UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VINÍCIUS RODRIGUES FERRAZ

ANÁLISE DE SENTIMENTOS E CLASSIFICAÇÃO MULTICLASSE DE TEXTOS APLICADAS AO *CUSTOMER SUCCESS*

BAURU
Maio/2020

VINÍCIUS RODRIGUES FERRAZ

ANÁLISE DE SENTIMENTOS E CLASSIFICAÇÃO MULTICLASSE DE TEXTOS APLICADAS AO *CUSTOMER SUCCESS*

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Associado Aparecido Nilceu Marana

BAURU
Maio/2020

Vinícius Rodrigues Ferraz ANÁLISE DE SENTIMENTOS E CLASSIFICAÇÃO MULTICLASSE DE TEXTOS APLICADAS AO *CUSTOMER SUCCESS*/ Vinícius Rodrigues Ferraz. – Bauru, Maio/2020- 52 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Associado Aparecido Nilceu Marana
Trabalho de Conclusão de Curso – universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Maio/2020.
1. Análise de Sentimentos 2. Classificação Multiclasse 3. Customer Success

Vinícius Rodrigues Ferraz

ANÁLISE DE SENTIMENTOS E CLASSIFICAÇÃO MULTICLASSE DE TEXTOS APLICADAS AO *CUSTOMER SUCCESS*

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Associado Aparecido Nilceu Marana
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Andrea Carla Gonçalves Vianna
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, 11 de Julho de 2020.

Aos meus pais, irmãos e a todos os meus familiares e amigos.

Agradecimentos

A Deus, pela minha vida, e por me permitir ultrapassar todos os obstáculos encontrados ao longo dessa jornada. Aos meus pais e irmãos, que sempre estiveram ao meu lado me apoiando, incentivando e me dando suporte, principalmente nos momentos difíceis. Ao professor Nilceu, por ter sido meu orientador e ter desempenhado tal função com dedicação, amizade e compreensão. A todos os amigos que direta ou indiretamente participaram da minha formação.

Nada te perturbe, Nada te espante,
Tudo passa, Deus não muda,
A paciência tudo alcança;
Quem a Deus tem, Nada lhe falta:
Só Deus basta.

(Santa Tereza D'Avila)

Resumo

Muitas pessoas usam sites de *microblogging* para fazer comentários sobre aquilo que acontece em seu dia-a-dia, sobre o que estão fazendo e sobre suas relações com empresas, quando interagem com elas. De fato, o crescimento da área de tecnologia, e principalmente da Ciência de Dados, tem levado diversas empresas a buscar informações importantes nos sites de *microblogging* e fazer o uso delas para garantir vantagem sobre seus concorrentes. Com o grande crescimento da Ciência de Dados, os modelos de inteligência artificial tem se tornado cada vez mais precisos e robustos e obter informações relevantes tem se tornado cada vez mais fácil. Pensando nisso, esse trabalho propõe o uso das redes neurais recorrentes para realizar análise de sentimentos em textos publicados por usuários em sites de *microblogging*, visando obter informações sobre o sentimento dos usuários sobre empresas. Além disso, para possibilitar a obtenção de um maior nível de detalhamento, realizou-se também a classificação multiclasse de textos negativos em categorias pré-definidas, encontrando assim mais detalhes sobre a opinião dos usuários em relação às empresas.

Palavras-chave: Microblogging, Redes Neurais, LSTM, Redes Neurais Recorrentes, Customer Success.

Abstract

Many people use microblogging sites to comment on what happens in their daily lives, what they are doing and about their relationships with companies, when they interact with them. In fact, the growth in the technology area, and especially in Data Science, has led several companies to search for important information on the microblogging sites and use them to guarantee an advantage over their competitors.

With the great growth of Data Science, artificial intelligence models have become increasingly accurate and robust and obtaining relevant information has become easy. With this in mind, this work proposes the use of Recurrent Neural Networks to perform sentiment analysis in texts published by users on microblogging sites, in order to obtain information about users' feelings about companies. In addition, to enable a greater level of detail to be obtained, the multiclass classification of negative texts in pre-defined categories was also carried out, thus finding more details about the users' opinion in relation to the companies.

Keywords: Microblogging, Neural Networks, LSTM, Recurrent Neural Networks, Customer Success.

Lista de figuras

Lista de quadros

Sumário

1 Introdução

Sites de *Microblogging*, como o Twitter, se tornaram uma fonte de variados tipos de informação. Isso se deve à natureza dos *microblogs*, onde as pessoas postam em tempo real mensagens sobre suas opiniões em uma grande variedade de tópicos, comentam sobre suas atividades diárias, discutem seus problemas e reclamam ou expressam sentimentos positivos de produtos ou serviços que costumam usar em seu dia a dia. De fato, companhias que vendem ou fabricam produtos, e companhias que prestam serviços começaram a se interessar por coletar e trabalhar com este tipo de informação, uma vez que podem, com o uso da Inteligencia Artificial, ou mais especificamente com o uso da análise de sentimentos, obter informações sobre como está o sentimento do público ou seja, dos clientes, em relação ao seu produto ou serviço. Ainda, com este tipo de análise, é possível inferir informações sobre a imagem empresarial da companhia, que é a percepção transmitida ao público-alvo sobre a empresa, que por sua vez pode alavancar ou piorar os lucros de uma companhia (Agarwal et al., 2011, p. 1).

Um dos maiores sites de *microblogging* é o Twitter. Segundo o FAQ do próprio site,

> "O Twitter é um serviço para amigos, familiares e colegas de trabalho se comunicarem e permanecerem conectados por meio da troca de mensagens rápidas e frequentes. As pessoas postam tweets, que podem conter fotos, vídeos, links e texto. Essas mensagens são postadas no seu perfil, enviadas aos seus seguidores e podem ser pesquisadas na pesquisa do Twitter".

O Twitter disponibiliza uma API para que desenvolvedores e pesquisadores possam coletar e trabalhar com as informações encontradas no site. A API de pesquisa padrão do Twitter permite consultas simples de tweets, retornando uma amostra de tweets recentes publicados nos últimos sete dias. Além da API, o Twitter permite, com algumas restrições, a prática de *Web Scraping*, que é uma técnica de extração de dados utilizada para coletar dados de sites por meio de processos automatizados, copiando e armazenando as informações coletadas.

Fazer com que seus clientes tenham o resultado esperado por meio de suas interações com a empresa, o que inclui cada uma das etapas que levaram o cliente até o produto ou serviço, como a compra, o pagamento e a entrega, entre outros, é um desafio que toda companhia enfrenta. Esse processo é conhecido como *Customer Success*, que se resume em satisfazer o cliente e torná-lo fiel à empresa, criando um relacionamento benéfico para ambas as partes.

A análise de sentimentos, cujo objetivo é classificar sentenças, ou um conjunto de sentenças como positivas, negativas ou neutras, juntamente com a classificação multiclasse de textos, que permite separar os textos em diferentes classes, aplicadas às mensagens que os clientes publicam nas redes sociais ou em sites de *micoblogging*, pode ajudar uma empresa a

entender melhor o que seu público-alvo está dizendo publicamente sobre a empresa e sobre seus processos e, através desse entendimento, os gestores podem tomar melhores decisões de negócio como por exemplo distribuir melhor os investimentos da companhia, de modo a melhorar os procedimentos que têm apresentado problema, atendendo assim a satisfação do cliente e atingindo então o *Customer Success*.

De fato, ao utilizar sites de *microblogging*, as pessoas tendem a postar diversas mensagens sobre suas opiniões. Tendo isso em mente, muitas empresas passaram a se interessar pela coleta e análise desse tipo de dado, uma vez que a oportunidade de capturar o sentimento do público em geral sobre sua companhia pode trazer uma enorme vantagem competitiva em relação a seus competidores no mercado.

Em razão dessa grande oportunidade de aumentar seus lucros, melhorar a imagem da empresa em relação aos concorrentes e criar um relacionamento com o cliente atingindo um bom posicionamento de *Customer Success*, a análise de sentimentos surge como um meio de compreender por meio de algoritmos o emocional do público em relação a empresa. Saber se seus clientes estão contentes ou decepcionados com seus produtos ou serviços e saber quais áreas da empresa tem causado este desconforto, fazendo com que eles publiquem mensagens negativas sobre a empresa pode, por exemplo, ajudar uma companhia a investir melhor seu dinheiro, priorizando uma área que tem causado problemas e reduzindo as queixas desses clientes, ou ainda, utilizar o conteúdo das mensagens positivas em seu marketing, criando a partir dele melhores atrativos para tornar clientes, aqueles que ainda o são. (PANG; LEE, 2008, p. 1-2)

Além de detectar o sentimento de uma mensagem, uma empresa pode se interessar também em classificar as mensagens em grupos de interesse, como por exemplo, grupos de problemas para as mensagens com teor negativo. Segundo Aly (2005) "os algoritmos supervisionados de classificação multiclasse visam atribuir um rótulo de classe para cada exemplo de entrada, neste caso, as mensagens publicadas".

Pode-se, a partir da classificação multiclasse, classificar os textos em diferentes classes de problemas que podem ser definidas de acordo com a área de atuação da companhia. Saber que o que mais incomoda os clientes é a falta de atendimento ou talvez a demora para entregar o produto ou serviço, pode trazer a uma empresa a oportunidade de direcionar seus investimentos para o problema em questão, reduzindo assim as queixas de seus clientes e, consequentemente, reduzindo também a quantidade de mensagens públicas com teor negativo sobre a empresa , melhorando assim a imagem empresarial da companhia perante o público.

Considerando que, tendo um conjunto de dados para treinamento onde para cada entrada x tem-se uma saída y1, correspondente ao sentimento (Positivo, Neutro ou Negativo), e uma saída y2, correspondente à classe do problema para as entradas com sentimento negativo, o objetivo em ambos os casos será encontrar um modelo (algoritmo) capaz de produzir a saída correta para dados de entrada que não estavam inclusos nos dados de treinamento. Ou seja, o

modelo deve ser genérico o suficiente para inferir o sentimento ou a classe dos novos dados não vistos anteriormente. Dessa forma, ao aplicar o modelo para dados reais, o mesmo será capaz de produzir respostas aceitáveis que poderão ser usadas para a tomada de decisão em uma empresa.

1.1 Problemas

Segundo Porter e Millar (2016), um conceito importante que destaca o papel da tecnologia da informação na competição entre empresas é a 'cadeia de valor'. O conceito divide as atividades de uma empresa em diferentes atividades executadas técnica e economicamente para os negócios, denominadas 'atividades de valor'. Porter e Millar ainda afirmam que a vantagem competitiva pode ser dividida em duas áreas: custo e diferenciação.

O perfil de custo da empresa refere-se ao custo de realizar todas as atividades de valor relacionadas aos seus concorrentes ou ainda, o quão eficiente em termos monetários são os seus processos e atividades e a capacidade da empresa de se diferenciar reflete a contribuição de cada atividade de valor para atender à demanda do comprador. Em outras palavras é o modo como cada atividade da empresa age para que ela seja diferenciada em relação aos concorrentes para o comprador. Vale destacar que não apenas os produtos ou serviços são capazes de tornar uma empresa diferente, mas também todos os outros procedimentos envolvidos desde a pré-venda até o pós-venda. Isso inclui questões como atendimento efetivo, boa comunicação e tempo de entrega rápido, entre outros.

Tendo em vista que as necessidades do comprador, por sua vez, não depende apenas do impacto do produto ou do serviço, pode-se observar que o entendimento dos textos publicados pelos compradores nos canais de *microblogging* pode trazer para uma companhia uma carga de informação muito valiosa, pois seu conteúdo apresenta a opinião do comprador sobre a capacidade da companhia em atender suas necessidades. Em outras palavras, o fato de os compradores manifestarem publicamente suas opiniões sobre as empresas com as quais se relacionam, torna esse tipo de informação muito valiosa pois, a partir delas, existe a possibilidade de se extrair ideias sobre como melhorar os procedimentos da empresa, tendo em vista a atração e manutenção de clientes.

