PONTIFICIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

(PUCRS)

MPA em Inovação e o Novo Setor Público

REDES NEURAIS APLICADAS NA PREVISÃO DE RECEITA DE ICMS DO RIO GRANDE DO SUL

Gabriel Zuanazzi Dornelles

Professor Orientador:

Prof Dr. Rodrigo da Rosa Righi

Porto Alegre, setembro de 2023.

Resumo: Este artigo aborda a importância da previsão de séries temporais e sua aplicação na política pública do Estado do Rio Grande do Sul. O modelo proposto neste artigo é baseado em Redes Neurais de Memória de Longo e Curto Prazo (LSTM) e é aplicado à previsão do ICMS no Rio Grande do Sul. Ele é comparado a modelos clássicos da econometria (SARIMAX e Estrutural) em termos de desempenho. O modelo LSTM desenvolvido é apresentado, detalhando sua construção e desempenho em comparação com modelos clássicos. A transparência e a possibilidade de aprimoramento são incentivadas por meio do compartilhamento do código aberto. Demonstra-se que o modelo utilizando redes neurais apresentou acurácia superior aos outros dois modelos testados para o período de 14 previsões de um passo a frente (T+1), com RMSE de 170,62 e desvio percentual acumulado de 0,0253%, apresentando uma opção viável, gratuita e acessível como ferramenta complementar.

Abstract: This article addresses the significance of time series forecasting and its application in the public policy of the State of Rio Grande do Sul. The model proposed in this article is based on Long Short-Term Memory (LSTM) Neural Networks and is applied to forecast the ICMS (Tax on Circulation of Goods and Services, Interstate and Intermunicipal Transportation, and Communication) in Rio Grande do Sul. It is compared to classical econometric models (SARIMA and Structural) in terms of performance. The developed LSTM model is presented, detailing its construction and performance in comparison to classical models. Transparency and the potential for improvement are encouraged through the sharing of open-source code. It is demonstrated that the neural network-based model exhibited superior accuracy compared to the other two tested models for a 14-step ahead forecasting horizon (T+1), with RMSE of 170.62 and accumulated percentage deviation of 0.0253%, offering a viable, free, and accessible option as a complementary tool.

Palavras-chave: Séries Temporais. Redes Neurais. ICMS. LSTM. Python.

1. Introdução

Uma série temporal é uma sequência de observações feitas sequencialmente em um período de tempo (BOX, JENKINS). Apesar de definição concisa, a previsão de séries temporais representa um campo de pesquisa intrinsecamente rico e desafiador, caracterizado pela complexidade subjacente aos seus estudos. Diariamente, profissionais dedicados a essa área enfrentam tarefas diversas em diferentes temáticas, envolvendo desde a estimativa de condições climáticas, análise de tendências de mercado, projeção de volumes de vendas e até mesmo a previsão da propagação de vírus em períodos de pandemia. Essas demandas, exigentes em natureza, impulsionam a busca por métodos sofisticados e técnicas avançadas de análise.

Neste contexto, a política pública do Estado do Rio Grande do Sul é fortemente influenciada por estas previsões. A Constituição Brasileira, bem como a Lei de Responsabilidade Fiscal (LRF) e outros regramentos infraconstitucionais, determinam que os entes federativos realizem o adequado planejamento orçamentário, publicizados através de suas leis orçamentárias. Os diferentes instrumentos legais, que materializam este orçamento público, possuem um ponto de interesse para este artigo: as despesas futuras são fixadas enquanto as receitas futuras são estimadas. Justamente neste último ponto, a estimativa de receitas, que o desenvolvimento e aplicação de modelos preditivos cada vez mais robustos se faz necessário na perspectiva fiscal do Rio Grande do Sul.

As estimativas das receitas estaduais influenciam diretamente na saúde fiscal do Estado. A partir desta definição, são calculadas, por exemplo, a meta de resultado primário do ano, o resultado orçamentário e a variação da dívida prevista. Ademais, este impacto na execução orçamentária desdobra na necessidade ou não de contingenciamento de recursos financeiros por parte dos

Poderes e Órgãos com autonomia financeira. O próprio estabelecimento do orçamento de todos os órgãos da administração pública, bem como os mínimos constitucionais para saúde e educação, depende desta previsão, o que, em última instância, influenciará na qualidade do gasto final em favor do cidadão. Por este fato, a LRF determina que as previsões de arrecadação devem atender um rigor técnico, constituindo requisitos essenciais da responsabilidade fiscal:

> Art. 11. Constituem requisitos essenciais da responsabilidade na gestão fiscal a instituição, previsão e efetiva arrecadação de todos os tributos da competência constitucional do ente da Federação.
> ...
> Art. 12. As previsões de receita observarão as normas técnicas e legais, considerarão os efeitos das alterações na legislação, da variação do índice de preços, do crescimento econômico ou de qualquer outro fator relevante e serão acompanhadas de demonstrativo de sua evolução nos últimos três anos, da projeção para os dois seguintes àquele a que se referirem, e da metodologia de cálculo e premissas utilizadas.

De acordo com o Estudo do Banco Central Europeu (LEAL, 2007), embora a área de previsões fiscais seja vista predominantemente como uma arte do que pura ciência, é consenso que esta deve ser lastreada por métodos transparentes e procedimentos claros. Somente deste modo agentes políticos, participantes do mercado e o público em geral podem ter uma informação precisa sobre a evolução orçamentária. Tais previsões são a principal ferramenta para os formuladores de políticas tomarem decisões fiscais adequadas, para o público avaliar a necessidade de tais decisões e, posteriormente, avaliar o sucesso de suas implementações. Destarte, este estudo destaca a necessidade de obtenção de técnicas de previsões econômicas e fiscais cada vez mais consistentes e acuradas, com ênfase na transparência metodológica.

Este artigo tem como objetivo apresentar o modelo de previsão da arrecadação do ICMS (Imposto sobre Circulação de Mercadorias e Prestação de Serviços de Transporte Interestadual e Intermunicipal e de Comunicação) construído a partir de uma estrutura de Redes Neurais de Memória de Longo e Curto Prazo (LSTM). Ademais detalhar sua construção e desenvolvimento, bem como avaliar os resultados alcançados durante seu período de efetiva utilização no Tesouro do Estado do Rio Grande do Sul (agosto 2021- dezembro2022), comparando com dois outros modelos clássicos da econometria (SARIMAX e Estrutural). Por fim, com intuito de posicionar tal publicação como um ponto de partida e não um trabalho com fim em si, garantindo a devida transparência e possibilidade de aprimoramentos por terceiros, disponibiliza-se o código aberto desenvolvido no Anexo A.

2. Fundamentação Teórica

Esta seção apresenta os conceitos que serão importantes para entender o funcionamento do modelo proposto desde conceitos mais genéricos como a definição de uma série temporal até a caracterização específica das redes LSTM. Neste interstício, apresenta-se também as Redes Neurais Artificiais e seu subtipo Redes Neurais Recorrentes.

