UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
TRABALHO FINAL DE GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

Comparação de modelos de aprendizado profundo na classificação de comentários contendo discurso de ódio na internet

Rafael Greca Vieira

5 de dezembro de 2021

Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ - UNIFEI
TRABALHO FINAL DE GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

Rafael Greca Vieira

Comparação de modelos de aprendizado profundo na classificação de comentários contendo discurso de ódio na internet

Monografia apresentada como trabalho final de graduação, requisito parcial para obtenção do título de Bacharel em Ciência da Computação, sob orientação da Profa. Dra. Isabela Neves Drummond.

Orientador: Profa. Dra. Isabela Neves Drummond

5 de dezembro de 2021
Itajubá

Agradecimentos

Agradeço aos meus pais, Sérgio e Cristiane, e a minha irmã, Isabela, por sempre terem me apoiado, incentivado e acreditado em mim durante toda a minha trajetória. Sem vocês nada disso teria sido possível.

A todos os professores do Instituto de Matemática e Computação da Universidade Federal de Itajubá - Campus Itajubá pelos ensinamentos durante todo o curso de Ciência da Computação, em especial à Profa. Dra. Isabela Neves Drummond por todas as aulas ministradas, por ter aceitado me orientar neste trabalho de conclusão de curso e por todo ensinamento, paciência e confiança.

A todos os amigos do curso de Ciência da Computação e Sistemas de Informação que fiz durante a faculdade, que me acompanharam e apoiaram durante todos esses anos, por todo conhecimento compartilhado e pela amizade.

A todos os meus amigos de fora da faculdade, desde a infância até hoje, por fazerem parte da minha trajetória e sempre terem me apoiado.

"Todos os modelos estão errados, mas alguns são úteis."
(George Box)

Resumo

A grande evolução da internet nos últimos anos desencadeou o surgimento das redes sociais, onde é possível se conectar com outras pessoas de forma remota e em tempo real. A área que foi a maior beneficiada com esse evento foi a de Processamento de Linguagem Natural devido ao grande volume de dados gerados e disponíveis diariamente. Um dos campos de estudo dessa área é a chamada Análise de Sentimentos, que tem como objetivo analisar os sentimentos e opiniões das pessoas em relação a um determinado indivíduo, produto, serviço ou evento. Este trabalho se propõe a estudar dois modelos de aprendizado profundo: uma Rede Neural Convolucional e uma rede do tipo *Long Short-Term Memory*, para classificação de *tweets* contendo discurso de ódio. A base de dados empregada foi retirada de uma competição disponível no site Analytics Vidhya. Ademais, o estudo busca analisar o efeito da utilização de técnicas de balanceamento e pré-processamento de dados nos modelos citados. Os resultados obtidos mostram que a técnica de balanceamento trouxe melhorias em ambos algoritmos, o que não foi possível observar para a técnica de pré-processamento, e que o modelo que obteve o melhor resultado tanto em tempo de execução, quanto em desempenho foi a rede *Long Short-Term Memory*.

Palavras-chaves: Análise de Sentimentos; Aprendizado Profundo; Rede Neural Convolucional; *Long Short-Term Memory*; Sobreamostragem de Dados; Pré-processamento de Dados.

Abstract

The great evolution of the internet in recent years has triggered the emergence of social networks, where it is possible to connect with other people remotely and in real time. The area that benefited the most from this event was Natural Language Processing, due to the large volume of data generated and available daily. One of the fields of study in this area is called Sentiment Analysis, which aims to analyze people's feelings and opinions in relation to a particular individual, product, service or event. This paper proposes to study two deep learning models: a Convolutional Neural Network and a Long Short-Term Memory type network, to classify tweets containing hate speech. The database used was taken from a competition available on the Analytics Vidhya's website. Furthermore, the study aims to analyze the effect of using data balancing and data pre-processing in the aforementioned models. The results obtained show that the balancing technique brought improvements in both algorithms, which was not possible to observe for the pre-processing technique, and that the model that obtained the best result both in execution time and performance was the Long Short-Term Memory network.

Key-words: Sentiment Analysis; Deep Learning; Convolutional Neural Network; Long Short-Term Memory; Data Oversampling; Data Preprocessing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

FN	Falso Negativo
FP	Falso Positivo
LSTM	Long Short-Term Memory
PLN	Processamento de Linguagem Natural
RNA	Rede Neural Artificial
RNC	Rede Neural Convolucional
RNR	Rede Neural Recorrente
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

Sumário

1 Introdução

Com a grande evolução da internet nos anos 90, não demorou muito para que as primeiras redes sociais começassem a surgir. Porém, sua popularização se deu apenas depois dos anos 2000 com a criação de algumas plataformas como: MySpace, Twitter e o Facebook. Nelas é possível se conectar com pessoas diferentes, postar fotos, vídeos, publicações e até conversar, em tempo real, com outra pessoa através do computador.

Essa popularização teve como consequência um aumento exponencial do volume de dados, de vários tipos, disponíveis e gerados todos os dias, tornando indispensável uma evolução computacional para que fosse possível lidar com a grande quantidade de dados. Este trabalho final de graduação tem como foco a análise de dados em formato de texto. Eles são trabalhados na subárea de Inteligência Artificial chamada de Processamento de Linguagem Natural (PLN), cujo objetivo é fazer com que o computador consiga compreender as palavras utilizadas nos mais diversos idiomas (Chopra, Prashar, & Sain, 2013).

Um dos principais campos de estudo na área de PLN, e que será abordado neste trabalho, é a análise de sentimentos, também chamada de mineração de opinião. Descrevendo de uma maneira bem sucinta, essa área de estudo tem como principal finalidade analisar a opinião humana, extraída de comentários publicados nas redes sociais, fóruns de discussão e sites de venda, por exemplo, seus sentimentos e emoções sobre um determinado assunto ou produto (Torfi, Shirvani, Keneshloo, Tavaf, & Fox, 2021).

Os algoritmos de aprendizado profundo, aqueles que são capazes de extrair um alto nível de representação dos dados, estão se destacando cada vez mais na área de inteligência artificial devido a sua baixa dependência de interferência humana no processo de aprendizado, alto volume de dados disponibilizados publicamente e a evolução do poder computacional das máquinas atuais. O uso desses modelos tem alcançado resultados promissores na análise de sentimentos e nos diversos campos de estudo da área (LeCun, Bengio, & Hinton, 2015). Sendo assim, os modelos de aprendizado profundo escolhidos para serem utilizados no trabalho foram a Rede Neural Convolucional (RNC) e a *Long Short-Term Memory* (LSTM).

O objetivo principal deste trabalho é avaliar e comparar o desempenho de algoritmos de aprendizado profundo na classificação de *tweets* contendo discurso de ódio. De acordo com Badjatiya, Gupta, Gupta, e Varma (2017), um *tweet* pode ser classificado como discurso de ódio quando apresenta um discurso abusivo em relação a um determinado indivíduo ou grupo (LGBT, gênero, religião, dentre outros). O trabalho também tem como objetivo trazer uma análise a respeito da utilização de técnicas de pré-processamento e balanceamento de dados, avaliando o impacto do emprego destas técnicas no desempenho dos modelos de classificação.

Para a realização deste trabalho foi utilizado um conjunto de dados chamado *Twitter*

Sentiment Analysis, disponibilizado pelo Kaggle e pelo Analytics Vidhya, que é composto por *tweets* em inglês que podem ou não apresentar discurso de ódio. Os resultados dos modelos obtidos na etapa de experimentos foram comparados entre si, onde a utilização da técnica de balanceamento trouxe melhorias para ambos os algoritmos, ao contrário da utilização do pré-processamento. Em relação à performance dos modelos, ambos tiveram resultados bem próximos, sendo o menor resultado obtido para a RNC foi 99,06% e 99,56% para a LSTM. Já em relação ao tempo de execução, a LSTM apresentou um resultado bem superior a RNC.

Esta monografia possui a seguinte estrutura, além deste capítulo de introdução:

- O Capítulo 2 apresenta os conceitos teóricos a respeito do campo de estudo escolhido, análise de sentimentos, as áreas de aprendizado de máquina e aprendizado profundo, os algoritmos selecionados, a métrica de avaliação utilizada e, por fim, alguns trabalhos relacionados ao tema do estudo, e que empregam os modelos de aprendizado profundo abordados;
- O Capítulo 3 detalha a metodologia seguida para este trabalho, descrevendo a base de dados utilizada, o pré-processamento realizado nos dados, como é feito o balanceamento dos dados, informações do ambiente onde foram executados os experimentos e como os modelos desenvolvidos são avaliados;
- O Capítulo 4 traz os experimentos e resultados obtidos para cada um dos algoritmos de aprendizado profundo;
- Por fim, no Capítulo 5, estão as principais conclusões acerca do trabalho desenvolvido, além das propostas de trabalhos futuros.

2 Fundamentação Teórica

Este Capítulo tem como finalidade descrever os conceitos mais importantes e fundamentais sobre os tópicos abordados no trabalho. Nele, será apresentado sobre análise de sentimentos na Seção 2.1, aprendizado de máquina na Seção 2.2 e aprendizado profundo na Seção 2.3. A seção que aborda o aprendizado profundo discute um pouco de Rede Neural Artificial, Rede Neural Convolucional e Rede Neural Recorrente. Por fim, na seção 2.4 será detalhada a métrica principal utilizada na avaliação dos modelos desenvolvidos e serão apresentados na seção 2.5 outros trabalhos da literatura que estão relacionados com o tema do trabalho.

2.1 Análise de Sentimentos

A área de PLN tem como principal foco a interação entre a linguagem humana e o computador, possibilitando que o computador consiga entendê-la. Essa área possui diversas aplicações, como reconhecimento de fala, segmentação de tópicos, sumarização de texto, tradução de idiomas e análise de sentimentos (Lopez & Kalita, 2017), sendo a última um dos campos de estudo mais ativos dessa área e que visa analisar os sentimentos, opiniões e emoções das pessoas em relação a um determinado alvo, como um indivíduo, produto, serviço ou evento (B. Liu, 2012).

A análise de sentimentos, também chamada de mineração de opinião, vem ganhando ainda mais destaque nos últimos anos. Isso se deve, principalmente, ao grande crescimento das redes sociais, como o Twitter, e blogs pessoais. Neles os usuários são livres para falarem sobre o que quiserem, seja a respeito de uma opinião política, opinião sobre um determinado assunto, avaliação de um produto ou empresa, dentre outras possibilidades, criando uma vasta gama de tópicos explorados (Kouloumpis, Wilson, & Moore, 2011).

Além das redes sociais e blogs, os fóruns de discussão, onde milhares de pessoas se reúnem com o mesmo propósito de discutir e opinar sobre um determinado assunto, e sites de venda, onde os clientes podem avaliar os produtos e a qualidade do serviço de uma determinada empresa, também proporcionam uma grande visibilidade nesse campo de estudo, já que milhões de dados são gerados diariamente.

A análise de sentimento pode ser tratada como um problema de classificação, pois almeja detectar qual a polaridade e o sentimento contido em um determinado texto. Com o avanço dos estudos nessa área, essa classificação não está mais limitada em ser apenas um sentimento positivo ou negativo, mas também é possível detectar as emoções em diferentes tópicos (Ramadhani & Goo, 2017).

Trata-se de uma área muito abrangente, que possui aplicações diversas. Algumas delas

são: análise sobre investimentos no setor financeiro, reputação de uma marca nas redes sociais e avaliação dos clientes sobre um determinado produto ou serviço (Feldman, 2013).

2.2 Aprendizado de Máquina

Diversos projetos na área de inteligência artificial utilizam regras de inferência lógica para tomar suas decisões a partir de uma base de conhecimento predefinida. A grande dependência desse tipo de sistema ocasionou a criação de um sistema que possui capacidade de adquirir seu próprio conhecimento por meio da extração de padrões em dados brutos. Essa capacidade é chamada de aprendizado de máquina (Goodfellow, Bengio, & Courville, 2016).

A etapa de aprendizagem de um modelo é realizada durante o processo de treinamento, onde, além de produzir os resultados desejados para os dados utilizados no treinamento, o modelo aprende a generalizar visando produzir os resultados esperados para dados nunca antes vistos (El Naqa & Murphy, 2015).