1.2 Justificativa

Pode-se resumir inteligência de negócio como sendo a tomada de decisão assertiva por parte de uma empresa a partir do entendimento de diversos dados coletados, como números referentes as atividades de valor, ou sobre o posicionamento da companhia no mercado em relação a seus concorrentes. Conclui-se então que a compreensão do sentimento e visão que os compradores possuem em relação à empresa , sobre cada um de seus processos e sobre seus

produtos ou serviços, pode trazer uma carga elevada de informação que por sua vez poderá ser usada como parte da inteligência de negócio de determinada companhia. Sendo assim, pode-se dizer que é de interesse dos gestores possuir esse tipo de informação, uma vez que poderão tomar decisões mais assertivas com a inteligência adquirida.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral desse Trabalho de Conclusão de Curso foi estudar os conceitos básicos de redes neurais recorrentes e as técnicas utilizadas em sua aplicação na análise de sentimentos e na classificação multiclasse de textos. Isso foi utilizado para o desenvolvimento de uma aplicação capaz de receber textos do Twitter, classificar os sentimentos desses textos e realizar naqueles com classificação negativa uma nova classificação, identificando quais áreas da empresa tem causado mais problemas e insatisfações para os clientes.

1.3.2 Objetivos Específicos

Os objetivos específicos desse Trabalho de Conclusão de Curso foram:

(i) Realizar uma revisão de literatura sobre redes neurais recorrentes e sua aplicação na análise de sentimentos e na classificação multiclasse de textos, determinando o método a ser utilizado em cada etapa;

(ii) Pesquisar e selecionar quais ferramentas seriam utilizadas no desenvolvimento da aplicação;

(iii) Realizar a implementação dos métodos com as ferramentas escolhidas;

(iv) Realizar os testes para apurar a eficácia dos métodos.

2 Fundamentação Teórica

Neste capítulo são apresentados e descritos conceitos abordados ao longo do trabalho, pertencentes ao domínio de inteligência artificial, em especial de redes neurais recorrentes, e sua aplicação na análise de sentimento e na classificação multiclasse.

2.1 Redes Neurais

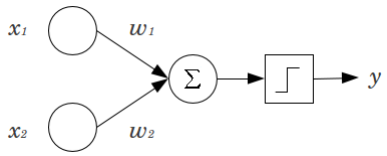
Segundo Kröse e Smagt (1996), uma rede neural artificial, do inglês *Artificial Neural Network* (ANN), pode ser descrita como modelo computacional que possui características particulares, como por exemplo o fato de ser capaz de se adaptar, aprender e agrupar dados. Além disso, afirmam que as ANNs tem suas operações baseadas em processamento paralelo. De acordo com Rauber (1996), as ANNs possuem uma estrutura baseada no sistema nervoso humano (biológico), cuja estrutura básica consiste em neurônios, que são formados por núcleo e corpo. A saída de informação, ou seja, dos sinais elétricos dos neurônios, se dá através do axônio se propagando por várias ramificações para os dendritos, receptores de outros neurônios em uma ligação conhecida como sinapse. Além disso, os neurônios são interconectados em uma estrutura complexa formando uma rede de neurônios, a rede neural.

Assim como no sistema nervoso biológico, as ANNs também são compostas por uma unidade simples, os neurônios artificias. De acordo com Castro e Zuben (2001), nas ANNs, os sinais são transmitidos entre neurônios através de sinapses, cuja eficiência é representada por um peso associado a cada interconexão entre os neurônios da rede. Ademais, no caso das ANNs, o conhecimento se dá através de um processo conhecido como aprendizagem, que pode ser resumida em uma atualização dos valores dos pesos dessas interconexões.

O perceptron é a mais antiga e mais simples ANN sendo composta por apenas um único neurônio. Ele recebe n entradas multiplicadas pelos seus pesos correspondentes w. O núcleo do neurônio é responsável por fazer uma função de soma E, cujo resultado vai ser avaliado por um limiar que tem como objetivo excitar ou não a saída correspondente que no caso do perceptron é única.

Pode-se explicar o perceptron de uma forma intuitiva, usando como exemplo, notas de provas de alunos de determinado curso, para saber se foram aprovados ou reprovados. A Figura 1 mostra a estrutura básica de um perceptron. Supondo que as entradas $x1$ e $x2$ sejam as notas das provas 1 e 2, com pesos $w1$ e $w2$ igual a 0.5, o núcleo seria responsável por fazer a soma ponderada das entradas multiplicadas por seus respectivos pesos, e o limiar igual a 5 faria com que o perceptron produzisse resposta 1 (aprovado) caso a soma ponderada seja $>=$ 5 ou 0 (reprovado) caso seja $<$ 5.

Figura 1 – Perceptron.



Fonte: Elaborada pelo autor.

Rauber (1996) ainda explica que, a partir dessa estrutura básica de rede neural, é possível criar diversas outras topologias que surgem da combinação de camadas de neurônios interligadas, onde a saída de um neurônio é transmitida como entrada para outro neurônio e processada, até uma última camada de saída que produzirá o *output*. A Figura 2 apresenta diversas topologias de redes neurais.

Dentre todas as possíveis topologias de Redes Neurais, a escolhida para esse projeto foram as Redes Neurais Recorrentes, do inglês *Recurrent Neural Networks* (RNNs). Segundo Goodfellow, Bengio e Courville (2016), as RNNs são uma família de redes neurais artificiais especializadas para o processamento de dados sequenciais, do tipo $X1$, $X2$, ... , Xn.

Segundo Nicholson (2020), as RNNs possuem uma estrutura baseada em ciclos que permite que elas levem em consideração o tempo e a sequência dos *inputs*. Para as RNNs, o *input* resulta de uma combinação entre a saída do passo anterior $t-1$, com a entrada do passo atual t. Deste modo uma RNN leva em consideração o resultado obtido em um passado recente, além do *input* recebido no passo atual, um efeito que se assemelha à memória humana. A adição de memória às Redes Neurais tem como objetivo poder utilizar em suas predições as informações contidas na própria sequência, que para muitos casos é uma informação de grande relevância. A Figura 3 mostra a topologia básica de uma RNN.

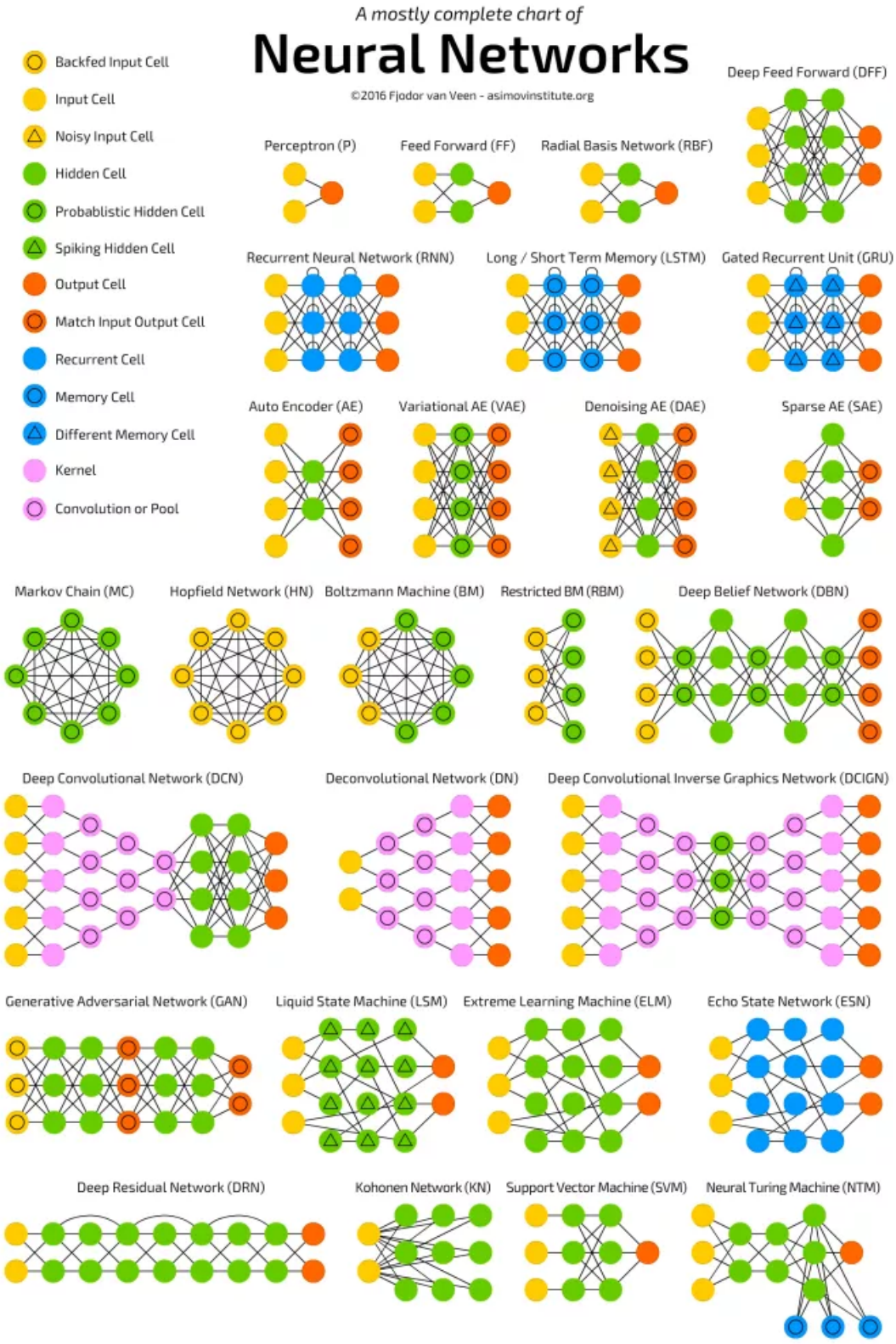
Nicholson (2020) ainda explica que as informações sequenciais são preservadas em um estado oculto da RNN, em inglês *Hidden State*, cujo objetivo é estender a informação, fazendo-a persistir durante os diversos passos de tempo afetando sempre os resultados dos passos posteriores. Isso faz com que existam correlações entre os diversos eventos de uma sequência, em outras palavras, entre o evento em t e os eventos anteriores à ele. Essas correlações são chamadas de Dependências de Longo Prazo, em inglês *Long-Term Dependencies*.

Pode-se descrever matematicamente esse processo de levar adiante o resultado de um passo anterior da seguinte maneira, onde o *hidden state* do passo atual t, h_t é dado pela função de entrada θ das entradas atuais X_t, modificadas pela matriz de peso W adicionada ao *hidden state* do passo anterior $t-1$ multiplicado por sua própria matriz U, uma matriz de transição semelhante à uma cadeia de Markov:

$$h_t = \phi(Wx_t + Uh_{t-1})$$

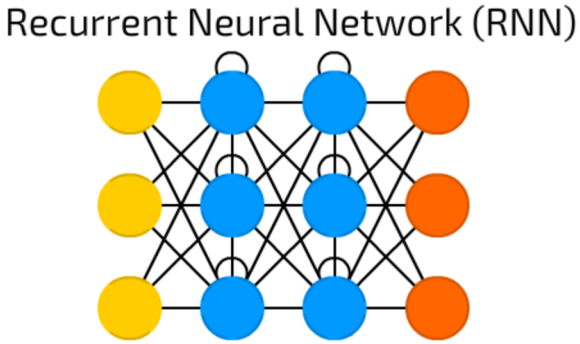
Existem diversas arquiteturas de RNNs. Dentre elas, uma das mais conhecidas é a *Long*

Figura 2 – Topologias de Redes Neurais.



Fonte: DATA SCIENCE ACADEMY(2020))

Figura 3 – Topologia de uma Rede Neural Recorrente.



Fonte: DATA SCIENCE ACADEMY(2020).

Short-Term Memories(LSTM). As LSTMs são um tipo especial de RNNs capazes de aprender dependências de longo prazo. Isso se deve ao fato de que, diferentemente de uma Rede Neural Recorrente comum, as LSTMs realizam uma porção de operações matemáticas em cada passo. Deste modo conseguem ter uma melhor memória sobre os passos anteriores. As LSTMs, junto com as GRUs, *Gated Recurrent Units* pertencem à uma classe de RNNs chamadas *Gated RNNs*, que são, basicamente, RNNs com 'portões'.

