UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Análise de Desempenho de Algoritmos de Configuração Automática de Parâmetros de LSTM Aplicadas a Séries Temporais

Trabalho de Conclusão de Curso

Jadson Carvalho Pereira

São Cristóvão – Sergipe

2023

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Jadson Carvalho Pereira

Análise de Desempenho de Algoritmos de Configuração Automática de Parâmetros de LSTM Aplicadas a Séries Temporais

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): André Britto de Carvalho
Coorientador(a): Yúri Faro Dantas de Sant'anna

São Cristóvão – Sergipe

2023

Agradecimentos

Venho aqui dedicar esse pequeno trecho para agradecer à todos que me acompanharam nessa jornada, me motivando a continuar por esse caminho, e oferecendo apoio quando necessário.

Agradeço também ao meu orientador, André Britto, cuja paciência, compreensão e influência me possibilitaram a realizar esse trabalho. À Yuri Faro, mentor na época em que passei da SEFAZ/SE e coorientador desse trabalho, por toda a ajuda para torná-lo possível. Meu agradecimento também vai ao professor Renê Pereira, que dedicou seu tempo para ler e fazer parte da banca que vai avaliar esse trabalho.

Meus agradecimentos especiais à minha mãe, Maria do Carmo, e ao meu pai, Joelson, cujos sacrifícios feitos para me apoiar nunca inteiramente terei a dimensão nem poderei retribuir. Ao meu irmão, Joebson, e minha irmã, Bianca, que desempenharam seus papéis de irmãos da melhor forma que poderia esperar.

Ao Eduardo, meu irmão de coração, que tive o prazer de partilhar moradia por todo seu tempo de graduação, e cujas visitas diárias sempre me confortavam. À Michelle, minha irmã de coração, com sua sabedoria contagiante e empatia que transparece, sempre me trazendo a calma quando necessitado.

E à minha querida companheira, Sophia Helena, que com seu enorme carisma e empatia está sempre ao meu lado. Me fornecendo luz em momentos que só há escuridão, e paz quando mais preciso. Para essas pessoas serei eternamente grato.

Resumo

A área de Aprendizagem de Máquina é um campo que visa o estudo de técnicas e algoritmos para resolver problemas complexos. Substituindo a necessidade de obter todo um conhecimento do problema e coletar dados de exemplos do problema para que o algoritmo se comporte adequadamente. Um dos recentes usos dessa área foi para solucionar o problema de previsão em Séries Temporais, das quais um valor da série é dependente dos valores anteriores. Devido a capacidade de armazenar informações, a Rede Neural Recorrente do tipo *Long-Short Term Memory*, ou LSTM, vem sendo utilizada para previsões nesse tipo de série. Entretanto, existe uma grande quantidade de parâmetros para configuração desse tipo de rede. Configurações essas que moldam a previsão final. Nesse trabalho é proposta uma análise dos métodos de configuração automática desses parâmetros na previsão da arrecadação tributária do estado de Sergipe, definida como uma série temporal. Serão utilizados os algoritmos *Hill Cimbing*, *Simulated Annealing*, *Genetic Algorithm* e *Social Network Optimization* para a otimização dos parâmetros. Os resultados das previsões da LSTM serão comparados com as previsões feitas pelo ARIMA e Holt-Winters.

Palavras-chave: Otimização. Hiper-Parâmetros. LSTM. Rede Neural Recorrente.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Sumário

1 Introdução

A Aprendizagem de máquina vem sendo cada vez mais reconhecida pelas diversas aplicações em todas as áreas do conhecimento, como por exemplo: filtros de spam em e-mails, reconhecimento facial, processamento de linguagem natural, recomendação de produtos para compras, e etc. Como definido por Rebala, Ravi e Churiwala (2019), aprendizagem de máquina é um campo de estudo de algoritmos e técnicas para a solução de problemas complexos demais para a programação convencional.

As soluções nessa área de estudo se resumem na capacidade de um computador entender um padrão em um conjunto de dados. Neste sentido, substitui-se a necessidade de obter um conhecimento primordial pela tarefa de coletar exemplos de dados, que demonstrem o comportamento do algoritmo desejado (SIMEONE, 2018). Esses exemplos de dados formam o conjunto de treinamento, que alimenta o modelo treinando-o para a tarefa desejada.

Como mostra Simeone (2018), existem três tipos de soluções de aprendizagem de máquina:

- Aprendizado supervisionado: é caracterizado pela presença de um conjunto de treinamento com dados de entrada e de saída esperada, com isso o modelo deve aprender esse padrão para outras entradas. As principais tarefas para esse tipo são: classificação, que prevê a classe de um dado de entrada; e regressão, que prevê um rótulo numérico para o dado de entrada.
- Aprendizado não-supervisionado: é caracterizado pela ausência de uma saída esperada, assim o conjunto de treinamento contém apenas dados de entrada. A principal tarefa para esse tipo é o agrupamento, que divide os dados de entrada em grupos diferentes de acordo com as características individuais desses dados.
- Aprendizagem por reforço: é o meio termo do aprendizado supervisionado e o não-supervisionado. Assim como o não-supervisionado, não existe uma saída esperada no

conjunto de treinamento, porém a supervisão acontece por meio de respostas recebidas do ambiente após selecionar uma saída dado um dado de entrada.

Séries temporais, como definido em Ratanamahatana e Keogh (2005), consistem em um conjunto de dados que contém uma sequência de eventos ordenados, com ou sem uma noção de tempo clara. Essa coleção de observações ocorre em uma grande variedade de campos, como por exemplo: economia, meteorologia, geofísica, marketing, geografia, e etc. Na área da economia, a previsão de séries temporais tem várias utilidades, como para a previsão da arrecadação de tributos ou na previsão de valores de ações no mercado financeiro. A arrecadação de tributos estaduais é de responsabilidade de suas Secretarias Estaduais da Fazenda. Os valores arrecadados variam de acordo com diversos fatores de meses anteriores, como por exemplo, o valor do dólar, exportação, até mesmo o valor arrecadado no mês anterior. Logo, o histórico de arrecadações mensais se caracteriza como uma série temporal por causa dessa dependência recorrente.

A predição de algum evento está entre os principais usos de séries temporais. Em Liu et al. (2021) é possível observar duas abordagens para a previsão, a primeira é utilizando equações clássicas onde os principais modelos são os seguintes:

- ARMA: é uma abreviatura para *Autoregressive Moving Average*, que é um modelo do tipo linear autorregressivo de média móvel.
- ARIMA: é uma abreviatura para *Autoregressive Integrated Moving Average*, que é um modelo também do linear autorregressivo de média móvel integrada.
- TAR: é uma abreviatura para *Threshold Autoregressive*, esse já é um modelo não-linear autorregressivo com limite.
- CCC: abreviatura para *Constant Conditional Correlation*, é um modelo não-linear de correlação constante condicional.
- Holt-Winters: é um modelo de suavização exponencial.

Modelos de equações clássicas apresentam um bom desempenho em séries temporais simples, como observado no trabalho de Liu et al. (2021), exceto em séries denominadas mais complexas. Essas séries são caracterizadas pela quantidade de variáveis que influenciam o evento e/ou a quantidade de valores que serão previstos. Assim é mais comum encontrar predições feitas com modelos de aprendizagem de máquina, como por exemplo: Rede Neural Artificial (GURNEY, 2018), Árvore de Decisão (QUINLAN, 1987), *Support Vector Machine* (CORTES; VAPNIK, 1995), e LSTM (HOCHREITER; SCHMIDHUBER, 1997a).

Uma Rede Neural Artificial é formada de um conjunto de unidades e nós interconectados cuja funcionalidade é reproduzir o comportamento dos neurônios animais, através de cálculos matemáticos (GURNEY, 2018). O processamento da rede está nos pesos obtidos durante o

processo de adaptação a partir dos padrões presentes em um conjunto de treinamento, esse processo é chamado de aprendizado.

Dentre os tipos de redes neurais encontra-se a Rede Neural Recorrente, definido em Hochreiter e Schmidhuber (1997a), como um rede que utiliza suas conexões entre os nós para armazenar os resultados que foi calculado em cada nó, na forma de ativações. Resultados esses que podem ser utilizados para novos aprendizados. Ou seja, é um tipo de rede que aprende através do que já foi aprendido. Redes Neurais Recorrentes possuem dois tipos principais de ativações: a "Memória de Curto Prazo", que muda os pesos de forma lenta; e a "Memória de Longo Prazo"que faz o oposto. Hochreiter e Schmidhuber (1997a) propõe *"Long Short-Tem Memory"* (LSTM - "Memória de Longo e Curto Prazo", em português) que é uma arquitetura de Rede Neural Recorrente com um algoritmo de aprendizagem baseado em gradiente.