O modelo de aprendizado de máquina pode ser categorizado levando em consideração o seu processo de aprendizagem. As categorias possíveis são: aprendizagem supervisionada, não-supervisionada, semi-supervisionada e por reforço (El Naqa & Murphy, 2015).

Os modelos supervisionados empregam amostras de treinamento, ou seja, dados que representam as classes presentes num conjunto de dados, no processo de treinamento. Os não-supervisionados aprendem através do reconhecimento de padrões ocultos nos dados, calculando a similaridade entre os dados de um conjunto. Já os modelos semi-supervisionados possuem os dois tipos de processos de aprendizado. E, por fim, os modelos de aprendizado por reforço podem ser vistos como um tipo de modelo supervisionado, porém a classe não é apresentada no processo de aprendizado, o modelo aprende o que fazer visando maximizar uma determinada recompensa (El Naqa & Murphy, 2015).

Além da rotulação dos modelos em relação a sua aprendizagem, eles também podem ser classificados de acordo com o tipo de tarefa que desejam solucionar, podendo ser de classificação ou regressão.

Para a tarefa de classificação, o resultado do modelo será um valor discreto e que representa uma classe dentro de um determinado conjunto de possíveis classes. Já para a tarefa de regressão, o resultado do modelo será um valor contínuo e de infinitas possibilidades (Rätsch, 2004).

Diferente algoritmos de aprendizado de máquina são definidos na literatura, sendo aplicados nos mais diversos tipos de dados, incluindo aqueles no formato de texto. O foco do presente trabalho é a aplicação de modelos de classificação a dados do tipo texto. Kajla, Hooda, Saini, et al. (2020) realizam uma comparação utilizando seis desses algoritmos na classificação de comentários tóxicos. São eles: Regressão Logística, Naive Bayes, Árvore de Decisão,

Florestas Randômicas, *K-Nearest Neighbors* e *Support Vector Machine* (SVM). O que teve o melhor resultado, em termos de desempenho, foi o algoritmo de Regressão Logística.

2.3 Aprendizado Profundo

O aprendizado profundo é uma subárea do aprendizado de máquina e possibilita que a máquina consiga extrair e aprender conceitos complexos através de uma hierarquia de conceitos mais simples. Essa técnica permite que a máquina adquira conhecimentos através de experiências (Goodfellow et al., 2016).

A essência dos algoritmos de aprendizado profundo é a rede neural artificial de multicamadas, ou *Perceptron* de multicamadas, representada na Figura 2 (Goodfellow et al., 2016). Hoje em dia, existem centenas de redes neurais com arquiteturas e classificações diferentes. Shrestha e Mahmood (2019) fazem uma revisão desses algoritmos, como a Rede Neural Convolucional (RNC), *Long Short-Term Memory* (LSTM), *Autoencoder*, entre outros, comparando-os e mostrando suas aplicações mais comuns.

Devido ao alto nível de representação e abstração das suas múltiplas camadas de processamento, o aprendizado profundo aperfeiçoou o estado da arte em diversas áreas, como: detecção de objetos, reconhecimento de voz, processamento de imagens, textos, videos, descoberta de drogas, dentre outras (LeCun et al., 2015).

2.3.1 Rede Neural Artificial

A Rede Neural Artificial (RNA) é inspirada nos modelos biológicos humanos. A rede neural é composta pelos seguintes elementos: neurônio artificial, que representa os neurônios biológicos; os dados de entrada, que representam os sinais que são recebidos pelos dendritos; pesos, que representam as forças dos dados de entrada; a função de ativação, que representa uma função que irá ou não ativar o neurônio (característica não linear); e a saída, representada pela impulso do neurônio (Abraham, 2005; Gershenson, 2003).

A saída é computada através da soma ponderada das entradas com seus respectivos pesos. Esse valor será passado para a função de ativação. Por fim, o resultado dessa função será utilizado em uma outra função de ativação que varia dependendo do objetivo do modelo. Quando não é utilizada a segunda função de ativação (o valor final é igual ao valor resultante da primeira função de ativação), o neurônio pode ser chamado de linear (Gershenson, 2003).

Um dos primeiros modelos, e o mais conhecido, de uma rede neural foi o *Perceptron*, composto por uma camada que possui apenas um neurônio, conforme mostrado na Figura 1.

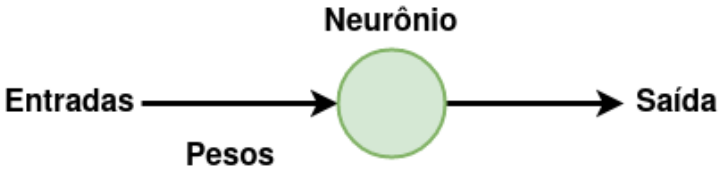


Figura 1 – Perceptron.

Uma evolução do modelo anterior, e consequentemente mais robusto, é a RNA multicamadas. Ela é composta por três camadas: camada de entrada, camada escondida e camada de saída. Através da camada de entrada, os dados são apresentados para a rede; Portanto, é a primeira camada da rede. A camada escondida vem logo em seguida, e recebe como entrada a saída dos neurônios da camada de entrada. Por fim, a última camada é a de saída, que gera o resultado para os dados apresentados (Abraham, 2005) (vide representação na Figura 2). Quando a rede possui mais de uma camada escondida ela é chamada de rede neural profunda.

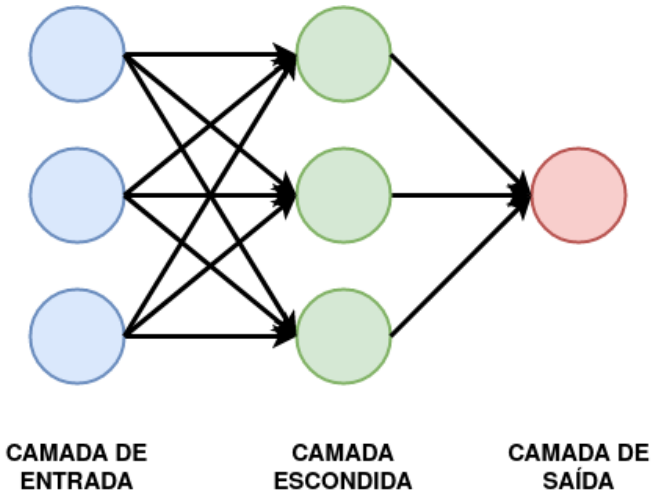


Figura 2 – Rede Neural Artificial Multicamadas

A RNA tradicional é o exemplo mais conhecido de uma rede neural *feed-forward*. Em uma rede *feed-forward*, as informações percorrem a rede neural em somente uma direção. As setas representadas na Figura 2 correspondem ao fluxo das informações na rede. Elas entram pela camada de entrada, depois seguem para a(s) camada(s) escondida(s) até chegar na de saída.

O processo de aprendizado de uma rede neural se dá pela atualização dos pesos durante a fase de treinamento por meio de um processo supervisionado. O erro é calculado pela diferença da saída esperada com a saída obtida da rede. O algoritmo de aprendizado *backpropagation* é responsável por atualizar os pesos da RNA, minimizando os erros, percorrendo o caminho oposto, ou seja, da camada de saída até a de entrada (Werbos, 1990).

2.3.2 Rede Neural Convolucional (RNC)

A Rede Neural Convolucional (RNC), ou Rede Convolucional, possui a mesma ideia e funcionamento que a RNA multicamadas. A grande diferença entre as duas é que a RNC é composta por camadas convolucionais ao invés da tradicional camada escondida da RNA multicamadas (Goodfellow et al., 2016). Um exemplo de arquitetura dessa rede pode ser visto na Figura 3.

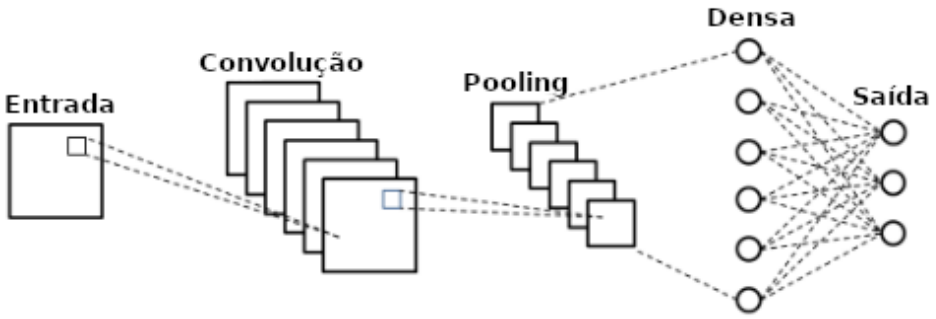


Figura 3 – Rede Neural Convolucional. Fonte: Adaptado de Phung et al. (2019)

Diferente de uma RNA comum, a RNC recebe uma entrada no formato de uma matriz numérica, contendo pelo menos uma dimensão, como uma imagem, texto, série temporal ou áudio (Goodfellow et al., 2016). A imagem é representada pelos valores dos seus pixels, o texto pelos valores numéricos das suas palavras através de um *embedding*, série temporal com suas informações em relação a um determinado período de tempo sequencial e áudio com o timbre e frequência da voz, por exemplo.

Além das camadas de entrada e saída, a arquitetura da RNC é composta de outras três camadas: convolucional, *pooling* e densa.

A camada convolucional é uma das principais camadas responsáveis pela extração de características dos dados. Ela utiliza filtros, também chamados de *kernels*, de pequena dimensão e que são aprendidos ao longo do treinamento da rede neural (O'Shea & Nash, 2015). Os filtros são deslocados pela matriz de entrada, de acordo com um determinado valor de *stride*, realizando a operação de convolução entre ela e o filtro. A saída dessa camada é um mapa de informações do mesmo tamanho que a quantidade de filtros utilizados (P. Kim, 2017).

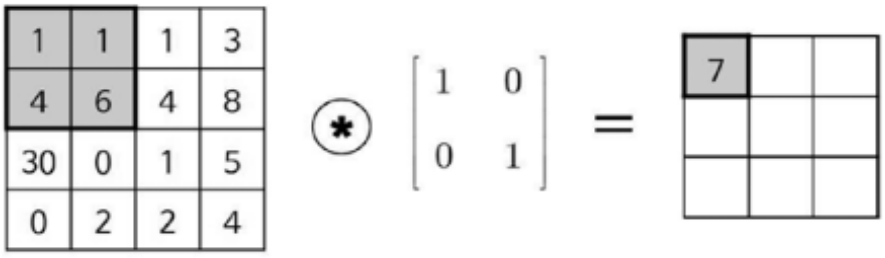


Figura 4 – Primeiro passo na aplicação do filtro. Fonte: P. Kim (2017)

Na Figura 4, a entrada é representada pela primeira matriz, a operação de convolução pelo símbolo do asterisco, o filtro pela segunda matriz e, por fim, o resultado gerado pela terceira matriz. Uma pequena área da matriz é selecionada, com dimensão igual a do filtro, realizando a operação de convolução entre eles e o resultado é então armazenado em uma nova matriz. Na Figura 5, foi selecionada outra área deslocando um espaço para o lado, *stride* de valor igual a 1 (um), e depois realizando as mesmas etapas mencionadas anteriormente. Esse processo é feito até que toda a área da entrada tenha sido utilizada, resultando em um novo mapa de informações. Para um número de filtros igual a 6 (seis), a saída da camada de convolução ficaria como a que está representada na Figura 3.

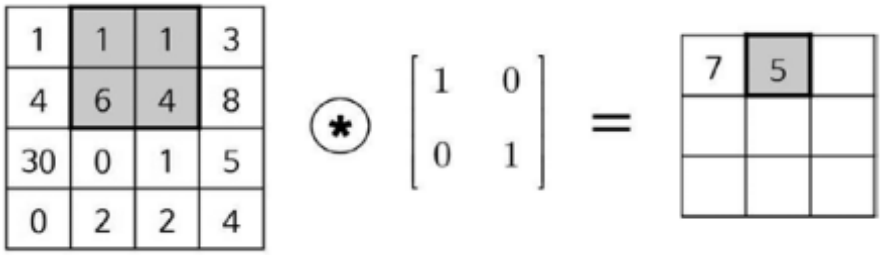


Figura 5 – Segundo passo na aplicação do filtro. Fonte: P. Kim (2017)

A camada *pooling* é a responsável por reduzir a dimensão da saída da camada anterior para as próximas camadas. Isso garante que o modelo fique menos complexo, pois terá menos parâmetros para serem treinados, sem perder as informações mais relevantes (O'Shea & Nash, 2015). Porém, por mais que o conteúdo dessas informações seja preservado, a sua informação espacial, como a posição original no dado, é perdida (Albawi, Mohammed, & Al-Zawi, 2017).