Segundo Goodfellow, Bengio e Courville (2016), uma LSTM tem sua estrutura dividida em portões *gates* responsáveis por fazer a seleção das informações que serão escolhidas pela célula, uma vez que pode, por exemplo, ser útil para uma rede neural esquecer o antigo estado. Por exemplo no caso de ser necessário gerar um novo texto, esses portões seriam úteis para que a RNN tenha em sua 'memória' que uma palavra já foi escrita, evitando assim o acontecimento de repetições. A figura 4 mostra arquitetura de uma LSTM

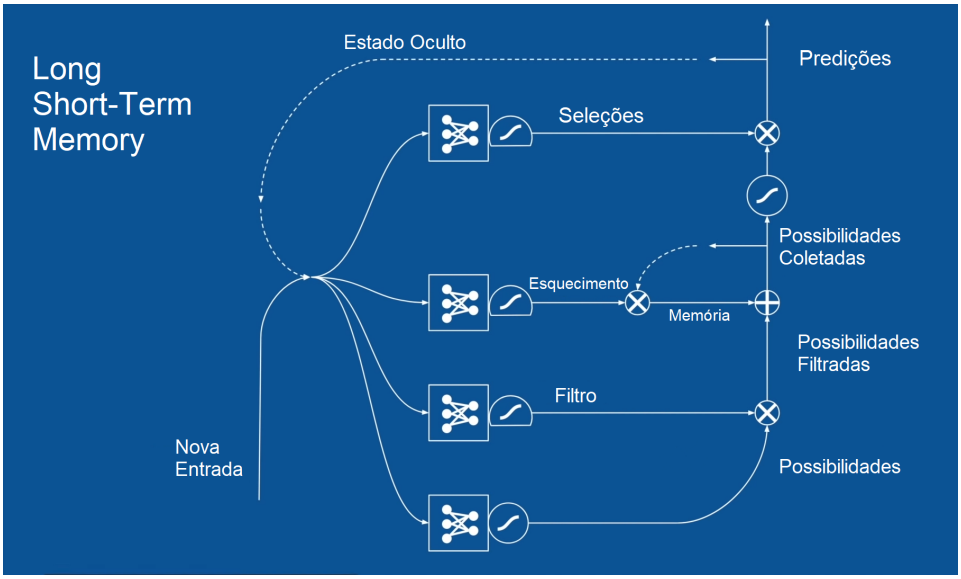
De acordo com Rohrer (2017), o Portão das Possibilidades, *Possibilities Gate*, é responsável por determinar quais são os possíveis *outputs* para a iteração atual. Exemplificando com o caso da geração de texto, esse portão seria responsável por escalar as possíveis próximas palavras.

O Portão de Filtro, *Ignoring Gate*, é responsável por selecionar quais entradas serão ignoradas naquela iteração, filtrando as informações irrelevantes para aquela etapa. Supondo que seja o início de um texto, não é desejado que o caractere de ponto final, '.', seja escalado, por exemplo.

As possibilidades geradas pelo *Possibilities Gate* passam então por uma operação de multiplicação elemento-por-elemento com o *output* do portão ***Ignoring Gate***, marcada pelo sinal $\otimes$, criando então as Possibilidades Filtradas, *Filtered Possibilities*

O Portão de Esquecimento, *Forgetting Gate*, é responsável por fazer a remoção de informações que não são mais úteis. O *Forgetting Gate* recebe o vetor de entradas do passo atual e o *hidden-state* do passo anterior, multiplica a entrada pelas matrizes de peso, adiciona

Figura 4 – Arquitetura de uma LSTM.



Fonte: Adaptado de Rohrer (2017)

o bias e executa a função de ativação que, nesse caso, resulta em uma saída binária onde valores com saída 0 serão esquecidos e valores com saída 1 permanecem para os procedimentos seguintes.

Em seguida, a saída fornecida pelo *Forgetting Gate* passa por uma operação de multiplicação elemento-por-elemento, denotada pelo sinal $\otimes$, com o vetor das possibilidades coletadas, que será explicado a seguir. O resultado dessa multiplicação é um vetor de valores a serem lembrados, o que pode ser chamado de memória.

Após as Possibilidades Filtradas serem obtidas pela multiplicação entre a saída do *Possibilities Gate* pela saída do *Ignoring Gate*, elas passam por uma operação de soma elemento-por-elemento com a memória, denotado pelo sinal $\oplus$, que resulta em um novo vetor de possibilidades, as Possibilidades Coletadas, que representam então o conjunto das informações importantes para o passo atual, ou seja, as informações que foram lembradas pela memória somadas às possibilidades importantes para o passo atual e as não ignoradas pelo *Ignoring Gate*. Deste modo, as possibilidades coletadas são um conjunto de possíveis respostas mais plausível para o passo atual.

Após esse procedimento, as possibilidades coletadas passam pela função de ativação Tanh, responsável por ajustar os valores coletados para valores entre -1 e 1. O resultado dessa operação é então somado às possibilidades selecionadas pelo portão *Selection Gate* que recebe como entrada o vetor de entradas do passo atual e o *hidden-state* do passo anterior e calcula as probabilidades de cada possibilidade. A saída do *Selection Gate* passa então por um novo procedimento de multiplicação elemento-por-elemento com o vetor de *output* da função Tanh. Esse procedimento resultará nas probabilidades de cada possibilidade, nos dizendo qual é a

predição, o *output*, para a célula atual. Esse *output*, também conhecido como *hidden-state* é enviado para a célula seguinte e o procedimento se repete.

Uma vez que as LSTMs trabalham com dados sequenciais, pode-se usá-las para fazer a classificação de textos de acordo com o sentimento que eles apresentam, uma vez que, textos nada mais são que uma sequência de palavras. De acordo com Wang et al. (2018), as LSTMs são capazes de capturar dependências entre palavras em textos curtos, além de salientar que para o processamento de linguagem natural, *Natural Language Processing*, popularmente conhecido por NLP, é útil analisar relações entre as palavras e suas ocorrências em uma sentença. Deste modo, as LSTMs podem ajudar na captura dessas dependências, uma vez que conseguem memorizar informações sobre as palavras já vistas em dada sequência.

2.2 Processamento de Linguagem Natural

O NLP é um ramo da inteligência artificial que busca entender a linguagem natural dos humanos, ou, em outras palavras, busca-se ensinar a máquina a entender como os humanos se comunicam. 'O Processamento de linguagem natural é um campo da inteligência artificial que oferece às máquinas a capacidade de ler, entender e derivar significado das linguagens humanas.' (YSE, 2019).

De acordo com Brownlee (2017), a Linguagem Natural, *Natural Language*, pode ser entendida como sendo a maneira pela qual os humanos se comunicam, que basicamente se resume em voz e texto, ou ainda, escrita e fala. Conforme Goldberg (2017), processar esse tipo de dado é uma tarefa cuja necessidade tem crescido muito nos últimos anos, acompanhando o crescimento da quantidade de dados sendo gerados e armazenados e a necessidade de existirem sistemas cada vez mais capazes de se comunicar com uma linguagem cada vez mais próxima e mais amigável. Porém, essa é uma tarefa difícil, uma vez que a linguagem dos humanos é altamente ambígua e muito variável.

2.2.1 Análise de Sentimentos

A análise de sentimentos, *sentiment analysis*, é um subcampo do Processamento de Linguagem Natural onde busca-se criar um modelo capaz de diferenciar o sentimento expresso por determinado texto, determinada sentença ou até mesmo um documento por inteiro. Duas são as formas primárias de análise de sentimento. A primeira consiste em classificar os textos perante à polarização, classificando-os entre positivos, caso um texto expresse bons sentimentos e, caso contrário, negativos. Nessa primeira forma, pode ser incluída a classe neutra. Já a segunda forma consiste em classificar os textos em classes de sentimentos mais específicas de acordo com o que cada texto expressa. Nessa forma entram classes como alegria, tristeza, raiva e empolgação, entre outras.

A análise de sentimentos é amplamente utilizada por empresas que visam entender o que seus clientes estão pensando sobre a companhia, quando, por exemplo, expressam suas opiniões em publicações nas redes sociais. Nesse contexto, a análise de sentimentos surge como um meio de compreender por meio de algoritmos o emocional do público em relação a empresa, possibilitando que interessados, normalmente gestores e executivos, possam entender se seus clientes estão contentes ou decepcionados com seus produtos ou serviços e a partir dessa informação podem tomar diferentes ações estratégicas em busca de corrigir os problemas ou de aproveitar os pontos positivos, gerando assim uma vantagem competitiva. Neste trabalho, a análise de sentimentos foi utilizada em textos coletados do Twitter que tem relação com algumas empresas selecionadas, dessa forma será possível entender qual o sentimento dos clientes quando se referem às empresas selecionadas.

2.2.2 Classificação Multiclasse de Textos

A Classificação Multiclasse de Textos, *Multi-Class Text Classification*, é um subproblema de uma classe de problemas conhecida como Classificação Multiclasse, que consiste em um problema de classificação com mais de duas classes. O objetivo é classificar os textos recebidos em uma entre diversas classes, como classificar notícias de jornal de acordo com o assunto, que pode ser esporte, economia ou tecnologia, entre outras tantas. Neste trabalho, o objetivo foi classificar os textos identificados como negativos pela análise de sentimento em classes de problema pré-definidas, conseguindo desse modo, encontrar em um nível mais detalhado os problemas que assolam determinadas empresas quando trata-se de textos publicados por seus clientes em redes sociais, neste caso, o Twitter.

2.3 Sucesso do Cliente

O sucesso do cliente, *Customer Success*, é quando o cliente alcança o resultado desejado por ele em suas interações com sua empresa, o que envolve todos os diversos aspectos e processos em uma compra ou contratação de serviço. Pode-se destacar algumas etapas como escolha do produto, atendimento, funcionários, recebimento, pagamento, preço, entrega e prazo, entre outras. Conforme explica Mehta ([201-]), o *Customer Success* é uma metodologia de negócios usada para garantir à gestão de determinada companhia que seus clientes estejam recebendo aquilo que esperam da empresa, além de ser usada como estratégia de relacionamento empresa-cliente, que pode, por sua vez, fazer com que determinado cliente se torne fiel à empresa.

Utilizar da análise de sentimentos em textos retirados das redes sociais pode ajudar um gestor a entender melhor como seus clientes se comportam ao interagir com a empresa e por meio de uma análise mais detalhada, pode-se chegar aos processos que apresentam sucesso ou falha. Com essa informação, a gestão de uma empresa fica capacitada a tomar decisões

de negócio mais precisas. Saber que a empresa tem um grande problema de qualidade nos produtos pode ajudar a gestão a decidir por uma troca de materiais, enquanto que descobrir a informação de que a velocidade de entrega agrada aos clientes pode gerar uma vantagem competitiva ao ser usada no marketing, por exemplo.

3 Desenvolvimento

Esse projeto consiste em uma estrutura de três arquivos. O primeiro contém um dicionário de *emoticons*, que será utilizado para convertê-los em *tokens*, o segundo contém as funções de pré-processamento dos textos, que serão utilizadas no tratamento dos dados e o terceiro arquivo contém a classe principal responsável por utilizar as funções de pré-processamento para realizar os tratamentos necessários nos dados, criar e treinar o modelo de *word embeddings*, criar e treinar o modelo de rede neural, avaliar o modelo criado e realizar predições para novos exemplos. Todos os procedimentos estão descritos nas sessões deste capítulo. A Figura 5 demonstra a criação da classe e seu construtor. Durante a criação da instância, o construtor recebe os parâmetros e se certifica de que estejam nos devidos formatos. Vale notar que alguns parâmetros possuem valores padrão, caso nenhum seja enviado. Após isso os parâmetros recebidos são atribuídos aos atributos da classe.

