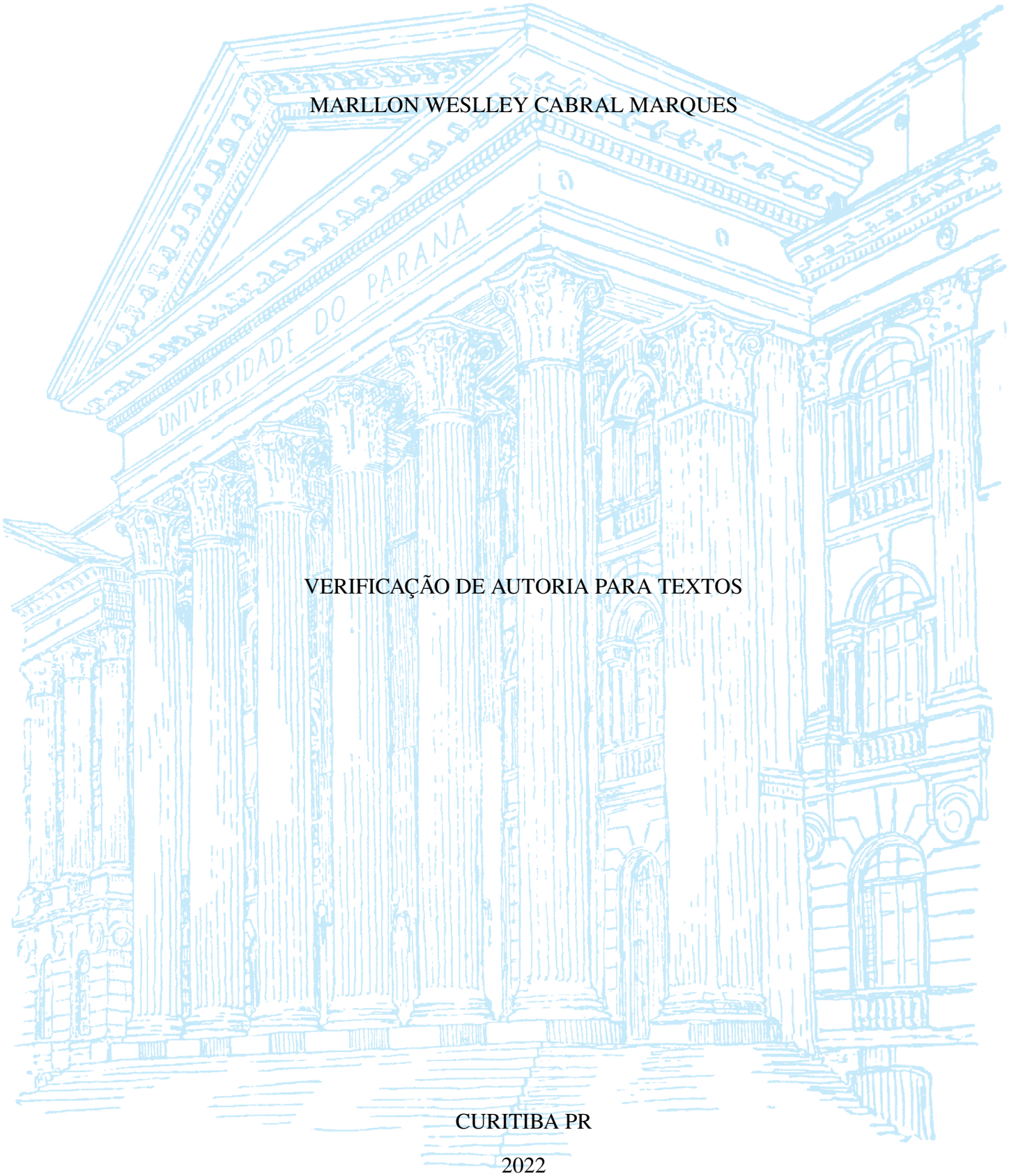
UNIVERSIDADE FEDERAL DO PARANÁ

MARLLON WESLLEY CABRAL MARQUES



VERIFICAÇÃO DE AUTORIA PARA TEXTOS

CURITIBA PR

2022

MARLLON WESLLEY CABRAL MARQUES

VERIFICAÇÃO DE AUTORIA PARA TEXTOS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Fabiano Silva.

