UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM CIENCIA DA COMPUTAÇÃO

Jeffson Carneiro Silva Simões

Classificação de Sentimento Utilizando Aprendizagem Profunda

RECIFE

2022

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

CURSO DE BACHARELADO EM CIENCIA DA COMPUTAÇÃO

Jeffson Carneiro Silva Simões

Classificação de Sentimento Utilizando Aprendizagem Profunda

Monografia apresentada ao Centro de Informática (CIN) da Universidade Federal de Pernambuco (UFPE), como requisito parcial para conclusão do Curso de Ciência da Computação, orientada pelo professor Tsang Ing Ren.

RECIFE

2022

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Simoes, Jeffson Carneiro Silva.
Classificação de sentimento utilizando aprendizagem profunda / Jeffson Carneiro Silva Simoes. - Recife, 2022.
59 p. : il., tab.

Orientador(a): Tsang Ing Ren
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2022.

1. Processamento de Linguagem Natural . 2. Analise de sentimentos. 3. Aprendizagem Profunda. I. Ren, Tsang Ing . (Orientação). II. Título.

000 CDD (22.ed.)

AGRADECIMENTOS

Agradeço à Deus por tudo.

A toda minha família pelo apoio, principalmente a minha mãe Marise que sempre esteve ao meu lado, sempre sendo a voz de sabedoria nos momentos mais difíceis.

Ao Professor Tsang Ing Ren que me acompanhou durante o desenvolvimento deste projeto sempre se preocupando em mostrar os melhores passos a serem tomados. Obrigado pela orientação, pelo conhecimento transmitido além da paciência durante o desenvolvimento deste trabalho.

Aos amigos José Alecrim e Geyzianny Sousa pela ajuda e apoio.

A todos os colegas que fiz durante a graduação.

Ao Centro de Informática e todos os seus funcionários que contribuem para que este seja um lugar de excelência.

Por fim, a todos que de forma direta ou indiretamente me apoiaram e fizeram parte dessa incrível jornada.

“The computer is incredibly fast, accurate and stupid. Man is unbelievably slow, inaccurate and brilliant. The marriage of the two is a challenge and opportunity beyond imagination.”

(Stuart G. Walesh)

RESUMO

Hoje em dia, as plataformas de mídia social estão em alta, com milhares de usuários gerando grandes quantidades de dados a todo instante, grande parte desses dados se dá de forma textual expressando suas opiniões sobre diversos temas. Para que empresas possam usar esses dados de forma a gerar valor, foi desenvolvido neste projeto um classificador de sentimentos usando processamento de linguagem natural e aprendizagem profunda. Na criação do modelo foi utilizado um tipo de rede neural recorrente chamado LSTM e uma base de dados do Twitter, disponível na plataforma Kaggle, composta com as mensagens e seus respectivos sentimentos. Utilizando as taxas de perda e acurácia como métricas de avaliação de desempenho, foram realizados alguns experimentos que apresentaram resultados diferentes, tais como erros de overfitting. Por fim atingimos uma taxa de acurácia de 93% no melhor experimento, alcançando assim o objetivo de criar um classificador de sentimentos.

Palavras-chave: Aprendizagem Profunda, Processamento de Linguagem Natural, Analise de sentimentos.

ABSTRACT

Today, social media platforms are booming, with thousands of users generating large amounts of data all the time, much of this data is in textual form expressing their opinions on various topics. In order for companies to be able to use this data in a way that generates value, a sentiment classifier using natural language processing and deep learning was developed in this project. The model was created using a type of recurrent neural network called LSTM and a Twitter database, available on the Kaggle platform, composed of messages and their respective sentiments. Using loss and accuracy rates as performance evaluation metrics, we performed some experiments that presented different results, such as overfitting errors. Finally we reached an accuracy rate of 93% in the best experiment, thus achieving the goal of creating a sentiment classifier.

Keywords: Deep Learning, natural Language Processing, Sentiment Analysis.

Sumário

1 INTRODUÇÃO 21
1.1 Motivação 21
1.2 Objetivos 22
1.3 Estrutura da Monografia 22
2 Fundamentos e Trabalhos Relacionados 23
2.1 Redes Neurais Artificiais 23
2.1.1 Redes Neurais Recorrentes 23
2.2 Processamento de Linguagem Natural – PLN 24
2.3 Classificação dos sentimentos usando PLN com análise de texto 26
2.4 Processamento de Linguagem Natural - Análise Sentimental usando LSTM 26
2.5 Classificação Sequencial com Redes Neurais Recorrentes (LSTM) em Pyhton com Keras 27
3 Classificador de Sentimentos 28
3.1 Tecnologias 28
3.1.1 Linguagem de Programação Python 28
3.1.2 Ambiente de Desenvolvimento – Colab e Kaggle 28
3.1.3 Biblioteca TensorFlow 29
3.1.4. Biblioteca NLTK 29
3.2 Pré-processamento 30
3.2.1 Remoção de Valores Nulos 30
3.2.2 *Label Encoding* 30
3.2.3 Normalização do Tamanho das Palavras 31
3.2.4 Remoção Caracteres Indesejados 31
3.2.5 Remoção *Stopwords* 31
3.2.6 *Stemming* 31
3.2.7 Lematização 31
3.2.8 Vetorização 32
3.2.9 Tokenização 32
3.2.10 *Padding* 32
3.3 Arquitetura 33
3.3.1 Design da Arquitetura 33
3.3.2 Camada de *Embedding* 35
3.3.3 Camada *Batch Normalization* 35
3.3.4 Camada *Dropout* 36

3.3.5 Camada LSTM (Long Short Term Memory) 37
3.3.6 Camada Bidirecional 39
3.3.7 Camada Densa - ReLu 39
3.3.8 Camada Saída – Softmax 40
4 Dados, Análises e Experimentos 42
4.1 Fonte de Dados 42
4.2 Analise Exploratória dos Dados 42
4.2.1 Informações Descritivas 42
4.2.2 Análise Estatísticas 43
4.3 Dados de Treinamento 46
4.4 Experimento 1 - Modelo Inicial 47
4.5 Experimento 2 – Modelo Final 49
4.5.1 Modificações 49
4.5.2 Experimento com Modificações 49
4.6 Baselines 51
5 Conclusões e Trabalhos Futuros 53
5.1 Trabalhos Futuros 53
Bibliografia 54

Lista de Figuras

Figura 1.....24

Figura 2.....25

Figura 3.....33

Figura 4.....34

Figura 5.....38

Figura 6.....44

Figura 7.....44

Figura 8.....45

Figura 9.....45

Figura 10.....46

Figura 11.....46

Figura 12.....48

Figura 13.....50

LISTA DE TABELAS

Tabela 1 – Resultados do Experimento Inicial. 48

Tabela 2 – Resultados do Experimento Final. 50

TABELA DE SIGLAS

Sigla	Significado	Página
RNN	*Recurrent Neural Networks* (Redes Neurais Recorrentes)	24
PLN	Processamento de Linguagem Natural	24
NLP	*Natural Languagen Processing*	24
LSTM	*Lont Short Term Memory*	26
GPU	*Graphic Processing Units*	29
API	*Application Programming Interface*	29
NLTK	*Natural Language ToolKit*	28
TF	*Term Frequency*	51
IDF	*Inverse Document Frequency*	51

1 INTRODUÇÃO

A era da Internet mudou a forma como as pessoas expressam as suas opiniões, agora essa ação é feita principalmente através de postagens em blogs, fóruns online, sites de revisão de produtos, meios sociais de comunicação, etc. Atualmente, milhões de pessoas utilizam as chamadas mídias sociais, plataformas como Facebook, Twitter, etc. para expressar as suas emoções, opiniões e compartilhá-las no dia a dia [1].

Com isso pode-se perceber a massiva quantidade de dados gerados todos os dias através dessas plataformas seja na forma de textos, vídeos, imagens, entre outros. Esses dados por sua vez carregam informações sobre comportamentos e sentimentos humanos que se bem utilizados podem gerar valor das mais diversas formas.

Um dos principais tópicos estudados dentro desse "ambiente" criado por essas plataformas é a análise de sentimentos que ao ser utilizada pode gerar informações sobre como os usuários dessas redes estão se sentindo sobre determinado assunto, empresa, evento etc.

A importância de conhecer o sentimento que as pessoas possuem sobre determinado assunto está ligada a possibilidade de trazer valor e impactar vários setores, seja no setor de negócios, política, ações públicas e entre outros. Tendo em mãos esse tipo de conhecimento as empresas por exemplo podem tomar decisões baseadas em tais informações ajustando assim seu relacionamento com o cliente da melhor maneira possível.

Nos últimos tempos essa tarefa vem sendo feita através de estratégias dentro do campo de inteligência artificial com aprendizado de máquina e/ou aprendizagem profunda. Com os recentes avanços na aprendizagem profunda, a capacidade dos algoritmos de análise de texto melhorou consideravelmente. A utilização criativa de técnicas avançadas de inteligência artificial pode ser uma ferramenta eficaz para fazer investigações no campo da análise de sentimentos mais aprofundadas [2].

1.1 Motivação

Devido as grandes quantidades de dados gerados atualmente que expressam opiniões e sentimentos, existe um desejo das empresas de poderem entender melhor o comportamento de seus clientes. Assim a grande motivação desse trabalho é o de criar uma solução para tal desafio.

1.2 Objetivos

O principal objetivo deste trabalho é o de entender melhor o desafio que é extrair conhecimento de textos fazendo a análise de sentimentos. Assim foi decidido desenvolver um modelo de aprendizagem profunda capaz de realizar a classificação de sentimentos sobre textos relacionados a diferentes entidades buscando a geração de valor, possibilitando que as mesmas possam tomar decisões baseadas em dados.

1.3 Estrutura da Monografia

No capítulo 2 será feito uma revisão dos conceitos das áreas usadas no projeto além de abordarmos alguns projetos que serviram de inspiração para o desenvolvimento da arquitetura de aprendizagem profunda.

No capítulo 3 será apresentado o desenvolvimento do modelo de aprendizagem profunda focando nos pontos principais que são as técnicas de pré-processamento de textos para a preparação dos dados de treinamento e a estrutura da rede neural.

O capítulo 4 abordara o conjunto de dados usados além das análises e experimentos feitos e os resultados alcançados.

Por fim no capitulo 5 será feita as considerações finais do projeto além de expressar possíveis desdobramentos do trabalho no futuro.

2 FUNDAMENTOS E TRABALHOS RELACIONADOS

Nesta seção iremos fazer uma breve revisão de alguns conceitos tais como rede neurais recorrentes e processamento de linguagem natural além de descrever alguns trabalhos que foram usados como inspiração no desenvolvimento do classificador de sentimentos.

2.1 Redes Neurais Artificiais

O conceito de *Deep Learning*, ou aprendizagem profunda, atualmente está bem popular parecendo como se fosse uma nova tecnologia e isto está longe da verdade. A verdade é que o *deep learning* que conhecemos começou por volta do ano 1940 sendo conhecido por vários nomes e apenas aparenta ser novo porque era relativamente impopular em seu início [45]. Alguns dos primeiros algoritmos de aprendizagem foram criados com o intuito de simular o funcionamento biológico de neurônios do cérebro, resultando em um dos nomes de deep learning ser as chamadas redes neurais artificiais [45].