Figura 5 – Definição da classe e do construtor.

```
class MulticlassClassification:

    def __init__(self, data: pd.DataFrame, text: str, target: str,
                 output_filename: str, tokenizer=None, model=None,
                 word_embeddings=None, max_len=100, stop_words=None):
        self.data = data
        self.text = text
        self.target = target
        self.output_filename = output_filename
        if tokenizer is None:
            self.tokenizer = Tokenizer()
        self.model = model
        self.w2v_model = word_embeddings
        self.max_len = max_len
        self.stop_words = stop_words
```

Fonte: Elaborado pelo autor

3.1 Redes Neurais

A proposta deste trabalho consistiu em utilizar as redes neurais recorrentes, em especial as LSTMs para apresentar um modelo capaz de realizar a análise de sentimento em textos retirados de redes sociais com comentários de clientes sobre algumas empresas. Em seguida, apresentar um segundo modelo, também de LSTM, capaz de classificar os textos classificados pelo primeiro modelo como negativos, entre algumas categorias pré-selecionadas de problemas. Deste modo, poder-se-á compreender, para as empresas tratadas, quais são as categorias de problema que apresentam maior intensidade e fazer algumas comparações entre as empresas.

3.2 Base de dados

A base de dados escolhida para treinar e testar as redes neurais foi retirada do Kaggle, contendo publicações do Twitter sobre seis linhas aéreas americanas. O conjunto de dados é chamado "*Twitter US Airline Sentiment*", que foi baixado do Kaggle como um arquivo csv. Sua fonte original era da biblioteca *Data for Everyone da Crowdflower*. Os tweets foram retirados do Twitter em fevereiro de 2015 sobre cada uma das principais companhias aéreas dos EUA. Os colaboradores então classificaram cada tweet como 'positivo', 'neutro' ou 'negativo', seguidos pela categorização de motivos, para os negativos.

A escolha dessa base foi realizado pelo fato de que os textos já estão previamente classificados e portanto prontos para serem aplicados como base de treino do modelo proposto. Essa escolha evitou a necessidade de um trabalho manual onde seria necessário coletar, analisar e classificar os textos para então poder inseri-los como base de treinamento. Não foi encontrada, no momento da escolha da base de dados, uma base de tweets escritos em português e previamente classificados. Outro fator para a escolha dessa base de dados foi o assunto, *Customer Success* que tem ganhado cada vez mais popularidade, em especial na área de gestão de empresas.

Os tweets dessa base foram classificados em relação ao sentimento e aqueles com sentimento negativo foram classificados quanto ao problema encontrado. As classes de problema são: Problemas com serviço de atendimento ao cliente, Voo atrasado, Voo cancelado, Bagagem perdida, Bagagem danificada, Voo ruim, Problema com reservas de voo, Reclamações sobre os atendentes, Filas longas e Outros.

3.3 Tratamento dos textos

Dados textuais apresentam uma enorme quantidade de dimensões. Pode-se notar isso pela enorme quantidade de palavras que uma língua pode ter. Além disso, esse tipo de dado é também muito esparso, uma vez que para formar uma frase, usa-se apenas algumas poucas palavras dentre todas as existentes. Também é importante notar que existem diversas maneiras de escrever a mesma frase, além de existirem algumas palavras com diversos sentidos diferentes. Por conta disso, torna-se muito difícil trabalhar com textos e para poder tirar melhor proveito das informações que eles fornecem, deve-se fazer uso de algumas técnicas de tratamento para transformar os textos em dados mais carregados de informação, o que permitirá que a rede neural capture informações como contexto, semântica e relacionamentos entre as palavras.

3.3.1 *Tagging*

A primeira etapa de tratamento é o *tagging*, onde usa-se algumas *tags* para corrigir dados que podem, de certa forma, atrapalhar os métodos em fazer uma predição correta. Para

esse projeto foram feitas *tags* para url como 'www.google.com', *emoticons*, como 🙂, medidas de tempo como '2 horas' ou horários, como '02:30'. Os processos para cada etapa de *tagging* estão descritos à seguir.

3.3.1.1 *Emoticons*

Emoticons, nome derivado das palavra em inglês *emotion*, emoção, e *icon*, icone, são um conjunto de símbolos muito utilizado nos meios de comunicação digitais, em especial nas redes sociais como Twitter e Facebook e também nos aplicativos de mensagem instantânea, como Whatsapp e Telegram. *Emoticons* são principalmente utilizados pelas pessoas para que possam expressar suas emoções como felicidade, 🙂 ou tristeza 🙁.

Com base nessa definição, pode-se notar que é de grande importância capturar os sentimentos expressos pelo *emoticons* nos textos publicados no Twitter sobre as companhias aéreas, uma vez que o sentimento expresso pelos *emoticons* pode auxiliar o modelo a encontrar a resposta correta para a classificação do texto quanto ao sentimento, na análise de sentimentos, ou até mesmo quanto à classe de problema que ele pertence, na classificação multiclasse, já que pode, por exemplo, acontecer de um *emoticon* ser muito utilizado em uma determinada categoria. O *emoticon* de relógio pode, por exemplo ser muito utilizado em textos da categoria 'atraso'. Para isso, foi criado um dicionário em Python com os *emoticons* associados a uma *tag* do seu respectivo sentimento. A Figura 6 apresenta uma parte desse dicionário.

Figura 6 – Dicionário de Emoticons.

```
' sunglasses ': ['😎'],
' killed ': ['😵'],
' smirking ': ['😏', '😼', '😈'],
' relieved ': ['😌'],
' shame ': ['😳'],
' clown ': ['🤡'],
' stars ': ['🤩'],
' mindblow ': ['🤯'],
' crazy ': ['🤪'],
' silence ': ['🤫'],
' ghost ': ['👻'],
' bye ': ['👋'],
' applause ': ['👏'],
' hope ': ['🤞', '🤗', '🙏'],
' deal ': ['🤝'],
```

Fonte: Elaborado pelo autor.

A Figura 7 apresenta o código que aplica o tratamento aos textos. A função recebe uma coluna de textos e retorna as colunas substituindo todos os *emoticons* por sua respectiva *tag*. Primeiro o dicionário é coletado pela variável *'emoticon'*, em seguida inverte-se o dicionário transformando sua estrutura que era de uma *tag* para diversos *emoticons* em um novo dicionário onde cada *emoticon* tem sua *tag*, mesmo que uma *tag* seja compartilhada entre vários *emoticons*.

Isso é necessário para que seja possível buscar os *emoticons* nas chaves do dicionário, facilitando o processo de busca. Em seguida, todos os textos são percorridos, e os *emoticons* encontrados são então substituídos pela *tag* correspondente.

Figura 7 – Função para Tratamento de Emoticons.

```
def create_emoticons_tags(column: pd.Series):
    """ Return texts with tags on emoticons. """

    emots = emoticons.emoticons_dict

    dict_emoticons = {}
    for key in emots.keys():
        for value in emots[key]:
            dict_emoticons.update({value: key})

    tagged = []
    for item in column:
        for emt in dict_emoticons.keys():
            if emt in item:
                item = re.sub(emt, dict_emoticons[emt], item)
        tagged.append(item)
    return tagged
```

Fonte: Elaborado pelo autor.

3.3.1.2 Urls

O tratamento de links, urls, é mais simples, pois precisa-se apenas passar uma busca utilizando regex, uma biblioteca do Python para fazer a captura de expressões regulares. Com ela pode-se encontrar padrões de texto e a partir daí fazer alguma operação. Neste caso, busca-se encontrar links, ou seja, sequências de string como 'www.google.com' ou 'https://www.example.com/foo/?bar=42quux' e substituí-los por uma tag, a escolhida foi 'URL'. A Figura 8 mostra a função responsável por fazer esta operação e a Figura 9 mostra a *string* utilizada para fazer a captura, que foi cortada na Figura 8.

Figura 8 – Função para Tratamento de urls.

```
def create_url_tags(column: pd.Series):
    """
        Return texts with tags on URLs
        http://www.link.com -> "URL"
    """

    for index, item in enumerate(column):
        for word in item.split():
            if re.match('(https?://(?:www\.|(?!www))[a-zA-Z0-9][a-zA-Z
                item = item.replace(word, 'URL')

    return pd.Series(column)
```

Fonte: Elaborado pelo autor.

Figura 9 – String de captura de urls.

```
'''(https?://(?:www\.|(?!www))[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|
www\.[a-zA-Z0-9][a-zA-Z0-9-]+[a-zA-Z0-9]\.[^\s]{2,}|
https?://(?:www\.|(?!www))[a-zA-Z0-9]+\.[^\s]{2,}|www\.[a-zA-Z0-9]+\.[^\s]{2,})'''
```

Fonte: Elaborado pelo autor.

3.3.1.3 Tempos e Horários

Fazendo uma breve análise do *dataset*, pode-se observar que existem muitas marcações de tempo, principalmente em textos onde o voo deveria sair em determinado horário ou está atrasado em determinada quantidade de minutos. Pensando nisso, nessa etapa foi feito o tratamento desses dados de tempo e horário, substituindo textos que contém contagem de horas, como '2h' ou '3 horas' pela *tag* '*hour*', o mesmo foi feito para contagens de minutos e segundos, substituindo por '*minute*' ou '*second*', respectivamente. Além disso, foi adicionada a *tag* '*time*', para aqueles textos que contém uma citação explícita de horário, como '02:30'. A Figura 10 mostra a função responsável por aplicar essa operação. Novamente, o regex foi utilizado para fazer a captura dos padrões de texto.

Figura 10 – Função para tratamento de tempos e horários.

```
def create_time_tags(column: pd.Series):
    """
        Return texts with tags on time
        13:30 -> 'time'
        13h -> 'hour'
        12 min -> 'minute'
        and so on..
    """
    clear_column = []
    for text in column:
        clear_text = []
        for word in text.split():
            word = re.sub('[1-9]hora|[0-5][0-9]horas|[0-5][0-9]h|[0-9]h', 'hour', word)
            word = re.sub('[1-9]minuto|[0-5][0-9]minutos|[0-5][0-9]min|[0-9]min|[0-5][0-9]m|[0-9]m', 'minute', word)
            word = re.sub("[1-9]segundo|[0-5][0-9]segundos|[0-5][0-9]seg|[0-9]seg|[0-5][0-9]s|[0-9]s", 'second', word)
            word = re.sub('(?:[01]\d|2[0-3]):[0-5][0-9]', 'time', word)

            clear_text.append(word.lower())
        clear_column.append(' '.join(clear_text))
    return clear_column
```

Fonte: Elaborado pelo autor.

3.3.2 Limpeza dos textos

Nessa etapa é realizada uma limpeza nos textos, excluindo palavras e caracteres que não são de interesse. Pontuações, acentos, espaços em branco e repetições de caracteres, como em 'exeeeeeemplo' e a remoção de *stopwords*, que são palavras que aparecem muitas vezes nos textos, porém não apresentam sentido ou conteúdo informativo à frase. O maior exemplo de *stopwords* é a classe das preposições. Palavras como 'a', 'em', 'de', 'para', 'então' aparecem repetidas vezes em textos, porém não são palavras que contém grandes cargas de informação e também é possível destacar que palavras desse tipo são usadas em todas as classes de

frases, fato que não ajuda o modelo a classificar corretamente. Nesta etapa também é feita a limpeza das citações de usuários do Twitter, que não apresentam informações relevantes, como '@usuario'. Além disso, foi escolhido manter as citações de *hashtags*, uma vez que existem muitas que podem denotar sentimentos, como por exemplo '#vergonha' ou '#amei'. A Figura 11 apresenta a função para limpeza dos textos.