Os neurônios da camada densa, também chamada de totalmente conectada, possui uma estrutura idêntica à maneira como os neurônios da RNA multicamadas estão conectados. Todos eles são conectados com os neurônios da camada anterior e posterior (Albawi et al., 2017).

Para que a RNC possa ser utilizada nas tarefas das áreas de PLN, é necessário adicionar uma nova camada na rede neural, a de *embedding*. Essa camada é a responsável por gerar o *embedding*, também chamado de matriz de sentenças, e que será a entrada da rede.

Todas as sentenças da base de dados são compostas por sequências de palavras, representadas neste trabalho pelos *tweets*. Um dicionário é criado com todas as palavras que estão contidas nessas sentenças. Cada uma delas será representada por um vetor de pequena dimensão de valores contínuos. O *embedding* se dá pela concatenação de todos os vetores das palavras do dicionário, formando uma matriz (Severyn & Moschitti, 2015).

Na Figura 6, está representado um modelo RNC *multichannel* e um exemplo da representação de um *embedding*.

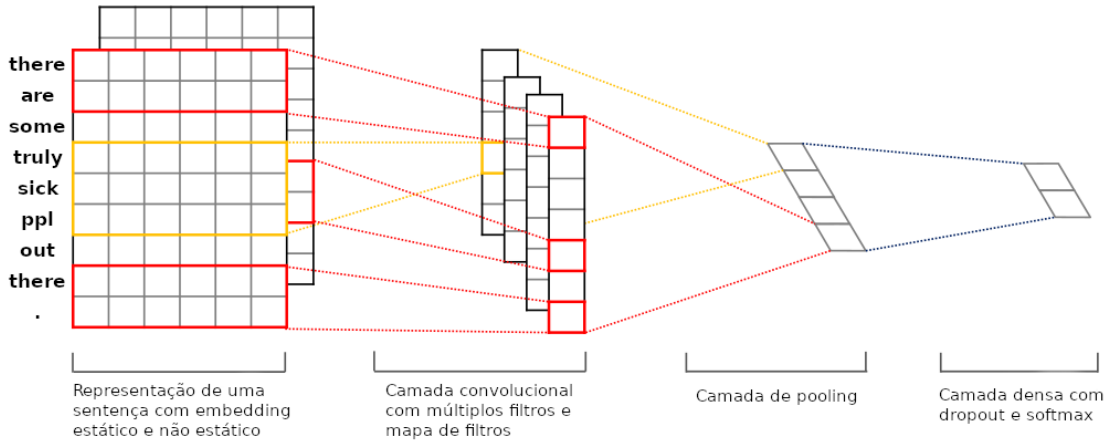


Figura 6 – Modelo RNC *multichannel*. Fonte: Adaptado de Y. Kim (2014)

2.3.3 Rede Neural Recorrente

Além da rede neural *feed-forward*, como a RNC e a RNA, citadas anteriormente, também existe a rede onde as informações não fluem em apenas uma direção. Ela é conhecida como Rede Neural Recorrente (RNR).

A principal característica da RNR é que elas são especialistas em processar dados sequenciais, como dados temporais e textos, e conseguem processar dados que possuem diferentes tamanhos (Goodfellow et al., 2016). Na Figura 7 é possível ver um modelo simples da RNR.

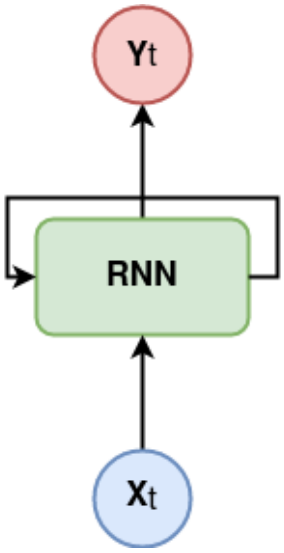


Figura 7 – Modelo RNR simples.

Ainda na Figura 7, o quadrado verde representa a camada RNR, composta por uma rede neural com uma determinada função de ativação, enquanto os círculos vermelho e azul representam a saída e a entrada dado um determinado tempo t, respectivamente. A camada RNR também gera o estado interno h_{t} no tempo t que será utilizada no tempo $t+1$, representado pela seta que conecta a camada RNR a ela mesma.

O estado interno h_{t} é calculado usando o estado interno anterior $h_{\mathrm{t\text{-}1}}$ e a entrada x_{t}, como mostra a Equação 2.1. A função de ativação mais utilizada é a tangente hiperbólica ($tanh$). W^{h} e W^{x} correspondem, respectivamente, ao peso do estado interno e da entrada.

$$h_{\mathrm{t}} = f(W^{\mathrm{h}} * h_{\mathrm{t\text{-}1}} + W^{\mathrm{x}} * x_{\mathrm{t}}) \tag{2.1}$$

A RNR possui uma característica importante: ela compartilha seus parâmetros ao longo de todo o modelo. Isso quer dizer que os pesos da entrada, saída e do estado interno são os mesmos para todas as entradas. Isso torna possível aplicar o modelo em diferentes formatos de sentenças, como o tamanho, e generalizar bem para sentenças não vistas antes (Goodfellow et al., 2016).

A Figura 8 representa a RNR da Figura 7 só que desdobrada, onde cada camada corresponde a uma palavra. Então, se uma sentença possui n palavras, a rede desdobrada terá n camadas.

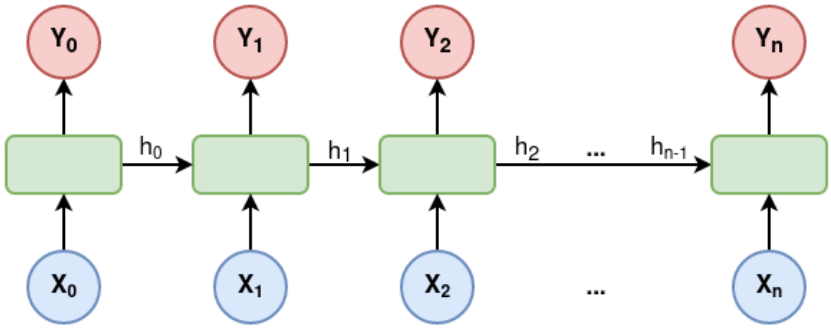


Figura 8 – Modelo RNR simples desdobrada.

Apesar da ideia de utilizar os estados internos como uma memória das entradas anteriores, a RNR possui um grande problema: durante o treinamento da rede, quando a sentença é muito grande, o gradiente pode aumentar (*exploding*) ou cair (*vashining*) exponencialmente, dificultando o aprendizado de longo prazo entre as palavras. Para solucionar esse problema foi proposta a *Long Short-Term Memory* (LSTM) (P. Liu, Qiu, & Huang, 2016).

2.3.3.1 *Long Short-Term Memory* (LSTM)

O principal objetivo da LSTM é solucionar o problema de aprender dependências de longo prazo encontrado na RNR tradicional. Essa rede possui, além do estado interno, uma célula de memória, também chamada de célula de estado, que possui seu conteúdo alterado no decorrer do treinamento (P. Liu et al., 2016).

A LSTM apresenta a mesma ideia de compartilhamento dos pesos e a mesma estrutura da RNR demostradas nas Figuras 7 e 8, porém, além dos estados citados anteriormente, ela possui quatro redes neurais dentro da camada ao invés de uma (Zhang, Wang, & Liu, 2018). Na Figura 9, está representado um modelo da LSTM, onde a célula de estado no tempo t é retratada como h(t), a entrada no tempo t como x(t), a célula de estado no tempo t como c(t) e as quatro redes neurais estão representadas pelos quadrados vermelhos.

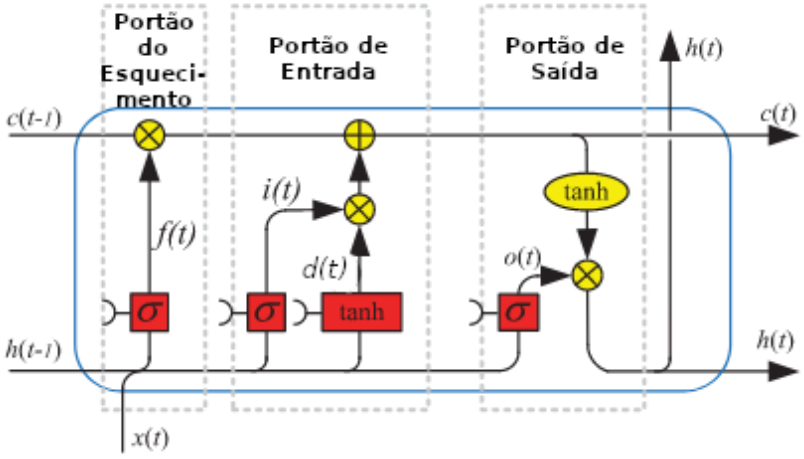


Figura 9 – Modelo LSTM. Fonte: Adaptado de Yu et al. (2019)

A primeira etapa é passar pelo *forget gate* ("portão do esquecimento", em tradução livre) onde a LSTM irá decidir, através da saída da função *sigmoid*, quais informações da célula de memória ela vai descartar. A função sigmoid gera uma saída com valores entre 0 e 1, sendo que 0 significa que a informação será totalmente excluída e 1 ela será totalmente preservada (Zhang et al., 2018). Essa saída está representada na Equação 2.2.

$$f(t) = \sigma(W^{\mathrm{f}} * h(t-1) + U^{\mathrm{f}} * x(t)) \tag{2.2}$$

A próxima etapa é o *input gate* ("portão de entrada", em tradução livre) onde a rede irá decidir quais novas informações serão armazenadas na célula de memória. Primeiro a função $sigmoid$ decidirá quais valores serão atualizados utilizando h$(t-1)$ e x(t), como mostra a Equação 2.3. Em seguida, utilizando a mesma entrada da função $sigmoid$, a função $tanh$ irá obter os novos valores, como mostra a Equação 2.4.

$$i(t) = \sigma(W^{\mathrm{i}} * h(t-1) + U^{\mathrm{i}} * x(t)) \tag{2.3}$$

$$d(t) = tanh(W^{\mathrm{n}} * h(t-1) + U^{\mathrm{n}} * x(t)) \tag{2.4}$$

Após essa etapa a célula de memória será atualizada utilizando os resultados obtidos nas Equações 2.2, 2.3 e 2.4, como mostra a Equação 2.5.

$$c(t) = c(t-1) * f(t) + i(t) * d(t) \tag{2.5}$$

Por fim, a LSTM passará pelo *output gate* ("portão de saída", em tradução livre) onde serão definidos os valores que formarão a saída. Essa etapa também possui a mesma entrada

que as anteriores, $h(t-1)$ e $x(t)$, e possui uma função $sigmoid$, como mostra a Equação 2.6. O resultado dessa equação será utilizado junto com a célula de memória para atualizar a célula de estado e a saída, como mostra a Equação 2.7.

$$o(t) = \sigma(W^{o} * h(t-1) + U^{o} * x(t)) \tag{2.6}$$

$$h(t) = tanh(c(t)) * o(t) \tag{2.7}$$

As redes do tipo LSTM apresentaram mais facilidade no aprendizado de longas dependências do que as RNR tradicionais, obtendo resultados bastante satisfatórios nas mais diversas aplicações, como reconhecimento de voz, tradução entre idiomas e descrição de imagens (Goodfellow et al., 2016).

2.4 Métrica de Avaliação

A avaliação do desempenho de um modelo de aprendizado supervisionado pode ser realizada empregando diferentes métricas. Um dos objetivos deste trabalho é comparar diferentes modelos de aprendizado profundo na tarefa de classificação, e para tanto, é fundamental a utilização de um conjunto de métricas para que esse objetivo seja alcançado. Portanto, a métrica escolhida para ser utilizada no trabalho foi a *F1-Score*, já que ela também é utilizada para avaliar o modelo na competição hospedada no site do Analytics Vidhya.