O lado neural da aprendizagem profunda é motivado por duas ideias principais. Uma delas é que o cérebro fornece uma prova pelo exemplo de que o comportamento inteligente é possível, e um caminho para construir a inteligência é a engenharia reversa dos princípios computacionais por trás do cérebro e a duplicação de sua funcionalidade [45]. Outra perspectiva é que seria profundamente interessante compreender o cérebro e os princípios subjacentes à inteligência humana [45].

Sobre os atuais tipos de modelos de aprendizagem de máquina, o termo “deep learning” vai além do cenário neurocientífico, apelando a um conceito de aprendizagem em múltiplos níveis de composição onde pode ser aplicado a estruturas de aprendizagem que não são necessariamente inspiradas pela parte neural [45].

A seguir teremos um subtópico sobre um tipo de rede neural bastante popular e que devido a sua estrutura foi escolhido para ser um dos principais componentes no desenvolvimento desse projeto, são elas as redes neurais recorrentes.

2.1.1 Redes Neurais Recorrentes

As redes neurais recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como som, dados de séries temporais ou linguagem natural [3]. Essas redes são um tipo de rede neural artificial construída para reconhecer padrões em sequências de dados.

A resposta de uma RNN definida em um tempo x-1 irá afetar a decisão no momento posterior x. As redes recorrentes têm duas fontes de entrada, o valor atual e o valor passado recentemente, que juntos irão determinar as novas decisões sobre os novos dados [42].

As redes recorrentes possuem um loop de *feedback* conectado às suas decisões passadas, consumindo suas próprias saídas momento após momento como entrada [42]. Sabendo que existe informações na própria sequência dos dados, foi feito uma adição de memória às redes, afim de que as redes recorrentes a utilizem para entender as informações da sequência e executar as suas tarefas [42].

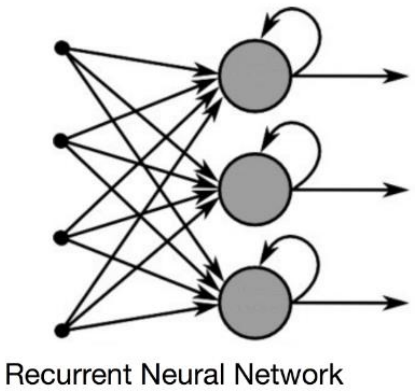

Figura 1. Representação de RNNs [42].

Utilizando um estado oculto para preservar essas informações, as RNN conseguem usar tais informações afim de afetar o processamento dos novos dados, fazendo com que cada processamento seja baseado a partir das informações antigas. Assim podemos dizer que as RNNs conseguem compartilhar informações ao longo do tempo [42].

2.2 Processamento de Linguagem Natural – PLN

O Processamento de Linguagem Natural (PLN) ou “Natural Language Processing (NLP)” é uma vertente da Inteligência Artificial que estuda a capacidade e as limitações de uma máquina em entender a linguagem dos seres humanos. Ou seja, é uma interface entre a linguagem homem-máquina. Dessa forma, o objetivo do PLN é fornecer aos computadores a capacidade de entender e compor textos [6].

A Inteligência Artificial e a aprendizagem máquina mudaram significativamente a forma como interagimos com o mundo. Embora muitas pessoas possam perceber isso, o PLN tornou-se uma parte cotidiana da vida de muitas pessoas. Um exemplo seria as 'assistentes inteligentes' como Siri e Alexa que utilizam o PLN para compreender e interpretar comandos falados [7].

O PLN inclui linguística computacional, estudo por computador, modelagem estatística, e aprendizagem profunda. Compreende o significado da linguagem humana através da análise de uma vasta gama de aspectos, tais como semântica, sintaxe e morfologia. Com a ajuda do PLN, as máquinas são capazes de realizar análises semânticas e emocionais e realizar tarefas de reconhecimento da fala e de resumo de texto. Pode também ser utilizado em serviços de tradução, para fornecer melhores traduções que transmitam não só a tradução literal, mas também manter o significado, subtexto e emoção tanto quanto possível [7].

Um dos principais tópicos quando falamos de PLN é o de análise de sentimentos. A Análise de sentimento é uma técnica usada no PLN para interpretar e classificar os sentimentos agregados a textos como e-mails, pesquisas, mídias sociais e assim por diante [8]. Esse processo automatizado geralmente classifica os sentimentos como positivos, negativos ou neutros.

Figura 2. Exemplificação dos tipos de sentimentos. Imagem adaptada de [9].

Assim, podemos definir a análise de sentimento como um processo que automatiza a mineração de atitudes, opiniões, pontos de vista e emoções a partir de texto, discurso, tweets e fontes de base de dados através do Processamento de Linguagem Natural [1]. A análise de sentimentos envolve a classificação de opiniões em texto em categorias como "positivas" ou "negativas" ou "neutras". É também referida como análise de subjetividade, extração de opinião e extração de avaliação. [1]

2.3 Classificação dos sentimentos usando PLN com análise de texto

Este artigo [10] propôs a criação de um classificador de sentimentos a partir de uma fonte de dados disponível na plataforma Kaggle que consiste em críticas sobre filmes e seus sentimentos relacionados – Positivo ou Negativo. Durante o desenvolvimento é explicado cada passo do *pipeline* para a construção do modelo desde o carregamento dos dados até as métricas usadas para avaliar o desempenho do mesmo. Os pontos principais estão na escolha dos modelos de predição no caso o artigo está usando aprendizagem de máquina – modelo *Naive Bayes* - e na parte do pré-processamento dos dados textuais onde se é usado algumas técnicas para vetorizar os textos, ou seja, mapear as sentenças para vetores numéricos que as representam. Sabendo que o tipo de classificação abordado foi o da classificação binária e as métricas usadas para avaliação dos modelos consistiu basicamente do uso da acurácia, os resultados finais que o artigo apresentou foi uma taxa de acurácia em torno de 98%-95%. Tendo uma taxa tão alta talvez fosse interessante investigar a corretude de fato, afim de verificar se houve algum problema, usando outras métricas.

Durante o artigo é demonstrado a importância de fazer uma boa representação para os textos visto que dependendo de como feito pode interferir na performance do modelo escolhido.

2.4 Processamento de Linguagem Natural - Análise Sentimental usando LSTM

Assim como o artigo anterior o artigo [11] passou por um pipeline parecido. Seus dados também foram provenientes da plataforma Kaggle sendo eles o dataset: Conjunto de dados de revisões de produtos Amazon Alexa. Diferente do anterior foi feita uma abordagem utilizando aprendizagem profunda, mais precisamente usando um tipo de rede neural recorrente chamado LSTM (*Long Short Term Memory*). Ao decorrer de sua explicação sobre as RNNs foi explicado o grande problema que existe em seu uso que é o desaparecimento de gradiente durante a fase de treinamento do modelo. Assim foi escolhido a LSTM, um tipo de RNN, devido a sua arquitetura ser mais robusta sobre esse problema. O tipo de classificação abordado nesse problema foi o de classificação binaria, sua criação foi feita utilizando um conjunto de dados de treino e sua avaliação feita em cima de dados de teste usando métricas como a acurácia para avaliar o modelo, chegando em uma taxa de 90% nos dados de teste. Talvez para comprovar tal taxa e verificar se existe algum problema com o resultado final seria interessante usar um

conjunto de dados para validação durante o treinamento e criar gráficos dos valores de acurácia dos conjuntos de treino e validação durante as épocas treinadas para comprovar os resultados.

2.5 Classificação Sequencial com Redes Neurais Recorrentes (LSTM) em Pyhton com Keras

Neste artigo [12] foi demonstrado vários usos da rede LSTM com o intuito de fazer a classificação de sentimentos dos dados de revisões de filme IMDB. Os dados foram coletados por pesquisadores de Stanford e utilizados num documento de 2011 em uma divisão 50/50 dos dados foi utilizada para treinamento e testes. Durante sua criação os pesquisadores alcançaram uma precisão de 88,89%. Os resultados das várias redes apresentadas neste artigo ficaram com taxas em torno de 85% a 88% bem próximo da taxa conseguida pelos pesquisadores.

Os pontos principais apresentados foram o da utilização de *word embedding* e camadas de *Dropout*. A seguir uma breve definição dos conceitos:

- *Word Embedding*: Esta é uma técnica em que as palavras são codificadas como vetores de valor real num espaço de alta dimensão, onde a semelhança entre palavras em termos de significado se traduz em proximidade no espaço vetorial.
- *Dropout*: As redes neurais recorrentes como a LSTM têm geralmente o problema de *Overfitting*, quando o modelo não consegue aprender e generalizar sobre dados desconhecidos. As camadas de *Dropout* ajudam a diminuir a ocorrência de tais problemas.

3 CLASSIFICADOR DE SENTIMENTOS

Este trabalho tem como o principal objetivo desenvolver uma inteligência artificial utilizando aprendizagem profunda com o intuito de fazer análise de sentimentos. Assim depois da construção do modelo será possível classificar textos em seus diferentes sentimentos possibilitando a tomada de decisões com base nessas informações.

Durante o planejamento desse trabalho foram escolhidas diferentes tecnologias voltadas para os diferentes aspectos do projeto, a seguir será explicado as tecnologias usadas, os estágios de pré-processamento feito sobre os textos usados no treinamento da rede neural e a arquitetura do modelo de aprendizagem profunda construída.

3.1 Tecnologias

Essa seção tem o objetivo de listar e explicar as principais tecnologias que foram escolhidas para o desenvolvimento do projeto tais como a linguagem de programação, o ambiente de desenvolvimento e certas bibliotecas chaves.

3.1.1 Linguagem de Programação Python

Em uma pesquisa com desenvolvedores em 2018 conduzido por StackOverflow revelou que Python era a linguagem de programação mais popular entre os cientistas de dados [13]. A suas popularidades se deu por causa de suas vantagens coincidirem com como projetos de IA são desenvolvidos. Assim primeiro temos de compreender que os projetos de inteligência artificial são diferentes dos projetos de software tradicionais em termos das tecnologias usadas e das competências necessárias. Portanto, escolher uma linguagem de programação que seja estável, flexível, e que tenha um conjunto diversificado de ferramentas é tão importante. Python reúne todas estas [13].

3.1.2 Ambiente de Desenvolvimento – Colab e Kaggle

O Colaboratory ou “Colab” é um produto do *Google Research*, área de pesquisas científicas do Google. O Colab permite que qualquer pessoa escreva e execute código Python

arbitrário pelo navegador e é especialmente adequado para o aprendizado de máquina, análise de dados e educação. Mais tecnicamente, o Colab é um serviço de notebooks hospedados do Jupyter que não requer nenhuma configuração para usar e oferece acesso sem custo financeiro a recursos de computação como GPUs [39].