Figura 11 – Função para limpeza dos textos.

```
def clean_texts(column: pd.Series, stop_words: list):
    """ Return texts free of stopwords, accents, punctuations and twitter mentions """

    string.punctuation = string.punctuation.replace('#', '')
    string.punctuation = string.punctuation.replace('@', '')

    clear_column = []
    for text in column:
        """ Removing stopwords. """
        clear_text = [word for word in str(text).split() if word not in stop_words]

        clear_words = []
        for word in clear_text:

            """ Removing multiple vocaaaaals. """
            word = re.sub('aa+', 'a', word.lower())
            word = re.sub('ee+', 'e', word.lower())
            word = re.sub('ii+', 'i', word.lower())
            word = re.sub('oo+', 'o', word.lower())
            word = re.sub('uu+', 'u', word.lower())

            """ Removing @user and not A-Z chars. """
            word = re.sub('@\S+|[^A-Za-z]+', ' ', word)
            clear_word = ''.join([letter for letter in word if letter not in string.punctuation])

            """ Removing grouped blank spaces. """
            clear_word = re.sub('(\s)+', ' ', clear_word)

            clear_words.append(clear_word.lower())
        clear_column.append(' '.join(clear_words))

    return clear_column
```

Fonte: Elaborado pelo autor.

3.3.3 Stemming

Por sua definição, o processo de *stemming* pode ser entendido como sendo o processo, na morfologia, de reduzir palavras flexionadas ou derivadas ao seu tronco, *stem*, base ou raiz. Com o processo de *stemming*, palavras como 'anda', 'andar', 'andou', 'andando', 'ando' seriam reduzidas a um mesmo *stem*, 'and'. Em NLP, esse é um processo muito útil pois permite reduzir palavras flexionadas a uma única palavra, neste caso o *stem*, que representa toda a 'família'. Deste modo os dados ficam mais densos, ou seja, tem-se um dicionário de palavras com menor número de dimensões. O Snowballstemmer é um algoritmo desenvolvido por Martin Porter, capaz de realizar o procedimento de *stemming* a partir da remoção de sufixos. Ele está disponível na biblioteca de Python para NLP, NLTK e foi escolhido por ser eficiente e de uso comum em diversos trabalhos de NLP. A Figura 12 mostra a função responsável por aplicar esse procedimento.

Figura 12 – Função de stemming.

```
def stemming(column: pd.Series, stemmer=None):
    """ Return texts stemmed. """
    if stemmer is None:
        stemmer = SnowballStemmer("english")
    stemmed_texts = []
    for text in column:
        stemmed_texts.append([' '.join(stemmer.stem(word)) for word in text.split()])

    return stemmed_texts
```

Fonte: Elaborado pelo autor.

3.3.4 Oversampling e Undersampling

O *oversampling* e o *undersampling* são técnicas utilizadas para tratar o desbalanceamento entre classes na base de dados, isto é, balancear bases de dados que possuem classes com maior representatividade. No caso da análise de sentimentos, nota-se que grande parte dos textos são negativos e treinar a rede neural sem fazer esse ajuste pode causar uma tendencia na rede para que classifique todos os textos como negativos. Como exemplo, se 90% dos textos são negativos e 10% são positivos, uma rede que classifica todos os exemplos recebidos como negativos ainda teria uma assertividade de 90%, apesar de não classificar como deveria os positivos. O *oversampling* faz esse ajuste replicando os dados das classes menos representadas, até que exista um número igual de exemplos para todas as classes. De modo contrário, o *undersampling* cria um ponto de corte nas classes de maior representatividade, de modo que todas tenham a mesma quantidade de exemplos. Ambos os métodos tem suas vantagens e desvantagens. Enquanto o *undersampling* joga parte dos dados para fora da aplicação, ignorando-os, o *oversampling* replica exemplos. Deste modo, a escolha do método depende da base de dados em que será aplicada. Como o *dataset* usado para este trabalho não é grande o suficiente para que, ao cortar dados com o *undersamplig*, ainda exista uma quantidade de dados suficiente para o treinamento do modelo, o *oversampling* foi escolhido. A Figura 13 mostra a função responsável por aplicar esse procedimento.

Figura 13 – Função de oversampling.

```
def oversampling(df: pd.DataFrame, column: str):
    """ Balance the DataFrame into equals samples of each class. """
    dict_ = {}
    for key in df[column].value_counts().keys():
        dict_.update({key: df[column].value_counts()[key]})

    sample = max(dict_.values())

    df_oversampled = pd.concat(
        [df[df[column] == key].sample(sample, replace=True, random_state=2) for key in dict_.keys()])

    return df_oversampled
```

Fonte: Elaborado pelo autor.

3.4 *Word Embeddings*

De acordo com Karani (2018), *word embeddings* são técnicas capazes de, ao criar um vetor de representação de palavras, extrair informações sobre as palavras como contexto, semântica, sintaxe e até mesmo correlacionamentos entre as palavras. As técnicas de *word embeddings* consistem em mapear palavras para um número ou em um vetor de números, de forma que, a partir disso seja possível usar a matemática para, por exemplo, calcular a proximidade entre as palavras.

Um exemplo simples de *word embedding* seria a representação das palavras 'Rei', 'Rainha' e Príncipe', mapeados a partir de um vetor de três dimensões de qualidades, com cada índice desse vetor representando respectivamente: Gênero, Nobreza, Maturidade. Sendo 0 para gênero masculino e 1 para feminino, 0 para nobre e 1 para desnobre. E 0 para jovem e 1 para adulto. Pode-se então mapear a palavra 'Rei' para o vetor [0, 0, 1] homem, nobre e adulto, 'Rainha' para [1,0,1], mulher, nobre e adulto, e 'Príncipe' para [0, 0, 0] homem, nobre e jovem. Com esse exemplo seria possível usar métodos como a similaridade do cosseno para calcular a distância entre os vetores e consequentemente a similaridade entre as palavas.

Esse é um exemplo simples que facilita o entendimento, mas vale lembrar que as técnicas de *word embedding* podem usar um número muito maior de dimensões, além de que suas dimensões não necessariamente representam ideias simples como gênero ou nobreza. Para esse trabalho foi escolhido arbitrariamente o valor de 100 dimensões, a fim de possuir um vetor de tamanho razoável e que não tornasse o código demasiadamente lento.

O Word2vec é uma das técnicas mais populares de *word embedding*, cuja formação dos vetores é realizada por uma rede neural. Nesse trabalho foi utilizada a estrutura disponibilizada pela gensim, uma biblioteca para Python, que permite a importação do modelo já criado, restando apenas a etapa de treinamento. Como hiperparâmetros foram utilizados: 'size=100' para que seja um vetor de dimensões razoável e que não vai levar muito tempo para ser treinado, 'window=7' para que cada palavra predita não esteja muito distante da palavra real, durante o treinamento do word2vec, 'min_count=10' para que todas as palavras que aparecem menos de 10 vezes durante todo o *dataset* sejam ignoradas, 'workers=8' para que sejam usadas todas as *threads* disponíveis no computador em que o modelo foi treinado e 'seed=5', valor arbitrário escolhido como padrão para que os resultados de valores aleatórios possam ser reproduzidos.

A Figura 14 demonstra a função utilizada para criar e treinar o word2vec. A Figura 15 mostra a lista de palavras classificadas pelo modelo treinado de word2vec mais parecidas com a palavra bolsa, 'bag'. Pode-se observar que as palavras mais parecidas são 'bolsas', 'bagagem' e 'mala de viagem'.

Figura 14 – Função de Word Embedding.

```
def w2v(df: pd.DataFrame, column: str, max_len: int):
    """ Creates and train word2vec model. """

    w2v_model = gensim.models.word2vec.Word2Vec(size=max_len, window=7, min_count=10, workers=8, seed=5)

    """ Splitting all texts into words. """
    tweets = [text.split() for text in df[column]]

    """ Building the vocabulary. """
    w2v_model.build_vocab(tweets)

    """ Getting size of vocab. """
    # words = w2v_model.wv.vocab.keys()
    # vocab_size = len(words)
    # print("Vocab size: ", vocab_size)

    """ Training the w2v model. """
    w2v_model.train(tweets, total_examples=len(tweets), epochs=32)

    return w2v_model
```

Fonte: Elaborado pelo autor.

Figura 15 – Palavras similares à 'bag'.

```
model.w2v_model.wv.similar_by_word("bag")
```

```
[('bags', 0.5797210931777954),
 ('luggage', 0.5625419616699219),
 ('suitcase', 0.4971080422401428),
 ('stroller', 0.483508825302124),
 ('carry', 0.47925350069999695),
 ('suitcases', 0.4725591838359833),
 ('baggage', 0.46182191371917725),
 ('meds', 0.4520452916622162),
 ('item', 0.43623819947242737),
 ('backpack', 0.4172065258026123)]
```

Fonte: Elaborado pelo autor.

3.5 Estrutura do Modelo

Esse capítulo explica a estrutura final utilizada para a criação do modelo de rede neural, bem como a motivação para a escolha dos hiperparâmetros e as diferenças entre o modelo de análise de sentimentos e o modelo de classificação multiclasse de textos.

O modelo escolhido para este trabalho foi, como já citado, um modelo de LSTM, que recebe como *feature* o vetor de *word embeddings* dos textos do dataset tratados pelos métodos de preprocessamento e como target a classificação correta do sentimento ou da classe de problema, também provindo do *dataset*. A Figura 16 mostra a função de criação da estrutura do modelo, que recebe como parâmetros a quantidade de unidades, que tem como valor padrão 100, a quantidade de neurônios a serem utilizadas na última camada, que deve ser igual à quantidade de classes distintas na variável *target* e que tem como valor padrão 3, que foi

escolhido por conveniência já que o primeiro modelo a ser criado é o de análise de sentimentos que possui 3 classes na variável *target* (positivo, negativo e neutro) e a camada de *embedding* gerada à partir do vocabulário do modelo de word2vec.

Figura 16 – Estrutura do modelo de rede neural.

```
@staticmethod
def create_model(units=100, dense_neurons=3, embedding_layer=None):
    # print('creating model \n')
    model = Sequential()
    model.add(embedding_layer)
    model.add(Dropout(0.4, seed=5))
    model.add(LSTM(units, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(dense_neurons, activation='softmax'))

    model.summary()
    return model
```

Fonte: Elaborado pelo autor.

Em resumo, o modelo é definido como sendo um modelo sequencial, isto é, um modelo estruturado em uma sequência de camadas interligadas. Em seguida é adicionada ao modelo a camada de *embedding*, responsável por receber os dados transformados pelo word2vec e ajustá-los em vetores de tamanhos únicos de forma que fiquem aptos a serem inseridos nas próximas camadas e, principalmente, na camada de LSTM. A seguir, é adicionada uma camada de *Dropout*, responsável por desligar algumas unidades, como neurônios, da rede neural fazendo com que, dessa forma, seja evitada a criação de co-dependências entre os neurônios prevenindo a ocorrência de *overfitting*, ou seja, prevenindo que o modelo se ajuste demasiadamente ao conjunto de dados de teste a ponto de se tornar incapaz de realizar corretamente a predição para novas entradas. Na sequência, é adicionada a camada de LSTM, que será criada com a quantidade de unidades recebida pelo parâmetro da função, com *dropout* interno igual a 0.2 (20%) e *dropout* recorrente igual a 0.2 (20%). Por último é adicionada uma camada de *output* que consiste em uma camada de rede neural densa com n neurônios, onde cada neurônio representa uma classe da variável *target*. Essa última camada é criada com a função de ativação Softmax, que faz com que o *output* do modelo seja um vetor de probabilidades, onde, a classe com maior probabilidade é escolhida como *output* para o texto de entrada.

3.5.1 Camada de Embedding

Como dito anteriormente, a primeira camada do modelo consiste em uma camada de *embedding* que é responsável por receber os textos transformados pelo word2vec e inseri-los dentro do modelo, no formato correto. Essa camada é recebida através do parâmetro da função de criação do modelo que recebe o modelo pré-treinado de *word embeddings*, ou seja, o word2vec, que é criado de acordo com o mostrado pela Figura 17. Caso nenhum modelo de word2vec seja enviado para a classe durante a criação, a função w2v é chamada

para criar e treinar o modelo word2vec, recebendo como parâmetros o *dataset*, a coluna que contém os textos e o parâmetro max_len que determina, neste caso, o tamanho das dimensões de treinamento do modelo word2vec. O atributo w2v_model é utilizado posteriormente na chamada da função create_model, podendo ele ter sido criado pela função w2v conforme descrito acima ou recebido como parâmetro na criação da instância.

