UNIVERSIDADE FEEVALE

LUIS MIGUEL PIRES

CONSTRUÇÃO DE UM MODELO PARA PREDIÇÃO DE CLIQUES EM SISTEMAS BASEADOS EM INTERFACES GRÁFICAS

Novo Hamburgo

2020

LUIS MIGUEL PIRES

CONSTRUÇÃO DE UM MODELO PARA PREDIÇÃO DE CLIQUES EM SISTEMAS BASEADOS EM INTERFACES GRÁFICAS

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do grau de Bacharel em
Ciência da Computação pela
Universidade Feevale

Orientador: Gabriel da Silva Simões

Novo Hamburgo
2020

LUIS MIGUEL PIRES

CONSTRUÇÃO DE UM MODELO PARA PREDIÇÃO DE CLIQUES EM SISTEMAS BASEADOS EM INTERFACES GRÁFICAS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale.

Aprovado por:

_____

Orientador: Gabriel da Silva Simões

_____

Professor avaliador: Edvar Bergmann Araujo

_____

Professor avaliador: Rodrigo Rafael Villarreal Goulart

Data da aprovação:

AGRADECIMENTOS

Agradeço a todos que, de alguma forma, contribuíram para a realização deste trabalho, tão como na minha jornada acadêmica até agora. Especialmente aos meus pais, que sempre me incentivaram a continuar com os estudos e me proporcionaram a chance de estudar e trabalhar na área que escolhi. Agradeço também à minha esposa que sempre apoiou minhas escolhas e esteve sempre presente durante minha jornada. Gostaria também de agradecer meu orientador, que foi de extrema importância para o andamento do projeto e esteve sempre disposto a ajudar na evolução do mesmo, e também aos profissionais da empresa que disponibilizou os dados que tornaram este trabalho possível.

RESUMO

Recentemente o modelo de entrega de softwares denominado *Software as a Service* (SaaS) têm ganhado muita popularidade. A previsão de crescimento deste modelo de software entre 2018 e 2023 é de 21,2% ao ano. Essa rápida expansão gera um esforço para o treinamento de novos usuários e de equipe de suporte técnico aos clientes, ocasionando um alto custo operacional e dificultando a competitividade das empresas. Existem diversos serviços que visam facilitar a adaptação de novos usuários em sistemas online, contudo, estes sistemas necessitam de mão de obra técnica para serem configurados e podem apresentar inconsistências ao longo do tempo de vida do software. Com base nesse cenário, o presente trabalho propõe um modelo preditivo com base no monitoramento das ações de usuários de um determinado sistema, permitindo a inferência de futuras interações realizadas em uma dada tela por novos usuários, automatizando o processo de configuração de guias interativos. Para a realização dos treinamentos do modelo preditivo, foi necessário coletar dados de interações dos usuários ativos através de um *plugin* criado especificamente para este fim. De acordo com os experimentos realizados, identificou-se que a melhor configuração da rede neural para o cenário estudado atingiu uma acurácia de 58,14% na predição de um conjunto com 81 classes possíveis. Entende-se que os resultados obtidos neste trabalho são satisfatórios, dado que a probabilidade arbitrária de acerto em um problema com a mesma quantidade de classes é de 1,23%.

Palavras-chave: modelos preditivos, aprendizado de máquina*, user onboarding*.

ABSTRACT

Recently the software delivery model called Software as a Service (SaaS) has gained a lot of popularity. The growth forecast for this software model between 2018 and 2023 is 21.2% per year. This rapid expansion creates an effort to train new users and customer support staff, resulting in high operational costs, making it difficult for companies to compete. There are several services aimed at facilitating the adaptation of new users to online systems, however, these systems require technical skills to be configured and may present inconsistencies over the lifecycle of the software. Based on this scenario, this paper proposes a predictive model based on the monitoring of actions of active users in a system, allowing the inference of future interactions performed on a given screen by new users, automating the process of configuring interactive walkthroughs. To perform the training of the predictive model, it was necessary to collect interaction data from active users through a plugin created specifically for this purpose. According to the executed experiments, it was found that the best configuration of the neural network for the studied scenario reached an accuracy of 58.14% in the prediction of a set with 81 possible classes. It is understood that the results obtained in this work are satisfactory, once the arbitrary probability of success in a problem with the same number of classes is 1.23%.

Keywords: predictive models, machine learning, user onboarding.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
FGV	Fundação Getulio Vargas
FIFO	*First in First Out*
GPU	*Graphics processing unit*
LSTM	*Long Short-Term Memory*
RNA	Rede Neural Artificial
RNNs	*Recurrent Neural Networks*
SaaS	*Software as a Service*
TLU	*Threshold Logic Unit*

SUMÁRIO

1 INTRODUÇÃO

A utilização de sistemas de informação para gerenciamento de empresas vêm crescendo a cada dia, acompanhando a facilidade de acesso aos computadores e à internet no Brasil. De acordo com pesquisa realizada pela Fundação Getulio Vargas (FGV), a base instalada de computadores no Brasil teve uma evolução de 9% em 2018, com um total de 177,5 milhões de computadores em uso neste período (MEIRELLES, 2019).

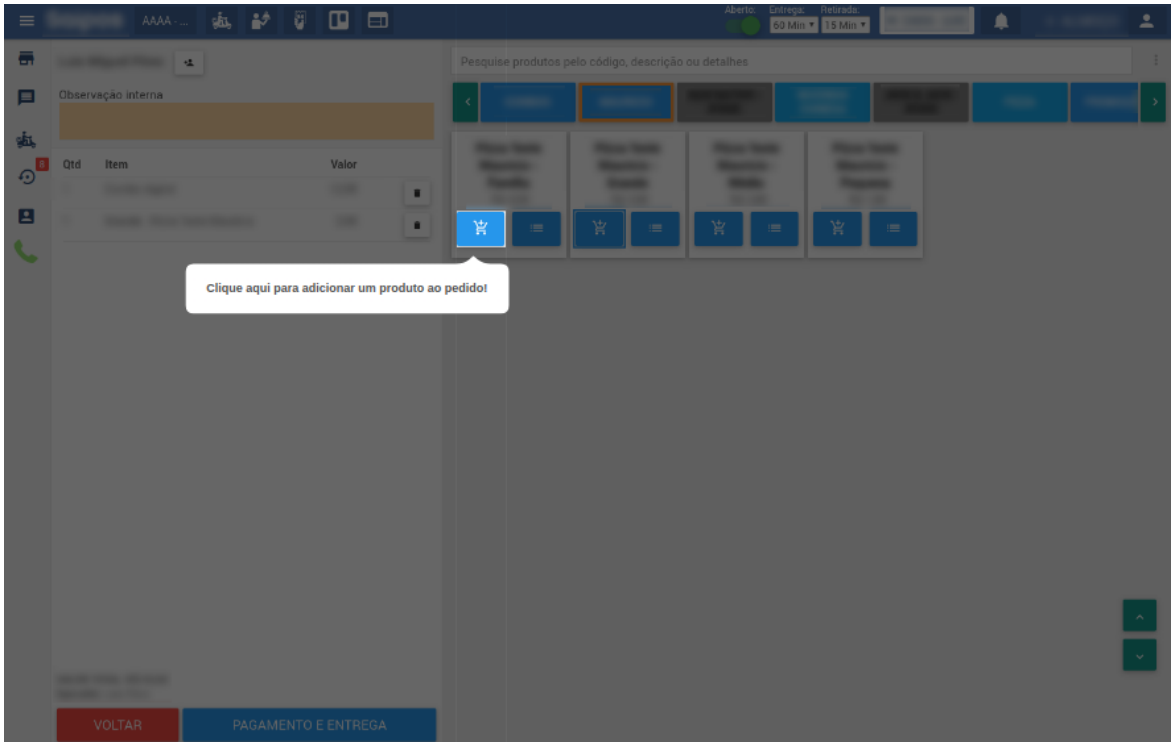
Acompanhando esta constante evolução, empresas de *software* buscam aprimorar a forma de entrega e licenciamento dos seus produtos adotando um modelo denominado *Software as a Service* (SaaS), que tem ganhado popularidade, com uma previsão de crescimento à uma taxa anual de 21,2% entre os anos de 2018 e 2023 (BUSINESS WIRE, 2018). No modelo SaaS, o cliente paga pela utilização de um sistema sem que haja necessidade de uma instalação local em um servidor com *hardware* dedicado, já que o acesso ao serviço é feito totalmente *online* através de um computador, *tablet* ou *smartphone*. Essa modalidade de licenciamento permite ao cliente ter acesso ao sistema de uma forma muito mais barata e rápida quando comparado aos modelos tradicionais de comercialização (DUBEY; WAGLE, 2007). Ainda segundo Dubey (2007), o modelo SaaS isenta o cliente de problemas comuns à manutenção de *softwares* em estruturas próprias, como atualização de servidores ou desenvolvimento de estruturas de segurança.

Uma dada empresa de *softwares* situada na região metropolitana de Porto Alegre, fornece acesso a um sistema para gerenciamento de restaurantes no modelo SaaS. Atualmente esta empresa possui aproximadamente 610 clientes ativos com uma taxa de crescimento de aproximadamente 11% ao mês. Essa taxa de crescimento gera uma alta demanda de suporte técnico e treinamento, exigindo uma equipe de 13 profissionais no setor responsável por estas atividades, resultando em um custo direto de aproximadamente R$ 27.000,00 ao mês para a empresa. Em média, cada novo usuário necessita de aproximadamente 1 hora e 30 minutos de treinamento e, de acordo com a taxa atual de crescimento,

aproximadamente 105 horas ao mês são investidas para que novos clientes iniciem a utilização do sistema. Neste contexto o *User Onboarding*, processo pelo qual o usuário passa no primeiro contato com o sistema, torna-se um fator crucial nos negócios, uma vez que, caso o usuário não saiba como usar a aplicação, ele irá procurar uma solução alternativa (HUCKO et al., 2019).

Diversos modelos de *User Onboarding* são conhecidos, tais como: tutoriais em vídeo, manuais em texto, sessões práticas guiadas, tanto online quanto presenciais. Recentemente algumas soluções têm apresentado uma abordagem de guias interativos através de elementos visuais sobrepostos na tela do sistema, como é o caso do serviço chamado *Appcues*. Outros serviços que também oferecem soluções similares são: *Userlane, Intercom e Userpilot.* O exemplo de um guia interativo implementado em uma tela de um sistema *web* pode ser visto na Figura 1.

Figura 1. Exemplo de guia interativo.



Fonte: Elaborado pelo autor.

Para que seja possível exibir os guias interativos no sistema é necessário baixar uma ferramenta para configuração dos elementos que serão exibidos aos usuários. Após cadastrar as telas onde serão exibidos os guias, deve-se iniciar o processo de configuração, definindo em cada tela do sistema quais elementos serão exibidos, suas ordens, e suas características tais como: i) posicionamento, ii) formato, iii) cores, iv) textos e, finalmente, v) ícones (APPCUES, 2019). Configurar estes guias e mantê-los atualizados é uma tarefa que depende de mão de obra técnica com alto entendimento de regras de negócio do sistema onde o referido guia será aplicado.

Para melhorar este cenário, o presente trabalho estudou e aplicou técnicas que possibilitaram descobrir padrões de comportamento dos usuários já ativos do sistema, para assim prever possíveis ações a serem tomadas em uma determinada tela por novos usuários. Em função da falta de algoritmos facilmente descritíveis para solução de problemas neste contexto, dada a quantidade e complexidade das possíveis entradas (interações dos usuários), serão utilizadas técnicas de Aprendizado de Máquina (AM) para encontrar uma solução viável. Para fins de padronização, a partir deste momento o sistema utilizado como base para esta pesquisa será denominado “sistema monitorado”.

A opção adotada por este trabalho em utilizar técnicas de aprendizado de máquina assume o pressuposto de que um sistema pode adquirir conhecimento de forma automática com base em exemplos ou experiências acumuladas pela solução de problemas anteriores (MONARD; BARANAUSKAS, 2003, p. 39). O conceito de rede neural utilizado neste trabalho foi o *Long-Short Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997), uma estratégia recorrente de Redes Neurais, também conhecidas por *Recurrent Neural Networks* (RNNs). Com base na literatura (NELSON; PEREIRA; DE OLIVEIRA, 2017), RNNs apresentam resultados promissores quando os atributos não podem ser avaliados individualmente para cada instância isolada, mas sim quando o problema depende de sequências temporais (LIPTON; BERKOWITZ, 2015). RNNs são redes em que os dados de saída retroalimentam os dados de entrada da mesma rede, ou seja, as saídas atuais

são influenciadas pelas saídas anteriores (VELLASCO, 2007). No contexto deste trabalho, estes métodos de aprendizado de máquina permitem induzir modelos preditivos com base em exemplos conhecidos que, neste caso, foram coletados de usuários já ativos no sistema monitorado.

Para coletar os dados necessários para a criação dos modelos preditivos foi desenvolvido, no contexto deste trabalho, um *plugin* que possibilitou a captura das interações dos usuários no sistema monitorado. Após a captura, os dados passaram por um pré-processamento, onde foram extraídos os atributos necessários para criação dos conjuntos utilizados no treinamento dos modelos.