Coorientador: Adelaide H. Pescatori Silva.

CURITIBA PR

2022

A minha mãe, que me deu suporte durante toda essa jornada.

AGRADECIMENTOS

Aos meus orientadores, Fabiano Silva e Adelaide H. Pescatori Silva, por todos os conselhos, sugestões e conhecimentos fornecidos durante este trabalho.

Aos professores do departamento de informática da UFPR, por terem ajudado a construir todo o conhecimento que obtive durante a graduação. Um agradecimento em especial para o professor Eduardo Spinosa, suas aulas animadas e forma de promover o aprendizado me fizeram enxergar novos horizontes do conhecimento.

Aos amigos que estiveram presentes durante toda a jornada da minha graduação, os quais escutaram todas as minhas reclamações, fardos e alegrias. Um agradecimento especial para o grupo que tive a felicidade de conhecer no primeiro dia de aula e me acompanharam até o último dia, obrigado Rafael, Lucas, Douglas, Brendon e Andrey. Sou muito grato também a todos os meus amigos que não foram citados aqui, vocês foram de grande importância para conclusão da minha graduação e desse trabalho, agradeço de coração pelo suporte de cada um.

À minha psicóloga, Laíse, sem sua ajuda e ensinamentos para organização, este trabalho nunca teria sido finalizado. Obrigado por ser uma profissional incrível.

Aos meus tios, Ednaldo e Edson, por terem dado suporte em diversos momentos da minha vida, junto a minha mãe, eles colaboraram de forma essencial para o meu desenvolvimento e percurso de vida.

À minha mãe, Davenilda Ribeiro Cabral, ou apenas Nilda como a maioria das pessoas a conhecem, agradeço por todos os incentivos, motivações, todo amor e crença em minhas habilidades que você me forneceu ao longo dos anos, ela é o maior motivo por eu ter conseguido chegar até aqui.

Por fim, agradeço a todos que de alguma forma contribuíram na minha caminhada até o presente momento, principalmente para os que forneceram incentivos e acreditaram na minha capacidade de fazer as coisas e chegar até onde cheguei.

RESUMO

Processamento de linguagem natural é uma área de grande interesse para a computação, principalmente pela grande variedade de aplicações para o cotidiano humano. Neste trabalho é criado um *dataset* com dados em português brasileiro para verificação de autoria de textos. Os dados utilizados foram retirados de obras brasileiras de romance dos escritores Machado de Assis e José de Alencar. O *dataset* criado contribui para o ambiente de NLP brasileiro, já que, a escassez de dados em português é uma barreira para realização de trabalhos com o idioma português como foco. O processo de criação do *dataset* é realizado com etapas manuais e automatizadas, de forma que o processo manual tem foco em descobrir características especiais de cada texto, enquanto o processo automatizado tem foco em tratar o texto utilizando das características especiais obtidas no processo manual. Utilizando três variações de configuração para rede neural LSTM, sendo essas configurações: otimizada, tradicional, e com *embeddings* pré-treinados. Como uma forma de verificação da veracidade obtida pela classificação feita pela rede neural, foi criada uma variação do *dataset* com uma classe de ruído com dados pegos de páginas escolhidas de forma aleatória no site do Wikipédia. Além do *dataset* com ruído, foi criado um *dataset* especialmente para testes de generalização, onde os dados foram pegos de obras diferentes das que foram utilizadas para treinamento, dessa forma foi possível validar a capacidade de generalização do aprendizado dos modelos.

Palavras-chave: LSTM. Processamento de Linguagem Natural. Dataset.