Figura 17 – Criação da camada de embedding.

```
# Creating our Word2Vec model if it not exists
if self.w2v_model is None:
    self.w2v_model = w2v(self.data, column=self.text, max_len=self.max_len)
```

Fonte: Elaborado pelo autor.

3.5.2 Camada de Dropout

A camada de *Dropout* é responsável por realizar o 'desligamento' aleatório de algumas unidades, por exemplo neurônios, de uma rede neural fazendo, dessa forma, com que durante o aprendizado cada neurônio aprenda a trabalhar com diferentes conjuntos de neurônios. Esse procedimento vai evitar que os neurônios criem co-dependências entre si ou que determinado conjunto de neurônios assuma grandes pesos dentro da rede e se torne demasiadamente influente na tomada de decisão, o que por sua vez evita que a rede neural seja sobre-ajustada, em outras palavras, evita o acontecimento de *overfitting*.

Para introduzir o conceito de *overfitting*, deve-se primeiramente introduzir o conceito de generalização. A generalização de um modelo refere-se à capacidade de um modelo de conseguir fazer predições para conjuntos de dados não utilizados em seu treinamento, ou seja, fazer predições corretas mesmo para dados nunca vistos. O *Overfitting*, juntamente com o *underfitting*, é uma deficiência criada em uma rede neural ou algoritmo de *machine learning* durante o aprendizado, que torna o modelo não genérico. No caso do *overfitting*, o modelo se ajusta demasiadamente aos dados de treinamento que não é capaz de fazer predições corretas para casos nunca vistos. Algumas técnicas podem ser utilizadas para evitar o *overfitting*, elas são conhecidas como técnicas de regularização. O *Dropout* é apenas uma dessas técnicas.

3.5.3 Camada LSTM

A camada de LSTM é o coração dessa estrutura de rede neural. É nessa camada que a principal parte do aprendizado vai acontecer. Como explicado anteriormente, as LSTMs possuem uma grande capacidade de tratar dados sequenciais, inclusive sequências de textos, utilizando uma estrutura de memorização para capturar interdependências entre os dados de uma sequência. Essa camada receberá os textos de entrada da camada de *embedding* e vai,

a partir do treinamento, aprender a classificá-los corretamente entre as classes pré-definidas. Importante notar que nessa camada também é inserida a técnica de *dropout* que acontecerá em neurônios diferentes daqueles desligados pela camada de *dropout*, isso também vai ajudar a evitar *overfitting*. Além disso foi adicionado o *dropout* recorrente que realizará o *dropout* nos estados recorrentes da LSTM.

3.5.4 Camada densa

A última camada, a camada densa, também conhecida como camada de *output*, é a camada responsável por ajustar o aprendizado das camadas anteriores para um formato que seja compreensível, neste caso, utilizando a função Softmax, a camada de *output* terá como saída os dados formatados em um vetor de C dimensões, cada qual correspondendo a uma classe dentre todas as diferentes classes da nossa variável *target* cujos valores representam a probabilidade de que aquele determinado *input* pertença a cada classe. No caso da análise de sentimentos, por exemplo, tem-se um vetor de probabilidades de 3 dimensões, uma representando a classe 'positivo', outra representando 'negativo' e a última representando 'neutro', não necessariamente nessa ordem. A Figura 18 mostra o resumo da estrutura completa do modelo, feita a partir da função summary() disponibilizada pela biblioteca Keras.

Figura 18 – Resumo da estrutura do modelo.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          1636300
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 100)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 404
=================================================================
Total params: 1,717,104
Trainable params: 80,804
Non-trainable params: 1,636,300
```

Fonte: Elaborado pelo autor.

3.6 O método classification

O método 'classification', foi implementado na classe principal, para ser responsável por realizar todos os principais procedimentos desse trabalho. A partir dele, os métodos de pre-processamento dos dados são chamados, o modelo de word2vec é criado e treinado e por fim o modelo da rede neural é criado e treinado com os textos já tratados, usando como parâmetro

para a camada de *embedding* o modelo treinado de word2vec. Para usar este método, deve-se primeiro criar uma instância da classe principal, enviando os parâmetros necessários: o *dataset*, o nome da coluna onde estão os textos, o nome da coluna onde estão as classes corretas, para que seja feito o treinamento e o nome do arquivo onde o modelo treinado será salvo. Criada a instância, basta chamar o método e todos os procedimentos acima descritos serão realizados, retornando o modelo treinado que pode ser usado para fazer predições.

Como mostrado pela Figura 19 , a etapa 1 do método classification consiste em realizar as chamadas para os métodos de preprocessamento dos dados. Em primeiro lugar, caso nenhum dicionário de stopwords tenha sido enviado como parâmetro, o dicionário padrão da biblioteca NLTK é atribuído ao atributo stop_words. Em seguida, os dados são enviados para a função pipeline, que irá executar os métodos de preprocessamento em sequência na coluna onde estão localizados os textos. A Figura 20 mostra a função pipeline. Após tratados os textos, o *dataset* é dividido entre treino e teste, assim, pode-se treinar o modelo nos dados de treino e depois verificar se o modelo está generalizado o suficiente ao aplicá-lo nos dados de teste e comparar os resultados da predição com os verdadeiros valores. Em seguida, os dados de treino passam pelo processo de *oversampling*, garantindo que todas as classes terão igual representação dentro dos dados, durante o treinamento. A criação e treinamento do modelo de word2vec finaliza a etapa 1 do método e é importante notar que esse modelo é treinado em cima dos dados originais, sem que haja influência do *oversampling* nesta etapa.

Figura 19 – Método classification - Etapa 1.

```
def classification(self):
    """ Returns a classification model, trained with 'text' inputs and 'target' labels in 'data' file
        also, saves the model with the 'output filename'.
    """

    # Clean texts
    if self.stop_words is None:
        from nltk.corpus import stopwords
        self.stop_words = stopwords.words('english')

    # pipeline will execute all preprocessing processes:
        # create_emoticons_tags, create_url_tags, create_time_tags, clean_texts
    self.data = pipeline(self.data, column=self.text, stop_words=self.stop_words)

    # Oversampling the data to fix class imbalance and spliting into train/test
    df_train, df_test = train_test_split(self.data, test_size=0.3, random_state=52)
    df_train = oversampling(df_train, column=self.target)


    # Creating our Word2Vec model if it not exists
    if self.w2v_model is None:
        self.w2v_model = w2v(self.data, column=self.text, max_len=self.max_len)
```

Fonte: Elaborado pelo autor.

A etapa 2 da função classification, mostrada na Figura 21, finaliza a preparação dos dados para que finalmente possam ser inseridos no modelo. Primeiramente usa-se o método Tokenizer, disponibilizado pela biblioteca Keras para quebrar cada texto em um vetor de palavras. Em seguida, utiliza-se o atributo word_index do método Tokenizer para encontrar o tamanho do vocabulário que será utilizado posteriormente para a criação da matriz de

Figura 20 – Função pipeline.

```
def pipeline(df: pd.DataFrame, column: str, stop_words):
    """ Executes all pre-processing methods. """

    print(df[column])
    df[column] = create_emoticons_tags(df[column])
    df[column] = create_url_tags(df[column])
    df[column] = create_time_tags(df[column])
    df[column] = clean_texts(df[column], stop_words)

    return df
```

Fonte: Elaborado pelo autor.

embedding. Por fim, as variáveis x e y são criadas. A variável x recebe os textos de *input*, retornados após um procedimento de *padding*, que irá garantir que todos os vetores de entrada tenham o mesmo tamanho. Esse tamanho é recebido através do atributo max_len, se max_len for igual a 100, então todos os vetores de textos terão tamanho igual a 100, isso é necessário para que os dados sejam inseridos corretamente no modelo.

Como no *dataset* enviado para a classe, as classes alvo podem ser do tipo *string*, foi utilizado o método LabelEncoder, disponibilizado pela biblioteca ScikitLearn, para codificar as classes em valores entre 0 e a quantidade de classes -1. Para as classes 'negativo', 'neutro' e 'positivo', o LabelEncoder irá retornar 0 sempre que encontrar a classe 'negativo, 1 sempre que encontrar 'neutro' e 2 sempre que encontrar 'positivo', por exemplo. Isso garante que cada classe seja representada por um valor numérico. Em seguida, foi utilizado o método to_categorial disponibilizado pela biblioteca Keras, para transformar os dados categóricos, embora numéricos, em vetores *one hot encoded*. Vetores onde cada dimensão representa uma classe e somente a classe atribuída recebe valor 1, usando o exemplo anterior, as entradas com classe 'negativo' teriam na variável y o vetor [1,0,0], as entradas com classe 'neutro', [0,1,0] e as entradas com classe 'positivo', [0,0,1].

Figura 21 – Método classification - Etapa 2.

```
# Tokenizing
self.tokenizer.fit_on_texts(self.data[self.text])
vocab_size = len(self.tokenizer.word_index) + 1

# Padding texts to create our input data X
x_train = pad_sequences(self.tokenizer.texts_to_sequences(df_train[self.text]), maxlen=self.max_len)
x_test = pad_sequences(self.tokenizer.texts_to_sequences(df_test[self.text]), maxlen=self.max_len)

# Creating the encoder to set our Y in categorical format
encoder = LabelEncoder()
encoder.fit(self.data[self.target].tolist())
y_encoded_tr = encoder.transform(df_train[self.target].tolist())
y_train = np_utils.to_categorical(y_encoded_tr)
y_encoded_test = encoder.transform(df_test[self.target].tolist())
y_test = np_utils.to_categorical(y_encoded_test)
```

Fonte: Elaborado pelo autor.

A etapa 3 do método classification é responsável por criar e treinar o modelo proposto. Inicialmente, a camada de *embedding* é criada a partir do modelo de word2vec previamente treinado. Para isso deve-se criar uma matriz quadrada de pesos com número de dimensão igual ao tamanho do vocabulário. Essa matriz irá guardar todas as características de cada palavra conforme descrito anteriormente. Em seguida, caso nenhum modelo tenha sido previamente fornecido, um novo modelo é criado e treinado usando as especificações determinadas na criação da instância. A seguir, com o modelo treinado, é feita uma avaliação do modelo a partir do método make_evaluation, que irá mostrar os resultados para as métricas de avaliação para o modelo, conforme mostrado pela Figura 22. Por fim, o modelo é salvo em um arquivo pickle, podendo ser reutilizado posteriormente.

Figura 22 – Método classification - Etapa 3.