Com os conjuntos de dados pré-processados e compilados, foram implementadas rotinas para o treinamento, validação e predição utilizando a API Keras (KERAS, 2020) e o *framework* TensorFlow (TENSORFLOW, 2020). Através de variações nos parâmetros da rede neural, foi possível realizar experimentos para determinar quais configurações apresentam melhores resultados no cenário estudado. A partir destes resultados foi possível determinar a acurácia que um modelo baseado em redes neurais recorrentes pode atingir ao predizer cliques de um usuário em um sistema baseado em interfaces gráficas.

Este trabalho está organizado em 7 capítulos, sendo que o primeiro trata-se desta Introdução. O segundo capítulo faz uma análise de 5 trabalhos relacionados encontrados durante as pesquisas. No terceiro capítulo é realizada uma revisão bibliográfica sobre aprendizado de máquina e redes LSTM. A seguir, no quarto capítulo, é apresentada uma descrição sobre a ferramenta de coleta de dados desenvolvida e os passos de pré-processamento aplicados nos dados coletados. O quinto capítulo descreve a criação e configuração do algoritmo utilizado nos treinos dos modelos utilizados na análise experimental, que encontra-se no capítulo 6 e descreve detalhadamente os resultados obtidos ao executar o treinamento dos modelos variando parâmetros da rede neural. Por fim, no capítulo 7 encontra-se a conclusão dos resultados obtidos e alguns apontamentos para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Diversos trabalhos têm realizado experimentos na predição de sequências de acessos em sistemas *web* com o objetivo de otimizar serviços de *cache* para melhorar a experiência de navegação dos usuários (NANOPOULOS; KATSAROS; MANOLOPOULOS, 2001; FRIAS-MARTINEZ; KARAMCHETI, 2002; GÜNDÜZ; ÖZSU, 2003; BERNHARD et al., 2016). Acompanhando a evolução no estudo de uso de redes neurais para treinamento de modelos preditivos, vários autores buscam aperfeiçoar a acurácia dos modelos e torná-los mais robustos para atender suas necessidades. A seguir serão apresentadas quatro iniciativas relacionadas à temática abordada por este trabalho.

2.1 UTILIZAÇÃO DE MODELOS PREDITIVOS PARA *PREFETCHING*

Nanopoulos, Katsaros e Manolopoulos (2001) definem o objetivo do *prefetching* como a redução da latência de resposta percebida pelos usuários. Fontes potenciais de latência são: i) a alta carga em servidores de aplicação, ii) sobrecarga de rede, iii) baixa largura de banda, iv) banda subutilizada e, finalmente, v) atrasos de propagação das informações. A solução óbvia para resolver estes problemas, aumentar a largura de banda da rede de internet, é por muitas vezes inviável, uma vez que a infraestrutura da rede não pode ser facilmente modificada sem grandes investimentos econômicos (NANOPOULOS; KATSAROS; MANOLOPOULOS, 2001).

As principais vantagens de implementar a utilização de um sistema de *prefetching* são: i) a prevenção de subutilização de largura de banda da rede e ii) a redução da percepção da latência de resposta. Mesmo assim, um modelo “agressivo” pode causar um tráfego excessivo na rede, transitando dados que não serão efetivamente utilizados pelos usuários (NANOPOULOS; KATSAROS; MANOLOPOULOS, 2001).

O trabalho de Nanopoulos, Katsaros e Manolopoulos (2001) indica que o melhor local para realizar predições sobre futuras requisições é no próprio servidor, uma vez que este armazena parte significante dos *logs* de acessos. Sendo assim, os autores propõe que exista uma arquitetura com um “motor de predição” em que o servidor envia “dicas de *prefetching*” anexadas às respostas dos documentos solicitados pelos usuários, possibilitando assim que o cliente (neste caso o navegador) possa realizar o processo de *prefetching* quando estiver ocioso (NANOPOULOS; KATSAROS; MANOLOPOULOS, 2001).

Para possibilitar o envio das dicas de *prefetching* é proposto um algoritmo com base na avaliação de outros 4 algoritmos com o mesmo objetivo. Diversos testes foram realizados e, de acordo com os autores, utilizando uma base de dados real, a acurácia do algoritmo proposto foi de 29% no melhor caso (NANOPOULOS; KATSAROS; MANOLOPOULOS, 2015).

2.2 MODELO PREDITIVO COM MÉTODO DE PERSONALIZAÇÃO

Frias-Martinez e Karamcheti (2002) utilizam o conceito de “sequência de acessos” para criar um modelo de regras de associação sequencial que possibilita capturar sequencialmente e temporalmente quais páginas são visitadas em um site. Também é utilizado o conceito de “distância” entre os eventos, que permite ao modelo não apenas prever quais páginas serão acessadas, mas também quando isso ocorrerá (FRIAS-MARTINEZ; KARAMCHETI, 2002).

Os dados utilizados para realizar o trabalho são *logs* de requisições em servidores *web* contendo: IP do usuário, data e hora do acesso, método da requisição (*GET, POST,* etc*)*, URL da página acessada, protocolo utilizado (HTTP 1.0, HTTP 1.1, etc), código de retorno e número de *bytes* transferidos. Com estes dados é possível definir um conjunto de sessões servidas pelo site e então agrupá-las de acordo com cada usuário (neste caso o endereço IP) (FRIAS-MARTINEZ; KARAMCHETI, 2002).

Os testes foram realizados em três conjuntos de dados, visando contemplar diferentes tipos de sites, desde sites pequenos (menos complexos) com poucos usuários, até sites complexos com um grande volume de visitas. De acordo com os autores, o site com menor complexidade apresentou resultado satisfatório na predição de acessos com uma taxa de 45% de acerto contra 15% do site com maior complexidade e maior número de acessos (FRIAS-MARTINEZ; KARAMCHETI, 2002). Segundo Frias-Martinez e Karamcheti (2002), a baixa acurácia do modelo em sites de grande complexidade se deve ao fato de que os mesmos apresentam uma estrutura altamente interconectada (não seguindo a estrutura de árvore) e, diferente dos sites com menor número de páginas e usuários, não podem ser descritos através de um Modelo Sequencial de Comportamento.

Buscando aperfeiçoar os resultados para sites de alta complexidade, Frias-Martinez e Karamcheti (2002) introduzem a utilização de um método de “personalização” no modelo proposto, elevando a taxa de acertos para 53% nestes tipos de sites. Esta abordagem não causa um grande aumento no armazenamento necessário para os *logs*, porém é limitada à usuários que visitam o site com frequência, uma vez que não é possível criar regras personalizadas para usuários que não mantém uma certa taxa de visitas (FRIAS-MARTINEZ; KARAMCHETI, 2002).

2.3 SISTEMA DE RECOMENDAÇÕES BASEADO EM SESSÕES

Gündüz e Özsu (2003) propuseram um modelo que considera tanto a sequência das páginas visitadas quanto o tempo de permanência dos usuários nas mesmas. A justificativa para a inclusão do tempo de permanência ao modelo preditivo é de que utilizar apenas a sequência de páginas visitadas não é suficiente para descrever o comportamento de navegação dos usuários com acurácia. O modelo proposto é utilizado para um sistema de recomendação mas, conforme os autores, pode ser aplicado também à um sistema de *prefetching*.

O algoritmo apresentado pode ser resumido da seguinte maneira: As sessões dos usuários são agrupadas baseado em sua similaridade e então, quando uma requisição é recebida de um usuário ativo, o algoritmo cria uma lista de recomendações contendo três diferentes páginas ainda não visitadas por este usuário, utilizando como base a sessão com maior similaridade do modelo. De acordo com os autores, a novidade proposta pelo modelo é a forma de calcular a similaridade das sessões de usuários e como agrupá-las (GÜNDÜZ; ÖZSU, 2003).

Para realizar o treinamento dos modelos foram capturados dados de *logs* de dois servidores e então realizado um pré-processamento para extrair as informações necessárias e calcular o tempo de permanência em cada página, que é determinado através a diferença entre o tempo entre duas requisições em uma mesma sessão. A saída deste pré-processamento foi um conjunto de sessões de usuários contendo as requisições ordenadas por data e hora de acesso (GÜNDÜZ; ÖZSU, 2003).

Após a etapa de pré-processamento os dados foram separados para efetivamente serem utilizados no treinamento dos modelos, onde aproximadamente 30% do volume de dados foi randomicamente separado para a etapa de testes e o restante para treino. No resultados, os autores consideram duas medidas: *Hit-Ratio*, onde "*Hit*" significa um clique em qualquer uma das três recomendações na requisição imediatamente após a última requisição; *Click-Soon-Ratio,* onde "*Click-Soon"* significa que o usuário visitou uma das páginas recomendadas em qualquer momento até o final da sessão atual (GÜNDÜZ; ÖZSU, 2003).

Em um primeiro teste, o algoritmo foi executado sem a rotina de agrupamento de sessões por similaridade e apresentou um *Hit-Ratio* de 61,61% no melhor caso e 51,29% no pior caso. O teste seguinte foi realizado adicionando-se a rotina de agrupamento e variando parâmetros como número total de grupos, normalização do tempo de permanência na página e quantidade de nodos utilizados no processo de predição, apresentando um *Hit-Ratio* de 59,79% no melhor caso e 35,65% no pior caso (GÜNDÜZ; ÖZSU, 2003).

2.4 PREDIÇÕES DE *CLICKSTREAM* UTILIZANDO CADEIAS DE MARKOV

Bernhard et al. (2016) utilizam o termo *Clickstream* para descrever os registros de cliques sequenciais de usuários navegando na internet. O trabalho destaca que a aplicação de modelos de predição em sequências de cliques ajuda a aprimorar o design das páginas web, uma vez que os padrões descobertos podem gerar conhecimento sobre caminhos de navegação comuns que podem ser “encurtados”, criando-se links diretos para páginas frequentemente visitadas em sequência. Também é citada a possibilidade de melhoria em serviços de *cache* das páginas, aprimorando o desempenho de aplicações *web* de uma forma geral através do *prefetching,* carregando recursos no *cache* antes mesmo da solicitação explícita do usuário, (BERNHARD et al., 2016).

A pesquisa de Bernhard et al. destaca que muitos algoritmos “exatos” de mineração de dados podem ter um alto custo computacional, pois dependem de muitas iterações no volume de dados durante o processo de mineração, tornando inviável realizá-lo dinamicamente enquanto os usuários visitam a aplicação. Para resolver este problema o trabalho propõe três “algoritmos de mineração sequencial aproximados” que tem a capacidade de descobrir sequências frequentes em um *clickstream* e usa essas sequências para construir uma Cadeia de Markov que pode representar os dados através de um modelo estatístico (BERNHARD et al., 2016).

O conjunto de dados utilizado para criação dos algoritmos possui dados de acesso à um site contendo 17 diferentes categorias em que os usuários podem navegar, onde cada registro corresponde à um *clickstream* de um usuário específico. A matriz de transição inicial foi construída com base em 10.000 registros e outros 10.000 foram usados para determinar a acurácia das predições me cada um dos algoritmos (BERNHARD et al., 2016).

De acordo com os resultados do trabalho, o algoritmo que apresentou o melhor resultado, teve uma média de tempo de execução de 279 milissegundos e uma taxa de 47.48% de acerto. Este mesmo algoritmo também foi o que apresentou

uma menor taxa de crescimento de tempo de execução quando utilizado em um volume de dados com mais categorias (BERNHARD et al., 2016).

2.5 CONSIDERAÇÕES

Os trabalhos apresentados acima descrevem modelos para predição de visitas de usuários em páginas *web*. Estes trabalhos utilizam *logs* de acessos em servidores para treinamento de seus modelos, uma vez que o objetivo dos mesmos é a predição de acessos à documentos (ou páginas), não importando a forma como o usuário chegou até este recurso.

O presente trabalho diferencia-se das pesquisas acima citadas pelo fato de que foram utilizados dados de cliques em elementos gráficos de telas do sistema monitorado, ao invés de *logs* de acesso aos servidores para inferir modelos preditivos. Dessa forma foi possível treinar modelos para predizer sequências de cliques de um usuário baseado no conjunto de dados coletados no sistema monitorado, permitindo um monitoramento fino em nível de componentes de tela, ao invés de acessos a páginas como um todo.

3 APRENDIZADO DE MÁQUINA

Segundo Monard e Baranauskas (2003, p. 39), Aprendizado de Máquina é uma área de inteligência artificial onde os sistemas são capazes de adquirir conhecimento de forma automática por meio de experiências acumuladas através da solução de problemas anteriores. Formalmente, Mitchell (1997, p. 2) define aprendizado de máquina como: “Diz-se que um programa de computador aprende com a experiência *E* em relação a alguma classe de tarefas *T* e medida de desempenho *P*, se seu desempenho nas tarefas em *T*, medido por *P*, melhorar com a experiência *E*.”

A inferência indutiva permite aprender através de exemplos para então predizer exemplos futuros. O aprendizado indutivo é realizado a partir de exemplos fornecidos ao sistema de aprendizado e pode ser classificado como supervisionado e não-supervisionado (MONARD; BARANAUSKAS, 2003, p. 40). Para realizar o aprendizado supervisionado, é necessário fornecer ao algoritmo de aprendizado exemplos em que já se conhece o rótulo da classe associada para problemas de classificação, ou valor real para problemas de regressão (ALPAYDIN, 2010, p. 11).