A escolha de um ambiente de Desenvolvimento em cloud como o Colab se deu pelo fato das limitações das maquinas disponíveis, exemplo a possibilidade de usar GPU durante o treinamento dos modelos de *Deep Learning*.

Uma observação a ser feita é que também foi utilizada a Plataforma de Desenvolvimento do Kaggle, como ambiente auxiliar, quando era atingido o limite diário de uso da GPU no Colab.

3.1.3 Biblioteca TensorFlow

Criado pela equipe do Google Brain e inicialmente lançado ao público em 2015, TensorFlow é uma biblioteca de código aberto para computação numérica e aprendizagem de máquinas em grande escala. TensorFlow reúne uma série de modelos e algoritmos de aprendizagem de máquinas e de aprendizagem profunda (também conhecidos como redes neurais) e torna-os úteis por meio de metáforas programáticas comuns. Utiliza Python ou JavaScript para fornecer uma API conveniente para construir aplicações, enquanto executa essas aplicações em C++ de alto desempenho [14].

O maior benefício que TensorFlow proporciona para o desenvolvimento da aprendizagem de máquinas são a abstração. Em vez de lidar com os detalhes da implementação de algoritmos, ou de descobrir formas adequadas de ligar a saída de uma função à entrada de outra, o programador pode concentrar-se na lógica geral da aplicação [14].

3.1.4. Biblioteca NLTK

NLTK é uma das principais bibliotecas para a construção de projetos Python para trabalhar com dados da linguagem humana. Nos fornece de interfaces simples de utilização a mais de 50 corpora e ativos lexicais como o WordNet, juntamente com uma configuração de bibliotecas de pré-processamento de texto para etiquetagem, análise, classificação, stemming, tokenização e elementos de raciocínio semântico para bibliotecas de Processamento de Linguagem Natural (PLN) e uma discussão ativa de conversação. NLTK é acessível para

Windows, Mac OS e Linux. A melhor parte é que a NLTK é um empreendimento livre, de código aberto e local [15].

3.2 Pré-processamento

O pré-processamento de dados é um passo essencial na construção de um modelo de aprendizagem profunda e dependendo de quão bem os dados tenham sido processados, os resultados podem apresentar uma performance diferente [16].

Quando normalizamos o texto, tentamos reduzir a sua aleatoriedade, aproximando-o de um "padrão" pré-definido. Isto ajuda-nos a reduzir a quantidade de informação diferente com que o computador tem de lidar, e por conseguinte melhora a eficiência. O objetivo das técnicas de normalização como o stemming e a lematização é reduzir as formas inflexíveis e, por vezes, as formas derivadas de uma palavra a uma forma base comum [17].

Assim a seguir iremos introduzir algumas das técnicas usadas para o processamento dos textos utilizadas na construção do classificador.

3.2.1 Remoção de Valores Nulos

Os valores nulos são um grande problema quanto ao uso de aprendizagem de máquinas e na aprendizagem profunda. Ao utilizar o TensorFlow ou outros pacotes de aprendizagem profunda, é necessário fazer um tratamento dos valores nulos antes de passar os seus dados para o treinamento do modelo [40].

3.2.2 *Label Encoding*

A codificação de etiquetas ou *Label Encoding* refere-se à conversão das etiquetas em uma forma numérica, de modo a convertê-las na forma legível por máquina. Os algoritmos de aprendizagem da máquina podem então decidir de uma forma melhor como essas etiquetas devem ser operadas. É uma etapa importante de pré-processamento para o conjunto de dados estruturados na aprendizagem supervisionada [18].

3.2.3 Normalização do Tamanho das Palavras

Um dos passos que se podemos fazer sobre as palavras é transformar todos os caracteres das palavras para minúsculo garantindo assim que palavras iguais com letras minúsculas ou maiúsculas representem a mesma palavra durante a representação das mensagens [41].

3.2.4 Remoção Caracteres Indesejados

Alguns caracteres indesejados podem ser: pontuações, caracteres especiais, caracteres repetidos de uma palavra e até mesmo números encontrados dentro de textos. Assim fazer a remoção de tais caracteres pode ajudar na hora de criar a representação dos textos [41].

3.2.5 Remoção *Stopwords*

Uma das principais formas de pré-processamento é a filtragem de dados inúteis. No processamento em linguagem natural, palavras inúteis (dados), são referidas como *stopwords* [19].

3.2.6 *Stemming*

O *Stemming* é o processo de reduzir as palavras à sua forma de “caule” ou “raiz”. O objetivo do stemming é reduzir as palavras relacionadas para o mesmo caule/raiz, mesmo que o “caule” não seja uma palavra de dicionário. Por exemplo, as palavras “*connection*”, “*connecting*”, “*connected*” são reduzidas a uma palavra comum "*connect*" [17].

Stemming refere-se a um processo heurístico grosseiro que corta as pontas das palavras na esperança de alcançar este objetivo corretamente a maior parte do tempo. Porem as vezes pode resultar em palavras que não são palavras reais [17].

3.2.7 Lematização

A lematização reduz as palavras à sua palavra de base, reduzindo adequadamente as palavras flexionadas e assegurando que a palavra de raiz pertence à língua. É normalmente mais sofisticado do que o stemming, uma vez que o stemmer trabalha em uma palavra individual sem conhecimento do contexto. Na lematização, uma palavra de raiz é chamada lemma. Um lema é a forma canónica, forma de dicionário, ou forma de citação de um conjunto de palavras [17].

3.2.8 Vetorização

Vetorização é o jargão para uma abordagem clássica de conversão de dados de entrada do seu formato bruto (ou seja, texto) em vetores de números reais, que é o formato que os modelos de aprendizagem profunda suportam. Esta abordagem existe desde que os computadores foram construídos pela primeira vez, tem funcionado maravilhosamente em vários domínios, e é agora utilizada em PLN [20].

3.2.9 Tokenização

Em Processamento de Linguagem Natural, a tokenização significa quebrar o texto bruto em unidades únicas (também conhecidas como tokens). Um token pode ser sentenças, frases ou palavras. Cada token tem uma token-id único. O objetivo da tokenização é que podemos utilizar esses tokens (ou os token_ids) para representar o texto original [21].

3.2.10 *Padding*

Depois de feita a vetorização/tokenização dos textos teremos vários vetores de tamanhos diferentes. Porem a maioria (se não todas) das redes neurais requer a sequência de entrada de dados com o mesmo comprimento, e é por isso que precisamos de acolchoamento (*padding*): para truncar ou adicionar sequência de padding (normalmente completar com 0s) no mesmo comprimento [21].

Assim depois de realizar o *padding* teremos sequencias de mesmo comprimento prontas para serem passadas para a rede neural. A seguir uma imagem que exemplifica o processo de *padding*.

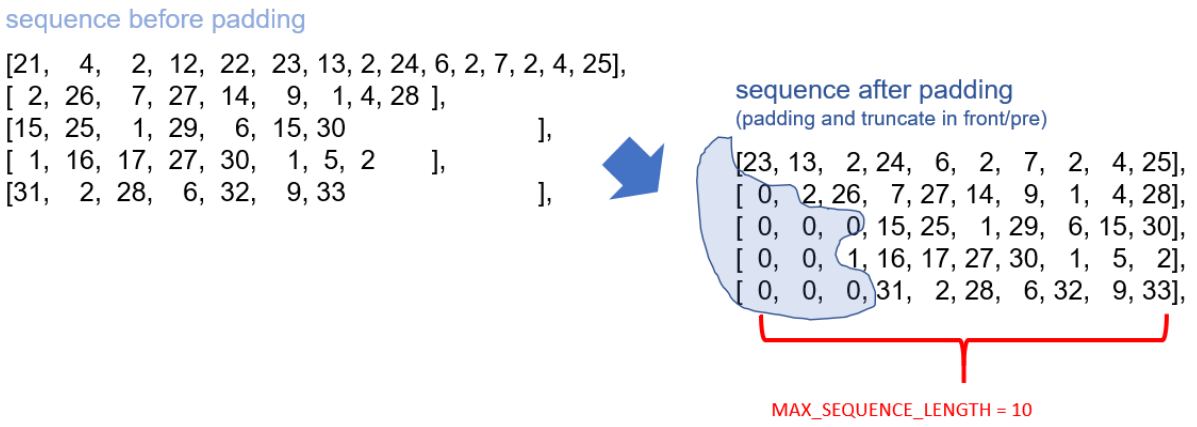

Figura 3. *Padding* do tipo pré onde se acrescenta os zeros no começo do vetor. Imagem adaptada de [21].

3.3 Arquitetura

O Classificador foi idealizado com base nos artigos do capitulo 2 como referência. Ele apresenta 8 camadas. A arquitetura é composta de camadas de *embedding*, *batch normalization*, bidirecional, *lstm*, *Dropout*, densa e uma camada final com função de ativação softmax responsável pela classificação.

A seguir o design completo da rede neural construída e a explicação do que cada camada utilizada faz e sua importância.

3.3.1 Design da Arquitetura

A arquitetura da rede foi inspirada nos exemplos usados no artigo comentado na seção 2.5, principalmente no exemplo onde a camada Bidirecional-*LSTM* é usada [12]. Apesar da inspiração a rede sofreu algumas alterações a fim de ser utilizada com os dados usados neste trabalho.

Quando se trabalha com textos, uma técnica bastante popular é o chamado *word embedding* onde criamos uma forma de representar palavras em um espaço vetorial de ‘n’ dimensões, n é um parâmetro definido previamente, onde a semelhança entre as palavras em termos de significado se traduz no espaço vetorial [12]. Assim utilizamos como camada de entrada a camada de *embedding* que fica responsável por definir tais representações. Ao longo da rede foram adicionadas camadas com o intuito de ajudar no aprendizado da rede tentando evitar erros de *overfitting*, foram elas as camadas de *Batch Normalization* e *Dropout* [44]. A

camada *LSTM* foi escolhida por ser uma variação das RNN's que trabalham bem com dados sequenciais e por sua estrutura ser um pouco diferente das RNN's comuns, sendo seu funcionamento descrito na seção 3.3.4. As vezes a ordem da sequência pode influenciar nos resultados onde a sequência em ordem pode não traz bons resultados, as vezes a ordem inversa da mesma pode ser melhor e pode ter situações que ambas sozinhas não tenham o resultado esperado. Assim foi decidido usar a camada bidirecional para avaliarmos as sequencias em ordem e na ordem reversa. Ao final foi utilizado duas camadas totalmente conectadas com funções de ativação *ReLu* e *softmax* (camada de saída). A partir dos dados que usamos temos um problema de classificação multiclasse e por isso foi escolhido usar a função *softmax* na última camada.

A seguir a imagem do design final da arquitetura construída, demonstrando as camadas utilizadas na construção do modelo, desde as camadas de entrada até a última camada responsável pela classificação.