```
# Now that we have our data preprocessed, we can split it into train/test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=52)
#x_train = x
#y_train = y
# Creating Embedding Matrix
embedding_matrix = np.zeros((vocab_size, self.max_len))
for word, i in self.tokenizer.word_index.items():
    if word in self.w2v_model.wv:
        embedding_matrix[i] = self.w2v_model.wv[word]

# Creating Embedding Layer with Embedding Matrix
embedding_layer = Embedding(vocab_size, self.max_len, weights=[embedding_matrix], input_length=self.max_len,
                            trainable=False)

# Creating the Model
if self.model is None:
    self.model = self.create_model(
        dense_neurons=self.data[self.target].value_counts().count(),
        embedding_layer=embedding_layer
    )

    # Compile and Run the model
    self.model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])
    callbacks = [ReduceLROnPlateau(monitor='loss', patience=5, cooldown=0),
                 EarlyStopping(monitor='accuracy', min_delta=1e-4, patience=5)]

    self.model.fit(x_train, y_train, batch_size=1024, epochs=100, validation_split=0.3,
                   verbose=1, callbacks=callbacks)

# Evaluating
self.make_evaluation(x_test, y_test)

# saving model to output file
filename = str(self.output_filename + '.sav')
pickle.dump(self.model, open(filename, 'wb'))
return self
```

Fonte: Elaborado pelo autor.

3.7 Treinamento

Com a estrutura de modelo criada pelo método create_model, o próximo passo é realizar o treinamento. Para isso precisa-se definir mais alguns hiperparâmetros como a função de custo para o modelo, o algoritmo de otimização, que é responsável por atualizar os vetores de peso da rede neural, afim de minimizar a função de custo. Além disso nessa etapa são inseridos alguns *callbacks*, técnicas responsáveis por monitorar o aprendizado e realizar ações para garantir

maior controle sobre o treinamento. Neste trabalho são utilizadas duas técnicas, uma para reduzir a taxa de aprendizado conforme o treinamento acontece e outra para interromper o aprendizado em um momento oportuno, retornando um resultado de minimização otimizado.

3.7.1 Função de custo

Como destacado anteriormente, a função de custo é a função que deve ser minimizada para que o modelo aprenda a classificar os dados. A função de custo compara os resultados da classificação de cada época do treinamento com os resultados verdadeiros ou, em outras palavras, compara o que a rede previu com o que realmente ela deveria ter previsto, criando um valor de custo. Minimizar esse valor significa reduzir o erro da rede e consequentemente melhorar seus resultados.

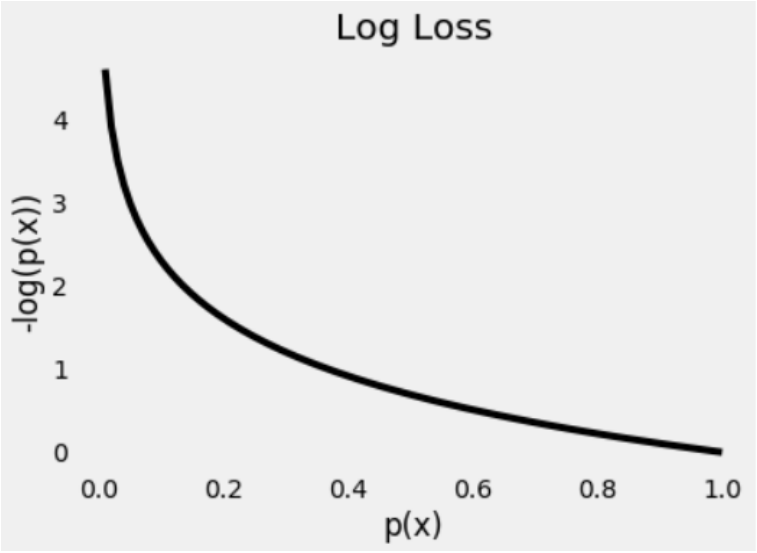
Para esse trabalho, a função de custo escolhida foi a *Binary Cross-Entropy* ou, '*log loss*'. Essa função de custo tem como base em seu funcionamento o vetor das probabilidades para cada classe. Para um problema com C classes, tem-se um vetor de C dimensões, com cada uma representando uma classe diferente.

Como explica Brownlee (2019), na etapa de treinamento de um modelo cada exemplo possui uma probabilidade igual 1.0 para a classe conhecida, ou seja, para a classe correta, e uma probabilidade 0.0 para todas as outras classes, ou seja, para as classes incorretas. Além disso, tem-se que um modelo, nesse caso um modelo de rede neural, pode prever as probabilidades de determinada entrada pertencer a cada classe. Deste modo, a entropia cruzada pode ser utilizada para calcular a diferença entre as duas distribuições de probabilidade. De forma mais visual, pode-se ver na Figura 23 a demonstração do gráfico da função *log loss* para as probabilidades de 0 a 1 preditas para uma classe correta. A função do *log loss* é dada pelo negativo do logaritmo da probabilidade ($-log(p(x))$), é possível observar no gráfico que, conforme a probabilidade de uma entrada correta se aproxima de 1, o custo se aproxima de 0 e de modo contrário, quando a probabilidade de uma entrada correta se aproxima de 0, o custo cresce exponencialmente. Essa função de custo foi escolhida por se encaixar bem no problema, além de ser uma função bastante utilizada.

3.7.2 Algoritmo de otimização

De acordo com Géron (2017), o aprendizado de uma rede neural se deve resumidamente ao mecanismo da retro-propagação, *Backpropagation*. A cada época, ou passo, a rede neural é alimentada com um *input* de dados e então cada um dos neurônios de cada uma das suas camadas calcula seu respectivo *output* resultando, ao final do passo, em uma previsão. Esse procedimento é conhecido como 'passo para frente', o *forward pass*. Então, através de uma função de custo a rede calcula o erro para aquele passo e o quanto cada neurônio da rede contribuiu para a rede, começando pela última camada, até chegar na camada de *input*. Essa

Figura 23 – Gráfico da função log loss.



Fonte: Elaborado pelo autor.

etapa calcula o gradiente de erro para cada neurônio e, por fim, através do método do gradiente descendente, os pesos de cada neurônio são atualizados, caracterizando o aprendizado. De outro modo, pode-se dizer que o aprendizado de uma rede neural consiste em obter os pesos ótimos para cada neurônio da rede que irão minimizar a função de custo. O método da descida estocástica do gradiente é o mais simples dos métodos de otimização e foi a partir dele que surgiram diversos outros métodos que procuram de maneira primordial alcançar o mínimo da função de erro de forma mais rápida, além de evitar os problemas de explosão ou desaparecimento do gradiente, que ocorrem quando o gradiente assume valores maiores que 1.0 ou muito próximos de 0.0, respectivamente.

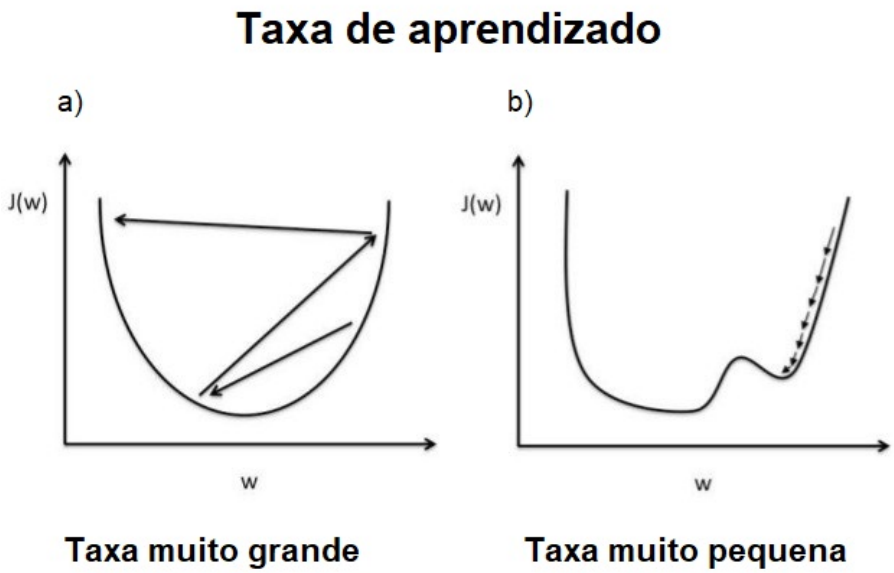
Segundo Brownlee (2017), o Adam, algoritmo de otimização utilizado neste projeto, deriva da combinação das vantagens de dois outros métodos de otimização derivados da decida estocástica do gradiente, o AdaGrad (*Adaptive Gradient Algorithm*) que funciona bem com gradientes esparsos e o RMSProp (*Root Mean Square Propagation*) que funciona bem para problemas online e não estacionários. "O Adam utiliza o primeiro e o segundo momento médio dos gradientes, calculando dessa forma uma média móvel exponencial do gradiente e do gradiente quadrado e com parâmetros beta1 e beta2 controlando essas médias." (Brownlee 2017, tradução nossa). O algoritmo Adam é popularmente conhecido por ter ótimos resultados nos treinamentos de redes neurais e por esse motivo foi escolhido para este trabalho.

3.7.3 Taxa de aprendizado

Em *machine learning* e redes neurais, a taxa de aprendizado refere-se a quantidade em que os pesos são ajustados durante o treinamento, ou ainda, ao tamanho do passo realizado na descida do gradiente. Deve-se ter atenção ao fato de que, algumas vezes, quando o tamanho

do passo for demasiadamente grande, o algoritmo pode demorar muito para convergir pois fica ultrapassando diversas vezes o ponto de mínimo, como mostrado pelo item a) da Figura 24, enquanto que, quando o tamanho do passo for demasiadamente pequeno, o algoritmo pode demorar muito para convergir, podendo inclusive ficar presa em um ponto de mínimo local, como mostrado pelo item b) da Figura 24.

Figura 24 – Taxas de aprendizado irregulares.



Fonte: Elaborado pelo autor.

Para evitar esse problema, foi utilizado nesse trabalho uma técnica de monitoramento do aprendizado, cujo objetivo é reduzir a taxa de aprendizado aos poucos, de modo que o algoritmo possa chegar ao ponto de mínimo de forma otimizada, esse método é fornecido pela biblioteca Keras através da função de callback *ReduceLROnPlateau* que irá reduzir a taxa de aprendizado quando uma métrica determinada parar de melhorar.

3.7.4 Parada antecipada

Em redes neurais e *machine learning*, a parada antecipada (do inglês *Early Stopping*) é uma forma de regularização utilizada para que não aconteçam ajustes excessivos, parando o aprendizado em um momento oportuno e evitando *overfitting*. De forma similar ao método anterior, o *early stopping* irá checar, a cada iteração, se a métrica monitorada deixou de melhorar e caso essa resposta seja positiva, a aprendizagem é parada. A biblioteca Keras também oferece essa função como um método de *callback* através da função 'EarlyStopping', que também foi utilizado nesse trabalho para que seja possível obter um bom resultado.

3.8 Avaliando o modelo

Para entender melhor o funcionamento das métricas de avaliação, é suposto um problema onde textos são classificados como positivos ou negativos, após a classificação feita pelo modelo, tem-se a matriz de confusão, mostrada na Figura 25. Verdadeiro Positivo(VP), refere-se a quando o modelo previu positivo e o texto era realmente positivo, Verdadeiro Negativo(VN), refere-se a quando o modelo previu negativo e o texto realmente era negativo, Falso Positivo(FP) refere-se à quando o modelo previu positivo e o texto não era positivo e por fim, Falso Negativo(FN) refere-se a quando o modelo previu negativo e o texto não era negativo. Tendo como base esse exemplo, pode-se destacar as quatro principais métricas para avaliar um modelo. São elas: a Acurácia, a Precisão, a Revocação e o *F-Score*. A Acurácia mede a quantidade, em porcentagem, que o modelo conseguiu acertar, dentre todas as predições feitas, sua fórmula é dada por:

$$\frac{VP + VN}{VP + VN + FP + FN}$$

Já para a precisão, pode-se dizer que mede quantos foram classificados como positivos dentre os que realmente era positivos. Sua formula é dada por:

$$\frac{VP}{VP + FP}$$

A Revocação retorna a frequência que o classificador tem escolhido determinada classe. Sua formula é dada por:

$$\frac{VP}{VP + FN}$$

Já o *F-Score* busca combinar Revocação e Precisão para que tragam um único valor, afim de obter uma métrica equilibrada entre falsos positivos e falsos negativos. Sua formula é dada por:

$$\frac{2 * Precisão * Revocação}{Precisão + Revocação}$$

Figura 25 – Matriz de Confusão.

		Valor predito	
		Positivo	Negativo
Valor verdadeiro	Positivo	Verdadeiro Positivo	Falso Negativo
	Negativo	Falso Positivo	Verdadeiro Negativo

Fonte: Elaborado pelo autor.

A Figura 26 mostra a função responsável por realizar a avaliação do modelo, retornando em tela o resultado para o *F-Score* e os resultados para algumas das principais métricas, além de mostrar a função ohe, utilizada para que o *output* da rede neural, dada como um vetor de probabilidades seja colocado no formato *one-hot*, onde o index do vetor correspondente à classe com maior probabilidade recebe o valor 1 enquanto os outros recebem 0. Para esse projeto, faz bastante sentido avaliar o *F-Score*, uma vez que busca-se diminuir os erros, independente do tipo.

Figura 26 – Função para avaliação do modelo.