A métrica *F1-Score* é composta de outras duas métricas: precisão e revocação (tradução livre de *Precision* e *Recall*, respectivamente). A precisão é calculada utilizando Verdadeiro Positivo (VP) e Falso Positivo (FP), como mostra a Equação 2.8. Já a revocação é calculada utilizando VP e Falso Negativo (FN), como mostra a Equação 2.9.

$$Precisão = \frac{VP}{VP + FP} \tag{2.8}$$

$$Revocação = \frac{VP}{VP + FN} \tag{2.9}$$

A fórmula do *F1-Score* está representada na Equação 2.10.

$$F1 - Score = \frac{2 * (Revocação * Precisão)}{Revocação + Precisão} \tag{2.10}$$

2.5 Trabalhos Relacionados

Existem inúmeros trabalhos na litetura com estudos voltados para a área de análise de sentimentos, não apenas com foco na classificação de discurso de ódio, e que utilizam diferentes

abordagens na tratativa do conjunto de dados e modelos de classificação, sendo eles de aprendizado profundo ou de máquina. Esta seção reúne um conjunto desses trabalhos que empregam, mas não limitados, modelos de aprendizado profundo.

Ibrahim, Torki, e El-Makky (2018) utilizaram técnicas de pré-processamento e balanceamento de dados na detecção de comentários tóxicos em um conjunto de dados disponibilizados no Kaggle. Foram realizados dois tipos de classificação: binária (possui toxicidade ou não) e múltiplas categorias. Para cada uma das classificações, foram utilizados três modelos de aprendizado profundo: RNC, LSTM bidirecional e *Gated Recurrent Unit* (GRU) bidirecional. O modelo que apresentou o melhor resultado em ambas classificações foi uma combinação dos três modelos citados anteriormente.

Lee, Yoon, e Jung (2018) realizaram uma análise de discurso de ódio, no nível de caracteres e de palavras, em *tweets* utilizando técnicas de pré-processamento e comparando os resultados obtidos a partir de alguns algoritmos de aprendizado profundo (RNC e RNR) e de aprendizado de máquina clássico (Regressão Logística, *Naive Bayes*, *Gradient Boosted Trees*, Floresta Aleatória e *Support Vector Machine*). O modelo que obteve o melhor resultado foi uma variação da RNR realizando a análise no nível de palavras.

Badjatiya et al. (2017) compararam resultados obtidos a partir de algoritmos de aprendizado profundo (RNC, LSTM e *FastText*), aprendizado de máquina clássico (*Gradient Boosted Decision Trees* (GBDT)) e modelos de base (TF-IDF, *Char n-grams* e *Bag of Words*) na classificação de *tweets* contendo discurso de ódio. O modelo que obteve o melhor resultado foi o que utiliza a combinação da LSTM com o modelo GBDT.

Heikal, Torki, e El-Makky (2018) realizaram uma análise de sentimentos em *tweets* árabes usando técnicas de pré-processamento e modelos de aprendizado profundo (RNC e LSTM bidirecional). O melhor resultado foi obtido a partir do modelo que utiliza a combinação dos dois citados anteriormente, superando o estado da arte no conjunto de dados utilizado.

3 Metodologia

Este Capítulo tem como objetivo descrever todas as etapas executadas neste trabalho, desde a coleta da base de dados utilizada até a avaliação dos modelos selecionados, conforme mostra o fluxograma da Figura 10.

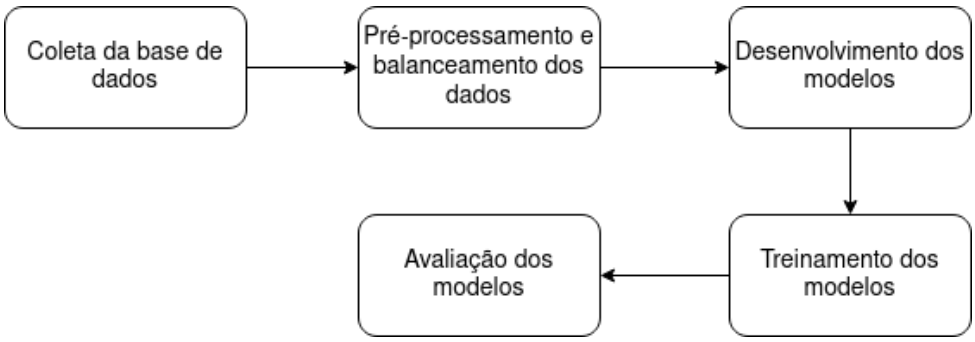


Figura 10 – Fluxograma das etapas do trabalho.

3.1 Ferramentas Utilizadas

Os códigos desenvolvidos durante o trabalho foram feitos utilizando a versão 3.8.10 da linguagem de programação *Python* 3. Devido a sua simplicidade, flexibilidade e poder, a biblioteca Keras é bastante utilizada no meio acadêmico e em grandes empresas no mercado de trabalho, como NASA e YouTube, (Chollet et al., 2015). Portanto, ela foi escolhida para ser utilizada no trabalho na criação dos modelos de aprendizado profundo, além de também ter sido utilizada na etapa de obtenção dos *tokens* e no preenchimento das sentenças (mais conhecido como *padding*).

Para a etapa de pré-processamento dos dados, foi utilizada a versão 3.6.2 da biblioteca NLTK (Loper & Bird, 2002) para a remoção de *stop words* e a versão 2.2.1 da biblioteca RE, nativa do Python, para o desenvolvimento das expressões regulares das funções descritas nesta etapa.

Para a etapa de balanceamento dos dados, foi utilizada a versão 0.8.0 da biblioteca *Imbalanced-Learn* (Lemaître, Nogueira, & Aridas, 2017).

Por fim, para a etapa de avaliação dos modelos foi utilizada a versão 0.24.2 da biblioteca *scikit-learn* (Pedregosa et al., 2011). As métricas escolhidas para a avaliação dos modelos foram a matriz de confusão e o *F1-Score*.

3.2 Base de Dados

A base de dados utilizada no trabalho se chama *Twitter Sentiment Analysis* e foi originalmente disponibilizada em uma competição no site Analytics Vidhya[1], uma comunidade que proporciona cursos e diversas competições na área de inteligência artificial, que teve seu início no começo de 2018 e tem como objetivo avaliar o desempenho dos modelos desenvolvidos pelos membros da comunidade. Mais tarde, esse conjunto de dados também foi disponibilizado no site Kaggle.

A base é composta por publicações em inglês feitas por usuários do Twitter. Ela é dividida em dois arquivos: treinamento, que possui aproximadamente 32 mil *tweets* categorizados como contendo ou não discurso de ódio, e o arquivo de teste, que possui aproxidamente 17 mil *tweets* e é utilizado dentro da competição para avaliar o modelo desenvolvido. A Figura 11 mostra alguns dos *tweets* que estão contidos na base de treinamento.

id	label	tweet
1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
3	0	bihday your majesty
4	0	#model i love u take with u all the time in urð±!!! ðððð¦ð¦ð¦
5	0	factsguide: society now #motivation
6	0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo
7	0	@user camping tomorrow @user @user @user @user @user @user @user dannyâ¦
8	0	the next school year is the year for exams.ð¯ can't think about that ð #school #exams #hate #imagine #actorslife #revolutionschool #girl
9	0	we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers â¦
10	0	@user @user welcome here ! i'm it's so #gr8 !
11	0	â #ireland consumer price index (mom) climbed from previous 0.2% to 0.5% in may #blog #silver #gold #forex
12	0	we are so selfish. #orlando #standwithorlando #pulseshooting #orlandoshooting #biggerproblems #selfish #heabreaking #values #love #
13	0	i get to see my daddy today!! #80days #gettingfed
14	1	@user #cnn calls #michigan middle school 'build the wall' chant " #tcot
15	1	no comment! in #australia #opkillingbay #seashepherd #helpcovedolphins #thecove #helpcovedolphins
16	0	ouch...junior is angryð#got7 #junior #yugyoem #omg
17	0	i am thankful for having a paner. #thankful #positive

Figura 11 – Exemplos de mensagens da base de dados.

Cada *tweet* apresenta um id, que é um número único de identificação, um booleano identificando se contém ou não discurso de ódio e, por fim, o seu conteúdo bruto. Na Figura 12, é possível visualizar como os dados estão distribuídos dentro da base de treinamento. Os *tweets* que não contêm discurso de ódio são a maioria, quase 30 mil, enquanto os *tweets* que possuem discurso de ódio são apenas um pouco mais de 2 mil.

[1] https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/

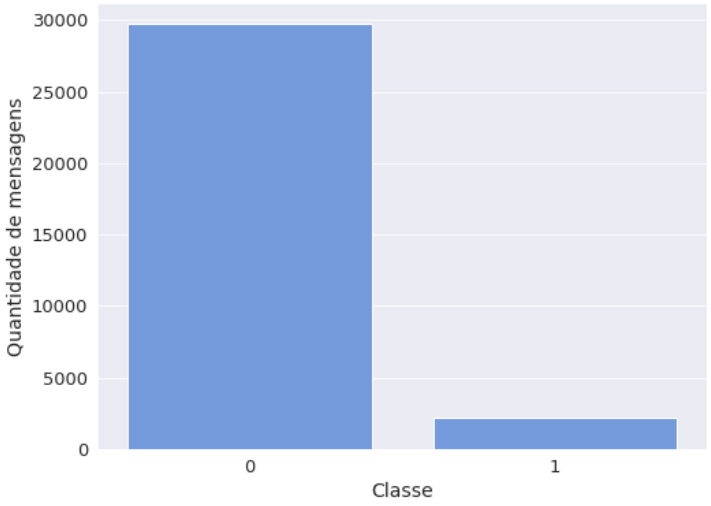


Figura 12 – Distribuição dos dados.

3.3 Pré-processamento dos Dados

Os dados coletados das redes sociais, principalmente o Twitter, possuem uma característica que deve ser levada em consideração: não têm uma estrutura definida. Os *tweets* são limitados, máximo de 280 caracteres, e podem conter alguns elementos específicos, como: menção de usuário, *hashtags*, *links*, gírias, abreviações, erros ortográficos e *emojis*. Para este trabalho, optou-se por realizar os testes dos modelos com e sem a utilização do pré-processamento na base de dados.

Esta etapa utiliza um pré-processamento com base no que foi utilizado em (Ramadhani & Goo, 2017) e (Neethu & Rajasree, 2013), onde são removidos itens das mensagens que não representam informações significativas, e efetuadas correções nas palavras, buscando melhor representação da informação. Para a realização desta etapa, foram criadas expressões regulares específicas para cada tipo de remoção.

3.3.1 Remoção de menções de usuário

Um usuário é mencionado no *tweet* quando é utilizado o símbolo @ (arroba) seguido do nome do usuário que será mencionado. Por não possuir nenhum valor informativo, as menções de usuários são removidas.

3.3.2 Remoção de pontuações e *hashtags*

A *hashtag* é um elemento muito utilizado em redes sociais como Twitter e Instagram, e representa uma palavra-chave sobre aquilo que está sendo falado no *tweet*. Ela é utilizada com o símbolo # (*hashtag*) seguido do termo. Por possuírem uma informação importante sobre aquilo

que está sendo falado, elas são mantidas. Porém, o símbolo da *hashtag* é removido. Além dele, todos as pontuações (como vírgula, ponto e vírgula, exclamação, interrogação, ponto, dentre outros) também são removidas.

Os caracteres com acentos também são removidos, sendo substituídos pelo mesmo caractere, só que sem o acento, por exemplo: após a remoção dos acentos, "café"passa a ser "cafe".

3.3.3 Remoção de *links* e *hyperlinks*

Todos os *links*, seja de outro site ou do próprio Twitter, são removidos por não possuirem um valor informativo. Como os dados estão sendo extraídos de uma página na internet, pode acontecer de coletar informações de elementos HTML, também conhecidos por *hyperlinks*, por engano junto com o conteúdo do *tweet*. Esses elementos também são removidos.

3.3.4 Remoção de *stop words*

As *stop words* (ou palavras de parada, em tradução livre) são palavras que aparecem com muita frequência nos textos e, por isso, não agregam nenhum valor semântico. Para a remoção dessas palavras é utilizada a biblioteca NTLK e seu conjunto de *stop words* de palavras em inglês.