2.1 Série Temporal

Uma série temporal é uma sucessão de valores de uma determinada variável observada em intervalos regulares de tempo. A variável de controle é o tempo e as séries temporais são ordenadas cronologicamente (SILVA; SILVA, 1999). Conforme Hyndman (2018), as séries temporais muitas vezes exibem padrões complexos, como tendências, flutuações sazonais e variações aleatórias. Essas características demandam técnicas analíticas especiais para modelagem e previsão, uma vez que os métodos tradicionais de análise de dados nem sempre são aplicáveis devido à dependência temporal intrínseca dos pontos de dados. Segundo Box e Jenkins (1976), existem 5 importantes

problemas práticos ou etapas a serem levadas em conta na análise de séries temporais: Identificação do modelo, estimação dos parâmetros, diagnóstico do modelo, previsão e, por fim, controle. Compreender e modelar adequadamente esses padrões é fundamental para realizar previsões precisas e embasar decisões informadas em diversas áreas.

2.2 Rede Neural Artificial

Uma rede neural artificial (RNA) é a parte de um sistema de computação projetada para simular a maneira como o cérebro humano analisa e processa informações. É a base da inteligência artificial (IA) e resolve problemas que seriam impossíveis ou difíceis para os padrões humanos ou estatísticos. As RNAs têm recursos de autoaprendizagem que lhes permitem produzir melhores resultados à medida que mais dados se tornam disponíveis. Segundo Haykin (2001), para garantirem um desempenho relevante, as redes neurais utilizam uma interligação maciça de células computacionais simples, denominadas “neurônios” ou unidades de processamento.

Uma RNA possui centenas ou milhares de neurônios artificiais chamados unidades de processamento, que são interconectados por nós. Essas unidades de processamento são compostas por unidades de entrada e saída. As unidades de entrada recebem várias formas e estruturas de informação com base em um sistema de ponderação interno, sendo que a rede neural tenta aprender sobre as informações apresentadas para produzir um relatório de saída. Assim como os humanos precisam de regras e diretrizes para chegar a um resultado ou saída, as RNAs também usam um conjunto de regras de aprendizado chamado retro propagação (backpropagation), uma abreviatura para propagação reversa de erro, para aperfeiçoar seus resultados de saída.

Uma RNA inicialmente passa por uma fase de treinamento, onde aprende a reconhecer padrões em dados, seja visual, auditiva ou textualmente. Durante

essa fase supervisionada, a rede compara a saída real produzida com o que deveria produzir - a saída desejada. A diferença entre os dois resultados é ajustada usando retro propagação (backpropagation). Isso significa que a rede trabalha para trás, indo da unidade de saída para as unidades de entrada para ajustar o peso de suas conexões entre as unidades até que a diferença entre o resultado real e o desejado produza o menor erro possível.

2.3 Redes Neural Recorrente

Segundo (Martinez & Chen, 2021) uma rede neural recorrente (RNN) é um tipo de rede neural artificial que usa dados sequenciais ou dados de série temporal. Esses algoritmos de aprendizado profundo são comumente usados para problemas ordinais ou temporais, como tradução de linguagem, processamento de linguagem natural (NLP), reconhecimento de fala e legendagem de imagens; eles são incorporados a aplicativos populares, como Siri, pesquisa por voz e Google Translate. Como as redes neurais de feedforward e convolucionais (CNNs), as redes neurais recorrentes utilizam dados de treinamento para aprender. Elas se distinguem por sua “memória”, pois recebem informações de entradas anteriores para influenciar a entrada e a saída atuais. Enquanto as redes neurais profundas tradicionais presumem que as entradas e saídas são independentes umas das outras, a saída das redes neurais recorrentes depende dos elementos anteriores dentro da sequência. Embora eventos futuros também sejam úteis para determinar a saída de uma determinada sequência, as redes neurais recorrentes unidirecionais não conseguem levar em conta esses eventos em suas previsões.

2.4 Rede Long Short Term Memory

Redes Neurais de Memória de Longo e Curto Prazo (LSTM) são um tipo de Rede Neural Recorrente em Aprendizado Profundo, apresentado por Hochreiter e Schmiduber em 1997, sendo desenvolvido especificamente para o uso do tratamento de problemas de predição sequencial. Por exemplo: previsão

do tempo, previsão do mercado de ações, recomendação de produto, geração de texto / imagem / manuscrito e tradução de textos. Entretanto, é importante destacar que tais tipos de redes neurais demandam mais recursos computacionais de processamento e memória devido a sua estrutura em cascata dependente que causam gargalos para realizar a operação de inferência (Wang et al., 2019; Gao et al., 2020).

Como outras redes neurais, elas contêm neurônios para realizar cálculos; no entanto, para as LSTM, eles são frequentemente chamados de células de memória ou simplesmente células. Essas células contêm pesos e portas; as portas são a característica distintiva dos modelos LSTM. Existem 3 portas dentro de cada célula. A porta de entrada, a porta de esquecimento e a porta de saída.

Os modelos LSTM são capazes de olhar para trás em dados e decisões anteriores e tomar decisões a partir deles. Também usam o mesmo processo para fazer suposições / previsões fundamentadas sobre o que pode acontecer. Portanto, sendo alimentado com dados sequenciais, encontrará tendências e as usará para prever os resultados futuros.