De forma geral, estes exemplos são constituídos por uma série de atributos (*features*) e um rótulo (*label*) da classe associada, sendo que o objetivo do modelo é construir um classificador que possa determinar a classe de novos elementos que ainda não se conhece o rótulo. Este tipo de problema é chamado de *classificação*, pois para cada conjunto de atributos existe um único rótulo de classe correspondente (MONARD; BARANAUSKAS, 2003, p. 40). Um exemplo de problema de classificação é a detecção de fraudes em compras, onde os atributos de entrada são os dados da compra e o rótulo é a informação de “fraude” ou “compra legítima”.

Quando a saída do modelo assume valores contínuos, o problema passa a ser chamado de *regressão* (MONARD; BARANAUSKAS, 2003, p. 40). Em problemas de regressão, dado uma quantidade de atributos como, marca do veículo,

potência do motor, cor, quantidade de *airbags*, etc, pode-se inferir o valor de venda do mesmo.

Além dos problemas clássicos de regressão e classificação, métodos de aprendizado de máquina baseados em redes neurais vem definindo o novo estado-da-arte para problema de segmentação semântica (LONG; SHELHAMER; DARRELL, 2015), geração automática de imagens (GOODFELLOW et al., 2014), detecção de objetos (REN et al., 2015) e geração automática de textos (HE; DENG, 2017). Em comum, todos os métodos dependem de arquiteturas de redes neurais profundas e de um expressivo volume de dados para treinamento.

No aprendizado não-supervisionado, o modelo tenta formar agrupamentos através dos exemplos fornecidos para que, a partir de então, em uma análise com base no contexto do problema seja determinado o que cada agrupamento significa (MONARD; BARANAUSKAS, 2003, p. 40).

3.1 REDES NEURAIS ARTIFICIAIS

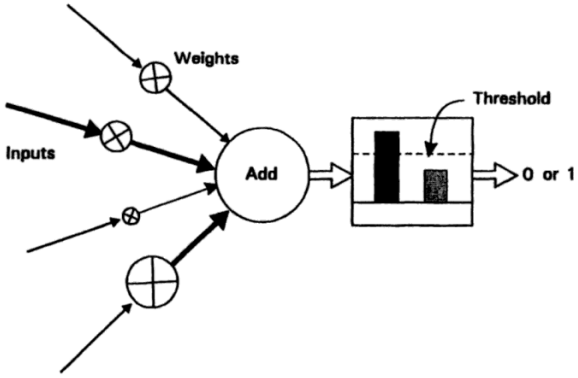
De acordo com Vellasco (2007), uma Rede Neural Artificial (RNA) tem o objetivo de imitar a capacidade que o cérebro tem de reconhecer, associar e generalizar padrões. Dessa forma é possível resolver uma gama de problemas complexos não-lineares, sendo útil quando não é possível definir um modelo explícito ou uma lista de regras para o algoritmo. O processo de aprendizagem de uma rede neural consiste em otimizar iterativamente os parâmetros (também conhecidos como pesos) da rede para um determinado contexto, de maneira a produzir resultados coerentes com este contexto. (CORRÊA, 2008).

3.1.1 Neurônio Artificial

Rauber (2005) define um neurônio artificial como uma entidade de processamento simples, responsável por calcular uma função de saída a partir de entradas e pesos, aplicando um método de ativação predefinido. O modelo mais

simples e mais antigo de neurônio artificial é chamado *Threshold Logic Unit* (TLU) (GURNEY, 1997, p. 14) e pode ser representado conforme a Figura 2. Ele computa uma soma ponderada de entradas, compara essa soma a um valor limiar e retorna 1 caso o valor exceda este limiar ou 0 caso contrário (NILSSON, 1998).

Figura 2. Representação de uma TLU.



Fonte: GURNEY (1997)

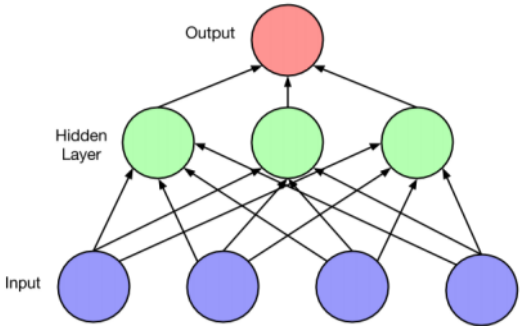
3.1.2 Topologia

As RNAs são formadas por camadas de neurônios ligados através de conexões sinápticas, sendo que alguns destes neurônios recebem estímulos externos à rede, formando a camada de entrada da mesma. Outra camada importante da RNA é a camada de saída, através da qual uma rede neural representa os resultados preditivos para uma dada instância de entrada. Além destas, podem existir camadas “escondidas”, distribuídas entre a camada de entrada e a camada de saída, que possibilitam a uma RNA resolver problemas não linearmente separáveis (BARRETO, 2002). Existem basicamente dois tipos de topologia de RNAs: Redes Recorrentes e Redes não Recorrentes (VELLASCO, 2007).

As Redes não Recorrentes (*Feedfoward*) são aquelas cujo grafo não possui ciclos (BARRETO, 2002), ou seja, o fluxo de dados é unidirecional e não existe relação temporal entre as diferentes predições (VELLASCO, 2007). Exemplos desse

tipo de rede são o *perceptron* (ROSENBLATT, 1958) e o *perceptron* multi-camada (RUMELHART; HINTON; WILLIAMS, 1985). A Figura 3 apresenta um exemplo deste tipo de rede.

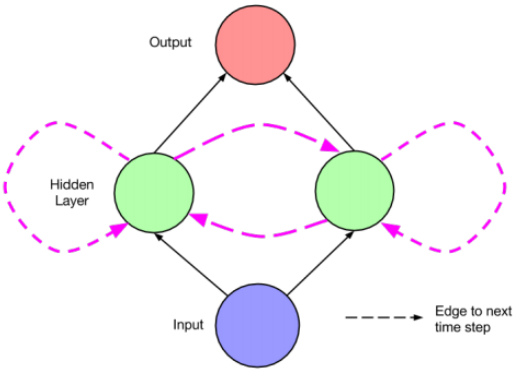
Figura 3. Rede não Recorrente.



Fonte: LIPTON (2015)

Já as *Recurrent Neural Networks* (RNNs) são redes que possuem retroalimentação das saídas para as entradas, sendo às saídas atuais influenciadas pelas saídas anteriores. Dessa forma, considera-se que as redes recorrentes dispõe de um mecanismo de memória temporal, já que uma saída retroalimenta a próxima entrada, influenciando o cálculo da nova saída (VELLASCO, 2007). A Figura 4 apresenta um exemplo deste tipo de rede.

Figura 4. Rede Recorrente.



Fonte: LIPTON (2015)

RNNs são recomendadas especialmente para tarefas de aprendizado de máquina onde os atributos não podem ser avaliados individualmente, ou seja, onde o resultado depende de uma sequência temporal dos dados (LIPTON; BERKOWITZ, 2015).

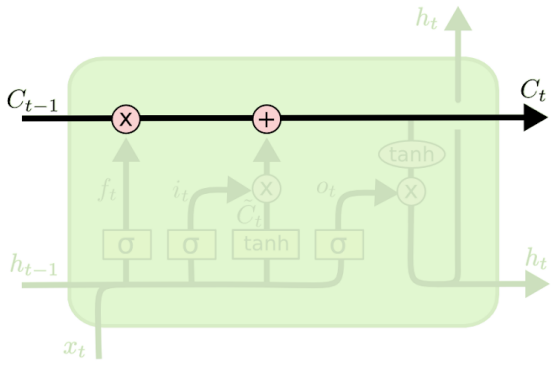
3.2 *LONG SHORT-TERM MEMORY* (LSTM)

Hochreiter e Schmidhuber (1997) introduziram o modelo *Long-Short Term Memory* (LSTM) que assemelha-se à uma rede neural recorrente simples com uma camada escondida, com a diferença de que cada neurônio nesta camada é substituído por uma célula de estado (LIPTON; BERKOWITZ, 2015). Este modelo foi criado para que, quando estão trabalhando com grandes sequências de dados, não apresente problemas comuns em arquiteturas de RNNs existentes até então denominados “fuga de gradientes” (*vanish gradient*) e “explosão de gradientes” (exploding gradientes) (HOCHREITER; SCHMIDHUBER, 1997; BENITEZ-DAVALOS et al., 2019; HOCHREITER et al., 2001). O problema da fuga e explosão de gradientes faz com que a retroalimentação das informações entre os estados cresça ou decaia exponencialmente ao longo do tempo (SUNDERMEYER; SCHLUTER; NEY, 2012).

3.2.1 Célula de Estado

A ideia principal por trás do modelo LSTM é uma célula de estado que age como uma esteira transportadora por onde a informação pode passar facilmente para toda a cadeia da rede, sem dificuldades (OLAH, 2015). A Figura 5 mostra um exemplo de uma célula de estado.

Figura 5. Célula de estado.

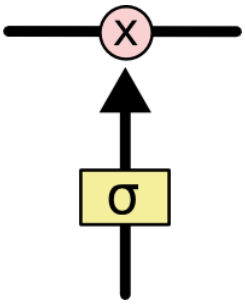


Fonte: OLAH (2015)

3.2.2 Portões

Para controlar os dados que passam pela esteira da rede, o modelo tradicional LSTM possui três estruturas denominadas “portões” (Figura 6) (OLAH, 2015). Estes portões tem a capacidade de determinar se um dado passará para a célula de estado ou se será “esquecido”. Os portões são compostos por uma função sigmóide, que resulta em valores de saída no intervalo [0...1), onde 0 significa o bloqueio total da informação e ≃1 a passagem completa da mesma (LIPTON; BERKOWITZ, 2015; OLAH, 2015), dado que sigmóide é uma função assintótica superiormente limitada em 1.

Figura 6. Portão de uma célula LSTM.

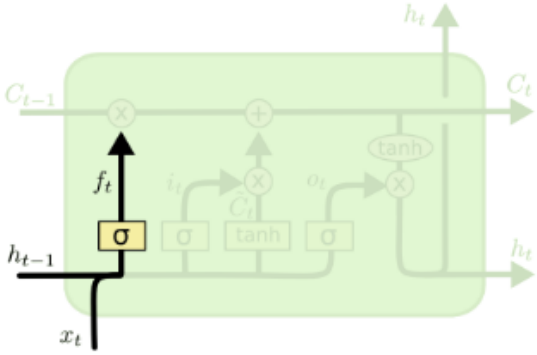


Fonte: OLAH (2015)

3.2.3 *Forget Gate Layer*

O primeiro portão de uma rede LSTM é chamado de “*forget gate layer*” (Figura 7) e tem a função de decidir quais informações serão esquecidas pela célula de estado. Esta decisão é realizada com base no resultado da função sigmóide deste portão, onde 0 significa “remover completamente a informação” e ≃1 “manter completamente a informação” (OLAH, 2015).

Figura 7. *Forget Gate Layer*.



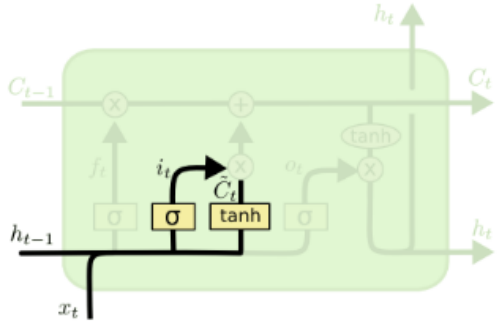
Fonte: OLAH (2015)

3.2.4 *Input Gate Layer*

O segundo portão define quais novas informações deverão ser armazenadas na célula de estado (OLAH, 2015). Este processo é composto por duas etapas: Na primeira, uma camada com uma função sigmóide chamada “*input gate layer*” (Figura 8) define quais valores serão atualizados. Na segunda etapa, uma função tanh (Figura 8) normaliza os valores no intervalo (-1...1) e, após esse processo, multiplica a saída da função sigmóide, citada anteriormente, pelo resultado normalizado. Pode-se dizer que a função sigmóide define apenas a intensidade com que a informação será adicionada à célula de estado, enquanto a função tanh define também se os valores irão permanecer na mesma “direção”. Dado que o resultado

gerado pela tanh respeita o intervalo assintótico (-1...1), além de regular a intensidade, a variação no sinal permite o ajuste na direção do volume gerado, podendo este definir ajustes positivos ou negativos. Este processo possibilita que o *input gate layer* determine o que será adicionado ou não à célula de estado (OLAH, 2015).

Figura 8. *Input Gate Layer*.

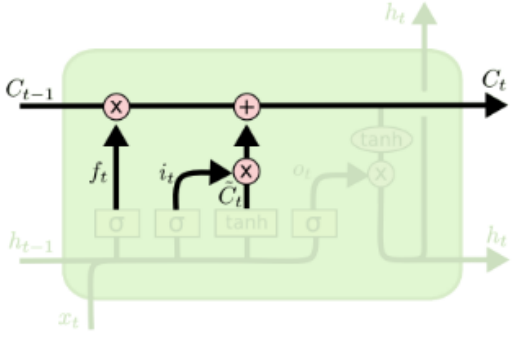


Fonte: OLAH (2015)

3.2.5 Atualização da Célula de Estado

O próximo passo é a junção do "*input gate layer"* com a função tanh para atualizar o estado da célula. Isso ocorre da seguinte maneira: multiplica-se o valor antigo da célula pelo valor do "*forget gate layer*" e então adiciona-se ao valor da multiplicação do "*input gate layer"* pela função tanh (Figura 9) (OLAH, 2015).

Figura 9. Atualização da célula de estado.