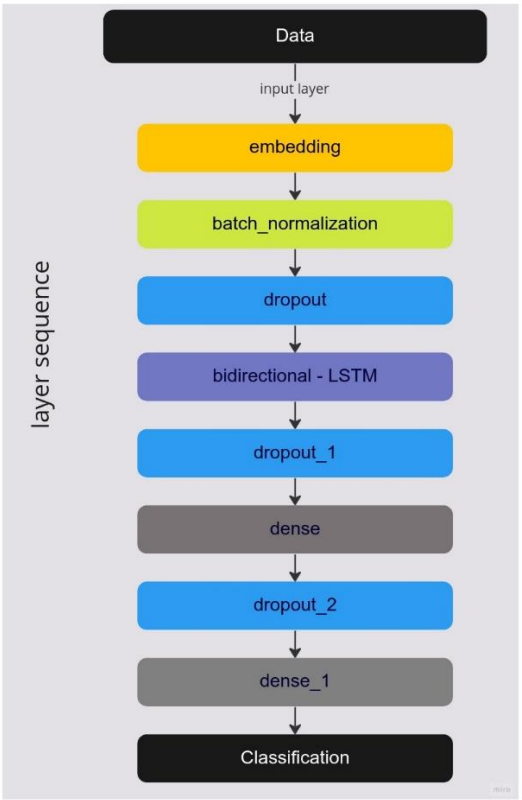

Figura 4. Camadas da rede neural – *embedding*, *batch Normalization*, bidirecional-*lstm*, *Dropout*, *dense_1 e dense_2*. Imagem demonstra fluxo percorrido pelos dados até a classificação do mesmo nas categorias definidas, a classificação se dá através da função *softmax* aplicada na camada *dense_1* [35].

3.3.2 Camada de *Embedding*

A biblioteca Tensorflow oferece uma camada *Embedding* que pode ser usada para redes neurais em dados de texto. Esta camada requer que os dados de entrada sejam codificados para valores numéricos inteiros, de modo a que cada palavra seja representada por um inteiro único. Esta etapa de preparação de dados foi executada utilizando o *Tokenizer API* também fornecido com Tensorflow [22].

A camada de embedding é inicializada com pesos aleatórios e aprenderá uma representação para todas as palavras no conjunto de dados de treino. Ela é uma camada flexível que pode ser utilizada de várias maneiras, como por exemplo:

- Pode ser utilizado sozinho para aprender uma embedding de palavras que pode ser guardada e utilizada posteriormente em outro modelo [22].
- Pode ser usado como parte de um modelo de aprendizagem profunda onde o embedding será aprendido juntamente com o próprio modelo [22].
- Pode ser utilizado para carregar um conjunto de embedding pré-treinados em outro modelo, um tipo de aprendizagem de transferência [22].

A camada de embedding é definida como a primeira camada oculta de uma rede. Deve especificar 3 argumentos:

- *input_dim*: Este é o tamanho do vocabulário nos dados do texto. Por exemplo, se os dados estiverem codificados com valores entre 0-10, então o tamanho do vocabulário seria de 11 palavras.
- *output_dim*: Este é o tamanho do espaço vetorial no qual as palavras serão incorporadas. Define o tamanho dos vetores de saída a partir desta camada para cada palavra. Por exemplo, poderia ser 32 ou 100 ou até maior.
- *input_length*: Este é o comprimento das sequências de entrada, como definiria para qualquer camada de entrada de um modelo Keras. Por exemplo, se todos os seus documentos de entrada forem compostos por 1000 palavras, isto seria 1000.

3.3.3 Camada *Batch Normalization*

O treino de redes neurais profundas, por exemplo, redes com dezenas de camadas ocultas, é um desafio. Um aspecto deste desafio é que o modelo é atualizado camada por camada para trás desde a saída até à entrada, utilizando uma estimativa de erro que assume que os pesos das camadas antes da camada atual são fixos. Como todas as camadas são alteradas durante uma atualização, o procedimento de atualização está sempre a perseguir um alvo em movimento. Por exemplo, os pesos de uma camada são atualizados dada uma expectativa de que os valores de saída da camada anterior com uma dada distribuição. Esta distribuição é provavelmente alterada depois de os pesos da camada anterior serem atualizados [23].

O treino de Redes Neurais Profundas é complicado pelo fato de a distribuição dos *inputs* de cada camada mudar durante o treino, uma vez que os parâmetros das camadas anteriores mudam. Isto atrasa o treino, exigindo taxas de aprendizagem mais baixas e uma inicialização cuidadosa dos parâmetros, e torna notoriamente difícil treinar modelos com não-linearidades saturantes [24].

A normalização de lotes é proposta como uma técnica para ajudar a coordenar a atualização de múltiplas camadas no modelo. Faz esta normalização da saída da camada, especificamente padronizando as ativações de cada variável de entrada por mini batch, tais como as ativações de um nó da camada anterior. Recordar que a padronização se refere ao redimensionamento dos dados para ter uma média de zero e um desvio padrão de um, por exemplo, um gaussiano padrão. A padronização das ativações da camada anterior significa que as suposições que a camada seguinte faz sobre a dispersão e distribuição dos inputs durante a atualização de peso não se alterarão, pelo menos não dramaticamente. Isto tem o efeito de estabilizar e acelerar o processo de treino de redes neurais profundas. A normalização das entradas para a camada tem um efeito no treinamento do modelo, reduzindo drasticamente o número de épocas necessárias. Pode também ter um efeito regularizador, reduzindo o erro de generalização, tal como a utilização da regularização de ativação [23].

3.3.4 Camada *Dropout*

Dropout é uma técnica de regularização para modelos de redes neurais proposta por Srivastava et al. no seu documento de 2014 "*Dropout: A Simple Way to Prevent Neural Networks from Overfitting*" [25].

Dropout é uma técnica em que neurónios selecionados aleatoriamente são ignorados durante o treino. Eles são "abandonados" aleatoriamente. Isto significa que a sua contribuição

para a ativação dos neurónios é temporariamente removida no passe para a frente, e quaisquer atualizações de peso não são aplicadas ao neurónio no passe para trás. Quando uma rede neural aprende, os pesos dos neurónios fixam-se no seu contexto dentro da rede. Os pesos dos neurónios são afinados para características específicas, proporcionando alguma especialização. Os neurónios vizinhos confiam nesta especialização, que, se levada demasiado longe, pode resultar num modelo frágil e demasiado especializado para os dados de treino. Esta dependência do contexto para um neurónio durante o treino é referida como coadaptarão complexa. Pode-se imaginar que se os neurónios forem abandonados aleatoriamente da rede durante o treino, outros neurónios terão de intervir e lidar com a representação necessária para fazer previsões para os neurónios em falta. Acredita-se que isto resulte em múltiplas representações internas independentes a serem aprendidas pela rede [25][26].

O efeito é que a rede se torna menos sensível aos pesos específicos dos neurónios. Isto, por sua vez, resulta numa rede capaz de uma melhor generalização e menos susceptível de sobreajustar (*Overfitting*) os dados de treino [26].

3.3.5 Camada LSTM (Long Short Term Memory)

A LSTM é uma variação das RNNs e é usada em diversos cenários de processamento de linguagem natural, a mesma consegue lembrar valores em intervalos aleatórios [27].

A rede LSTM consegue recordar por causa de sua estrutura de células, a mesma possui o que podemos chamar de “células LSTM” onde tais células apresentam uma recorrência interna além da recorrência externas da RNN [45]. Cada célula tem as mesmas entradas e saídas de uma rede recorrente comum, mas também possui mais parâmetros e um sistema de portões que controla o fluxo de informação. [45].

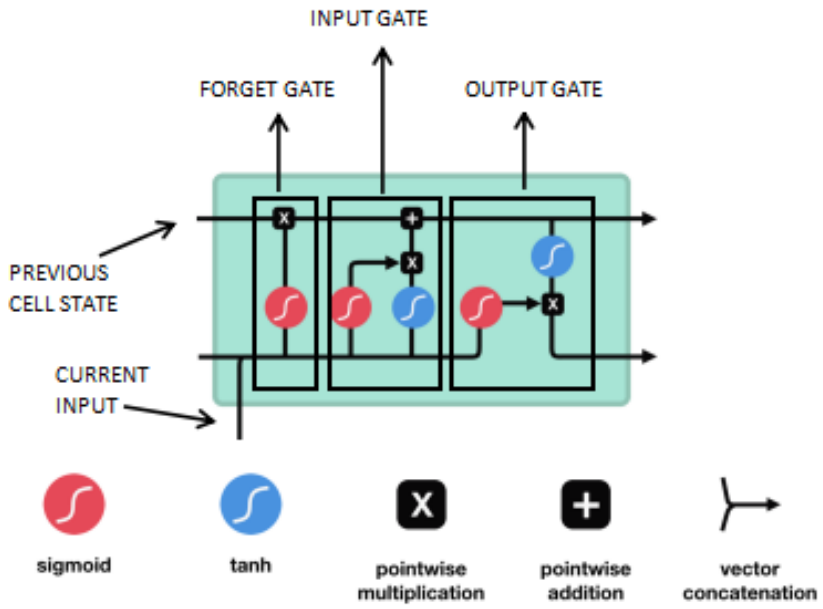

Figura 5. Célula LSTM e suas operações [47].

As informações são retidas pelas células e as manipulações, saber o que deve ser guardado ou esquecido são feitos pelos portões. Existem três portões:

Forget Gate: Este portão decide que informação deve ser esquecida ou guardada. A informação do estado oculto anterior e a informação da entrada atual é passada através da função sigmoid. A função mapeia os valores para o intervalo de 0 a 1, assim, quanto mais próximo de zero mais próximo de ser esquecido e quanto mais próximo de 1 mais próximo de ser mantido [46].

Input Gate: Para atualizar o estado da célula, temos o portão de entrada. Onde primeiro, passamos o estado anterior oculto e a entrada atual para uma função *sigmoid*, isso decidira quais valores serão atualizados, transformando os valores entre 0 e 1 – 0 menos importante e 1 mais importante. Também se passa o estado oculto e a entrada atual para a função *tanh* que por sua vez irá mapear os valores entre -1 e 1, para ajudar a regular a rede. Depois multiplica-se a saída da função *tanh* com a saída da *sigmoid*. A saída *sigmoid* decidirá qual a informação importante a manter da saída *tanh* [46].

Output Gate: O portão de saída decide qual deve ser o próximo estado oculto. Devemos lembrar de que o estado oculto contém informações sobre as entradas anteriores além de também ser utilizado para previsões. Assim, inicialmente, passamos o estado oculto anterior e a entrada atual para uma função *sigmoid*. Depois, passamos o estado da célula recentemente modificada para a função *tanh*. Multiplicamos a saída *tanh* com a saída *sigmoid* para decidir que informação o estado oculto deve conter. A saída é o estado oculto.

Assim podemos concluir que, o forget gate decide o que é relevante para a manter das etapas anteriores. o input gate decide que informação é relevante a acrescentar a partir da etapa atual. E por fim, o output gate determina qual deve ser o próximo estado oculto.

3.3.6 Camada Bidirecional

Maximiza a sensibilidade da ordem dos RNNs: consiste essencialmente em duas RNNs (LSTMs) que processam a sequência de entrada numa direção diferente para finalmente fundir as representações. Ao fazer isto, são capazes de capturar padrões mais complexos do que uma única camada de RNN capturaria. Por outras palavras, uma das camadas interpreta as sequências na ordem original e a segunda a sequência inversa, razão pela qual os RNNs bidirecionais são amplamente utilizados, porque oferecem maior desempenho do que os RNNs regulares [28].