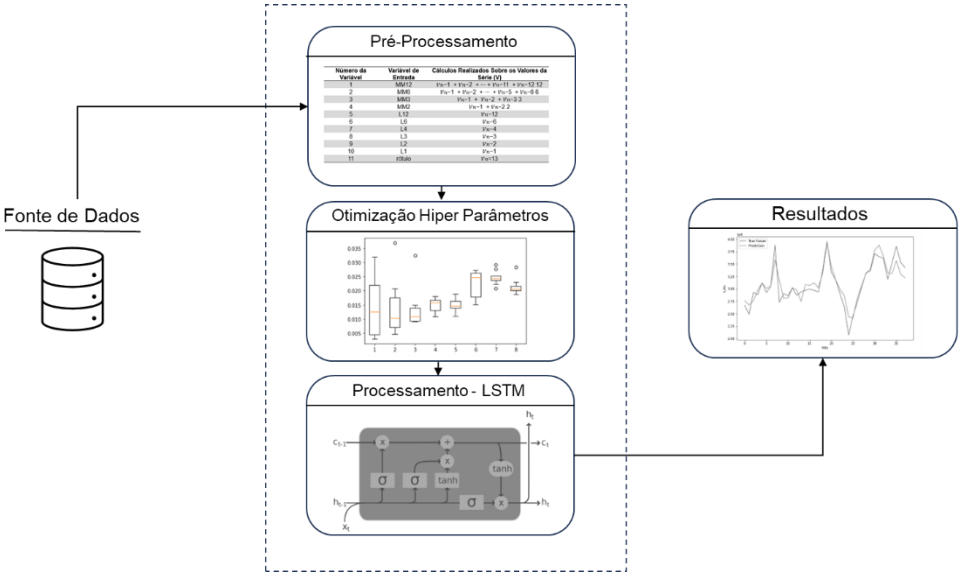
3. Modelo Computacional – Modelo LSTM

A metodologia utilizada nesse trabalho envolve modelos de Redes Neurais Recorrentes do tipo LSTM, devido à eficácia histórica de seus resultados em relação à previsão de séries temporais não lineares e a exploração e extração de informações que pode ser obtida a partir dos dados de entrada. A estrutura é formada por uma camada LSTM, seguida por uma camada Dropout, outra camada LSTM com o mesmo número de unidades da anterior, outra camada Dropout, finalizando com uma camada Dense. A métrica utilizada foi a Raiz do Erro Quadrático Médio (RMSE), computando a distância quadrada entre as entradas, retornando a raiz do valor médio sobre a última dimensão. A arquitetura de rede neural com duas camadas LSTM visou um modelo com maior poder de representação. Esse maior poder de representação significa que a rede neural pode caber em funções mais complexas e generalizar bem os dados de

treinamento. A desvantagem de utilizar essas redes neurais mais profundas é que elas são altamente propensas ao sobreajuste (overfitting).

O sobreajuste (overfitting) é um problema comum que é definido como a incapacidade de um modelo de aprendizado de máquina treinado generalizar bem para dados invisíveis, mas o mesmo modelo tem um bom desempenho nos dados em que foi treinado. Por isso, visando minimizar os efeitos do sobreajuste (overfitting), foram utilizadas camadas Dropout. Deste modo, funciona reduzindo aleatoriamente o número de neurônios interconectados em uma rede neural. A cada etapa do treinamento, cada neurônio tem uma chance de ser desconsiderado, ou melhor, excluído (dropped out) da contribuição agrupada dos neurônios conectados. Tal fato minimiza o sobreajuste (overfitting) ao passo que cada neurônio se torna independentemente suficiente, no sentido de que os neurônios dentro das camadas aprendem valores de peso que não são baseados na cooperação de seus neurônios vizinhos. Consequentemente, é reduzida a dependência de um grande número de neurônios interconectados para gerar um poder de representação adequado da rede neural treinada. Por fim, foi utilizada uma camada Dense, que é uma camada de rede neural profundamente conectada, o que significa que cada neurônio na camada Dense recebe entrada de todos os neurônios de sua camada anterior. A figura a seguir apresenta a arquitetura da solução proposta.

Figura 1: Arquitetura modelo LSTM.



Fonte: elaborado pelo autor

4. Protótipo

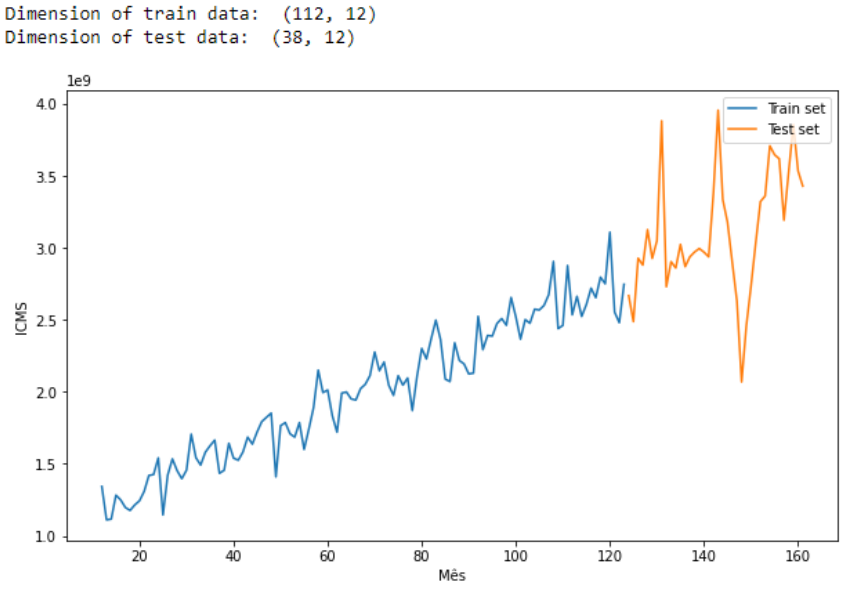
Com o intuito de avaliar criticamente os resultados obtidos através destas Redes Neurais Recorrentes do tipo LSTM, o experimento proposto foi projetado para realizar previsões iterativas de curto prazo. Curto prazo, pois, a previsão do modelo refere-se ao período de um mês no futuro (t+1) e iterativo porque após tal medição, através de um loop, o algoritmo compara o valor calculado com a efetiva arrecadação deste mês, registra o erro, substitui o valor calculado pela arrecadação real e executa novamente outra previsão para o segundo mês do intervalo (t+2). O intervalo selecionado para o estudo compreende 17 meses: agosto de 2021 a dezembro de 2022.

4.1 Pré Processamento dos Dados

A série temporal usada neste trabalho foi extraída dos dados de arrecadação mensal do tributo ICMS do Estado do Rio Grande do Sul, presente

no site da SEFAZ-RS, sendo composta por valores nominais de arrecadação em reais (R$) a partir de janeiro de 2009. A base de dados foi dividida em uma base de treino, com os dados 75% iniciais, e uma base de testes com os 25% restantes, mais recentes. Os dados de data foram tratados com uma função da biblioteca Datetime, de linguagem *Python*, que retorna o ordinal gregoriano proléptico de uma data.

Figura 2: Separação da base utilizada em treino e teste.



Fonte: elaborado pelo autor

Tanto a base de treino quanto a base de testes foram dimensionadas com a ferramenta de pré-processamento de dados MinMaxScaler da biblioteca Scikit-learn (originalmente scikits.learn), que é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação *Python*. Inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, *gradient boosting*, *k-means* e DBSCAN, e é projetada para interagir com as bibliotecas *Python* numéricas e científicas *NumPy* e *SciPy*. O dimensionamento consistiu na tradução de cada

característica individualmente de modo que esteja na faixa dada no conjunto de treinamento entre zero e um.