3.3.5 Remoção de números, caracteres não-alfanuméricos e soltos

Na base de dados em questão é bem comum encontrar números e caracteres não - alfanuméricos, isto é, caracteres que não são números e nem letras do alfabeto (como ’¦’, ’ð’ e ’©’), nos *tweets*. Eles também são removidos, mantendo apenas os caracteres e espaços.

Caracteres soltos são aqueles que possuem somente um caractere cercado por espaços. Eles podem ser *stop words* (como ’a’, ’e’ e ’o’) ou o resultado de um erro de digitação do usuário. Por não possuírem valor semântico, eles são removidos.

3.3.6 Transformação de caracteres em minúsculo

Todos os caracteres dos *tweets* serão transformados para a sua versão em minúsculo. Consequentemente, palavras escritas de maneiras diferentes, como "Pizza"e "PIZZA", tornar-se-ão escritas da mesma forma: "pizza". Essa transformação resulta na diminuição da quantidade de palavras únicas dentro do conjunto de dados.

3.3.7 Correção de gírias e contrações

Através de um dicionário, criado baseado em códigos disponibilizados na plataforma Kaggle, algumas gírias comuns foram corrigidas para suas palavras escritas de forma correta através da utilização de uma expressão regular. As contrações são muito utilizadas no idioma

inglês para encurtar palavras e até mesmo algumas expressões. Elas foram transformadas para a sua estrutura original utilizando uma expressão regular.

Após a limpeza realizada na etapa de pré-processamento, muitos *tweets* apresentaram excesso de espaços em branco devido à remoção de palavras. Foi utilizada uma expressão regular para remoção desses espaços extras. Além disso, os *tweets* duplicados foram retirados utilizando a função *drop duplicates* da biblioteca Pandas, mantendo apenas uma das ocorrências.

A Tabela 1 mostra alguns *tweets* extraídos da base de dados antes e depois de passar por toda etapa de pré-processamento.

Tabela 1 – Exemplos de *tweets* antes e depois da etapa de pré-processamento

Antes do pré-processamento	**Depois do pré-processamento**
model i love u take with u all the time in urð±!!! ðððððð¦ð¦ð¦	model love take time
@user #cnn calls #michigan middle school 'build the wall' chant " #tcot	cnn calls michigan middle school build wall chant tcot
@user i'll always hope that one day i'll get to hug you, but i don't think that it's gonna happen anytime soon...	always hope one day get hug think gonna happen anytime soon
i am thankful for good friends. #thankful #positive	thankful good friends thankful positive
ð©the white establishment can't have blk folx running around loving themselves and promoting our greatness	white establishment blk folx running around loving promoting greatness
words r free, it's how u use em that can cost you! #verbal #abuse #hu #love #adult #teen @user	words free use em cost verbal abuse hu love adult teen

3.4 Balanceamento dos Dados

Na Figura 12, é possível notar que a base de dados utilizada é totalmente desbalanceada, já que a classe majoritária representa, aproximadamente, 93% da quantidade total. Essa distruição não normal das classes pode afetar o aprendizado do modelo, pois a classificação será mais tendenciosa para a classe majoritária (Wang et al., 2016).

Para este trabalho, também optou-se por realizar os testes dos modelos com e sem balanceamento da base de dados. Para o balanceamento dos dados foi utilizada a função *Random Over Sampler* da biblioteca *Imbalanced-learn*. Trata-se de uma técnica de sobreamostragem que incrementa o volume da classe minoritária até que ela e a classe majoritária fiquem com o mesmo volume. A Figura 13 apresenta a nova distribuição dos dados após a realização do balanceamento.

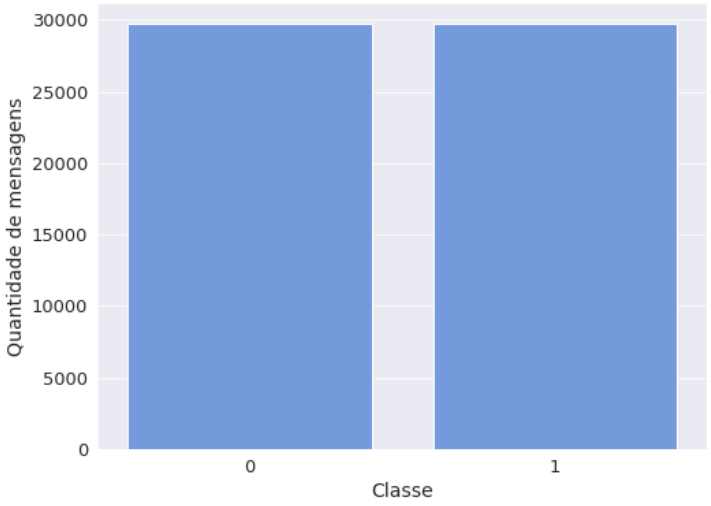


Figura 13 – Nova distribuição dos dados.

3.5 Modelos de Classificação

Esta Seção tem como objetivo descrever os modelos de aprendizado profundo que foram utilizados visando concluir o objetivo proposto neste trabalho.

3.5.1 RNC

Por ter apresentado uma melhoria no estado da arte na área de análise de sentimentos na época que o artigo foi publicado, o modelo de RNC selecionado para estudo é aquele proposto por Y. Kim (2014). O autor propõe quatro variações para o modelo: *rand*, *static*, *non-static* e *multichannel*. Todos possuem a mesma arquitetura: uma camada *embedding* de 300 dimensões, uma camada convolucional formada por três camadas convolucionais de tamanhos diferentes para os filtros (3, 4 e 5) e que foram concatenadas verticalmente, uma camada *dropout* com valor de 0.5 e uma camada densa com dois neurônios de saída.

A diferença entre os quatro modelos está na camada do *embedding*. No modelo *rand*, ele é inicializado de forma aleatória. No modelo *static* é utilizado um *embedding* pré-treinado e que será mantido durante todo o treinamento. Já no modelo *non-static*, é utilizado o mesmo *embedding* do modelo anterior, porém seus valores são treinados durante o treinamento. Por fim, o modelo *multichannel* utiliza duas camadas do *embedding* pré-treinado dos modelos anteriores. Porém um será mantido fixo e o outro refinado durante o treinamento.

O código disponibilizado pelo autor do artigo foi feito utilizando a biblioteca *Theano*. As principais funções utilizadas no desenvolvimento da arquitetura das variantes dos modelos citados anteriormente foram: *Embedding*, para a criação da camada de *embedding*; *Dense*, para a criação da camada de saída; *Convolution2D*, para a criação das camadas convolucionais e

Concatenate para concatená-las; *MaxPooling2D*, para a criação das camadas de *pooling*. Todas essas funções se encontram na biblioteca Keras.

O *embedding* utilizado nos modelos *static* e *non-static* é o *Word2Vec* pré-treinado com 100 bilhões de palavras usando notícias extraídas do Google e que é o mesmo utilizado pelo autor.

Por se tratar de um problema binário, a quantidade total dos neurônios na camada densa foi alterada para apenas um. Além disso, a arquitetura e os hiperparâmetros principais dos modelos utilizados foram mantidos com o valor descrito no artigo. Para o tamanho do *mini-batch* foi utilizado 100, tamanho do *pool* igual a 2, 10 épocas para o treinamento e o ADAM foi utilizado como o otimizador.

3.5.2 LSTM

Devido a sua simplicidade e aos bons resultados obtidos, o modelo LSTM escolhido foi o proposto pelo Badjatiya et al. (2017). O autor propõe duas variações para o mesmo modelo: LSTM com o *embedding* inicializado aleatoriamente e LSTM utilizando um *embedding* pré-treinado. Também é proposta uma variação dos dois modelos citados, onde eles são utilizados junto com um algoritmo de Árvore de Decisão, mas esta versão não é empregada neste trabalho. O código disponibilizado pelo autor foi empregado como base para a construção do modelo aqui testado.

O *embedding* pré-treinado utilizado no modelo é o *GloVe* treinado com 2 bilhões de *tweets* e que possui 1.2 milhões de palavras, o mesmo que foi utilizado pelo autor.

Os modelos possuem a mesma arquitetura: uma camada *embedding* de 200 dimensões, uma camada *dropout* com valor de 0.25, uma camada LSTM com 50 unidades, *dropout* com valor de 0.5 e uma camada densa com três neurônios de saída.

Por se tratar de um problema binário, a quantidade total dos neurônios na camada densa foi alterada para apenas um. Os outros hiperparâmetros foram mantidos com os valores encontrados no código disponibilizado pelo autor. A quantidade de *mini-batch*, épocas de treinamento e otimizador foram as mesmas citadas no modelo RNC.

3.6 Ambiente de Execução

Todas as etapas do trabalho foram executadas utilizando a plataforma *Google Colaboratory*, também chamada de *Google Colab*. Nela, é possível executar, de forma gratuita e remota pelo navegador, um código em *Python* no formato de um arquivo similar ao *Jupyter Notebook* e que está armazenado no *Google Drive*. Essa plataforma também oferece acesso a *GPUs*, aproximadamente 12.7 GB de memória RAM disponíveis e cerca de 107.7 GB de armazenamento no total (Research, 2018).

3.7 Avaliação dos Modelos

Para a avaliação dos modelos, foi utilizada a métrica *F1-Score* descrita no Capítulo 2. Por se tratar de uma base de dados desbalanceada e para que a performance do modelo não seja enviesada, o valor utilizado no parâmetro *average* do *F1-Score* é o *weighted*. Além desta métrica, também foi empregada a matriz de confusão, composta por valores obtidos a partir dos conceitos de VP, FP, FN e Verdadeiro Negativo (VN). As funções que implementam as métricas citadas estão disponíveis na biblioteca *scikit-learn*.

4 Experimentos e Resultados

Este Capítulo tem como objetivo descrever os experimentos realizados e os resultados obtidos. Primeiro, foi realizado um experimento para verificar o impacto que as etapas de pré-processamento e balanceamento podem causar no modelo. O modelo que apresentou a melhor performance foi selecionado para ter seus hiperparâmetros variados visando melhorar a sua avaliação.

A base de dados de treinamento foi dividida em 85% dos dados para serem utilizados no treinamento do modelo e os outros 15% para a validação do modelo obtido no treinamento. Para permitir a reprodução dos experimentos e comparação dos resultados, a execução foi realizada utilizando a CPU e duas sementes, 23 e 2109, para evitar a geração de números aleatórios na inicialização do modelo toda vez que for executado e que foram escolhidas aleatoriamente pelo autor.

4.1 RNC

O experimento que envolve a etapa de pré-processamento e balanceamento dos dados foi realizado nas variações do modelo de RNC e seus hiperparâmetros foram descritos no Capítulo 3. A Tabela 2 mostra os resultados obtidos para os modelos testados e a Tabela 3 mostra o tempo de execução, em segundos, para cada modelo executado. Assim como nas tabelas posteriores, os melhores resultados estão destacados em negrito.