3.3.7 Camada Densa - ReLu

Depois de passar pela camada Bidirecional LSTM e camadas de Dropout, antes da camada de saída temos uma camada que utiliza a função de ativação ReLu em seu processamento. A função ReLu é a unidade linear retificada. É definida pela fórmula: **f (x) = max (0, x)** [29].

Atualmente a função de ativação mais utilizada nas camadas escondidas é a ReLu, basicamente ela se tornou a função de ativação padrão para muitos tipos de redes neurais porque um modelo que o utiliza é mais fácil de treinar e geralmente alcança um melhor desempenho [48].

Dado os valores passados para a função, ela retornará zero para quaisquer valores negativos e o próprio valor para valores positivos [48]. É uma função computacionalmente leve. Uma das principais vantagens de se usar a ReLu é por causa de sua representação esparsa onde dado que os dados de entrada sejam valores negativos os mesmos serão convertidos para zero, consequentemente os neurônios que receberem tais valores não serão ativados fazendo com que apenas alguns sejam ativados isso pode acelerar a aprendizagem e simplificar o modelo [48][29].

3.3.8 Camada Saída – Softmax

A camada de saída é a camada responsável pela predição da rede neural. Nela escolheremos a função de ativação dependendo do tipo de problema que estamos solucionando [43]. No caso deste trabalho foi utilizado a função softmax.

A função softmax é útil quando enfrentamos problemas de classificação. Ela transforma as saídas para cada classe para valores entre 0 e 1 dividindo esses valores pela soma das saídas, nos dando assim a probabilidade de a entrada estar em uma determinada saída [29].

A definição matemática da função softmax é dada por: Onde todos os valores zi são os elementos do vetor de entrada e podem assumir qualquer valor real. O termo no fundo da fórmula a seguir é o termo de normalização que assegura que todos os valores de saída da função se somarão a 1, constituindo assim uma distribuição de probabilidade válida [31].

A fórmula da função softmax pode ser definida da seguinte maneira [31]:

$$O\left(\underset{z}{\rightarrow}\right)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Sendo:

$\underset{z}{\rightarrow}$ = O vetor de entrada para a função softmax, constituído por (z0, ... zK) [31].

z_i = Todos os valores zi são os elementos do vetor de entrada para a função softmax, e podem tomar qualquer valor real, positivo, zero ou negativo. Por exemplo, uma rede neural poderia ter um vetor de saída como (-0,62, 8,12, 2,53), o que não é uma distribuição de probabilidade válida, daí a razão pela qual a softmax seria necessária [31].

e^{z_i} = A função exponencial padrão é aplicada a cada elemento do vetor de entrada. Isto dá um valor positivo acima de 0, que será muito pequeno se o input for negativo, e muito grande se o input for grande. No entanto, ainda não está fixado no intervalo (0, 1), que é o que é exigido de uma probabilidade [31].

$\sum_{j=1}^{k} e^{z_i}$ = O termo na parte inferior da fórmula é o termo de normalização. Assegura que todos os valores de saída da função se somarão a 1 e cada um estará no intervalo (0, 1), constituindo assim uma distribuição de probabilidade válida [31].

K = O número de classes no classificador multiclasse [31].

Assim digamos, por exemplo, que temos as saídas como [1.2, 0.9, 0.75], quando aplicamos a função softmax, obteríamos [0.42, 0.31, 0.27]. Agora podemos usá-los como probabilidades de que o valor seja de cada classe [29].

A função softmax é idealmente usada na camada de saída de um classificador, onde estamos tentando gerar as probabilidades para definir a classe de cada entrada [29].

4 DADOS, ANÁLISES E EXPERIMENTOS

Nesta seção será apresentado os dados utilizados na construção do classificador, as análises feitas sobre os mesmos e os resultados dos testes realizados com o modelo além das comparações com alguns projetos que usaram a mesma base de dados.

4.1 Fonte de Dados

Durante a busca dos dados a serem utilizados no desenvolvimento do projeto decidiu-se utilizar um dataset disponível na Plataforma Kaggle. O conjunto de dados utilizado para construção do modelo de Deep Learning foi o dataset: *Twitter Sentiment Analysis* [32].

Este é um conjunto de dados de análise de sentimentos a nível de entidade do twitter, focado na linguagem inglesa. Dada uma mensagem e uma entidade, a tarefa consiste em julgar o sentimento da mensagem sobre a entidade. Há quatro classes neste conjunto de dados: Positivo, Negativo, Neutro e Irrelevante [32].

Assim esse dataset foi utilizada na construção do classificador de sentimentos.

4.2 Analise Exploratória dos Dados

Durante o ciclo de vida de projetos de aprendizagem profundo não há atalhos. Não podemos simplesmente saltar para a fase de construção do modelo após a escolha dos dados. Precisamos planejar a nossa abordagem de uma forma estruturada e a fase de análise exploratória de dados desempenha um enorme papel nesse sentido [33].

Existem várias técnicas para realizar a exploração e os próximos subtópicos explicaram os passos que foram utilizados nesse projeto.

4.2.1 Informações Descritivas

Neste passo foi feita uma verificação simples dos dados, onde foi observado as seguintes informações:

- Dados existentes no *dataset* (colunas):

- Tweet ID: id do *tweet* (tipo numérico inteiro).
- *Entity*: entidade (exemplo: empresa Apple) ao qual o texto está relacionado (tipo objeto string).
- Sentiment: Os dados vieram etiquetados com os seguintes sentimentos: Positivo, Neutro, Negativo e Irrelevante, o ultimo sentimento são mensagens que não tem muita relevância para com a entidade ao qual este foi associado (tipo objeto string).
- Tweet Content: corpo textual - mensagem/tweet - (tipo objeto string).

- Quantidade de observações nos datasets (treino e teste):
  - Treino: dataset de treino possui cerca de 74682 observações com as 4 colunas já descritas acima.
  - Teste: dataset de treino possui cerca de 1000 observações com as 4 colunas já descritas acima.

- Entidades: Foi verificado a existência de 32 entidades distintas entre elas estão: Microsoft, Nvidia, Amazon, Facebook entre outras. As mensagens que existem no *dataset* estão relacionadas a essas entidades, ou seja, essas mensagens trazem consigo os sentimentos relacionados as entidades aqui citadas.

- Verificação de dados nulos
  - Dados de treino: foi observado a existência de 686 recordes com campos nulos.
  - Dados de teste: não foi encontrado nenhum recorde com campo nulo.

4.2.2 Análise Estatísticas

As visualizações de estatísticas de texto são técnicas simples, mas muito perspicazes. A seguir os passos tomados neste estágio [34].

- Número de caracteres presentes em cada mensagem: o gráfico a seguir mostra que a quantidade de caracteres das mensagens varia do tamanho mínimo zero a um valor próximo de 400, sendo sua grande porcentagem em mensagens possuindo menos de 200 caracteres.

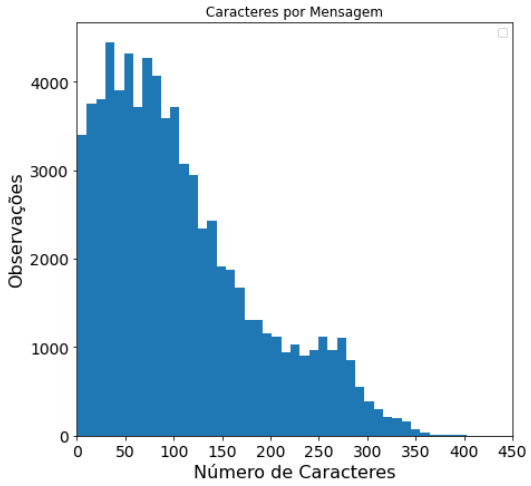

Figura 6. Quantidade de observações em relação ao número de caracteres por mensagens**.** Imagem adaptada de [35].

- Número de palavras que aparecem em cada mensagem: o gráfico a seguir mostra a quantidade de palavras dentro das mensagens e a quantidade de observações de tal número. Assim foi verificado que as mensagens possuem em sua maioria uma quantidade menor do que 40 palavras por mensagens.

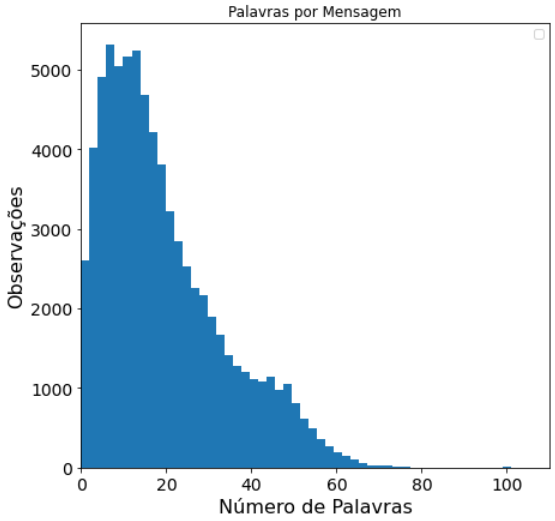

Figura 7. Quantidade de observações em relação ao número de palavras por mensagens. Imagem adaptada de [35].

- Foi verificado o comprimento médio das palavras por mensagem. Podemos verificar que o comprimento médio das palavras em cada mensagem é menor que 10 mas isso pode ter ocorrido devido a existencia das muitas *stopwords*. *Stopwords* é a definição

dada as palavras mais comumentes usadas na lingua, essas palavras são pequenas em sua maioria, podendo ser esse o motivo da média de comprimento ser baixa [34].

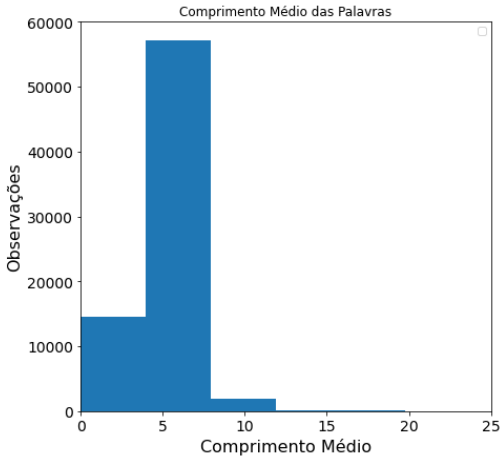

Figura 8. Quantidade de Observações em relação ao comprimento médio das palavras por mensagem**.** Imagem adaptada de [35].

- Verificação de *Stopwords*: abaixo segue um gráfico que mostra as *stopwords* mais frequentes no conjunto de treinamento.