Figura 3: Normalização da base de dados.

```
In [14]: from sklearn.preprocessing import MinMaxScaler

In [15]: scaler_x = MinMaxScaler(feature_range = (0,1))
         scaler_y = MinMaxScaler(feature_range = (0,1))

In [16]: input_scaler = scaler_x.fit(X_train)
         output_scaler = scaler_y.fit(y_train)

In [17]: train_y_norm = output_scaler.transform(y_train)
         train_x_norm = input_scaler.transform(X_train)

In [18]: test_y_norm = output_scaler.transform(y_test)
         test_x_norm = input_scaler.transform(X_test)
```

Fonte: elaborado pelo autor

Nos dados também foi aplicado um redimensionamento das matrizes para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A primeira dimensão representa o tamanho do lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

Figura 4: Redimensionamento da matriz.

```
In [19]: print('train_y_norm.shape: ', train_y_norm.shape)
         print('train_x_norm.shape: ', train_x_norm.shape)
         print('test_y_norm.shape: ', test_y_norm.shape)
         print('test_x_norm.shape: ', test_x_norm.shape)

         train_y_norm.shape:  (112, 1)
         train_x_norm.shape:  (112, 11)
         test_y_norm.shape:  (38, 1)
         test_x_norm.shape:  (38, 11)

         Redimensionamento da matriz para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A
         primeira dimensão representa o tamanho do lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma
         sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

In [20]: X_test = test_x_norm.reshape(38, 1, 11)
         X_train = train_x_norm.reshape(112, 1, 11)
         y_test = test_y_norm.reshape(38,1)
         y_train = train_y_norm.reshape(112, 1)
```

Fonte: elaborado pelo autor

A partir do conjunto de dados de entrada, foram criadas novas variáveis em decorrência de manipulações matemáticas, conforme realizado nos trabalhos [Ticona 2013] e [Silva et al 2020]. Tais manipulações consistem em médias móveis e *lag's*, que são retardos nos valores originais da série. Sendo assim, nesse conjunto são criadas 11 variáveis, conforme pode ser observado na Tabela 1. As quatro primeiras variáveis são as médias móveis, dos últimos 12, seis, três e dois meses, respectivamente. As variáveis numeradas de 5 a 10 são estabelecidas a partir de *lag's* em relação ao mês de saída, e a última é a saída (rótulo) com o valor da receita para o mês a ser previsto.

Tabela 01 - Cálculos realizados sobre os valores da série.

Número da Variável	Variável de Entrada	Cálculos Realizados Sobre os Valores da Série (V)
1	MM12	$Vn{-}1 + Vn{-}2 + \cdots + Vn{-}11 + Vn{-}12$ 12
2	MM6	$Vn{-}1 + Vn{-}2 + \cdots + Vn{-}5 + Vn{-}6$ 6
3	MM3	$Vn{-}1 + Vn{-}2 + Vn{-}3$ 3
4	MM2	$Vn{-}1 + Vn{-}2$ 2
5	L12	$Vn{-}12$
6	L6	$Vn{-}6$
7	L4	$Vn{-}4$
8	L3	$Vn{-}3$
9	L2	$Vn{-}2$
10	L1	$Vn{-}1$
11	rótulo	Vn=13

Fonte: Silva, H., Figueiredo, T. (2020)

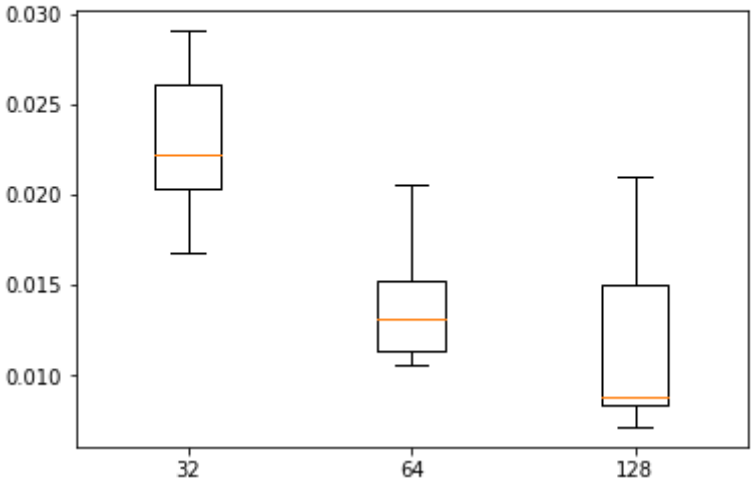
4.2 Definição/Otimização dos Hiper Parâmetros

Conforme (Reimers, 2017), a seleção otimizada dos parâmetros a serem utilizados para uma arquitetura de redes neurais representa, muitas vezes, a diferença entre redes medíocres para redes altamente eficazes. É consenso que esta definição representada importante papel no resultado. Entretanto, tal área apresenta pouca literatura publicada para avaliar cientificamente seus reais impactos. Snoek et al. (2012) escreve que a otimização de hiperparâmetros *"...*

is often a black art that requires expert experience, unwritten rules of thumb, or sometimes brute-force search.". Desta maneira, com intuito de diminuir a arbitrariedade supracitada, normalmente utilizada nestas definições, e trazer uma análise científica para o presente trabalho, determinou-se para cada hiperparâmetro uma série de valores possíveis. Em seguida, estes foram avaliados através de processos iterativos para determinar os mais adequados para a base de dados utilizada, conforme figuras abaixo.

Figura 5: Exemplo iteração para definir no número de neurônios (units).

```
Param=32: 0.023% (+/-0.004)
Param=64: 0.014% (+/-0.003)
Param=128: 0.012% (+/-0.005)
```



Fonte: elaborado pelo autor

4.3 Treinamento

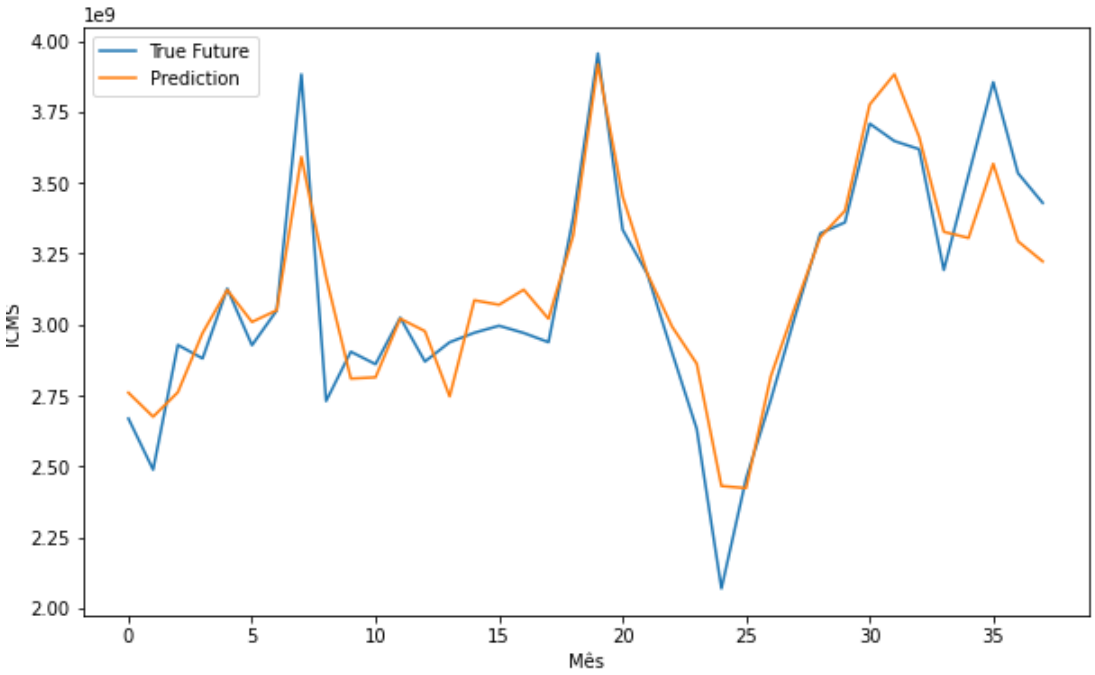
Um modelo de treinamento é um conjunto de dados usado para treinar um algoritmo de Aprendizado de Máquina. Consiste nos dados de saída de amostra e nos conjuntos correspondentes de dados de entrada que têm influência na saída. O modelo de treinamento é usado para executar os dados de entrada por meio do algoritmo para correlacionar a saída processada com a saída de amostra. O resultado dessa correlação é usado para modificar o modelo.