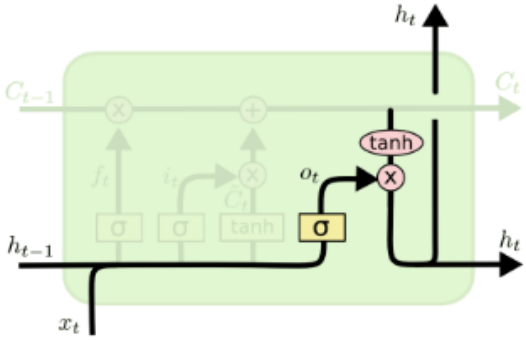


Fonte: OLAH (2015)

3.2.6 *Output Gate Layer*

O último portão, chamado de “*output gate”* (Figura 10), é responsável por quais informações serão enviadas para a saída da célula. Esta saída será baseada no estado da célula mas apresentará uma versão filtrada do mesmo, que é passado por uma função tanh, responsável por normalizar os valores no intervalo (-1...1) e então multiplicá-los pelo resultado de uma função sigmóide (OLAH, 2015).

Figura 10. *Output Gate Layer*.



Fonte: OLAH (2015)

3.3 SÉRIES TEMPORAIS

Séries temporais podem ser definidas como uma sequência de vetores com dados variando ao longo do tempo*, s*endo que os dados destes vetores podem ser qualquer variável observada. Alguns exemplos desse tipo de conjunto de dados podem ser: temperatura do ar em um edifício, quantidade de nascimentos de uma certa cidade ou a quantidade de água consumida por uma comunidade (DORFFNER, 1996).

De acordo com Lipton e Berkowitz (2015), é recomendado o uso de RNNs para realizar o processo de aprendizado de máquina em conjunto de dados dessa natureza. Gers, Eck e Schmidhuber ainda destacam que redes LSTM têm demonstrado superar RNNs tradicionais ao serem utilizadas para tarefas com séries temporais.

4 COLETA DE DADOS

Para realizar o treinamento dos modelos avaliados neste trabalho, foram necessários dados de cliques dos usuários ativos no sistema monitorado contendo as seguintes informações: identificador do usuário que realizou a ação, data e hora da interação capturada, tela em que esta interação foi realizada, elemento que sofreu a ação (botões, campos de texto, caixas de seleção, etc) e coordenadas do *mouse* no momento do clique. Como estas informações geralmente não são capturadas em *logs* de acessos a servidores ou serviços de análise de tráfego, foi necessário criar um *plugin* que capturasse os eventos de cliques em todo o sistema monitorado, armazenando-os em um banco local e então enviando os dados coletados para um repositório central na nuvem, possibilitando assim que tais dados fossem baixados e pré-processados para posteriormente servirem de exemplos ao algoritmo de treinamento dos modelos preditivos.

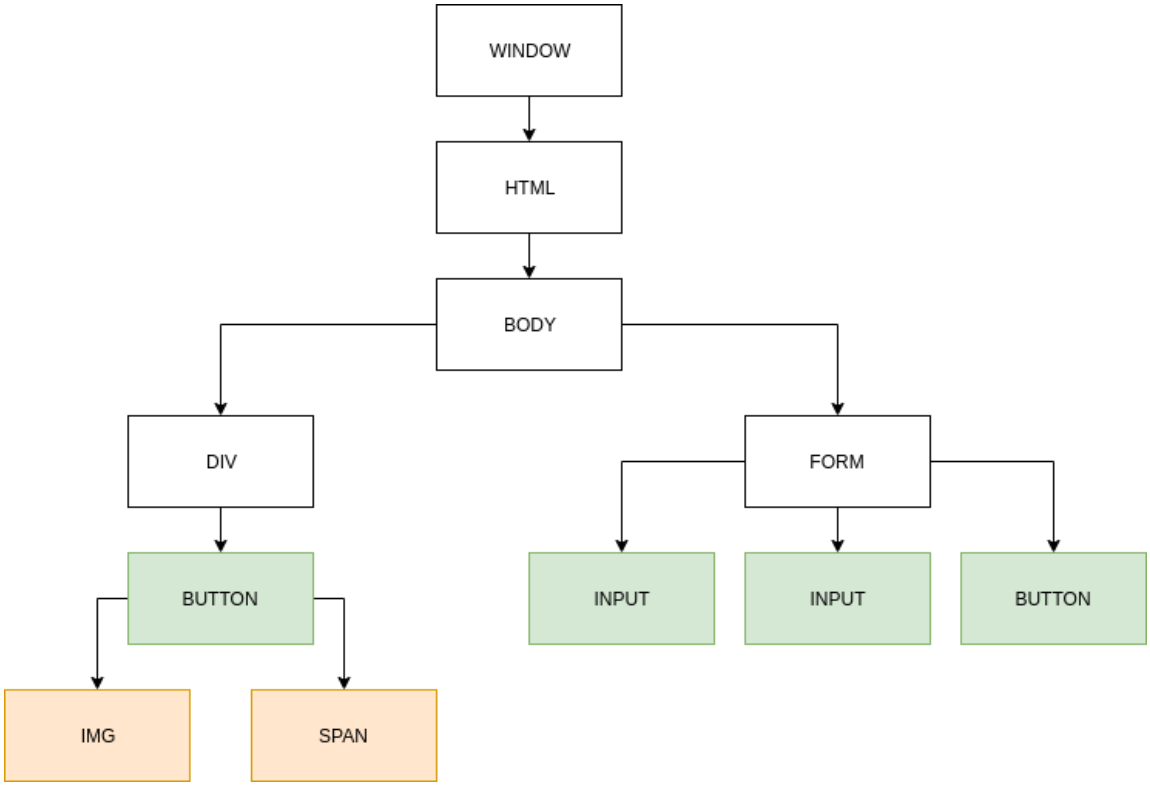
Foram pré-selecionados usuários com mais de um ano de experiência na utilização do sistema monitorado, visando maximizar a coleta de informações relevantes ao objetivo deste trabalho. Os usuários selecionados foram monitorados durante o período de outubro de 2019 até abril de 2020 e, após esse período, os registros do usuário com maior número de interações capturadas foram utilizados para composição do conjunto de dados final. Esta abordagem de seleção foi escolhida pois desejava-se criar um modelo especialista e, na falta de melhores critérios, foi utilizado o usuário com maior número de interações registradas pelo plugin de captura. Desta maneira, buscou-se aliar a necessidade de suprir um volume de dados expressivo, fundamental para treinamento dos modelos preditivos, além de contar com a experiência de um dos usuários mais ativos no sistema.

4.1 *PLUGIN* PARA COLETA DE CLIQUES

O *plugin* para coleta das interações foi desenvolvido em *javascript* e tem o papel de injetar um código no sistema monitorado. O *plugin* utiliza a função

addEventListener (W3C, 2019), definida na documentação da linguagem *javascript*, sendo padronizada nos *browsers* contemporâneos. Dessa forma, é possível capturar todos os cliques efetuados pelos usuários no sistema e, através de funções *javascript*, compilar as informações coletadas. Quando um evento é capturado, ele passa por uma rotina de pré-processamento que cria um identificador único do elemento clicado. Este identificador é criado de acordo com a hierarquia HTML da página em que foi gerado o evento capturado. Um exemplo básico de hierarquia HTML pode ser verificado na Figura 11, onde os nodos *BUTTON e INPUT* (identificados pela cor verde) são os elementos passíveis de receber interações dos usuários.

Figura 11. Hierarquia HTML.



Fonte: Elaborado pelo autor.

O processo de criação do identificador único do elemento é realizado iterativamente sobre a propriedade *path* do evento, concatenando os atributos *nodeName* (identificador do tipo de elemento clicado) e *className* (atributo auxiliar na distinção de elementos de um mesmo tipo em um mesmo nível hierárquico) em uma única *string*. Também são adicionadas informações capturadas do armazenamento local do navegador (*localStorage*) para identificação do usuário que realizou a interação.

Assim que o objeto de evento é criado, ilustrado na Figura 12, ele é adicionado em uma fila *First in First Out* (FIFO) na memória do navegador, que é consumida em intervalos pré-definidos por uma função responsável por realizar o envio dos dados ao repositório central de eventos mantido na nuvem.

Figura 12. Arquivo gerado pelo *plugin* de captura de eventos.


```
{
    "key": "2946-1580516649004",
    "user": "2946/1552",
    "date": 1580516649004,
    "location": "https://conta.saipos.com/#/app/sale/delivery/kanban/search-customer",
    "position": "1223,152",
    "resolution": "1280x832",
    "full_path": "window > #document > HTML.ng-scope > BODY.modal-open > DATA.ng-scope >
                  SECTION.ng-scope > SECTION > DATA.ng-scope > DIV.container.ng-scope >
                  DATA.ng-scope > DIV.main-kanban.ng-scope >
                  PARTNER-SALE-RESOLVE-ISSUES.ng-isolate-scope >
                  DIV.card.partner-sale-resolve-issues.ng-scope > DIV.card-body >
                  TABLE.table.table-hover > TBODY > TR.ng-scope > TD.text-right >
                  BUTTON.btn.btn-primary.waves-effect",
    "content": "IMPORTAR"
}
```


Fonte: Elaborado pelo autor.

O processo de criação e envio dos objetos de eventos é completamente isolado do sistema monitorado. Caso sejam identificados erros, a rotina de envio assume comportamento de precaução e entra em estado de “pausa”, prevenindo assim uma má experiência durante a utilização do sistema monitorado. Ao final do processo, cada evento enviado gera um arquivo no formato *JSON* contendo as informações descritas na Tabela 1.

Tabela 1. Descrição dos atributos do arquivo gerado pelo plugin.

Atributo	Descrição
key	Formada pelo ID do usuário e um *timestamp* do momento da ação que foi capturada.
user	Identificador único do usuário que realizou a ação.
date	*Timestamp* do momento em que a ação foi capturada.
location	Tela em que o evento foi capturado.
position	Posição em que o ponteiro do mouse estava no momento do clique.
resolution	Resolução da tela em que o evento foi capturado.
full_path	*NodeName* e *className* de todos os elementos na hierarquia HTML.
content	Conteúdo interno do elemento clicado.

Fonte: Elaborado pelo autor.

4.2 PRÉ-PROCESSAMENTO

Como o plugin de captura registra todos os cliques executados pelos usuários, o primeiro passo do pré-processamento foi remover os registros em elementos que não representam efetivamente uma ação, deixando apenas cliques que foram executados em elementos HTML dos tipos “*a”, “button”, “input”, “select”* e *“textarea”.* Este processo foi necessário já que identificou-se a existência de diversos eventos capturados por “vícios de utilização” dos usuários, como cliques na tela para dar foco no navegador, ou até cliques imprecisos.

O próximo passo foi normalizar as *URLs* e elementos*,* pois em alguns casos existiam parâmetros identificadores de registros. Em uma edição de venda, por exemplo, a *URL* é utilizada para passar o atributo de identificador desta venda na base de dados. Como para o treinamento dos modelos deste trabalho não julgou-se necessária a identificação dos registros (vendas, produtos, clientes, etc), estes

trechos das *URLs* e elementos foram removidos e considerados apenas os identificadores de telas.

Em seguida foi feito um pré-processamento dos identificadores dos elementos clicados, quando foram removidas informações de nodos filhos dos elementos dos tipos “*a”, “button”, “input”, “select”* e *“textarea”*. Esta remoção acontece já que um elemento *“button”*, por exemplo, pode conter outros elementos visuais internos (ícones, blocos de texto, imagens, etc) que não são importantes para identificar o clique. Um exemplo desse tipo de nodo pode ser observado na Figura 11, onde os nodos destacados em verde apresentam elementos que devem capturar eventos, enquanto seus nodos filhos são apenas elementos visuais auxiliares. Para executar este processo foi necessária a utilização de expressões regulares, pois desejava-se localizar e substituir trechos em uma *string* onde não se conhecia o conteúdo exato, mas sim o padrão que a formava (MDN, 2019).

O próximo passo foi a transformação dos atributos em identificadores numéricos únicos, sendo que, para cada atributo diferente na lista de eventos foi gerado um identificador decimal correspondente. Após o processo de transformação dos atributos, os dados foram ordenados de acordo com a data e hora do evento para então serem gravados em um único arquivo.

O problema de aprendizado de máquina supervisionado requer dados rotulados para que seja possível realizar o treino dos modelos. Neste caso foram usados o identificadores numéricos únicos para representar as classes.

Verificou-se que a quantidade de identificadores únicos é proporcional ao volume de dados processados, ou seja, quanto mais eventos são capturados, mais identificadores únicos são criados. Tal comportamento foi percebido já que as atualizações no sistema monitorado alteram a estrutura hierárquica da página, alterando também o resultado do cálculo para geração do identificador dos elementos. Este problema possivelmente impactou os resultados da análise experimental.