O LSTM vem apresentando bons resultados para problemas de previsão em séries temporais, como em Shelatkar et al. (2020a). Issso ocorre pela capacidade dessas redes de armazenar informações durante o processo de aprendizado e previsão, assim compreendendo a dependência dessas séries pelo seu histórico. Essas redes também possuem uma alta flexibilidade na configuração de parâmetros, que transformam o comportamento do modelo para a solução desejada. Dentre esses estão: número de neurônios, quantidade de épocas, quantidade de camadas, funções de ativação, entre outros. Além desses parâmetros mudarem o resultado final, também modificam o tempo de execução do treinamento. Assim se faz necessário um estudo sobre esses parâmetros para obter um melhor resultado dentre todas as opções.

O problema de otimização se resume em encontrar a melhor solução que maximiza ou minimiza alguma função, chamada de função objetivo. No caso da configuração de parâmetros os algoritmos otimizadores dessa área são usados como meta-modelos. Onde as variáveis de decisão são os parâmetros à serem otimizados, e a função objetivo vem da avaliação do comportamento do modelo com uma combinação de parâmetros provida do algoritmo. Também é possível utilizar um algoritmo exaustivo, ou seja, ele executa todas as combinações de parâmetros em busca da melhor. Entretanto a quantidade de parâmetros pode tornar essa alternativa inviável pelo tempo de execução.

1.1 Objetivos

Esse trabalho tem como objetivo o estudo de métodos de configuração automática de parâmetros de LSTM na previsão de séries temporais. Para isso serão aplicados métodos de otimização no conjunto de parâmetros de um modelo LSTM. Sendo o *Hill Climbing*, *Simulated Annealing*, *Social Network Optimization*, e *Genetic Algorithm* esse métodos. Os mesmos também serão aplicados no ARIMA e Holt-Winters, que são técnicas clássicas de séries temporais. Os resultados dessas otimizações serão comparados com a otimização dos hiper-parâmetros de uma LSTM.

1.2 Metodologia

Para a configuração automática de parâmetros são utilizadas duas técnicas iterativas e duas populacionais. Essas técnicas são descritas a seguir:

- *Hill Climbing* (JACOBSON; SULLIVAN; JOHNSON, 1998) é um algoritmo de otimização que otimiza escolhendo sempre a melhor solução comparando a solução atual e uma versão modificada da mesma.
- *Simulated Annealing* (KIRKPATRICK; JR; VECCHI, 1983) é um algoritmo de otimização que, diferente do *Hill Climbing*, otimiza fazendo escolhas com base em cálculos probabilísticos, onde a chance de escolher a melhor solução aumenta com a quantidade iterações.
- *Genetic Algorithm* (HOLLAND, 1992) é um algoritmo de otimização evolucionário que possui uma população de soluções. Ele otimiza de forma análoga à seleção natural, onde ocorre seleções, cruzamentos, e mutações que criam novas gerações. A melhor solução é escolhida entre todas as gerações.
- *Social Network Optimization* (SHERAFAT, 2017) é um algoritmo de otimização populacional que se baseia no comportamento de um indivíduo em uma rede social. Cada indivíduo (solução) aprende com outros contatos dessa rede, e esses contatos aprendem com outros. No fim a melhor solução é escolhida entre todos os indivíduos.

Para analisar o comportamento dos modelos ARIMA, Holt-Winters e LSTM, os experimentos serão executados em quatro cenários diferentes, relativo ao intervalo de tempo de previsão. Esses cenários são rodados a partir da base de dados que contém os valores de arrecadação tributária de Sergipe de 2005 à 2021. Os quatro algoritmos de otimização terão seu tempo de otimização medidos, esse tempo também inclui a avaliação da função objetivo. Por fim, será feita uma previsão, em cada execução, dessa arrecadação onde será calculado a Média do Erro Absoluto, que indicará o quão divergente está a previsão do valor arrecadado.

1.3 Organização do trabalho

O trabalho está organizado da seguinte forma: no Capítulo 2 são descritos os métodos utilizados para previsão em séries temporais e otimização dos parâmetros, no Capítulo 3 é feita uma discussão do que foi implementado no projeto, no Capítulo 4 é mostrado como foi feito os experimentos e são apresentados os resultados, e no Capítulo 5 é concluído o trabalho.

2

Fundamentação Teórica

2.1 Aprendizagem de Máquina

De modo geral, a Aprendizagem de Máquina depende do aprendizado de um modelo que retorna a saída correta dado uma certa entrada. As entradas, também chamadas de atributos, são valores que representam os parâmetros que definem o problema, enquanto a saída é um valor que representa a solução (GAMBELLA; GHADDAR; NAOUM-SAWAYA, 2021). A nomenclatura das saídas varia de acordo com o problema. Problemas de classificação tem por saída classes, problemas de regressão são rótulos, e agrupamento são grupos. As representações dessas saídas são numéricas.

Para o modelo aprender é necessário uma etapa de treinamento, essa etapa se dá pela identificação de um padrão presente na coleção de exemplos, que contém valores de entrada e valores de saída correspondentes, essa coleção é chamada de conjunto de treinamento. Dependendo do problema, será necessário a utilização de outra coleção de exemplos, chamada de conjunto de testes. Esse conjunto serve para, após o treinamento, avaliar o aprendizado obtido pelo modelo.

Aprendizagem de máquina possui duas categorias principais: o aprendizado supervisionado e o não-supervisionado. O aprendizado supervisionado se baseia em aprender o padrão de associação dos valores de entradas com os valores de saída desejados, dados esses presentes no conjunto de treinamento. Uma função f representa o padrão de associação aprendido no treinamento com uma margem de erro razoável. A acurácia da previsão é avaliada usando uma função que calcula a distância medida entre o valor previsto e o valor real (GAMBELLA; GHADDAR; NAOUM-SAWAYA, 2021). De forma geral a melhor configuração é aquela que minimiza essa função.

O aprendizado não-supervisionado não possui a saída esperado no conjunto de treinamento, o objetivo é aprender as características distintivas presentes nas observações. Assim esse tipo de

aprendizado tenta compreender a distribuição, os atributos distintos e as associações dos dados (GAMBELLA; GHADDAR; NAOUM-SAWAYA, 2021). Um dos principais uso das técnicas dessa categoria é para análise exploratória de dados.

Regressão é um tipo de modelo supervisionado que, devido a sua simplicidade e eficiência, vem sendo bastante utilizado para predição de valores quantitativos. A relação entre as variáveis independentes (valores de entrada) e as variáveis dependentes (valores de saída) é representada por uma função de regressão que possui uma precisão razoável (GAMBELLA; GHADDAR; NAOUM-SAWAYA, 2021). Assim o objetivo principal desses modelos é encontrar a função de regressão que minimize a função de perda. A função de perda quantifica um tipo de taxa de erro para a função de regressão atual. A Figura 1 mostra um exemplo de regressão linear.

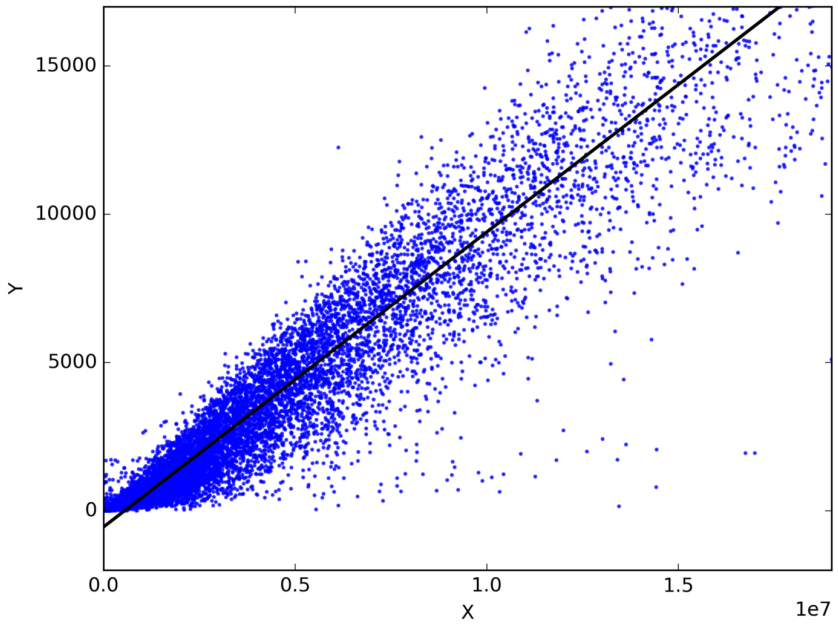

Figura 1 – Exemplo de Regressão Linear onde os pontos azuis representam os dados e a linha vermelha representa o comportamento da função de regressão.

Rede Neural Artificial é um modelo de aprendizagem computacional inspirada pela forma que os neurônios se comportam no cérebro humano com estímulos exteriores. Esses sistema possui uma camada de entrada que possui os dados de onde serão tiradas as observações, essa informação é passada para a camada oculta que possui um conjunto de neurônios responsáveis pelos cálculos nos dados de entrada. Por fim, o resultado de todos os cálculos é enviado para a camada de saída (GURNEY, 2018). A Figura 2 mostra um diagrama com as camadas citadas.

