DNN tem-se ainda várias camadas que podem afetar a função ativação das outras, deixando-a mais complexa.

- Em DNN, é possível de se implementar uma rede neural recorrente (do inglês *recurrent neural network,* RNN), fazendo com que as diferentes unidades das camadas possam reutilizar os dados ao longo do tempo.

4.3.2 Rede Neural *Feedforward*

O modelo rede neural *feedforward* (do inglês f*eedforward neural network*, FFNN) é um modelo que transforma os dados de entrada usando um conjunto de pesos e funções de ativação (Hastie e Friedman, 2009). Os pesos são primeiramente treinados usando um conjunto *T* de dados de entrada, representados por: $X = [x_1, \dots, x_t, \dots, x_T]$. Como mencionado em Ozanich, Gerstoft e Niu (2020), os dados de saída do FFNN é normalmente formado por uma distribuição, que a cada tempo $\hat{y}_t = [\hat{y}_t, \dots, \hat{y}_t]$:

$$\hat{y}_t = f\left(\sum_{i=1}^{D} w^i x^i\right) = f(w^T x_t) \qquad (4.9)$$

sendo $f = (\cdot)$ em uma função arbitraria e $\boldsymbol{w}$ o vetor de pesos. A otimização localizada de seus pesos $\boldsymbol{w}$ é estimada por meio da inversão de minimização da função custo, *J*, sobre as amostras de entrada $t = 1, \dots, T$:

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\arg min}\left\{-\sum_{t=1}^{T}\left(J\left(y_t, \hat{y}_t(\boldsymbol{w}, x_t)\right)\right)\right\} \qquad (4.10)$$

4.3.3 Redes de Conexões Aleatórias

A RVFL é um tipo de rede neural *feedforward* aleatória (do inglês *randomized feedforward neural network,* RFNN). A principal diferença entre eles é que existe uma conexão direta entre as camadas de entrada e as camadas de saída que permite uma melhora na generalização dos resultados (Li *et al.*, 2021). A Figura 4.7 mostra uma estrutura clássica de RVFL.

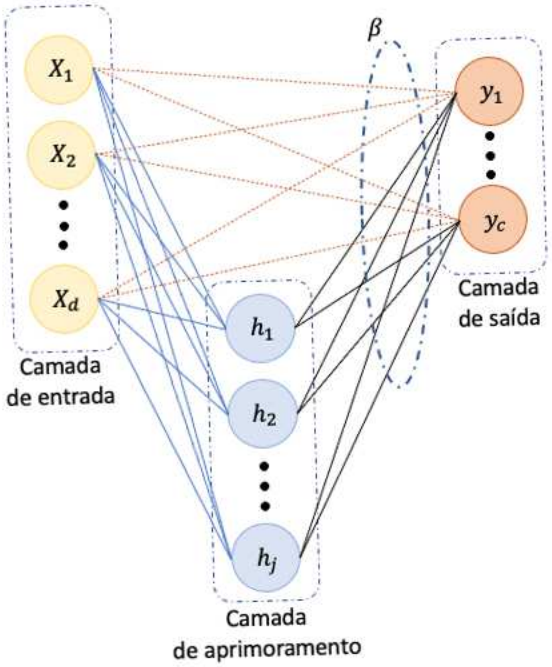

Figura 4.7: Fluxograma do RVFL (Li *et al.*, 2021)

Em RVFL, o aprimoramento dos pesos de entrada e suas polarizações são gerados aleatoriamente e mantidos durante todo o processo de aprendizado. Então a combinação da representação não-linear das características aprendidas na camada de aprimoramento e os dados originais são conjugados para determinar os pesos dos valores de saída. Então, considerando-se um banco de dados $\Xi = \{x_i\}_{i=1}^{N}$, onde $x_i \in \Re^d$ onde i seria a *i*-ésima amostra tem-se o RVFL com j nós aprimorados podem ser construídos tal que

$$f(x_i) = \sum_{j=1}^{J} \beta_j h_j(u_j, b_j, x_i) + \sum_{j=J+1}^{J+d} \beta_j x_{ij}, i = 1, \dots, N \qquad (4.11)$$

onde $\beta \in \Re^{(J+d)\times c}$ é o peso da camada de saída; u_j e $b_j$são, respectivamente, os valores aleatórios dos pesos e *bias* da camada de entrada, x_{ij} é a característica *j*-ésima da amostra x_i e $h_j(.,.,x_i)$ é a função ativação para o j-ésimo nó aprimorado da amostra x_i, que pode ser representado como

$$h_j(u_j, b_j, x_i) = h\left(\frac{\| x_i - u_j \|}{b_j}\right) u_j \in \Re^d, b_j \in \Re \qquad (4.12)$$

O último termo da equação 4.12 pode ser entendido como uma conexão direta entre a camada de entrada e a camada de saída. Consequentemente, para toda a base de dados $x_i, t_i{}_{i=1}^{N}$, a função custo pode então ser definida como

$$L = \sum_{i=1}^{N} \left\| \sum_{j=1}^{J} \beta_j h_j(u_j, b_j, x_i) + \sum_{j=J+1}^{J+d} \beta_j x_{ij} - y_i \right\|^2 \quad (4.13)$$

E simplificando-se a equação 4.13, que pode ser reescrita como uma matriz

$$\underset{\beta}{\operatorname{argmin}} \frac{1}{2} \|H\beta - Y\|^2 \quad (4.14)$$

onde

$$H = \begin{bmatrix} h(u_1, b_1, x_1) & \cdots & h(u_J, b_J, x_1) \\ \vdots & \ldots & \vdots \\ h(u_1, b_1, x_N) & \cdots & h(u_J, b_J, x_N) \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & \cdots & \vdots \\ x_{N1} & \cdots & x_{Nd} \end{bmatrix}_{N \times (J+d)} \quad (4.15)$$

então a solução fechada para a equação 4.14 pode ser obtida pela pseudo-inversa de Moore-Penrose como $\beta = H^{\dagger}Y$. Entretanto, tal solução pode estar propensa a erro de sobre ajuste. Então, para se prevenir tal problema e simplificar tal complexidade, é aplicado um algoritmo de mínimos quadrados para se resolver a equação 4.13 e então ter a solução calculada tal que

$$\beta^* = \begin{cases} \left(H^T H + \frac{I}{\lambda}\right)^{-1} H^T Y, N \geq J + d \\ H^T \left(H^T H + \frac{I}{\lambda}\right)^{-1} Y, N < J + d \end{cases} \quad (4.16)$$

para uma amostra de teste, seu rótulo previsto pode ser determinado como

$$j = arg\max\{[h(u, b, x), x]\beta^*\} \quad (4.17)$$

4.3.3.1 Redes Profundas de Conexões Aleatórias

As redes profundas de conexões aleatórias é uma extensão aplicada ao RVFL que aplica uma rede neural profunda, caracterizada por uso de camadas escondidas, onde a entrada de uma camada é formada pela saída da camada predecessora. Em cada uma das camadas escondidas, uma representação interna dos dados de entrada irá ser passado (Del Ser *et al.*, 2021). Este tipo de estrutura apresenta uma flexibilidade em seu projeto, permitindo um tamanho da rede e os números de neurônios de cada camada serem ajustados conforme a necessidade.

Considerando-se um número de camadas ocultas K, um mesmo número de neurônios $\widetilde{N}$ e os dados de treinamento $\{(x_i, y_i)\}_{i=1}^{n}$, pode-se então dizer que os passos executados por DRVFL são:

1. Determina-se um valor de forma aleatória para os pesos de entrada $\boldsymbol{W}^{(1)}, \dots, \boldsymbol{W}^{(K)}$ e seus vieses $\boldsymbol{B}^{(1)}, \dots, \boldsymbol{B}^{(K)}$, que representam a conexão entre a camada de entrada e a primeira camada oculta. Esses valores irão ser mantidos durante toda a fase de treinamento;
2. Calcula-se a matriz de saída $\boldsymbol{H}^{(1)}$ da primeira camada oculta, que consiste em uma transformação não linear das características de entrada, como: $\boldsymbol{H}^{(1)} = g\left(\boldsymbol{X}\boldsymbol{W}^{(1)}\right)$, onde $\boldsymbol{X}$ é a matriz de entrada dos dados de treinamento e $g(.)$ é a função ativação dos neurônios;
3. Enquanto o número de camadas ocultas for maior que 1, os dados de saída das camadas ocultas podem ser obtidos seguindo-se: $\boldsymbol{H}^{(K)} = g\left(\boldsymbol{H}^{(K-1)}\boldsymbol{W}^{(K)}\right)$;
4. Na camada de saída, os dados de entrada seriam representados por:

$$\boldsymbol{I} = \left[\boldsymbol{H}^{(1)}; \boldsymbol{H}^{(2)}; \dots; \boldsymbol{H}^{(K-1)}; \boldsymbol{H}^{(K)}; \boldsymbol{X}\right] \quad (4.18)$$

5. Somente os pesos de saída (β) são obtidos durante o processo de treinamento;
6. Finalmente, os valores de saída para o algoritmo são definidos como:

$$\hat{Y}(x) = \boldsymbol{I}\boldsymbol{\beta} \quad (4.19)$$

4.3.4 Aprendizado Profundo Extremo

A literatura traz que a velocidade de aprendizado de redes neurais *feedforward* é geralmente lenta, fazendo-se com que seja um dos maiores gargalos para se usar tais métodos em aplicações com resposta em tempo real (Ghosh *et al.,* 2019). São dois os principais motivos para isso:

- Os algoritmos de aprendizado baseados em gradiente são extensivamente usados no treinamento das redes neurais artificiais;
- Todos os parâmetros da rede são interativamente ajustados usando tais algoritmos.

Diferentemente dessas implementações convencionais, o aprendizado profundo extremo, sugere uma nova estratégia (Huang *et al.*, 2006). Tal estratégia é aplicada nas redes neurais *feedforward* com uma única ou múltiplas camadas ocultas, onde é escolhido aleatoriamente os nós ocultos e analiticamente determina-se os pesos a serem aplicados na saída.

Sua vantagem, como é esperado, é que resultados satisfatórios sejam obtidos em velocidade rápida para seu aprendizado pois não tem um processo iterativo de aprendizado.

Dado uma única camada escondida onde a função de saída seja representada por:

$$h_i(x) = G(a_i, b_i, x) \tag{4.20}$$

onde a_i e b_i são os parâmetros da *i*-ésima camada oculta. Neste contexto, para a função de saída com múltiplas camadas ocultas tem-se:

$$f_L(x) = \sum_{i=1}^{L} \beta_i h_i(x) \tag{4.21}$$

onde β_i é o peso da saída da *i*-ésima camada oculta.

4.3.5 Rede Neural com Estado de Eco

A rede neural com estado de eco (do inglês *echo state network*, ESN), por ser uma rede recorrente tem potencialidades para previsões de séries temporais. Suas camadas podem ser divididas em três partes: camada de entrada; seu reservatório dinâmico, que corresponde a uma camada interna que consiste em vários neurônios conectados esparsamente e que contribuem para o processamento da informação de maneira mais rápida; e a camada de saída (Jager, 2001).

Um dos motivos para que a ESN tenha uma convergência rápida se dá pelo motivo que apenas os pesos de saída usam algoritmos de regressão linear durante a fase de treinamento. A Figura 4.8 mostra uma arquitetura ESN e as suas três camadas, contendo K unidades de entrada para ativar a rede, N neurônios em seu reservatório dinâmico que forma seu estado interno e L neurônios de saída que produzem o sinal de saída.

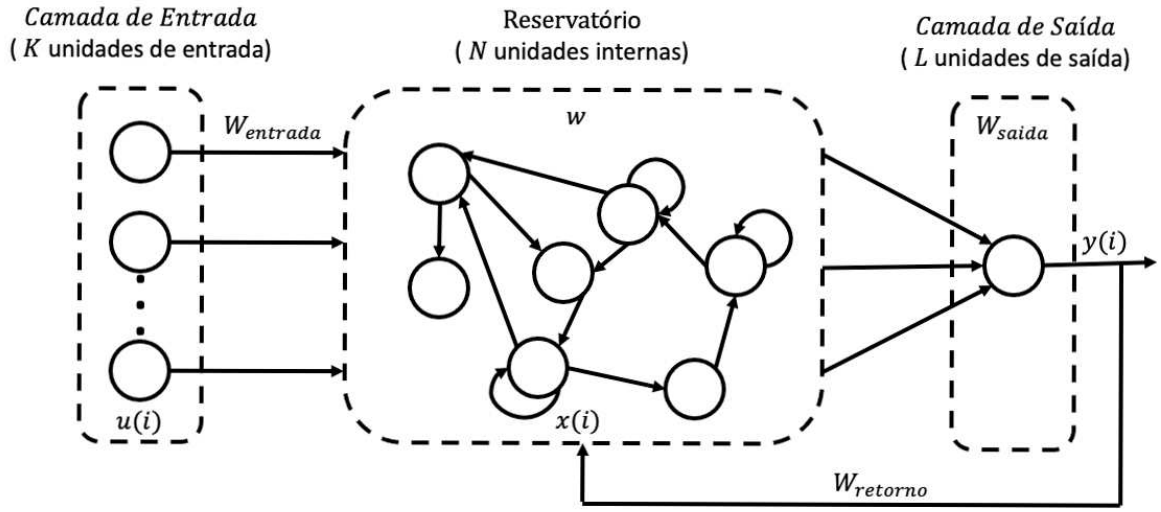

Figura 4.8: Arquitetura de uma rede ESN (Hu *et al.*, 2020)

Supondo-se que em um determinado tempo i, os dados da matrix de entrada podem ser definidos como:

$$u(i) = [u_1(i), u_2(i), \dots, u_K(i)]^T \quad (4.22)$$

A matriz dos dados do reservatório dinâmico então pode ser definida como

(4.23)

$$x(i) = [x_1(i), x_2(i), \dots, x_N(i)]^T$$

e seus dados de saída seriam

$$y(i) = [y_1(i), y_2(i), \dots, y_L(i)]^T \quad (4.24)$$

onde pode-se definir $i = 1, 2, \dots, I$ sendo que o total do tempo de treinamento seria representado por I.

Para a atualização dos dados nos estados internos dos reservatórios representados como $x(i+1)$ e para os estados de saída, representados como $y(i+1)$, tal que:

$$\begin{cases} x(i+1) = f\big(w_{in} * u(i+1) + w * x(i) + w_{back} * y(i)\big) \\ y(i+1) = g\big(w_{out} * x(i+1)\big) \end{cases} \quad (4.25)$$

onde as funções de ativação para os neurônios no reservatório e $g = [g_1, g_2, \dots, g_N]$ para os neurônios de saída podem ou não serem lineares. Adicionalmente, pode-se notar que existe matrizes de pesos, sendo w_{in} de $N * K$ que conecta os estados de entrada com os estados internos, o peso w de $N * N$ que habilita as conexões internas dos estados internos do reservatório, o peso w_{back} de $N * L$ que conecta os estados de saída com os estados internos do reservatório, e finalmente o peso w_{out} de $L * N$ que conecta os estados do reservatório com os estados de saída.

4.4 Rede Neural Convolucional

As redes neurais convolucionais (em inglês *convolutional neural network*, CNNs ou ConvNets) são focadas principalmente em aplicações de processamento de imagens, pois foi inspirada na parte do cérebro humano relacionado a imagens, denominado córtex visual primário (do inglês *primary visual cortex,* PVC) (LeCun *et al.*, 1998). Em uma explicação simples, o PVC é alimentado por sinais recebidos por neurônios que estão conectados com a retina do olho e que são estimulados pela luz. Esses neurônios, por sua vez, realizam um processamento básico da imagem recebida, porém os mesmos não alteram a sua representação. A imagem é então enviada a nervo óptico, que por sua vez envia para o PVC.

Os neurônios corticais individuais respondem a estímulos apenas em regiões restritas do campo de visão conhecidas como campos receptivos. Os campos receptivos de diferentes neurônios se sobrepõem parcialmente de forma a cobrir o campo de visão.

A convolução é uma função com dois argumentos: a entrada e o núcleo. A saída é normalmente referenciada como *feature map*. A entrada é geralmente um vetor multidimensional de valores e o núcleo é um vetor multidimensional de parâmetros de ajuste para o algoritmo de aprendizado. Estes parâmetros normalmente são denominados de tensores. A equação 4.26 representa a convolução discreta de um *kernel* K bidimensional, também chamado de filtros, fazendo com que o sistema consiga lidar com variações de distorção, rotação e translação da imagem tal que

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \qquad (4.26)$$

onde I representa a imagem a ser analisada, sendo representada por seus pixels em posições i e j.

Existem quatro operações principais que normalmente são consideradas críticas para um típico exemplo de CNN: convolução, não linearidade, *pooling* (ou subamostras) e classificação. Essas operações são obtidas usando três camadas fundamentais na arquitetura da rede: camada convolucional, camada de *pooling* e camada totalmente conectada. Por exemplo, LeNet5 (Lecun *et al.*, 1998) é um exemplo clássico de uma arquitetura que pode ser usado para se entender como essas camadas são usadas. Na Figura 4.9 é ilustrada a estrutura de uma rede convolucional 1D aplicada para a previsão de uma série temporal.

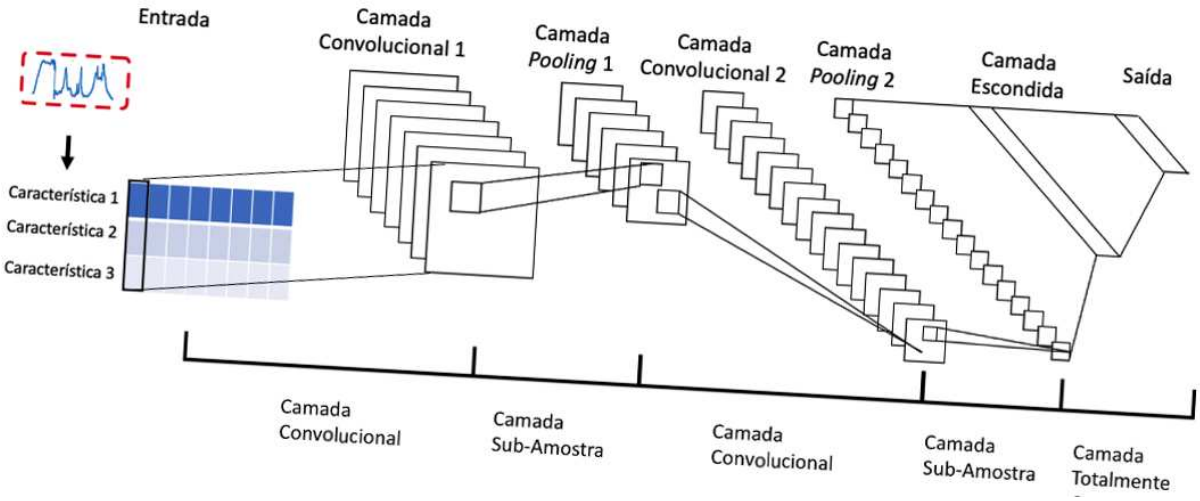

Figura 4.9: Arquitetura uma rede convolucional 1D para previsão de uma série temporal

Na camada convolucional ocorre a extração das características da imagem, denominada *feature maps* ou matriz de ativação. Nesta etapa, os dados espaciais da imagem são preservados nos *pixels*, usando filtros. Filtros ou o núcleo em CNNs são como uma parte ou uma matriz que desliza sobre a imagem. Durante esse processo, os filtros calculam o produto dos correspondentes pixels da imagem e produzem o *feature maps*. Os filtros agem como detectores da imagem de entrada e diferentes valores da matriz de entrada irão produzir diferentes mapas de características.

Na camada de *pooling* (ou camada de subamostras) os mapas de características têm seus tamanhos reduzidos. Nesta etapa, nenhum processo de aprendizado é realizado. O princípio é que a entrada será dividida em múltiplos elementos retangulares que se sobrepõe e unidades entre cada elemento são usadas para criar uma única unidade de saída. Isso acaba diminuindo o tamanho da camada de saída e ao mesmo tempo preservando as informações mais importantes da imagem de entrada.

Uma potencialidade desta camada é a de introduzir a invariância translacional. Por exemplo esta potencialidade é a ativação de funcionalidades em diferentes posições no espaço de entrada que podem ter o mesmo resultado se aumentar a generalização. Isso faz com que o *pooling* máximo seja superior aos outros tipos de operações de *pooling* (Scherer *et al.*, 2010).

Finalmente, a última camada denominada camada totalmente conectada nada mais é do que uma tradicional rede neural multicamada com uma função ativação do tipo *SoftMax*. O nome originalmente veio do fato de que cada neurônio nesta camada está conectado com todos os outros neurônios da camada anterior. Esta camada na

maioria dos casos é a responsável por fazer classificação de dados. Uma vez que os dados de saída das outras camadas contêm informações de alto nível, obtidos da imagem de entrada, nesta última camada tais informações são usadas para aprimorar os dados especificados pelos dados de treinamento.

4.5 Rede Neural Recorrente

Em CNNs não existe uma maneira de fazer com que a rede se lembre sobre os dados do passado. Porém, com a introdução das redes neurais recorrentes, proposto por Rumelhart *et al.* (1986), foi possível solucionar este problema. Uma RNN é uma classe de redes neurais que inclui conexões ponderadas dentro de uma camada, sendo diferentes das redes neurais mais tradicionais onde se conecta a alimentação apenas nas camadas mais profundas. Com isso, tais redes conseguem armazenar informações ao processar novas entradas, fazendo com que as entradas anteriores também sejam consideradas.

Na Figura 4.10 é mostrada uma representação de uma rede neural recorrente.

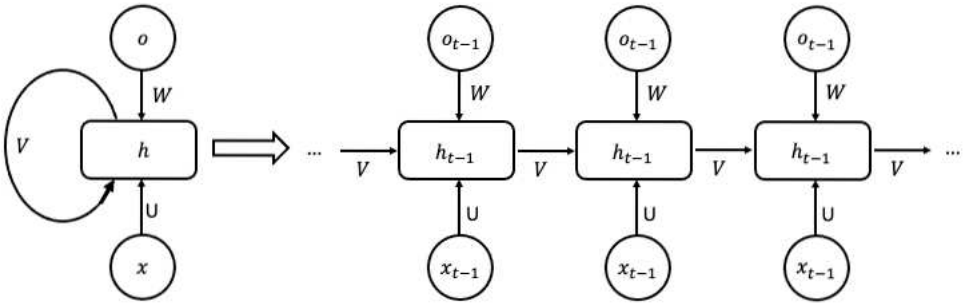

Figura 4.10: Representação gráfica de uma rede neural recorrente

Na parte mais à esquerda do desenho, tem-se um modelo compacto. Tem-se o vetor de entrada x que está conectado a camada oculta h e por meio da conexão U que representa os parâmetros de entrada da rede. A camada oculta então gera um sinal de saída o que está conectada por uma conexão W que contém os parâmetros da rede para saída. E as camadas estão conectadas entre si por meio da conexão V. Ao se expandir tal modelo, como representado no desenho mais a direita, tem-se a rede representada ao longo do tempo t. Em cada instante t, tem-se então que a entrada corrente é a combinação de x_t mais a saída que ocorreu em x_{t-1}.

4.5.1 Memória Longa a Curto Prazo

A memória longa a curto prazo (do inglês *long short-term memory*, LSTM) é uma rede neural recorrente que é baseada em modelos FFNN, porém com a capacidade de se lembrar de padrões que ocorreram ao longo do tempo, uma vez que a sua estrutura permite receber dados anteriores de suas conexões (Nguyen *et al.*, 2021). Por isso, consegue-se melhorar o problema de desaparecimento de gradiente, que é um problema comum para outros modelos de redes neurais recorrentes. Sua arquitetura, mostrada na Figura 4.11 mostra que diferentemente de modelos RNN, o LSTM usa múltiplas entradas que agem como células de memória que gerenciam o fluxo de entrada e saída dentro da rede.

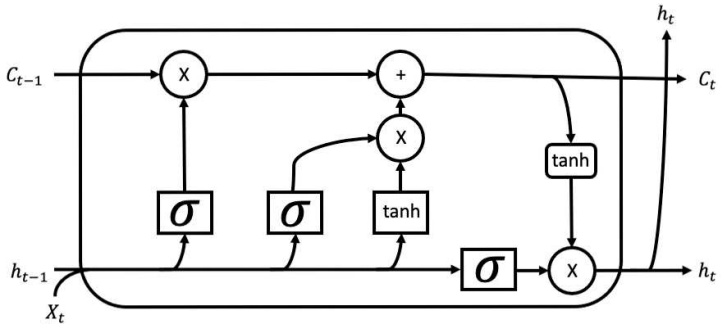

Figura 4.11: Fluxograma do LSTM (Yu et al., 2021)

Em um determinado tempo t, o elemento de entrada corrente, representado como x_t, e os resultados do passo anterior h_{t-1} produzem os dados de entrada na camada escondida (oculta) corrente. Consequentemente, a rede pode mapear a sequência de entrada x_t em uma sequência de saída $\mathrm{y_t}$, sendo que cada y_t, depende de todos os valores anteriores de x_t, acordo com a seguinte relação:

$$\begin{cases} h_t = \Re(\mathrm{W}_h h_{t-1} + U_i x_t + b_h) \\ y_t = \Re(\mathrm{U_o h_t} + b_o) \end{cases} \quad (4.27)$$

sendo $\Re$ a representação para função ativação, b_h e b_o são *bias* aplicados e U_i, W_h e U_o são os pesos das camadas.

Uma das principais diferenças entre redes LSTM e outras redes RNN é que o LSTM usa portas (ou *gates*) ao invés de neurônios de camadas recorrentes (Bukhari *et al.*, 2020). Tais portas tem uma conexão direta com o próximo passo e uma unidade de tomada de decisões que controla a limpeza da memória. Tais portas conseguem então controlar o fluxo de informação dentro da rede. Os tipos de portas existentes

são: *forget*, entrada e saída. As portas de *forget,* como o nome sugere, deve decidir se a informação ainda útil e mantê-la ou removê-la da rede. Tais decisões tem duas possíveis respostas: **zero**, que significa remover tais dados da célula, e **um**, que mantém os valores originais da célula. A porta de *forget* pode então ser representada por*:*

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t\,] + b_f\right) \quad (4.28)$$

onde σ representa a função ativação que frequentemente usa uma função sigmoide representado por:

$$\sigma(X) = \frac{1}{1 - e^{-x}} \quad (4.29)$$

Na próxima etapa, a porta de entrada e o resultado da sigmoide são agrupados para se criar um estado C_t que move para o próximo passo que renova os estados das outras células. As equações que representam tal agrupamento são representadas por:

$$\begin{cases} i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_t) \\ C_t' = \tanh(W_c[h_{t-1}, x_t] + b_c) \end{cases} \quad (4.30)$$

onde a função *tanh* é uma função tangente hiperbólica que pode variar entre -1 e 1 tal que

$$\tanh(X) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (4.31)$$

4.6 Floresta Aleatória

A floresta aleatória *(do inglês, random forest,* RF) consegue prover uma combinação única entre precisão de suas predições e a interoperabilidade do modelo ao longo dos métodos mais populares para aprendizado de máquina. As amostras aleatórias e as estratégias de *ensemble* utilizadas em suas árvores habilitam alcançar predições precisas e melhores generalizações. Sua propriedade de generalização

vem da implementação de um esquema chamado “ensacamento” (do inglês *bagging*), que aprimora a generalização por meio da diminuição de sua variância, enquanto outros métodos, tal como *boosting*, realizam tal efeito por meio da diminuição do *bias* (Qi, 2012).

RFs são um conjunto de árvores que usam n observações idependentes, que podem ser representadas como:

$$(Y_i, X_i), \qquad i = 1, \dots n \tag{4.32}$$

Em RF, cada ramo é associado com a função de divisão $f(\boldsymbol{x}; \theta)$ que pode ser representado por:

$$f(\boldsymbol{x}; \theta) = \begin{cases} 1 \text{ se } x(\theta_1) < \theta_2 \\ 0 \text{ outros casos} \end{cases} \tag{4.33}$$

onde $\theta_1 \in \{1, 2, \dots, d\}$ é uma característica selecionada e $\theta_2 \in \mathbb{R}$ é um limite. O resultado determina para qual ramo filho os valores de $\boldsymbol{x}$ são redirecionados. Por exemplo, o valor 0 pode representar que seja o ramo filho a esquerda, enquanto o valor 1 representaria que seria o ramo filho a direita. Os ramos, da árvore são armazenados usando-se uma distribuição probabilística ou rótulos baseados nas amostras recebidas durante o treinamento.

Durante os testes, para uma amostra $\boldsymbol{x}$, cada árvore retorna uma distribuição probabilística $p_t(y|\boldsymbol{x})$ armazenada no ramo no qual o valor acaba passando. Finalmente, o rótulo é obtido por meio de uma votação no qual o maior valor vence ou por uma mediana que pode ser representada por:

$$y^*(\boldsymbol{x}) = \underset{y}{\arg max} \frac{1}{T} \sum_{t=1}^{T} p_t(y|\boldsymbol{x}) \tag{4.34}$$

4.6.1 Floresta Aleatória Quantílica

A floresta aleatória quantílica (do inglês, *quantile random forest,* QRF) é um melhoramento do método original de floresta aleatória combinado com regressão baseada em quartis, no qual resulta em uma distribuição condicional e não apenas

um valor singular retornado pela média de suas árvores (He *et al.*, 2020). A distribuição condicional de Y pode ser expressa por:

$$F(y|X = x) = P(Y \leq y|X = x) = E\left(l_{\{Y_i \leq y\}}|X = x\right) \quad (4.35)$$

sendo que $E\left(l_{\{Y_i \leq y\}}|X = x\right)$, quando usando-se RF, é definida como a média dos valores de observação. Sendo que para QRF, seria a mediana dos pesos de suas observações, como expressado matematicamente aqui:

$$\hat{F}(y|X = x) = \sum_{i=1}^{n} w_i(x) l_{\{Y_i \leq y\}} \quad (4.36)$$

sendo que a sequência de acontecimentos pode ser resumida pelos seguintes aspectos:

- As K árvores de decisão $T(\theta_t), t = 1, \dots, k$ são geradas usando-se RF e as observações de cada ramo em cada árvore de decisão é considerada e armazenada;
- Para $X = x$, interaja-se com todas as árvores de decisão e calcula-se os pesos de cada uma das árvores de decisões. O peso $w_i(x, \theta_t)$ de cada observação $i \in \{1, \dots, n\}$ então é obtida pela mediana dos pesos das árvores de decisão;
- Para todos os $y \in \mathbb{R}$, calcula-se a estimativa da função distribuição com a função representada pela equação 4.36 usando-se os pesos obtidos no passo anterior.

4.6.2 Gradiente Extremo Aumentado

Gradiente extremo aumentado (*do inglês extreme gradient boosting*, XGBoost), é um modelo melhorado da árvore de decisão que usa como base o *gradient boosting* (Friedman, 2002). Uma de suas principais características é a construção de árvores de decisão mais eficientes (o algoritmo é otimizado para os valores da função custo) e que rodem em paralelo.

O XGBoost constrói suas árvores de decisão para usar logicamente os valores obtidos nos seus resultados da função custo, indicando então a importância de cada uma das características para o modelo de treinamento (Kiangala *et al.*, 2021).

O algoritmo usa três fatores de importância: **ganho**, **frequência** e **uso** (Friedman, Hastie e Tibshirani, 2001). O **ganho** é o principal fator de importância de uma característica dos ramos da árvore. **Frequência**, que seria uma versão simplificada do ganho, é o número de características construídas em todas as árvores de decisão. E por fim, o **uso** seria os valores relativos da característica observada.

Na Figura 4.12, é possível notar o funcionamento do XGBoost, aonde a cada iteração do *gradient boosting*, os resíduos da predição atual são usados na próxima predição para que a função de perda seja otimizada por meio da correção do preditor.

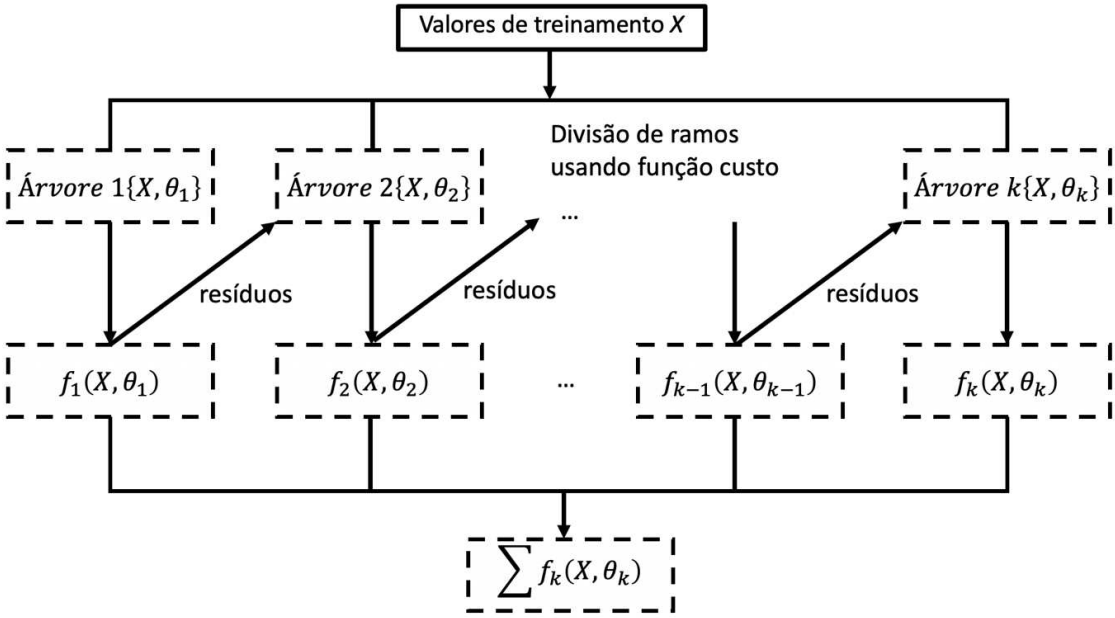

Figura 4.12: Fluxograma do XGBoost (Zhang *et al.*, 2018)

Zhang *et al.* (2018) sugeriu como parte das melhorias no XGBoost, uma regularização que foi adicionada para a função de perda para estabelecer a função objetivo do modelo, que poder ser representado por:

$$J(\theta) = L(\theta) + \Omega(\theta) \qquad (4.37)$$

onde θ representa os valores treinados para os dados de entrada; L é a função de perda do treinamento, como por exemplo a perda quadrática ou perda logística, que mede quão bem o modelo respondeu aos dados de entrada. O parâmetro Ω é o valor

de regularização, como a norma $\mathrm{L1}$ ou $\mathrm{L2}$, que mede a complexidade do modelo. Modelos simples, tendem a ter melhor desempenho contra o sobre ajuste. Sendo que a base do modelo é uma árvore de decisões, a saída do modelo $\hat{y}_i$ é votada ou a mediana de uma coleção de F de k árvores tais que

$$\hat{y}_i = \sum_{i=1}^{k} f_k(x_i), f_k \in F \quad (4.38)$$

onde a função objetivo (custo) para a iteração acontecendo no tempo t pode ser representada como:

$$J^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{t} \Omega(f_k) \quad (4.39)$$

aonde n representa o número de predições. Para $\hat{y}_i^{(t)}$ tem-se:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \quad (4.40)$$

finalmente, para o termo de regularização da árvore de decisão $\Omega(f_k)$, representa-se como:

$$\Omega(f_k) = yT + \frac{1}{2}\lambda \sum_{j=1}^{t} w_j^2 \quad (4.41)$$

onde y representa a complexidade de cada ramo, T é o número de ramos na árvore de decisão, λ é o parâmetro para scalar a penalidade, e w é o vetor de *scores* nos ramos.