```
@staticmethod
def ohe(score):
    """ Returns One Hot Encoding of score """
    index = score.argmax()
    vector = np.zeros(len(score), dtype=int)
    vector[index] = 1
    return vector

def make_evaluation(self, x_test, y_test):
    #ev_score = self.model.evaluate(x_test, y_test, batch_size=1024)
    # print("Accuracy = ", ev_score[1])
    # print("\nLoss = ", ev_score[0])

    f1_y = np.array([self.ohe(y) for y in self.model.predict(x_test)])
    print('F1-Score:',
          f1_score(y_test, f1_y, average='micro'))

    print('Precision, Recall, F-Score, Support',
          precision_recall_fscore_support(y_test, f1_y, average='micro'))
```

Fonte: Elaborado pelo autor.

3.9 Predições

Para realizar as predições com o modelo já treinado, foi implementada uma função, make_prediction, e uma função de apoio, decode_class. A função make_prediction recebe um texto, e então o ajusta para que corresponda ao formato correto. Em seguida pede ao modelo que classifique aquele texto, retornando a classificação na variável 'score'. Após isso, a ordem alfabética das classes é coletada pela variável label_order e enviada junto com a predição para a função decode_class que retornará a *string* correspondente à classe predita pelo modelo. Dessa forma pode-se enviar uma frase e obter como resultado a *string* 'negativo', para o caso da análise de sentimentos, por exemplo. A Figura 27 mostra as funções necessárias para realizar uma predição.

Figura 27 – Funções para realizar uma predição.

```
@staticmethod
def decode_class(score, label_order: list):
    return list(dict(label_order).keys())[score.argmax()]

def make_prediction(self, sample: str):
    # start_at = time.time()

    # Tokenize text
    x_test = pad_sequences(self.tokenizer.texts_to_sequences([sample]), maxlen=self.max_len)

    # Predict
    score = self.model.predict([x_test])[0]

    label_order = self.data[self.target].value_counts().sort_index()

    # Decode sentiment
    label = self.decode_class(score, label_order)
    print('Predicted: ', label)

    return label
```

Fonte: Elaborado pelo autor.

4 Resultados

Nesse capítulo são mostrados os resultados obtidos ao aplicar o modelo na base de dados escolhida. Os parâmetros escolhidos foram os parâmetros padrão para cada variável. A Figura 28 mostra como é feita a chamada para o modelo onde primeiro cria-se a instância, enviando como parâmetros o conjunto de dados, o nome da coluna onde estão os textos, o nome da coluna *target* onde estão as classes e o nome do arquivo, para que o modelo seja salvo. Criada a instância, basta fazer uma chamada à função principal (classification), para que realize todos os procedimentos necessários.

Figura 28 – Funções para realizar uma predição.

```
sent_model = MulticlassClassification(data=df, text='text', target='sentiment',
                                      output_filename='sent_model')
sent_model = sent_model.classification()
```

Fonte: Elaborado pelo autor.

Para que se possa aplicar as métricas de avaliação, a base de dados com 14640 tweets, sendo desses 9178 negativos, 3099 neutros e 2363 positivos foi dividida em treino e teste, sendo 70% dos dados para treino e 30% para teste. Na etapa do treinamento, os dados de teste são novamente divididos, separando 20% desses dados para validação. Desse modo é possível fazer validações iniciais durante a aprendizagem do modelo.

O método para a análise de sentimentos foi treinado e em seguida avaliado pelo método 'make_evaluation' e o Quadro 1 mostra os resultados em cada métrica na etapa de teste. Além disso, são mostrados os valores da acurácia do modelo para os dados de treino e validação.

Quadro 1 – Modelo de análise de sentimentos.

	Acurácia	**Precisão**	**Revocação**	**F1-Score**
Treino	93,78	-	-	-
Validação	90,27	-	-	-
Teste	83,53	75,02	75,02	75,02

Fonte: Elaborado pelo autor.

O método para a classificação multiclasse de textos foi treinado e em seguida avaliado pelo método 'make_evaluation' e o Quadro 2 mostra os resultados em cada métrica na etapa de teste. Além disso, são mostrados os valores da acurácia do modelo para os dados de treino e validação.

A Figura 29 mostra o resultado da análise de sentimentos para alguns exemplos, enquanto a Figura 30 mostra o resultado da classificação multiclasse para alguns exemplos.

Figura 29 – Resultados Análise de Sentimento.

```
sent_model.make_prediction("such a bad flight.. worst airline")
```

```
Predicted:  negative
'negative'
```

```
sent_model.make_prediction("I lose my bag, when i found it, it was broken.. so disrespectfull")
```

```
Predicted:  negative
'negative'
```

```
sent_model.make_prediction("I love you, wonderful flight")
```

```
Predicted:  positive
'positive'
```

```
sent_model.make_prediction("the cat is black")
```

```
Predicted:  neutral
'neutral'
```

```
sent_model.make_prediction("JetBlue gave me bad service. My media console was spoilt")
```

```
Predicted:  negative
'negative'
```

```
sent_model.make_prediction("JetBlue was okay, nothing special")
```

```
Predicted:  neutral
```

Fonte: Elaborado pelo autor.

Figura 30 – Resultados Classificação Multiclasse.

```
reason_model.make_prediction("such a bad flight.. worst airline")
```

```
Predicted:  Can't Tell
"Can't Tell"
```

```
reason_model.make_prediction("I lose my bag, when i found it, it was broken.. so disrespectfull")
```

```
Predicted:  Lost Luggage
'Lost Luggage'
```

```
reason_model.make_prediction("JetBlue gave me bad service. My media console was spoilt")
```

```
Predicted:  Customer Service Issue
'Customer Service Issue'
```

```
reason_model.make_prediction("My flight was canceled, I'm desperate")
```

```
Predicted:  Bad Flight
'Bad Flight'
```

Fonte: Elaborado pelo autor.

Quadro 2 – Modelo de classificação multiclasse.

	Acurácia	**Precisão**	**Revocação**	**F1-Score**
Treino	98,01	-	-	-
Validação	8179	-	-	-
Teste	92,38	59,47	59,47	59,47

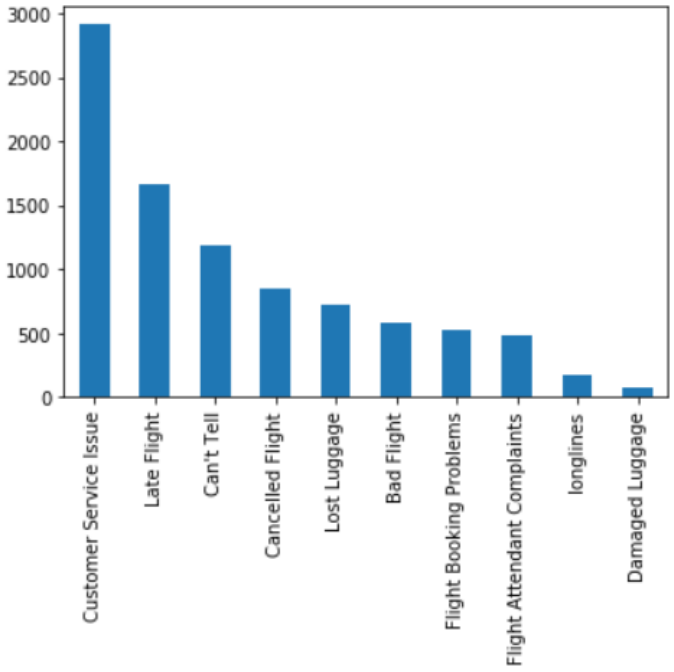
Fonte: Elaborado pelo autor.

5 Conclusão

Após fazer alguns testes e avaliar os modelos, pode-se concluir que o modelo para a análise de sentimentos teve bons resultados, acertando de forma coerente os exemplos testados, além de obter bons resultados em nossas métricas. Porém, ainda há espaço para melhorias. O uso de técnicas como *POS-Tagging*, para capturar maiores informações sobre as palavras ao identificá-las como verbos, pronomes, adjetivos, etc. pode trazer um nível maior de assertividade, enquanto o uso de técnicas de validação cruzada podem otimizar o modelo. Além disso, aumentar o número de exemplos poderia ser uma etapa bastante importante

Por outro lado, o modelo de classificação multiclasse não se saiu tão bem. Apesar de alguns acertos e a acurácia ser bem alta, o restante das métricas retornaram valores baixos e isso se deve principalmente ao pouco número de exemplos, uma vez que os 9178 exemplos negativos estão divididos em 10 diferentes classes. A Figura 31 mostra a distribuição dos textos entre as classes. É possível notar um valor muito baixo de exemplos, menos de 1000, para pelo menos 7 das 10 classes. Isso torna difícil a aprendizagem do modelo, tendo em vista que esses dados ainda foram divididos entre treino e teste. Os métodos adicionais descritos para a análise de sentimentos também poderiam ajudar esse segundo modelo, mas a adição de mais dados é crucial.

Figura 31 – Resultados Classificação Multiclasse.



Fonte: Elaborado pelo autor.

Referências

AGARWAL, Apoorv et al. Sentiment Analysis of Twitter Data. Nova York, 2011, p. 1-3.

ALY, Mohamed. Survey on Multiclass Classification Methods. .Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.175.107rep=rep1type=pdf>. Acesso em: 26 Fev. 2020.

BROWNLEE, Jason. Deep Learning for Natural Language Processing, 2017.

CASTRO, Leandro N. de; VON ZUBEN, Fernando J.. Redes Neurais Artificiais. In: VON ZUBEN, Fernando J.; VON ZUBEN, Fernando J.. Computação Natural. São Paulo: Unicamp. Cap. 5.

DATA SCIENCE ACADEMY – Deep Learning e a Tempestade Perfeita. Deep Learning Book, 2020. Disponível em: <http://deeplearningbook.com.br/> Acesso em: 20, Mai. 2020.

GOLDBERG, Yoav.Neural Network Methods for Natural Language Processing. 2019.

GOODFELLOW, Ian; BENGIO Yoshua, COURVILLE Aaron. Deep Learning. MIT Press, 2016.

KARANI, Dhruvil. Introduction to Word Embedding and Word2Vec, 2018. Disponível em: <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>. Acesso em: 18, Mai. 2020.

KROSE, B.; SMAGT, P. van der. An introduction to neural networks. 1996.

METHA, Nick. The Essential Guide to Customer Success Disponível em: <https://www.gainsight.com/guides/the-essential-guide-to-customer-success/>. Acesso em: 18, Mai. 2020.

NICHOLSON, Chris. A Beginner's Guide to LSTMs and Recurrent Neural Networks, 2020. Disponível em: <https://pathmind.com/wiki/lstm> Acesso em: 20, Mai. 2020.

OLAH, Christopher. Understanding LSTM Networks, 2015. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> Acesso em: 26, Fev. 2020.

PANG, Bo; LEE, Lillian. Opinion Mining and Sentiment Analysis. Boston, 2008.

RAUBER, T. W. Redes neurais artificiais. Vitória, 1996.

RODRIGUES, Jéssica. O que é o Processamento de Linguagem Natural? 2017. Disponível em:<https://medium.com/botsbrasil/o-que-%C3%A9-o-processamento-de-linguagem-natural-49ece9371cff> Acesso em: 26 Fev. 2020.

ROHRER, B. 1 Video (26 min). Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM). Publicado pelo canal Brandon Rohrer, 2017. Disponível em <https://www.youtube.com/watch?v=WCUNPb-5EYIt=1348s>. Acesso em: 25 mai 2020.

SONI, Devin; Dealing with Imbalanced Classes in Machine Learning. Disponível em: <https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learning-d43d6fa19d2>. Acesso em: 26 Fev. 2020.

TWITTER, Inc. New User FAQ, Twitter, 2020. Disponível em: <https://help.twitter.com/en/new-user-faq>. Acesso em: 26 Fev. 2020.

TWITTER, Inc. Standard Search API. Twitter, 2020. Disponível em: <https://developer.twitter.com/en/docs/tweets/search/overview/standard>. Acesso em: 26 Fev. 2020.

WANG, Jenq-Haur; LUO Xiong; LIU Ting-Wei; WANG Long. An LSTM Approach to Short Text Sentiment Classification with Word Embeddings. Taipei, 2018.

YSE, Diego lopez. Your Guide to Natural Language Processing (NLP), 2020. Disponível em: <https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1>. Acesso em: 20, Mai. 2020.