Rede Neural Recorrente (RNR) é um tipo especial de Rede Neural Artificial desenvolvida para trabalhar com dados que estão organizados de acordo com alguma sequência. Os neurônios da camada oculta da RNR possuem uma comunicação entre si. Esse modelo também possui uma espécie de memória para que algumas informações passadas possam ser utilizadas no aprendizado de novas características. A Figura 3 mostra um diagrama de uma RNR. *Long Short-Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997b) é uma das arquiteturas de RNR que obteve

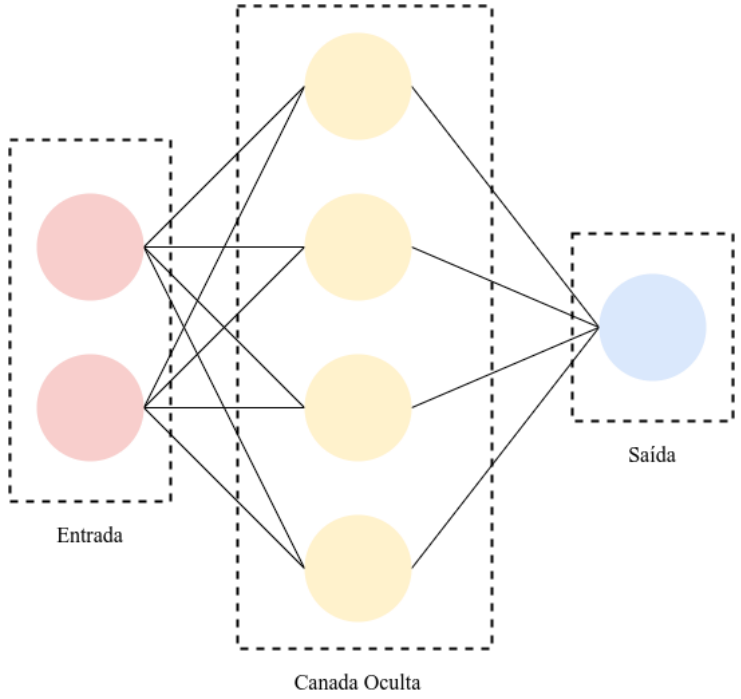

Figura 2 – Diagrama de uma Rede Neural Artificial.

um impacto notável em aplicações que envolvem transcrição, modelagem de linguagem, entre outras.

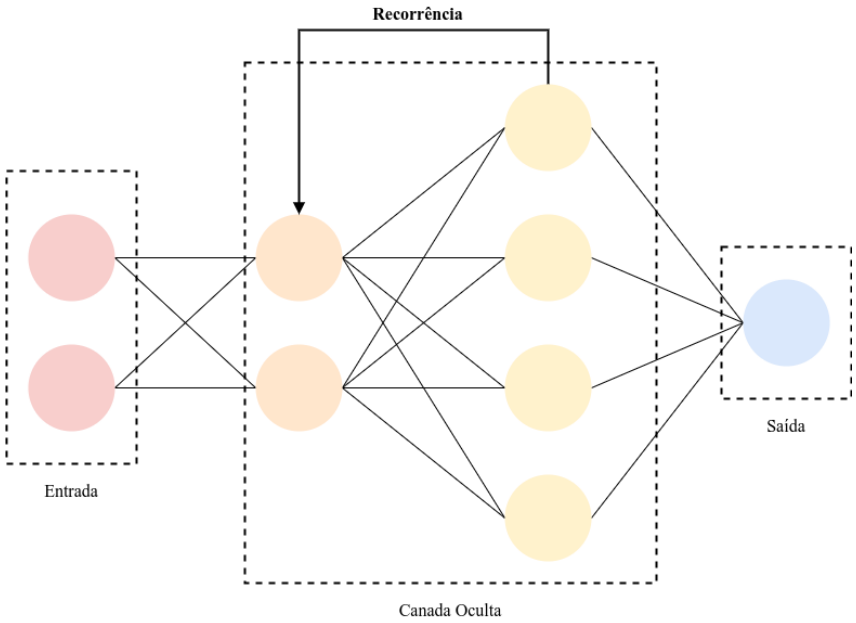

Figura 3 – Diagrama de uma Rede Neural Recorrente, com as camadas de entrada, oculta e de saída. A seta na parte superior representa o processo em que o que foi aprendido retorna como entrada para os neurônios da camada oculta.

2.2 Séries temporais

O principio da análise de séries temporais é encontrar padrões nos dados e prever valores futuros de acordo com uma ordem histórica de observações. As aplicações de previsões em séries temporais baseados em probabilidade e estatística vem rendendo grandes resultados nas áreas de meteorologia, industria, finanças,e etc (LIU et al., 2021).

O ARIMA(*p, d, q*) é um modelo auto-regressivo de média móvel integrada para séries temporais que captura apenas comportamentos lineares dos dados (LIU et al., 2021). Os parâmetros p e q representam as ordens do modelo, e d representa a ordem de diferenciação. Como apontado por Shelatkar et al. (2020b), a diferenciação feita no modelo ARIMA significa a formação de uma nova série temporal subtraindo uma observação anterior do tempo atual. Essa diferenciação elimina tendências, sazonalidades e variâncias inconsistentes dos dados da série temporal.

O Holt-Winters faz previsões baseadas em médias ponderadas de observações passadas e presentes, onde os pesos maiores estão focadas em dados mais recentes para dar menos significância a dados mais antigos (HEYDARI et al., 2019). Esse modelo pode ser desenvolvido para séries temporais com tendências e variações sazonais. O Holt-Winters possui a versão aditiva e a multiplicativa, sendo a segunda a versão padrão. Heydari et al. (2019) define a versão multiplicativa do Holt-Winters como:

$$\hat{y}_{n+1|n} = (m_n + lb_n)c_{n-s+1} \quad l = 1, 2, \ldots \tag{2.1}$$

Onde m_n é uma parte do nível, b_n é um componente da inclinação, e c_{n-s+1} é a parte relevante para a sazonalidade. Por exemplo, para previsões de séries temporais mensais, teríamos $s = 12$, logo a equação ficaria da seguinte forma:

$$\hat{y}_{n+1|n} = (m_n + b_n)c_{n-11} \tag{2.2}$$

2.3 Configuração automática de parâmetros de redes neurais

O desempenho de modelos complexos de aprendizagem de máquina, como o LSTM, varia de forma significativa de acordo os parâmetros escolhidos. Esses modelos podem contabilizar dezenas ou até centenas de parâmetros, tornando o processo de configuração de parâmetros essencial para obtenção de melhores resultados (CAI; LONG; SHAO, 2019).

Uma das formas para tornar esse processo automático é a modelagem desse problema como um problema de otimização, onde o objetivo será encontrar um conjunto de parâmetros (solução) que minimiza a taxa de erro, aqui será utilizada o Média do Erro Absoluto, ou MAE (*Mean Absolute Error*, em inglês) como taxa de erro. O modelo LSTM é representado pela função objetivo $g(\mathbf{x})$ onde $\mathbf{x}$ é o conjunto de parâmetros encontrado por meio do processo de otimização. As quatro técnicas de otimização utilizadas são descritas a seguir.

O *Hill Climbing* é uma técnica de otimização que compara a solução atual com outra próxima da atual e escolhe a melhor entre as duas. Novas soluções são encontradas através de processos de mutação, que são pequenas mudanças aleatórias na solução atual (LONES, 2011).

Simulated Annealing difere do *Hill Climbing* na escolha da próxima solução, pois não é certo que ele escolherá a melhor, isso só acontecerá de acordo com uma certa probabilidade. Essa probabilidade tem relação com a similaridade de ambas soluções e outro fator chamado de temperatura, esse fator tem uma relação proporcionalmente inversa à iteração do algoritmo, ou seja, quanto maior a iteração menor será a temperatura e maior será a probabilidade da melhor solução ser escolhida (LONES, 2011).

O Algoritmo Genético é um algoritmo evolucionário que se baseia na seleção natural. Esse algoritmo é dividido em processos chamados de Operadores Genéticos, que são: a seleção, o cruzamento e a mutação. Assim que uma população é criada ocorre a seleção alguns soluções onde pares são cruzadas. O cruzamento é o processo na qual partes de ambas as soluções dos pares são combinadas e uma nova solução nasce dessa combinação. Por fim essa solução sofrerá uma mutação. Esse processo ocorre até que uma nova população seja criada. O algoritmo finalizará quando a melhor solução for encontrada ou por outro critério programado (LONES, 2011).

Sherafat (2017) propôs um algoritmo de otimização chamado de *Social Network Optimization* (SNO), que é um algoritmo baseado em aprendizagem e interação de indivíduos em uma rede social, onde um membro tem contato com uma quantidade limitada de outros membros, ou amigos. Esses outros amigos mantém contato com outros membros, que podem ou não, serem amigos em comum do primeiro individuo. Cada individuo aprende algo do seu ciclo de amigos. No final, a melhor solução é escolhida dentre toda a população.

Nesse trabalho é feito uma otimização utilizando cada uma dessas técnicas, onde a solução com menor MAE é selecionada como a configuração para o modelo.