Tabela 2 – Resultado da variação das etapas de pré-processamento e balanceamento na RNC

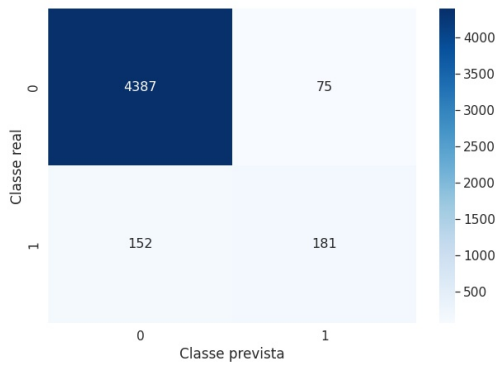
Modelo	**Balanceamento**	**Pré-processamento**	***F1-Score* validação**	***F1-Score* teste**	**Semente**
rand	Não	Não	94,98%	63,33%	23
	Não	Sim	93,14%	39,74%	23
	Sim	**Não**	**98,89%**	**71,33%**	**23**
	Sim	Sim	98,58%	70,6%	23
non-static	Não	Não	95,32%	65,88%	23
	Não	Sim	93,35%	42,7%	23
	Sim	Não	97,83%	65,93%	23
	Sim	Sim	97,66%	66,1%	23
rand	Não	Não	94,42%	65,85%	2109
	Não	Sim	94,53%	57,77%	2109
	Sim	**Não**	**99,13%**	**72,94%**	**2109**
	Sim	Sim	98,23%	73,3%	2109
non-static	Não	Não	94,28%	61,38%	2109
	Não	Sim	95,05%	63,37%	2109
	Sim	Não	97,64%	66,03%	2109
	Sim	Sim	94,98%	66,87%	2109

Tabela 3 – Tempo de execução da variação dos modelos RNC

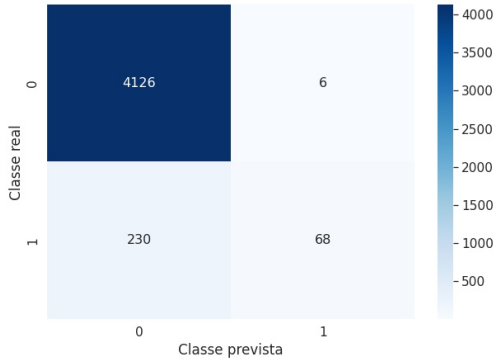
Modelo	**Balanceamento**	**Pré-processamento**	**Tempo de execução**	**Semente**
rand	Não	Não	1164,4	23
	Não	Sim	960,43	23
	Sim	**Não**	**1958,06**	**23**
	Sim	Sim	1779,55	23
non-static	Não	Não	1055,2	23
	Não	Sim	996,57	23
	Sim	Não	1961,18	23
	Sim	Sim	1779,35	23
rand	Não	Não	1176,54	2109
	Não	Sim	1057,28	2109
	Sim	**Não**	**2140,11**	**2109**
	Sim	Sim	1960,7	2109
non-static	Não	Não	1116,49	2109
	Não	Sim	1057,43	2109
	Sim	Não	2079,83	2109
	Sim	Sim	1900,76	2109

O modelo *static* também foi testado, porém não apresentou bons resultados. Isso pode ter acontecido porque o autor do artigo original utilizou um *embedding* que foi treinado usando notícias extraídas da Google e as palavras que não estavam contidas no *embedding* foram inicializadas aleatoriamente. Como a base utilizada no trabalho foi extraída do Twitter, muitas palavras podem não estar presentes no *embedding* pré-treinado.

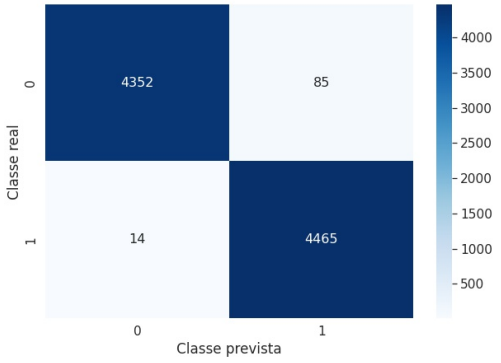
Com a utilização da técnica de balanceamento de dados, a RNC apresenta uma melhora significativa no desempenho com os dados de validação, como apresentado na Tabela 2. Quando o pré-processamento é utilizado junto com o balanceamento, houve uma pequena piora. Também é possível notar, na Tabela 2, que o modelo que obteve o melhor resultado para as duas sementes foi o *rand*; O que obteve o melhor resultado foi o modelo que utiliza apenas a técnica de balanceamento. Na Figura 14, é possível visualizar a matriz de confusão obtida a partir desses modelos.



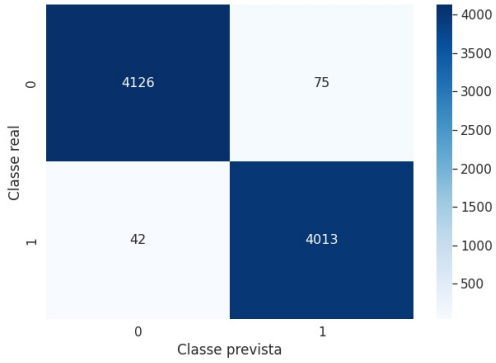
(a) Matriz de confusão *rand* sem técnicas e semente 23



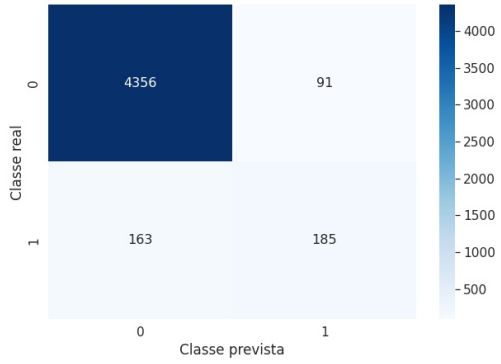
(b) Matriz de confusão *rand* com pré-processamento e semente 23



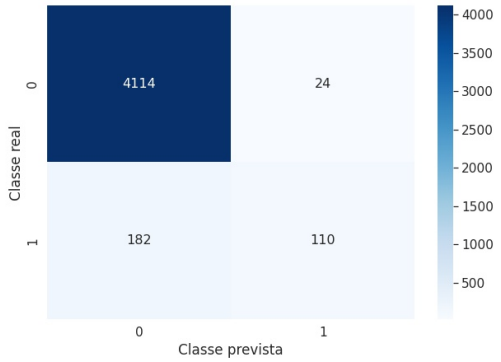
(c) Matriz de confusão *rand* com balanceamento e semente 23



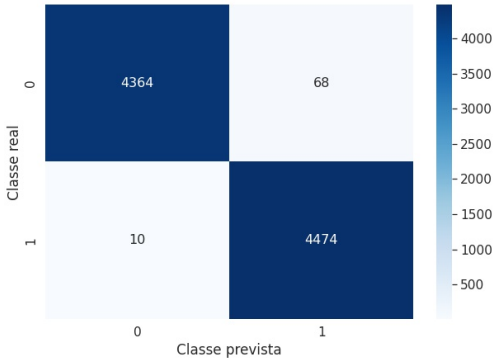
(d) Matriz de confusão *rand* com ambas as técnicas e semente 23



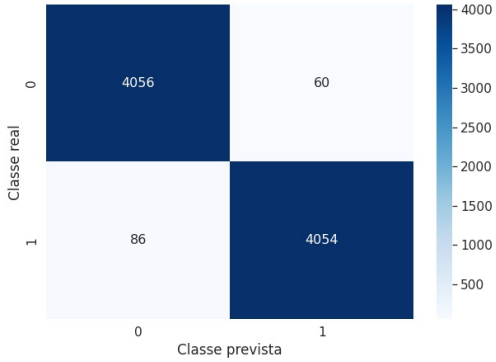
(e) Matriz de confusão *rand* sem técnicas e semente 2109



(f) Matriz de confusão *rand* com pré-processamento e semente 2109



(g) Matriz de confusão *rand* com balanceamento e semente 2109



(h) Matriz de confusão *rand* com balanceamento e semente 2109

Figura 14 – Matriz de confusão dos modelos RNC e suas variações

Na Figura 14, é possível perceber que, quando as classes estão desbalanceadas, a taxa de acerto da classe minoritária é baixa e que tende a piorar com a utilização do pré-processamento.

Após a seleção do melhor modelo de cada semente, alguns hiperparâmetros foram selecionados para serem variados: quantidade e tamanho dos filtros das camadas convolucionais, tamanho do *pool* das camadas de *pooling*, o valor do *dropout* e a constante de regularização $l2$ das camadas convolucionais.

Utilizando como base o melhor modelo encontrado em cada semente anteriormente, o primeiro experimento realizado foi a variação na quantidade de filtros nas camadas convolucionais. Para a semente 2109, não foi observada nenhuma melhoria; Já para a semente 23, os resultados obtidos estão apresentados na Tabela 4 com seus respectivos tempos de execução, medido em segundos.

Tabela 4 – Resultado da variação na quantidade de filtros na RNC

Quantidade de filtros	***F1-Score* validação**	**Tempo de execução**
80	98,52%	1857,52
90	98,51%	2016,6
110	**99,06%**	**2261,32**
120	98,53%	2501,93
130	98,38%	2503,46
140	98,87%	2565,25

Para os próximos experimentos, foi utilizado o modelo com 110 filtros para a semente 23, e o modelo inicial para a 2109. Os parâmetros variados foram o tamanho do filtro e a constante de regularização $l2$ das camadas convolucionais com os valores descritos na Tabela 5. As listas contidas no filtro correspondem aos valores da primeira, segunda e terceira camada convolucional, respectivamente. Porém, não foram obtidos modelos com melhor desempenho.

Tabela 5 – Valores testados para tamanho do filtro, *pool* e regularização l2 na RNC

Hiperparâmetro	**Valores**
Tamanho do filtro	[3, 3, 3], [4, 4, 4], [5, 5, 5], [4, 5, 6], [2, 4, 6], [3, 5, 7] e [4, 6, 8]
Tamanho do *pool*	4, 6, 8, 10 e 12
Regularização l2	1, 2, 4, 5, 6 e 7

Por fim, ainda utilizando o modelo com 110 filtros para a semente 23 e o modelo inicial para a 2109, foi realizada a variação do valor de probabilidade da camada de *dropout*. Os resultados obtidos para as duas sementes estão representados na Tabela 6.

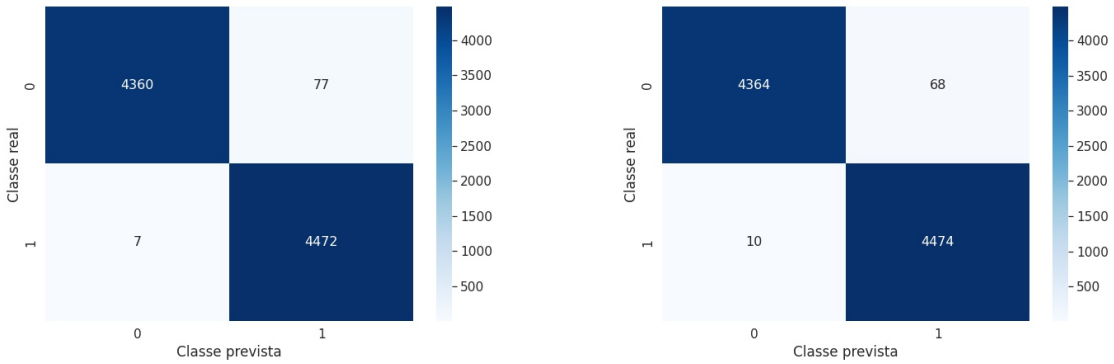
Tabela 6 – Resultado da variação no valor do dropout na RNC

Semente	**Valor *dropout***	***F1-Score* validação**	**Tempo de execução**
23	0,2	98,51%	2140,7
	0,3	98,55%	2120,9
	0,4	**99,06%**	**2140,79**
	0,6	98,75%	2133,33
	0,7	98,55%	2200,91
	0,8	98,74%	2200,71
2109	0,2	98,84%	2141,56
	0,3	**99,13%**	**2094,8**
	0,4	98,74%	2080,4
	0,6	98,87%	2140,02
	0,7	98,87%	2079,7
	0,8	98,69%	2139,92

Os valores finais dos hiperparâmetros das RNCs estão descritos na Tabela 7 e a matriz de confusão desses modelos está representada na Figura 15. O modelo obtido na semente 23 apresentou um *F1-Score* final de 99,06% nos dados de validação e 72,36% nos dados de teste. Já o modelo obtido na semente 2109 apresentou um *F1-Score* de 99,13% e 72,94% para os dados de validação e de teste, respectivamente.

Tabela 7 – Hiperparâmetros RNC

Hiperparâmetro	**Semente 23**	**Semente 2109**
Tamanho da entrada	30	30
Tamanho do *embedding*	300	300
Quantidade de filtros	110	100
Tamanho dos *kernels*	3, 4 e 5	3, 4 e 5
Tamanho do *pool*	2	2
Dropout	0,4	0,3
Regularização l2	3	30
Otimizador	ADAM	ADAM



(a) Matriz de confusão do modelo final da semente 23

(b) Matriz de confusão do modelo final da semente 2109

Figura 15 – Matriz de confusão das RNC finais

4.2 LSTM

O experimento que envolve a etapa de pré-processamento e balanceamento dos dados foi realizado nas variações do modelo de LSTM e seus hiperparâmetros foram descritos no Capítulo 3. Porém, como o autor do artigo não cita se a camada de *embedding* do modelo com o *embedding* pré-treinado foi mantida ou refinada durante o treinamento, as duas abordagens foram desenvolvidas. A Tabela 8 mostra os resultados obtidos para os modelos testados e a Tabela 9 mostra o tempo de execução, em segundos, para cada modelo executado.