ABSTRACT

Natural language processing is an area of great interest for computer science, mainly because of its wide variety of applications for everyday situations. In this work, a dataset with data in Brazilian Portuguese is created to verify the authorship of texts. The data used were taken from Brazilian novels by the writers Machado de Assis and José de Alencar. The dataset created contributes to the Brazilian NLP environment, since the scarcity of data in Portuguese is a barrier to carrying out works with the Portuguese language as a focus. The dataset creation process is carried out with manual and automated steps, the manual process focuses on discovering special characteristics of each text, while the automated process focuses on treating the text using the special characteristics obtained in the manual process. Three configuration variations for LSTM neural network were used: optimized, traditional, and with pre-trained embeddings. As a way of verifying the veracity obtained by the classification made by the neural network, a variation of the dataset was created with a noise class with data taken from randomly chosen pages on the Wikipedia site. In addition to the noisy dataset, a dataset was created especially for generalization tests, where the data was taken from works other than those used for training, in this way it was possible to validate the generalization capacity of the learning of the models.

Keywords: LSTM. Natural language processing. Dataset.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

CNN	Convolutional Neural Network
LSTM	Long short-term memory
RNN	Recurrent neural network
API	Application Programming Interface
GPU	Graphics processing unit
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous bag of words
IA	Inteligência Artificial
cuDNN	CUDA Deep Neural Network
NILC	Núcleo Interinstitucional de Linguística Computacional
DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Este capítulo visa informar ao leitor sobre alguns aspectos desse trabalho, fornecendo informações chave sobre o que será encontrado ao decorrer da leitura.

1.1 PROPOSTA

Este trabalho tem como principal objetivo a construção de um *dataset* utilizando de textos de acesso público em português-brasileiro, focado em verificar se uma rede neural LSTM simples é capaz de identificar corretamente o autor de cada texto. A análise da capacidade de verificação é mensurada por meio dos resultados de classificação individual por autoria presentes no *dataset*.

1.2 DESAFIOS

O desenvolvimento deste trabalho contou com os seguintes desafios:

- Criação do *dataset* em português para análise de autoria;
- Analise manual dos textos;
- Analise dos resultados;
- Verificação de funcionalidade de uma rede neural LSTM para classificação de textos em português.

1.3 MOTIVAÇÃO

A área de processamento de linguagem natural é vasta e com diversos desafios a serem trabalhados (Khurana et al., 2022). Atualmente, como um dos desafios a serem trabalhados temos a área de identificação de autoria.

Dentre as aplicações para os estudos de identificação de autoria temos a mais famosa, verificação e detecção de plágio em textos. Apesar da detecção de plágio ser a mais famosa dentre as aplicações para a área de determinação de autoria de textos, pode ser dado como exemplo uma outra aplicação que não é tão conhecida pelo público geral, essa aplicação é dentro da área de criminalística com técnicas de linguística forense. Um bom exemplo de aplicação dentro da criminalística pode ser o caso descrito por Sousa, em que são aplicados os conhecimentos de linguística para se determinar a autoria de um bilhete de confissão de um crime (Sousa-Silva (2020)).

A motivação para realização deste trabalho está relacionada à vastidão de possibilidades para aplicações da área de processamento de linguagem natural. Técnicas de processamento de linguagem natural podem ser aplicadas para qualquer idioma humano, desde que existam dados o suficiente para treinar redes neurais para realização das tarefas desejadas.

Como comentado anteriormente, é necessário dados para se poder trabalhar com o idioma pretendido, e é nesse ponto em que surge a principal barreira para trabalhos nessa área, a escassez de dados abertos disponíveis em idiomas diferentes do inglês. Devido a existência de poucos conjuntos de dados (*datasets*) em português, viu-se a necessidade e oportunidade de

tentar criar um *dataset* em português brasileiro utilizando de obras literárias como principal fonte de dados. Normalmente os *datasets* mais comumente encontrados para uso são construídos utilizando de dados muito mais abrangentes e de fácil aquisição, como por exemplo: *tweets*, avaliação de sites, avaliação de filmes. Para dados em português, o fórum AILab[1] é o que possui a melhor visibilidade e variedade de *datasets* para uso

1.4 TRABALHOS RELACIONADOS

Nos últimos anos diversos pesquisadores focaram seus esforços em trabalhos relacionados à atribuição ou identificação de autoria de diferentes tipos de textos. Contendo trabalhos com diferentes tipos de foco, como por exemplo: testar diferentes tipos de *datasets* para diferentes tipos de redes neurais (Chen Qian, 2017), ou com o foco em testar a possibilidade de acertar se dois documentos com pouco texto foram escritos pelo mesmo autor (Koppel e Winter, 2014).