2.4 Trabalhos relacionados

O Liu et al. (2021) traz um estudo sobre vários métodos para previsão em séries temporais. Esse trabalho separa os métodos de previsão entre modelos clássicos lineares e não-lineares, abrangendo vários problemas que possuem séries temporais. Além disso ele introduz questões sobre o aspecto dos dados nesse tipo de série. Por fim, ele sumariza possíveis direções de pesquisa e problemas não resolvidos, como programação parelela e preprocessamento de dados em séries temporais.

Em Bakhashwain e Sagheer (2021) é proposto uma técnica de configuração automática de parâmetros de uma Deep LSTM (ou DLSTM) aplicado à séries temporais de forma dinâmica, que se adapta a qualquer a qualquer aplicação nesse tipo de série. A otimização dos hiper-parâmetros

é feita com um *Genetic Algorithm*, e o modelo sugerido se basea na ideia que o aumento de camadas escondidas é favorável ao desempenho geral. Os experimentos utilizam como métricos a Média do Erro Absoluto e a Raiz Quadrada da Média de Erro. Os resultados mostram que a configuração dinâmica dos hiper-parâmetros da DLSTM tem um melhor desempenho.

Shelatkar et al. (2020a) propõem uma previsão para tráfego de rede utilizando uma combinação de ARIMA e LSTM. Para contornar o problema do ruído, os dados são divididos em alta e baixa frequência. O ARIMA para prever as frequências altas e o LSTM para as frequências baixas. Isso permite que o ARIMA foque na previsão de componentes lineares e o LSTM de componentes não-lineares. Os resultados do trabalho mostram uma boa previsão da tendência pelo ARIMA, e uma previsão de picos pelo LSTM. Mostrando que essa combinação disponibiliza previsões com precisão.

Sherafat (2017) propõe um algoritmo de otimização que se baseia em uma rede social. Funcionando de forma análoga a como um indivíduo mantém contato e aprende com amigos em um ciclo social. Esse relacionamento do individuo com seus contatos é representado por um grafo onde as vértices representam os membros da rede e as arestas indica se existe um contato entre esses membros. O trabalho mostrou uma capacidade do SNO de evitar ótimos locais e achar melhores soluções.

3

Configuração de Hiper-Parâmetros em Previsores de Séries Temporais

Nesse capítulo serão discutidas como foram implementados as principais técnicas para previsão da série temporal, e os algoritmos para otimização dos hiper-parâmetros desses previsores. Também é mostrado a base de dados que originou na série temporal.

3.1 Previsão da arrecadação do estado de Sergipe

A Secretaria da Fazendo do Estado de Sergipe, ou SEFAZ/SE, é um órgão do poder executivo estadual responsável pelo controle financeiro, arrecadação tributária, e planejamento/execução do orçamento do estado. As maiores fontes tributárias estaduais vêm de impostos como IPTU, IPVA, e ICMS. Esse trabalho traz uma análise dos métodos para prever essa arrecadação. E também uma solução para configuração automática dos parâmetros desses previsores para diminuir a taxa de erro presente na previsão. A previsão da arrecadação visa facilitar esse planejamento orçamentário disponibilizando esboços do quanto será arrecadado em intervalos de um, três, seis e doze meses. A Figura 4 mostra a arrecadação de tributos do ano de 2019.

A previsão é feita através dos dados disponibilizados pela instituição, com valores de 2005 até abril de 2021. O desenvolvimento da previsão foi dividida em duas etapas principais: a previsão da série temporal utilizando a Rede Neural Recorrente, e métodos clássicos para comparação de resultado; e a otimização dos hiper-parâmetros para minimizar a porcentagem de erro da previsão. Ambas etapas são discutidas a seguir.

3.2 Visão Geral do Sistema

Os modelos construídos para previsão da série temporal seguem o padrão da classe abstrata `Model`, que possui os métodos abstratos `avaliate` e `predict`, e o método implementado `mae` que recebe a previsão e os valores reais e calcula a Média do Erro Absoluto, MAE (*Mean*

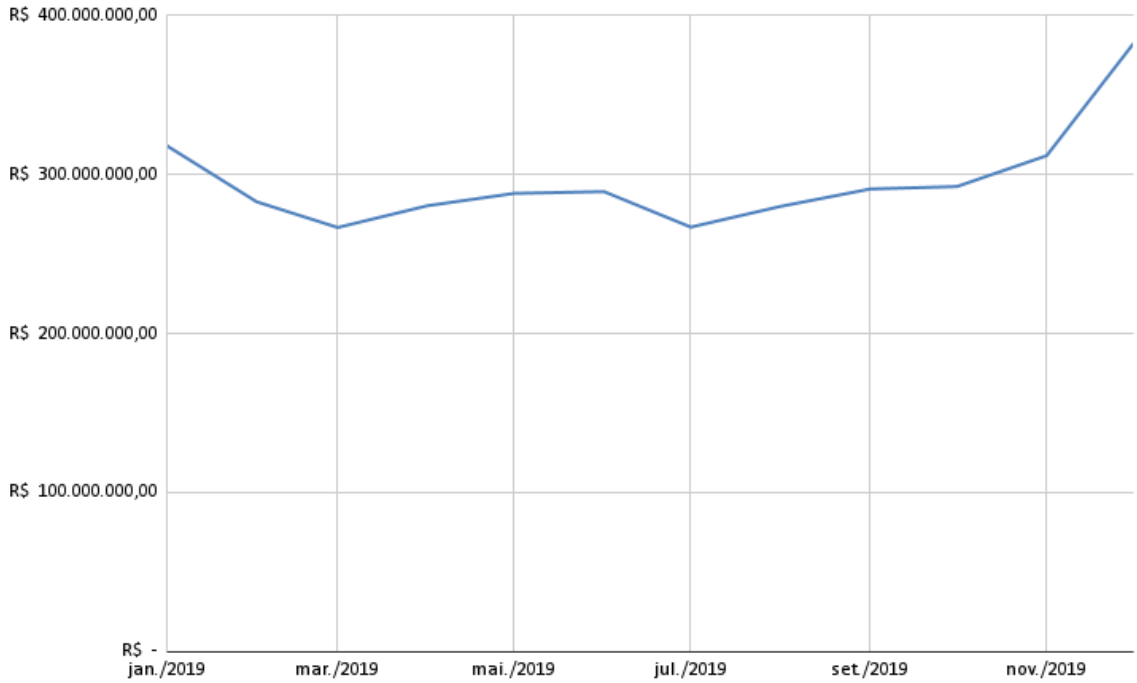

Figura 4 – Arrecadação de tributos para o ano de 2019.

Absolute Error, em inglês). O método `avaliate` retorna o MAE para a solução de entrada, já o `predict` retorna a previsão de acordo com os dados de entrada e o intervalo parametrizado na subclasse.

A Rede Neural Recorrente é construído a partir do modelo *Sequential* e as camadas *Dense*, LSTM, e *Dropout* da biblioteca *keras*, (CHOLLET et al., 2015). As camadas de recorrência são feitas com a LSTM, e a camada final é feita com a do tipo Dense. O *Dropout* serve para regularizar a rede sem modificar as funções de ativação.

Para os métodos clássicos de previsão foram modelados o ARIMA e o Holt-Winters. Ambas técnicas são implementadas a partir da biblioteca *statsmodels*, (SEABOLD; PERKTOLD, 2010).

3.3 Problema de otimização

O problema de otimização de hiper-parâmetros consiste na minimização da Média de Erro Absoluto a partir de uma combinação de parâmetros. Assim, as variáveis de decisão são representadas por um vetor de números reais, entre 0 e 1, que representam a combinação de parâmetros que configuram o modelo sendo otimizado. A função objetivo vem do cálculo da Média de Erro Absoluta, que é um cálculo feito entre o valor previsto da série temporal pelo modelo, com a combinação de parâmetros, e o valor real arrecadado.

3.4 Algoritmos de otimização

Os parâmetros que serão otimizados para configuração dos previsores, são separados em três tipos de dados: discreto, inteiro, e real. Foi construído um codificador para transformar esses parâmetros em números reais, com valor entre 0 e 1, que são as representações das variáveis de decisão. Esse codificador possui uma máscara para decodificação, que é feita na avaliação da função objetivo. É necessário configurar um arquivo *json* com os parâmetros que serão otimizados, e para cada parâmetro deve haver os campos *type*, para definir o tipo, e *discrete*, com o valor verdadeiro ou falso para definir se esse parâmetro é discreto.

Para os parâmetros discretos é necessário o campo *itens* com as opções para aquele parâmetro em formato de vetor. Para parâmetros inteiro e real deve haver um campo chamado *limits*, no formato de vetor, com os limites inferior e superior, representados por números reais entre 0 e 1. No caso do tipo inteiro esses limites devem ser divididos pela mascara.

Os algoritmos escolhidos para otimização dos hiper-parâmetros seguem o padrão da classe abstrata `algorithm` que possui os métodos abstratos: `initiate_solution` e `execute`. O método `initiate_solution` inicia o vetor solução com representações numéricas dos parâmetros. Esse valores são gerados aleatoriamente, respeitando os limites de cada parâmetro. Já o método `execute` possui o fluxo de execução do algoritmo de otimização. E um método implementado chamado de `avaliate` que calcula a Média do Erro Absoluto a partir dos valores previstos, e o valor real arrecadado.