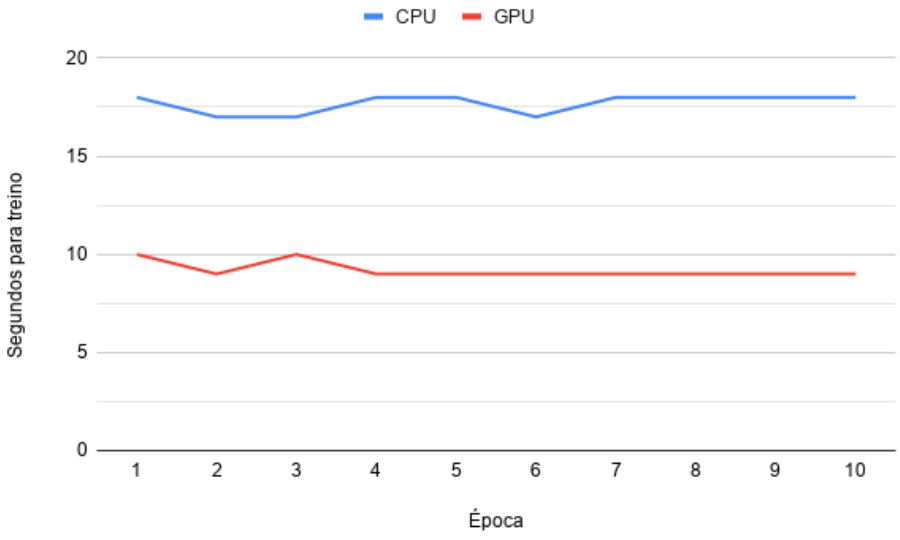
5 TREINAMENTO DOS MODELOS

Para realizar o treinamento dos modelos foi utilizada a linguagem de programação Python com framework Keras executado sobre *Graphics Processing Unit* (GPU) em um container Docker com ambiente Linux. A linguagem Python é amplamente utilizada para computação científica e numérica (PYTHON, 2020), adequada para a realização deste trabalho.

Para criação dos algoritmos de treinamento, validação e predição, foi utilizado o Keras, uma API de alto nível para redes neurais, capaz de rodar sobre o *framework* TensorFlow (KERAS, 2020). o TensorFlow é uma plataforma de aprendizado de máquina de código aberto utilizada para o desenvolvimento de aplicações científicas e corporativas (TENSORFLOW, 2020).

Visando um melhor desempenho na rotina de treinamento dos modelos, o *framework* foi configurado para executar sobre GPU, favorecendo computações paralelas. De acordo com os testes realizados observou-se que, sobre GPU obtém-se até 40% de ganho em performance com relação ao tempo de execução quando comparado ao mesmo algoritmo executado em CPU.

Figura 13. Comparação entre CPU e GPU no treinamento dos modelos.



Fonte: Elaborado pelo autor.

Pode-se verificar no gráfico ilustrado pela Figura 13 que o tempo necessário para treinamento dos modelos em GPU se mantém em torno de 40% inferior ao tempo necessário para treinamento em CPU. Nesta comparação foram observados os tempos de treinamento por época de um modelo utilizando um conjunto de dados com 5.000 registros, porém espera-se que o mesmo comportamento ocorra em conjuntos maiores. Esta comparação tomou por base um treinamento de 10 épocas.

5.1 CONFIGURAÇÃO DO COMPUTADOR DE TREINAMENTO

O computador utilizado no processo de treinamento, validação e testes dos modelos utilizados nos experimentos promovidos por este trabalho tem a seguinte configuração:

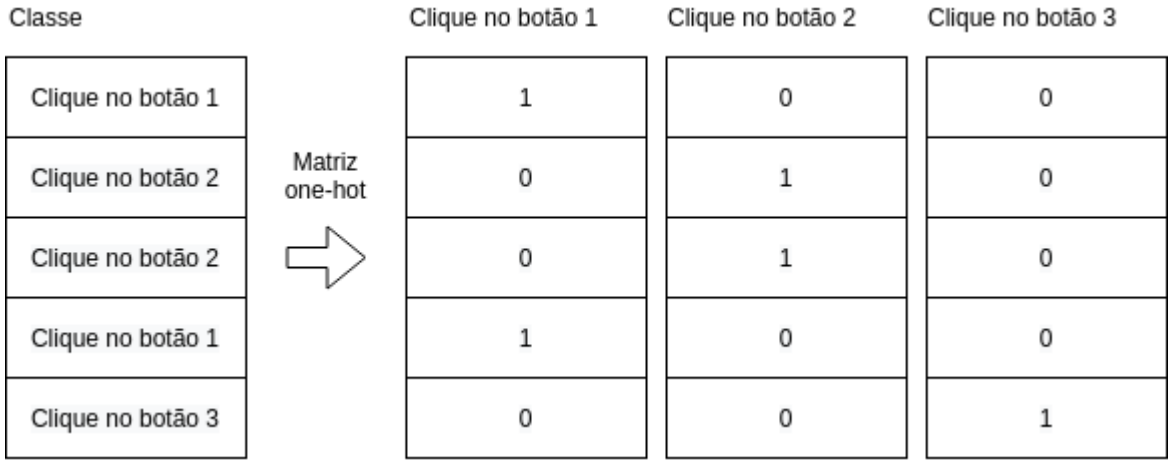
- CPU: Intel Core i5 - 9300H - 2.4GHz
- Memória RAM: 16 GB DDR4 2400MHz
- GPU: NVIDIA GeForce GTX 1050 - 3GB
- Driver NVIDIA: NVIDIA-SMI 440.82 (CUDA 10.2)
- Armazenamento: SSD Kingston M.2 NVMe A2000
- Sistema Operacional: Ubuntu 18.04.3 LTS

5.2 ALGORITMO

O primeiro passo do algoritmo desenvolvido é carregar o arquivo CSV gerado pelo pré-processamento e então dividi-lo em três subconjuntos diferentes. O primeiro subconjunto corresponde a 70% dos registros e é usado para treinamento do modelo preditivo. O segundo possui 20% do número total de eventos e é utilizado para realizar a validação de cada época do treino. Os últimos 10% são dados utilizados para testar o modelo e determinar sua acurácia. Os subconjuntos passam então por um processo de transformação dos valores das classes para uma matriz *one-hot*. O processo de transformação das classes para a matriz *one-hot* pode ser

exemplificado de acordo com a Figura 14, onde cada classe é transformada em uma coluna na matriz e as linhas desta matriz correspondem aos eventos da janela temporal.

Figura 14. Exemplo de transformação de uma janela de 5 eventos (para um dado problema de 3 classes) em uma matriz one-hot.



Fonte: Elaborado pelo autor.

Após a divisão dos dados nos subconjuntos, é realizada a criação dos geradores de dados responsáveis por fornecer as informações para as etapas de treino, validação e testes do modelo. Estes geradores seguem o conceito de séries temporais, onde o tamanho da janela de tempo teve seus valores experimentados para determinar qual valor apresenta melhor resultado para o problema estudado.

O próximo passo do algoritmo é criar as camadas do modelo preditivo. Para isso é utilizado um modelo sequencial do framework Keras com três camadas descritas abaixo:

1. LSTM: Onde a quantidade de células foi variada no decorrer do capítulo de análise experimental.
2. *Activation*: Camada de ativação da rede utilizando uma função *tanh*.
3. *Dense*: Utilizando a função *softmax* para ativação.

Além das camadas, o modelo possui duas funções que são executadas ao final de cada época. A primeira função, chamada de *Early stopping*, é responsável pelo encerramento da execução do treino ao detectar que não houve melhora nos resultados da validação após um certo período, sendo este período determinado pelo parâmetro de "*paciência*" da rede. A segunda função é responsável por armazenar o modelo com melhor desempenho após cada etapa de validação para que este modelo seja então utilizado na rotina de testes.

A etapa de testes acontece iterando-se os registros do subconjunto separado para este fim e então cruzando o resultado da predição executado pelo modelo treinado com o resultado real dos dados capturados.

6 ANÁLISE EXPERIMENTAL

Para a análise experimental do trabalho foram realizados testes com diferentes configurações da rede neural e diferentes tamanhos de conjunto de dados, sendo estes compostos por 1.000, 5.000, 10.000, 50.000 e 100.000 registros, todos extraídos da mesma base de dados e com os mesmos parâmetros de criação dos identificadores únicos. A Tabela 2 apresenta a quantidade de classes, que são determinadas de acordo com os identificadores únicos gerados a partir dos evento coletados, para cada conjunto de dados utilizado nos experimentos.

Tabela 2. Quantidade de classes em cada conjunto de dados.

Tamanho do conjunto	Número de classes
1.000	53
5.000	81
10.000	99
50.000	141
100.000	157

Fonte: Elaborado pelo autor.

Os parâmetros experimentados foram: Quantidade de células LSTM, Tamanho da janela da série temporal e Paciência do Early stopping. As informações apresentadas após a execução de cada experimento foram: Quantidade total de épocas na execução do treino, tempo necessário para treino do modelo e acurácia dos testes.

6.1 QUANTIDADE DE CÉLULAS LSTM

Esta seção apresenta os resultados de cada tamanho de conjunto de dados, tempo para treinamento do modelo e acurácia atingida nos testes sendo que, para

cada configuração de quantidade de células LSTM foi gerado um novo gráfico de resultados. Os valores de tamanho na janela da série temporal e paciência do *early stopping* não foram alterados no decorrer dos testes sendo utilizada a seguinte configuração:

- Tamanho do conjunto de dados: 1.000, 5.000, 10.000, 50.000 e 100.000 registros;
- Quantidade de células LSTM: 1, 8, 16 e 32 células;
- Tamanho da janela da série temporal: 5 registros;
- Paciência do *Early stopping*: 5 épocas.

A Tabela 3 apresenta de forma resumida os resultados obtidos após os testes com as configurações propostas acima. É possível verificar que houve uma melhora da acurácia quando o número de células LSTM passou de 1 para 8 e que o melhor percentual foi obtido com o conjunto de dados com 5.000 registros, chegando a 57,98%.

Tabela 3. Análise experimental com variação na quantidade de células LSTM.

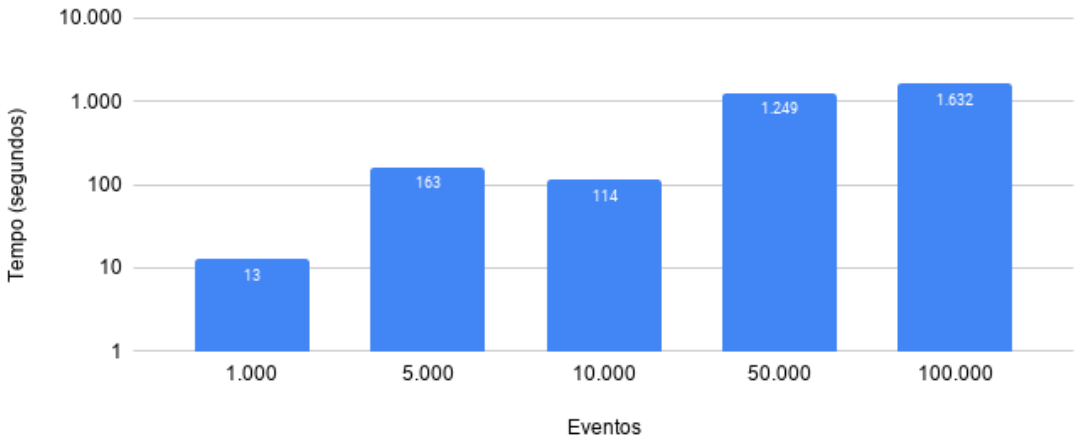
Eventos	1 célula LSTM			8 células LSTM			16 células LSTM			32 células LSTM		
	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)
1.000	6	13	17,71	11	23	30,21	14	31	43,4	11	21	42,71
5.000	18	163	23,1	21	187	**57,98**	13	119	57,71	12	106	57,71
10.000	6	114	18,16	6	117	39,77	6	106	41,87	6	113	51,79
50.000	14	1.249	22,37	18	1.612	52,45	14	1.331	52,97	12	1.038	54,09
100.000	9	1.632	22,57	9	1.518	48,44	15	2.865	49,57	11	1.997	50,41

Fonte: Elaborado pelo autor.

As Figuras 14 e 15 apresentam o resultado com a rede configurada com 1 célula LSTM e tamanho do conjunto de dados variando entre 1.000 e 100.000

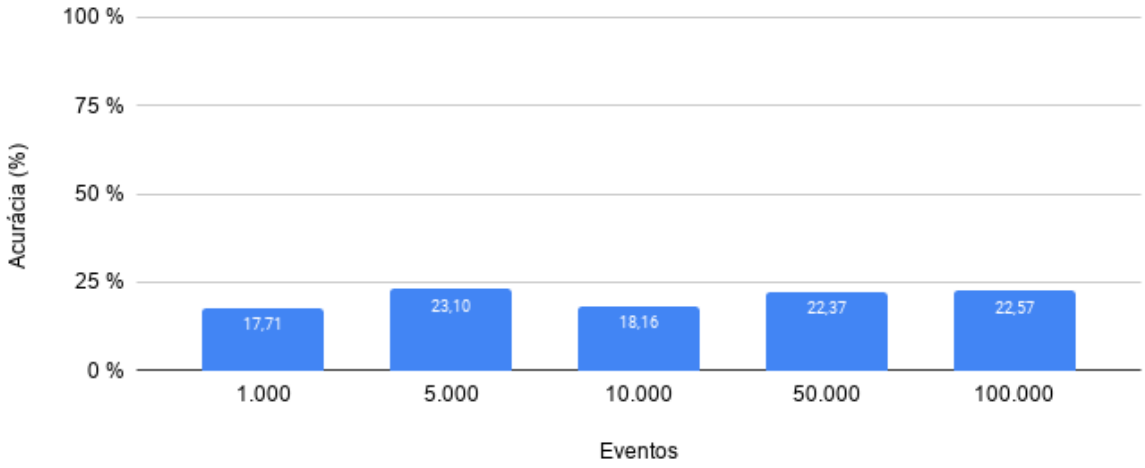
eventos. Pode-se verificar na Figura 15 o aumento no tempo necessário para treinamento para os conjuntos maiores sem que a acurácia, ilustrada pela Figura 16, acompanhe essa tendência. Também é possível verificar que houve um aumento significativo no tempo necessário para treino quando o conjunto de dados passou de 10.000 para 50.000 registros.