Nesta seção do texto serão apresentados alguns dos trabalhos realizados na área de identificação ou atribuição de autoria.

1.4.1 Trabalhos na área de atribuição de autoria

No trabalho feito por Chen Qian (Chen Qian, 2017) foram projetados e implementados 3 tipos de modelos para identificação de autoria e um modelo para verificação de autoria. O trabalho é focado em analisar o comportamento dos modelos propostos utilizando de diferentes tipos de *dataset* e diferentes configurações para as redes (diferentes valores para o tamanho da camada oculta, e diferentes valores para o parâmetro *regularization*[2]).

A identificação de autoria foca em uma técnica que dado um conjunto de dados de um determinado grupo de autores e um conjunto de fragmentos sem identificação de autores, se é possível identificar quem são os autores dos fragmentos fornecidos, em contra partida a verificação de autoria foca em verificar se um dado texto realmente é atribuído ao autor supostamente relacionado.

No trabalho de Koppel e Winter (Koppel e Winter, 2014), os autores focam em verificar se é possível determinar se dois textos curtos foram escritos por um mesmo autor. O trabalho consiste em utilizar textos retirados de postagens de blogs (podendo ser textos integrais, ou fragmentos), e verificar se os métodos propostos para comparação de similaridade de autoria são eficazes para textos pequenos (de no máximo 500 palavras).

No trabalho de Sanderson e Guenter (Sanderson e Guenter, 2006), os autores focam em investigar o funcionamento e a eficácia de três abordagens para atribuição de autoria para textos curtos. As abordagens analisadas foram: técnicas baseadas em *Sequence Kernal, Markov Chain, Author Unmasking*.

1.5 DECISÕES DE PROJETO

Durante o projeto foram tomadas algumas decisões para fins de funcionalidade do projeto. O maior foco do desenvolvimento foi dado a criação dos *datasets* devido a motivação do trabalho. Portanto, os modelos construídos e utilizados neste trabalho acabaram por serem um passo secundário para verificar se o *dataset* teria alguma capacidade de uso.

Foi decidido que *tokens* ou palavras seriam todos os termos que aparecem entre espaço em branco, isso também inclui pontuação. Por exemplo, o nome da cidade Rio de Janeiro, após

[1] https://forum.ailab.unb.br/t/datasets-em-portugues/251

[2] https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a

processo de tokenização ficaria “Rio” “de” “Janeiro”, gerando então três palavras. Essa decisão foi tomada com o intuito de simplificar o processo de criação de *tokens*. A decisão de não remover pontuação, palavras vazias (*stopwords*) e acentuação, se deve à possibilidade de que esses atributos pudessem ser parte do padrão de escrita de determinado autor.

Como uma forma de adição de ruído aos *datasets*, foi feita a escolha de utilização de páginas aleatórias em português do site Wikipédia[3]. Devido a pouca quantidade de tempo disponível, não foi possível utilizar outros autores para reforçar a construção do ruido nos *datasets*. Ficou decidido pela utilização do ruído nas fases de treino, teste e validação.

A decisão de filtrar e limpar manualmente os dados provindos pelas páginas do Wikipédia, se deve ao fato de que não existe uma forma de conhecer previamente todas as características especiais presentes nos textos lá escritos. Portanto, foi decidido que a forma mais rápida e eficiente de realizar esse procedimento, seria por base de ação humana.

Os dados extraídos das páginas do Wikipédia receberam menos cuidado e atenção durante o tratamento, pois, foi decidido que essas entradas serviram como uma espécie de ruído para verificar se o modelo estava realmente reconhecendo algum tipo de padrão nas sentenças e não apenas classificando de forma aleatorizada. Portanto, existem algumas entradas dessa classe que podem possuir ruídos adicionais devido a falha humana durante a seleção de conteúdo.