4.6.3 Máquina Simplificada de Gradiente Aumentado

A máquina simplificada de gradiente aumentado (do inglês *light gradient boosting machine,* LightGBM) é também um *framework* que é baseado no algoritmo

de g*radient boosting*, que diferencialmente de outros algoritmos, distribui os melhores resultados entre as suas folhas, ao contrário de outros modelos que fazem a distribuição em profundidade (ou *level*) na árvore. Isso permite com que os valores de perda acabem sendo reduzidos conforme se cresce as folhas do mesmo (Ke *et al.*, 2017).

Sua estratégia envolve em fazer a separação das folhas que estão no mesmo nível simultaneamente, porém, cada uma das folhas contém valores únicos que permitem um ganho de informação único. Tais ganhos indicam uma redução esperada em sua entropia, causada pelas divisões baseadas em atributos, que podem ser determinados por:

$$\mathrm{IG}(B,V) = En(B) - \sum_{v \in Values(V)} \frac{|B_v|}{B} En(B_v) \qquad (4.42)$$

onde $En(B)$ é a entropia da informação coletada de B, que pode ser representada por:

$$En(B) = \sum_{d=1}^{D} -p_d \log_2 p_d \qquad (4.43)$$

onde p_d é o ratio B pertinente a categoria d e D é o número de categorias existentes, v é o valor do atributo V e B_v é um subset de B para o atributo que contém o valor de v.

Como potencialidades que se tem para o uso de LightGBM, pode-se citar o seu rápido treinamento, uma vez que usa paralelismo entre as suas folhas, o baixo consumo de memória, uma vez que os valores são continuamente modificados e não é necessário mantê-los em memória e a possibilidade de ser usado em grande banco de dados, uma vez que usa menos dados para serem mantidos em memória.

5 OTIMIZAÇÃO POR METAHEURÍSTICA

As heurísticas são procedimentos de solução que, em vários casos, se apoiam em uma abordagem intuitiva, na qual a estrutura particular do problema possa ser considerada e explorada de forma “inteligente”, para a obtenção de uma solução adequada (Cunha, 1997).

Até o final da década de 1980, os métodos heurísticos propostos para resolver problemas de otimização eram, em sua maioria, específicos e dedicados a um dado problema. A partir daí esse paradigma evoluiu e surgiu um crescente interesse em técnicas que sejam mais flexíveis e por isso, aplicáveis a diversos problemas. Estas técnicas são conhecidas por meta-heurísticas. Dentre as quais destacam-se: *simulated annealing*, busca tabu, algoritmos genéticos, *scatter search*, GRASP (do inglês *greedy randomized adaptive search procedures*), colônia de formigas, busca em vizinhança variável (do inglês v*ariable neighbourhood search*, VNS), entre outras abordagens na maior parte oriundas da computação evolutiva e inteligência de enxames. A seguir são mencionados os algoritmos de otimização Bayesiana e a evolução diferencial, ambos adotados em procedimento de sintonia de hiperparâmetros dos modelos de aprendizado de máquina avaliados na tese.

5.1 Otimização por Algoritmo Bayesiano

Como na maioria dos casos, o algoritmo de otimização Bayesiana (do inglês *bayesian optimization algorithm*, BOA), procura achar o menor valor para a função $f(x)$ em região limitada por X, que é um subset de $\mathbb{R}^D$. Sua característica é que ela constrói o modelo probabilístico do modelo $f(x)$ e então explora-o para fazer decisões sobre onde em X deve-se ir em “regiões ainda incertas” (Pelikan ***et al.***, 1999). A principal filosofia é de se usar todas as informações obtidas de análises anteriores de $f(x)$ e não apenas em gradientes locais ou aproximações Hessianas. Esta abordagem resulta em facilitar a descoberta do mínimo valor para a função custo com poucos passos, porém, usando mais cálculos para determinar os próximos pontos a serem testados. Quando usado para determinar $f(x)$ que sejam custosos de executar, como é o caso de aprendizado de máquinas, então é fácil de se entender o motivo de se fazer mais cálculos para se obter melhores decisões.

Tendo-se uma função arbitraria representada por $f: X \rightarrow \mathbb{R}$ definida sobre o set convexo $X \subset \mathbb{R}^d$ que pode calculada como $x \in X$, produzindo observações ruidosas do tipo $y \sim N(f(x), \sigma_{obs}^2)$.

Existem duas principais escolhas que podem ser feitas quando se usa otimização Bayesiana: o primeiro seria qual o método de avaliação para os valores retornados pelo modelo a ser retornado e o segundo seria a escolha de uma função aquisição, que é usada para construir uma subfunção do modelo anterior que permite assim calcular os próximos pontos a serem analisados.

5.1.1 Processo Gaussiano

O processo Gaussiano (do inglês *gaussian process*, GP) é um eficiente método para fazer a distribuição de valores na função (Zhang e Xu, 2021), tal que:

$$f : X \rightarrow \mathbb{R} \qquad (5.1)$$

o GP então pode ser definido pela propriedade que qualquer set finito de N amostras $\{x_n \in X\}_{n=1}^{N}$ que induz a distribuição Gaussiana multivariada, ou seja, no domínio $\mathbb{R}^N$. O n elemento destes pontos pode ser usados então para a função custo $f(x_n)$ e os valores das propriedades da distribuição Gaussiana permitem calcular os valores marginais e condicionais em um formato definido.

5.2 Otimização por Algoritmo de Evolução Diferencial

A evolução diferencial é um algoritmo de otimização estocástico da área de computação evolutiva que usa a diferença de orientação entre suas soluções candidatas para se “orientar” no espaço de busca discreto (Storn e Price, 1997). Sua população, com *NP* soluções candidatas (ou indivíduos), utiliza três operadores (mutação, cruzamento e seleção) a cada geração. A cada geração, as três operações são executadas, que gera então um vetor mutante na operação de mutação, que gera um terceiro vetor (vetor de testes) na operação de cruzamento que é gerado por meio do cruzamento do vetor mutação e o vetor alvo. Sendo que finalmente na última fase, na operação de seleção, apenas a melhor solução candidata é mantida.

5.2.1 Operadores da Evolução Diferencial

A evolução diferencial adota as seguintes operações no seu ciclo de otimização:

Operação de mutação: Sejam os vetores X_α, X_β e X_γ escolhidos aleatoriamente e distintos entre si. Na geração q um par de vetores (X_β, X_γ) define uma diferença X_β – X_γ. Esta diferença é multiplicada por F > 0, sendo denotada por diferença ponderada, e é usada para perturbar o terceiro vetor X_α ou o melhor vetor, denominado de X_{best}, da população. Este processo que resulta o vetor doador $V^{(q+1)}$ pode ser escrito pela expressão

$$V^{(q+1)} = X_\alpha^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) ou\, V^{(q+1)} = X_{best}^{(q)} + F\left(X_\beta^{(q)} - X_\gamma^{(q)}\right) \qquad (5.2)$$

onde os índices escolhidos aleatoriamente baseados em geração de índices aleatórios com distribuição uniforme para $\alpha, \beta, \gamma \in \{1, \dots, N_p\}$ são inteiros distintos entre si e diferentes do índice d. F é um número $F \in \Re$ pertencente ao intervalo [0,2] que controla a amplitude da diferença ponderada. A Figura 5.1 mostra um exemplo bidimensional que ilustra os diferentes vetores que participam da geração do vetor doador $V^{(q+1)}$.

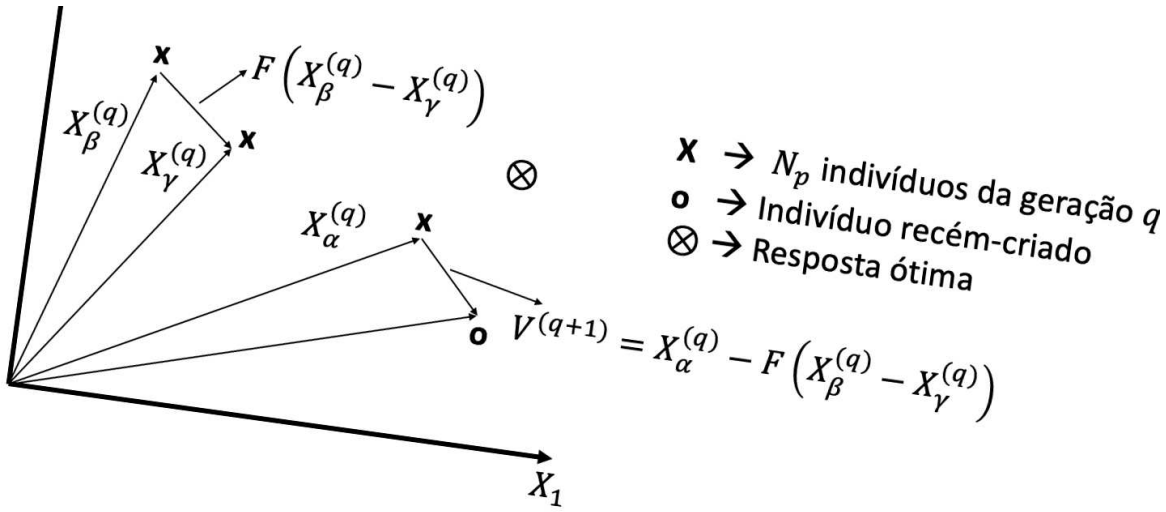

Figura 5.1: Processo de gerar o vetor doador $V^{(q+1)}$ para uma função objetivo de um problema bidimensional.

Se o número de indivíduos da população é grande o suficiente, a diversidade da população pode ser melhorada usando duas diferenças ponderadas para perturbar um vetor existente, ou seja, cinco vetores distintos são escolhidos aleatoriamente na população atual. O vetor diferença ponderada usa dois pares de diferenças ponderadas e é usado para perturbar o quinto vetor ou o melhor vetor da população atual. Este processo pode dado por:

$$V^{(q+1)} = X_{\alpha}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \quad (5.3)$$

ou

$$V^{(q+1)} = X_{best}^{(q)} + F\left(X_{\lambda}^{(q)} - X_{\beta}^{(q)} + X_{\gamma}^{(q)} - X_{\delta}^{(q)}\right) \quad (5.4)$$

onde os índices escolhidos aleatoriamente $\alpha, \beta, \gamma, \lambda \in \{1, \ldots, N_p\}$, são inteiros mutuamente distintos e diferentes do índice *d*, tal que $N_p \geq 6$. E $X_{best}^{(q)}$ é o indivíduo da população que se encontra no momento com o melhor resultado obtido pela função ativação.

Operação de cruzamento: O cruzamento é introduzido para aumentar a diversidade dos indivíduos que sofreram a mutação. Assim, as componentes do vetor experimental $U^{(q+1)}$ são formadas conforme a expressão:

$$U_i^{(q+1)} = \begin{cases} V_i^{(q+1)}, se\ r_i \leq CR \\ x_{d,i}^{(q)}, se\ r_i > CR, i = 1, \ldots, n \end{cases} \quad (5.5)$$

onde r_i é um número gerado aleatoriamente com distribuição uniforme no intervalo [0,1]. Os elementos $x_{d,i}$ constituem o vetor alvo $X_d^{(q)}$, CR é a probabilidade de o cruzamento ocorrer, representando a probabilidade de o vetor experimental herdar os valores das variáveis do vetor doador, e está compreendida entre 0 e 1, sendo fornecida pelo usuário ou projetista. Quando $CR = 1$, por exemplo, todas as componentes do vetor experimental virão do vetor doador $V^{(q+1)}$. Por outro lado, se $CR = 0$, todas as componentes do vetor experimental virão do vetor alvo $X_d^{(q)}$.

Se após o cruzamento uma ou mais componentes do vetor experimental estiver fora da região de busca, fazem-se as correções:

$$\begin{cases} \text{Se } u_j < x_i^L \text{, então } u_i = x_i^L \\ \text{Se } u_j < x_i^U \text{, então } u_i = x_i^U \end{cases} \quad (5.6)$$

Operação de seleção: A seleção é o processo de produzir melhores filhos. Diferentemente de outros algoritmos evolutivos, a evolução diferencial não usa hierarquia (elitismo) nem seleção proporcional. Em vez disso, o custo do vetor experimental $U^{(q+1)}$ é calculado e comparado com o custo do vetor alvo $X_d^{(q)}$. Se o custo do vetor alvo for menor que o custo do vetor experimental, o vetor alvo é permitido avançar para a próxima geração. Caso contrário, o vetor experimental substitui o vetor alvo na geração seguinte. Em outras palavras, a operação pode ser formulada por:

$$\begin{cases} \text{Se } f\left(U^{(q+1)}\right) \leq f\left(X_d^{(q)}\right) \text{, então } X_d^{(q+1)} = U^{(q+1)} \\ \text{Se } f\left(U^{(q+1)}\right) > f\left(X_d^{(q)}\right) \text{, então } X_d^{(q+1)} = X_d^{(q)} \end{cases} \quad (5.7)$$

O procedimento mencionado é finalizado por meio de algum critério de parada, sendo que um número máximo de gerações deve ser estabelecido.

5.3 Métricas de desempenho

A análise dos resultados de um modelo é uma parte importante e essencial para a construção de um modelo efetivo de aprendizado de máquina. Várias métricas de desempenho já foram sugeridas pela comunidade científica sendo diferentes métricas de desempenho são usadas para diferentes tipos de problemas (Srivastava, 2019).

Em problemas de classificação, algumas das métricas mais usadas é a MSE, a raiz de erro quadrático médio (do inglês root mean squared error, RMSE) e a RMSLE:

- MSE : é uma das métricas de desempenho mais simples, que apenas faz a soma dos quadrados dos resíduos calculado pela diferença entre os valores

previstos P e os valores reais O dividida pelo número de termos. É definida como

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(O_i - P_i)^2 \quad (5.8)$$

- RMSE: É uma das mais populares métricas de desempenho usada para problemas de regressão. Ela assume que o erro é livre de bias e que segue uma distribuição normal. Seu valor pode ser definido como

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(O_i - P_i)^2} \quad (5.9)$$

  Ao contrário do MSE, possui uma unidade (dimensão), representada por n, igual à dimensão dos valores observados e preditos. Interpreta-se seu valor como uma medida do desvio médio entre observado e predito, porém observe que as diferenças entre o valor original, O e o valor previsto P são elevadas ao quadrado.
- RMSLE: Está métrica de desempenho permite uma melhor robustez para base de dados que acabem tendo muitos *outliers* e acaba usando o erro relativo, já que é o valor logaritmo que é usado, acabando então penalizando mais os valores previstos que forem menores que os valores reais. É definida como:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(O_i + 1) - \log(P_i + 1))^2} \quad (5.10)$$

onde novamente se tem que os valores originais estão sendo representados por O e os valores previstos por P.

5.4 Seleção de Características

Uma característica (*feature* em inglês) é uma propriedade mensurável de um processo que está sendo observado. Usando um conjunto de características, os sistemas de aprendizado de máquina podem aprender e executar predições. Nota-se que para uma coleção total de *N* características, o número de subconjuntos de tais características pode ser representado por:

$$\frac{N!}{(m! * (N-m)!)} \quad (5.11)$$

onde m representa a quantidade de características existente no espaço de dados (Jensen, 2005).

Várias técnicas então foram sugeridas para que se possa analisar tais característica para que se remova as características irrelevantes e redundantes, o que permite os requerimentos computacionais serem menores, uma vez que se obtém uma representação mínima dos dados, aumenta a performance de predição, já que não é necessária nenhuma informação adicional sobre os dados, como limiares e apenas os dados mais relevantes acabam sendo analisados. (Raj *et al.*, 2020).

A RFE é baseada em um sistema de *wrapper* que usa um subconjunto das características e as usa para treinar o modelo ao contrário de um sistema de filtro, que geralmente acaba sendo um pré-processamento dos dados. O RFE usa a precisão em predições para validar o *subset* de características. Tais métodos usam um sistema de aprendizado de máquina como uma caixa preta para pontuar as características dependendo em sua habilidade em predição (Elavarasan, 2020). RFE é fundamentalmente um processo recursivo que classifica tais características baseado em suas medidas de significância. RFE começa então eliminando um subconjunto por vez, conforme a sua performance, até que o seu critério de parada seja atingido. A necessidade de se fazer isso recursivamente é porque pode existir uma correlação entre os subconjuntos restantes que acabe mudando significativamente o resultado. O resultado, como esperado, seria a obtenção da quantidade de características pré-determinadas pelo usuário como critério de parada.

O uso de árvore aleatória (em inglês *random forest*) como o sistema para analisar a predição das características é um dos modelos mais usados, conhecido

como método RF-RFE (Govindarajan, 2020). O seu critério para avaliar a importância das características é a seguinte:

- No início das iterações, as características remanescentes são misturadas e como resultado, um modelo de previsão do erro é calculado;

- Naturalmente, quando se usa o modelo gerado contra cada uma das características, as características com menor significância não irão ter sua predição afetadas significativamente, enquanto os valores com maior significância irão agir de maneira inversa;

- A perda correspondente aos valores atuais e a base de dados misturadas é então associado com a eficiência das características misturadas;

- A característica com a menor significância é então removida e uma nova iteração é realizada.

6 EXPERIMENTOS

Neste capítulo uma descrição da base de dados, ambientes computacionais adotados, estrutura de projeto e análise de resultados são apresentados.

6.1 Base de Dados

A análise dos resultados de previsão foi realizada baseada em duas bases de dados.

A base de dados do Brasil foi coletada diretamente no site do Operador Nacional do Sistema Elétrico do Brasil (ONS, 2021), e possui características com valores por hora e em MegaWatts. Os dados abordados nesta tese são referentes ao mês de janeiro de 2017, totalizando assim 673 amostras. Desse total, foram utilizadas as duas primeiras semanas de dados (com 336 entradas, 50% do total) para a etapa de treinamento dos modelos, a próxima semana consequente para validação abrangendo 168 amostras (12,5% do total), e finalmente a última semana para teste dos modelos com 169 amostras (12,5% do total).

A outra base de dados é a *Poland Electricity Power* (Aalto, 2021), que contém valores relacionados ao consumo diário de energia na Polônia em 1990 em GigaWatts. Essa base de dados contém ao todo 1601 entradas, que se apresentam divididas em 1200 valores de treinamento (75% do total), 200 amostras para validação (12.5% do total) e 201 para teste (12.5% do total) dos modelos de previsão.

6.2 Ambiente Computacional

Os códigos fonte vinculados a tese foram escritos em linguagem Python® 3.7, rodando um sistema operacional Windows versão 10. As bibliotecas utilizadas também foram implementadas em Python e todas são códigos abertos, que podem ser obtidos em sítios na Internet:

- Numpy: Esta biblioteca permite o uso de cálculos matemáticos para grandes listas (*arrays*), sendo usada em aprendizado de máquina por permitir o uso de conjunto de dados com acentuada quantidade de dados. Esta biblioteca também é muito usada para transformações matemáticas e operações com matrizes (Harris *et al.*, 2020).
- Scikit-learn: Também conhecida como sklearn, é uma biblioteca com várias métodos e algoritmos para o uso em aprendizado de máquina.

Entre os principais motivos para se usar tal biblioteca é que a mesma permite uma padronização dos dados que precisam ser usados para seus métodos, permitindo uma adequada avaliação entre eles (Pedregosa *et al.,* 2011).

- Pandas: Biblioteca frequentemente usada em manipulação e análise de dados. Uma das suas principais funcionalidades é manipulação de tabelas inteiras e não apenas listas. Permite também uma manipulação mais rápida do que o uso de dicionários em Python porque usa uma otimização interna de seus dados para que permita a leitura rápida de dados (The Pandas Development team, 2020).
- Tensorflow: Principal biblioteca usada para o uso de aprendizado de máquina baseado em linguagem Python. Seu código é considerado de baixo nível o que exige um conhecimento avançado da linguagem, porém, outras bibliotecas encapsulam esta biblioteca, permitindo uma maneira mais fácil de seu uso. Um dos diferenciais desta linguagem é a possibilidade de se usar o *graphical processing unit* (GPU) das placas de vídeos diretamente (Abadi *et al.,* 2016).
- Keras: Esta biblioteca é uma interface para o uso de redes neurais artificiais, sendo que internamente pode realizar a chamada de outra biblioteca aqui citada: a Tensorflow (Chollet, 2018).
- pyESN: Biblioteca especialmente feita para o uso de ESN em python®, sendo que internamente usa as bibliotecas tensorflow e numpy (Stewart, 2020).
- pmdArima: Biblioteca que implementa os modelos ARIMA e SARIMA em Python. (Smith *et al.,* 2021)
- bayes_opt: Biblioteca que implementa o algoritmo de otimização Bayesiana em Python. (Nogueira, 2014)
- XGBoost: Biblioteca usada para o uso de *gradient boosting*, com a possibilidade de distribuir em um grupo de máquinas o processamento a ser realizado para obter os modelos (Chen e Guestrin, 2016)

Para geração das figuras, foi usada outra ferramenta de código aberto, denominada Jupyter, que também executa códigos implementados em Python.

porém, a ferramenta computacional Jupyter permite a criação de blocos de códigos que podem ser rodados em diferente ordem e que permitem o uso de componentes gráficos para se analisar os dados. Neste caso, a biblioteca suportada pela ferramenta é o MatplotLib que permite a inclusão de diferentes dados em um mesmo gráfico.

6.3 Descrição da Arquitetura

Inicialmente, os dados precisaram ser analisados e separados em diferentes partes para que possam ser usados em diferentes momentos do sistema. Logo, os itens 1 a 6 relacionados na Figura 6.1, correspondem aos sinais de entrada e saída que podem ser os dados extraídos da série temporal.

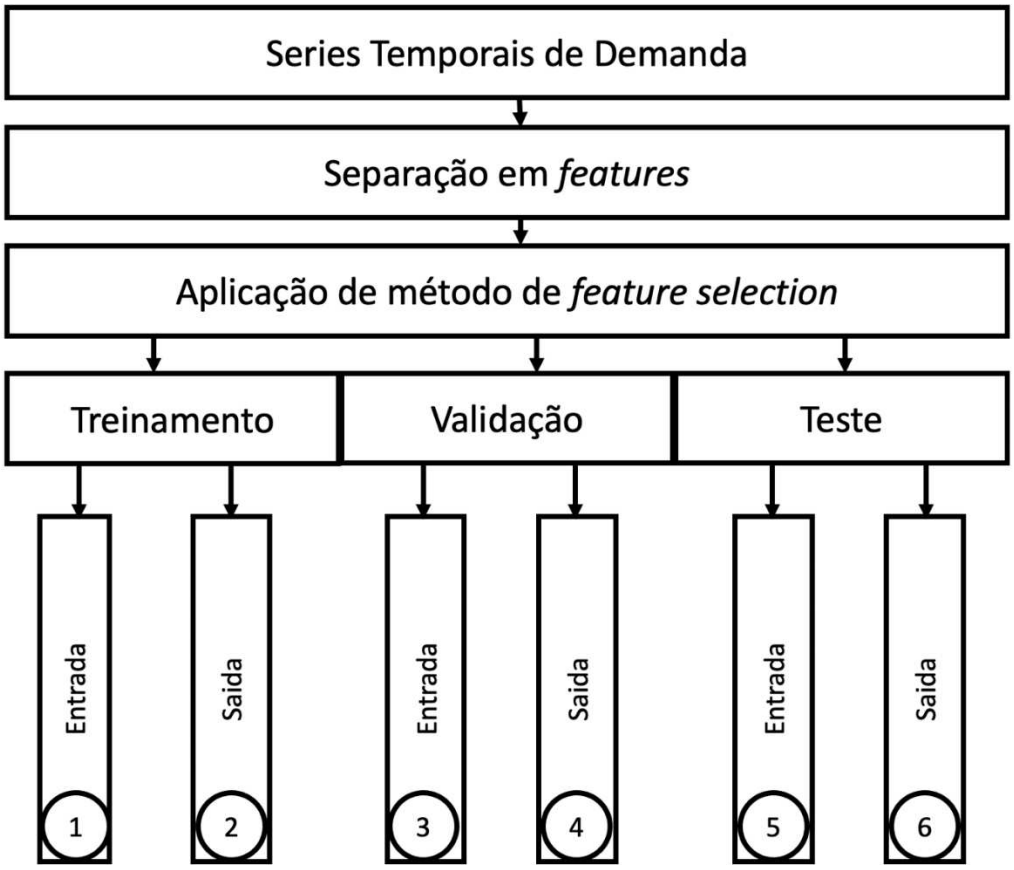

Figura 6.1: Ilustração das extrações dos valores da série temporal

Na Tabela 6.1 encontra-se os valores relativos aos valores extraídos neste processo para as bases de dados da demanda de consumo da região Sul do Brasil e da Polônia.

	Região Sul do Brasil	Polônia
Treinamento-Entrada (1)	336	1200
Treinamento-Saída (2)	336	1200
Validação-Entrada (3)	168	200
Validação-Saída (4)	168	200
Teste-Entrada (5)	169	201
Teste-Saída (6)	169	201

Tabela 6.1: Tamanho dos vetores usados no experimento para a região Sul do Brasil e da Polônia

Além disso, os valores dos dados de entrada original foram quebrados de acordo com a sua sazonalidade, criando diferentes blocos de dados denominados características.

Em seguida, usou-se o RFE para se analisar tais características e escolher apenas as 10 principais características com maior importância para uso com variáveis previsoras das séries temporais. Para analisar o uso apenas das características mais significativas nos resultados, foi executado também testes usando todas as características disponíveis, para que fosse possível fazer um comparativo.

Os algoritmos de aprendizado de máquina possuem além de suas variáveis internas que são ajustadas durante a fase de treinamento, parâmetros que permanecem inalterados durante o aprendizado e que estão relacionadas com a estrutura do modelo, conhecidos como hiperparâmetros. O conjunto de hiperparâmetros ótimo é normalmente identificado por meio de um processo de otimização.

A Figura 6.2 apresenta o fluxograma do sistema por algoritmo por execução, que seguindo-se os resultados da Figura 6.1, temos que os valores 1 e 2 são os dados de treinamento, aplicado ao algoritmo, sendo então passado para o algoritmo treinado os valores de validação de entrada 3, que é validado usando-se os valores de saída de treinamento 4. Enquanto o critério de parada do algoritmo de otimização não for atingido, tal procedimento ocorre múltiplas vezes, salvando-se sempre o melhor modelo obtido até o momento. Finalmente, ao se obter o melhor modelo otimizado, aplica-se os valores de entrada de teste (5). Que é verificado uma última vez pelos valores de saída do teste (6) usando-se o RMSLE.

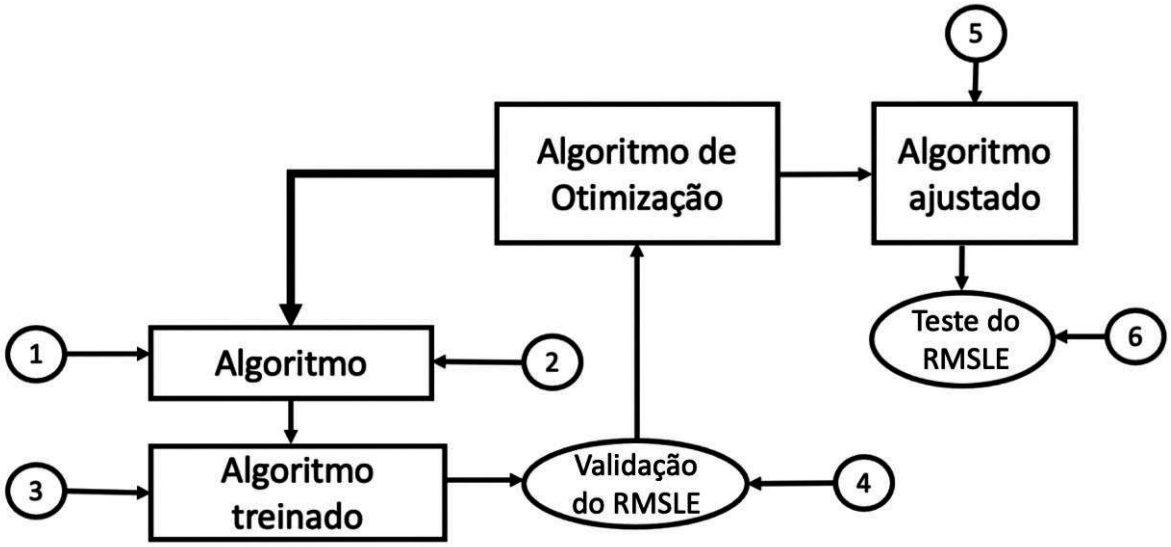

Figura 6.2: Fluxograma do sistema

Para obter uma análise detalhada, o mesmo fluxograma foi executado com o uso de todas as características disponíveis e depois, apenas com as dez melhores características, permitindo-se uma análise dos dados para confirmar ou não a melhora do algoritmo quando se é eliminada características que não tem relevância na contribuição final, mas que possam gerar bias ou ruídos não desejados.

6.4 Hiperparâmetros

Cada um dos algoritmos (modelos) usados para previsão de séries temporais, tem variáveis de hiperparâmetros distintas que podem ser treinadas para que consigam ficar otimizadas para uma melhor previsão para a série temporal a ser analisada. Algumas configurações foram comuns para todos os modelos, para tentar acomodar um padrão entre os valores de testes. Entre todos os modelos, por exemplo, para a quantidade de camadas ocultas os valores usados foram entre 2 e 100. Outros valores, acabaram sendo usados os valores padrões das bibliotecas e apenas os parâmetros mais usados na literatura ou sugeridos pelas bibliotecas acabaram sendo usados para ajuste. Tais valores estão sendo explicados em seguida.

6.4.1 Modelo ARIMA

ARIMA tem como base os seus três parâmetros de ordem *p*, *d* e *q*, todos no domínio dos valores inteiros, que se não tratados com valores iniciais, contém valores aleatórios. Por causa de várias limitações encontradas em valores altos pela biblioteca

pmdArima, como por exemplo, o valor de *q* não pode ser múltiplo de valores de *p*, usou-se então como limites para os algoritmos de otimização para *p* e *q* entre 1 e 3 e para *d* valores entre 1 e 2.

6.4.2 CNN

Para CNN, a configuração usada para se criar a rede foi o uso de uma camada simples de convolução do tipo 1D com uma camada do tipo *MaxPooling* usando como função de perda o RMSLE e otimizador do tipo *adam*. Já para os valores dos hiperparâmetros foram usados três valores para serem otimizados nos algoritmos de otimização: os filtros na camada de convolação inicial, que variou entre números inteiros de 1 e 100, o tamanho máximo para o *max-pooling*, que é o tamanho máximo da região a ser analisada como uma única característica, que variou entre números inteiros de 1 e 99 (sendo que 100 não era permitido pela biblioteca), o quantidade de unidades de saída da penúltima camada, que poderia variar entre números inteiros de 1 e 100. Uma camada mais densa significa mais características por resultado, porém que podem gerar um sobreajuste ao longo do treinamento.

6.4.3 RF

RF permitiu o uso de seis variáveis para serem otimizadas no total: o número de árvores na floresta: no intervalo de números inteiros entre 2 e 100, a máxima profundidade na árvore, quanto maior a profundidade maior a quantidade de características analisadas por árvore, porém maior as chances de se ter um sobreajuste, que variou entre números inteiros de 1 e 100, o mínimo número de amostras para que fosse possível uma divisão interna no ramo, que variou entre números inteiros de 2 e 10 a mínima quantidade de exemplos por ramo, que variou entre números de ponto flutuante entre 1 e 10, a mínima fração de peso que um ramo tem que ter em relação a todos os pesos somados, que variou entre números de ponto flutuante entre 0 e 0,5 e a máxima quantidade de folhas por ramo que variou entre números inteiros de 2 e 10.

A biblioteca sklearn ainda poderia ter alguns outros parâmetros sendo analisados pelo otimizador, porém, isso poderia afetar a qualidade dos resultados dos outros parâmetros, conforme explicado na página oficial da biblioteca. Por isso,

escolheu-se não executar todos os possíveis parâmetros, mas apenas os mais importantes, que são citados na página oficial da biblioteca.

6.4.4 ESN

A biblioteca usada, pyESN, permitiu o uso de quatro variáveis que poderiam ser otimizadas: o número de itens no reservatório, que variou entre números inteiros de 1 e 100, o rádio espectral do peso da matriz que variou entre números de ponto flutuante de 0,01 e 1, o número de dados de entrada antes de inseri-los na rede interna, que variou entre números inteiros de 0 e 50 e o valor a ser usado para multiplicar os valores de entrada antes de jogá-los na rede: variou entre números de ponto flutuante de 0,5 e 1.

6.4.5 SVR

O SVR teve seis possíveis valores de hiperparâmetros sendo analisados, sendo eles o tipo de *kernel* (*linear*, *poly*, *rbf*, *sigmoid*, *precomputed*), o grau (do inglês *degree*) da função em caso de que a função *kernel* seja do tipo *poly*, o *gamma*, que variou números de ponto flutuante de 0,01 e 1, o valor do termo independente do *kernel*, que variou números de ponto flutuante de 0,01 e 1 (apenas caso o *kernel* seja do tipo *poly* or *sigmoid*), a tolerância do critério de parada, que variou números de ponto flutuante de $1e^{-5}$ e 0,1, e ainda o valor de épsilon, que variou números de ponto flutuante de 0,1 e 1.