Tabela 8 – Resultado da variação das etapas de pré-processamento e balanceamento na LSTM

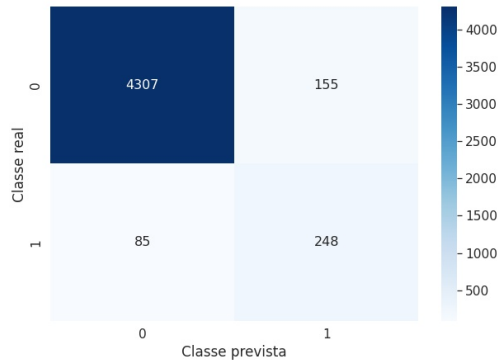
Modelo	**Balanceamento**	**Pré-processamento**	***F1-Score* validação**	***F1-Score* teste**	**Semente**
randômico	Não	Não	95,21%	69,71%	23
	Não	Sim	94,99%	69,33%	23
	Sim	**Não**	**99,35%**	**71,17%**	**23**
	Sim	Sim	99,24%	67,66%	23
estático	Não	Não	94,86%	54,87%	23
	Não	Sim	93,7%	53,44%	23
	Sim	Não	95,47%	49,74%	23
	Sim	Sim	95,2%	49,86%	23
não estático	Não	Não	96,03%	68,24%	23
	Não	Sim	95,32%	69,52%	23
	Sim	Não	99,28%	73,07%	23
	Sim	Sim	98,3%	69,83%	23
randômico	Não	Não	95,70%	70,33%	2109
	Não	Sim	95,01%	69,67%	2109
	Sim	**Não**	**99,23%**	**73,68%**	**2109**
	Sim	Sim	99,09%	70,82%	2109
estático	Não	Não	93,89%	55,71%	2109
	Não	Sim	93,97%	53,45%	2109
	Sim	Não	95,81%	48,87%	2109
	Sim	Sim	95,98%	55,38%	2109
não estático	Não	Não	95,73%	68,82%	2109
	Não	Sim	95,55%	67,89%	2109
	Sim	Não	98,83%	70,6%	2109
	Sim	Sim	98,37%	68,17%	2109

Tabela 9 – Tempo de execução da variação dos modelos LSTM

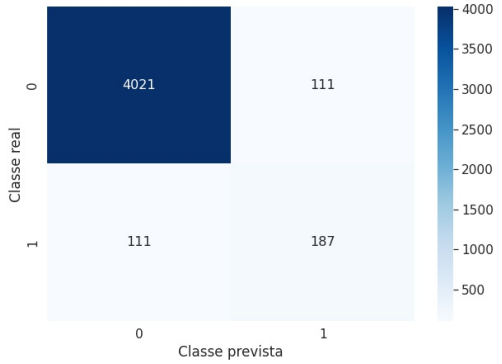
Modelo	Balanceamento	Pré-processamento	Tempo de execução	Semente
randômico	Não	Não	451,3	23
	Não	Sim	391,39	23
	Sim	**Não**	**812,62**	**23**
	Sim	Sim	753,78	23
estático	Não	Não	150,31	23
	Não	Sim	119,83	23
	Sim	Não	271,87	23
	Sim	Sim	273,34	23
não estático	Não	Não	451,02	23
	Não	Sim	392,36	23
	Sim	Não	812,58	23
	Sim	Sim	754,04	23
randômico	Não	Não	451,62	2109
	Não	Sim	403,87	2109
	Sim	**Não**	**873,23**	**2109**
	Sim	Sim	744,41	2109
estático	Não	Não	150,47	2109
	Não	Sim	151,74	2109
	Sim	Não	271,9	2109
	Sim	Sim	273,37	2109
não estático	Não	Não	431,84	2109
	Não	Sim	391,67	2109
	Sim	Não	812,37	2109
	Sim	Sim	713,37	2109

O modelo com o *embedding* estático foi o que obteve o melhor resultado em questão de tempo de execução, já que não foi necessário treinar a camada de *embedding*. Porém, foi o que obteve o pior resultado em questão de desempenho. Isso pode ser explicado pelo fato de se utilizar um *embedding* que foi treinado em outra base de dados.

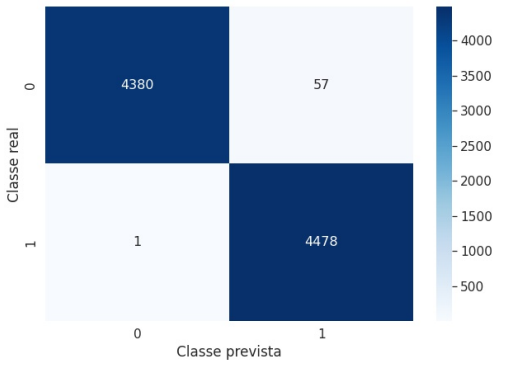
Com a utilização da técnica de balanceamento de dados, a LSTM apresenta uma melhora significativa no desempenho com os dados de validação, como apresentado na Tabela 8. Quando o pré-processamento é utilizado junto com o balanceamento, também houve uma pequena piora. Também é possível notar, na Tabela 8, que o modelo que obteve o melhor resultado para ambas as sementes foi o randômico. Entre esses modelos randômicos, o que obteve o melhor resultado foi o modelo que utiliza apenas a técnica de balanceamento. Na Figura 16, é possível visualizar a matriz de confusão gerada por esses modelos.



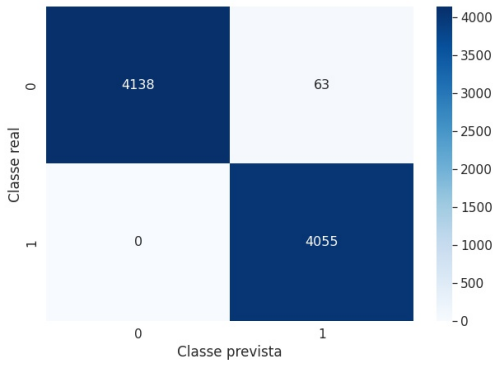
(a) Matriz de confusão modelo randômico sem técnicas e semente 23



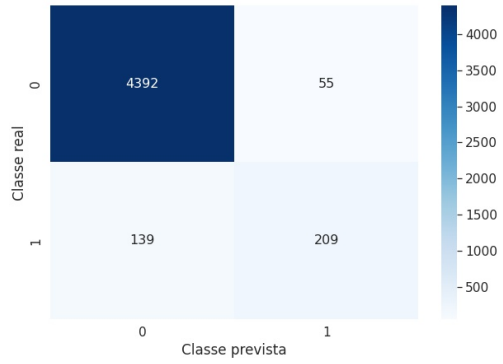
(b) Matriz de confusão modelo randômico com pré-processamento e semente 23



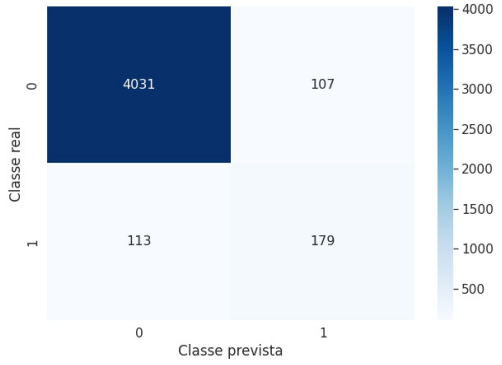
(c) Matriz de confusão modelo randômico com balanceamento e semente 23



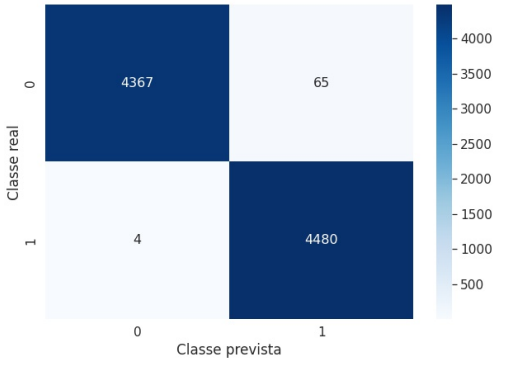
(d) Matriz de confusão modelo randômico com ambas as técnicas e semente 23



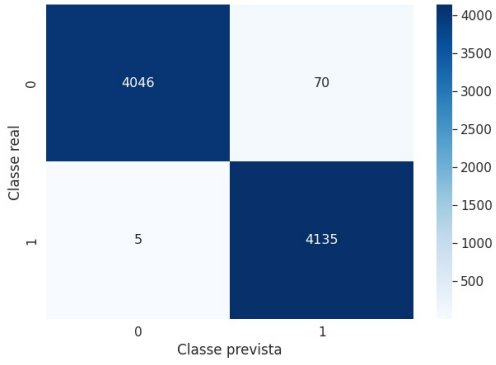
(e) Matriz de confusão modelo randômico sem técnicas e semente 2109



(f) Matriz de confusão modelo randômico com pré-processamento e semente 2109



(g) Matriz de confusão modelo randômico com balanceamento e semente 2109



(h) Matriz de confusão modelo randômico com balanceamento e semente 2109

Figura 16 – Matriz de confusão dos modelos LSTM e suas variações

Na Figura 16, é possível perceber que a LSTM possui uma taxa de acerto mais alta para a classe minoritária em relação à RNC, demonstrada na Figura 14, quando a base de dados está desbalanceada e que piora com a utilização do pré-processamento. Essa diferença entre os dois modelos é refletida nos resultados quando são avaliados em uma base com dados desconhecidos, onde a LSTM, considerando apenas os modelos randômicos e não estáticos, apresenta uma performance superior em dados desbalanceados se comparado a RNC.

Após a seleção do melhor modelo de cada semente, foram variados os seguintes hiperparâmetros: quantidade de unidades na camada LSTM, valor do *dropout* para ambas as camadas e a função de ativação da camada LSTM.

Utilizando como base o melhor modelo encontrado em cada semente anteriormente, o primeiro experimento realizado foi a variação na quantidade de unidades na camada LSTM. Os resultados encontrados em ambas as sementes se encontram na Tabela 10 com seus respectivos tempos de execução em segundos.

Tabela 10 – Variação na quantidade de unidades na LSTM

Semente	**Quantidade de unidades**	***F1-Score validação***	**Tempo de execução**
23	25	99,84%	627,03
	75	**99,87%**	**821,15**
	100	99,81%	916,1
	125	99,76%	1197,46
	150	99,8%	1849,56
2109	**25**	**99,56%**	**681,63**
	75	99,26%	934,03
	100	99,2%	997,47
	125	99,16%	1124,01
	150	99,43%	1246,4

Para os próximos experimentos, foi utilizado o modelo com 75 unidades para a semente 23, e o modelo com 25 unidades para a semente 2109. Os parâmetros variados foram os valores da primeira camada de *dropout*. Somente houve uma melhoria na semente 23, como mostra a Tabela 11.

Tabela 11 – Variação no valor do primeiro *dropout* na LSTM

Valor *dropout*	***F1-Score* validação**	**Tempo de execução**
0 (Sem)	**99,91%**	**1174,87**
0,1	99,83%	934,74
0,2	99,73%	1055,27
0,3	99,83%	1056,06
0,4	99,8%	1054,31
0,5	99,81%	993,35
0,6	99,81%	985,28

Para os próximos experimentos, foram utilizados os hiperparâmetros do valor da segunda camada de *dropout* e a função de ativação da LSTM com os valores presentes na Tabela 12. Porém, nenhuma melhoria foi obtida nos modelos.