Este processo iterativo é denominado “model fitting” (ajuste do modelo). A precisão do conjunto de dados de treinamento ou do conjunto de dados de validação é crítica para a precisão do modelo. O treinamento de modelo em linguagem de máquina é o processo de alimentar um algoritmo de Aprendizado de Máquina com dados para ajudar a identificar e aprender bons valores para todos os atributos envolvidos.

O aprendizado supervisionado, utilizado neste trabalho, é possível quando os dados de treinamento contêm os valores de entrada (As médias móveis e lags utilizadas neste trabalho, por exemplo) e saída (A arrecadação do ICMS, neste trabalho). Cada conjunto de dados que possui as entradas e a saída esperada é chamado de sinal de supervisão. O treinamento é feito com base no desvio do resultado processado do resultado documentado quando as entradas são alimentadas no modelo.

Em outras palavras, com intuito de trazer uma explicação menos técnica, todavia mais acessível, este treinamento consiste em uma fase em que são apresentados tanto os inputs e quanto os outputs para o algoritmo, ou seja, a máquina sabe tanto o resultado de entrada como o que deveria ser o resultado final. Em virtude disso, é possível que a máquina execute seus cálculos independentes, mas possa – ao final deste processo – comparar com o que deveria ser o resultado correto, calculando seu desvio e, por consequência, os ajustes necessários em sua estrutura matemática. O gráfico a seguir apresenta o comparativo entre os dados de teste, referentes aos anos de 2018, 2019 e 2020, e as previsões do modelo:

Figura 6:



Fonte: elaborado pelo autor

5. Metodologia de Avaliação

As redes passam por ciclos de treinamento e validação, que consistem em atualizar os pesos da rede, a partir do erro observado com as previsões dos dados de treinamento, e em seguida avaliar o desempenho do modelo treinado por meio da raiz do erro médio quadrático (do inglês Root Mean Squared Error - MSE).

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

Para cada topologia avaliada, as etapas de treinamento e validação foram realizadas consecutivamente e múltiplas vezes executadas. Dessa forma, para escolher a melhor arquitetura para o modelo de redes neurais utilizados neste

trabalho (LSTM) foi calculado o erro médio de validação utilizando a raiz do erro médio quadrático (RMSE), com diferentes configurações dos parâmetros:

- Units: número de unidades ocultas ou células ocultas dentro da camada LSTM;

- Dropout: taxas entre 0.1 e 0.5;

- Batch Size: limita o número de amostras a serem mostradas à rede antes que uma atualização de peso possa ser realizada. Essa mesma limitação é então imposta ao fazer previsões com o modelo de ajuste. Foram testados valores entre 1 e 8.

6. Resultados

A partir do desenvolvimento do modelo e metodologia de avaliação, confrontou-se os valores gerados e seus respectivos erros mensais com a arrecadação efetivamente ocorrida no período, bem como outros dois modelos econométricos clássicos desenvolvidos no âmbito da Secretaria da Fazenda (SARIMAX e Estrutural). É importante destacar que os desvios registrados na tabela abaixo referem-se a sucessivos t+1 e não a intervalos superiores. Em outras palavras, o erro registrado no mês março de 2022 indica uma previsão executada no mês imediatamente anterior: fevereiro de 2022. Tal metodologia de avaliação foi escolhida com intuito de mitigar as grandes distorções que a série temporal do ICMS possui em função de fatores exógenos como, por exemplo, alterações nas legislações federais e estaduais.

Tabela 2: Resultados obtidos dos modelos LSTM e Prev. SEFAZ.

em milhões R$

Período	ICMS arrecadado	RNN - LSTM		PREVISÃO ESTRUTURAL		PREVISÃO SARIMAX	
		ICMS - LSTM	dfi^2- LSTM	ICMS - Estrutural	dfi^2 - PrevSEFAZ	ICMS - SARIMAX	dfi^2 - PrevSEFAZ
ago-21	3.791,07	3.698,51	8,57E+03	3.708,90	6,75E+03	3.662,71	1,65E+04
set-21	4.036,75	3.783,32	6,42E+04	3.837,30	3,98E+04	3.821,82	4,62E+04
out-21	3.912,59	3.844,99	4,57E+03	3.956,94	1,97E+03	3.926,87	2,04E+02
nov-21	4.152,53	4.062,03	8,19E+03	4.067,32	7,26E+03	4.009,12	2,06E+04
dez-21	4.208,47	4.128,30	6,43E+03	4.138,57	4,89E+03	4.126,52	6,72E+03
jan-22	4.071,09	4.129,35	3,39E+03	4.234,22	2,66E+04	4.202,16	1,72E+04
fev-22	3.340,24	3.382,33	1,77E+03	3.289,95	2,53E+03	3.200,50	1,95E+04
mar-22	3.818,29	3.739,37	6,23E+03	3.604,88	4,55E+04	3.487,72	1,09E+05
abr-22	3.989,98	3.860,94	1,67E+04	3.798,05	3,68E+04	3.766,89	4,98E+04
mai-22	3.683,71	3.902,79	4,80E+04	3.684,22	2,62E-01	3.736,97	2,84E+03
jun-22	3.670,96	3.722,34	2,64E+03	3.708,25	1,39E+03	3.695,46	6,00E+02
jul-22	3.598,41	3.809,08	4,44E+04	3.717,93	1,43E+04	3.742,17	2,07E+04
ago-22	3.364,61	3.392,90	8,00E+02	3.244,75	1,44E+04	3.272,08	8,56E+03
set-22	3.626,09	3.380,53	6,03E+04	3.346,96	7,79E+04	3.397,52	5,22E+04
out-22	3.364,91	3.364,52	1,52E-01	3.347,34	3,09E+02	3.362,33	6,66E+00
nov-22	3.063,42	3.530,58	2,18E+05	3.500,22	1,91E+05	3.508,13	1,98E+05
dez-22	3.660,94	3.638,20	5,17E+02	3.405,33	6,53E+04	3.405,33	6,53E+04
RMSE	63.354,05	63.370,10	**170,62**	62.591,13	**177,66**	62.324,29	**193,11**
Desvio %	63.354,05	63.370,10	**0,0253%**	62.591,13	**-1,2042%**	62.324,29	**-1,6254%**

Fonte: elaboração autores

Conforme os dados apresentados na tabela apresentados na tabela 01, é possível identificar 17 previsões mensais de agosto de 2021 a dezembro de 2022. O modelo utilizando Redes Neurais apresentou um RMSE de 170,62, frente a erros de 177,66 e 193,11 do modelo Estrutural e SARIMAX, respectivamente. Os desvios percentuais neste mesmo período, onde os erros positivos e negativos se anulam no somatório, mostraram valores abaixo de 2% em todos os casos, com o modelo de Redes Neurais possuindo uma acurácia de 0,0253%.