6.4.6 LSTM

No caso do LSTM, apenas dois valores foram analisados, sendo novamente o *kernel*, que variou entre (*sigmoid, linear, tanh e softmax*) e o número de unidades internas que o LSTM poderia ter, que variou entre números inteiros de 100 e 1000.

6.4.7 OLS

O OLS teve os números de *threads (*tarefas rodando em paralelo) variando entre números inteiros de 1 e 100 e se os valores de y deveriam começar em 0 ou não (o que significaria que os dados não estavam centralizados)

6.4.8 XGBoost

O XGBoost, de acordo com a página oficial da biblioteca, permite mais de 30 parâmetros de configuração, porém, a própria página sugere o uso apenas de alguns parâmetros como: a profundidade máxima das árvores de decisão (Maximum *Depth*), que variou entre números inteiros de 10 e 100, a curva de aprendizado (*Learning Rate*), que variou entre números de ponto flutuante de 0,1 e 0,9, o tamanho das subamostras, que variou entre números de ponto flutuante 0,1 e 0,9 e o mínimo valor de peso que um ramo pode ter, que variou entre números inteiros de 1 e 5.

6.4.9 QRF

No QRF, os números de estimadores variaram entre números inteiros de 2 e 200, a profundidade máxima do galho variou entre números inteiros de 1 e 100, o número mínimo de amostras para que haja uma divisão que variou entre números inteiros de 2 e 10 e finalmente o número máximo de nós em um mesmo ramo que variou entre números inteiros de 2 e 10.

6.4.10 FFNN

O FFNN, teve as mesmas configurações apresentadas para o LSTM, uma vez que a biblioteca usada para ambos é a mesma e a mudança apenas é o método usado para os nós que se encontram no meio da solução.

6.4.11 Modelo SARIMA

Para o SARIMA, as configurações para o ARIMA foram aplicadas novos valores para a sazonalidade, que seriam três variáveis variando entre números inteiros de 0 e 5. É importante notar que o SARIMA internamente tem restrições entre tais valores, como por exemplo, os valores de P e Q não podem ser os mesmos. Por isso, foi necessário ignorar tais erros quando ocorreram e apenas retornar um valor alto para a otimização para que os mesmos não ocorressem novamente.

6.4.12 LightGBM

Para o LightGBM, seis diferentes valores foram usados, sendo o *maximum_depth*, responsável por limitar a profundidade máxima que a árvore pode ter, que variou entre números inteiros de 1 e 100, o *learning_rate*, responsável por

controlar os efeitos das variações das amostras, que variou entre números de ponto flutuante de 0,001 e 0,999, *number_of_leaves*, responsável pela quantidade de ganhos nas árvores, que variou entre números inteiros de 0 e 2, o *boosting_type*, que poderia ser o *Gradient Boosting*, *Random Forest*, *Multiple Additive Regression trees* e *Gradient-based One-Side Sampling*. Ainda pode-se mudar os valores de frequência de fração para o *Baggings*, que é usado para aleatoriamente selecionar partes dos dados sem precisar fazer uma nova amostragem, que variou entre números inteiros de 1 e 10.

6.4.13 DRVFL

No total, foram usadas apenas duas camadas internas para o DRVFL, sendo que também possível controlar os números de neurônios na camada de entrada e de saída. Logo, quatro possíveis valores foram otimizados, sendo que os valores poderiam variar entre números inteiros de 2 e 100

6.5 Análise dos Resultados

Inicialmente, aplicou-se uma normalização nos dados, usando-se o método *min-max* que ajusta os valores em um intervalo determinado de [0,1], o que permitiu conservar o formato original da série temporal. Em seguida, ao se analisar a Figura 6.3 e Figura 6.4 nota-se uma sequência recorrente que permite ser usada para a criação das diferentes características das series na previsão de 1 passo a frente. No caso da base de dados do Brasil, foram usados atrasos de 24 entradas para gerar novas características, enquanto no caso dos dados da Polônia, seriam de 7. Esses foram os valores usados na primeira etapa, que seria a criação de uma base com características que representam tais valores.

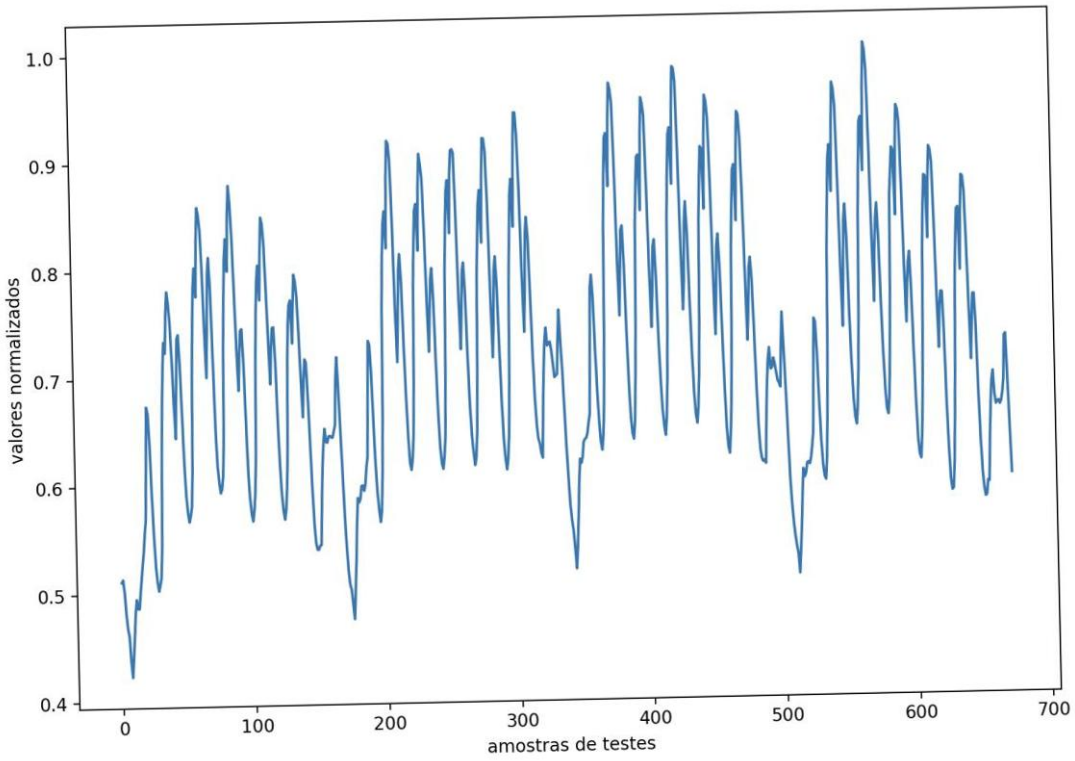

Figura 6.3: Valores normalizados para a série temporal de consumo da região Sul do Brasil

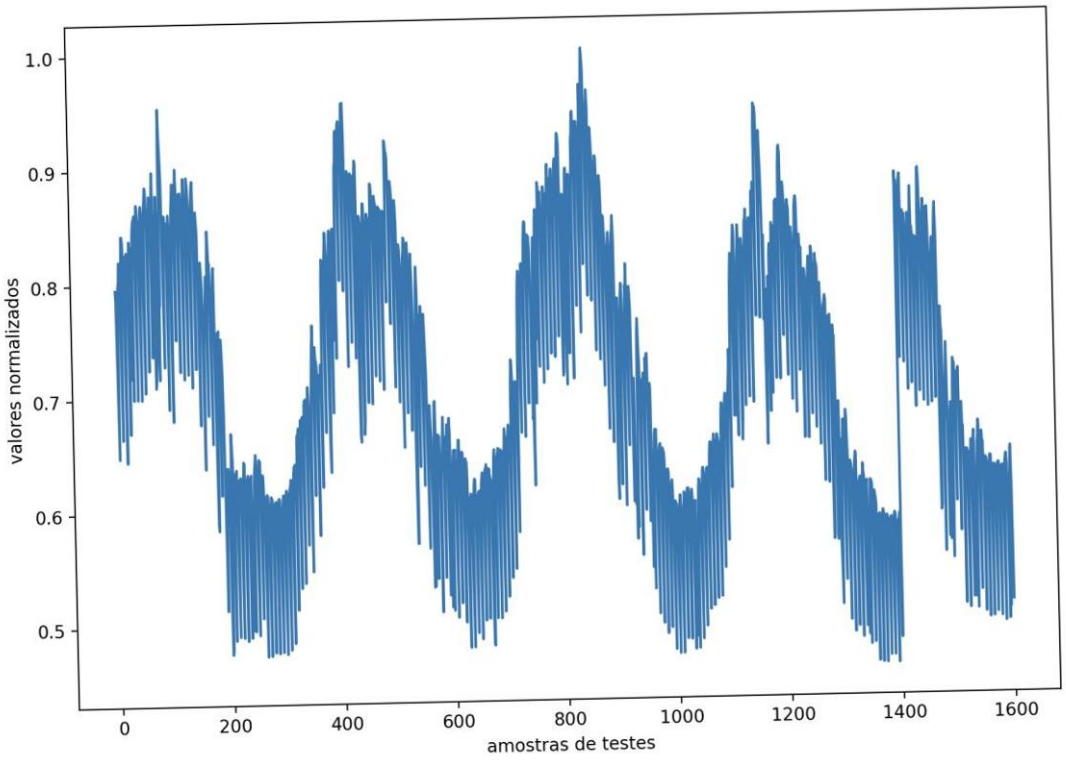

Figura 6.4: Valores normalizados para a série temporal de consumo na Polônia

Usou-se como critério de desempenho o RMSLE, sendo que o modelo mais adequado para a base de dados é aquele que possuiu um menor valor para o RMSLE, sendo que quanto mais perto de 0, melhor é a métrica e melhor é a qualidade do modelo. No caso de uso do RFE, foi usado um regressor de floresta aleatória, contendo uma profundidade máxima de número inteiro de 50, a quantidade de estimadores de número inteiro 100, o critério de seleção interno para a biblioteca foi a métrica de MSE (A biblioteca *sklearn* não permite o uso de RMSLE), o mínimo de amostras para se ter em um ramo é 1, e o mínimo de amostras necessárias em um único ramo para se poder dividir os dados em um novo ramo é de 2.

Já as Figura 6.5 e Figura 6.6, representam valores normalizados apenas para os dados de testes, que são os valores esperados para que os modelos obtenham.

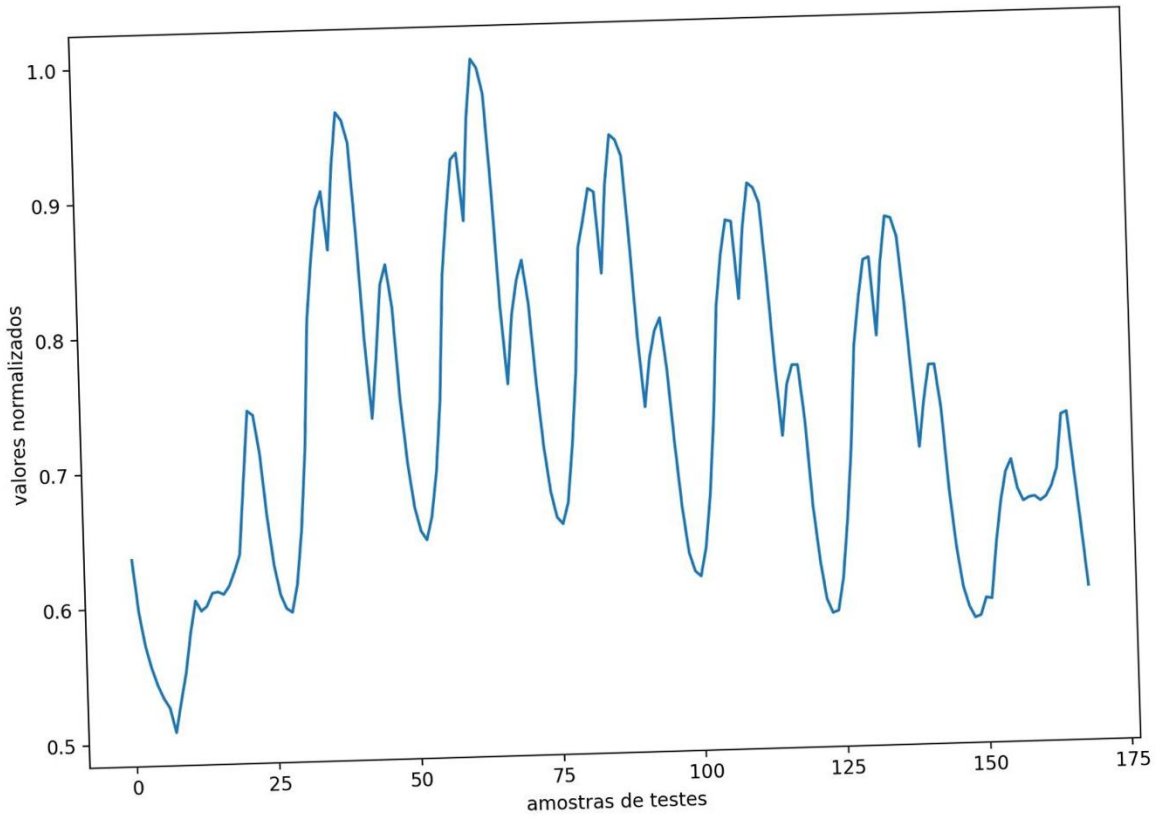

Figura 6.5: Valores normalizados para a série temporal de consumo da região Sul do Brasil, apenas usando os valores de teste

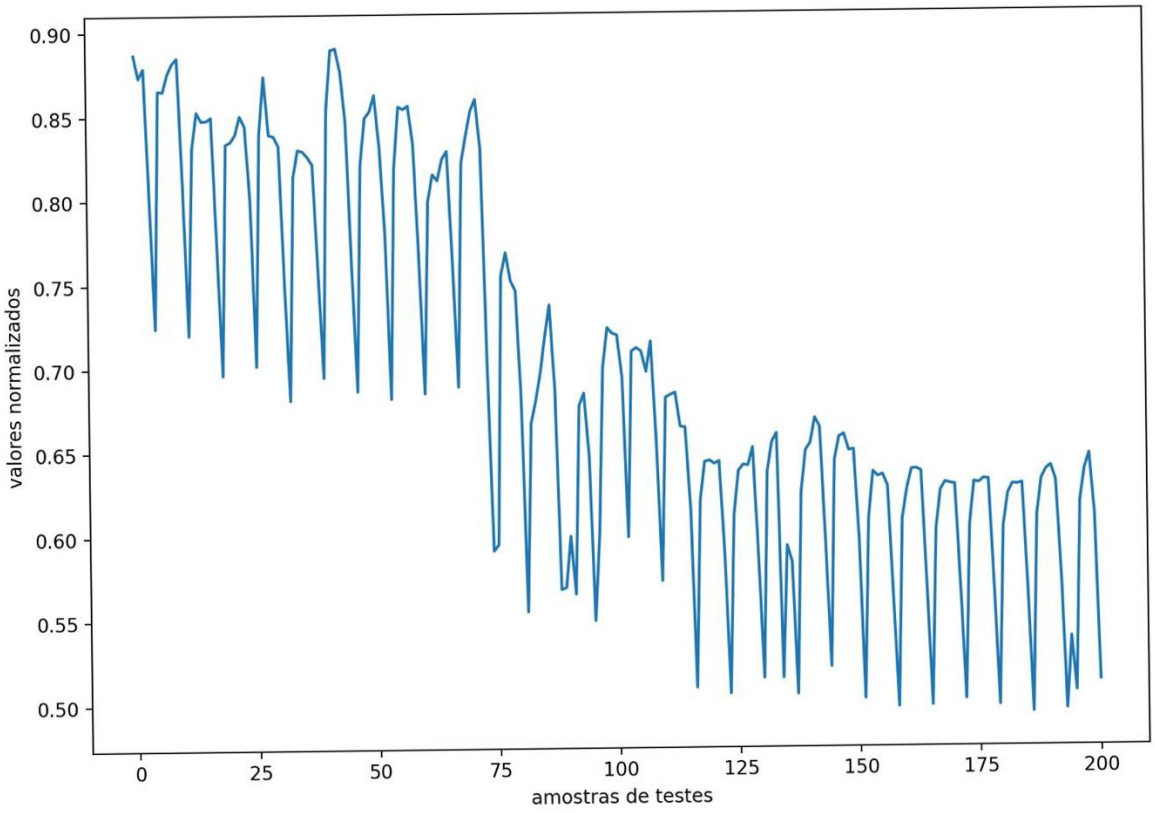

Figura 6.6: Valores normalizados para a série temporal de consumo na Polônia, apenas usando os valores de teste

Em todos os testes, as configurações dos otimizadores foram as mesmas, o número de iterações também foi o mesmo entre ambos, com o número inteiro de 50 possíveis iterações e a semente usada foi a mesma, com o número inteiro de 42. Já as configurações especificas por otimizador foram:

- Otimizador DE: O tamanho da população foi o número inteiro de 15, a estratégia usada foi a padrão, que é a *best1bin*, a tolerância para convergência foi o número de pontos flutuantes 0,01, os valores para mutação, chamado de *F* na literatura, com o número de ponto flutuante mínimo de 0,5 e de número de ponto flutuante de máximo de 1, a constante de recombinação, chamada de CR na literatura, com o número de ponto flutuante 0,7;
- Otimizador Bayesiano: O otimizador Bayesiano teve como o tipo de regressor configurado para ser do tipo *alpha*, que teve um número de ponto flutuante de $1e^{-6}$ e o seu *kernel* que foi do tipo RBF.

Por causa da natureza do ESN, que explora as características sequenciais das séries temporais, o uso de validação cruzada, que quebra a série temporal em pequenos segmentos (em inglês chamado de *k-folds),* não foi possível de ser usado.

Os resultados para 1 passo a frente para a base de dados brasileira, que podem ser observados na Tabela 6.2, mostram o desempenho RMSLE de todos os métodos para os dados do Brasil (os resultados foram arredondados usando notação cientifica), sendo que os melhores resultados foram realçados em verde, enquanto os piores foram realçados em vermelho. O DRVFL obteve os melhores valores entre todos os modelos testados, sendo que o uso do RFE melhorou ainda mais os seus resultados. Em geral, é possível de se notar que o uso ou não do RFE depende de qual algoritmo usado, pois em alguns dos casos, houve uma piora com o seu uso. Isso acontece porque as características removidas, por mais que não sejam as mais significativas (com o uso do RFE, apenas as 10 mais significativas em termos da métrica de desempenho MSE foram usadas), elas contribuíram para melhorar a predição do sistema. Por outro lado, temos o ARIMA, que obteve o pior desempenho em todos os casos e que irá ter uma explicação mais detalhada do motivo na sequência desta seção.

Algoritmo	Sem RFE		Com RFE	
	Bayesiano	DE	Bayesiano	DE
ARIMA	$6{,}1868e^{-2}$	$6{,}1868e^{-2}$	$6{,}1868e^{-2}$	$6{,}1868e^{-2}$
CNN	$2{,}4259e^{-2}$	$2{,}2054e^{-2}$	$2{,}9659e^{-2}$	$2{,}4697e^{-2}$
DRVFL	$6{,}1978e^{-3}$	$6{,}1989e^{-3}$	$6{,}1114e^{-3}$	$6{,}0252e^{-3}$
ELM	$7{,}2261e^{-3}$	$7{,}3993e^{-3}$	$7{,}6194e^{-3}$	$8{,}3236e^{-3}$
ESN	$6{,}9044e^{-3}$	$7{,}1800e^{-3}$	$6{,}5582e^{-3}$	$6{,}2175e^{-3}$
FFNN	$1{,}3309e^{-2}$	$1{,}1763e^{-2}$	$1{,}1020e^{-2}$	$1{,}1909e^{-2}$
LightGBM	$1{,}1949e^{-2}$	$1{,}2285e^{-2}$	$1{,}2188e^{-2}$	$1{,}1980e^{-2}$
LSTM	$3{,}2087e^{-2}$	$3{,}1810e^{-2}$	$3{,}2597e^{-2}$	$3{,}5530e^{-2}$
OLS	$1{,}0753e^{-2}$	$1{,}0753e^{-2}$	$1{,}5537e^{-2}$	$1{,}5537e^{-2}$
QRF	$1{,}9932e^{-2}$	$2{,}0009e^{-2}$	$2{,}0038e^{-2}$	$2{,}0300e^{-2}$
RF	$2{,}0708e^{-2}$	$2{,}0239e^{-2}$	$2{,}0896e^{-2}$	$2{,}0648e^{-2}$
SARIMA	$1{,}0968e^{-2}$	$1{,}1974e^{-2}$	$1{,}1510e^{-2}$	$1{,}0901e^{-2}$
SVR	$2{,}6127e^{-2}$	$1{,}9173e^{-2}$	$2{,}5189e^{-2}$	$2{,}0480e^{-2}$
XGBoost	$1{,}0708e^{-2}$	$1{,}1783e^{-2}$	$1{,}1354e^{-2}$	$1{,}4202e^{-2}$

Tabela 6.2: Comparativo do RMSLE para a série temporal de consumo da região Sul do Brasil

Na Figura 6.7, encontra-se os valores para o modelo ARIMA, com ou sem RFE, usando otimizadores DE e Bayesiano.

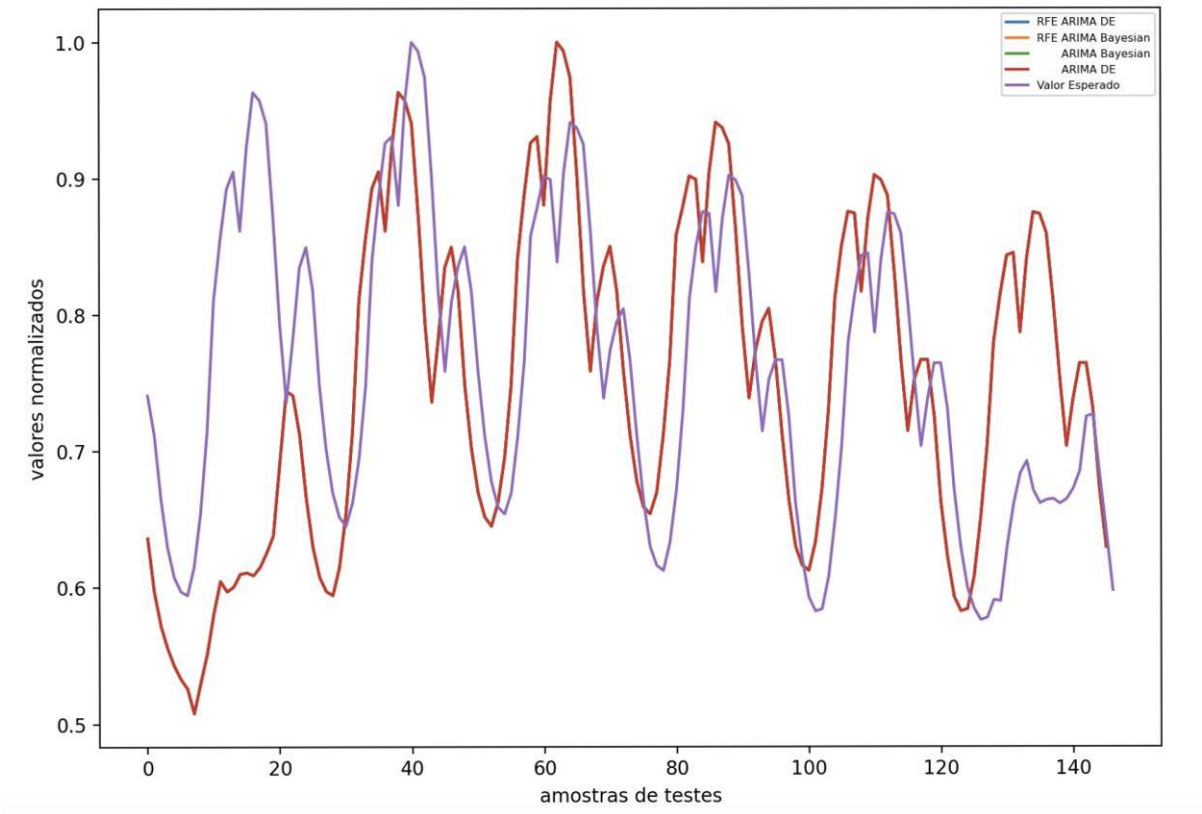

Figura 6.7: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão ARIMA

Os valores otimizados de seus hiperparâmetros podem ser encontrados na Tabela 6.3 que foram obtidos pelos otimizadores DE e Bayesiano com ou sem o uso de RFE, mostrando que houve pouca variação nos valores. Vale salientar que para o modelo ARIMA, a variação possível da área de busca para os otimizadores era pequena, variando entre 0 e 3.

modelo ARIMA	P	Q	D
Com RFE - DE	1	0	0
Com RFE- Bayesiana	1	0	0
Sem RFE – DE	2	0	0
Sem RFE - Bayesiana	2	0	2

Tabela 6.3: Valores de hiperparâmetros para o ARIMA com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Ao analisar o gráfico da Figura 6.7 e valores da métrica RMSLE na Tabela 6.2 percebe-se que o desempenho do modelo está muito ruim, sendo que primeiramente,

não houve variação entre os valores de RMSLE (Os valores tiveram diferenças, mas acabaram sendo arredondados), causando uma sobreposição dos valores. Isso aconteceu principalmente porque o modelo ARIMA não pode ser usado em casos em que há uma sazonalidade, sendo então recomendado o modelo SARIMA para esses casos.

O modelo CNN, próximo a ser analisado, teve os seus valores de hiperparâmetros configurados como apresentado na Tabela 6.4, mostra uma variação grande para todos os valores, mostrando que a busca pelo melhor valor pelos otimizadores acabou sendo feita por todo o espectro disponível.

modelo CNN	Convolução inicial	*Max-pooling*	Número saídas penúltima camada
Com RFE - DE	94	2	67
Com RFE- Bayesiana	46	1	22
Sem RFE – DE	10	7	56
Sem RFE - Bayesiana	68	4	22

Tabela 6.4: Valores de hiperparâmetros para o CNN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Em seguida, a Figura 6.8 os resultados obtidos para a modelo CNN sendo otimizado por DE ou Bayesiana e com ou sem o uso de RFE. É interessante notar que o modelo inicialmente se ajusta corretamente aos valores esperados, apenas em valores altos acaba não conseguindo prever corretamente. Os valores encontrados no RMSLE apresentam valores entre $2{,}20e^{-2}$ e $2{,}96e^{-2}$, o que mostra uma melhora significativa em relação ao modelo ARIMA.

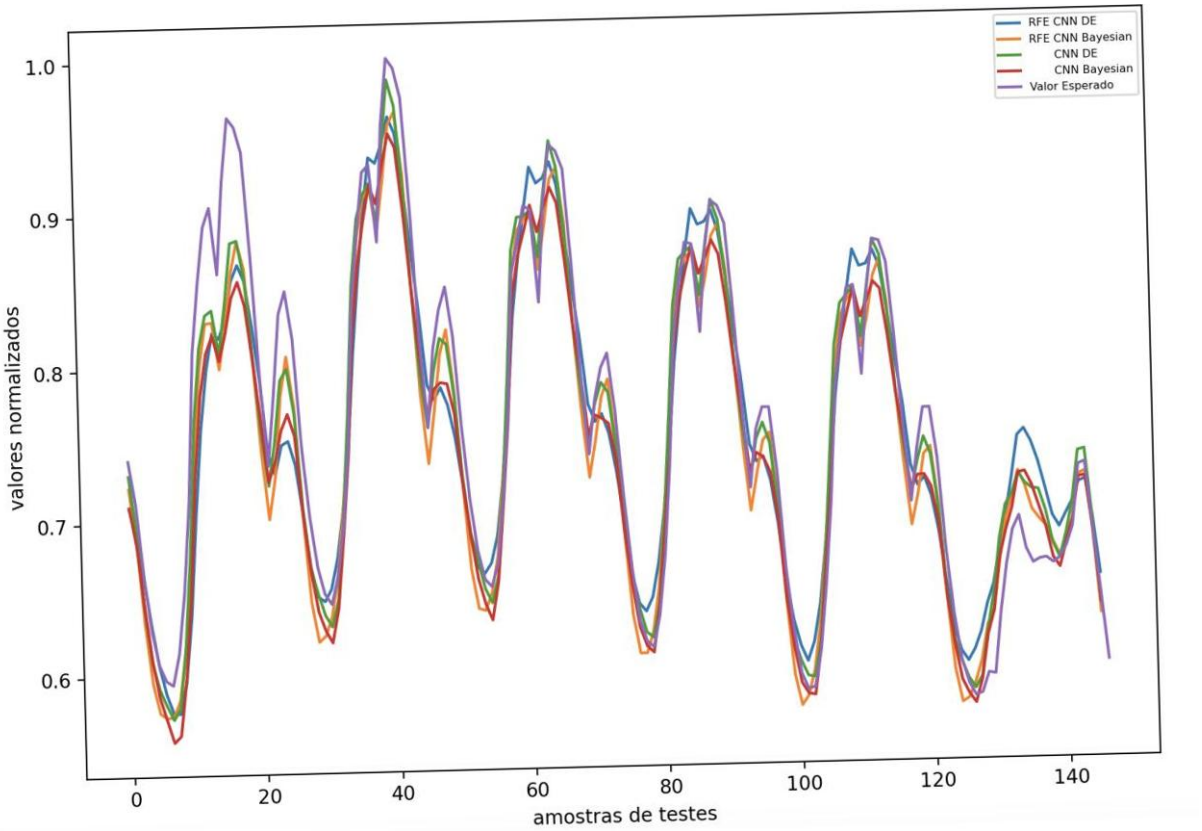

Figura 6.8: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão CNN

O modelo DRVFL mostra na Figura 6.9 que também teve uma boa previsão, mostrando quase uma sobreposição ao longo de toda a série. Entre os otimizadores, nota-se uma pequena piora para o otimizador Bayesiano, quando existe um novo começo de sazonalidade na série.

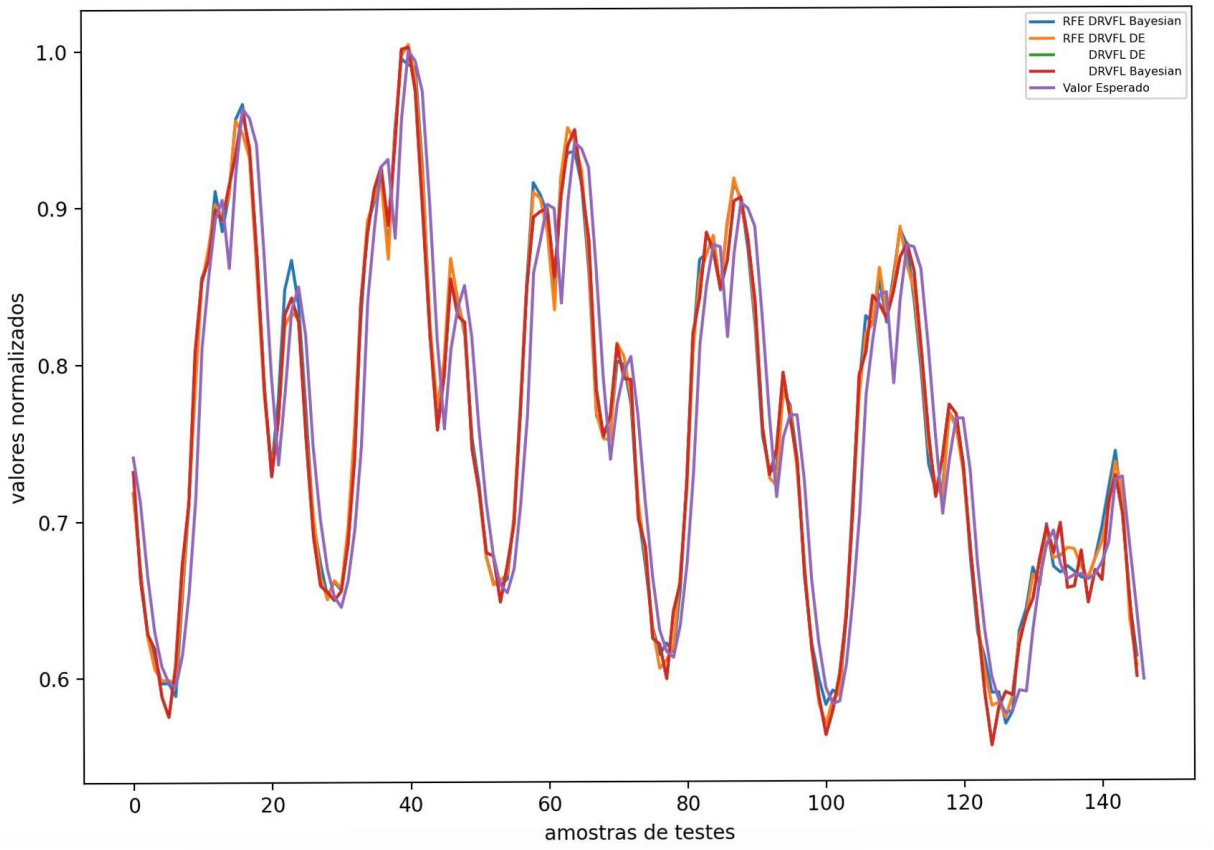

Figura 6.9: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão DVRFL

Os valores de hiperparâmetros encontrados pelos otimizadores podem ser encontrados na Tabela 6.5 que mostra valores altos para o número de neurônios, o que contribuiu para uma melhor otimização para o modelo. Ao se verificar a métrica de desempenho RMSLE, na Tabela 6.2, os valores variam entre $6{,}02e^{-3}$ e $6{,}19e^{-3}$, sendo que uma a opção de RFE com otimizador DE é o melhor resultado obtido pela métrica para a previsão de série do consumo do Sul do Brasil.

modelo DVRFL	neurônios camada de entrada	neurônios camada 1	neurônios camada 2	neurônios camada saída
Com RFE - DE	76	21	35	68
Com RFE- Bayesiana	91	59	79	84
Sem RFE – DE	99	96	98	36
Sem RFE - Bayesiana	99	64	78	7

Tabela 6.5: Valores de hiperparâmetros para o DVRFL com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Outro modelo, o ELM, como pode-se ser visto na Tabela 6.6, teve seus valores na camada oculta também próximos aos valores máximos permitidos, o que mostra que a série não causou sobreajuste durante a fase de treinamento. Outro ponto importante é que se nota que os valores poderiam ser ainda maiores, podendo-se ter aumentado os valores máximos para a quantidade de camadas ocultas.

modelo ELM	Núm. de camadas ocultas	alpha	Tamanho do rbf	Função ativação
Com RFE - DE	95	0,75924	0,76323	tanh
Com RFE- Bayesiana	91	0,97742	0,34296	inv_multiquadric
Sem RFE – DE	94	0,8	0,3	sigmoid
Sem RFE - Bayesiana	93	1,0	1,0	inv_multiquadric

Tabela 6.6: Valores de hiperparâmetros para o DVRFL com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Na Figura 6.10, pode-se verificar uma boa previsão para ambos os otimizadores, porém o algoritmo Bayesiano tem uma piora em prever a frequência da série. Para os métrica de desempenho RMSLE, os valores variam entre $7{,}22e^{-3}$ e $8{,}32e^{-3}$ o que mostra uma acurácia menor do que o notado no modelo DVRFL.