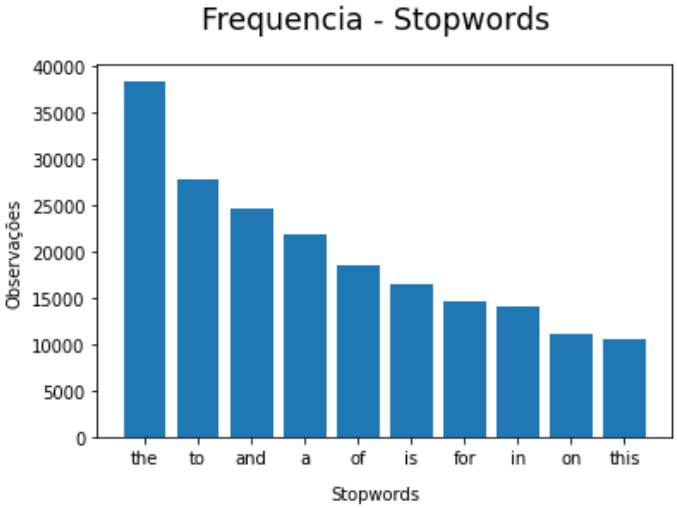

Figura 9. *Stopwords* mais frequentes no conjunto de dados de treino**.** Imagem adaptada de [35].

- Verificar as palavras frequentes: abaixo segue um gráfico que mostra as palavras/tokens mais frequentes em nosso conjunto de dados. Dá para perceber que nas mensagens existem muitos caracteres especiais, números e pontuações que podem vir a prejudicar na hora que formos fazer a representação dessas mensagens para a criação do modelo.

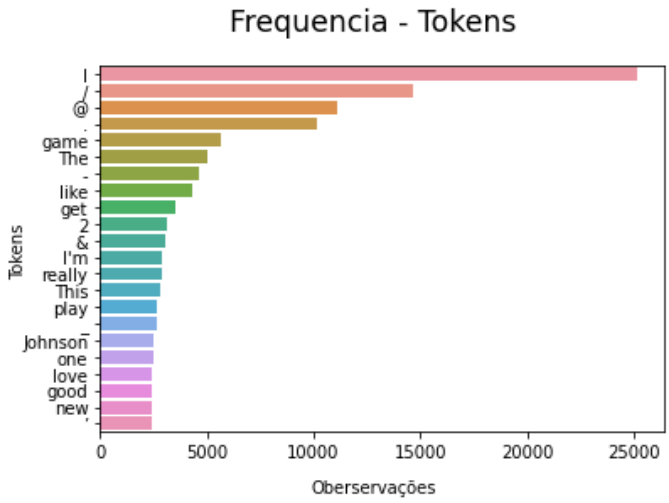

Figura 10**.** Palavras (tokens) mais frequentes no conjunto de dados Imagem adaptada de [35].

- Foi verificado também a distribuição da quantidade de mensagens por sentimento no dataset. Podemos perceber que os dados estão um pouco desbalanceados principalmente em relação a classe irrelevante, isso dever ser levado em consideração na hora da modelagem do classificador e no momento que for feita a avaliação das métricas do mesmo.

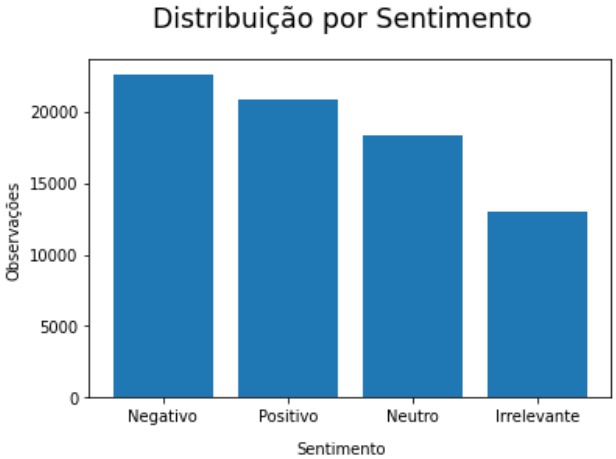

Figura 11. Quantidade de Observações por sentimento. Imagem adaptada de [35].

4.3 Dados de Treinamento

Depois de ter concluído a fase exploratória dos dados foi feito a preparação dos mesmos para o treinamento e teste, desde a divisão dos dados até o final do pré-processamento dos textos afim de gerar uma boa representação dos textos para passa-los na entrada da rede construída. Os passos feitos para a preparação dos dados foram:

1. Conjunto de dados de treinamento dividido com uma taxa de 70% treinamento e 30% validação.
2. Todos os passos de pré-processamento a seguir foram aplicados nos conjuntos de dados de treino e validação.
3. Remoção dos valores Nulos do conjunto de dados descrito na seção pré-processamento 3.2.
4. Encodificação das classes descrito na seção pré-processamento 3.2.
5. Normalização do tamanho das palavras descrito na seção pré-processamento 3.2.
6. Remoção de caracteres indesejados descrito na seção pré-processamento 3.2.
7. Remoção das Stopwords descrito na seção pré-processamento 3.2.
8. Utilizado a técnica de lematização descrito na seção pré-processamento 3.2.
9. Utilizada a técnica de tokenização descrito na seção pré-processamento 3.2.
10. Vetorização das Mensagens descrita na seção pré-processamento 3.2.
11. Utilizado as técnicas de padding descrito na seção pré-processamento 3.2.

Depois de passar por todos esses passos, definimos a camada de *input* da rede como uma camada de *embedding* descrito na seção 3.3.2 com o intuito de fazer com que nossa rede aprendesse uma boa representação (*word embedding*) do conjunto de palavras passados ao mesmo tempo que a rede seria treinada.

4.4 Experimento 1 - Modelo Inicial

Depois que os textos dos conjuntos de treino e validação passaram pela etapa de preparação de dados descrita na seção 4.3, foi realizado os experimentos com o modelo inicial.

Realizamos o treinamento do modelo passando os dados de treino e validação para o método *". fit"* da rede, esse treinamento rodou durante 50 épocas para a construção do modelo e foi utilizado as métricas de acurácia e a função de perda *"sparse categorical crossentropy"* para a avaliação do mesmo, seu treinamento levou em média 1 hora de treinamento no Colab,

ou Kaggle, usando a GPU para o processamento. A seguir uma tabela e gráfico com os resultados obtidos neste experimento.

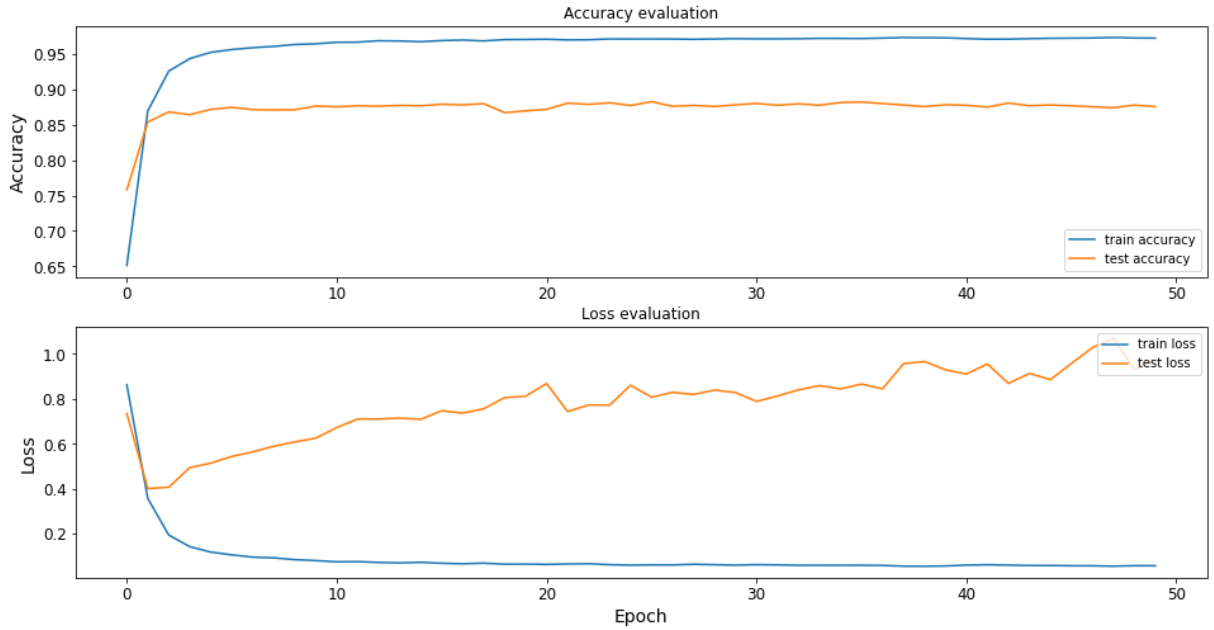

Figura 12**.** Taxas de perda e acurácia do modelo inicial ao longo das 50 épocas de treinamento. Imagem adaptada de [35].

	Taxa de Erro	**Taxa de Acurácia**
Treino	0.052	97%
Validação	0.978	87%
Teste	0.538	94%

Tabela 1. Resultados das Métricas, taxa de erro (*sparse categorical crossentropy*) e acurácia dos testes do modelo inicial.

Depois de analisar os resultados obtidos tanto na tabela 1 quanto na figura 12, foi observado alguns comportamentos estranhos, um deles seria que a taxa de acurácia no conjunto de testes 94% está próxima da taxa obtida no conjunto de treino 97%, porém a taxa de erro 0.538 do conjunto de teste está mais próxima da obtida no conjunto de validação 0.978 e ao observar a figura 13, verificamos que a curva da taxa de erro calculado pela função de perda *"sparse categorical crossentropy"* no conjunto de validação tem um comportamento não esperado, aonde em determinado ponto a taxa de erro começa a aumentar sendo um indicativo da possível ocorrência de erros como *overfitting* [38]. Assim foi decido fazer algumas modificações na rede afim de evitar tais erros.

4.5 Experimento 2 – Modelo Final

Antes de fazer o experimento foram feitas as modificações que achamos mais adequadas para evitar os erros apresentados no experimento inicial descrito na seção 4.4. Assim esta seção será dividida em uma parte para explicar as modificações e a outra para explicar o novo experimento e seus resultados.

4.5.1 Modificações

Devido ao problema encontrado no experimento inicial da seção 4.4 foi planejado acrescentar algumas estrategias para evitar o *overfitting* da rede. As estrategias adotadas para evitar o *overfitting* foram:

- **Diminuir a complexidade da rede**: foi diminuído o número de neurônios usados ao longo da rede.
- **Dropout**: foi adicionado uma camada extra Dropout antes da camada LSTM.
- **Regularização** [37]: foi adicionado funções de regularização na camada Bidirecional LSTM e na camada densa posterior. Essas funções adicionam uma penalidade a função de perda, penalizando pesos grandes. A função que teve um melhor desempenho foi a função L1L2 que combina a robustez e flexibilidade das funções L1 e L2. Os valores utilizados para cada uma foram feitos com uma busca de tentativa e erro tendo os valores usados expressado o melhor resultado.
    - L1 A regularização, também chamada de regressão lasso, adiciona o "valor absoluto de magnitude" do coeficiente como um termo de penalização à função de perda [36].
    - L2 A regularização, também chamada regressão da ridge, adiciona o "valor da magnitude ao quadrado" do coeficiente como termo de penalização à função de perda [36].