Figura 15. Tempo para treino com 1 célula LSTM



Fonte: Elaborado pelo autor.

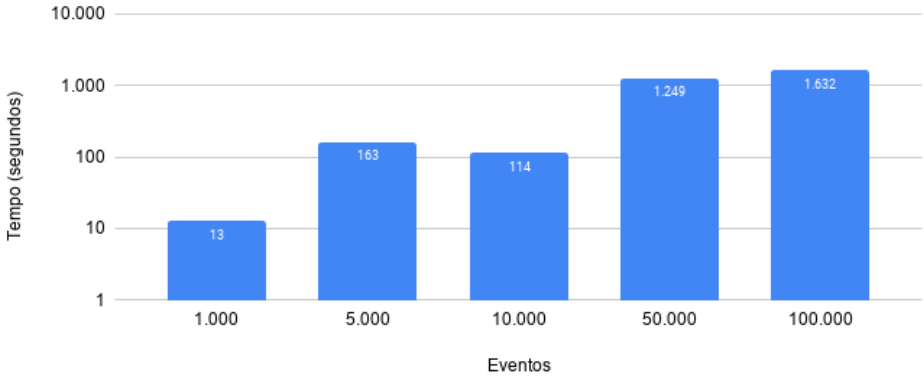
Figura 16. Acurácia do modelo com 1 célula LSTM



Fonte: Elaborado pelo autor.

Ao aumentar a quantidade de células para 8, ilustrado pelas Figuras 16 e 17, nota-se uma melhora significativa na acurácia da rede em comparação ao resultado com apenas 1 célula verificado na Figura 16. Observou-se o mesmo comportamento de aumento considerado do tempo necessário para treinamento nos conjuntos de dados de 50.000 e 100.000 registros.

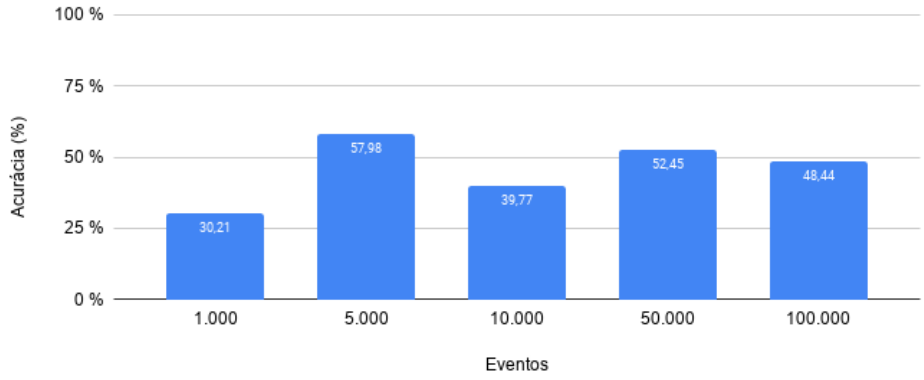
Figura 17. Tempo para treino com 8 célula LSTM



Fonte: Elaborado pelo autor.

Percebe-se também que a acurácia desta configuração, apresentada na Figura 18, não tende a melhorar com conjuntos de dados maiores em relação aos conjuntos com menos registros. De acordo com o gráfico (Figura 18) é possível verificar que o conjunto de dados com melhor acurácia foi o com 5.000 registros.

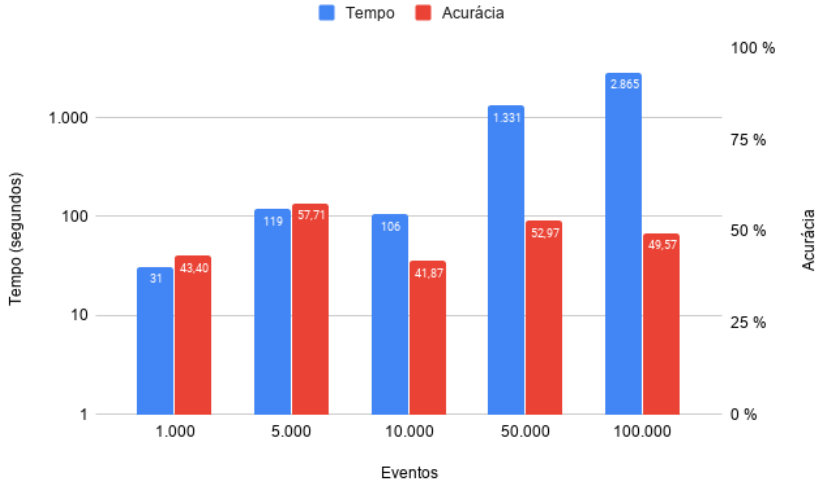
Figura 18. Acurácia do modelo com 8 células LSTM



Fonte: Elaborado pelo autor.

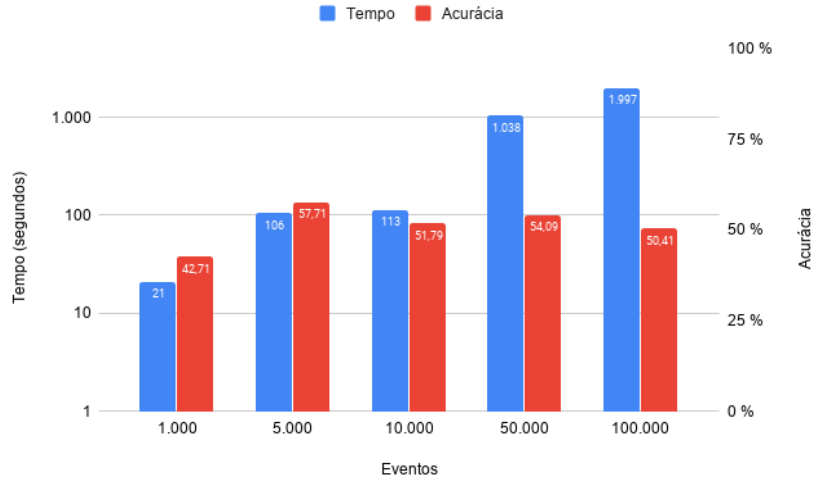
As Figuras 18 e 19 apresentam o mesmo teste com quantidades de células de 16 e 32, respectivamente. Percebeu-se que nesse cenário não houve melhora significativa na acurácia em relação à configuração com 8 células. É possível verificar que o conjunto de dados com 5.000 registros segue apresentando a melhor acurácia entre todos os testes, e que o tempo necessário para treino do modelo seguiu aumentando com os conjuntos de dados com mais registros.

Figura 19. Análise experimental com 16 células LSTM



Fonte: Elaborado pelo autor.

Figura 20. Análise experimental com 32 células LSTM



Fonte: Elaborado pelo autor.

De forma geral, pode-se verificar que o tempo necessário para treino dos modelos aumenta proporcionalmente ao tamanho do conjunto de dados, mesmo que a acurácia não siga esta tendência. Houve também uma melhora na acurácia dos testes quando a rede foi configurada com 8 células LSTM em relação à apenas 1 célula, onde foi atingida a acurácia de 57,98% com o conjunto de 5.000 registros. Verificou-se que, ao aumentar o número de células para 16 ou 32, não houve melhora significativa na acurácia do modelo. Desta maneira, os próximos experimentos empregam a configuração com 8 células. Também são descartados os conjuntos com 50.000 e 100.000 eventos, pois não houve uma melhora na acurácia proporcional ao tempo necessário para treino dos modelos.

6.2 PACIÊNCIA DO *EARLY STOPPING*

O teste realizado a seguir foi a variação do parâmetro de paciência do *early stopping* do treino da rede. Como nos resultados da seção 6.1 a melhor acurácia foi obtida com 8 células LSTM, este valor de configuração foi selecionado para dar sequência nos testes, que foram realizados com as seguintes configurações:

- Tamanho do conjunto de dados: 1.000, 5.000 e 10.000 registros;
- Quantidade de células LSTM: 8 células;
- Tamanho da janela da série temporal: 5 registros;
- Paciência do *Early stopping*: 5, 10, 20 e 50 épocas.

Na Tabela 4 é possível verificar os resultados dos testes realizados com as configurações citadas acima. Percebe-se que o conjunto de dados com maior acurácia continua sendo o de 5.000 registros e que mesmo com um maior número de épocas, não houve melhora no percentual de acertos nos testes.

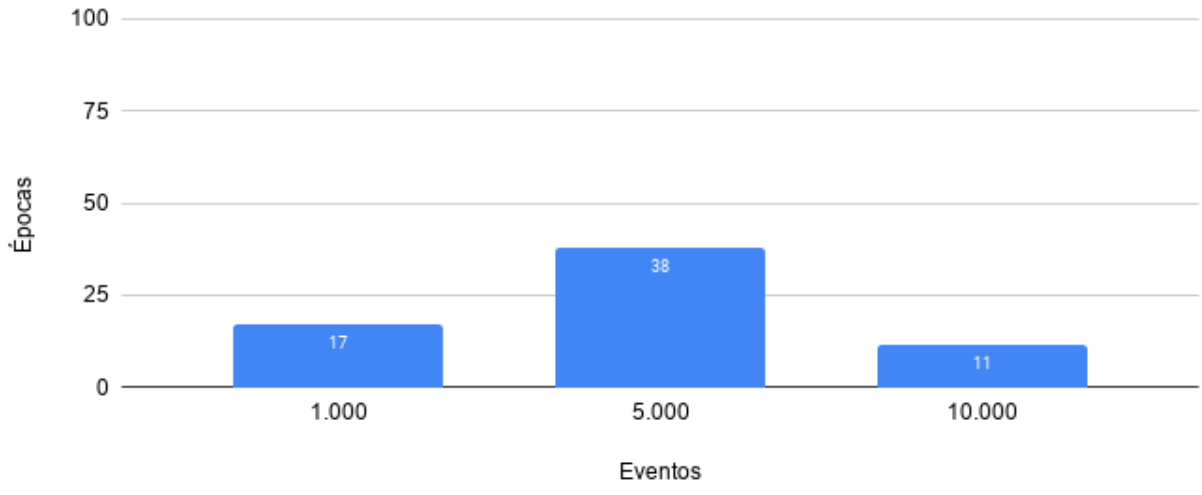
Tabela 4. Análise experimental com variação no parâmetro de *early stopping.*

Eventos	Early Stopping = 5			Early Stopping = 10			Early Stopping = 20			Early Stopping = 50		
	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)	Épocas	Tempo (segundos)	Acurácia (%)
1.000	11	23	30,21	17	31	24,54	31	57	36,35	61	112	35
5.000	21	187	57,98	38	394	58,26	70	734	57,44	111	1.046	58,44
10.000	6	117	39,77	11	242	38,07	21	405	39,32	51	1.036	38,02

Fonte: Elaborado pelo autor.

A Figura 21 apresenta a quantidade de épocas utilizadas para treinamento do modelo com a configuração de 10 épocas para paciência do *early stopping*. Pode-se verificar que o conjunto de dados com 5.000 registros executou o treinamento por 38 épocas contra 17 do conjunto com 1.000 registros, seguido de 11 épocas no conjunto com 10.000 registros.

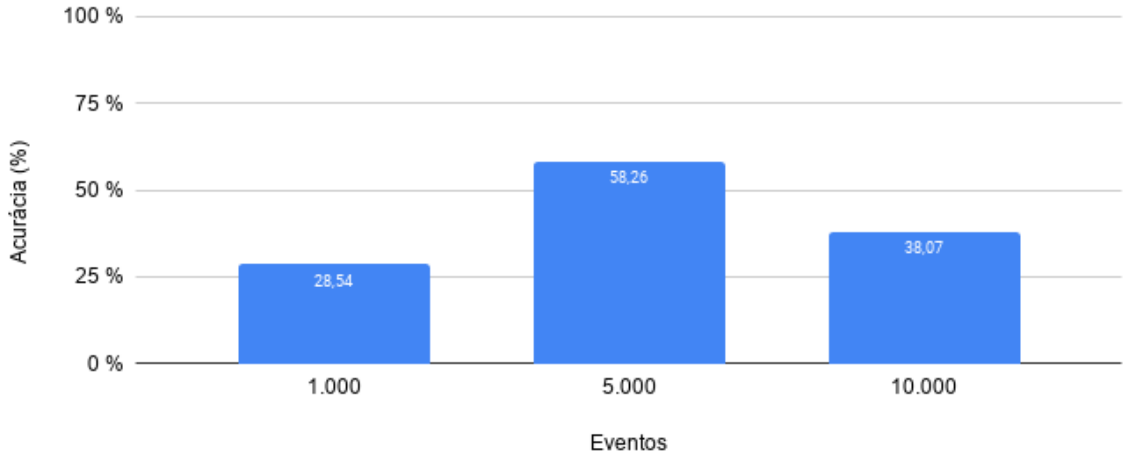
Figura 21. Quantidade de épocas utilizadas para treinamento do modelo com a configuração de 10 épocas para paciência do early stopping



Fonte: Elaborado pelo autor.