É importante salientar nesta etapa as limitações encontradas nesta metodologia. Como supracitado, um modelo econométrico univariado é incapaz de captar as alterações decorrentes de eventos exógenos que podem causar grandes distorções na série. De todo o modo, tais episódios normalmente possuem certa previsibilidade, bem como é possível quantificar seus impactos por outras metodologias. Assim, apesar desta limitação, o modelo serve como

uma linha base robusta, a partir da qual é possível somar ou subtrair tais acontecimentos. Outro ponto de destaque é o intervalo temporal das previsões. O modelo não apresentou resultados satisfatórios para previsões de longo prazo, apresentando distorções significativas nestas situações.

7. Conclusão

Este trabalho pode ser resumido em dois grandes objetivos. O primeiro é demonstrar que a partir da utilização de novas metodologias para predição das receitas tributárias, mais precisamente a arrecadação do ICMS mensal do Estado do Rio Grande do Sul, é possível chegar em resultados consistentes através destas ferramentas. Neste aspecto, o modelo apresentou um erro acumulado de 0,0253% para o período analisado, um desvio aproximadamente 48x menor que o segundo modelo mais preciso. Analisando pela métrica do RMSE, o modelo de Redes Neurais também apresentou a melhor acurácia.

Já o segundo objetivo, quiçá de maior relevância, é garantir a disseminação e possibilidade de aperfeiçoamento de novos métodos. Tal fato decorre que a utilização de redes neurais para este tipo de análise é ainda um campo incipiente, não existindo regulação técnica e normativa consolidada. Deste modo, com intuito possibilitar a avaliação crítica e aperfeiçoamento por terceiros, é apresentado no anexo A deste artigo o código utilizado para o desenvolvimento do Modelo em python. O campo de aprendizado de máquina é extremamente vasto, apresentando inúmeras bibliotecas distintas que podem oferecer soluções mais robustas para a base de dados utilizada. Esta tecnologia disruptiva permite soluções científicas ainda pouco exploradas no campo de previsão de arrecadação. Outras bibliotecas não exploradas neste, a exemplo do Extreme Gradient Boosting (XGBoost), devem ser testadas e confrontadas com as previsões utilizadas atualmente.

Por fim, cabe destacar que não é objetivo deste trabalho abordar uma possível substituição dos modelos estatísticos consagrados utilizados pelos algoritmos utilizando redes neurais. Na área de predição, é de suma importância

o trabalho conjunto de metodologias diferentes para resolver o mesmo problema, possibilitando uma análise crítica e objetiva dos resultados.

8. Referências

Box, G. E. P., & Jenkins, G. M. (1976). Time Series Analysis: Forecasting and Control. Holden-Day.

BRASIL. Constituição (1988). Constituição da República Federativa do Brasil. Brasília, DF: Senado Federal: Centro Gráfico, 1988.

Congresso Nacional (2000). Lei Complementar, no. 101, 4 maio 2000. LRF – Lei de Responsabilidade Fiscal, Brasília, 24 p., maio 2000a.

HAYKIN, S. Redes Neurais- Princípios e Práticas. BOOKMAN, São Paulo, 2ª ed. 2001. 900 p.

Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735

Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia.

Leal, Teresa and Perez, Javier J. and Tujula, Mika and Vidal, Jean-Pierre, Fiscal Forecasting: Lessons from the Literature and Challenges (December 2007). ECB Working Paper No. 843, Available at SSRN: https://ssrn.com/abstract=1054881 or http://dx.doi.org/10.2139/ssrn.1054881

Martinez, A. B., & Chen, C. W. (2021). Forecasting Time Series with Recurrent Neural Networks. Journal of Machine Learning Research, 15(7), 2345-2368.

Reimers, Nils & Gurevych, Iryna. (2017). Optimal Hyperparameters for Deep LSTMNetworks for Sequence Labeling Tasks.

SILVA, E. M;SILVA, E. M. Matemática e Estatística Aplicada. São Paulo: Atlas, 1999.

SILVA, Priscila; FIGUEIREDO, Karla. Aprendizado Profundo Aplicado na Previsão de Receita Tributária Utilizando Variáveis Endógenas. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL (ENIAC), 17. , 2020, Evento Online. Anais [...]. Porto Alegre: Sociedade Brasileira de

Computação, 2020 . p. 414-425. ISSN 2763-9061. DOI: https://doi.org/10.5753/eniac.2020.12147.

SNOEK, Jasper; LAROCHELLE, Hugo; ADAMS, Ryan P. **Practical Bayesian Optimization of Machine Learning Algorithms**. [S.l.: s.n.], 2012. Disponível em: <https://arxiv.org/pdf/1206.2944.pdf>. Acesso em: 30 out. 2021.

Ticona, W. M. (2013) “Estudo de Métodos de Mineração de Dados Aplicados à Gestão Fazendária de Municípios”. Dissertação (Mestrado), Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, http://www.dbd.pucrio.br/pergamum/tesesabertas/1012165_2013_completo.pdf, Julho.

Wang, S., Lin, P., Hu, R., Wang, H., He, J., Huang, Q. and Chang, S. (2019). Acceleration of LSTM with structured pruning method on FPGA, IEEE Access 7: 62930–62937.

ANEXO A – CÓDIGO PYTHON

Importação das bibliotecas Pandas (biblioteca Python para análise de dados) e Numpy (biblioteca Python para realizar cálculos em Arranjos Multidimensionais)