O regularizador que foi utilizado chama-se L1L2 disponível pelo Keras, o mesmo faz uma aplicação das penalidades de ambas funções nas camadas da rede que são utilizadas.

4.5.2 Experimento com Modificações

Depois que os textos dos conjuntos de treino e validação passaram pela etapa de preparação de dados descrita na seção 4.3 e feito as modificações na rede inicial descritos na seção 4.5.1 foi realizado os experimentos com o novo modelo.

Realizamos o treinamento do modelo passando os dados de treino e validação para o método *". fit"* da rede, esse treinamento rodou durante 50 épocas para a construção do modelo e foi utilizado as métricas de acurácia e a função de perda *"sparse categorical crossentropy"* para a avaliação do mesmo, seu treinamento levou em média 1 hora de treinamento no Colab, ou Kaggle, usando a GPU para o processamento. A seguir uma tabela e gráfico com os resultados obtidos neste experimento.

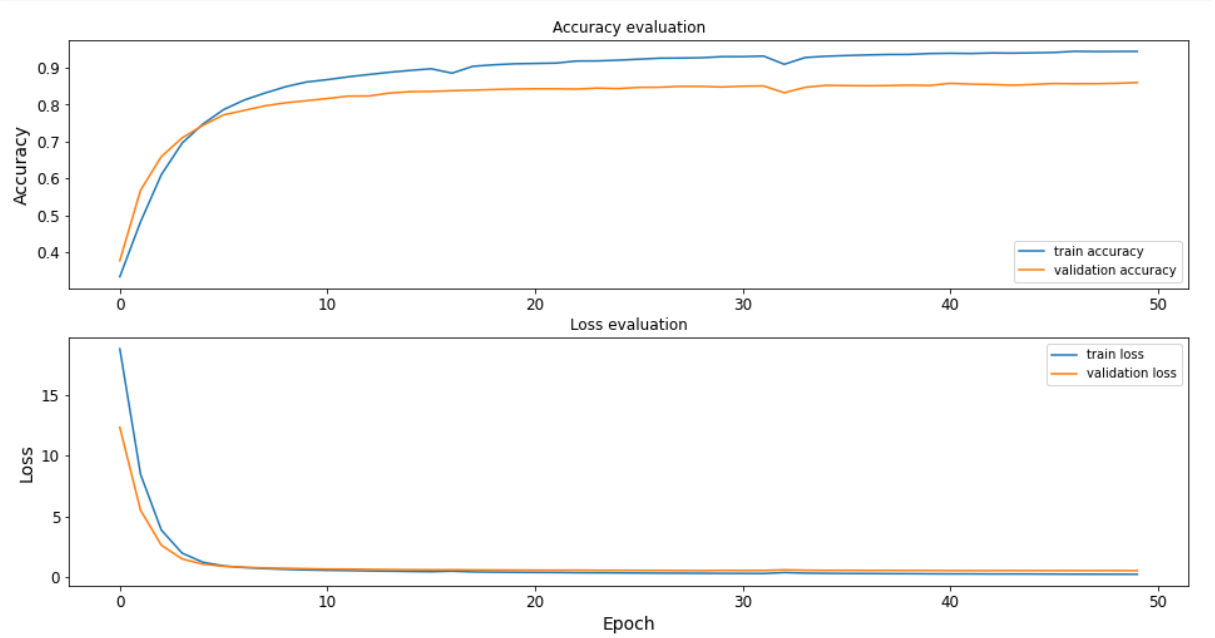

Figura 13**.** Resultados – taxas de perda e acurácia - do modelo final durante 50 épocas de treinamento. Imagem adaptada de [35].

	Taxa de Erro	**Taxa de Acurácia**
Treino	0.177	95%
Validação	0.513	85%
Teste	0.325	93%

Tabela 2. Resultados das Métricas, taxa de erro-loss e acurácia dos testes do modelo final modificado.

Depois de analisarmos os novos resultados, observamos que apesar das taxas não apresentarem grandes mudanças em relação aos resultados obtidos na seção 4.4, chegando a piorar um pouco, olhando principalmente para a curva da taxa de erro, do conjunto de validação, calculada pela função *"sparse categorical crossentropy"* podemos perceber que a mesma

demonstra um bom resultado ao estar acompanhando a curva do conjunto de treinamento sem ser necessariamente igual [38] completamente diferente do que observamos na figura 12, o que nos leva a conclusão de que as modificações, explicadas em 4.5.1, impactaram de forma positiva na rede.

Assim podemos dizer que conseguimos construir um classificador de sentimentos, com uma taxa de 93% de acurácia no conjunto de testes e que tem um comportamento bom de acordo com os gráficos das métricas.

4.6 Baselines

Nesta seção iremos revisar alguns projetos que foram feitos usando a mesma base de dados que utilizamos para o desenvolvimento do classificador mostrando as diferentes abordagens usadas.

- Exemplo 1:

Neste exemplo temos um projeto com o título de *Sentiment Analysis in Twitter 93% Test Acc* [49]. Como o próprio título sugere, este projeto obteve resultados semelhantes ao classificador desenvolvido na seção 4.5.

No desenvolvimento desse projeto foi feita o processamento dos textos, com a eliminação de pontuações, lematização, vetorização usado TF-IDF além de outros passos. Já na parte dos modelos foram utilizados alguns com ênfase em dois: *Logistic Regression* que obteve 91% de acurácia e *Neural Network* com 93% de acurácia [49].

- Exemplo 2:

Neste exemplo foram abordados mais a utilização de modelos de machine learning. A seguir os modelos usados e seus resultados [50].

  - *Multinomial Naive Bayes:* acurácia de 64%.
  - *Logistic Regression:* acurácia de 69%.
  - *Decision Tree:* acurácia de 76%.
  - *Random Forest:* acurácia de 87%.

Dado os resultados desse exemplo é bastante interessante quando observamos o resultado obtido pela Logistic Regression e comparamos com o Logistic Regression do

exemplo 1, enquanto o do exemplo 1 obteve cerca de 91% de acurácia o deste exemplo obteve cerca de 69% valores bem diferentes o que pode indicar a diferença no processamento dos textos e o impacto de que tal passo causa no resultado final.

- Exemplo 3:

Aqui temos um exemplo que utiliza LSTM para criar o classificador de sentimentos, semelhante ao classificador desenvolvido neste trabalho porem sem a utilização das camadas bidirecionais e as funções de normalização nas camadas [51]. O resultado no conjunto de teste chega a 90% de acurácia porem quando observamos o gráfico das métricas, mais precisamente o gráfico da taxa de erro podemos verificar que o comportamento é semelhante aos resultados obtidos na seção 4.4 [51].

Por fim, podemos perceber que apesar de as semelhanças das abordagens, aqui citadas, com o classificador construído neste trabalho temos alguns pontos diferentes tais como utilização da camada bidirecional e as funções de regularização.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi apresentado uma solução para a classificação de sentimentos através de textos da plataforma digital Twitter utilizando estrategias de aprendizagem profunda. O classificador foi desenvolvido com o intuito de categorizar os textos nas classes: Positivo, Negativo, Neutro e Irrelevante. Por sua vez apresentou um acurácia em cerca de 93% sobre os dados de teste demonstrando assim um resultado aceitável, visto que, está bem próximo dos resultados obtidos nos exemplos dos *baselines* citados na seção 4.6.

Durante o desenvolvimento foi experenciado a importância de um bom pré processamento dos textos onde o mesmo tem um grande impacto no desempenho do modelo além de outro ponto que foi evidenciado é que dependendo dos dados a rede neural deverá sofrer alguns ajustes para que venha a demonstrar o seu melhor resultado como foi o caso do classificador que teve que utilizar funções de regularização para melhorar o desempenho e fugir de erros como o *overfitting*.

Desta forma, agora temos um melhor entendimento das áreas abordadas nesse trabalho, Processamento de Linguagem Natural e Aprendizagem Profunda, além da importância de seus usos nos desafios que temos atualmente.

5.1 Trabalhos Futuros

Depois de adquirir e aprofundar o conhecimento nas áreas abordadas – Aprendizagem Profunda e PLN – temos em mente a vontade de expandir o trabalho aqui apresentado acrescentando novas técnicas ou usando-o como base para alcançar novos objetivos.

Como exemplo de desenvolvimentos futuros, temos o intuito de expandir o desenvolvimento da abordagem aqui apresentada acrescentando novos tipos de pré processamento como a utilização de embeddings pré-treinados, aumento na quantidade de dados usados para treinamento e quem sabe tentar generalizar a rede para não só classificar sentimentos de *tweets*, textos do Twitter, e sim, se possível tentar generalizar a rede para classificação de sentimentos em texto no geral independente de sua fonte. Outro ponto que sugerimos é o da possibilidade de usar o classificador aqui desenvolvido para compor uma solução para o desafio de predizer ações na bolsa de valores usando sentimentos.

Assim esperamos que esse trabalho possa servir de base ou inspiração para projetos futuros.

BIBLIOGRAFIA

[1] A. KHARDE, Vishal. Sentiment Analysis of Twitter Data: A Survey of Techniques. International Journal of Computer Applications (0975 – 8887) Volume 139 – No.11, April 2016, Department of Computer Engg, Pune Institute of Computer Technology,Pune University of Pune (India), p. 1-5, 1 abr. 2016.

[2] GUPTA, Shashank. Sentiment Analysis: Concept, Analysis and Applications. [S. l.], 7 jan. 2018. Disponível em: https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17. Acesso em: 10 out. 2022.

[3] Data Science Academy. Deep Learning Book, 2022. Disponível em: https://www.deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais. Acesso em: 10 outubro. 2022.

[4] ZHANG, Aston et al. 9. Recurrent Neural Networks. In: DIVE into Deep Learning. [S. l.: s. n.], 2021. Disponível em: https://d2l.ai/chapter_recurrent-neural-networks/index.html. Acesso em: 10 out. 2022.

[5] VIRAHONDA, Sergio. An easy tutorial about Sentiment Analysis with Deep Learning and Keras: Learn how to easily build, train and validate a Recurrent Neural Network. Recurrent Neural Networks made easy, [s. l.], 8 out. 2020. Disponível em: https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91. Acesso em: 10 out. 2022.

[6] INTRODUÇÃO ao Processamento de Linguagem Natural — Natural Language Processing(NLP): O que é processamento de linguagem natural?. [S. l.], 2 maio 2021. Disponível em: https://medium.com/data-hackers/introdu%C3%A7%C3%A3o-ao-processamento-de-linguagem-natural-natural-language-processing-nlp-be907cd06c71. Acesso em: 10 out. 2022.

[7] AN Overview of Deep Learning applied to Natural Language Processing: Deep learning combined with natural language processing empowers AI to comprehend and create human language. Read on to learn how and where this is used.. [S. l.], 1 abr. 2022. Disponível em: https://kili-technology.com/blog/nlp-deep-learning. Acesso em: 10 out. 2022.