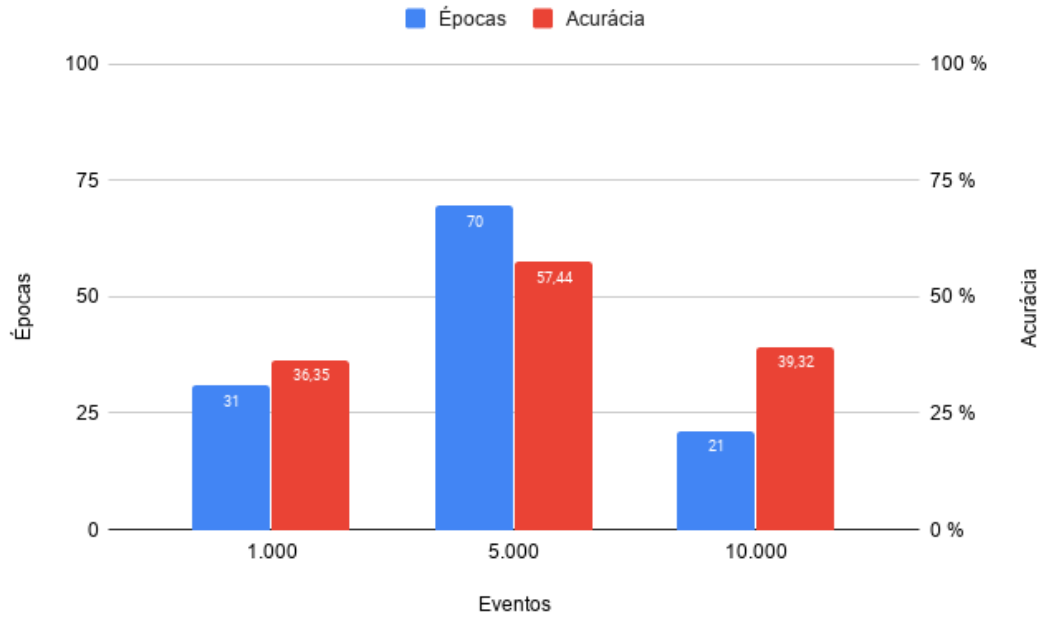
É possível verificar na Figura 22 que a melhor acurácia com esta configuração foi atingida novamente com o conjunto de dados de 5.000 registros, atingindo 58,26%. A acurácia dos demais conjuntos de dados ficaram em 28,54% e 38,07 nos conjuntos de 1.000 e 10.000 registros respectivamente.

Figura 22. Acurácia do modelo com a configuração de 10 épocas para paciência do early stopping



Fonte: Elaborado pelo autor.

Figura 23. Análise experimental com 20 épocas de *early stopping*

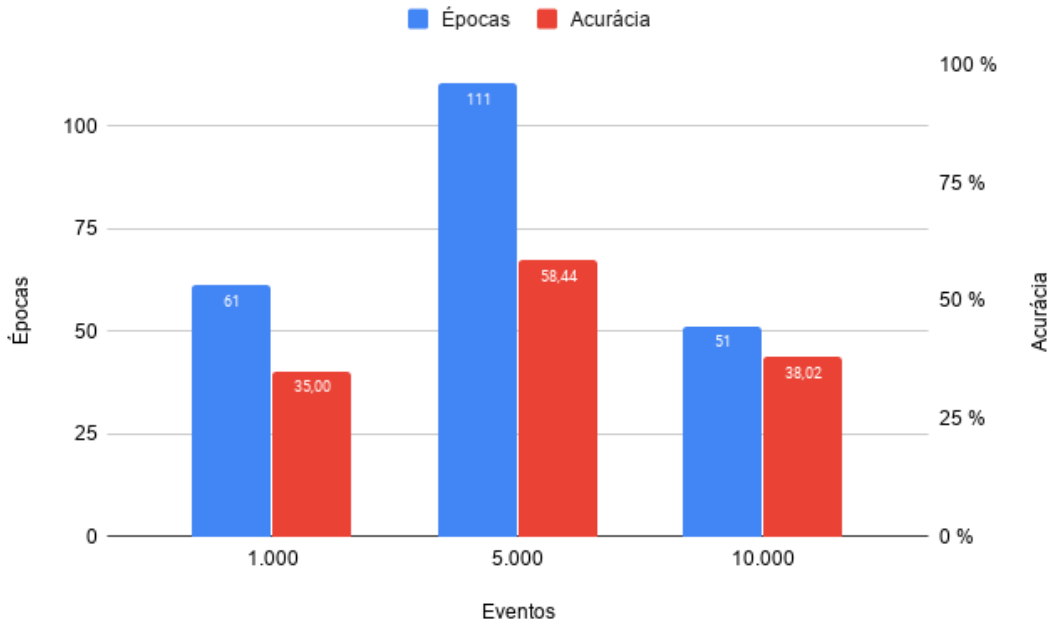


Fonte: Elaborado pelo autor.

Na Figura 23 é apresentado o gráfico com os resultados dos testes com 20 épocas de paciência do *early stopping*. Neste cenário houve um aumento esperado no número de épocas executadas no treino, porém não foi observada melhora na acurácia dos testes com exceção do conjunto com 1.000 registros, onde é possível verificar uma melhora nos resultados, porém ainda, muito abaixo da acurácia alcançada pelo conjunto com 5.000 registros.

Ao aumentar o parâmetro de paciência do *early stopping* para 50 épocas, como apresentado na Figura 24, não percebeu-se melhora na acurácia apesar do aumento na quantidade de épocas realizadas no treino do modelo. Verificou-se que as acurácias apresentadas ficaram muito próximas às obtidas nos testes com menos épocas de paciência do *early stopping*, registrando o melhor resultado no conjunto de dados com 5.000 registros com 58,44%.

Figura 24. Análise experimental com 50 épocas de *early stopping*



Fonte: Elaborado pelo autor.

Foi observado nos testes da seção 6.1 que o conjunto de dados com melhor desempenho na acurácia foi o de 5.000 registros. Não foi percebida melhora

significativa na acurácia ao aumentar a quantidade de épocas do parâmetro de paciência do *early stopping*. Sendo assim, para os próximos testes será utilizado o conjunto de dados de 5.000 registros com 8 células LSTM e 5 épocas de paciência do *early stopping.*

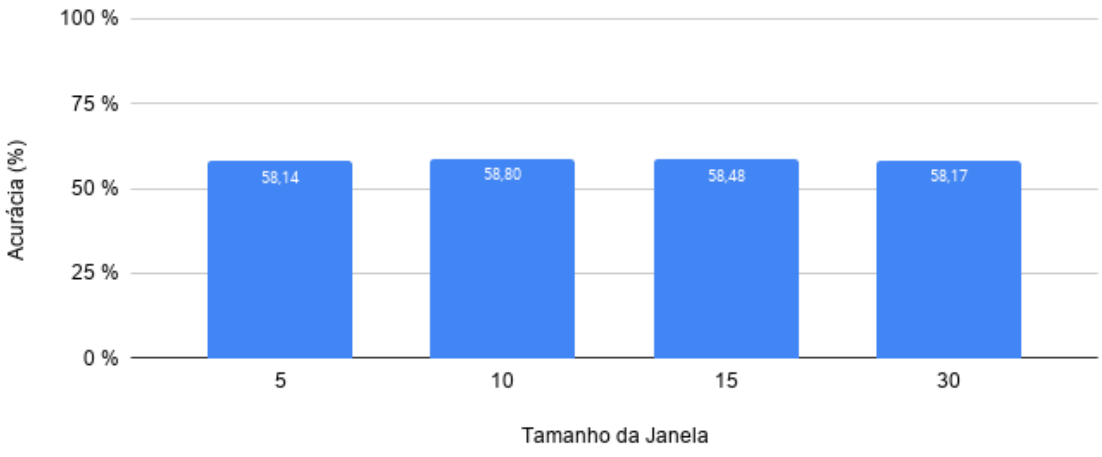
6.3 TAMANHO DA JANELA TEMPORAL

O próximo experimento variou o tamanho da janela temporal do conjunto de dados. Foram mantidos os demais parâmetros com melhor desempenho nos testes anteriores sendo utilizada a seguinte configuração:

- Tamanho do conjunto de dados: 5.000 registros;
- Quantidade de células LSTM: 8 células;
- Tamanho da janela da série temporal: 5, 10, 15 e 30 registros;
- Paciência do *Early stopping*: 5 épocas.

Na Figura 25 é possível verificar que a acurácia dos testes não teve melhora com diferentes tamanhos de janela, mantendo-se entre 58,14% e 58,80%.

Figura 25. Análise experimental com diferentes janelas temporais

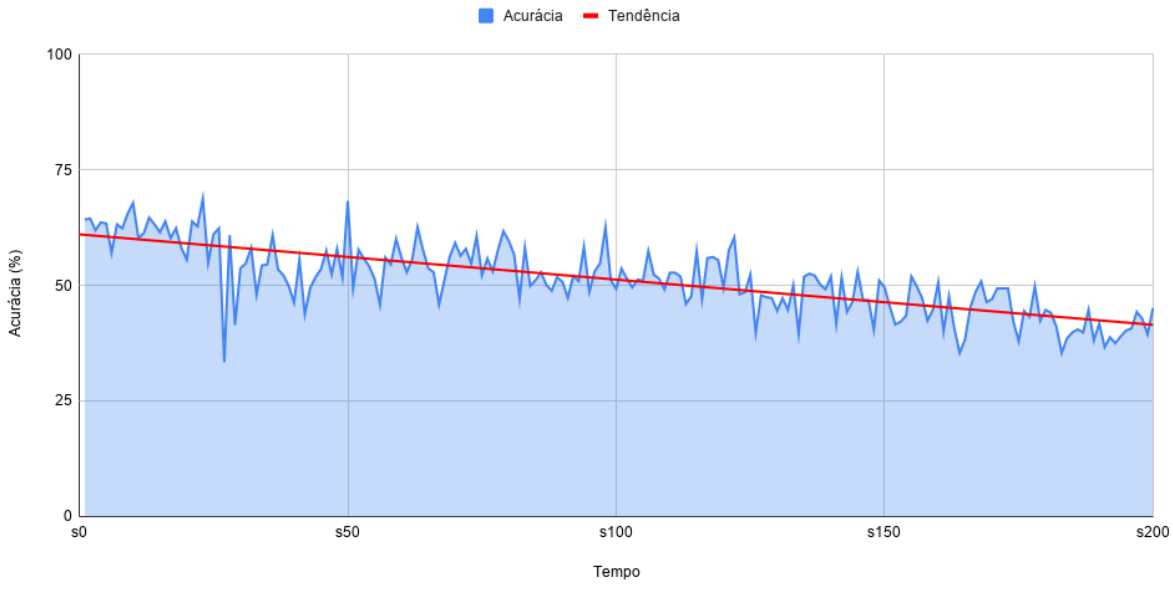


Fonte: Elaborado pelo autor.

6.4 TESTE DE ACURÁCIA AO LONGO DO TEMPO

O experimento a seguir foi realizado para verificar a acurácia do modelo preditivo ao longo do tempo, avaliando 200 subconjuntos de dados posteriores ao subconjunto utilizado para treinamento e validação da rede. Foi utilizado um conjunto de dados com um total de 100.000 registros, sendo que 3.500 foram aplicados no treinamento do modelo e 1.000 na validação do mesmo. Os outros 95.500 registros foram separados em 200 subconjuntos sequenciais ao longo do tempo. A Figura 26 apresenta os resultados dos testes realizados, onde é possível verificar que houve uma tendência de queda da acurácia ao longo do tempo. O eixo horizontal do gráfico apresenta cada subconjunto testado, variando entre *s0* à *s200*, onde s0 representa o conjunto com dados coletados imediatamente após os dados utilizados para treino e validação, s1 representa os dados posteriores a s0 e assim sucessivamente até chegar ao s200, que representa os dados mais distantes do conjunto utilizado para treino.

Figura 26. Teste de acurácia ao longo do tempo



Fonte: Elaborado pelo autor.

Pode-se verificar no gráfico da Figura 26 que o primeiro quadrante apresenta uma média de acurácia de 57,66%, seguido de 54,09%, 49,90% e 43,21% de média nos quadrantes subsequentes. Com essa informação é possível determinar um prazo para que o modelo tenha que passar por uma nova fase de treinamento, mantendo assim a acurácia próxima à um valor aceitável.

6.5 APLICABILIDADE

Com base nos experimentos realizados, é possível determinar a melhor configuração dos parâmetros de uma rede neural LSTM para o cenário estudado, no caso 8 células LSTM com 5 épocas de paciência de *early stopping* e janela temporal composta por 5 itens em um conjunto de 5.000 registros e 81 classes. A partir deste achado seria possível realizar o desenvolvimento de um serviço que realize o treinamento de modelos preditivos com base nos dados coletados pelo plugin de captura e então, baseado nos melhores resultados das predições, faça um destaque visual do elemento predito pela rede neural, auxiliando automaticamente os usuários do sistema.

Também é possível determinar que a rede neural deveria passar por um processo de treinamento a cada 23.875 registros capturados, de acordo com avaliação temporal que pode ser observada na seção 6.4 deste trabalho, visando manter a acurácia da rede sempre em seu melhor desempenho. Esta sugestão é realizada com base na avaliação do gráfico ilustrado pela Figura 26, onde é possível verificar que no primeiro quadrante a média de acurácia é de 57,66%, muito próxima ao melhor resultado obtido por este trabalho, enquanto que, no segundo quadrante, esta média cai para 54,09%.