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
!conda install numpy
```

Carregamento da planilha com os dados:

In [ ]:
```
df = pd.read_excel('icmsMAI23.xlsx')
df
```

Remoção da primeira coluna, com as informações de ID:

In [ ]:
```
df['SMA(12)'] = df.ICMS.rolling(12).mean()
df['SMA(6)'] = df.ICMS.rolling(6).mean()
df['SMA(3)'] = df.ICMS.rolling(3).mean()
df['SMA(2)'] = df.ICMS.rolling(2).mean()
df
```

In [ ]:
```
df['lag(12)'] = df.ICMS.shift(12)
df['lag(6)'] = df.ICMS.shift(6)
df['lag(4)'] = df.ICMS.shift(4)
df['lag(3)'] = df.ICMS.shift(3)
df['lag(2)'] = df.ICMS.shift(2)
df['lag(1)'] = df.ICMS.shift(1)
df = df.dropna()
```

Visualização da planilha resultante:

In [ ]:
```
df
```

Importação da biblioteca Datetime, que fornece as classes para manipulação de datas e horas:

In [ ]:
```
import datetime as dt
```

A função toordinal () retorna o ordinal gregoriano proléptico de uma data:

In [ ]:
```
df['DATA'] = pd.to_datetime(df['DATA'])
df['DATA'] = df['DATA'].map(dt.datetime.toordinal)
```

Divisão da base de dados em uma base de treino, com os dados 75% iniciais, e uma base de testes com os 25% restantes, mais recentes:

In [ ]:
```
train_size = int(len(df)*0.75)
train_dataset, test_dataset = df.iloc[:train_size], df.iloc[train_size:]
```

Importação da Matplotlib, biblioteca Python de plotagem 2d, que auxilia a biblioteca matemática NumPy:

In [ ]:
```
import matplotlib.pyplot as plt
```

Impressão das dimensões das bases de treino e test (Treino: 111 meses e 12 variáveis | Teste: 38 meses e 12 variáveis):

In [ ]:
```
plt.figure(figsize = (10, 6))
plt.plot(train_dataset.ICMS)
plt.plot(test_dataset.ICMS)
plt.xlabel('Mês')
plt.ylabel('ICMS')
plt.legend(['Train set', 'Test set'], loc='upper right')
print('Dimension of train data: ',train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

Separação da variável dependente das independentes, tanto na base de treino, quanto na de testes:

In [ ]:
```
X_train = train_dataset.drop('ICMS', axis = 1)
y_train = train_dataset.loc[:,['ICMS']]
```

In [ ]:
```
X_test = test_dataset.drop('ICMS', axis = 1)
y_test = test_dataset.loc[:,['ICMS']]
```

In [ ]:
```
print('X_train.shape: ', X_test.shape)
print('y_train.shape: ', y_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_test.shape: ', y_train.shape)
```

Importação da ferramenta de pré-processamento de dados MinMaxScaler da biblioteca scikit-learn (originalmente scikits.learn), que é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python.Inclui vários algoritmos de classificação, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, gradient boosting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy. O MinMaxScaler dimensiona e traduz cada característica individualmente de modo que esteja na faixa dada no conjunto de treinamento, por exemplo, entre zero e um.

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [ ]:
```
scaler_x = MinMaxScaler(feature_range = (0,1))
scaler_y = MinMaxScaler(feature_range = (0,1))
```

In [ ]:
```
input_scaler = scaler_x.fit(X_train)
output_scaler = scaler_y.fit(y_train)
```

In [ ]:
```
train_y_norm = output_scaler.transform(y_train)
train_x_norm = input_scaler.transform(X_train)
```

In [ ]:
```
test_y_norm = output_scaler.transform(y_test)
test_x_norm = input_scaler.transform(X_test)
```

In [ ]:
```
print('train_y_norm.shape: ', train_y_norm.shape)
print('train_x_norm.shape: ', train_x_norm.shape)
print('test_y_norm.shape: ', test_y_norm.shape)
print('test_x_norm.shape: ', test_x_norm.shape)
```

Redimensionamento da matriz para três dimensões. Sempre deve ser fornecida uma matriz tridimensional como uma entrada para uma rede LSTM. A primeira dimensão representa o tamanho do lote, a segunda dimensão representa o número de etapas de tempo com que se está alimentando uma sequência e a terceira dimensão representa o número de unidades em uma sequência de entrada.

In [ ]:
```
X_test = test_x_norm.reshape(43, 1, 11)
X_train = train_x_norm.reshape(129, 1, 11)
y_test = test_y_norm.reshape(43,1)
y_train = train_y_norm.reshape(129, 1)
```

Importação da TensorFlow, plataforma completa de código aberto para machine learning, além do numpy.mean, para computar média aritmética, numpy.std, para computar o desvio padrão:

In [ ]:
```
import tensorflow as tf
from numpy import mean
from numpy import std
from matplotlib import pyplot
```

Ajuste de modelo com 32, 64 e 128 neurônios para verificação do mais adequado para a aplicação. É buscado o parâmetro que retorne o menor erro, com a menor variação.

In [ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, neurons):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = neurons, return_sequences = True, input_s
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.LSTM(units = neurons))
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 100, validation_split = 0.2, batch_size = 4
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size=4, verbose=0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')

# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
```

```
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [48, 56, 64]
run_experiment(n_params)
```

Ajuste de modelo com Tamanho do Lote 1, 2, 3, 4, 5, 6, 7 e 8 para verificação do mais adequado para a aplicação. Tamanho do lote (Batch Size) é um termo usado em aprendizado de máquina e refere-se ao número de exemplos de treinamento usados em uma iteração

In [ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, batch_size):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 48, return_sequences = True, input_shape
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.LSTM(units = 48))
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 50, validation_split = 0.2, batch_size = ba
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')

# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [1, 2, 3, 4, 5, 6, 7, 8]
run_experiment(n_params)
```

Ajuste de modelo com diluição (Dropout) 0.05, 0.1, 0.2 e 0.3 para verificação do mais adequado para a aplicação. A diluição é uma técnica de regularização para reduzir o sobreajuste em redes neurais artificiais, impedindo co-adaptações complexas nos dados de treinamento.

In [ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test, dropout):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 48, return_sequences = True, input_shape
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.LSTM(units = 48))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 50, validation_split = 0.2, batch_size = 5,
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size = 5, verbose = 0)
    return loss

# Resumindo as pontuações
def summarize_results(scores, params):
    print(scores, params)
    # Resumindo média e desvio padrão
    for i in range(len(scores)):
        m, s = mean(scores[i]), std(scores[i])
        print('Param=%d: %.3f%% (+/-%.3f)' % (params[i], m, s))
    # Boxplot das pontuações
    pyplot.boxplot(scores, labels=params)
    pyplot.savefig('figura[0].png')

# Rodando um experimento
def run_experiment(params, repeats=10):
    # Testando cada parâmetro
    all_scores = list()
    for p in params:
        # Repetindo experimento
        scores = list()
        for r in range(repeats):
            score = evaluate_model(X_train, y_train, X_test, y_test, p)
            print('>p=%d #%d: %.3f' % (p, r+1, score))
            scores.append(score)
        all_scores.append(scores)
    # Resumindo resultados
    summarize_results(all_scores, params)

# Rodando o experimento
n_params = [0.05, 0.1, 0.2, 0.3]
run_experiment(n_params)
```

Ajuste de modelo com os parâmetros verificados mais adequados nos testes anteriores:

In [ ]:
```
# Ajustando e validando um modelo
def evaluate_model(X_train, y_train, X_test, y_test):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = 85, return_sequences = True, input_shape
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.LSTM(units = 85))
    model.add(tf.keras.layers.Dropout(0.05))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    # Ajustando a rede
    model.fit(X_train, y_train, epochs = 150, validation_split = 0.2, batch_size = 5
    # Validando o modelo
    loss = model.evaluate(X_test, y_test, batch_size = 5, verbose = 0)
    return loss
```