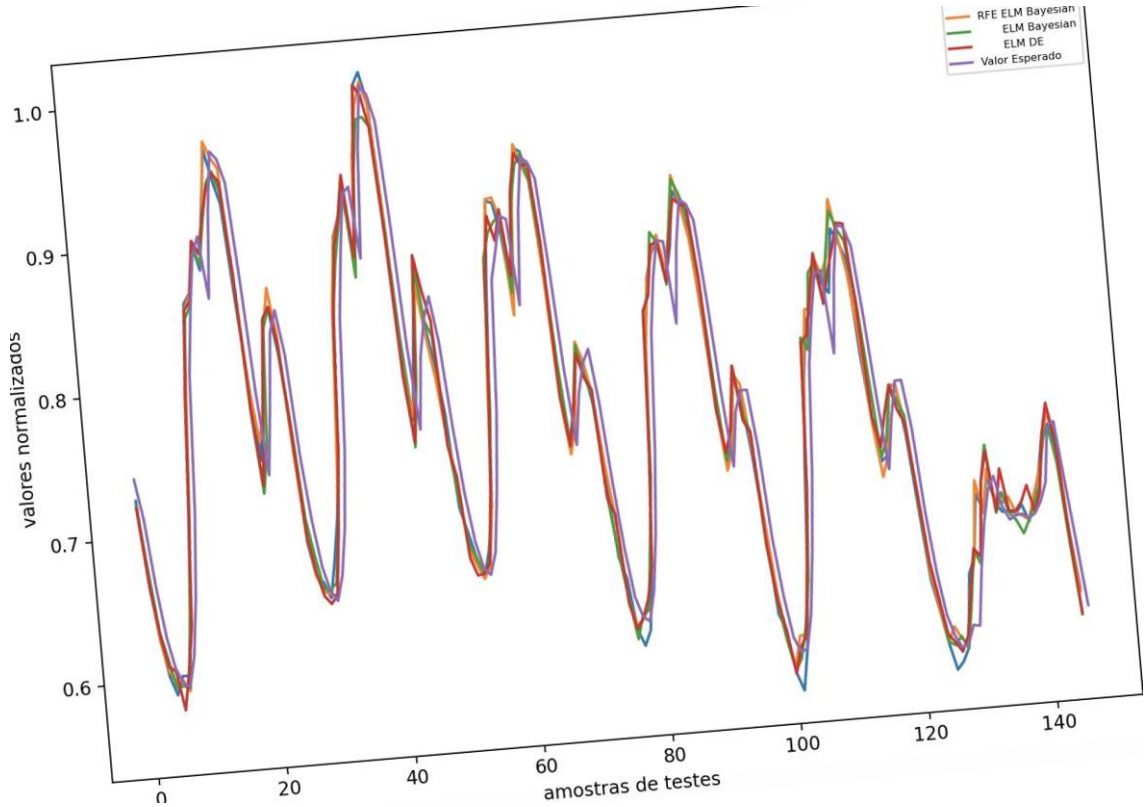

Figura 6.10: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão ELM

Verificando-se outro modelo, o ESN, temos na Tabela 6.7 os valores de seus hiperparâmetros mostrando uma variação grande, causada principalmente pela quantidade de hiperparâmetros que foram verificados.

modelo ESN	*Spectral radius*	Num. receptor	*Input Scaling*	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Com RFE - DE	0,73040	90	0,76892	0,12213	0,77701	0,34473	0
Com RFE-Bayesiana	0,82734	91	0,96573	0,38041	0,52573	0,37595	0
Sem RFE – DE	0,56287	98	0,88921	0,07288	0,29940	0,41741	0
Sem RFE - Bayesiana	0,64904	83	1,0	0,0	0,29940	-0,3549	0

Tabela 6.7: Valores de hiperparâmetros para o ESN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

A Figura 6.11, mostra os valores obtidos por ESN e, como esperado pelos valores de RMSLE que variou entre 6,21e^{-3} e 7,18e^{-3}, os valores de hiperparâmetros, acabaram não sendo otimizados como esperado, possivelmente por ser uma grande quantidade de hiperparâmetros que precisaria ser otimizado.

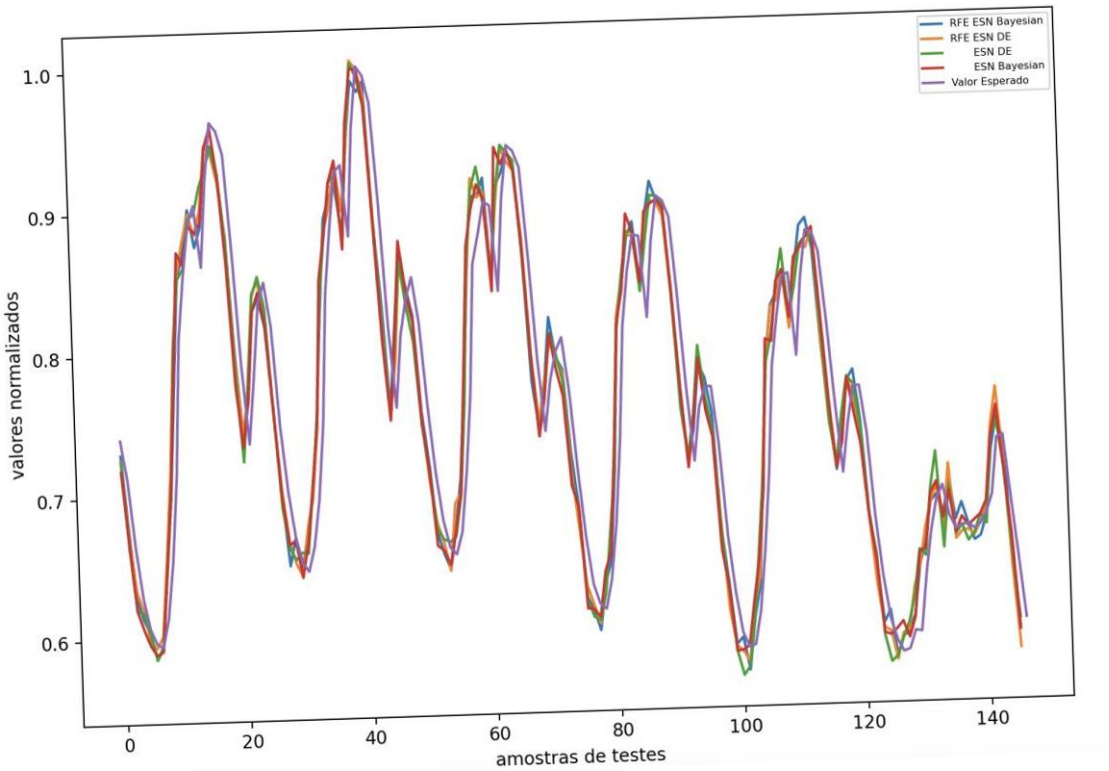

Figura 6.11: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão ESN

No caso do algoritmo FFNN, apenas um parâmetro foi otimizado, porém com uma variação entre 100 e 1000, os valores variaram bastante, conforme mostrado na Tabela 6.8.

modelo FFNN	Número de camadas ocultas
Com RFE - DE	684
Com RFE-Bayesiana	955
Sem RFE – DE	864
Sem RFE - Bayesiana	763

Tabela 6.8: Valores de hiperparâmetros para o FFNN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Mesmo assim, a métrica RMSLE pode ser considerada boa, uma vez que variou entre $1{,}1e^{-2}$ e $1{,}33e^{-2}$, o que acaba sendo notado na Figura 6.12, que mostra os locais da previsão que acabaram não seguindo os valores esperados.

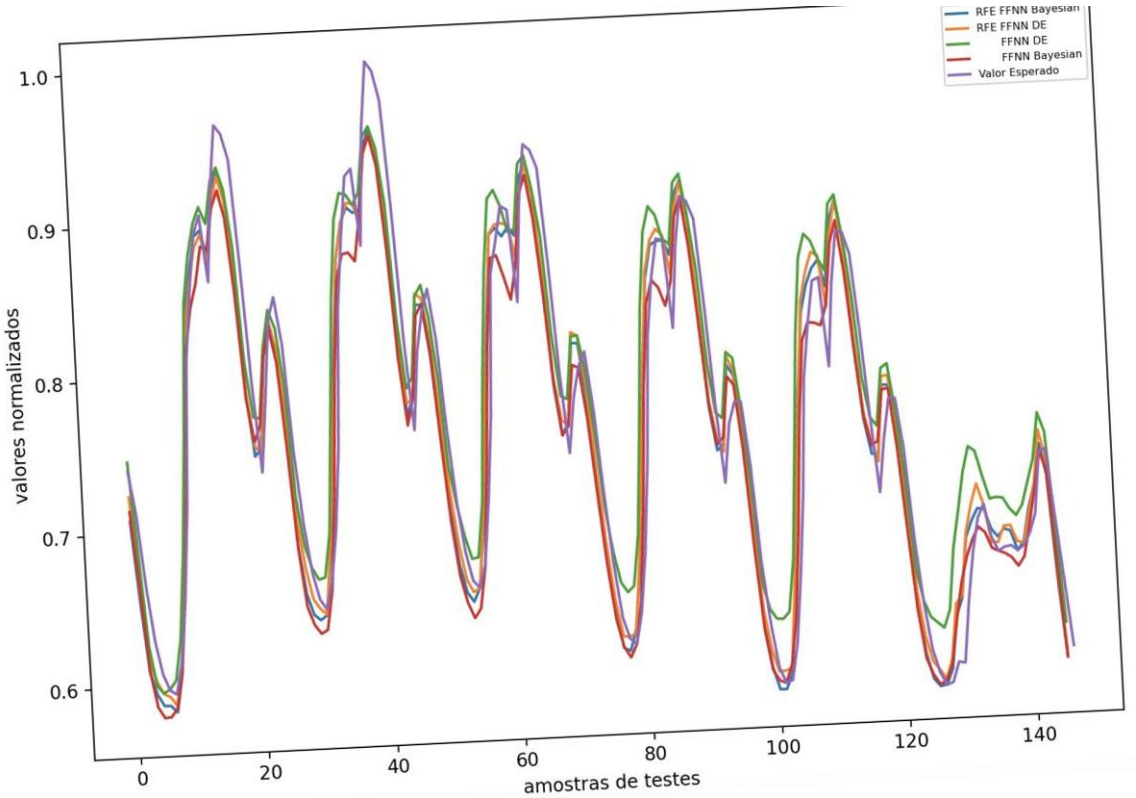

Figura 6.12: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão FFNN

Na sequência, foi verificado o LightGBM, que apresentou um RMSLE que variou entre $1{,}19e^{-2}$ e $1{,}22e^{-2}$ mostrando que a iteração entre os hiperparâmetros acabou não mostrando diferenças significativas como em outros algoritmos, mas ainda assim mostrando uma melhora. A Tabela 6.9 mostra os valores otimizados para o algoritmo que, no caso do ESN acabou tendo vários parâmetros para serem otimizados, notando-se que em todos o algoritmo para *boosting* foi o mesmo, usando o *gradient boosting decision trees* e o *learning rate* também muitos parecidos. Por ter valores de RMSLE muito próximos, nota-se que a profundidade máxima acabou não afetando os valores como esperado.

modelo LightGBM	Max. *depth*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Com RFE - DE	7	0,206477	29	gbdt	9	0,992357
Com RFE-Bayesiana	83	0,265536	18	gbdt	4	0,981621
Sem RFE – DE	52	0,185366	35	gbdt	1	0,723425
Sem RFE - Bayesiana	82	0,209297	18	gbdt	2	0,829098

Tabela 6.9: Valores de hiperparâmetros para o LightGBM com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Pode se verificar que as varrições quase não existiram ao se analisar a Figura 6.13, que mostra que os dados foram muitos próximos, variando apenas no final da previsão.

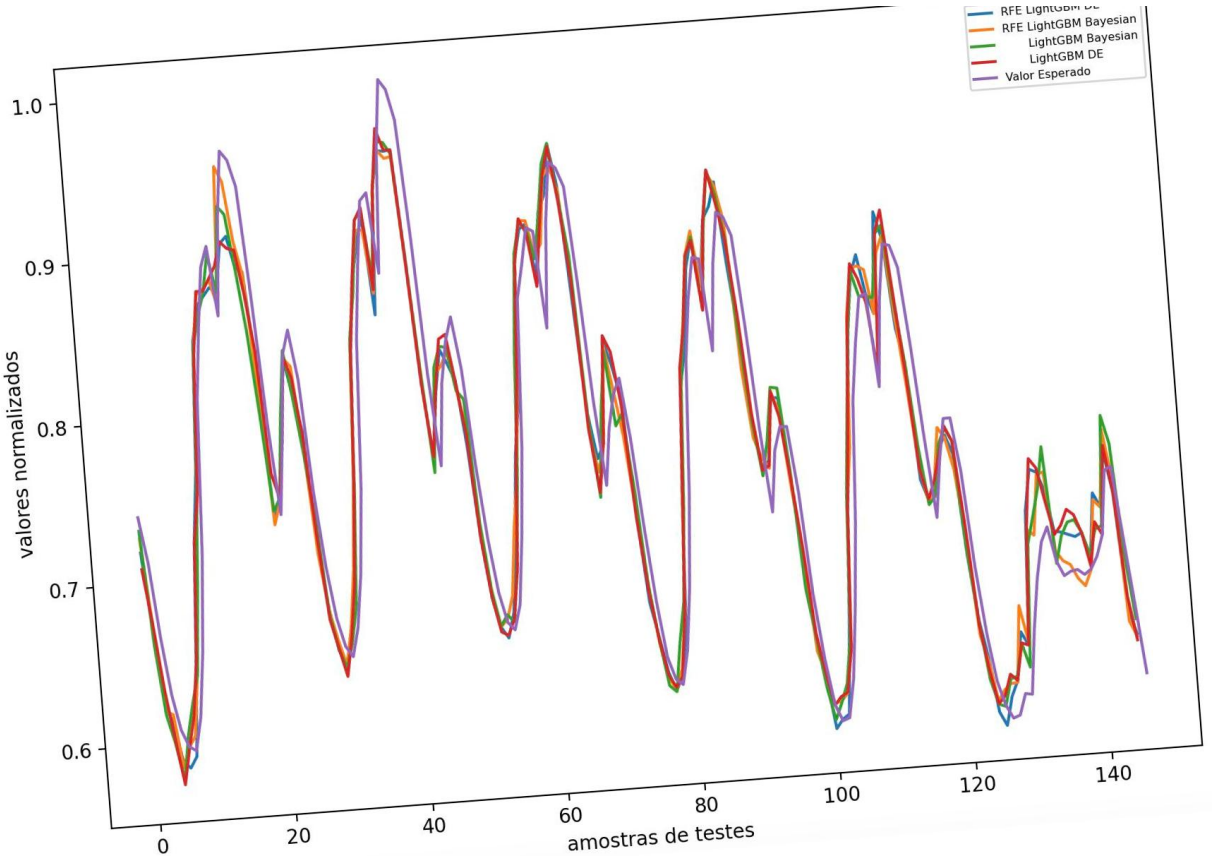

Figura 6.13: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão LightGBM

Próximo na lista, temos o LSTM que acabou tendo a mesma configuração de parâmetros que o FFNN, que é apenas o número de camadas ocultas, conforme mostrado na Tabela 6.10.

modelo LSTM	Número de camadas ocultas
Com RFE - DE	91
Com RFE-Bayesiana	88
Sem RFE – DE	91
Sem RFE - Bayesiana	94

Tabela 6.10: Valores de hiperparâmetros para o LSTM com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Já a Figura 6.14, mostra que a previsão do LSTM acabou correspondendo aos valores da métrica do RMSLE que variou entre $3{,}18e^{-2}$ e $3{,}55e^{-2}$ mostrando que em

vários momentos ao longo da previsão os valores previstos não se mostram perto do valor esperado.

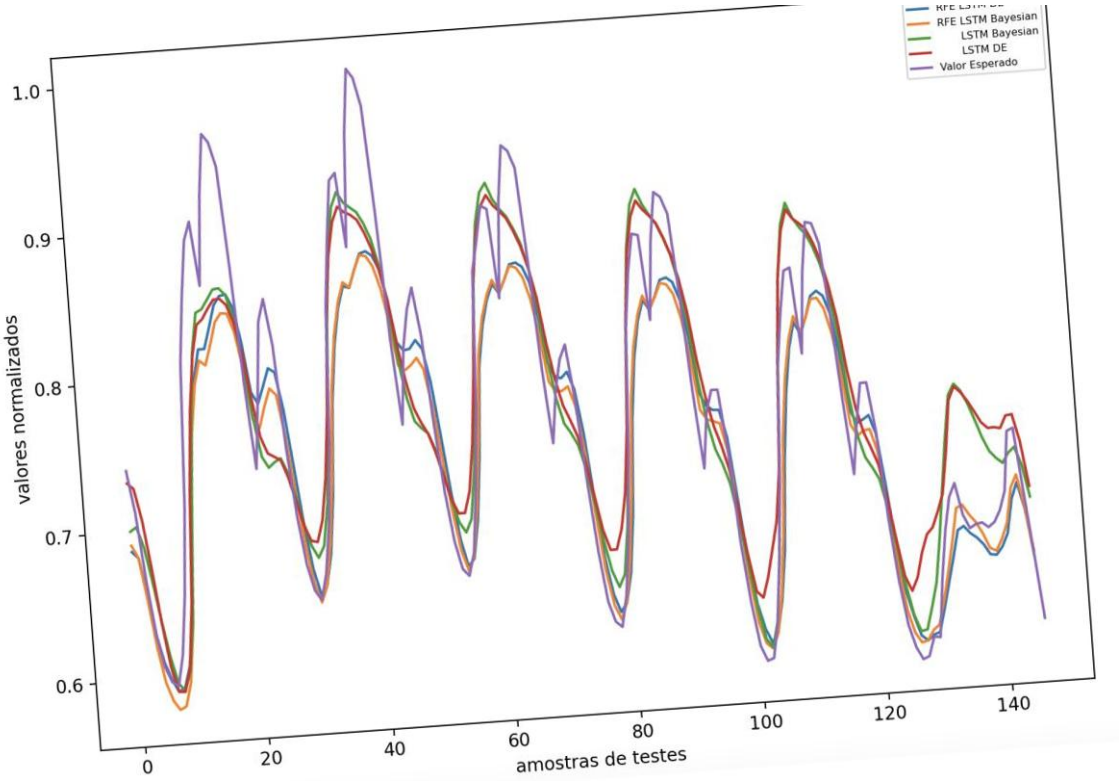

Figura 6.14: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão LSTM

Um dos algoritmos mais rápidos, conforme explicado mais abaixo, foi o próximo a ser executado. O OLS que contém apenas dois valores de hiperparâmetros para serem otimizados, tem os seus valores otimizados mostrado na Tabela 6.11 que apenas definiu se os valores de *y* devem começar em 0 ou não e se *threads* devem ser usadas, ajudando o algoritmo a ser mais rápido. Por isso que os valores acabaram não sendo modificados, porque acabam não afetando os valores encontrados.

modelo OLS	Uso dos valores interceptados?	Número de *jobs*
Com RFE - DE	Não	36
Com RFE-Bayesiana	Não	95
Sem RFE – DE	Não	36
Sem RFE - Bayesiana	Não	95

Tabela 6.11: Valores de hiperparâmetros para o OLS com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Neste caso, os valores da métrica RMSLE acabaram não mudando entre os otimizadores, apenas entre o uso ou não de RFE, mostrando que no caso do uso do RFE acabou piorando os resultados, com valor de $1{,}55e^{-2}$ enquanto os valores sem RFE foram de $1{,}07e^{-2}$, o que acaba sendo notado na Figura 6.15 que mostra pouca variação entre os resultados.

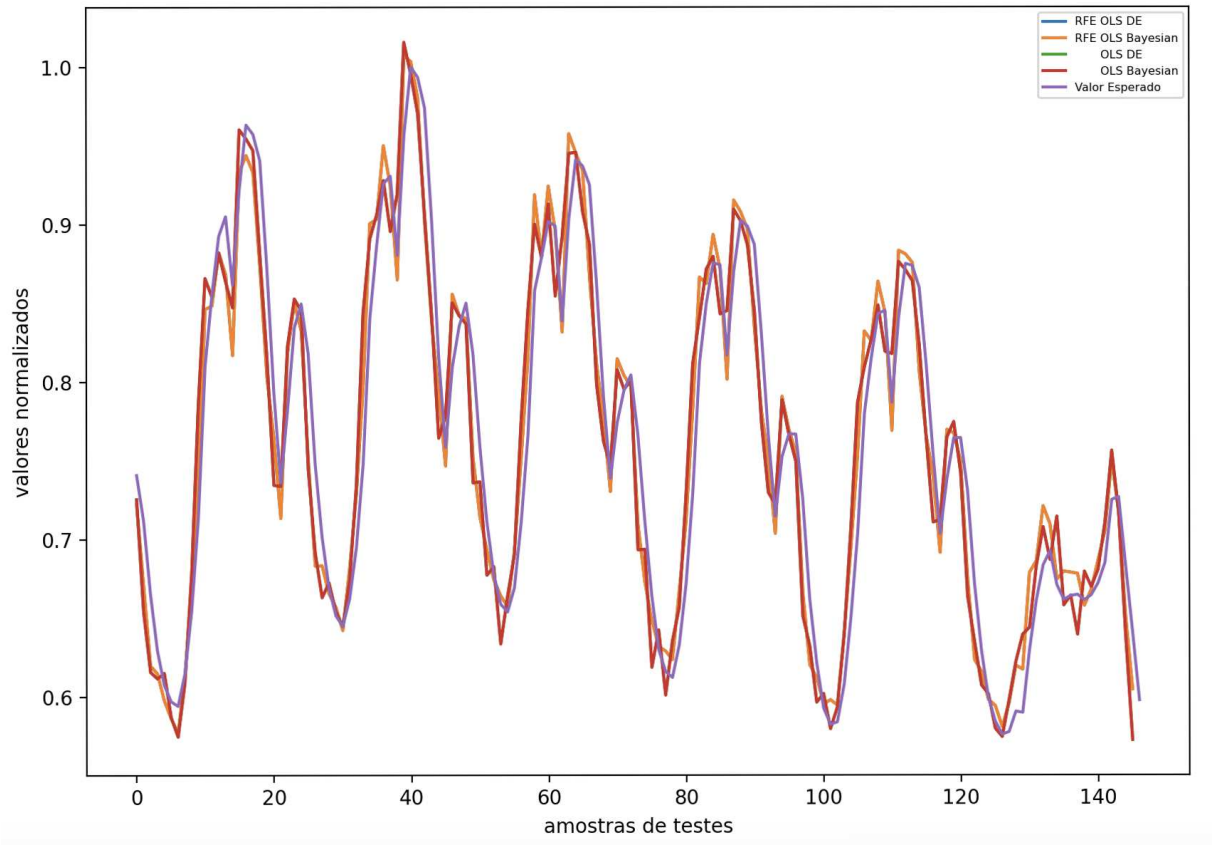

Figura 6.15: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão OLS

O QRF, por sua vez, teve os valores da métrica do RMSLE que variou entre $1{,}99e^{-2}$ e $2{,}00e^{-2}$, o que acaba mostrando uma estabilidade nos valores otimizados na Tabela 6.12, sendo que os valores dos pesos, mesmo sendo bastante diferentes, não são tão relevantes ao algoritmo como os outros parâmetros.

modelo QRF	Número Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Com RFE - DE	28	20	5	2	0,02890	8
Com RFE- Bayesiana	165	23	8	4	0,024303	9
Sem RFE – DE	27	24	5	4	0,00680	9
Sem RFE - Bayesiana	135	33	6	4	0,022003	7

Tabela 6.12: Valores de hiperparâmetros para o QRF com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Na Figura 6.16, é possível de se ver que os resultados entre todos os modelos acabaram sendo parecidos, o que se era esperado por causa dos valores obtidos pelo RMSLE.

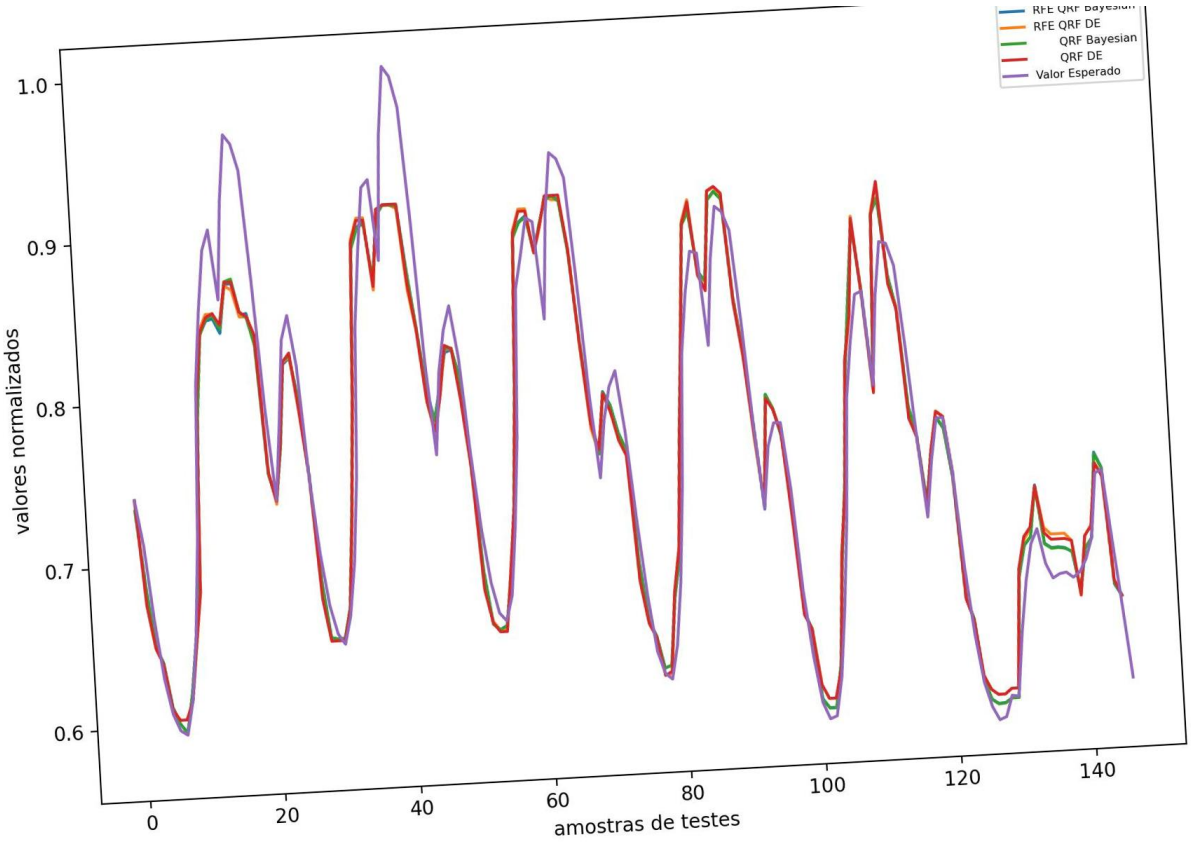

Figura 6.16: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão QRF

Já para o modelo similar RF, os mesmos tipos de hiperparâmetros acabaram sendo otimizados, uma vez que a biblioteca *sklearn* é a mesma entre ambos os algoritmos. Na Figura 6.17, nota-se uma similaridade grande entre QRF e RF, sendo que os valores da métrica mostraram-se um pouco pior para o RF, ficando 2,00e^{-2} para todos os casos.

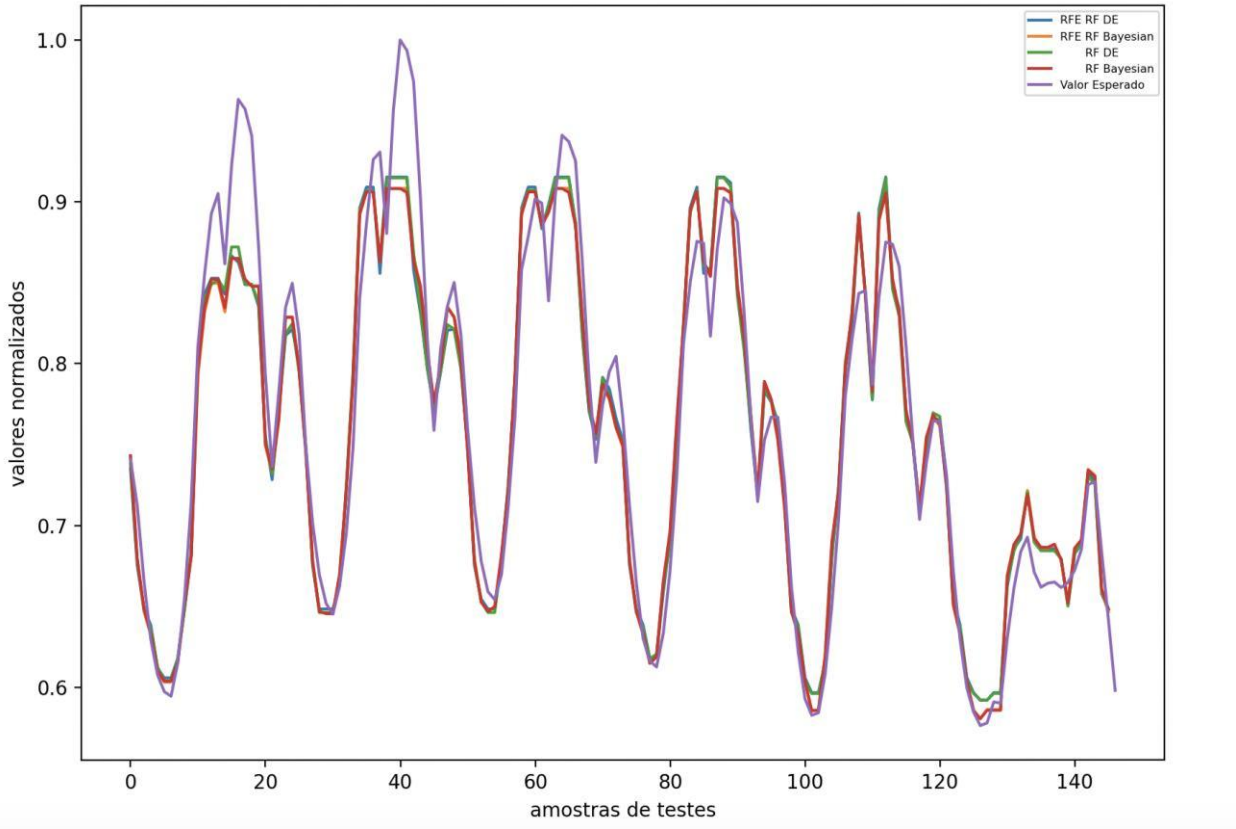

Figura 6.17: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão RF

Já a Tabela 6.13 mostra que houve uma variação diferente de QRF, o que acaba mostrando os valores de reposta um pouco diferente, mas muito similares.

modelo RF	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Com RFE - DE	19	25	7	5	0,00931	9
Com RFE-Bayesiana	35	81	2	1	0,075025	8
Sem RFE – DE	22	92	4	6	0,00667	9
Sem RFE - Bayesiana	24	76	4	1	0,077164	9

Tabela 6.13: Valores de hiperparâmetros para o RF com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

O SARIMA teve desempenho melhor que o seu modelo não-sazonal ARIMA, como se era esperado. Sua métrica de desempenho variou entre $1{,}09e^{-2}$ e $1{,}25e^{-2}$, bem abaixo do que o ARIMA, que teve o pior dos desempenhos. Seus resultados de previsão podem ser vistos na Figura 6.18.