[8] DESVENDANDO o NLP — Parte 3: saiba o que é a Análise de Sentimento. [S. l.], 13 jan. 2021. Disponível em: https://medium.com/dialograma/desvendando-o-nlp-parte-3-saiba-o-que-%C3%A9-a-an%C3%A1lise-de-sentimento-3e1ba8776222. Acesso em: 10 out. 2022.

[9] SENTIMENT Analysis: binds.co Introduz Funcionalidade de Análise de Sentimento. [S. l.], 15 out. 2020. Disponível em: https://blog.binds.co/novo-recurso-sentiment-analysis/. Acesso em: 10 out. 2022.

[10] KUMARI, Kajal. Sentiment classification using NLP With Text Analytics. Analytics Vidhya, 1 nov. 2022. Disponível em: https://www.analyticsvidhya.com/blog/2021/09/sentiment-classification-using-nlp-with-text-analytics/. Acesso em: 10 out. 2022.

[11] KUMAR T, Santhosh. Natural Language Processing – Sentiment Analysis using LSTM. Analytics Vidhya, 22 jun. 2022. Disponível em: https://www.analyticsvidhya.com/blog/2021/06/natural-language-processing-sentiment-analysis-using-lstm/. Acesso em: 10 out. 2022.

[12] BROWNLEE, Jason. Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras. Machine Learning Mastery, 26 jul. 2016. Disponível em: https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/. Acesso em: 10 out. 2022.

[13] BODEPUDI, Rita. Most Popular Programming Languages & Why They're Useful in Machine Learning. Neptune.ai, 21 nov. 2022. Disponível em: https://neptune.ai/blog/programming-languages-machine-learning. Acesso em: 10 out. 2022.

[14] YEGULALP, Serdar. What is TensorFlow? The machine learning library explained: TensorFlow is a Python-friendly open source library for numerical computation that makes machine learning and developing neural networks faster and easier.. [S. l.], 3 jun. 2022. Disponível em: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html. Acesso em: 10 out. 2022.

[15] GUPTA, Akshay. Top 8 Python Libraries For Natural Language Processing (NLP) in 2021. Analytics Vidhya, 1 maio 2021. Disponível em: https://www.analyticsvidhya.com/blog/2021/05/top-8-python-libraries-for-natural-language-processing-nlp-in-2021/. Acesso em: 10 out. 2022.

[16] HARSHITH, Harshith. Text Preprocessing in Natural Language Processing. [S. l.], 21 nov. 2019. Disponível em: https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8#:~:text=Data%20preprocessing%20is%20an%20essential,process%20of%20building%20a%20model. Acesso em: 10 out. 2022.

[17] LOPEZ YSE, Diego. Text Normalization for Natural Language Processing (NLP): Stemming and lemmatization with Python. [S. l.], 17 fev. 2021. Disponível em: https://towardsdatascience.com/text-normalization-for-natural-language-processing-nlp-70a314bfa646. Acesso em: 10 out. 2022.

[18] CHUGH, Aakarsha. ML | Label Encoding of datasets in Python. [S. l.], 23 ago. 2022. Disponível em: https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/. Acesso em: 10 out. 2022.

[19] GEEKSFORGEEKS, GeeksforGeeks. Removing stop words with NLTK in Python. [S. l.], 22 ago. 2022. Disponível em: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/. Acesso em: 10 out. 2022.

[20] JHA, Abhishek. Vectorization Techniques in NLP [Guide]. Neptune.ai, 21 jul. 2022. Disponível em: https://neptune.ai/blog/vectorization-techniques-in-nlp-guide. Acesso em: 10 out. 2022.

[21] MO, Kefei. Hands-on NLP Deep Learning Model Preparation in TensorFlow 2.X: This is a tutorial to walk through the NLP model preparation pipeline: tokenization, sequence

padding, word embeddings, and Embedding layer setups.. [S. l.], 17 ago. 2020. Disponível em: https://towardsdatascience.com/hands-on-nlp-deep-learning-model-preparation-in-tensorflow-2-x-2e8c9f3c7633. Acesso em: 10 out. 2022.

[22] BROWNLEE, Jason. How to Use Word Embedding Layers for Deep Learning with Keras: 2. Keras Embedding Layer. [S. l.], 4 out. 2017. Disponível em: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/#:~:text=The%20Embedding%20layer%20is%20defined,vocabulary%20would%20be%2011%20words. Acesso em: 10 out. 2022.

[23] BROWNLEE, Jason. A Gentle Introduction to Batch Normalization for Deep Neural Networks. [S. l.], 16 jan. 2019. Disponível em: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/. Acesso em: 10 out. 2022.

[24] IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shif. [S. l.], 6 jul. 2015. Disponível em: https://arxiv.org/pdf/1502.03167.pdf. Acesso em: 10 out. 2022.

[25] HINTON, Geoffrey et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research 15 (2014) 1929-1958, p. 1-30, 6 jul. 2015. DOI https://jmlr.org/papers/v15/srivastava14a.html. Disponível em: https://arxiv.org/pdf/1502.03167.pdf. Acesso em: 10 out. 2022.

[26] DROPOUT: A Simple Way to Prevent Neural Networks from Overfitting. In: BROWNLEE, Jason. Dropout Regularization in Deep Learning Models with Keras. [S. l.], 6 jul. 2015. Disponível em: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/. Acesso em: 10 out. 2022.

[27] Data Science Academy. Deep Learning Book, 2022. Disponível em: https://www.deeplearningbook.com.br/arquitetura-de-redes-neurais-long-short-term-memory/. Acesso em: 10 outubro. 2022.

[28] VIRAHONDA, Sergio. An easy tutorial about Sentiment Analysis with Deep Learning and Keras: Bidirectional layers. [S. l.], 8 out. 2020. Disponível em: https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91. Acesso em: 10 out. 2022

[29] Data Science Academy. Deep Learning Book, 2022. Disponível em: https://www.deeplearningbook.com.br/funcao-de-ativacao/. Acesso em: 10 outubro. 2022.

[30] BROWNLEE, Jason. Softmax Activation Function with Python. [S. l.], 19 out. 2020. Disponível em: https://machinelearningmastery.com/softmax-activation-function-with-python/. Acesso em: 10 out. 2022.

[31] WOOD, Thomas. What is the Softmax Function?. [S. l.], 2022. Disponível em: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer. Acesso em: 10 out. 2022.

[32] Passionate-nlp. Twitter Sentiment Analysis. 2. Kaggle, 2021. Disponível em: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis. Acesso em: 13 out. 2022.

[33] SHARMA, Abhishek. A Beginner's Guide to Exploratory Data Analysis (EDA) on Text Data (Amazon Case Study): The Importance of Exploratory Data Analysis (EDA). Analytics Vidhya, 27 abr. 2020. Disponível em: https://www.analyticsvidhya.com/blog/2020/04/beginners-guide-exploratory-data-analysis-text-data/. Acesso em: 10 out. 2022.

[34] ES, Shahul. Exploratory Data Analysis for Natural Language Processing: A Complete Guide to Python Tools. Neptune.ai, 21 set. 2022. Disponível em: https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools. Acesso em: 10 out. 2022.

[35] CARNEIRO S. SIMOES, Jeffson. Classificador de Sentimentos utilizando Aprendizagem Profunda. 1. Github, 2022. Disponível em: https://github.com/Jcss3/Classificador-de-Sentimentos-utilizando-Aprendizagem-Profunda. Acesso em: 13 out. 2022.

[36] NAGPAL, Anuja. L1 and L2 Regularization Methods, Explained: L1 and L2 regularization are the best ways to manage overfitting and perform feature selection when you've got a large set of features.. [S. l.], 5 jan. 2022. Disponível em: https://builtin.com/data-science/l2-regularization. Acesso em: 11 out. 2022.

[37] TEWARI, Ujwal. Regularization — Understanding L1 and L2 regularization for Deep Learning. [S. l.], 9 nov. 2021. Disponível em: https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf. Acesso em: 11 out. 2022.

[38] BROWNLEE, Jason. How to use Learning Curves to Diagnose Machine Learning Model Performance. [S. l.], 6 ago. 2019. Disponível em: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/. Acesso em: 12 out. 2022.

[39] GOOGLE, Google. Colaboratory: Perguntas frequentes. [S. l.], 2022. Disponível em: https://research.google.com/colaboratory/intl/pt-BR/faq.html#:~:text=O%20Colaboratory%20ou%20%E2%80%9CColab%E2%80%9D%20%C3%A9,an%C3%A1lise%20de%20dados%20e%20educa%C3%A7%C3%A3o. Acesso em: 13 out. 2022.

[40] MALADKAR, Kishan. 5 Ways To Handle Missing Values In Machine Learning Datasets. [S. l.], 9 fev. 2018. Disponível em: https://analyticsindiamag.com/5-ways-handle-missing-values-machine-learning-datasets/. Acesso em: 13 out. 2022.

[41] RASTOGI , Kashish. Text Cleaning Methods in NLP. [S. l.], 31 jan. 2022. Disponível em: https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/. Acesso em: 13 out. 2022.

[42] Data Science Academy. Deep Learning Book, 2022. Disponível em: https://www.deeplearningbook.com.br/redes-neurais-recorrentes/. Acesso em: 13 outubro. 2022.

[43] BROWNLEE, Jason. How to Choose an Activation Function for Deep Learning. [S. l.], 18 jan. 2021. Disponível em: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/#:~:text=Activation%20functions%20are%20a%20critical,predictions%20the%20model%20can%20make. Acesso em: 13 out. 2022.

[44] SAXENA , Shipra. Introduction to Batch Normalization. [S. l.], 9 mar. 2021. Disponível em: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/. Acesso em: 13 out. 2022.

[45] GOODFELLOW, Ian et al. Deep Learning: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. The MIT Press Cambridge, Massachusetts London, England: [s. n.], 2016.

[46] PHI, Michael. Illustrated Guide to LSTM's and GRU's: A step by step explanation. [S. l.], 24 set. 2018. Disponível em: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. Acesso em: 30 out. 2022.

[47] KHAN, Etqad. LSTM : What's the fuss about?: LSTM learns. LSTM remembers. Be like LSTM.. [S. l.], 17 jan. 2020. Disponível em: https://medium.com/analytics-vidhya/lstm-whats-the-fuss-about-1ae9d4c3e33e. Acesso em: 30 out. 2022.

[48] BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU). [S. l.], 9 jan. 2019. Disponível em: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. Acesso em: 30 out. 2022.

[49] KATE ARBUZOVA. Twitter Sentiment Analysis. 2. Sentiment Analysis in Twitter 93% Test Acc Kaggle, 2021. Disponível em: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis. Acesso em: 31 out. 2022.

[50] Rohan Paris. Twitter Sentiment Analysis. 2. Text Feature Cleaning/Generation/Model Building Kaggle, 2021. Disponível em: https://www.kaggle.com/code/parisrohan/text-feature-cleaning-generation-model-building. Acesso em: 31 out. 2022.

[51] JVK_CHAITANYA . Twitter Sentiment Analysis. 2. Twitter Sentiment Analysis using LSTM Kaggle, 2021. Disponível em: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis. Acesso em: 31 out. 2022.