O método `mutate`, Código 1, é utilizado em todas os quatro otimizadores. Ele é uma implementação do *Simple Random Mutation*, que é uma mutação simples realizadas em números reais. Ela funciona de acordo com uma probabilidade p que define as chances de um parâmetro no vetor solução sofrer alterações. Essa alteração é definida a partir da equação:

$$x_i = lower + ((upper - lower) * rand) \tag{3.1}$$

Onde x_i é o elemento que sofrerá a mutação, $lower$ e $upper$ são, respectivamente, os limites inferior e superior impostos para o parâmetro xi, e $rand$ é um número real com valor entre 0 e 1, gerado aleatoriamente.

Nas subseções a seguir é discutido como foram implementadas as subclasses: Hill Climbing, Simulated Annealing, Genetic Algorithm e Social Network Optimization.

3.4.1 Hill Climbing

A implementação do Hill Climbing, que foi feita baseada em (LUKE, 2013), possui os métodos: `initiate_solution`, `tweak` e `execute`. O fluxo principal do algoritmo se encontra no método `execute`, Código 2, ele começa iniciando um vetor de parâmetros com o método `initiate_solution`, e depois cria uma cópia desses parâmetros e, utilizando o método `tweak`, é feita uma mutação no vetor. Uma avaliação da função objetivo é feita para decidir qual vetor

Código 1 – Método de mutação baseada na *Simple Random Mutation*.

```
def mutate(self, solution):
    for i in range(self.solution_size):
        if random.rand() <= self.p:
            inf, sup = self.limits[i]
            value    = inf + ((sup - inf) * random.rand())

            solution.solution[i] = value

    return solution
```

possui os parâmetros com as melhores previsões. Esse processo é repetido até que seja atingido o número máximo de iterações.

Código 2 – Fluxo principal do Hill Climbing.

```
def execute(self, initial_solution=None):
    best = self.initiate_solution(initial_solution)

    best = self.avaliate(best)

    i = 0
    while i < self.max_iterations:
        solution = best.copy()
        solution = self.tweak(solution)
        solution = self.avaliate(solution)

        if best.avaliation > solution.avaliation:
            best = solution.clone()

        i += 1

    return best
```

3.4.2 Simulated Annealing

Essa implementação foi feita com base em (LUKE, 2013). O que difere esse algoritmo do Hill Climbing é a existência da chance de escolher um vetor onde os parâmetros escolhidos geraram uma previsão que o MAE foi maior que o do vetor atual. O fluxo principal é mostrado no Código 3. Assim como o Hill Climbing é iniciada um vetor de parâmetros com o `initiate_solution`, a cópia desse vetor passa pela processo de mutação através do método `tweak`, Código 1 (explicado na Seção 3.4). Após isso os parâmetros são comparados, e é nesse ponto que os algoritmos divergem. Se da mutação surgir um vetor com parâmetros piores, esse ainda pode ser escolhido

caso a seguinte seguinte condição seja satisfeita:

$$rand < \epsilon^{(Q_{best} - Q_{mutation})/t} \tag{3.2}$$

O valor $rand$, na equação, é um número real, entre 0 e 1, gerado aleatoriamente. Q_{best} e $Q_{mutation}$ são os valores do MAE para o vetor com a melhor combinação de parâmetros e o novo vetor que sofreu mutação, respectivamente. E por fim t é o valor que representa a temperatura, esse valor diminui a cada iteração através da Equação 3.3. O t também é utilizado como parada do algoritmo, isso ocorre quando o valor ficar menor ou igual a zero.

$$t = t - \log_{10}(i + 1) \tag{3.3}$$

Onde i representa a quantidade de iterações que já passaram.

Código 3 – Fluxo principal do Simulated Annealing.

```
1  def execute(self, initial_solution=None):
2      t = self.t
3
4      solution = self.initiate_solution(initial_solution)
5      solution = self.avaliate(solution)
6      best     = solution
7
8      i = 0
9      while i < self.max_iterations:
10         if t <= 0:
11             break
12
13         solution_tweak = self.tweak(best.copy())
14         solution_tweak = self.avaliate(solution_tweak)
15
16         exp = (best.avaliation - solution_tweak.avaliation)/t
17         if best.avaliation > solution_tweak.avaliation or
              random.rand() < np.power(np.e,exp):
18           solution = solution_tweak
19
20         t = self.decrease(t,i)
21
22         if best.avaliation > solution.avaliation:
23             best = solution.clone()
24             self.history.append(best.clone())
25
26         if solution.avaliation < np.Inf:
27             i += 1
28
29     return best
```

3.4.3 Genetic Algorithm

O Genetic Algorithm foi implementado com base em (LUKE, 2013), seu fluxo principal é mostrado no Código 4. O algoritmo inicia criando uma população de vetores com parâmetros cujos valores são aleatórios. O MAE de cada combinação de vetor é calculada. No próximo passo é iniciado a seleção de indivíduos, através do método `select`, que serão utilizados para recombinação, `crossover`. Cada descendente gerado sofre mutações utilizando o método `mutate`. Por fim ocorre, novamente, a avaliação das previsões é feita com cada vetor de parâmetros representados pelos descendentes. A população inicial é substituída por essa nova geração, esse processo ocorre até um certo número de gerações.

Esse algoritmo possui como componentes principais a seleção, a recombinação e a mutação, que são representados respectivamente pelos métodos: `select`, `crossover` e `mutate`. O método `select` utiliza a técnica de *Tournament Selection*, que, nesse caso, escolhe quatro indivíduos, combinações de parâmetros, aleatoriamente e os dois melhores, de acordo com a previsão feita a partir de cada combinação, são selecionados. Já o `crossover`, Código 5, utiliza a técnica *Whole Arithmetic Crossover*. A ocorrência da recombinação depende de um parâmetro de probabilidade p. Caso ocorra, os vetores serão recombinados aplicando as Equações 3.4 e 3.5, para cada parâmetro de cada de ambos vetores. Os elementos gerados das recombinações serão reparados para respeitarem os limites de cada parâmetro.

$$x = \alpha * V_i + (1 - \alpha) * W_i \tag{3.4}$$

$$y = \alpha * W_i + (1 - \alpha) * V_i \tag{3.5}$$

Onde α é um número entre 0 e 1 gerado aleatoriamente, V e W os vetores solução de entrada, e x e y os elementos gerados da recombinação.

3.4.4 Social Network Optimization

O *Social Network Optimization* (SNO), (SHERAFAT, 2017), é um algoritmo de otimização que se baseia no conceito de aprendizado através de uma rede de amigos. O fluxo principal da implementação é mostrado no Código 6. O algoritmo inicia uma população com o `initiate_population`, e então gera um grafo conectado, Código 7, com grau d a partir da população gerada. Para cada individuo da população é feita uma maximização do valor ϕ. O detalhe, gerado em `detail`, com o maior ϕ substitui o individuo da população. Ou seja, se a combinação de parâmetros desse detalhe, parâmetros aprendidos do amigo, gerar uma previsão com um MAE menor, ele substitui a combinação que foi gerada inicialmente.

O valor ϕ é calculado através da Equação 3.6. O método `detail` representa o detalhe, ou conjunto de parâmetros que um individuo aprende com o seu amigo da rede. Isso ocorre copiando os valores dos parâmetros do vetor do seu amigo nas posições de i a j. Essas posições

Código 4 – Fluxo principal do Genetic Algorithm.

```
def execute(self, initial_population=None):
  self.initiate_population(initial_population)
  best = None
  i    = 0

  while i < self.max_iterations:
    for individual in self.population:
      individual = self.avaliate(individual)

      if best is None or best.avaliation > individual.avaliation:
        best = individual

    offspring = set()

    while len(offspring) < int(self.pop_size/2):
      sol1 = self.select(self.population)
      sol2 = self.select(self.population)

      child1, child2 = self.crossover(sol1.copy(), sol2.copy())

      child1 = self.mutate(child1.copy())
      child2 = self.mutate(child2.copy())

      child1 = self.avaliate(child1)
      child2 = self.avaliate(child2)

      if child1.avaliation < np.Inf and child2.avaliation < np.Inf:
        offspring.add(child1)
        offspring.add(child2)

    self.population.clear()
    for c in offspring:
      self.population.append(c.copy())

    i += 1

  return best
```

Código 5 – Whole Arithmetic Crossover.

```
def crossover(self, v, w):
  offspring = (v.copy(), w.copy())

  if random.rand() <= self.p:
    alpha = random.rand()

    for i in range(len(sol1.solution)):
      lower, upper = self.limits[i]

      x = alpha*v.solution[i] + (1.0-alpha)*w.solution[i]
      y = alpha*w.solution[i] + (1.0-alpha)*v.solution[i]

      # solution repair
      x = max(lower, x)
      x = min(upper, x)

      y = max(lower, y)
      y = min(upper, y)

      offspring[0].solution[i] = x
      offspring[1].solution[i] = y

  return offspring
```

são definidas por números gerados aleatoriamente, respeitando os intervalos: $0 <= i < m$; e $i < j <= m$, onde m é o tamanho do vetor solução.

$$\phi = \alpha * (Q_x - Q_d) + \beta * (Q_x - Q_f) \tag{3.6}$$

Onde α e β são parâmetros da subclasse. Q_x, Q_d e Q_f, são, respectivamente, os valores do MAE calculados a partir dos vetores de parâmetros do indivíduo, do detalhe, e do amigo.