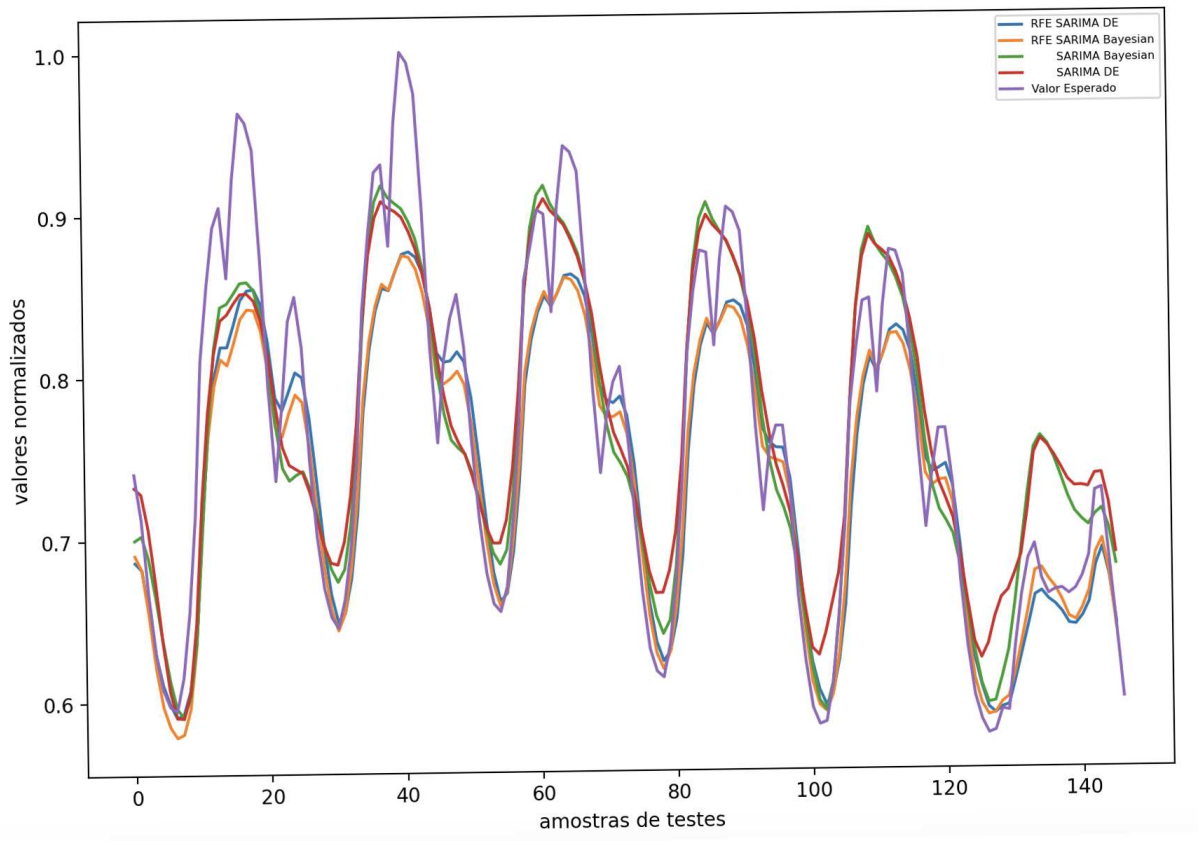

Figura 6.18: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão SARIMA

Já a Tabela 6.14 mostra que como o ARIMA, o SARIMA tem os 3 valores *p, d* e *q*, mas, para poder considerar a sazonalidade, 4 novos parâmetros também são otimizados.

modelo SARIMA	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Com RFE - DE	2	1	2	1	1	1	4
Com RFE-Bayesiana	0	1	2	0	1	0	3
Sem RFE – DE	1	1	1	1	1	1	2
Sem RFE - Bayesiana	0	1	0	2	1	2	4

Tabela 6.14: Valores de hiperparâmetros para o SARIMA com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

Outro modelo, o SVR, mostrou que ao se mudar os valores de hiperparâmetros melhorou-se a métrica de desempenho RMSLE, como é possível de se notar na Tabela 6.15, que mostra uma grande mudança entre os hiperparâmetros de tolerância, *épsilon* e *gama*, causando uma melhora significativa que variou de $1{,}91e^{-2}$ e $2{,}61e^{-2}$.

modelo SVR	Núm. *Kernels*	*gama*	coeficiente	tolerância	regularizador	*epsilon*
Com RFE - DE	1	0,00390	0,024737	0,005808	0,445063	0,026716
Com RFE-Bayesiana	1	0,01	0,001	0,0001	0,5	0,001
Sem RFE – DE	1	0,00620	0,075042	0,004655	0,487598	0,030244
Sem RFE - Bayesiana	1	0,00997	0,019631	0,008395	0,25797	0,007630

Tabela 6.15: Valores de hiperparâmetros para o SVR com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

A Figura 6.19 mostra os resultados para obtidos para o SVR, que mostra uma variação grande entre os modelos, como esperado por causa do RMSLE.

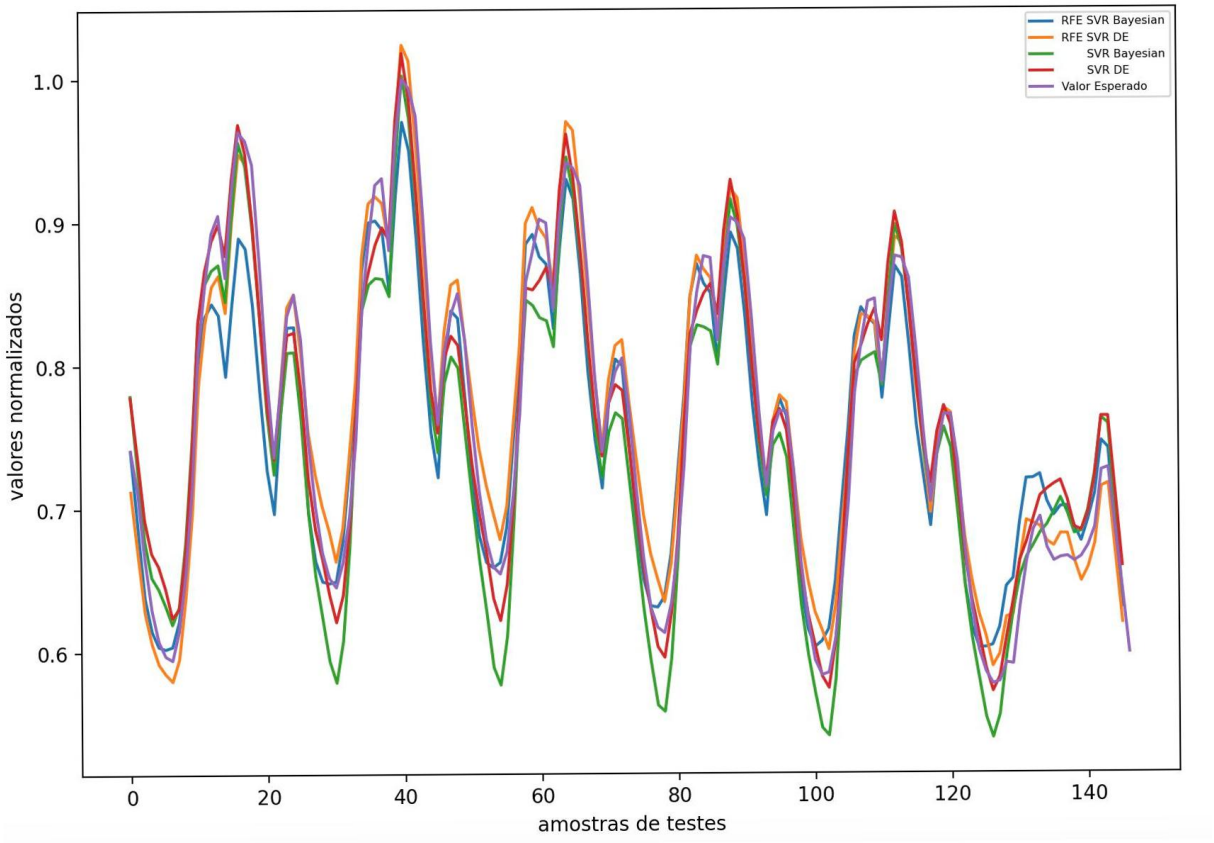

Figura 6.19: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão SVR

Finalmente, para o último modelo testado para a base de dados de previsão da série temporal de consumo da região Sul do Brasil, temos o XGBoost, que tem 5 hiperparâmetros, sendo que a Tabela 6.16 mostra os valores otimizados.

modelo XGboost	Max. *Depth*	Menor peso por *Children*	*Learning rate*	Sub-amostra	Sub-amostra de colunas
Com RFE - DE	44	4	0,215416	0,459857	0,694140
Com RFE-Bayesiana	29	4	0,1	0,582509	0,8
Sem RFE – DE	78	2	0,249159	0,857339	0,795080
Sem RFE - Bayesiana	32	3	0,198432	0,7375466	0,798851

Tabela 6.16: Valores de hiperparâmetros para o XGBoost com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

E na Figura 6.20, é possível mostrado valores próximos do valor esperado, conforme os valores da métrica RMSLE que variou de $1{,}0e^{-2}$ e $1{,}42e^{-2}$.

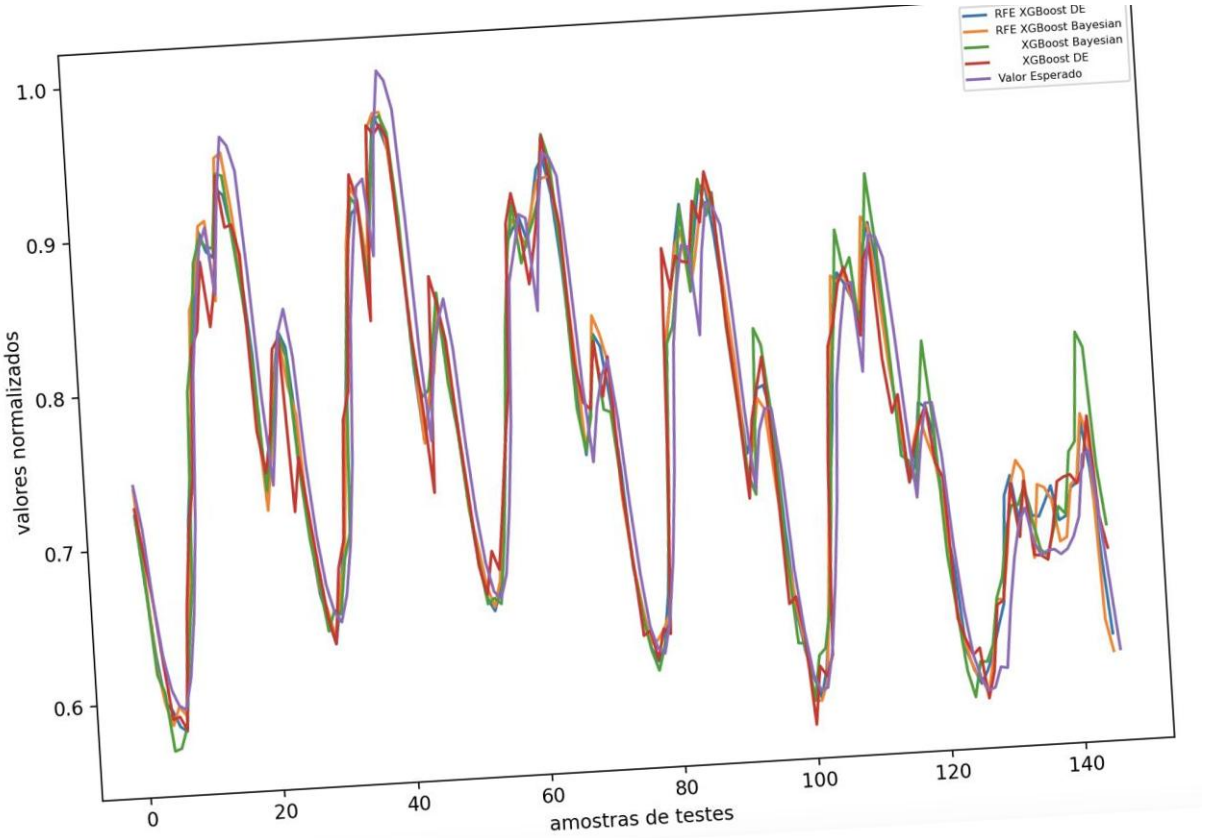

Figura 6.20: Resultados para a previsão da série temporal de consumo da região Sul do Brasil dos modelos/algoritmos de previsão XGBoost

Uma vez terminado a validação e execução dos dados da série temporal de consumo da região Sul do Brasil, a mesma sequência de algoritmos e modelos foram executados para a base de dados da Polônia. A Tabela 6.17 mostra o desempenho RMSLE de todos os algoritmos para os dados da Polônia. Dessa vez, o melhor desempenho foi obtido com o algoritmo XGBoost usando o RFE. Já sem o uso do RFE, temos o DRVFL com o otimizado Bayesiano, que de modo geral apresentou bons resultados, com ou sem o uso do RFE. E novamente, o pior desempenho foi para o ARIMA, que teve problemas em todos os modelos, mais explicações sobre o problema com o ARIMA pode ser encontrado logo abaixo.

algoritmo	Sem RFE		Com RFE	
	Bayesiano	DE	Bayesiano	DE
ARIMA	$6{,}1644e^{-2}$	$6{,}1644e^{-2}$	$1{,}2502e^{-1}$	$1{,}2502e^{-1}$
CNN	$1{,}8094e^{-2}$	$2{,}3114e^{-2}$	$1{,}9195e^{-2}$	$2{,}6104e^{-2}$
DRVFL	$1{,}6904e^{-2}$	$1{,}7658e^{-2}$	$1{,}8211e^{-2}$	$1{,}8251e^{-2}$
ELM	$1{,}8315e^{-2}$	$1{,}7668e^{-2}$	$1{,}8642e^{-2}$	$1{,}8832e^{-2}$
ESN	$1{,}7968e^{-2}$	$1{,}8869e^{-2}$	$1{,}7384e^{-2}$	$1{,}7170e^{-2}$
FFNN	$1{,}7027e^{-2}$	$2{,}6641e^{-2}$	$1{,}8298e^{-2}$	$1{,}7906e^{-2}$
LightGBM	$2{,}0542e^{-2}$	$2{,}0550e^{-2}$	$1{,}8375e^{-2}$	$2{,}1345e^{-2}$
LSTM	$2{,}6536e^{-2}$	$2{,}6841e^{-2}$	$2{,}4001e^{-2}$	$2{,}3300e^{-2}$
OLS	$1{,}7613e^{-2}$	$1{,}7613e^{-2}$	$1{,}8871e^{-2}$	$1{,}8871e^{-2}$
QRF	$2{,}3656e^{-2}$	$2{,}4684e^{-2}$	$2{,}3707e^{-2}$	$2{,}3710e^{-2}$
RF	$2{,}3484e^{-2}$	$2{,}4005e^{-2}$	$2{,}3554e^{-2}$	$2{,}4005e^{-2}$
SARIMA	$2{,}0257e^{-2}$	$2{,}1680e^{-2}$	$2{,}084e^{-2}$	$2{,}1621e^{-2}$
SVR	$6{,}1643e^{-2}$	$6{,}1644e^{-2}$	$6{,}1637e^{-2}$	$6{,}1637e^{-2}$
XGBoost	$1{,}7587e^{-2}$	$1{,}6945e^{-2}$	$2{,}0428e^{-2}$	$1{,}6087e^{-2}$

Tabela 6.17: Comparativo do RMSLE para a série temporal de consumo na Polônia

Começando-se novamente com o modelo ARIMA, a Figura 6.21 mostra os valores com ou sem RFE, usando otimizadores DE e Bayesiano.

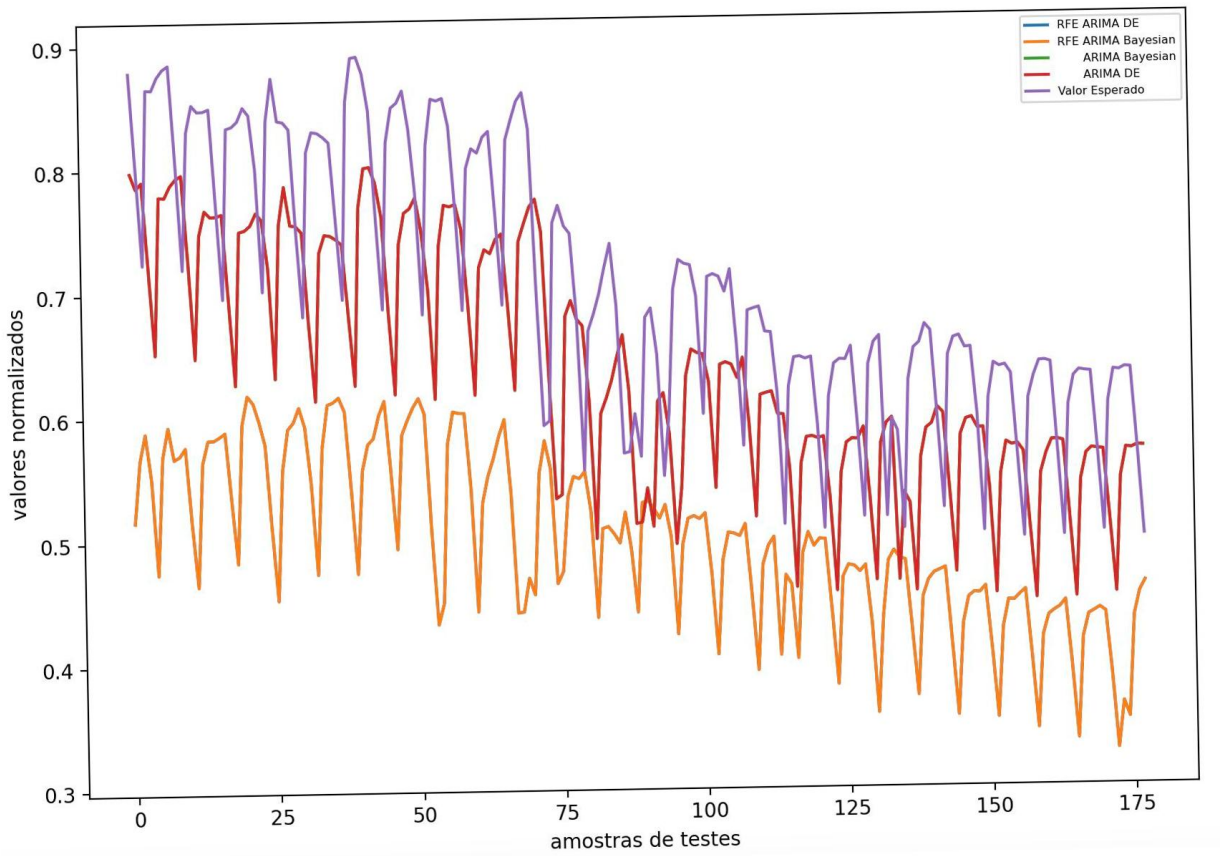

Figura 6.21: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão ARIMA

Como no caso da base de dados brasileira, os valores otimizados de seus hiperparâmetros podem ser encontrados na Tabela 6.18 que foram obtidos pelos otimizadores DE e Bayesiano com ou sem o uso de RFE, mostram que houve pouca variação nos valores.

modelo ARIMA	*P*	*Q*	*D*
Com RFE - DE	2	0	1
Com RFE- Bayesiana	2	1	1
Sem RFE – DE	2	0	1
Sem RFE - Bayesiana	2	1	1

Tabela 6.18: Valores de hiperparâmetros para o ARIMA com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Ao se analisar a Figura 6.21 e valores da métrica RMSLE na Tabela 6.2 percebe-se o mesmo problema que ocorreu para a base de dados brasileira, que não conseguiu prever valores perto do esperado e que os valores de RMSLE acabaram

sendo muito similares causando uma sobreposição dos valores. Isso aconteceu principalmente porque o modelo ARIMA não pode ser usado em casos em que há uma sazonalidade, sendo então usado o modelo SARIMA para esses casos. Nota-se também que o uso de RFE acabou piorando ainda mais tais previsões, pois acabou retirando características que são importantes para o modelo.

Para o algoritmo CNN, na Figura 6.22 nota-se que o sistema acaba não conseguindo prever valores que não seguem uma tendência construída durante a fase de validação, mas acaba conseguindo melhores previsões. A métrica RMSLE ficou entre $1{,}8e^{-2}$ e $2{,}61e^{-2}$.

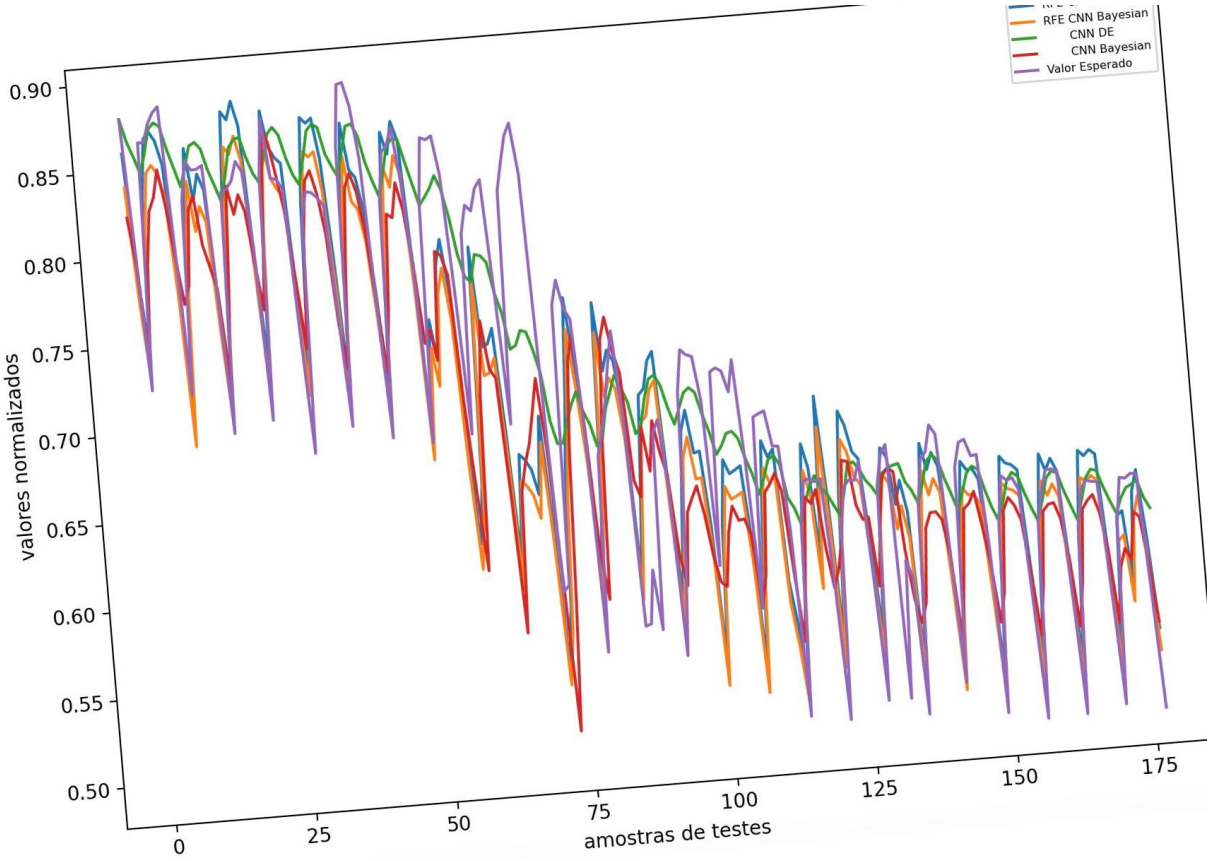

Figura 6.22: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão CNN

Já na Tabela 6.19, nota-se uma variação relativamente grande entre todo o espectro de busca, o que explica a grande variação do RMSLE.

modelo CNN	Convolação inicial	*Max-pooling*	Núm. saídas penúltima camada
Com RFE - DE	77	6	67
Com RFE- Bayesiana	59	1	84
Sem RFE – DE	10	7	56
Sem RFE - Bayesiana	62	5	84

Tabela 6.19: Valores de hiperparâmetros para o CNN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Como notado na Tabela 6.20, os valores dos hiperparâmetros do DVRFL variaram bastante, mas os valores da métrica de performance RMSLE mostra um resultado pouco variante, entre $1{,}6e^{-2}$ e $1{,}8e^{-2}$.

modelo DVRFL	neurônios camada de entrada	neurônios camada 1	neurônios camada 2	neurônios camada saída
Com RFE - DE	19	22	28	75
Com RFE- Bayesiana	2	3	76	14
Sem RFE – DE	12	12	59	65
Sem RFE - Bayesiana	39	100	3	92

Tabela 6.20: Valores de hiperparâmetros para o DVRFL com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Visualmente, a Figura 6.23 mostra que os dados previstos não conseguem valores perto do esperado quando há uma mudança de tendência na série.

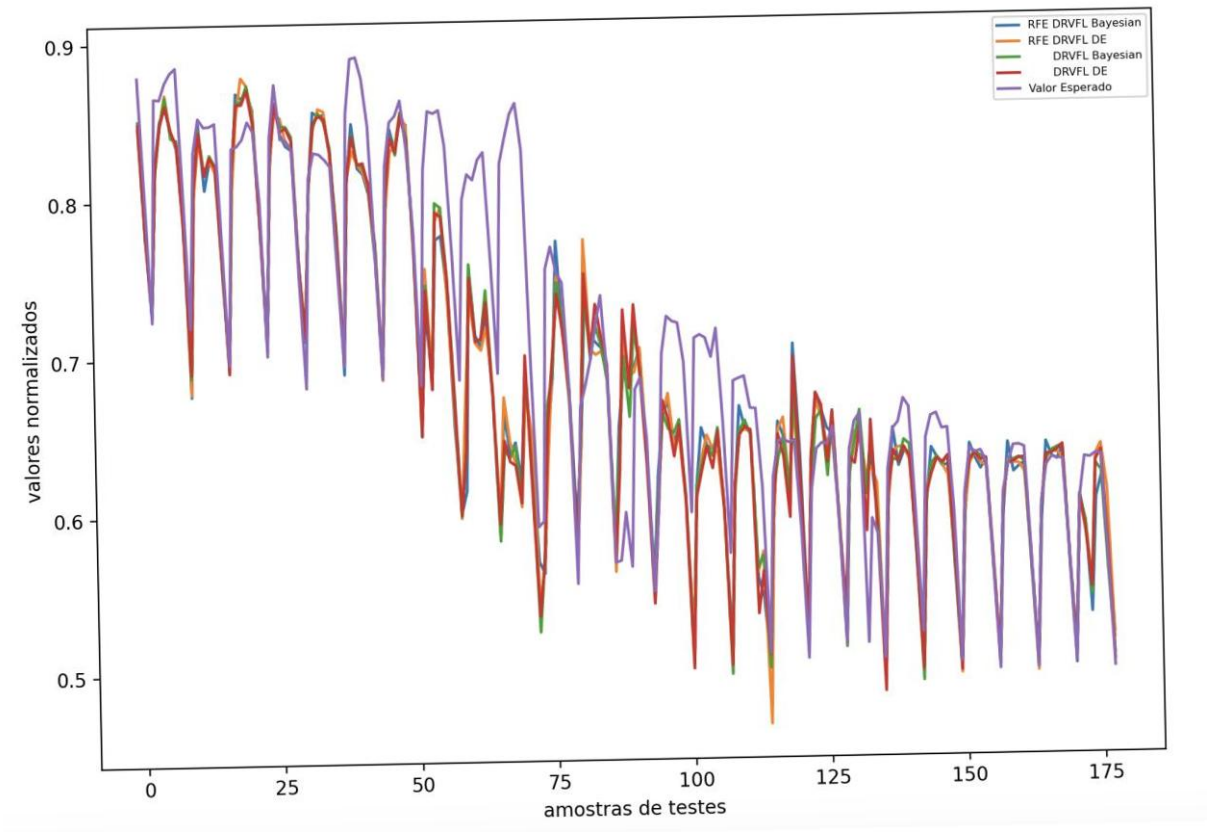

Figura 6.23: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão DRVFL

Para o algoritmo ELM, a métrica de desempenho RMSLE variou entre $1,7e^{-2}$ e $1,8e^{-2}$ o que é significativamente baixo, e que pode ser notado na Figura 6.24 que não mostra uma grande variação dos valores previstos. Novamente, ao se considerar o RMSLE, nota-se que o ELM é inferior em resultado do que o DVRFL.

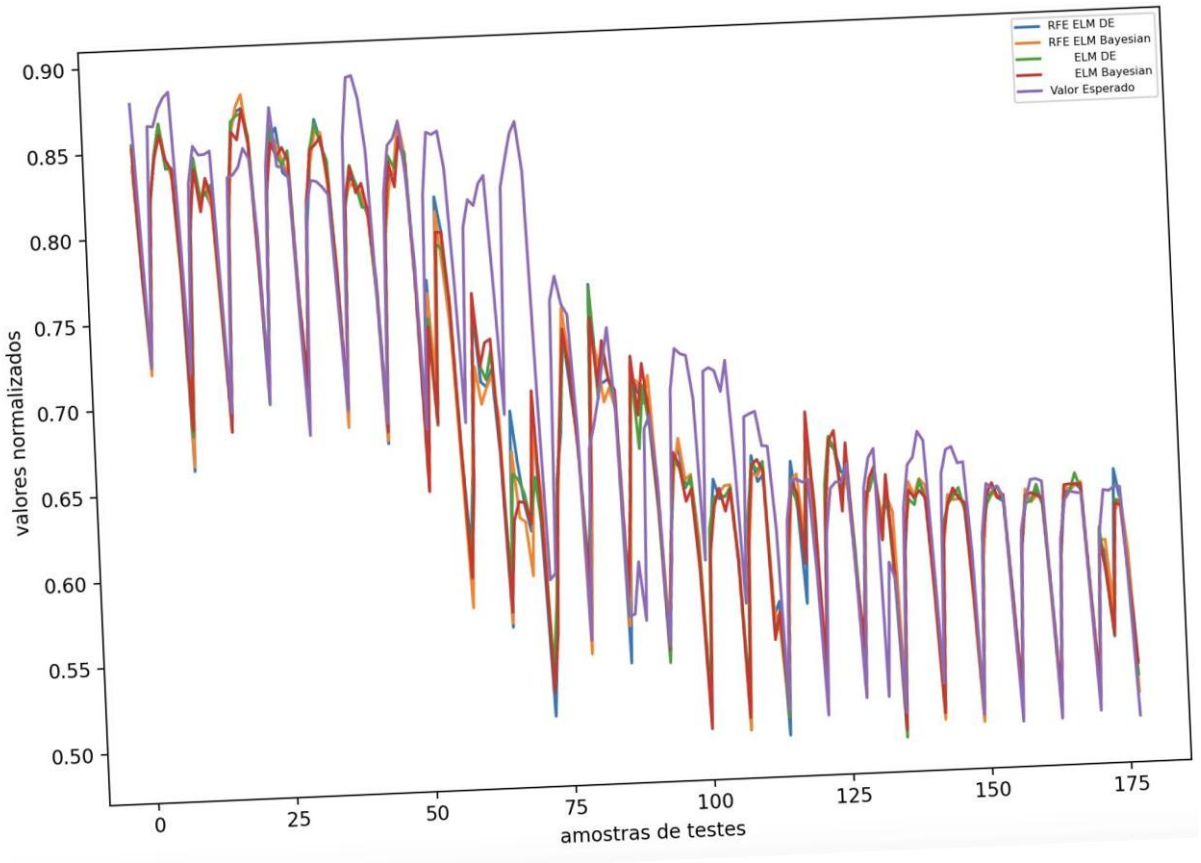

Figura 6.24: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão ELM

Já na Tabela 6.21, encontra-se os valores para os hiperparâmetros do ELM.

modelo ELM	Núm. de camadas ocultas	*alpha*	Tamanho do *rbf*	Função ativação
Com RFE - DE	59	0,716720	0,475565	*sine*
Com RFE- Bayesiana	32	0,8	0,3	*inv_tribas*
Sem RFE – DE	87	0,314449	0,339772	*inv_multiquadric*
Sem RFE - Bayesiana	79	0,497462	0,573864	*inv_multiquadric*

Tabela 6.21: Valores de hiperparâmetros para o DVRFL com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Usando-se o ESN, temos na Tabela 6.22, os valores dos hiperparâmetros que por serem 7, acabam não tendo todo o seu espectro de busca usado, mas acabou não causando uma acentuada variação no RMSLE, que variou entre $1{,}7e^{-2}$ e $1{,}8e^{-2}$.

modelo ESN	*Spectral radius*	Num. receptor	*Input Scaling*	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Com RFE - DE	0,70718	33	0,62819	0,27353	0,77743	0,25054	0
Com RFE-Bayesiana	0,84447	33	0,80565	0,25849	0,96762	0,18077	0
Sem RFE – DE	0,29950	55	0,13222	0,33911	0,64350	0,21138	0
Sem RFE - Bayesiana	0,40366	12	0,02565	0,16223	0,08283	0,34581	1

Tabela 6.22: Valores de hiperparâmetros para o ESN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Na Figura 6.25 pode-se verificar os valores previstos por ESN que mostram uma diferença significativa quando há uma mudança de tendência.

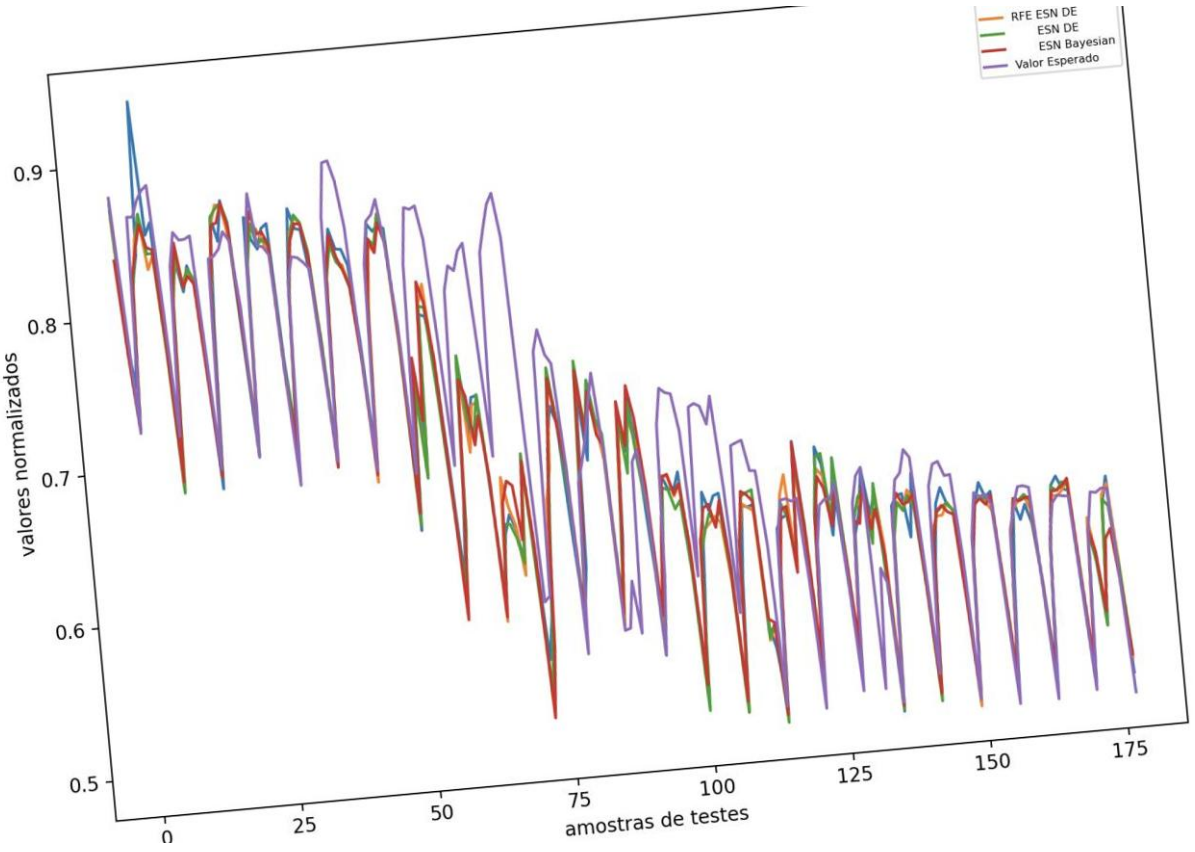

Figura 6.25: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão ESN

Já para o caso do FFNN, novamente tivemos apenas 1 hiperparâmetro para ser otimizado, que pode ser verificado na Tabela 6.23 e que mostra uma grande variação de valores, mas normalmente perto do limite superior de busca.

modelo FFNN	Número de camadas ocultas
Com RFE - DE	586
Com RFE-Bayesiana	941
Sem RFE – DE	824
Sem RFE - Bayesiana	936

Tabela 6.23: Valores de hiperparâmetros para o FFNN com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Já a Figura 6.26 mostra que o modelo não conseguiu prever em vários pontos da série, mesmo em momentos que não existe uma mudança de tendência.