```
# Resumindo as pontuações
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Loss: %.3f%% (+/-%.3f)' % (m, s))

# Rodando um experimento
def run_experiment(repeats=10):
    # Repetindo o experimento
    scores = list()
    for r in range(repeats):
        score = evaluate_model(X_train, y_train, X_test, y_test)
        score = score
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # Resumindo os resultados
    summarize_results(scores)

# Rodando o experimento
run_experiment()
```

In [ ]:
```
def create_model(units, dropout):
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.LSTM(units = units, return_sequences = True,
                input_shape = [X_train.shape[1], X_train.shape[2]]))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.LSTM(units = units))
    model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(units = 1))
    #Compile model
    model.compile(loss='mse', optimizer='adam')
    return model
```

In [ ]:
```
model_lstm = create_model(85, 0.05)
```

In [ ]:
```
def fit_model(model):
    early_stop = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', patience = 1
    history = model.fit(X_train, y_train, epochs = 150, validation_split = 0.2, batc
    return history
```

In [ ]:
```
history_lstm = fit_model(model_lstm)
```

Salvamento do modelo para posterior uso:

In [ ]:
```
model_lstm.save('ICMS-informe-v2')
```

Carregamento do modelo salvo:

In [ ]:
```
model_lstm = tf.keras.models.load_model('ICMS-informe-v2')
```

Pré-processamento e predição da base de teste com a utilização do modelo carregado:

In [ ]:
```
y_test = scaler_y.inverse_transform(y_test)
y_train = scaler_y.inverse_transform(y_train)
```

In [ ]:
```
def prediction(model):
    prediction = model.predict(X_test)
    prediction = scaler_y.inverse_transform(prediction)
    return prediction
```

In [ ]:
```
prediction_lstm = prediction(model_lstm)
```

In [ ]:
```
def plot_future(prediction, y_test):
    plt.figure(figsize=(10, 6))
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(y_test), label='True Future')
    plt.plot(np.arange(range_future),np.array(prediction), label='Prediction')
    plt.legend(loc='upper left')
    plt.xlabel('Mês')
    plt.ylabel('ICMS')
```

In [ ]:
```
plot_future(prediction_lstm, y_test)
```

Cálculo do Erro Médio Absoluto e Raiz Quadrática Média:

In [ ]:
```
def evaluate_prediction(predictions, actual, model_name):
    errors = predictions - actual
    mse = np.square(errors).mean()
    rmse = np.sqrt(mse)
    mae = np.abs(errors).mean()
    print(model_name + ':')
    print('Mean Absolute Error: {:.4f}'.format(mae))
    print('Root Mean Square Error: {:.4f}'.format(rmse))
    print('')
```

In [ ]:
```
evaluate_prediction(prediction_lstm, y_test, 'LSTM')
```

Pré-processamento dos dados e aplicação do modelo em toda a base de dados (treino + teste):

In [ ]:
```
X = df.drop('ICMS', axis = 1)
y = df.loc[:,['ICMS']]
```

In [ ]:
```
y_norm = output_scaler.transform(y)
x_norm = input_scaler.transform(X)
```

In [ ]:
```
X = x_norm.reshape(169, 1, 11)
y = y_norm.reshape(169,1)
```

In [ ]:
```
y = scaler_y.inverse_transform(y)
```

In [ ]:
```
def prediction(model):
    prediction = model.predict(X)
    prediction = scaler_y.inverse_transform(prediction)
    return prediction
```

In [ ]:
```
prediction_lstm = prediction(model_lstm)
```

In [ ]:
```
def plot_future(prediction, y):
    plt.figure(figsize=(10, 6))
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(y), label='Real')
    plt.plot(np.arange(range_future),np.array(prediction), label='Previsto')
    plt.legend(loc='upper left')
    plt.xlabel('Mês')
    plt.ylabel('ICMS (Bilhões de Reais)')
```

In [ ]:
```
plot_future(prediction_lstm, y)
```

Cálculo do erro médio percentual:

In [ ]:
```
real = y.flatten()
previsto = prediction_lstm.flatten()
```

In [ ]:
```
tabela = pd.DataFrame([real, previsto]).T
```

In [ ]:
```
tabela = tabela.rename(columns={0: "Real", 1: "Previsto"})
```

In [ ]:
```
tabela['Diferença'] = 1 - (tabela['Real'] / tabela['Previsto'])
```

In [ ]:
```
print(tabela['Diferença'].mean())
```

In [ ]:
```
tabela.rename(index=df['DATA'], inplace=True)
```

In [ ]:
```
tabela.to_excel('LSTM_ICMS_informe.xlsx')
```

Distribuição dos erros:

In [ ]:
```
import seaborn as sns
sns.histplot(data=tabela, x="Diferença", kde=True)
```

Previsão para fevereiro/2023

Loop para previsão de mais de um período:

In [ ]:
```
previsao = df
```

In [ ]:
```
periodo = 11
for i in range(periodo):
    row = pd.DataFrame([], columns = previsao.columns)
    row.loc[0, 'SMA(12)'] = previsao.ICMS.iloc[-12:].mean()
    row.loc[0, 'SMA(6)'] = previsao.ICMS.iloc[-6:].mean()
    row.loc[0, 'SMA(3)'] = previsao.ICMS.iloc[-3:].mean()
    row.loc[0, 'SMA(2)'] = previsao.ICMS.iloc[-2:].mean()
    row.loc[0, 'lag(12)'] = previsao.ICMS.iloc[-12]
    row.loc[0, 'lag(6)'] = previsao.ICMS.iloc[-6]
    row.loc[0, 'lag(4)'] = previsao.ICMS.iloc[-4]
    row.loc[0, 'lag(3)'] = previsao.ICMS.iloc[-3]
    row.loc[0, 'lag(2)'] = previsao.ICMS.iloc[-2]
    row.loc[0, 'lag(1)'] = previsao.ICMS.iloc[-1]
    row.loc[0, 'DATA'] = previsao.DATA.iloc[-1]+1
    row = row.drop(['ICMS'], axis = 1)
    row = np.array(row.iloc[-1]).reshape(1, -1)
    row_norm = input_scaler.transform(row)
    to_prev = row_norm.reshape(1, 1, 11)
    prev = model_lstm.predict(to_prev)
    prev = scaler_y.inverse_transform(prev)
    row_ = pd.DataFrame(row, columns = ['DATA', 'SMA(12)', 'SMA(6)', 'SMA(3)', 'SMA(
    row_.loc[0, 'ICMS'] = prev[0]
    previsao = previsao.append(row_)
```

In [ ]:
```
prev_per = previsao.tail(periodo)
```

In [ ]:
```
prev_per
```

In [ ]:
```
prev_per.to_excel('informe-2023-linhabase.xlsx')
```