O método `generate_graph`, Código 7, gera um grafo conectado com n vértices e d arestas, sendo n o tamanho da população e d um parâmetro da subclasse. As vértices representam os indivíduos da população e as arestas representam a amizade entre esses indivíduos. Assim d será a quantidade de amigos que cada individuo terá. O grafo é criado a partir de uma busca em profundidade.

Código 6 – Fluxo principal do Social Network Optimization.

```
def execute(self, initial_pop=None):
    graph = [list() for _ in range(self.population_size)]

    self.initiate_pop(initial_pop)

    graph = self.generate_graph(self.population)
    k = 0
    while k < self.max_iterations:
        pop = list(self.population)

        for x in range(len(pop)):
            maxPhi = -np.infty
            maxdetail = None

            pop[x] = self.avaliate(pop[x])
            for friend in graph[x]:
                friend = min(len(pop)-1, friend)

                d = self.detail(pop[x].copy(), pop[friend])
                d = self.avaliate(d)

                pop[friend] = self.avaliate(pop[friend])
                phi = self.alpha*(pop[x].avaliation - d.avaliation)
                phi += self.beta*(pop[x].avaliation -
                   pop[friend].avaliation)

                if phi >= maxPhi:
                    maxPhi = phi
                    maxdetail = d

            if maxPhi > 0:
                pop[x] = maxdetail

            k += 1

        self.population.clear()
        for i in pop:
            self.population.add(i)

    best = pop[0]
    for x in self.population:
        if x.avaliation < best.avaliation:
            best = x

    return best
```

Código 7 – Gerador de grafos.

```
def generate_graph(self, population):
    graph = [list() for _ in range(self.pop_size)]
    visited = [0 for _ in range(self.pop_size)]
    connected = set()
    for i in range(self.pop_size):
        connected.add(i)

    def __dfs(v):
        visited[v] = -1
        connected.remove(v)

        vertices = set()
        if len(connected) > 0:
            graph[v].append(random.choice(list(connected)))
            vertices.add(graph[v][0])
        while len(vertices) < self.d[v]:
            w = random.randint(low=0,high=self.pop_size)
            if w != v:
                vertices.add(w)

        for w in vertices:
            graph[v].append(w)

        for w in graph[v]:
            if visited[w] == 0:
                __dfs(w)

    __dfs(0)

    return graph
```

4
Experimentos

Os experimentos aqui apresentados consistem no cálculo da Média do Erro Absoluto entre a previsão do modelo, com o conjunto de hiper-parâmetros otimizado, e o valor real da arrecadação. A duração da otimização é outra métrica utilizada para comparar os otimizadores e os modelos. Visando a manutenção da integridade dos resultados, os algoritmos de otimização tiveram seus parâmetros devidamente configurados para que o número de iterações sejam similares.

4.1 Base de dados

A base de dados utilizada consiste em um arquivo *Comma-Separated Values*, ou CSV, com três colunas: ano, mês, e valor arrecadado. Os valores são de janeiro de 2005 à abril de 2021, e estão ordenados pelo ano e pelo mês de forma crescente. Para avaliar o comportamento de cada modelo em cenários diferentes é feita a otimização e previsão para os intervalos de um, três, seis e doze meses da arrecadação. Assim, primeiro são rodados os experimentos para a previsão de um mês de arrecadação. Depois para três meses e assim sucessivamente.

O dados da arrecadação são divididos em duas partes: a base de otimização e a de previsão. A base de otimização consiste nos dados que serão utilizados no processo de otimização dos parâmetros, e possuem as linhas $[0, n - T_{intervalo})$, onde n é a quantidade total de linhas do arquivo e $T_{intervalo}$ é o intervalo de tempo. Já a base de previsão possuem as linhas $[n - T_{intervalo}, n)$, e é utilizada para fazer a previsão final já com os parâmetros otimizados, cujo valores são usados para avaliar o desempenho do otimizador e do modelo através da Média do Erro Absoluto.

4.2 Métricas

Ao final de cada otimização é feita uma previsão com os modelos otimizados, para o intervalo em questão, esses valores previstos são usados em conjunto com os valores arrecadados para calcular a Média do Erro Absoluto, ou MAE (*Mean Absolute Error*, em inglês). O MAE é calculado de acordo com a Equação 4.1. Com essa métrica é possível observar o quanto a previsão está divergindo do valor arrecadado.

$$MAE = \frac{1}{T}\sum_{i=1}^{T} |\frac{\hat{y}_i - y_i}{y_i}| \tag{4.1}$$

Onde T e o tamanho do intervalo previsto. $\hat{y}_i$ e y_i são, respectivamente, o valor previsto e o valor arrecadado relativo ao mês com índice i.

O tempo de execução, em segundos, é outra métrica utilizada para comparação dos resultados. O cronômetro implementado é ativado com o inicio da otimização de cada modelo. Ele contabiliza todas as operações que ocorrem durante esse processo. Como, por exemplo, as operações de geração de soluções aleatórias, treinamento de modelo, previsão, mutações, e etc. O cronômetro só para quando o número máximo de iterações for atingido. Essa métrica visa destacar os algoritmos que possuem o menor custo computacional.

4.3 Algoritmos e parâmetros

Os experimentos são executados para cada combinação de otimizador, modelo, e intervalo. Cada combinação é executada 10 vezes. Sendo ARIMA, *Holt-Winters*, e LSTM os modelos previsores. Os parâmetros dos algoritmos de otimização foram configurados de forma que a quantidade de iterações seja similar. Assim, a quantidade máxima de iterações do *Genetic Algorithm* e do SNO foram dividas pelo tamanho das populações. Todos os otimizadores possuem probabilidade $p = 0.2$. Os algoritmos e demais parâmetros são mostrados na Tabela 4.3.

Tabela 1 – Parâmetros dos Algoritmos de Otimização

Algoritmo	**Parâmetro**	**Valor**
Hill Climbing	max_iterations	200
Simulated Annealing	max_iterations	200
Simulated Annealing	t	2000
Genetic Algorithm	max_iterations	20
Genetic Algorithm	pop_size	10
Social Network Optimization	max_iterations	20
Social Network Optimization	pop_size	10
Social Network Optimization	d	[5,5,5,5,5,5,5,5,5,5]
Social Network Optimization	alpha	0.7
Social Network Optimization	beta	0.7

Para a otimização de hiper-parâmetros do ARIMA, foram utilizados oito parâmetros que variam entre inteiros, booleanos, e discretos do tipo *string*. Esses parâmetros são descritos a seguir:

- *enforce_stationarity*: é um parâmetro booleano que indica se é para reforçar a estacionariedade na parte de Auto-Regressão do modelo.
- *enforce_invertibility*: é um parâmetro booleano que indica se é para reforçar a invertibilidade na parte de Média Móvel do modelo.
- *concentrate_scale*: é um parâmetro booleano que indica se é para concentrar a escala reduzindo o número de parâmetros estimados pelo máximo de verossimilhança em um.
- *cov_type*: é um parâmetro do tipo *string* que define o método para calcular a matriz de covariância dos parâmetros estimados. Os métodos são:
  - "opg"para o produto externo do estimador do gradiente;
  - "oim"para o estimador da matriz de informações observadas, calculada utilizando o método de Havey;
  - "approx"para o estimador da matriz de informações observadas, utilizando uma aproximação numérica da matriz Hessiana;
  - "robust"para uma aproximação da matriz de covariância que pode ser válida mesmo com especificações incorretas. Cálculos intermediários utilizam o método "oim";
  - "robust_approx"é o mesmo que "robust"exceto que utiliza o método "approx"para cálculos intermediários; e
  - "none"para a ausência de cálculos da matriz de covariância.
- *method*: é um parâmetro do tipo *string* que determina qual a solução de *scipy.optimize* é utilizada. As opções são as seguintes:
  - "newton"para *Newton-Raphson*;
  - "nm"para *Nelder-Mead*;
  - "bfgs"para *Broyden-Fletcher-Goldfarb-Shanno* (BFGS);
  - "lbfgs"para BFGS com memória limitada;
  - "powell"para o método modificado de Powell;
  - "cg"para gradiente conjugado;
  - "ncg"para o gradiente conjugado de Newton;
  - "basinhopping"para a solução *basin-hopping* global.
- p: Número inteiro que indica a ordem de Auto-Regressão.
- d: Número inteiro que indica a ordem de integração do processo.
- q: Número inteiro que indica a ordem da Média Móvel.