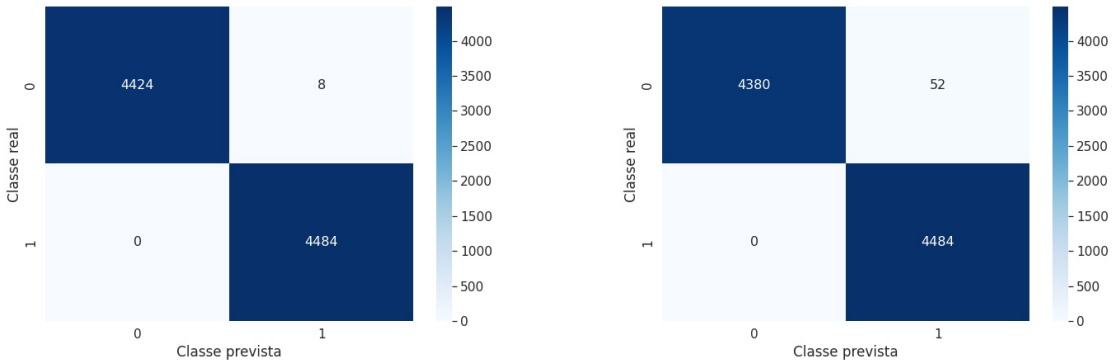
Tabela 12 – Valores testados para o segundo *dropout* e funções de ativação

Hiperparâmetro	**Valores**
Valores *dropout*	0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7
Função de ativação	softmax, relu, sigmoid

Os valores finais dos hiperparâmetros das LSTM estão descritos na Tabela 13 e a matriz de confusão desses modelos está representada na Figura 17. O modelo obtido para a semente 23 apresentou um *F1-Score* final de 99,91% para os dados de validação e 73,13% para os dados de teste. Já o modelo obtido na semente 2109 apresentou um *F1-Score* de 99,56% e 74,34% para validação e teste, respectivamente.

Tabela 13 – Hiperparâmetros LSTM

Hiperparâmetro	**Semente 23**	**Semente 2109**
Tamanho da entrada	30	30
Tamanho do *embedding*	200	200
Quantidade de filtros	75	25
Dropouts	0 (Sem) e 0,5	0,25 e 0,5
Otimizador	ADAM	ADAM



(a) Matriz de confusão do modelo final da semente 23

(b) Matriz de confusão do modelo final da semente 2109

Figura 17 – Matriz de confusão das LSTM finais

Através das Figuras 15 e 17, é possível notar que, mesmo os modelos obtendo um desempenho muito parecido nos dados de validação, o modelo LSTM tem um desempenho melhor na classificação de *tweets* contendo discurso de ódio.

4.3 Considerações Finais

É importante ressaltar que não foram encontrados artigos e nem outros trabalhos da literatura que utilizam a mesma base de dados utilizada neste trabalho. Sendo assim, os modelos foram comparados entre si utilizando como critério a base de dados de validação e a de teste, disponibilizada pela competição do Analytics Vidhya.

Como demonstrado nas Tabelas 2 e 8, em todos os casos foi possível observar que o melhor resultado é obtido utilizando somente a técnica de balanceamento dos dados e que a utilização do pré-processamento não produziu modelos com o melhor desempenho.

É possível notar também que, em alguns experimentos, um maior valor da medida *F1-Score* para os dados de validação não indica que o mesmo acontece para os dados de teste. Assim, foram levantadas algumas hipóteses:

- A competição também utiliza a técnica de *F1-Score* para obtenção do resultado, porém não indica qual o valor utilizado no parâmetro *average*. Neste trabalho, foi utilizado o *weighted*, que atribui peso as classes de acordo com a quantidade de instâncias verdadeiras, para considerar o desbalanceamento da base no cálculo do *F1-Score*. Caso o valor for outro (*micro*, *macro*, *samples* ou *binary*) e a base de teste também estiver desbalanceada, isso pode afetar o resultado final.

- Foi observado que menos da metade das palavras presentes na base de teste também estão contidas no vocabulário utilizado no treinamento dos modelos. Devido à simplicidade dos modelos testados e à utilização das etapas de pré-processamento e balanceamento dos

dados, pode ocorrer afetamento na forma como o modelo se comporta com palavras fora do vocabulário.

Sendo assim, na etapa de variação dos hiperparâmetros, foi utilizado como critério que o modelo apresentasse um ganho de perfomance em ambos conjuntos de dados, porém os de validação foram priorizados por se tratar de um conjunto de dados e um vocabulário de palavras conhecido pelo modelo.

Através da análise realizada no início de novembro de 2021, foi observado que existem 17.100 inscritos na competição e 1.267 submissões públicas no site. Após a etapa de variação dos hiperparâmetros da RNC e da LSTM, os melhores modelos obtidos foram utilizados nos dados de teste, submetidos para avaliação na competição e comparados com os resultados encontrados pelos outros participantes da competição do Analytics Vidhya. Através da Tabela 14, é possível visualizar que a RNC e a LSTM tiveram desempenhos muito próximos para as duas sementes, porém a LSTM obteve o melhor resultado nos dados de validação e de teste e com um tempo de execução muito inferior ao da RNC.

Tabela 14 – Comparação dos resultados dos melhores modelos obtidos

Semente	**Modelo**	***F1-Score* validação**	***F1-Score* teste**	**Tempo de execução**	**Classificação**
23	**LSTM**	**99,91%**	**73,13%**	**1174,87**	**259°**
	RNC	99,06%	72,25%	2140,79	320°
2109	**LSTM**	**99,56%**	**74,34%**	**681,63**	**177°**
	RNC	99,13%	72,94%	2094,8	273°

5 Conclusão

O objetivo principal deste trabalho final de graduação foi entender o funcionamento dos modelos de aprendizado profundo, RNC e LSTM, e realizar uma comparação do desempenho destes modelos quando aplicados para a classificação de uma base de dados composta por *tweets* contendo discurso de ódio. A base de dados empregada no estudo está hospedada no Analytics Vidhya.

Nesse contexto, por se tratar de um conjunto de dados desbalanceado e os *tweets* conterem diversas sentenças comuns do Twitter, como menção de usuário, *hashtags*, *links*, dentre outros, foram aplicadas técnicas de balanceamento e pré-processamento nos dados, visando comparar a avaliação dos modelos frente a diferentes bases de dados.

Comparando os resultados obtidos para a base de dados com e sem balanceamento e pré-processamento, todos os resultados obtiveram valores de avaliação superiores a 93% nos modelos para os dados de validação. Entretanto, quando não é utilizado o balanceamento, o desempenho do modelo fica enviesado, principalmente na RNC, devido ao alto desbalanceamento das classes. Entre as combinações das técnicas, a que obteve o melhor resultado em todos os modelos foi utilizando somente o balanceamento. A utilização da técnica de pré-processamento não gerou melhorias em praticamente todos os experimentos, principalmente nos melhores modelos, tendo um maior impacto na RNC quando as classes estão desbalanceadas, em relação a não utilização dessa técnica.

Para os melhores modelos, a RNC e a LSTM apresentaram *F1-Score* muito próximos nos dados de validação, porém o que obteve o melhor resultado para as duas sementes foi a LSTM.

No que se refere ao tempo de execução, a LSTM também obteve o melhor resultado, sendo executado em menos da metade do tempo gasto pela RNC. Isso pode ser explicado devido à simplicidade da arquitetura utilizada e pelo modelo em si ser especializado em dados sequenciais.

Como o PLN é uma área que começou a ganhar visibilidade recentemente, este trabalho pode ser expandido de inúmeras maneiras. Em relação às técnicas de comparação e execução, um trabalho futuro é a utilização da GPU na execução dos algoritmos e a técnica de validação cruzada, presente na biblioteca *Scikit-learn*, para a etapa de treinamento dos modelos a fim de obter uma comparação mais fiel e geral dos seus desempenhos.

Em relação ao pré-processamento, Camacho-Collados e Pilehvar (2017) conseguiram obter uma melhoria em um modelo de RNC utilizando um pré-processamento mais simples. Seria interessante realizar um experimento com a utilização de técnicas de pré-processamento

mais simples e completas para verificar se apresenta um resultado diferente do que foi apresentado neste trabalho.

Em relação aos modelos, seria interessante realizar testes com modelos de aprendizado profundo mais robustos, como o BERT (Devlin, Chang, Lee, & Toutanova, 2018), e modelos de aprendizado de máquina clássicos. Com isso, seria possível verificar se a utilização de algoritmos baseados em redes neurais apresenta um ganho significativo no desempenho em relação aos modelos estatísticos mais simples na base de dados em questão.

Uma outra linha de exploração do trabalho, é a utilização de bibliotecas já existentes na etapa de variação dos hiperparâmetros, como a Keras Turner (O'Malley et al., 2019) ou as funções *Grid Search* e *Randomized Search* presentes na biblioteca *Scikit-learn*, para otimizar o processo e testar mais variações de hiperparâmetros.

Referências

Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design*.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (icet)* (pp. 1–6).

Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017). Deep learning for hate speech detection in tweets. In *Proceedings of the 26th international conference on world wide web companion* (pp. 759–760).

Camacho-Collados, J., & Pilehvar, M. T. (2017). On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis. *arXiv preprint arXiv:1707.01780*.

Chollet, F., et al. (2015). *Keras*. `https://github.com/fchollet/keras`. GitHub.

Chopra, A., Prashar, A., & Sain, C. (2013). Natural language processing. *International journal of technology enhancements and emerging engineering research*, *1*(4), 131–134.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Retrieved from `http://arxiv.org/abs/1810.04805`

El Naqa, I., & Murphy, M. J. (2015). What is machine learning? In *machine learning in radiation oncology* (pp. 3–11). Springer.

Feldman, R. (2013). Techniques and applications for sentiment analysis. *Communications of the ACM*, *56*(4), 82–89.

Gershenson, C. (2003). Artificial neural networks for beginners. *arXiv preprint cs/0308031*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Heikal, M., Torki, M., & El-Makky, N. (2018). Sentiment analysis of arabic tweets using deep learning. *Procedia Computer Science*, *142*, 114–122.

Ibrahim, M., Torki, M., & El-Makky, N. (2018). Imbalanced toxic comments classification using data augmentation and deep learning. In *2018 17th ieee international conference on machine learning and applications (icmla)* (pp. 875–878).

Kajla, H., Hooda, J., Saini, G., et al. (2020). Classification of online toxic comments using machine learning algorithms. In *2020 4th international conference on intelligent computing and control systems (iciccs)* (pp. 1119–1123).

Kim, P. (2017). Convolutional neural network. In *Matlab deep learning* (pp. 121–147). Springer.

Kim, Y. (2014). *Convolutional neural networks for sentence classification.*

Kouloumpis, E., Wilson, T., & Moore, J. (2011). Twitter sentiment analysis: The good the bad

and the omg! In *Fifth international aaai conference on weblogs and social media.*

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436–444.

Lee, Y., Yoon, S., & Jung, K. (2018). Comparative studies of detecting abusive language on twitter. *arXiv preprint arXiv:1808.10245*.

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, *18*(17), 1-5. Retrieved from `http://jmlr.org/papers/v18/16-365`

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, *5*(1), 1–167.

Liu, P., Qiu, X., & Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*.

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.

Lopez, M. M., & Kalita, J. (2017). Deep learning applied to NLP. *CoRR*, *abs/1703.03091*. Retrieved from `http://arxiv.org/abs/1703.03091`

Neethu, M., & Rajasree, R. (2013). Sentiment analysis in twitter using machine learning techniques. In *2013 fourth international conference on computing, communications and networking technologies (icccnt)* (pp. 1–5).

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. (2019). *Kerastuner.* `https://github.com/keras-team/keras-tuner`.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... others (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, *12*, 2825–2830.

Phung, V. H., Rhee, E. J., et al. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, *9*(21), 4500.

Ramadhani, A. M., & Goo, H. S. (2017). Twitter sentiment analysis using deep learning methods. In *2017 7th international annual engineering seminar (inaes)* (pp. 1–4).

Rätsch, G. (2004). A brief introduction into machine learning. *Friedrich Miescher Laboratory of the Max Planck Society*.

Research, G. (2018). *Google colaboratory.* `https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index`. (Acessado em 2021-08-10)

Severyn, A., & Moschitti, A. (2015). Twitter sentiment analysis with deep convolutional neural networks. In *Proceedings of the 38th international acm sigir conference on research and development in information retrieval* (pp. 959–962).

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE Access*, *7*, 53040–53065.

Torfi, A., Shirvani, R. A., Keneshloo, Y., Tavaf, N., & Fox, E. A. (2021). *Natural language processing advancements by deep learning: A survey.*

Wang, S., Liu, W., Wu, J., Cao, L., Meng, Q., & Kennedy, P. J. (2016). Training deep neural networks on imbalanced data sets. In *2016 international joint conference on neural networks (ijcnn)* (pp. 4368–4374).

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, *31*(7), 1235–1270.

Zhang, L., Wang, S., & Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1253.