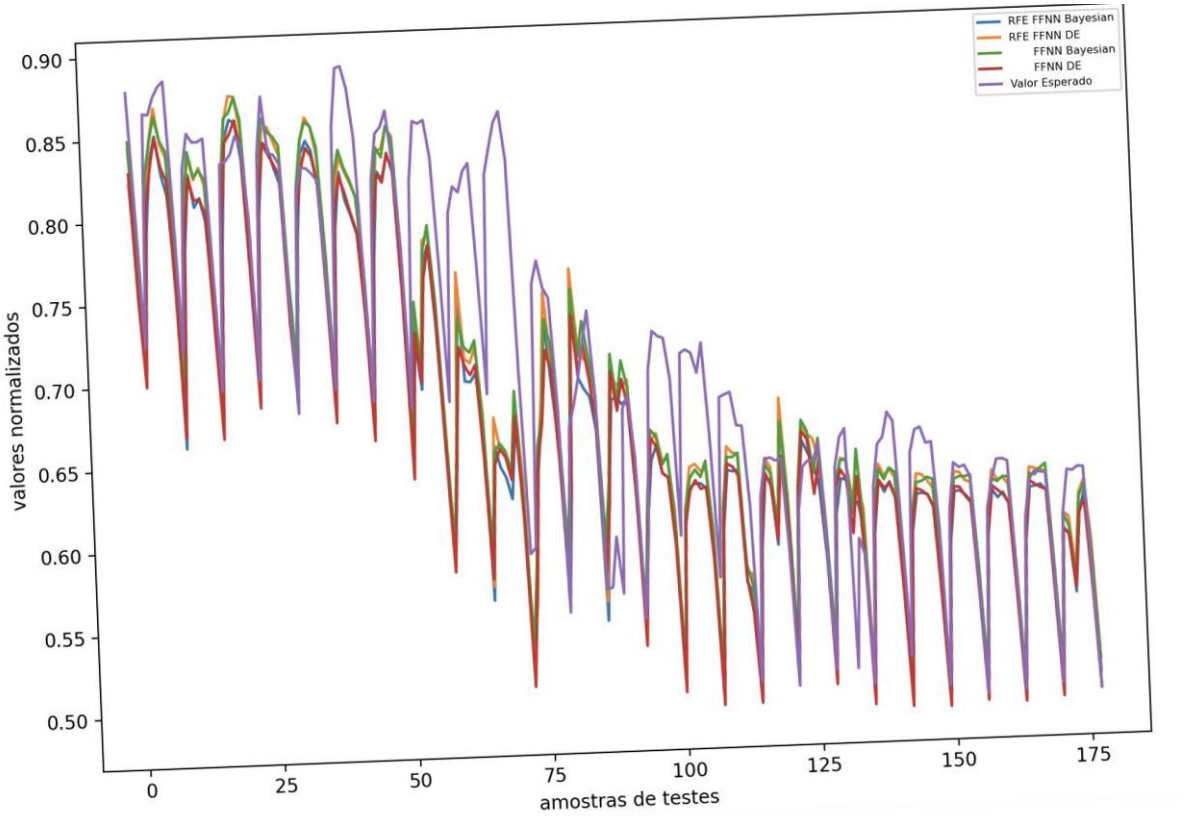

Figura 6.26: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão FFNN

Continuando com a lista de algoritmos, temos o LightGBM, que tem os resultados mostrados na Figura 6.27, mostrando uma variação entre os modelos analisados, principalmente os otimizados por otimizador Bayesiano.

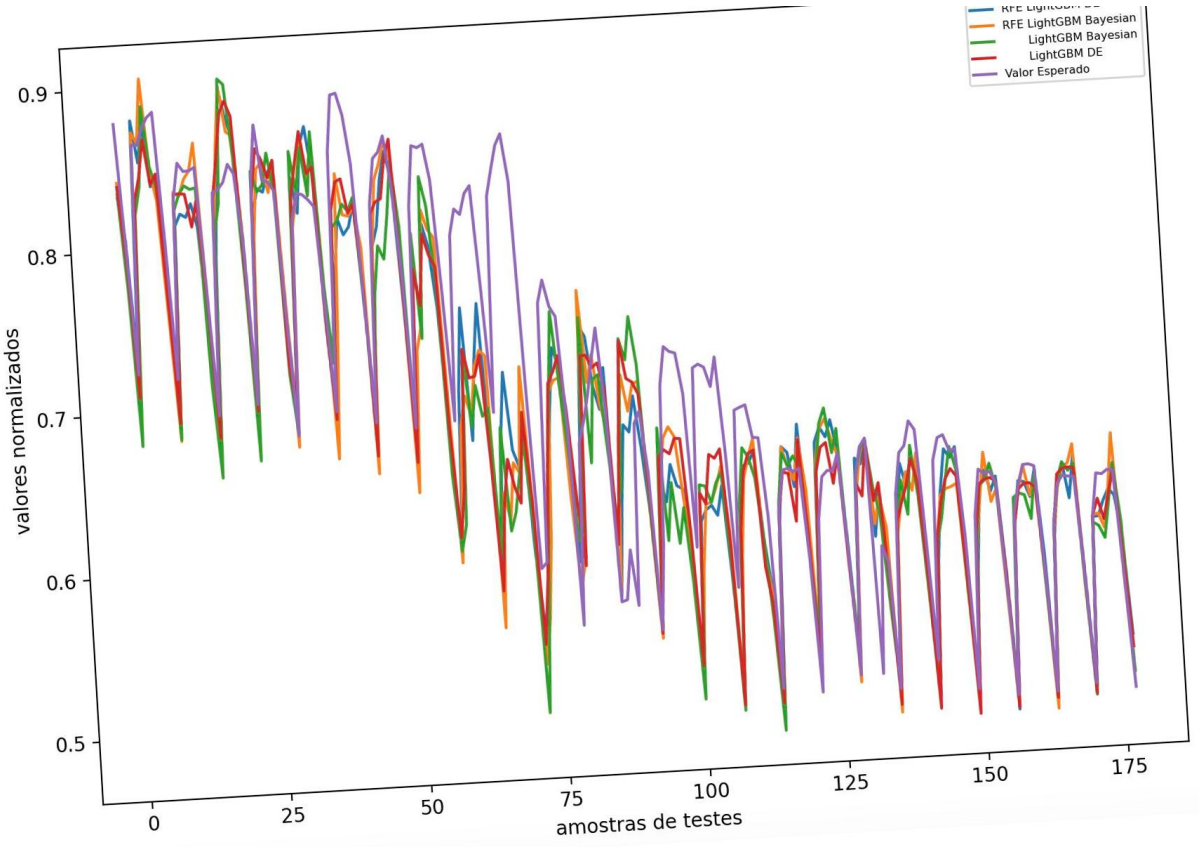

Figura 6.27: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão LightGBM

Já a Tabela 6.24 mostra os valores dos hiperparâmetros, que foram usados para se obter a métrica de desempenho RMSLE que variou entre $1{,}8e^{-2}$ e $2{,}0e^{-2}$. Interessante notar que para a otimização Bayesiana sem RFE, o *boosting* usado for o *dart (dropouts meet multiple additive regression trees)*, que é uma regressão de árvores múltiplas.

modelo LightGBM	Max *depth*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Com RFE - DE	86	0,560418	4	gbdt	4	0,532119
Com RFE-Bayesiana	83	0,212914	19	gbdt	3	0,666804
Sem RFE – DE	56	0,137753	97	gbdt	7	0,383847
Sem RFE - Bayesiana	96	0.589549	6	dart	1	0.869361

Tabela 6.24: Valores de hiperparâmetros para o LightGBM com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Seguindo-se para o próximo algoritmo, rodou-se o LSTM, que como explicado anteriormente, usa a mesma biblioteca sklearn do FFNN, logo, tendo apenas um hiperparâmetro para ser otimizado, que é mostrado na Tabela 6.25. Os valores otimizados foram perto do limite superior do espectro de busca, mostrando que quanto mais camadas, neste caso, foi melhor.

modelo LSTM	Número de camadas ocultas
Com RFE - DE	97
Com RFE-Bayesiana	92
Sem RFE – DE	85
Sem RFE - Bayesiana	96

Tabela 6.25: Valores de hiperparâmetros para o LSTM com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Já os valores da métrica de desempenho variaram entre $2{,}3e^{-2}$ e $2{,}6e^{-2}$, sendo valores bem altos entre todos os modelos, mostrando que houve erros de previsão, que podem ser verificados na Figura 6.28.

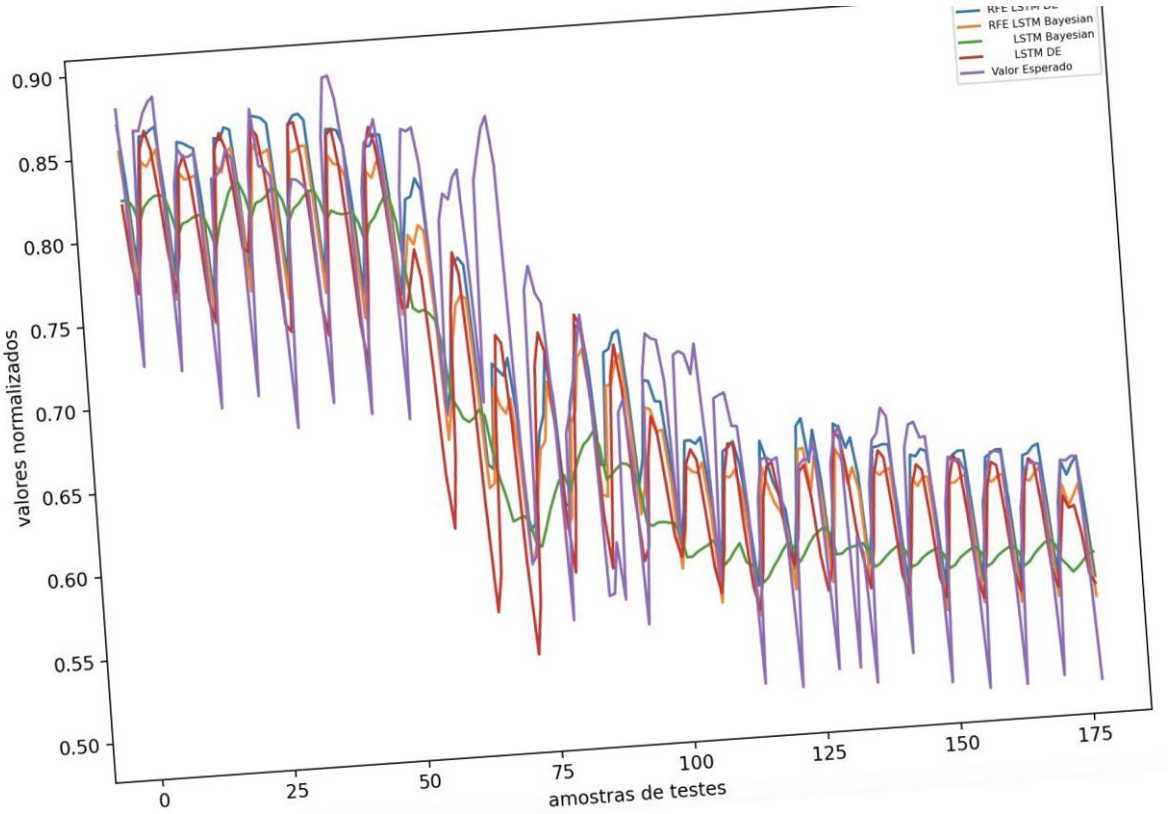

Figura 6.28: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão LSTM

Como explicado anteriormente, os valores de hiperparâmetros para o OLS foram 2, sendo que um apenas melhora o tempo de execução e o outro é para especificar se o algoritmo deve começar os valores de *y* em 0. A Tabela 6.26 mostra que não houve variação entre os otimizadores, a métrica de desempenho RMLSE foi diferente apenas por causa do uso ou não do RFE, ficando em $1{,}7e^{-2}$ e $1{,}8e^{-2}$.

modelo OLS	Uso dos valores interceptados?	Número de *jobs*
Com RFE - DE	Sim	36
Com RFE-Bayesiana	Sim	60
Sem RFE – DE	Sim	36
Sem RFE - Bayesiana	Sim	60

Tabela 6.26: Valores de hiperparâmetros para o OLS com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da região Sul do Brasil

A Figura 6.29 mostra tais resultados, sendo que como são iguais os valores dos otimizadores, eles se sobrepõem.

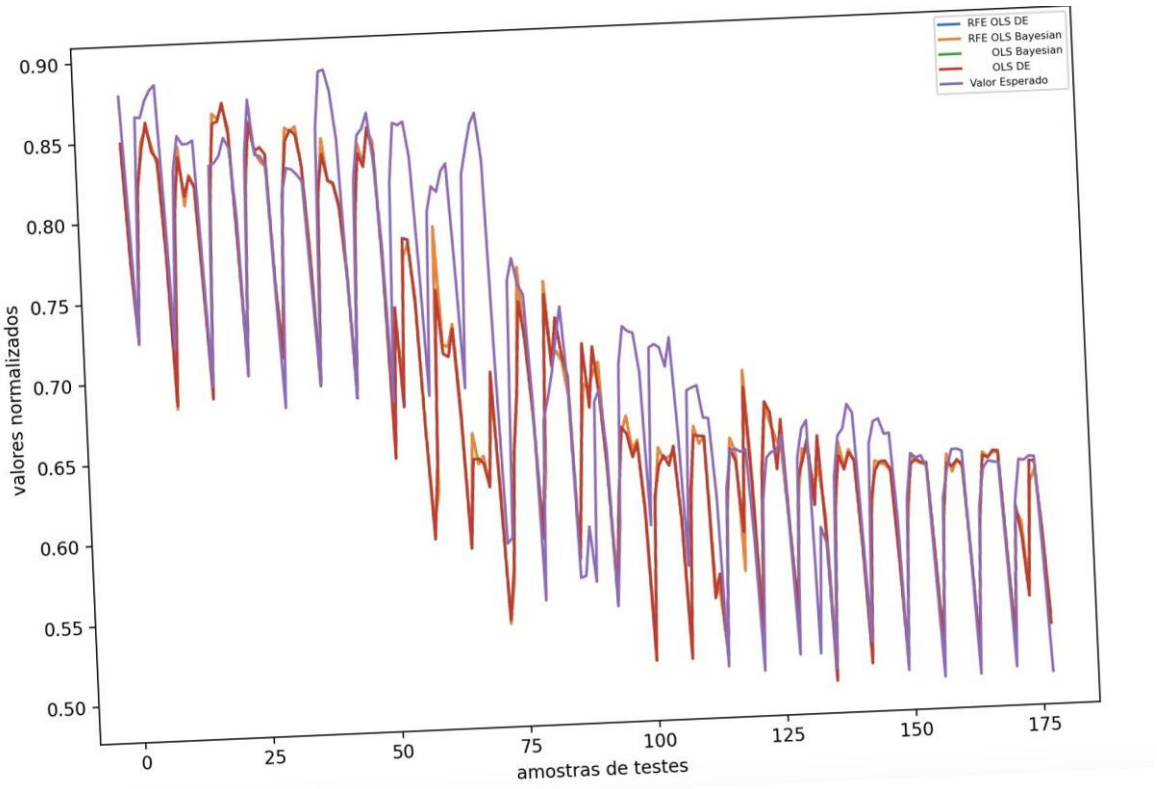

Figura 6.29: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão OLS

Já para o caso de QRF, os 6 possíveis valores de hiperparâmetros acabaram não variando muito, conforme notado na Tabela 6.26 causando uma variação da métrica de desempenho RMSLE que variou entre $2{,}3e^{-2}$ e $2{,}4e^{-2}$ o que pode ser considerado alto entre os modelos analisados.

modelo QRF	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Com RFE - DE	11	62	9	8	0,00680	9
Com RFE- Bayesiana	5	45	7	5	0,03584	8
Sem RFE – DE	8	33	6	9	0,06568	9
Sem RFE - Bayesiana	13	86	6	7	0,010292	9

Tabela 6.27: Valores de hiperparâmetros para o QRF com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

E na Figura 6.30 é possível de se ver os valores previstos pelo modelo.

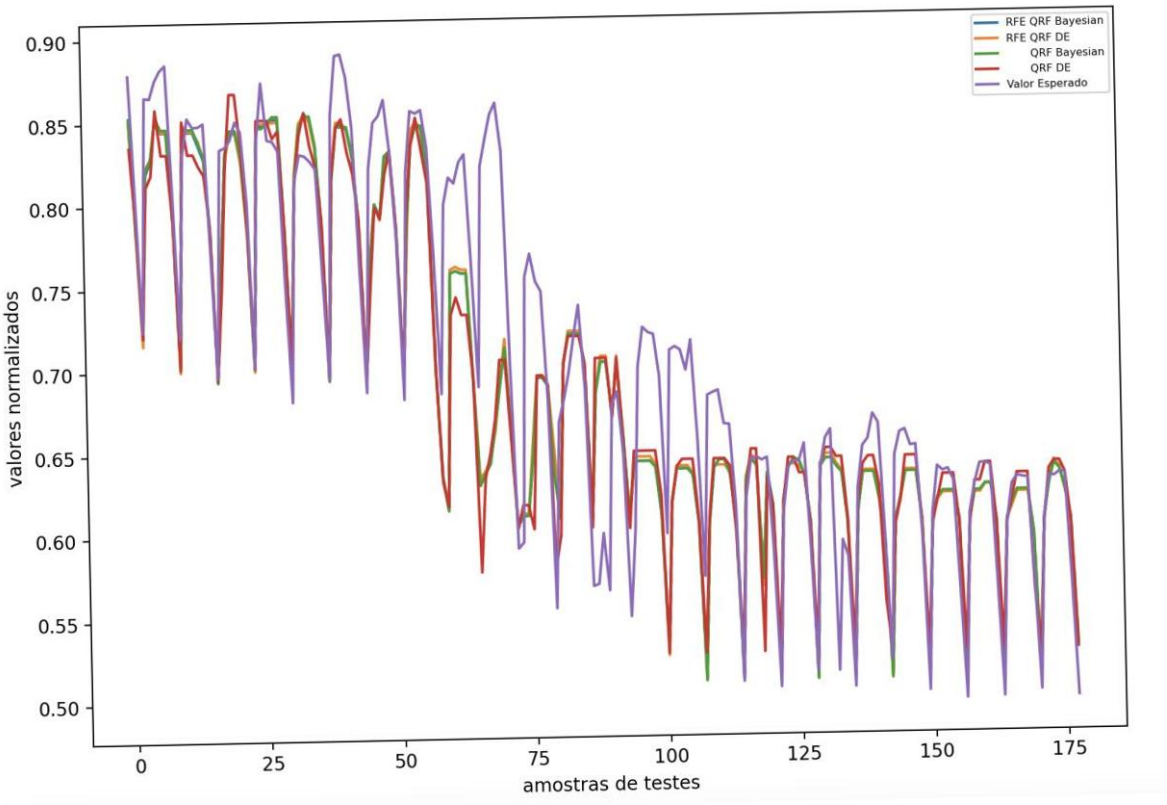

Figura 6.30: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão QRF

Seguindo a mesma estrutura do QRF, temos o RF, que pela proximidade dos algoritmos, acaba tendo os mesmos hiperparâmetros a serem otimizados. A Tabela 6.28 mostra os valores obtidos pela otimização.

modelo RF	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Com RFE - DE	9	50	4	2	0,07582	5
Com RFE-Bayesiana	7	56	8	7	0,010285	8
Sem RFE – DE	7	59	9	5	0,02862	9
Sem RFE - Bayesiana	7	86	6	7	0,010292	9

Tabela 6.28: Valores de hiperparâmetros para o RF com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia.

Já a Figura 6.31, apresenta os valores das previsões, confirmando os valores da métrica de desempenho RMSLE que variaram entre $2{,}3e^{-2}$ e $2{,}4e^{-}2$ o que pode ser considerado estáveis e mostrando uma grande similaridade com os valores encontrados de QRF.

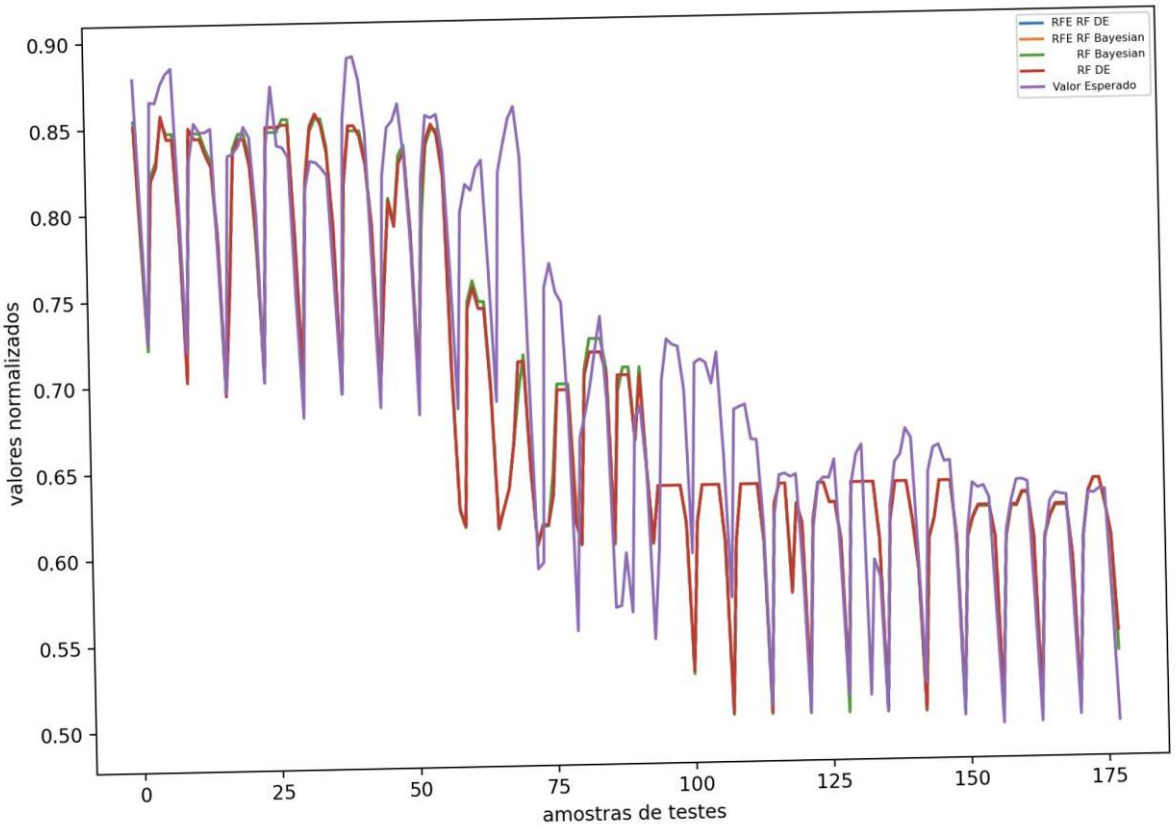

Figura 6.31: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão RF

No caso do algoritmo SARIMA, notou-se que ele funcionou melhor que algoritmo ARIMA, conforme pode ser visto na Figura 6.32 uma vez que ele suporta a sazonalidade que a série apresenta. Neste caso, nota-se ainda uma melhora significativa no caso do uso da do otimizador DE, sendo que os valores da métrica de desempenho RMSLE apresentaram menores valores para os casos que usaram DE ficando em $2e^{-2}$ enquanto no caso de otimização Bayesiana ficou em torno de $2{,}2e^{-2}$.

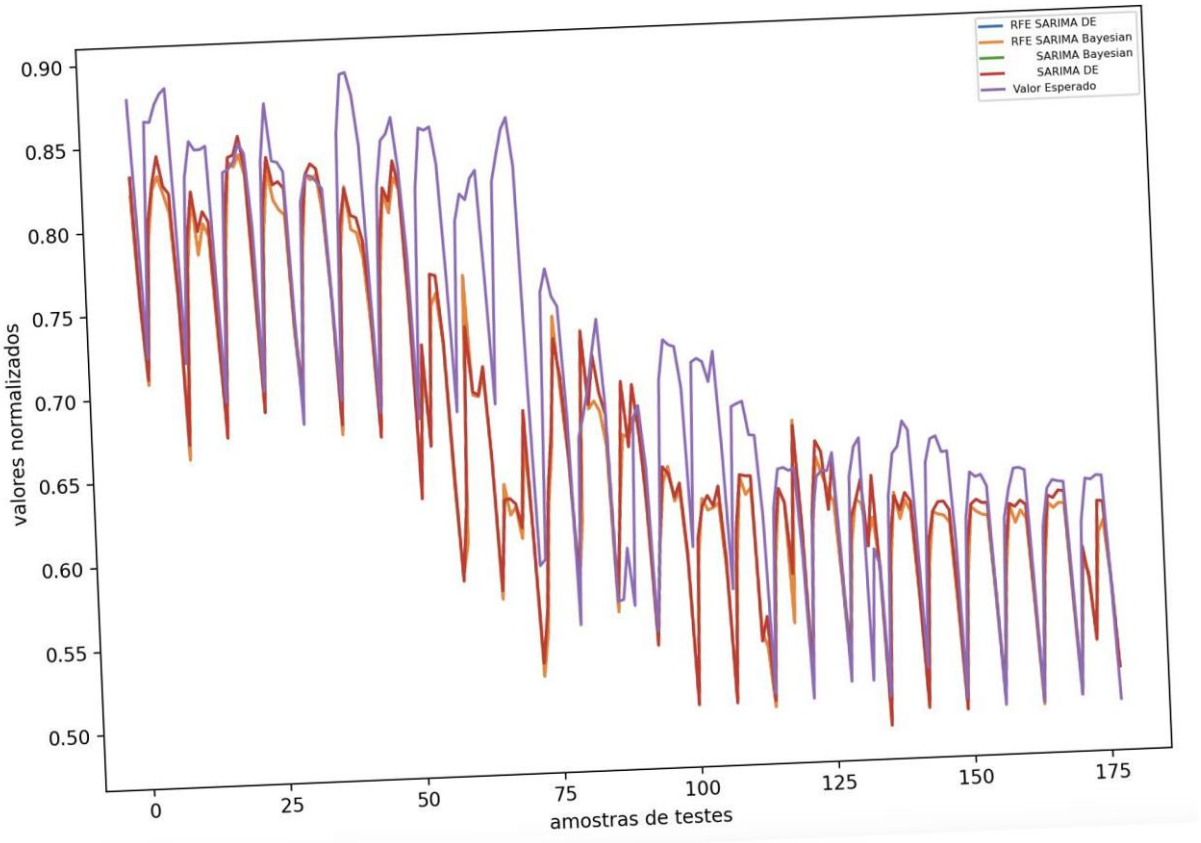

Figura 6.32: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão SARIMA

Já a Tabela 6.29 mostra os valores otimizados para os hiperparâmetros do SARIMA, mostrando uma pequena diferença nos valores de seus hiperparâmetros.

modelo SARIMA	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Com RFE - DE	2	0	0	2	0	2	4
Com RFE-Bayesiana	2	0	0	2	0	2	4
Sem RFE – DE	1	0	1	1	0	1	3
Sem RFE - Bayesiana	1	0	1	0	0	1	4

Tabela 6.29: Valores de hiperparâmetros para o SARIMA com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

O próximo algoritmo, o SVN, teve como métrica de desempenho de RMSLE, como vistos na Tabela 6.17, uma variação entre $2{,}4e^{-2}$ e $2{,}6e^{-2}$, que podem ter sido

afetados por 6 hiperparâmetros que variaram bastante, conforme mostrado na Tabela 6.30.

modelo SVR	Núm. *Kernels*	gama	coeficiente	tolerância	Regularizador	epsilon
Com RFE - DE	1	0,00501	0,032203	0,007741	0,479160	0,009556
Com RFE-Bayesiana	1	0,001	0,01	0,01	0,5	0,001
Sem RFE – DE	1	0,00910	0,059598	0,004305	0,305519	0,028450
Sem RFE - Bayesiana	1	0,007881	0,048387	0,006613	0,11060	0,016131

Tabela 6.30: Valores de hiperparâmetros para o SVR com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

E na Figura 6.33 pode-se notar uma grande diferença entre os valores esperados e os valores previstos, mostrando que o sistema durante a fase de treinamento não conseguiu obter as características apresentadas na fase de testes.

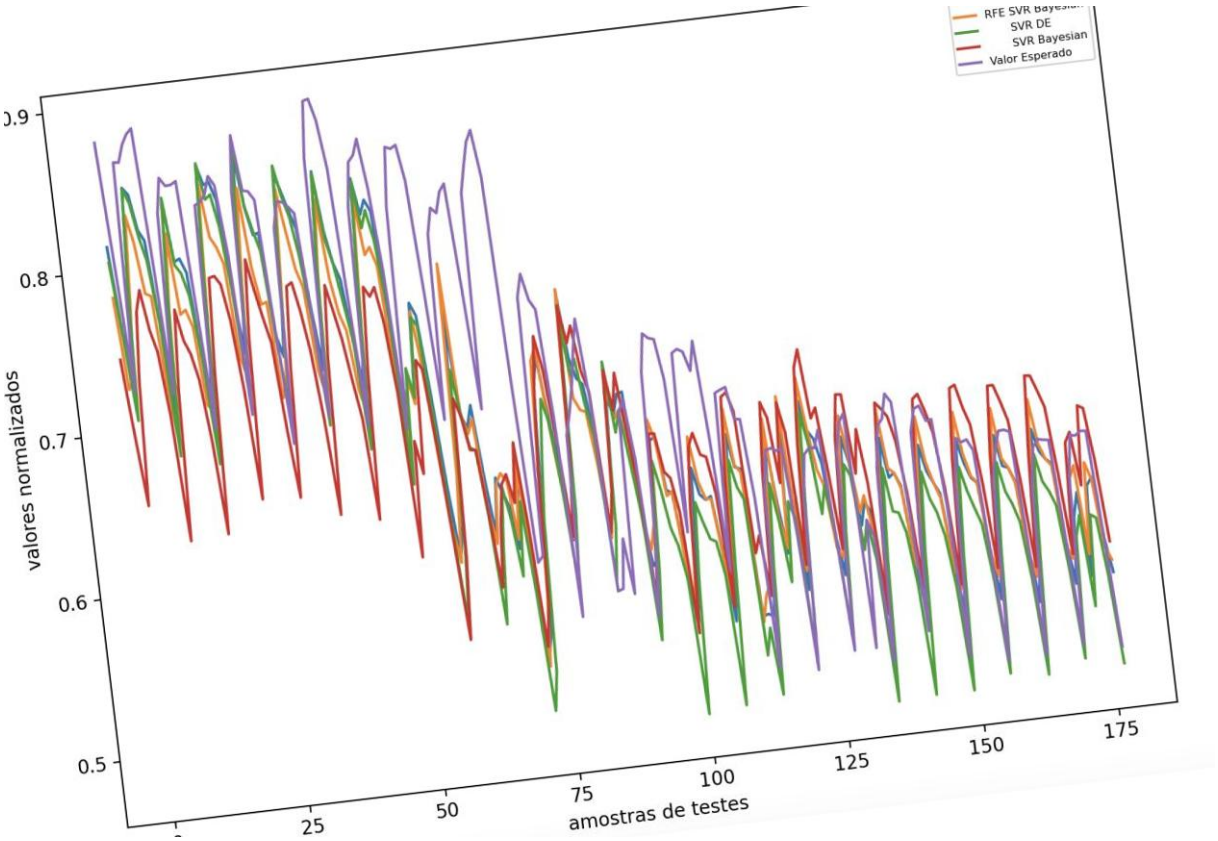

Figura 6.33: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão SVR

Finalmente, o último algoritmo testado, o XGBoost, foi também o que apresentou os melhores resultados quando usando a métrica de desempenho RMSLE, que variou entre $1{,}6e^{-2}$ e $2{,}0e^{-2}$, sendo que a otimização DE apresentou melhores resultados que o otimizador Bayesiano. O resultado encontra-se na Figura 6.34.