7 CONCLUSÃO

Em decorrência da grande taxa de crescimento do modelo de entrega de *softwares* denominado SaaS, um fator crucial para competitividade das empresas que dependem deste tipo de licenciamento é a facilidade com que seus usuários estarão adaptados ao sistema em suas primeiras utilizações, gerando um menor custo operacional para a empresa e maior satisfação do cliente ao contratar o serviço. Sistemas de *User Onboarding* são ferramentas que buscam amenizar a curva de aprendizado de novos usuários. Por outro lado, estes sistemas são fundamentalmente dependentes de uma mão de obra técnica para serem configurados. Dado este cenário, foi criado um modelo preditivo com base em dados coletados dos usuários já ativos no sistema monitorado. O referido módulo pode predizer ações que um usuário irá realizar em uma determinada tela do sistema.

Para possibilitar a criação do modelo preditivo foi desenvolvido um *plugin* que realizou a captura das interações dos usuários já ativos no sistema monitorado, resultando em um conjunto de dados com um total de 100.000 registros. Ao todo, as ações capturadas são compostas por 157 diferentes ações que podem ser executadas pelo usuário no sistema. Posteriormente, estes dados foram divididos em 4 outros conjuntos de 1.000, 5.000, 10.000 e 50.000 registros, para que fossem comparados os resultados dos experimentos com diferentes tamanhos de conjuntos de dados, além do conjunto original com 100.000 registros.

A partir da construção dos conjuntos de dados de treinamento, foram realizados experimentos com diferentes configurações de redes neurais onde registrou-se o tempo necessário para treinamento dos modelos, bem como a acurácia de teste atingida pelos mesmos. Os experimentos foram realizados variando os parâmetros de “Quantidade de células LSTM”, “Tamanho da janela da série temporal” e “Paciência do *Early stopping*”.

O primeiro experimento foi realizado variando a quantidade de células LSTM da arquitetura de rede neural entre 1, 8, 16 e 32 células. Neste cenário, o melhor resultado observado foi obtido com a configuração de 8 células LSTM, atingindo

57,98% de acurácia com o conjunto de 5.000 registros, onde estão presentes 81 diferentes classes. Neste experimento, também foi verificado um aumento no tempo necessário para treinamento dos conjuntos com 50.000 e 100.000 registros. Sendo assim, ambos os conjuntos foram descartados para os próximos experimentos.

Na segunda etapa de experimentos, foram comparadas variações do parâmetro de paciência do *early stopping*, assumindo os valores 5, 10, 20 e 50 para número de épocas. Verificou-se novamente que o conjunto com 5.000 registros obteve uma melhor acurácia, atingindo 58,26% com a configuração de 10 épocas de paciência do *early stopping*. Este mesmo conjunto obteve uma acurácia de 57,98% com o parâmetro setado para 5 épocas, porém com um tempo 52% menor em comparação com 10 épocas (187 segundos contra 394 segundos, respectivamente).

Em uma etapa adicional, foram realizados experimentos variando o tamanho da janela temporal dos conjuntos de dados, comparando os resultados com 5, 10, 15 e 30 registros em cada janela. Neste cenário, não foi verificada melhora na acurácia, que manteve-se entre 58,14% e 58,80%.

Com base nos resultados dos experimentos realizados, foi possível determinar que a melhor configuração da rede neural para o cenário estudado contém 8 células LSTM, 5 épocas de paciência de *early stopping* e janela temporal com 5 itens em um conjunto de 5.000 registros. Em seu melhor resultado, esta configuração atingiu uma acurácia de 58,14% na predição de um conjunto com 81 classes possíveis.

O presente trabalho buscou criar um modelo capaz de predizer ações de um usuário em um sistema baseado em interfaces gráficas. Assumindo que, para um problema de 81 classes, a probabilidade arbitrária de acerto é de 1,23%, dado que o melhor modelo treinado e avaliado pelo trabalho atinge uma acurácia de 58,14%, conclui-se que os resultados obtidos são satisfatórios, podendo assim contribuir em aplicações práticas que demandem a predição de ações dos usuários. A partir desta observação, estima-se ser possível a redução de custos com *User Onboarding* através da construção de assistentes automáticos que possam melhorar seu desempenho com base nos dados resultantes das predições. Estes assistentes

podem fazer o uso dos modelos preditivos para destacar elementos visuais na tela do sistema, auxiliando assim novos usuários de forma automática.

Sugere-se uma continuidade dos estudos com o intuito de aplicar o modelo preditivo em uma interface do sistema, verificando assim sua eficácia em auxiliar novos usuários. Melhorias podem ser implementadas na ferramenta de captura a fim de reduzir a quantidade de classes ou melhorar a classificação dos elementos dos conjuntos de dados, bem como podem ser testados outros parâmetros da rede neural, visando uma melhor acurácia das predições.

REFERÊNCIAS BIBLIOGRÁFICAS

ALPAYDIN, Ethem. **Introduction to Machine Learning**. 2. ed. Londres: The MIT Press, 2010. 537 p. ISBN 978-0-262-01243-0.

APPCUES. **Appcues**. 2019. Disponível em: <https://www.appcues.com/>. Acesso em: 29 ago. 2019.

BARRETO, Jorge M. **Introdução às Redes Neurais Artificiais**. 2002. Disponível em: <http://www.inf.ufsc.br/~j.barreto/tutoriais/Survey.pdf>. Acesso em: 10 out. 2019.

BENITEZ-DAVALOS, Walter Ramon et al. **Previsão das séries temporais dos deslocamentos horizontais dos blocos da barragem de Itaipu por meio de redes neurais recorrentes do tipo LSTM**. 2019. Disponível em: <https://www.researchgate.net/publication/332353377_Previsao_das_series_temporais_dos_deslocamentos_horizontais_dos_blocos_da_barragem_de_Itaipu_por_meio_de_redes_neurais_recorrentes_do_tipo_LSTM>. Acesso em: 23 mai. 2020.

BERNHARD, Shelby D. et al. **Clickstream Prediction Using Sequential Stream Mining Techniques with Markov Chains**. 2016. Disponível em: <https://dl.acm.org/citation.cfm?doid=2938503.2938535>. Acesso em: 03 nov. 2019.

BUSINESS WIRE. **Global Software-as-a-Service (SaaS) Market (2018-2023)**. Disponível em: <https://www.businesswire.com/news/home/20181114005369/en/Global-Software-as-a-Service-SaaS-Market-Outlook-2018-2023-Expected>. Acesso em: 28 ago. 2019.

CORRÊA, Débora Cristina. **Sistema baseado em redes neurais para composição musical assistida por computador**. 2008. Disponível em: <https://repositorio.ufscar.br/handle/ufscar/379>. Acesso em: 26 out. 2019.

DORFFNER, Georg. **Neural Networks for Time Series Processing**. 1996. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.5697>.

Acesso em: 09 abr. 2020.

DUBEY, Abhijit; WAGLE, Dilip. **Delivering software as a service**. 2007. Disponível em: <http://www.pocsolutions.net/Delivering_software_as_a_service.pdf>. Acesso em: 28 ago. 2019.

FRIAS-MARTINEZ, Enrique; KARAMCHETI, Vijay. **A prediction model for user access sequences**. 2002. Disponível em: <https://www.researchgate.net/profile/Enrique_Frias-Martinez2/publication/228952893_A_Scalable_Behavior_Model_for_Temporal_Prediction_of_Web_User_Access_Sequences/links/02e7e529313f1c413e000000.pdf>. Acesso em: 07 nov. 2019.

GERS, Felix A.; ECK, Douglas; SCHMIDHUBER, Jürgen. **Applying LSTM to Time Series Predictable Through Time-Window Approaches**. 2002. Disponível em: <https://link.springer.com/chapter/10.1007/978-1-4471-0219-9_20>. Acesso em: 09 abr. 2020.

GOODFELLOW, Ian J. et al. **Generative adversarial nets**. 2014. Disponível em: <https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>. Acesso em: 31 mai. 2020.

GÜNDÜZ, Şule; ÖZSU, M. Tamer. **A Web Page Prediction Model Based on Click-Stream Tree Representation of User Behavior**. 2003. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.1067&rep=rep1&type=pdf>. Acesso em: 12 nov. 2019.

GURNEY, Kevin. **An Introduction to Neural Networks**. Londres: Routledge, 1997. 317 p.

HE, Xiaodong; DENG, Li. **Deep learning for image-to-text generation: A technical overview**. 2017. Disponível em: <https://ieeexplore.ieee.org/document/8103169>. Acesso em: 31 mai. 2020.

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. **Long Short-term Memory**. 1997.

Disponível em: <https://www.bioinf.jku.at/publications/older/2604.pdf>. Acesso em: 24 out. 2019.

HOCHREITER, Sepp et al. **Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies**. 2001. Disponível em: <https://www.bioinf.jku.at/publications/older/ch7.pdf>. Acesso em: 23 mai. 2020.

HUCKO, Michal et al. **YesElf: Personalized Onboarding for Web Applications**. Disponível em: <https://dl.acm.org/citation.cfm?doid=3314183.3324978>. Acesso em: 10 out. 2019.

KERAS. **Keras**. 2020. Disponível em: <https://keras.io/>. Acesso em: 28 mar. 2020.

LIPTON, Zachary C.; BERKOWITZ, John. **A Critical Review of Recurrent Neural Networks for Sequence Learning**. 2015. Disponível em: <https://arxiv.org/pdf/1506.00019.pdf>. Acesso em: 10 out. 2019.

LONG, Jonathan; SHELHAMER, Evan; DARRELL, Trevor. **Fully convolutional networks for semantic segmentation**. 2015. Disponível em: <https://ieeexplore.ieee.org/document/7298965>. Acesso em: 31 mai. 2020.

MDN Web Docs. **Expressões Regulares**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_Expressions>. Acesso em: 26 out. 2019.

MEIRELLES, Fernando S. **30ª Pesquisa Anual do Uso de TI nas Empresas, 2019**. 2019. Disponível em: <https://eaesp.fgv.br/sites/eaesp.fgv.br/files/pesti2019fgvciappt_2019.pdf>. Acesso em: 24 ago. 2019

MITCHELL, Tom M. **Machine Learning**. EUA: McGraw-Hill Education - Europe, 1997. 352 p. ISBN 0071154671.

MONARD, Maria Carolina, BARANAUSKAS, José Augusto. Conceitos Sobre Aprendizado de Máquina. **Sistemas Inteligentes Fundamentos e Aplicações**. 1.

ed. Barueri-SP: Manole Ltda, 2003. p. 89--114. ISBN 85-204-168.

NANOPOULOS, Alexandros; KATSAROS, Dimitris; MANOLOPOULOS, Yannis. **Effective Prediction of Web-user Accesses: a Data Mining Approach**. 2001. Disponível em: <https://www.researchgate.net/publication/2361417_Effective_Prediction_of_Web-user_Accesses_a_Data_Mining_Approach>. Acesso em: 07 nov. 2019.

NELSON, David; PEREIRA, Adriano; DE OLIVEIRA, Renato. **Stock market's price movement prediction with LSTM neural networks**. 2017. Disponível em: <https://www.researchgate.net/publication/318329563_Stock_market's_price_movement_prediction_with_LSTM_neural_networks>. Acesso em: 23 mai. 2020.

NILSSON, Nils Johan. **Artificial Intelligence:** A New Synthesis. 1. ed. San Francisco-CA: Morgan Kaufmann Publishers Inc, 1998. 513 p. ISBN 1-55860-467-7.

OLAH, Christopher. **Understanding LSTM Networks**. 2015. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acesso em: 27 out. 2019.

PYTHON. **Python**. 2020. Disponível em: <https://www.python.org/about/apps/>. Acesso em: 28 mar. 2020.

RAUBER, Thomas Walter. **Redes Neurais Artificiais**. 2005.

REN, Shaoqing et al. **Faster r-cnn: Towards real-time object detection with region proposal networks**. 2015. Disponível em: <https://ieeexplore.ieee.org/document/7485869>. Acesso em: 31 mai. 2020.

ROSENBLATT, F. **The perceptron**: a probabilistic model for information storage and organization in the brain. 1958. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>. Acesso em: 10 out. 2019.

RUMELHART, David. E.; HINTON, Geoffrey. E.; WILLIAMS, Ronald. J. **Learning**

Internal Representations by Error Propagation. 1985. Disponível em: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf>. Acesso em: 10 out. 2019.

SUNDERMEYER, Martin; SCHLUTER, Ralf; NEY, Hermann. **LSTM Neural Networks for Language Modeling**. 2012. Disponível em: <https://www.isca-speech.org/archive/archive_papers/interspeech_2012/i12_0194.pdf>. Acesso em: 23 mai. 2020.

TENSORFLOW. **Why TensorFlow**. Disponível em: <https://www.tensorflow.org/about>. Acesso em: 28 mar. 2020.

VELLASCO, Marley Maria Bernardes Rebuzzi. **Redes Neurais Artificiais**. 2007. Disponível em: <http://www2.ica.ele.puc-rio.br/Downloads/33/ICA-introdu%C3%A7%C3%A3o%20RNs.pdf>. Acesso em: 10 out. 2019.

W3C. **Document Object Model Events**. Disponível em: <https://www.w3.org/TR/DOM-Level-2-Events/events.html>. Acesso em: 20 out. 2019.