A implementação escolhida para Holt-Wintes possui apenas cinco parâmetros a serem otimizados, esses variam entre tipos reais e booleanos, e são descritos a seguir:

- *exponential*: parâmetro booleano que indica o tipo de componente de tendência.
- *optimized*: parâmetro booleano que indica se é para estimar os parâmetros do modelo através da maximização da probabilidade logarítmica.
- *smoothing_level*: parâmetro do tipo real que representa o valor alfa da suavização exponencial simples.
- *smoothing_trend*: parâmetro do tipo real que representa o valor beta do método de tendência de Holt.
- *damping_trend*: parâmetro do tipo real que representa o valor phi do método amortecido.

Por fim, os parâmetros da LSTM foram reduzidos devido ao tamanho de possibilidades que poderiam ser explorados com a otimização. A seleção aqui feita foi com base em experiências antigas e na literatura. Os tipos variam entre números reais, números inteiros e *strings* com as opções reduzidas devido ao tempo de otimização. Esses parâmetros são descritos a seguir:

- *units*: número inteiro que representa a dimensionalidade do espaço da saída.
- *dropout*: número real, entre 1 e 0, que representa a fração de unidades que serão descartadas para a transformação linear do estado de recorrência.
- *activation*: parâmetro do tipo *string* que contém a função de ativação que será utilizada. Dentre as opções, temos: "relu", "sigmoid", "elu", "tanh", "selu", "softmax", "softsign".
- *final_activation*: parâmetro do tipo *string* que contém a função de ativação que será utilizada na camada de saída. As opções são as mesmas do item anterior.
- *optimizer*: parâmetro do tipo *string* que define qual o otimizador que será utilizado na compilação do modelo. As opções são:"sgd","rmsprop","adam","adadelta","adagrad".

4.4 Resultados

Essa seção mostra os resultados para os experimentos aqui realizados. Para cada intervalo foram calculados os valores mínimo, médio e o desvio padrão do MAE. Nas tabelas também é possível observar a duração média, em segundos, para cada otimizador, onde é considerado o tempo de treinamento e previsão. Em negrito estão os otimizadores que obtiveram o menor MAE médio nas dez execuções para cada previsor.

Os resultados para o experimento realizado com intervalo de um mês é mostrado na Tabela 2. Apesar do *Simulated Annealing* obter um MAE médio menor para o modelo ARIMA, o

Hill Climbing se destaca pelo baixo tempo de execução, baixo desvio padrão e por ter encontrado uma solução com o menor MAE para esse modelo. Para o *Holt-Winters*, o SNO foi o que obteve o menor MAE médio e com o menor tempo de execução. E, novamente, o *Hill Climbing* apresentou um bom resultado com o menor MAE mínimo, o menor desvio padrão e o valor médio possui pouca diferença comparado com o do SNO. O LSTM obteve as maiores médias de MAE para esse intervalo, e, como esperado, uma alta média de duração.

O experimento realizado para o intervalo de um mês se caracteriza como uma série temporal simples por se tratar da previsão de apenas um valor. Devido a essa característica os previsores baseados em equações clássicas são os que obtiveram um maior destaque. O ARIMA ter sido o modelo com a menor média se dá pela sua linearidade, pois equações lineares tendem a fazer boas previsões quando a saída se trata de um valor único.

Tabela 2 – Resultado para o intervalo de 1 mês.

Modelo	Otimizador	MAE - Minímo	MAE - Média	MAE - Desvio Padrão	Duração(s)
ARIMA	Genetic Algorithm	0.0083	0.0729	0.1001	254.1703
ARIMA	Hill Climbing	0.0025	0.0220	0.0213	128.5629
ARIMA	**Simulated Annealing**	**0.0050**	**0.0197**	**0.0239**	**5,065.3878**
ARIMA	SNO	1.2482	5.5542	6.6192	276.2424
Holt-Winters	Genetic Algorithm	0.0168	0.0877	0.0690	12.2600
Holt-Winters	Hill Climbing	0.0021	0.0749	0.0489	5.8003
Holt-Winters	Simulated Annealing	0.0403	0.1301	0.0628	6.4990
Holt-Winters	**SNO**	**0.0057**	**0.0699**	**0.0621**	**3.4064**
LSTM	**Genetic Algorithm**	**0.0256**	**0.3064**	**0.2218**	**990.7013**
LSTM	Hill Climbing	0.1367	0.5914	0.3536	945.7312
LSTM	Simulated Annealing	0.1355	0.6182	0.6466	830.4104
LSTM	SNO	0.3479	0.5899	0.1525	534.4735

O experimento realizado com intervalo de três meses é mostrado na Tabela 3. A menor média do MAE para esse intervalo foi atingida pelo *Simulated Annealing* para o ARIMA. Esse otimizador achou a menor solução, junto com o *Hill Climbing*, e obteve o menor desvio padrão com uma duração baixa. Para o *Holt-Winters* a menor média veio também com o *Simulated Annealing*, com baixo desvio padrão e baixo tempo de execução médio. O SNO também se destacou por ter encontrado a melhor solução para esse modelo. Apesar de ter sido encontradas boas soluções com o LSTM para esse intervalo, ainda é longe do que foi encontrado com os outros modelos. O tempo de execução continua alto, porém observa-se uma consistência nas soluções encontradas pelo SNO de acordo com desvio padrão baixo.

Como esse intervalo apresenta apenas três pontos, a série continua com baixa complexidade. Assim, o ARIMA, com sua característica de linearidade, mantém o mesmo destaque obtido nos resultados com intervalo de um mês.

A Tabela 4 mostra os resultados para o experimento realizado com um intervalo de seis meses. Alguns resultados inválidos foram encontrados para o ARIMA, representado pelo *inf*. Entretanto esse foi o modelo que obteve a menor média através do *Simulated annealing*, também com o menor desvio padrão e baixa média de duração. O *Holt-Winters* também obteve soluções invalidas em alguma das 10 execuções. A melhor média foi encontrada pelo SNO, com o menor

Tabela 3 – Resultado para o intervalo de 3 meses.

Modelo	Otimizador	MAE - Minímo	MAE - Média	MAE - Desvio Padrão	Duração(s)
ARIMA	Genetic Algorithm	0.0299	0.1845	0.1436	242.2829
ARIMA	Hill Climbing	0.0252	0.1401	0.1660	109.8820
ARIMA	**Simulated Annealing**	**0.0252**	**0.1188**	**0.1204**	**140.4589**
ARIMA	SNO	0.2581	11.7236	22.1103	271.9537
Holt-Winters	Genetic Algorithm	0.2272	0.2701	0.0433	13.7139
Holt-Winters	Hill Climbing	0.2316	0.2966	0.0683	6.3116
Holt-Winters	**Simulated Annealing**	**0.0655**	**0.2412**	**0.0896**	**6.2585**
Holt-Winters	SNO	0.0359	0.2675	0.1239	3.8173
LSTM	Genetic Algorithm	0.1250	0.4559	0.3075	862.9097
LSTM	Hill Climbing	0.2484	1.0870	0.9732	702.1872
LSTM	Simulated Annealing	0.1293	0.7372	0.6232	696.3791
LSTM	**SNO**	**0.1581**	**0.4668**	**0.1881**	**446.0731**

desvio padrão, tempo médio de execução. O SNO teve destaque também para a otimização do LSTM, com a menor média de MAE, o menor desvio padrão e a menor média de duração.

Esse intervalo possui mais pontos que os anteriores, e consequentemente sua complexidade aumenta. Dessa forma, um modelo mais complexo, como o LSTM, encontra soluções que se aproximam das encontradas pelo métodos clássicos. Diferentes dos intervalos anteriores, aqui foram encontradas soluções invalidas. Isso se dá por alguma combinação inválida de parâmetros, o que não ocorre com a LSTM.

Tabela 4 – Resultado para o intervalo de 6 meses.

Modelo	Otimizador	MAE - Minímo	MAE - Média	MAE - Desvio Padrão	Duração(s)
ARIMA	Genetic Algorithm	0.1050	0.2339	0.2436	248.0334
ARIMA	Hill Climbing	0.1018	0.1638	0.0954	108.9324
ARIMA	**Simulated Annealing**	**0.1101**	**0.1506**	**0.0320**	**130.2331**
ARIMA	SNO	0.4304	inf	inf	289.5430
Holt-Winters	Genetic Algorithm	0.1047	inf	inf	12.8932
Holt-Winters	Hill Climbing	0.1477	0.2491	0.0877	6.2056
Holt-Winters	Simulated Annealing	0.1804	0.3448	0.1614	5.7482
Holt-Winters	**SNO**	**0.0824**	**0.1834**	**0.0627**	**3.8507**
LSTM	Genetic Algorithm	0.1216	1.7940	1.5848	1,076.4518
LSTM	Hill Climbing	0.1091	1.4480	1.3018	894.9133
LSTM	Simulated Annealing	0.1866	0.6681	0.3002	852.7977
LSTM	**SNO**	**0.2719**	**0.4990**	**0.2229**	**591.4698**

Por fim, na Tabela 5 encontramos os resultados para o intervalo de um ano. Diferente dos outros intervalos, nesse o melhor resultado foi encontrado com o LSTM, através do *Simulated Annealing*, com o desvio padrão um pouco alto. Porém o ARIMA, otimizado pelo *Simulated Annealing*, foi o que obteve a menor média e o menor desvio padrão, com uma média de duração baixa. Soluções inválidas foram encontradas no ARIMA e no *Holt-Winters*, o LSTM permaneceu intacto nesse quesito.