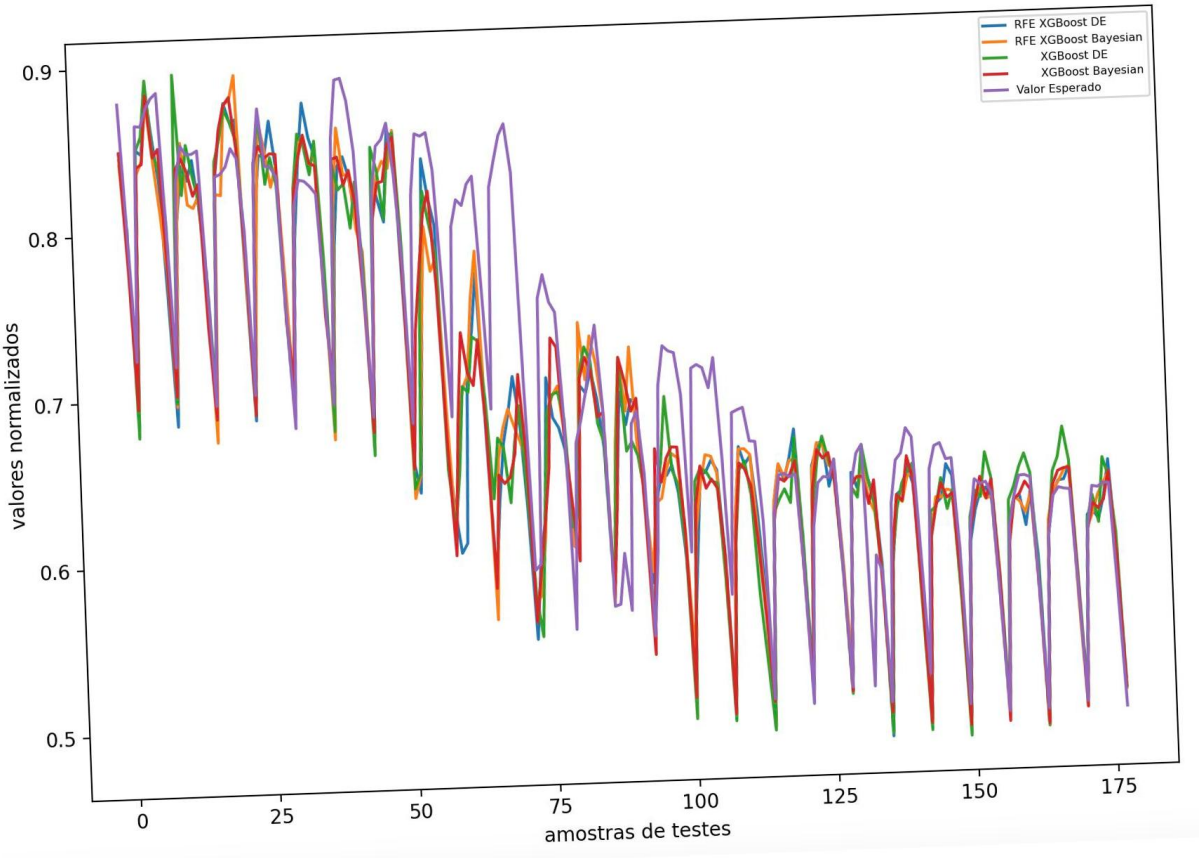

Figura 6.34: Resultados para a previsão da série temporal de consumo da Polônia dos modelos/algoritmos de previsão XGBoost

Já os seus hiperparâmetros podem ser encontrados na Tabela 6.31 a seguir.

modelo XGboost	Max. *Depth*	Menor peso por *children*	*Learning rate*	Subamostra	Subamostra de colunas
Com RFE - DE	33	3	0,102218	0,549257	0,649768
Com RFE-Bayesiana	19	3	0,415264	0,704153	0,728707
Sem RFE – DE	20	4	0,119497	0,687534	0,785308
Sem RFE - Bayesiana	21	5	0,1	0,754834	0,8

Tabela 6.31: Valores de hiperparâmetros para o XGBoost com otimizadores DE e Bayesiano e com ou sem uso de RFE para a previsão da série temporal de consumo da Polônia

Outro dado analisado é o de tempo de execução. Para tal, o tempo analisado começa a partir do ponto de execução do treinamento até a obtenção dos dados de teste e o cálculo do seu RMSLE. Todos os modelos foram rodados em um mesmo projeto, o que permite tal análise de maneira mais detalhada. A Tabela 6.32 mostra o tempo de execução em segundos de todos os métodos para os dados do Brasil (os resultados foram arredondados para apenas duas casas decimais), sendo que os melhores resultados foram realçados em verde, enquanto os piores foram realçados em vermelho.

algoritmo	Sem RFE		Com RFE	
	Bayesiano	DE	Bayesiano	DE
ARIMA	101s	188s	55s	150s
CNN	259s	3999s	244s	3640s
DRVFL	7s	11s	10s	16s
ELM	9s	2s	13s	7s
ESN	5s	6s	13s	14s
FFNN	3534s	5083s	4478s	4945s
LightGBM	9s	14s	11s	18 s
LSTM	2498s	3577s	1993s	3576s
OLS	5s	0,1s	11s	5s
QRF	15s	62s	20s	65s
RF	7s	22s	11s	23s
SARIMA	70s	384s	59s	345s
SVR	7s	1s	15s	6s
XGBoost	10s	18s	16s	34s

Tabela 6.32: Comparativo do tempo de execução para a série temporal de consumo da região Sul do Brasil

A otimização DE obteve melhores resultados, mas isso ocorreu no algoritmo OLS, que, pela natureza de sua implementação calcula em uma única iteração os valores de previsão. Se considerar-se o segundo modelo mais rápido, tem-se então o ELM, que também não conta com um sistema de iterações. Ao desconsiderar-se ambos, tem-se então o LightGBM que realmente é um modelo leve (como sugerido em seu nome), que conseguiu rodar rapidamente, mesmo com um número de iterações sendo de 50. Já o pior desempenho seria o FFNN, que demorou

aproximadamente 84 minutos para tentar melhorar a sua métrica de desempenho RMSLE durante a fase de otimização, mas não conseguiu. Uma possível solução para que isso não aconteça, é de se usar um critério de parada para que o sistema possa parar caso não esteja melhorando a sua performance.

Já a Tabela 6.33 mostra o tempo de execução em segundos de todos os algoritmos para os dados da Polônia. Novamente, o modelo OLS praticamente resolveu instantaneamente a previsão, sendo acompanhado por SARIMA. Nota-se que modelos com camadas escondidas, que precisam fazer rodadas de treinamento, acabam sendo mais devagar, mas acabam tendo melhores resultados, que é o caso do modelo XGBoost. Já o pior tempo foi novamente para o FFNN que demorou 94 minutos para terminar a execução.

algoritmo	Sem RFE		Com RFE	
	Bayesiano	DE	Bayesiano	DE
ARIMA	93s	246s	119s	495s
CNN	33s	4916s	407s	4952s
DRVFL	12s	64s	16s	45s
ELM	7s	4s	18s	11s
ESN	11s	14s	16s	21s
FFNN	5195s	4494s	4390s	5686s
LightGBM	12s	28s	16s	36s
LSTM	3709s	4903s	3071s	1340s
OLS	5s	0,1s	14s	7s
QRF	22s	112s	29s	96s
RF	42s	167s	50s	181s
SARIMA	407s	1784s	133s	642s
SVR	7s	2 s	18s	10s
XGBoost	17s	62s	34s	79s

Tabela 6.33: Comparativo do tempo de execução para a série temporal de consumo na Polônia

No apêndice A, encontram-se todos os resultados com os seus valores em números de pontos flutuantes e seus tempos de execução para cada um dos métodos, juntamente com os hiperparâmetros usados.

Os resultados obtidos mostram que o uso de RFE na fase de pré-processamento permitiu uma melhora de aproximadamente 5% para a base de dados

da Polônia, enquanto na base de dados do Brasil, a melhora foi de 10%. Nota-se também, que o melhor algoritmo para a base brasileira foi o DRVFL.

A otimização DE e Bayesiana para o DRVFL apresentam similaridades, sendo que ao se não usar a RFE, o melhor resultado foi usando-se a otimização Bayesiana, sendo que o melhor resultado geral foi obtido usando-se o DE. De modo geral, os modelos usando a otimização Bayesiana foram melhores que o DE, sendo que em alguns resultados, chegou-se a ter mais de 16% de diferença, que seria o caso do método CNN.

O pior desempenho em termos de RMSLE foi o ARIMA, que não conseguiu prever dados que melhorassem a métrica de desempenho RMSLE mesmo após ter 50 iterações. A razão para isso ocorre porque ARIMA não espera ter sazonalidade no sistema, que seria o caso do banco de dados usado. Neste caso, pode-se usar o modelo SARIMA, que, como esperado, conseguiu obter resultados satisfatórios.

No banco de dados da Polônia, continua-se notando uma melhora significativa no uso de RFE, sendo os ganhos muito similares com a base brasileira, o que já era esperado. Nota-se também um desempenho parecido entre as técnicas de otimização, sendo que a técnica Bayesiana é ligeiramente melhor do que a DE.

O melhor algoritmo para a base polonesa foi o XGBoost, que mostrou um ganho de aproximadamente 5 % em relação ao segundo melhor modelo que foi o DRVFL e mais de 300% quando comparado ao pior modelo. Nota-se ainda que o uso de DE apresentou um ganho de aproximadamente 9% em relação ao otimizador Bayesiano e que o uso de RFE melhorou em aproximadamente 5% os resultados quando se comparando a métrica de desempenho RMSLE.

7 CONSIDERAÇÕES FINAIS

A principal proposta da tese foi propor um novo modelo de previsão de séries temporais combinando DVRFL com otimização DE. A partir da análise dos resultados de previsão obtidos, é possível notar que o modelo proposto atingiu resultados promissores em termos de RMSLE (o objetivo é minimizar os valores obtidos por esta métrica) em quase todos as combinações apresentadas, sendo que ele foi o que teve a menor RMSLE no caso da base de dados brasileira e ficou em segundo lugar, apenas atrás do modelo XGBoost.

Também, como outra contribuição da tese, projetou-se além da otimização baseada em DE, uma abordagem de otimização Bayesiana aplicada para a otimização dos valores dos hiperparâmetros dos modelos de previsão avaliados, também mostrou resultados competitivos com outros modelos de previsão avaliados. Neste contexto, a otimização Bayesiana foi eficiente principalmente em casos em que o uso de RFE não foi aplicado, porém com esta abordagem obteve um maior custo computacional quando se compara a ordem de grandeza de tempo de execução quando comparado ao obtido pela DE.

Outra contribuição da tese foi uma análise de séries temporais em termos de pré-processamento relacionadas à demanda do consumo de energia elétrica pelo uso do RFE que ajudou, muitas vezes, na obtenção de resultados melhores em ambas as bases de dados avaliadas. No caso dos modelos analisados por esta tese uma pesquisa inicial pode melhorar o sistema em mais de 10%. O DRVFL por não ter um procedimento iterativo de otimização de pesos, acaba sendo um modelo de baixo custo computacional e acabou demonstrando isso com um desempenho de tempo satisfatório e com resultados competitivos com os outros modelos em termos da métrica RMSLE, sendo que para o melhor modelo para a série temporal de consumo de energia da região sul do Brasil usando o otimizador DE, foram adotadas apenas 2 camadas intermediárias (ocultas) para os testes. Mesmo não sendo o melhor método para a série temporal da Polônia, o DVRFL ficou em segundo lugar em termos da métrica RMSLE, sendo aproximadamente 50% mais rápido em seu tempo de execução do que o melhor modelo, que seria o XGBoost. Como esperado também, o DRVFL demonstrou prever melhores resultados que o modelo ELM, o que demonstra que ao se usar os dados de entrada diretamente na camada de saída contribui com que o modelo obtenha melhores resultados.

Com os resultados, é possível confirmar que o objetivo geral desta tese foi atingido com sucesso, tendo-se mostrado que o uso de DVRFL otimizado, tanto por otimização por meio da DE ou pelo uso de otimização Bayesiana mostraram resultados satisfatórios em termos de RMSLE na fase de testes para ambos os casos avaliados. O mencionado desempenho foi mais frequente quando houve uma pré-análise dos dados de entrada com variáveis a serem previstas com atrasos que foram usados para o treinamento, mostrando uma acurácia em termos de RMSLE quando comparado a outros 13 modelos de previsão que também foram otimizados de forma similar.

Também, ao longo do desenvolvimento da tese, notou-se possíveis inovações que podem ser aplicadas aos modelos avaliados, sendo que as tais estratégias serão discutidas na próxima seção.

7.1 Sugestões para Futura Pesquisa

Com as contribuições apresentadas, esperava-se obter um comparativo que permitiria analisar quais combinações de modelos de previsão seriam melhores para aplicar para previsão de séries temporais de demanda do consumo de energia elétrica. Porém, ao longo dos estudos baseados na literatura recente dos temas envolvidos nesta tese, foi possível observar que existe uma tendência em se analisar tal tarefa de modo mais profundo, dividindo-se a mesma em blocos construtivos que podem permitir diferentes abordagens:

- **Diferentes bases de dados**: Esta tese usou duas bases de dados bastante distintas, porém, na comunidade científica, é possível de se encontrar várias outras bases de dados que poderiam ser usadas, para poder se ter uma análise mais criteriosa da robustez dos sistemas. Por exemplo, base de dados maiores, que poderiam englobar maiores períodos de observância, como anos ou décadas poderiam ser usadas. Também podem ser considerados passos maiores para a previsão, como médio ou longo prazos.

- **Pré-processamento dos dados**: Neste contexto, abordagens de pré-processamento de dados e de engenharia de características pode ser um caminho para obtenção de variáveis previsoras relevantes. Conforme sugerido por Bauer *et al.* (2020), antes de se aplicar os dados para analisar suas características, pode-se aplicar uma extração de características por meio da análise de Fourier (Lai, Chen e Huang, 2020), usar métodos de decomposição da série temporal como o modelo

ensemble empirical mode decomposition (EEMD) (Da Silva *et al.*, 2021) ou uma variação do EEMD, chamada *complete ensemble empirical mode decomposition*(CEEMD) (Da Silva *et al.*, 2020) ou ainda o modelo *seasonal and trend decomposition using loess* (STL) (Da Silva *et al.*, 2020), o que ajudaria a obter diferentes características.

- **Construção do modelo de Previsão:** Uma vez que os termos de tendência e sazonalidade foram obtidos por alguma técnica de pré-processamento, pode-se então usá-las com suporte na construção do modelo. Um modelo que está tornando-se popular e que esta tese acabou não abrangendo seria o CatBoost (Dong *et al.*, 2021), onde neste pode-se adotar variáveis categóricas.

- Abordagens de comitês **previsão de séries temporais**: Neste bloco construtivo, pode-se então fazer um sistema de *ensemble*, que permite usar os valores da tendência e sazonalidade na forma de modelos de comitês com empilhamento (do inglês *stacking*).

REFERÊNCIAS

AALTO UNIVERSITY. **Applications of machine learning group home page**. Disponível em: https://research.cs.aalto.fi/aml/datasets.shtml (Acessado em 02 janeiro 2021)

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., ZHENG, X. **Tensorflow: Large-scale machine learning on heterogeneous distributed systems**. arXiv preprint arXiv:1603.04467, 2016.

ABU-MOSTAFA, Y. S.; MALIK, M.-I.; HSUAN-TIEN, L. **Learning from data.** AMLBook, Nova York, NY, EUA, vol. 4, 2012.

ABUSHANAB, W. S., ABD ELAZIZ, M., HANDOURAH, E. I., MOUSTAFA, E. B., ELSHEIKH, A. H. **A New Fine-Tuned Random Vector Functional Link Model Using Hunger Games Search Optimizer for Modeling Friction Stir Welding Process of Polymeric Materials**. Journal of Materials Research and Technology, 2021.

ACKLEY, D. H., HINTON, G. E., SEJNOWSKI, T. J. **A learning algorithm for Boltzmann machines**. Cognitive Science, vol. 9 no. 1, p. 147-169, 1985.

ADEGUN, I. P., VADAPALLI, H. B. **Facial micro-expression recognition: A machine learning approach.** Scientific African, vol. 8, 2020.

ALALIMI, A., PAN, L., AL-QANESS, M. A., EWEES, A. A., WANG, X., ABD ELAZIZ, M. **Optimized random vector functional link network to predict oil production from Tahe oil field in China.** Oil & Gas Science and Technology–Revue d'IFP Energies nouvelles, vol. 76, p. 3, 2021.

AMARAL, H. L. M. D. **Desenvolvimento de uma nova metodologia para previsão do consumo de energia elétrica de curto prazo utilizando redes neurais artificiais e decomposição de séries temporais,** Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, São Paulo, São Paulo, 2020.

ARDABILI, S., MOSAVI, A., VARKONYI-KOCZY, A. R. **Advances in machine learning modeling reviewing hybrid and ensemble methods**. International Conference on Global Research and Education (p. 215-227). Springer, Balatonfüred, Hungria, 2019.

ATKINSON, A. B. **The Box-Cox transformation: Review and extensions**. Statistical Science, vol. 36, no. 2, p. 239-255, 2020.

BARANDAS, M.; FOLGADO, D.; FERNANDES, L.; SANTOS, S.; ABREU, M.; BOTA, P.; GAMBOA, H. **Tsfel: Time series feature extraction library**. SoftwareX, vol. 11, p. 100456, 2020.

BAUER, A.; ZÜFLE, M.; HERBST, N.; KOUNEV, S.; CURTEF, V. **Telescope: An automatic feature extraction and transformation approach for time series forecasting on a level-playing field**. Em IEEE 36th International Conference on Data Engineering (ICDE), Dallas, EUA, p. 1902-1905, 2020.

BEE, V. **Development of an autonomous robotic air hockey player**, Thesis, Massachusetts Institute of Technology. Department of Mechanical Engineering, Massachusetts, EUA, 2013.

BENGIO, Y. **Learning deep architectures for AI**. **Foundations and Trends in Machine Learning**, New Publishers Inc, Nova York, NY, EUA, 2009.

BIANCHINI, M.; SCARSELLI, F. **On the complexity of neural network classifiers: a comparison between shallow and deep architectures**. IEEE Transactions on Neural Networks and Learning Systems, vol. 25, no. 8, p. 1553-1565, 2014.

BOUKTIF, S., FIAZ, A., OUNI, A., SERHANI, M. A. **Multi-sequence LSTM-RNN deep learning and metaheuristics for electric load forecasting**. Energies, vol. 13, no. 2, p. 391, 2020.

BOUSSAÏD, I.; LEPAGNOT, J.; SIARRY, P. **A survey on optimization metaheuristics.** Information Sciences, vol. 237, p. 83-117, 2013.

BOX, G. E. P.; JENKINS, G. M. **Time series analysis, forecasting and control**. Wiley, Hoboken, New Jersey, EUA, 1976.

BOX, T. M., WHITE, M. A., BARR, S. H. **A contingency model of new manufacturing firm performance**. Entrepreneurship Theory and Practice, vol. 18, no. 2, p. 31-45, 1994.

BRAEI, M., WAGNER, S. **Anomaly detection in univariate time-series: A survey on the state-of-the-art**. arXiv preprint arXiv:2004.00433, 2020

BREIMAN, L. **Bagging predictors**. Machine learning, vol. 24, no. 2, p. 123-140, 1996.

BROCKWELL, P. J., DAVIS, R. A. **Stationary ARMA Processes**. Em Time Series: Theory and Methods. Springer, New York, NY, EUA, p. 77-113, 1991.

BRUNAK, S.; LAUTRUP, B. **Neural networks - computers with intuition**. World Scientific, Singapura, 1990.

BUKHARI, A. H.; RAJA, M. A. Z.; SULAIMAN, M.; ISLAM, S.; SHOAIB, M.; KUMAM, P. **Fractional neuro-sequential ARFIMA-LSTM for financial market forecasting**. IEEE Access, vol. 8, p. 71326-71338, 2020.

CAMPOS, R. J., **Previsão de séries temporais com aplicações a séries de consumo de energia elétrica**, Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais, 2008.

CHATTERJES, S., DASH, A., BANDOPADHYAY, S. **Ensemble support vector machine algorithm for reliability estimation of a mining machine**. Quality and Reliability Engineering International, vol. 31, no. 8, p. 1503-1516, 2015.

CHEN, T., GUESTRIN, C. **XGBoost: A scalable tree boosting system**. Em Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, p. 785-794, 2016.

CHENG, X., FENG, Z. K., NIU, W. J. **Forecasting monthly runoff time series by single-layer feedforward artificial neural network and grey wolf optimizer**. IEEE Access, vol. 8, p. 157346-157355, 2020.

CHOLLET, F. **Keras: The python deep learning library**. Astrophysics Source Code Library, ascl-1806, 2018.

CUNHA, C. B. **Uma contribuição para o problema de roteirização de veículos com restrições operacionais**. Tese de Doutorado, Departamento de Engenharia de Transportes da Universidade de São Paulo, EPUSP-PTR, São Paulo, São Paulo, 1997.

DA SILVA, R. G., RIBEIRO, M. H. D. M., MORENO, S. R., MARIANI, V. C., DOS SANTOS COELHO, L. **A novel decomposition-ensemble learning framework for multi-step ahead wind energy forecasting**. Energy, vol. 216, p. 119174, 2021.

DADEBAYEV, D.; WEI, G. W.; XION, T. E. **A machine learning approach to EEG-based prediction of human affective states using recursive feature elimination method**. Em MATEC Web of Conferences, Conferência online, EDP Sciences, vol. 335, p. 04001, 2021.

DEL SER, J., CASILLAS-PEREZ, D., CORNEJO-BUENO, L., PIETRO-GODINO, L., SANZ-JUSTO, J., CASANOVA-MATEO, C., SALCEDO-SANZ, S. **Randomization-based machine learning in renewable energy prediction problems: critical literature review, new results and perspectives**. arXiv preprint arXiv:2103.14624, 2021.

DENG, W., LIU, H., XU, J., ZHAO, H., SONG, Y. **An improved quantum-inspired differential evolution algorithm for deep belief network**. IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 10, p. 7319-7327, 2020.

DOKEROGLU, T., SEVINC, E., KUCUKYILMAZ, T., COSAR, A. **A survey on new generation metaheuristic algorithms**. Computers & Industrial Engineering, vol. 137, 2019

DUMOULIN, V.; VISIN, F. **A guide to convolution arithmetic for deep learning.** arXiv preprint arXiv:1603.07285, 2016.

DONG, L., ZENG, W., WU, L., LEI, G., CHEN, H., SRIVASTAVA, A. K., & GAISER, T. **Estimating the Pan Evaporation in Northwest China by Coupling CatBoost with Bat Algorithm**. Water 2021, vol. 13, no. 256, 2021.

ELAVARASAN, D., VINCENT PM, D. R., SRINIVASAN, K., CHANG, C. Y. **A hybrid CFS filter and RF-RFE wrapper-based feature extraction for enhanced agricultural crop yield prediction modeling**. Agriculture, vol. 10, no. 9, p. 400, 2020.

ESHTAY, M., FARIS, H., OBEID, N. **Improving extreme learning machine by competitive swarm optimization and its application for medical diagnosis problems**. Expert Systems with Applications. vol. 104, p. 134-152, 2018.

FARSI, M., HOSAHALLI, D., MANJUNATHA, B. R., GAD, I., ATLAM, E. S., AHMED, A., GHONEIM, O. A. **Parallel genetic algorithms for optimizing the SARIMA model for better forecasting of the NCDC weather data**. Alexandria Engineering Journal, vol. 60, no. 1, p. 1299-1316, 2021.

FRIEDMAN, J.H. **Stochastic gradient boosting**. Computational statistics & data analysis, vol. 38, no. 4, p. 367–378, 2002.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. **The elements of statistical learning**; Springer Series in Statistics; Springer: Berlin, Alemanha; vol. 1, 2001.

GARRO, B. A.; SOSSA, H.; VAZQUEZ, R. A. **Design of artificial neural networks using a modified particle swarm optimization algorithm**. Em IEEE Proceedings of the 2009 International Joint Conference on Neural Networks. Atlanta, EUA, 2009.

GAO, K., MEI, G., PICCIALLI, F., CUOMO, S., TU, J., HUO, Z. **Julia language in machine learning: algorithms, applications, and open issues**. Computer Science Review, vol. 37, 2020.

GHASEMI, P., KARBASI, M., NOURI, A. Z., TABRIZI, M. S., AZAMATHULLA, H. M. **Application of Gaussian process regression to forecast multi-step ahead SPEI drought index**. Alexandria Engineering Journal, vol. 60, no. 6, p. 5375-5392, 2021.

GIUSTI, R. **Classificação de séries temporais utilizando diferentes representações de dados e ensembles**. Tese de Doutorado do programa de Pós-graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC) da Universidade de São Paulo, São Paulo, São Paulo, 2017.

GOLDBERG, D. E. **Genetic algorithms in search, optimization, and machine learning**. Addison, Reading, Reino Unido, vol. 36, 1989

GOVINDARAJAN, S.; ARDILA-REY, J. A.; KRITHIVASAN, K.; SUBBAIAH, J.; SANNIDHI, N.; BALASUBRAMANIAN, M. **Development of hypergraph based improved random forest algorithm for partial discharge pattern classification**. IEEE Access, vol. 9, p. 96-109., 2020.

GROUMPOS, P. P. **Deep learning vs. wise learning: a critical and challenging overview**. ScienceDirect, IFAC-PapersOnLine, vol. 49, no. 29, p. 180-189, 2016.

GHOSH, S., MUKHERJEE, H., OBAIDULLAH, S. M., SANTOSH, K. C., DAS, N., ROY, K. **A survey on extreme learning machine and evolution of its variants**. International Conference on Recent Trends in Image Processing and Pattern Recognition, Springer, Singapura, p. 572-583, 2018.

GURNANI, M.; KORKE, Y.; SHAH, P.; UDMALE, S.; SAMBHE, V.; BHIRUD, S. **Forecasting of sales by using fusion of machine learning techniques**. Em International Conference on Data Management, Analytics and Innovation (ICDMAI). Pune, India. p. 93-101, 2017.

HAJIRAHIMI, Z., KHASHEI, M. **Hybrid structures in time series modeling and forecasting: A review**. Engineering Applications of Artificial Intelligence, vol. 86, p. 83-106, 2019.

HAN, K.; YU, D.; TASHEV, L. **Speech emotion recognition using deep neural network and extreme learning machine**. 5th Annual Conference of the International Speech Communication Association. Singapura, p. 223-227, 2014.

HARRIS, C. R., MILLMAN, K. J., VAN DER WALT, S. J., GOMMERS, R., VIRTANEN, P., COURNAPEAU, D., OLIPHANT, T. E. **Array programming with NumPy**. Nature, vol. 585, no. 7825, p. 357-362, 2020.

HASTIE, T.; TIBSHRIANI, R. E FRIEDMAN, J. **The elements of statistical learning**, Springer Series em Statistics, Springer, Nova York, vol. 1, no. 10, p. 395, 2009.

HE, F.; ZHOU, J.; MO, L.; FENG, K.; LIU, G.; HE, Z. **Day-ahead short-term load probability density forecasting method with a decomposition-based quantile regression forest**. Applied Energy, vol. 262, 2020.

HENRIQUEZ, P. A., RUZ, G. A. **A non-iterative method for pruning hidden neurons in neural networks with random weights**. Applied Soft Computing, vol. 70, p. 1109-1121, 2018

HINTON, G. E.; DAYAN, P.; FREY B. J.; NEAL, R. M. **The "wake-sleep" llgorithm for unsupervised neural networks**, Science, vol. 268, no. 5214, p.1158-1161, 1995.

HINTON, G.; DENG, L.; YOU, D.; DAHL, G. E.; MOHAMED, A.; JAITLY, N.; SENIOR, A.; VANHOUCKE, V.; NGUYEN, P.; SAINATH T. N.; KINGSBURY, B. **Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups**. IEEE Signal Processing Magazine, vol. 29, no. 6, p. 82-97, 2012.

HINTON, G. E.; SRIVASTAVA, N.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. R. **Improving neural networks by preventing co-adaptation of feature detectors**. arXiv preprint arXiv:1207.0580, 2012.

HOCHREITER, S.; SCHMIDHUBER, J. **LSTM can solve hard long time lag problems**. Em Advances in Neural Information Processing Systems, Cambridge, EUA, p. 473-479, 1997.

HU, H.; WANG, L.; LV, S. X. **Forecasting energy consumption and wind power generation using deep echo state network**. Renewable Energy, vol. 154, p. 598-613, 2020.

HUANG, G.B., CHEN, L., SIEW, C.K. **Universal approximation using incremental constructive feedforward networks with random hidden nodes**. IEEE Transactions Neural Networks. Vol. 17, no. 4, p. 879–892, 2006.

HUANG, G. B.; ZHU, Q.-Y.; SIEW, C. K. **Extreme learning machine: theory and applications**. Elsevier, Neurocomputing. vol. 70, no. 1-3, p. 489-501, 2006.

HUSMEIER, D. **Random vector functional link (RVFL) networks**. Em Neural Networks for Conditional Probability Estimation, Springer, Londres, Reino Unido, p. 87-97, 1999.

HUSSAIN, K., SALLEH, M. N. M., CHENG, S., SHI, Y. **Metaheuristic research: a comprehensive survey.** Artificial Intelligence Review, vol. 52, no. 4, p. 2191-2233, 2019

HYNDMAN, R.J. ATHANASOPOULOS, G. **Forecasting: principles and practice**, OTexts, Melbourne, Australia, vol. 2, 2018.

IKEDA, S., NAGAI, T. **A novel optimization method combining metaheuristics and machine learning for daily optimal operations in building energy and storage systems**. Applied Energy, vol. 289, 2021.

JANA, R. K., GHOSH, I., SANYAL, M. K. **A granular deep learning approach for predicting energy consumption**. Applied Soft Computing, vol. 89, 2020.

JANIESCH, C., ZSCHECH, P., HEINRICH, K. **Machine learning and deep learning**. Electronic Markets, p. 1-11, 2021.

JAEGER, H. **The "echo state" approach to analyzing and training recurrent neural networks with an erratum note.** German National Research Center for Information Technology GMD Technical Report, Bonn, Alemamha. vol. 148, p. 13-34, 2001.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. **Artificial neural networks: a tutorial**, IEEE Computer Society, vol. 29, no. 3. p. 31-44, 1996.

JENSEN, R. **Combining rough and fuzzy sets for feature selection**. Tese de Doutorado na Escola de informática, Universidade de Edinburg. Edinburgo, Reino Unido, 2005.

JIANG, L., XING, R., CHEN, X., XUE, B. **A survey-based investigation of greenhouse gas and pollutant emissions from household energy consumption in the Qinghai-Tibet plateau of China**. Energy and Buildings, vol. 235, 2021.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. **Reinforcement learning: a survey**, Journal of artificial intelligence research, vol. 4, p. 237-285, 1996.

KANG, K.; BAE, C.; YEUNG, H.W.F.; CHUNG, Y. Y. **A hybrid gravitational search algorithm with swarm intelligence and deep convolutional feature for object tracking optimization**. ScienceDirect, Applied Soft Computing. vol. 66, p. 319-329, 2018.

KARA, A.; DOGAN, I. **Reinforcement learning approaches for specifying ordering policies of perishable inventory systems**. ScienceDirect, Expert Systems with Applications, vol. 91, p. 150-158, 2018.

KARABOGA, D.; AKAY, B. **Artificial bee colony (ABC) algorithm on training artificial neural networks**. Em IEEE Signal Processing and Communications Applications, Eskisehir, Turquia, 2007.

KARMY, J. P.; MALDONADO, S. **Hierarchical time series forecasting via support vector regression in the european travel retail industry**. Expert Systems with Applications, vol. 137, p. 59-73, 2019.

KATUWAL, R., SUGANTHAN, P. N. **Stacked autoencoder based deep random vector functional link neural network for classification**. Applied Soft Computing, vol. 85, 2019.

KATUWAL, R., SUGANTHAN, P. N., TANVEER, M. **Random vector functional link neural network-based ensemble deep learning**. arXiv preprint arXiv:1907.00350, 2019.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T. Y. **Lightgbm: A highly efficient gradient boosting decision tree**. Em Advances in neural information processing systems, Long Beach, EUA, vol. 30, p. 3146-3154, 2017.

KENDALL, A.; GAL, Y. **What uncertainties do we need in bayesian deep learning for computer vision?**, arXiv preprint arXiv:1703.04977, 2017.

KENDALL, A.; GAL, Y.; CIPOLLA, R. **Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.** Em Proceedings of the IEEE conference on computer vision and pattern recognition, Salt Lake City, EUA, p. 7482-7491, 2018.

KENNEDY, J.; EBERHART, R. **Particle swarm optimization.** IEEE Em Proceedings of ICNN'95 International Conference on Neural Networks, Perth, Australia, p. 1942–1948, 1995.

KER, A. P., TOLHURST, T. N. **On the treatment of heteroscedasticity in crop yield data**. American Journal of Agricultural Economics, vol. 101, no. 4, 1247-1261, 2019.

KHASHEI, M., BIJARI, M. **An artificial neural network (p, d, q) model for timeseries forecasting**. Expert Systems with applications, vol. 37, no. 1, p. 479-489, 2010.

KHASHEI, M., BIJARI, M., **A novel hybridization of artificial neural networks and ARIMA models for time series forecasting**. Applied Soft Computing. Vol. 11, p. 2664–2675., 2011.

KIANGALA, S. K.; WANG, Z. **An effective adaptive customization framework for small manufacturing plants using extreme gradient boosting-XGBoost and random forest ensemble learning algorithms in an Industry 4.0 environment**. Machine Learning with Applications, 2021.

KIM, H. J., CHOI, D. Y., SEO, D. **Development and verification of prototypical office buildings models using the national building energy consumption survey in Korea**. Sustainability, vol.13, no. 7, p. 3611, 2021.

KOBER, J.; PETERS, J. **Reinforcement learning in robotics: a survey**. Springer, Reinforcement Learning, vol. 12, p. 579-610, 2012.

KOÇ, O.; MAEDA, G.; PETERS, J. **Online optimal trajectory generation for robot table tennis**. ScienceDirect, Robotics and Autonomous Systems, vol. 105, p.121-137, 2018.

KOSTER, R., CHADWICK, M. J. **What can classic Atari video games tell us about the human brain?** Neuron, vol. 109, no. 4, p. 568-570, 2021.

LAKSHMI, K., MAHABOOB, B., RAJAIAH, M., NARAYANA, C. **Ordinary least squares estimation of parameters of linear model**. Journal Mathematical Computer Science, vol.11, no. 2, p. 2015-2030, 2021.

LAI, Z.; CHEN, Q.; HUANG, L. **Fourier series-based discrete element method for computational mechanics of irregular-shaped particles**. Computer Methods in Applied Mechanics and Engineering, vol. 362, p. 112873, 2020.

LAMMEY, R. **CrossRef text and data mining services**. Insights, vol. 28, no. 2, 2015.

LAWSON, C. L.; HANSON, R. J. **Solving least squares problems**. Society for Industrial and Applied Mathematics, 1995.