No último intervalo do experimento, encontra-se a série mais complexa aqui apresentada. Apesar disso, o número de pontos ainda é baixo para ser considerada uma série complexa, e também não existem variáveis exógenas. Logo, os métodos baseados em equações clássicas permanecem mostrando bons resultados. O LSTM foi o modelo que encontrou a melhor solução,

pois aqui sua memória de curto-prazo tem mais valor, por existirem mais pontos para serem usados na recorrência.

Tabela 5 – Resultado para o intervalo de 12 meses.

Modelo	Otimizador	MAE - Minímo	MAE - Média	MAE - Desvio Padrão	Duração(s)
ARIMA	SNO	0.2139	inf	inf	266.4057
ARIMA	**Simulated Annealing**	**0.1270**	**0.1663**	**0.0449**	**128.8625**
ARIMA	Genetic Algorithm	0.1270	0.2667	0.3108	220.1838
ARIMA	Hill Climbing	0.1270	0.1957	0.1594	121.2087
Holt-Winters	**SNO**	**0.1813**	**0.3866**	**0.2111**	**4.0453**
Holt-Winters	Genetic Algorithm	0.3559	0.4500	0.0636	14.0728
Holt-Winters	Hill Climbing	0.4281	inf	inf	6.6021
Holt-Winters	Simulated Annealing	0.3087	0.4632	0.0813	6.6964
LSTM	Simulated Annealing	0.0957	0.5740	0.4092	907.5365
LSTM	**SNO**	**0.2966**	**0.4950**	**0.1772**	**601.7424**
LSTM	Genetic Algorithm	0.1102	0.8973	0.8757	1,326.9063
LSTM	Hill Climbing	0.1938	1.2932	1.0147	778.2387

4.5 Considerações finais

Em vista do que foi apresentado nos resultados dos experimentos, os modelos apresentaram comportamentos peculiares para cada intervalo, como esperado. No cenário de um mês, o modelo ARIMA com o otimizador *Hill Climbing* obtêm os melhores resultados por ter encontrado uma solução com o MAE muito baixo, sua média também é baixa, e seu desvio padrão mostra uma consistência nas soluções encontradas além de um tempo de execução interessante.

Para os cenários de três e seis meses, a melhor escolha se torna o ARIMA otimizado pelo *Simulated Annealing*. Pois esse possui um MAE médio discrepante das outras opções em ambos os cenários, mantendo baixos desvio padrão e tempo de execução.

No último cenário, de doze meses, o LSTM otimizado pelo *Simulated Annealing*, consegue quebrar o limite imposto pelo ARIMA e acha uma solução que se destaca das demais. Entretanto, o ARIMA com o *Simulated Annealing* possui os melhores resultados com um baixo tempo de execução e a consistência nas soluções permanece para esse cenário.

5
Conclusão

Nesse foi trabalho foi feito um estudo sobre os métodos de configuração automática de parâmetros em modelos de previsão em séries temporais. Os resultados das otimizações dos modelos ARIMA e *Holt-Winters*, foram comparados com os resultados de uma Rede Neural Recorrente do tipo LSTM. Essa comparação foi feita em quatro intervalos de tempo diferentes, onde foi calculado a Média do Erro Absoluto (MAE) entre os valores da previsão e os reais.

Através da disponibilização dos dados pela SEFAZ/SE, foi possível fazer a previsão da arrecadação para períodos diferentes, com os três modelos mencionados. Assim sendo, foram construídos meta-modelos para otimizar os hiper-parâmetros de cada previsor tendo em vista a minimização do MAE. Esses meta-modelos, chamados de otimizadores, foram construídos a partir dos algoritmos: *Hill-Climbing*, *Simulated Annealing*, *Genetic Algorithm* e *Social Network Optimization*.

Com os resultados aqui apresentados foi possível observar uma dominância dos modelos clássicos ARIMA e *Holt-Winters*, sobre o LSTM. Essa dominância se dá todos os quesitos: média do MAE, desvio padrão e tempo de execução. A duração do treinamento da LSTM é bastante extensivo, e o espaço a ser explorado gera inúmeras possibilidades, as quais tiveram que ser podadas para tornar a otimização factível.

Para trabalhos futuros torna-se necessário expandir esse espaço levando em consideração um maior número de parâmetros para otimização, e um maior conjunto de opções para cada parâmetro. Para que haja a possibilidade disso ocorrer se torna imprescindível a aplicação de algum modelo *Surrogate* durante o processo de otimização do modelo LSTM. Também diminuindo o custo computacional. A aplicação de técnicas para divisão dos dados entre treinamento e teste, durante a fase de experimentos garantiriam uma maior consistência dos resultados.

Referências

BAKHASHWAIN, N.; SAGHEER, A. Online tuning of hyperparameters in deep lstm for time series applications. *International Journal of Intelligent Engineering and Systems*, v. 14, n. 1, p. 212–220, 2021. Citado na página 16.

CAI, Z.; LONG, Y.; SHAO, L. Classification complexity assessment for hyper-parameter optimization. *Pattern Recognition Letters*, Elsevier, v. 125, p. 396–403, 2019. Citado na página 15.

CHOLLET, F. et al. *Keras*. 2015. <https://keras.io>. Citado na página 19.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995. Citado na página 9.

GAMBELLA, C.; GHADDAR, B.; NAOUM-SAWAYA, J. Optimization problems for machine learning: A survey. *European Journal of Operational Research*, Elsevier, v. 290, n. 3, p. 807–828, 2021. Citado 2 vezes nas páginas 12 e 13.

GURNEY, K. *An introduction to neural networks*. [S.l.]: CRC press, 2018. Citado 2 vezes nas páginas 9 e 13.

HEYDARI, M. et al. Application of holt-winters time series models for predicting climatic parameters (case study: Robat garah-bil station, iran). *Polish J Environ Stud*, v. 29, p. 617–27, 2019. Citado na página 15.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, v. 9, p. 1735–80, 12 1997. Citado 2 vezes nas páginas 9 e 10.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 13.

HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: MIT press, 1992. Citado na página 11.

JACOBSON, S. H.; SULLIVAN, K. A.; JOHNSON, A. W. Discrete manufacturing process design optimization using computer simulation and generalized hill climbing algorithms. *Engineering Optimization*, Taylor & Francis, v. 31, n. 2, p. 247–260, 1998. Citado na página 11.

KIRKPATRICK, S.; JR, C. D. G.; VECCHI, M. P. Optimization by simulated annealing. *science*, American association for the advancement of science, v. 220, n. 4598, p. 671–680, 1983. Citado na página 11.

LIU, Z. et al. Forecast methods for time series data: A survey. *IEEE Access*, v. 9, p. 91896–91912, 2021. Citado 3 vezes nas páginas 9, 15 e 16.

LONES, M. *Sean Luke: essentials of metaheuristics*. [S.l.]: Springer, 2011. Citado na página 16.

LUKE, S. *Essentials of metaheuristics*. [S.l.]: Lulu Raleigh, 2013. v. 2. Citado 3 vezes nas páginas 20, 21 e 23.

QUINLAN, J. R. Simplifying decision trees. *International journal of man-machine studies*, Elsevier, v. 27, n. 3, p. 221–234, 1987. Citado na página 9.

RATANAMAHATANA, C. A.; KEOGH, E. Multimedia retrieval using time series representation and relevance feedback. In: FOX, E. A. et al. (Ed.). *Digital Libraries: Implementing Strategies and Sharing Experiences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 400–405. ISBN 978-3-540-32291-7. Citado na página 9.

REBALA, G.; RAVI, A.; CHURIWALA, S. Machine learning definition and basics. In: ____. *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2019. p. 1–17. ISBN 978-3-030-15729-6. Disponível em: <https://doi.org/10.1007/978-3-030-15729-6_1>. Citado na página 8.

SEABOLD, S.; PERKTOLD, J. statsmodels: Econometric and statistical modeling with python. In: *9th Python in Science Conference*. [S.l.: s.n.], 2010. Citado na página 19.

SHELATKAR, T. et al. Web traffic time series forecasting using arima and lstm rnn. In: EDP SCIENCES. *ITM Web of Conferences*. [S.l.], 2020. v. 32, p. 03017. Citado 2 vezes nas páginas 10 e 17.

SHELATKAR, T. et al. Web traffic time series forecasting using arima and lstm rnn. In: EDP SCIENCES. *ITM Web of Conferences*. [S.l.], 2020. v. 32, p. 03017. Citado na página 15.

SHERAFAT, H. Social network optimization a new methaheuristic for general optimization problems. *REVISTA GEINTEC-GESTAO INOVACAO E TECNOLOGIAS*, v. 7, n. 4, p. 4123–4130, 2017. Citado 4 vezes nas páginas 11, 16, 17 e 23.

SIMEONE, O. A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, IEEE, v. 4, n. 4, p. 648–664, 2018. Citado na página 8.