LECUN, Y.; BOTTOU, L.; ORR, G. B.; MÜLLER, K. **Efficient BackProp**. Springer, Em: ORR, G. B.; MÜLLER, K. Neural Networks: Tricks of the Trade., Berlin, Alemanha, vol. 1524, p. 9-48, 1998.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. **Gradient-based learning applied to document recognition**, Proceedings of the IEEE, vol. 86, no. 11, p. 2278-2324, 1998.

LI, X., YANG, Y., HU, N., CHENG, Z., CHENG, J. **Discriminative manifold random vector functional link neural network for rolling bearing fault diagnosis**. Knowledge-Based Systems, vol. 211, p. 106507, 2021.

LIAW, A., WIENER, M. **Classification and regression by random forest**. R news, vol. 2, no. 3, p. 18-22, 2002

LIM, B., ZOHREN, S. **Time series forecasting with deep learning: a survey**. arXiv preprint arXiv:2004.13408, 2020.

LIU, B., DING, M., SHAHAM, S., RAHAYU, W., FAROKHI, F., LIN, Z. When Machine Learning Meets Privacy: A Survey and Outlook. ACM Computing Surveys (CSUR), vol. 54, no. 2, p. 1-36, 2021.

LJUNG, G. M. **On outlier detection in time series**. Journal of the Royal Statistical Society: Series B (Methodological), vol. 55, no. 2, p. 559-567, 1993.

MAGAZZINO, C., MUTASCU, M., MELE, M., SARKODIE, S. A. **Energy consumption and economic growth in Italy: a wavelet analysis**. Energy Reports, vol. 7, p. 1520-1528, 2021.

MAKINEN, S., SKOGSTROM, H., LAAKSONEN, E., MIKKONEN, T. **Who needs MLOps: what data scientists seek to accomplish and how can MLOps help?**, arXiv preprint arXiv:2103.08942, 2021

MOCKUS, J. **On Bayesian methods for seeking the extremum.** Em Optimization techniques IFIP technical conference. Springer, Berlin, Heidelberg, p. 400-404, 1975.

MCCULLOCH, W. S., PITTS, W. **A logical calculus of ideas immanent in nervous activity.** Springer, The Bulletin of Mathematical Biophysics, vol. 5, p.115-133, 1943.

MENA, J.; PUJOL, O.; VITRIÀ, J. **Uncertainty-based rejection wrappers or black-box classifiers**. IEEE Access, vol. 8, p. 101721-101746, 2020.

MENG, T., JING, X., YAN, Z., PEDRICZ, W. **A survey on machine learning for data fusion**. Information Fusion, vol. 57, p. 115-129, 2020.

MOAYEDI, H., MOSAVI, A. **An innovative metaheuristic strategy for solar energy management through a neural networks framework**. Energies, vol. 14, no. 4, p. 1196, 2021

MOLINA, D., MORENO GARCIA, F., HERRERA, F. **Analysis among winners of different ieee cec competitions on real-parameters optimization: Is there always improvement?**. Em IEEE Congress on Evolutionary Computation (CEC), São Sebastião, Espanha. p. 805-812, Julho de 2017.

MOLINA, D., POYATOS, J., DEL SER, J., GARCÍA, S., HUSSAIN, A., HERRERA, F. **Comprehensive taxonomies of nature and bio-inspired**

optimization: inspiration versus algorithmic behavior, critical analysis recommendations. Cognitive Computation, vol. 12, no. 5, p. 897-939., 2020

MONTGOMERY, D. C., JENNINGS, C. L., M. Kulahci. **Introduction to time series analysis and forecasting**. John Wiley & Sons, New Jersey, EUA, vol. 2, 2015.

MOON, J., MOSSAIN, M. B., CHON, K. H. **AR and ARMA model order selection for time-series modeling with ImageNet classification**. Signal Processing, vol. 183, 2021.

MHASKAR, H.; LIAO, Q.; POGGIO, T. **When and why are deep networks better than shallow ones?** Em Proceedings of the AAAI Conference on Artificial Intelligence. São Francisco, EUA, vol. 31, no. 1, Fevereiro de 2017.

MITCHELL, T. **Machine learning**, Publicado por McGraw-Hill, Nova York, EUA, 1997.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLERMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. **Human-level control through deep reinforcement learning**, Nature, vol. 518, p. 529-533, 2015.

NGUYEN, H. D.; TRAN, K. P.; THOMASSEY, S.; HAMAD, M. **Forecasting and anomaly detection approaches using LSTM and LSTM autoencoder techniques with the applications in supply chain management**. International Journal of Information Management, vol. 57, 2021.

NOGUEIRA, F. **Bayesian Optimization: Open source constrained global optimization tool for Python**. GitHub. See https://github.com/fmfn/BayesianOptimization, 2014.

OHNSMAN, A. F. **Arizona governor bans self-driving Ubers after pedestrian fatality**, 26 março 2018. Disponível em: <https://www.forbes.com/sites/alanohnsman/2018/03/26/arizona-governor-bans-self-driving-ubers-after-pedestrian-fatality/#5864e17076ab>. Acessado em 17 de maio de 2021.

OPERADOR NACIONAL DO SISTEMA ELÉTRICO. **ONS home page**. Disponível em: http://www.ons.org.br/Paginas/resultados-da-operacao/historico-da-operacao/curva_carga_horaria.aspx. Acessado em 02 janeiro 2021.

OZANICH, E.; GERSTOFT, P.; NIU, H. **A feed forward neural network for direction-of-arrival estimation**. The Journal of the Acoustical Society of America, vol. 147, no. 3, p. 2035-2048, 2020.

PAGANI, S., MANOJ, P. S., JANTSCH, A., HENKEL, J. **Machine learning for power, energy, and thermal management on multicore processors**: A survey. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 1, p. 101-116, 2018.

PAO, Y. H., PARK, G. H., SOBAJIC, D. J. **Learning and generalization characteristics of the random vector functional-link net**. Neurocomputing, vol. 6, no. 2, p. 163-180, 1994.

PANESAR, A. **Machine learning and AI for healthcare**. Apress, Coventry, Reino Unido, p. 1-73, 2019.

PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., DUCHESNAY, E. **Scikit-learn: Machine learning in Python**. the Journal of machine Learning research, vol. 12, p. 2825-2830, 2011.

PELIKAN, M., GOLDBERG, D. E., CANTÚ-PAZ, E. **BOA: The Bayesian optimization algorithm**. Em Proceedings of the genetic and evolutionary computation conference GECCO-99, Orlando, EUA. vol. 1, p. 525-532, julho de 1999.

POHLMAN, J. T.; LEITNER, D. W. **A comparison of ordinary least squares and logistic regression**, Ohio Journal of Science, vol. 103, p. 118-125., 2003.

PUDIL, P., NOVOVICOVÁ, J., KITTLER, J. **Floating search methods in feature selection**. Pattern recognition letters, vol. 15, no. 11, p. 1119-1125, 1994

QARNAIN, S. S., SATTANATHAN, M., SANKARANARAYANAN, B., ALI, S. M. **Analyzing energy consumption factors during coronavirus (COVID-19) pandemic outbreak: a case study of residential society**. Energy Sources, Part A: Recovery, Utilization, and Environmental Effects, p. 1-20, 2020.

QI, Y. **Random forest for bioinformatics.** Em Ensemble machine learning . Springer, Boston, EUA, p. 307-323, 2012.

QUAN, Q.; HAO, Z.; XIFENG, H.; JINGCHUN, L. **Research on water temperature prediction based on improved support vector regression**. Neural Computing and Applications, p. 1-10, 2020.

RAJ, R. J. S.; SHOBANA, S. J.; PUSTOKHINA, I. V.; PUSTOKHIN, D. A.; GUPTA, D.; SHANKAR, K. **Optimal feature selection-based medical image**

classification using deep learning model in internet of medical things. IEEE Access, vol. 8, p. 58006-58017, 2020.

RAMSAUER, H., SCHAFL, B., LEHNER, J., SEIDL, P., WIDRICH, M., ADLER, T., HOCHREITER, S. **Hopfield networks is all you need**. arXiv preprint arXiv:2008.02217, 2020.

REN, Y., SUGANTHAN, P. N., SRIKANTH, N., AMARATUNGA, G. **Random vector functional link network for short-term electricity load demand forecasting**. Information Sciences, vol. 367, p. 1078-1093, 2016.

RIBEIRO, M. H. D. M., MARIANI, V. C., DOS SANTOS COELHO, L. **Multi-step ahead meningitis case forecasting based on decomposition and multi-objective optimization methods**. Journal of Biomedical Informatics, vol. 111, 2020.

ROSENBLATT, F. **The perceptron - a perceiving and recognizing automaton project**, Cornell Aeronautical Laboratory, vol. 85, no. 460, 1957.

ROSENBLATT, F. **Principles of neurodynamics: perceptrons and the theory of brain mechanisms**. Springer, Washington DC, EUA, 1961.

RUMELHART, D. E., HILTON, G. E., WILLIAMS, R. J. **Learning representations by back-propagating errors**. nature, vol. 323, no. 6088, p. 533-536, 1986.

SAMUEL, A. L. **Some studies in machine learning using the game of checkers**, IBM Journal of Research and Development, vol. 3, no. 3, p. 210-229, 1959.

SCHERER, D.; MÜLLER, A.; BEHNKE, S. **Evaluation of pooling operations in convolutional architectures for object recognition**. ICANN, Proceedings of the 20th international conference on Artificial neural Networks: Part III, Berlin, Alemanha, p. 92-101, 2010.

SHAHID, F., ZAMEER, A., MEHMOOD, A., RAJA, M. A. Z. **A novel wavenets long short-term memory paradigm for wind power prediction**. Applied Energy, vol. 269, 2020.

SHARDA, R., PATIl, R. B. **Connectionist approach to time series prediction: an empirical test**. Journal of Intelligent Manufacturing, vol. 3, no. 5, p. 317-323.

SHARIATI, M., MAFIPOUR, M. S., MEHRABI, P., ZANDI, Y., DEHGHANI, D., BAHADORI, A., POI-NGIAN, S. **Application of extreme learning machine (ELM) and genetic programming (GP) to design steel-concrete composite floor**

systems at elevated temperatures. Steel and Composite Structures, vol. 33, no. 3, p. 319-332, 2019.

SKANSI, S. **Introduction to deep learning: from logical calculus to artificial intelligence**. Springer, 2018.

SMITH, Taylor G., et al. **pmdarima: ARIMA estimators for Python**, http://www.alkaline-ml.com/pmdarima [Online; acessado em 31 de agosto de 2021], 2021.

SMYL, S. **A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting**. International Journal of Forecasting, vol. 36, no. 1, p. 75-85, 2020.

SOCHER, R.; HUANG, E. H.; PENNINGTON J.; NG A. Y. MANNING C. D. **Dynamic pooling and unfolding recursive autoencoders for paraphrase detection**. Em 24th International Conference on Neural Information Processing Systems. Granada, Espanha. vol. 24, p. 801–809, 2011.

SRIVASTAVA, T. **11 Important Model Evaluation Metrics for Machine Learning Everyone should know**. Acessado em 28 agosto de 2021, Disponível em https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/, 2019.

STORN, R., PRICE, K. **Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces.** Journal of global optimization, vol. 11 no. 4, p. 341-359, 1997.

STEWART, M. **Predicting Stock Prices with Echo State Networks**, 2020.

TALAGAL, N. **Why MLOps (and not just ML) is your business' new competitive frontier**. Airtrends, 2018.

TANG, L., WU, Y., YU, L. **A non-iterative decomposition-ensemble learning paradigm using RVFL network for crude oil price forecasting**. Applied Soft Computing, vol. 70, p. 1097-1108, 2018.

TEMKENG, S. D., FOFACK, A. D. **A Markov-switching dynamic regression analysis of the asymmetries related to the determinants of US crude oil production between 1982 and 2019**. Petroleum Science, vol. 18, no. 2, p. 679-686, 2021.

THE PANDAS DEVELOPMENT TEAM. **pandas-dev/pandas: Pandas (version latest)**. Zenodo. https://doi.org/10.5281/zenodo.3509134, 2020.

VAGROPOULOS, S. I.; CHOULIARAS, G. I.; KARDAKOS, E. G.; SIMOGLOU, C. K.; BAKIRTZIS, A. G. **Comparison of SARIMAX, SARIMA, modified SARIMA and ANN-based models for short-term PV generation forecasting**. In 2016 IEEE International Energy Conference (ENERGYCON), Ku Leuven, Belgica. p. 1-6, abril de 2016.

VALIPOUR, M. **Long-term runoff study using SARIMA and ARIMA models in the United States**. Meteorological Applications, vol. 22, no. 3, p. 592–598, Fevereiro de 2015.

VAN ECK, N. J., WALTMAN, L. **VOSviewer manual**. Leiden: Univeristeit Leiden, vol. 1, no. 1, p. 1-53, 2013.

VIHAROS, Z. J., JAKAB, R. **Reinforcement learning for statistical process control in manufacturing**, 17th IMEKO TC 10 and EUROLAB Virtual Conference: “Global Trends in Testing, Diagnostics & Inspection for 2030”, p. 225-234, 2020.

VISHWAKARMA, G. K., PAUL, C., ELSAWAH, A. M. **An algorithm for outlier detection in a time series model using backpropagation neural network**. Journal of King Saud University-Science, vol. 32, no. 8, p. 3328-3336, 2020

WANG, F.; ZHANG, H.; LI, K.; LIN, Z.; YANG, J.; SHEN, X. **A hybrid particle swarm optimization algorithm using adaptive learning strategy**. ScienceDirect, Information Sciences, vol. 436-437, p. 162-177, 2018.

WANG, G. G.; DEB, S.; COELHO, L. D. S. **Elephant herding optimization**, IEEE 3rd International Symposium on Computational and Business Intelligence (ISCBI), Bali, Indonesia, 2015.

WANG, J., GAO, R., XIE, Y. **Two-sample test using projected wasserstein distance: breaking the curse of dimensionality**. arXiv preprint arXiv:2010.11970, 2020.

WANG, Q., WANG, S., SHI, R., LI, Y. **A power transformer fault diagnosis method based on random vector functional-link neural network**. Mathematical Problems in Engineering, 2021.

WANG, Y.; LI, B.; WEISE, T.; WANG, T.; WANG, J.; YUAN, B.; TIAN, Q. **Self-adaptive learning based particle swarm optimization.** Elsevier, Information Sciences – Informatics and Computer Science, Intelligent Systems, Applications: An International Journal, vol. 181, no. 20, p. 4515-4538, 2011.

WANG, Y.; NI, X. S. **A XGBoost risk model via feature selection and Bayesian hyper-parameter optimization**. arXiv preprint arXiv:1901.08433, 2019.

WEI, N.; LI, C.; DUAN, J.; LIU, J.; ZENG, F. **Daily natural gas load forecasting based on a hybrid deep learning model**. Energies, vol. 12, no. 2, p. 218, 2019.

WIENER, N.; WILEY, J. **Cybernetics**. Sons, Nova York, EUA, p. 19, 1948.

WILLIAMS, C. **Dimensions from digital science**. Insights, vol. 31, 2018.

WU, J., MIU, F., LI, T. **Daily crude oil price forecasting based on improved CEEMDAN, SCA, and RVFL: a case study in WTI oil market**. Energies, vol. 13, no. 7, p. 1852, 2020.

YAO, X**. Evolving artificial neural networks** Proceedings of the IEEE, vol. 87, no. 9, p. 1423-1447, 1999.

ZHANG, C., JIANG, F., WANG, S., SHANG, W. **A novel hybrid approach with a decomposition method and the RVFL model for crude oil price prediction**. Em 2020 IEEE International Conference on Big Data (Big Data), Conferência virtual, p. 4446-4449, 2020.

ZHANG, G. P. **Time series forecasting using a hybrid ARIMA and neural network model**. Neurocomputing, vol. 50, p. 159-175, 2003.

ZHANG, L., SUGANTHAN, P. N. **Visual tracking with convolutional random vector functional link network**. IEEE transactions on Cybernetics, vol. 47, no. 10, p. 3243-3253, 2016

ZHANG, M. **Forbes: Google photos tags two African-Americans as gorillas through facial recognition software**, 01 julho 2015. Disponível em: <https://www.forbes.com/sites/mzhang/2015/07/01/google-photos-tags-two-african-americans-as-gorillas-through-facial-recognition-software/#2c77df20713d>. Acessado em 8 de junho de 2021.

ZHANG, M., LI, J., LI, Y., XU, R. **Deep learning for short-term voltage stability assessment of power systems**. IEEE Access, vol. 9, p. 29711-29718, 2021.

ZHANG, P., JIANG, X., HOLT, G. M., LAPTEV, N. P., KOMURLU, C., GAO, P., YU, Y. **Self-supervised learning for fast and scalable time series hyper-parameter tuning**. arXiv preprint arXiv:2102.05740, 2021.

ZHANG, S.; LANG, Z. Q. **Orthogonal least squares based fast feature selection for linear classification**. arXiv preprint arXiv:2101.08539, 2021.

ZHANG, Y., WU, J., CAI, Z., DU, B., PHILIP, S. Y. **An unsupervised parameter learning model for RVFL neural network**. Neural Networks, vol. 112, p. 85-97, 2019.

ZHANG, Y.; XU, X. **Fe-based superconducting transition temperature modeling through Gaussian process regression**. Journal of Low Temperature Physics, vol. 202, no. 1, p. 205-218, 2021.

ZHANG, Y.; YANG, H.; CUI, H.; CHEN, Q. **Comparison of the ability of ARIMA, WNN and SVM models for drought forecasting in the Sanjiang Plain, China**. Natural Resources Research, vol. 29, no. 2, p. 1447-1464, 2020.

ZHANG, Z., ZOHREN, S., ROBERTS, S. **Deep reinforcement learning for trading**. The Journal of Financial Data Science, vol. 2, no. 2, p. 25-40, 2020.

ZHAO, Y.; HU, Q.; XU, J.; LI, B.; HUANG, G.; PAN, Y. **A robust extreme learning machine for modeling a small-scale Turbojet engine**. ScienceDirect, Applied Energy. vol. 218, p. 22-35, 2018.

ZHU, Q.Y., QIN, A.K., SUGANTHAN, P.N., HUANG, G.B.: **Evolutionary extreme learning machine**. Pattern Recognition. vol. 38, no. 10, p. 1759–1763, 2005

ZOU, W.; YAO, F.; ZHANG, B.; GUAN, Z. **Back propagation convex extreme learning machine**. Springer, Proceedings of Experiments in Linguistic Meaning, Singapura, vol. 9, p. 259-272, 2016.

APÊNDICE A

Os valores abaixo mostram os valores obtidos para seus hiperparâmetros, e que foram usados na fase de testes. As tabelas mostram os resultados por diferentes métodos de aprendizado de máquina que foram usados nesta tese.

- A Tabela A.1 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ARIMA com a otimização DE.

	Tempo de Execução	RMSLE	*P*	*Q*	*D*
Sem FS	187,08s	0,061867	1	0	0
Com FS	149,88s	0,061867	2	0	2

Tabela A.1: Resultados para ARIMA, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.2 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ARIMA com a otimização Bayesiana.

•	Tempo de Execução	RMSLE	*P*	*Q*	*D*
Sem FS	100,37	0,061867	1	0	0
Com FS	55,35s	0,061867	2	0	2

Tabela A.2: Resultados para ARIMA, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.3 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo DRVFL com a otimização DE.

	Tempo de Execução	RMSLE	Núm. neurônios camada entrada	Núm. neurônios camada 1	Núm. neurônios camada 2	Núm. neurônios camada saída
Sem FS	10,97s	0,006198	99	96	98	36
Com FS	15,95s	0,006111	76	21	35	68

Tabela A.3: Resultados para ARIMA, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.4 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo DRVFL com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. neurônios camada entrada	Núm. neurônios camada 1	Núm. neurônios camada 2	Núm. neurônios camada saída
Sem FS	7,35s	0,006197	99	64	78	7
Com FS	10,48s	0,061867	91	59	79	84

Tabela A.4: Resultados para ARIMA, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.5 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ELM com a otimização DE.

	Tempo de Execução	RMSLE	Núm. de camadas ocultas	*alpha*	Tamanho do *rbf*	Função ativação
Sem FS	1,60s	0,00739	91	0,97742	0,34296	*inv_multiquadric*
Com FS	6,81s	0,00832	95	0,75924	0,76323	*tanh*

Tabela A.5: Resultados para ELM, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.6 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ELM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas	*alpha*	Tamanho do *rbf*	Função ativação
Sem FS	9,17s	0,007226	93	1,0	1,0	*inv_multiquadric*
Com FS	13,28s	0,007619	94	0,8	0,3	*sigmoid*

Tabela A.6: Resultados para ELM, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.7 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ESN com a otimização DE.

	Tempo de Execução	RMSLE	*Spectral radius*	Num. receptor	*Input Scaling*	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Sem FS	5,78s	0,00718	0,56287	98	0,88921	0,07288	0,29940	0,41741	0
Com FS	13,63s	0,00621	0,73040	90	0,76892	0,12213	0,77701	0,34473	0

Tabela A.7: Resultados para ESN, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.8 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo ESN com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*Spectral radius*	Num. receptor	Input Scaling	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Sem FS	5,07s	0,00690	0,64904	83	1,0	0,0	0,29940	-0,3549	0
Com FS	12,58s	0,00655	0,82734	91	0,96573	0,38041	0,52573	0,37595	0

Tabela A.8: Resultados para ESN, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.9 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo FFNN com a otimização DE.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	5083,47s	0,011763	864
Com FS	4944,65s	0,011909	684

Tabela A.9: Resultados para FFNN, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.10 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo FFNN com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	5355,75s	0,01330	763
Com FS	4477,64s	0,01102	955

Tabela A.10: Resultados para FFNN, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.11 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo LightGBM com a otimização DE.

	Tempo de Execução	RMSLE	Max *depth*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Sem FS	14,03s	0,1228	52	0,185366	35	*gbdt*	1	0,723425
Com FS	18,09s	0,01197	7	0,206477	29	*gbdt*	9	0,992357

Tabela A.11: Resultados para LightGBM, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.12 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo LightGBM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Max *depth*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Sem FS	8,87s	0,01194	82	0,209297	18	*gbdt*	2	0,829098
Com FS	10,93s	0,01218	83	0,265536	18	*gbdt*	4	0,981621

Tabela A.12: Resultados para LightGBM, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.13 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo LSTM com a otimização DE.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	3576,00s	0,031810	91,02271
Com FS	3575,82s	0,035529	91,88013

Tabela A.13: Resultados para LSTM, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.14 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo LSTM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	2498,36s	0,032086	94,64332
Com FS	1992,82s	0,032597	88,33317

Tabela A.14: Resultados para LSTM, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.15 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo OLS com a otimização DE.

	Tempo de Execução	RMSLE	Uso dos valores interceptados?	Número de *jobs*
Sem FS	0,09s	0,010752	Não	36
Com FS	5,24s	0,015537	Não	36

Tabela A.15: Resultados para OLS, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.16 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo OLS com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Uso dos valores interceptados?	Número de *jobs*
Sem FS	5,49s	0,010752	Não	95
Com FS	11,00s	0,015537	Não	95

Tabela A.16: Resultados para OLS, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.17 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo QRF com a otimização DE.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	61,35s	0,02008	27	24	5	4	0,00680	9
Com FS	65,37s	0,02029	28	20	5	2	0,02890	8

Tabela A.17: Resultados para QRF, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.18 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo QRF com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	14,77s	0,01993	135	33	6	4	0,022003	7
Com FS	19,49s	0,02003	165	23	8	4	0,024303	9

Tabela A.18: Resultados para QRF, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.19 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo RF com a otimização DE.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	22,04s	0,02023	22	92	4	6	0,00667	9
Com FS	22,91s	0,02064	19	25	7	5	0,00931	9

Tabela A.19: Resultados para RF, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.20 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo RF com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	6,62s	0,02070	24	76	4	1	0,077164	9
Com FS	10,74s	0,02089	35	81	2	1	0,075025	8

Tabela A.20: Resultados para RF, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.21 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo SARIMA com a otimização DE.

	Tempo de Execução	RMSLE	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Sem FS	384,30s	0,01096	1	1	1	1	1	1	2
Com FS	345,09s	0,01774	2	1	2	1	1	1	4

Tabela A.21: Resultados para SARIMA, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.22 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo SARIMA com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Sem FS	69,62s	0,01510	0	1	0	2	1	2	4
Com FS	59,13s	0,01090	0	1	2	0	1	0	3

Tabela A.22: Resultados para SARIMA, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.23 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo SVR com a otimização DE.

	Tempo de Execução	RMSLE	Núm. *Kernels*	gama	coeficiente	tolerância	Regularizador	*epsilon*
Sem FS	1,27s	0,01917	1	0,00620	0,075042	0,004655	0,487598	0,030244
Com FS	6,30s	0,02047	1	0,00390	0,024737	0,005808	0,445063	0,026716

Tabela A.23: Resultados para SVR, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.24 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo SVR com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. *Kernels*	gama	coeficiente	tolerância	Regularizador	*epsilon*
Sem FS	7,03s	0,02612	1	0,00997	0,019631	0,008395	0,25797	0,007630
Com FS	15,33s	0,02518	1	0,01	0,001	0,0001	0,5	0,001

Tabela A.24: Resultados para SVR, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.25 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo XGBoost com a otimização DE.

	Tempo de Execução	RMSLE	Max. *Depth*	Menor peso por *children*	*Learning rate*	Subamostra	Subamostra de colunas
Sem FS	17,93s	0,01178	78	2	0,249159	0,857339	0,795080
Com FS	34,10s	0,01420	44	4	0,215416	0,459857	0,694140

Tabela A.25: Resultados para XGBoost, com otimização DE para série temporal de consumo da região Sul do Brasil

- A Tabela A.26 mostra os valores obtidos para a série temporal de consumo da região sul do Brasil usando o modelo XGBoost com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Max. d*epth*	Menor peso por *children*	*Learning rate*	Subamostra	Subamostra de colunas
Sem FS	9,50s	0,01070	32	3	0,198432	0,7375466	0,798851
Com FS	15,75s	0,01135	29	4	0,1	0,582509	0,8

Tabela A.26: Resultados para XGBoost, com otimização Bayesiana para série temporal de consumo da região Sul do Brasil

- A Tabela A.27 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ARIMA com a otimização DE.

	Tempo de Execução	RMSLE	*P*	*Q*	*D*
Sem FS	246,17s	0,061644	2,40067	0,19188	1,00713
Com FS	495,11s	0,12502	2,90293	1,351071	1,77612

Tabela A.27: Resultados para ARIMA, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.28 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ARIMA com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*P*	*Q*	*D*
Sem FS	93,14s	0,061644	2,83970	0,001	1,64727
Com FS	118,60s	0,12502	2,05404	1,999	1,447224

Tabela A.28: Resultados para ARIMA, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.29 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo DRVFL com a otimização DE.

	Tempo de Execução	RMSLE	Núm. neurônios camada entrada	Núm. neurônios camada 1	Núm. neurônios camada 2	Núm. neurônios camada saída
Sem FS	166,81s	0,017658	12	12	59	65
Com FS	180,95s	0,018250	19	22	28	75

Tabela A.29: Resultados para ARIMA, com otimização DE série temporal de consumo na Polônia

- A Tabela A.30 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo DRVFL com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. neurônios camada entrada	Núm. neurônios camada 1	Núm. neurônios camada 2	Núm. neurônios camada saída
Sem FS	42,07s	0,016904	39	100	3	92
Com FS	49,90s	0,018210	2	3	76	96

Tabela A.30: Resultados para ARIMA, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.31 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ELM com a otimização DE.

	Tempo de Execução	RMSLE	Número de camadas ocultas	*alpha*	Tamanho do *rbf*	Função ativação
Sem FS	4,45s	0,017667	87	0,314449	0,339772	*inv_multiquadric*
Com FS	11,39s	0,018832	59	0,716720	0,475565	*sine*

Tabela A.31: Resultados para ELM, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.32 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ELM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas	*alpha*	Tamanho do *rbf*	Função ativação
Sem FS	6,63s	0,018315	79	0,497462	0,573864	*inv_multiquadric*
Com FS	18,31s	0,018642	32	0,8	0,3	*inv_tribas*

Tabela A.32: Resultados para ELM, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.33 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ESN com a otimização DE.

	Tempo de Execução	RMSLE	*Spectral radius*	Num. reservatórios	*Input Scaling*	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Sem FS	13,21	0,01886	0,29950	55	0,13222	0,33911	0,64350	0,21138	0
Com FS	21,04s	0,01716	0,70718	33	0,62819	0,27353	0,77743	0,25054	0

Tabela A.33: Resultados para ESN, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.34 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo ESN com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*Spectral radius*	Num. reservatórios	*Input Scaling*	*Input shift*	*Teacher Scaling*	*Teacher Shift*	*Teacher forcing*
Sem FS	11,44s	0,01796	0,40366	12	0,02565	0,16223	0,08283	0,34581	1
Com FS	15,56s	0,01738	0,84447	33	0,80565	0,25849	0,96762	0,18077	0

Tabela A.34: Resultados para ESN, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.35 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo FFNN com a otimização DE.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	4494,07s	0,026641	824,92621
Com FS	5686,22s	0,017906	586,01747

Tabela A.35: Resultados para FFNN, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.36 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo FFNN com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	5194,57s	0,01702	936,23516
Com FS	4390,13s	0,01829	941,64580

Tabela A.36: Resultados para FFNN, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.37 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo LightGBM com a otimização DE.

	Tempo de Execução	RMSLE	*Max depth*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Sem FS	28,31s	0,02055	56	0,137753	97	*gbdt*	7	0,383847
Com FS	36,15s	0,02134	86	0,560418	4	*gbdt*	4	0,532119

Tabela A.37: Resultados para LightGBM, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.38 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo LightGBM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*Max deep*	*Learning rate*	Núm. folhas	*Boosting*	*Bagging Freq.*	*Bagging Fraction*
Sem FS	11,60s	0,02054	96	0,589549	6	*dart*	1	0,869361
Com FS	15,71s	0,01837	83	0,212914	19	*gbdt*	3	0,666804

Tabela A.38: Resultados para LightGBM, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.39 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo LSTM com a otimização DE.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	4903,34s	0,026841	85,30214
Com FS	1340,14s	0,023300	97,10991

Tabela A.39: Resultados para LSTM, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.40 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo LSTM com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Número de camadas ocultas
Sem FS	3709,20s	0,026536	96,04321
Com FS	3071,22s	0,024001	92,63946

Tabela A.40: Resultados para LSTM, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.41 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo OLS com a otimização DE.

	Tempo de Execução	RMSLE	Uso dos valores interceptados?	Número de *jobs*
Sem FS	0,13s	0,017613	Sim	36
Com FS	7,38s	0,018871	Sim	36

Tabela A.41: Resultados para OLS, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.42 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo OLS com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Uso dos valores interceptados?	Número de *jobs*
Sem FS	5,18s	0,017613	Sim	60
Com FS	14,12s	0,018871	Sim	60

Tabela A.42: Resultados para OLS, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.43 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo QRF com a otimização DE.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	112,44s	0,02468	8	33	6	9	0,06568	9
Com FS	95,79s	0,02370	11	62	9	8	0,00680	9

Tabela A.43: Resultados para QRF, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.44 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo QRF com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	21,91s	0,02365	13	86	6	7	0,010292	9
Com FS	28,99s	0,02370	5	45	7	5	0,03584	8

Tabela A.44: Resultados para QRF, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.45 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo RF com a otimização DE.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	63,72s	0,02400	7	59	9	5	0,02862	9
Com FS	45,32s	0,02400	9	50	4	2	0,07582	5

Tabela A.45: Resultados para RF, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.46 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo RF com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. Estimador	Max *depth*	Min. Amostras para dividir	Min. Amostras de folhas	Min *weight*	Max. Folhas por ramo
Sem FS	11,87s	0,02348	7	86	6	7	0,010292	9
Com FS	15,57s	0,02355	7	56	8	7	0,010285	8

Tabela A.46: Resultados para RF, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.47 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo SARIMA com a otimização DE.

	Tempo de Execução	RMSLE	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Sem FS	1784,51s	0,020257	1	0	1	1	0	1	3
Com FS	642,42s	0,021680	2	0	0	2	0	2	4

Tabela A.47: Resultados para SARIMA, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.48 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo SARIMA com a otimização Bayesiana.

	Tempo de Execução	RMSLE	*p*	*d*	*q*	*P*	*D*	*Q*	Núm. período
Sem FS	406,77s	0,02084	1	0	1	0	0	1	4
Com FS	132,87s	0,02162	2	0	0	2	0	2	4

Tabela A.48: Resultados para SARIMA, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.49 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo SVR com a otimização DE.

	Tempo de Execução	RMSLE	Núm. *Kernels*	*gama*	coeficiente	tolerância	Regularizador	*epsilon*
Sem FS	2,37s	0,02516	1	0,00910	0,059598	0,004305	0,305519	0,028450
Com FS	10,42s	0,02162	1	0,00501	0,032203	0,007741	0,479160	0,009556

Tabela A.49: Resultados para SVR, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.50 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo SVR com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Núm. *Kernels*	*gama*	coeficiente	tolerância	Regularizador	*epsilon*
Sem FS	6,76s	0,02465	1	0,007881	0,048387	0,006613	0,11060	0,016131
Com FS	18,37s	0,02627	1	0,001	0,01	0,01	0,5	0,001

Tabela A.50: Resultados para SVR, com otimização Bayesiana para série temporal de consumo na Polônia

- A Tabela A.51 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo XGBoost com a otimização DE.

	Tempo de Execução	RMSLE	Max. *depth*	Menor peso por *children*	*Learning rate*	Subamostra	Subamostra de colunas
Sem FS	62,20s	0,01608	20	4	0,119497	0,687534	0,785308
Com FS	78,73s	0,01694	33	3	0,102218	0,549257	0,649768

Tabela A.51: Resultados para XGBoost, com otimização DE para série temporal de consumo na Polônia

- A Tabela A.52 mostra os valores obtidos para a série temporal de consumo na Polônia usando o modelo XGBoost com a otimização Bayesiana.

	Tempo de Execução	RMSLE	Max. *depth*	Menor peso por c*hildren*	*Learning rate*	Subamostra	Subamostra de colunas
Sem FS	17,42s	0,01758	21	5	0,1	0,754834	0,8
Com FS	34,14s	0,02042	19	3	0,415264	0,704153	0,728707

Tabela A.52: Resultados para XGBoost, com otimização Bayesiana para série temporal de consumo na Polônia