Usando como referência plataformas que possuem limite de caracteres (por exemplo, Twitter, campos de avaliação em sites, etc.), ficou decidido que cada sentença presente no *dataset* teria como padrão o tamanho de 100 *tokens*. Essa decisão foi tomada para não ser necessário realizar nenhum procedimento para que os vetores de sentenças possuam o mesmo tamanho.

Devido a escassez de dados obtidos por meio de livros, foi decidido que todos os elementos presentes nos *datasets* com dois autores, também estão presentes como elementos dos *datasets* com três autores, de forma que foram tomadas medidas para garantir que os dados de teste, treino e validação fossem os mesmos em ambas versões dos *datasets*. Por exemplo, a frase "AB"faz parte do *dataset* de treinamento para dois e três autores, logo, "AB"não está presente nos *datasets* de teste e nem nos de validação. E de forma análoga, se a frase "AB"está presente no *dataset* de teste, então, "AB"não está presente no *dataset* de treinamento e nem nos de validação.

A escolha dos autores José de Alencar e Machado de Assis se deu principalmente devido ao fato de que ambos possuem obras no domínio público brasileiro. Outros motivos para a escolha foram: ambos possuem obras no formato de romance, suas obras foram escritas em períodos de tempo próximos, e em um determinado período da carreira dos escritores os estilos de escrita eram similares.

1.6 CONTRIBUIÇÃO

Este trabalho realizou as seguintes contribuições

- Disponibilização de um *dataset* com material em português-brasileiro.
- Avaliação da eficácia da rede neural LSTM para atribuição de autoria em ambiente controlado.

1.7 ORGANIZAÇÃO DO DOCUMENTO

Este documento é composto por 8 capítulos.

[3]https://pt.wikipedia.org/

No capítulo 2 são fornecidas informações sobre conceitos e algumas definições que são fundamentais para este trabalho. Esse capítulo tenta fornecer informações chaves para compreensão das tecnologias usadas no trabalho, de forma que pessoas sem conhecimento prévio possam compreender este documento.

No capítulo 3 são descritas informações referentes a metodologia utilizada para a execução das fases do trabalho, assim como são descritas algumas das ferramentas utilizadas para construção deste trabalho.

No capítulo 4 são fornecidas informações sobre a construção do *dataset*. Essas informações visam tentar demonstrar o processo de tratamento de dados, decisões globais sobre tratamento dos dados, e a forma de construção dos *datasets*.

No capítulo 5 são relatadas informações relacionadas sobre os experimentos executados para este trabalho, e informações sobre configuração dos modelos LSTM.

No capítulo 6 são relatados os resultados dos experimentos, além de informar a interpretação que o autor teve sobre os dados.

No capítulo 7 são sugeridas pesquisas futuras que podem ser realizadas como uma forma de continuidade deste trabalho.

No capítulo 8 são apresentadas as conclusões obtidas neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos teóricos referente aos conteúdos abordados ao longo do trabalho.

2.1 PROCESSAMENTO DE LINGUAGEM NATURAL

Processamento de Linguagem Natural ou mais comumente conhecido como *Natural Language Processing* (NLP) é uma das muitas áreas presentes dentro da Inteligência Artificial (IA). Em (Chopra et al., 2013) os autores definem o processo de NLP como sendo uma área devota a fazer que a linguagem humana escrita e falada, seja compreendida por computadores. De forma similar, a definição de NLP aparece também no livro do Eisenstein (2019a) e é dita como sendo "Um conjunto de métodos para fazer a linguagem humana acessível aos computadores".

O ponto principal em mostrar essas duas definições praticamente iguais é que independente do autor, o objetivo da área é o mesmo, ou seja, ensinar um computador a processar linguagem humana. As técnicas utilizadas para tentativa de ensino são baseadas em modelos matemáticos, estatísticos e linguísticos. A combinação dessas áreas contribui com que sejam criados modelos computacionais que consigam aprender a linguagem humana e, eventualmente utilizá-la para tarefas do dia-a-dia da população. Exemplo de tarefas: filtros de spam, tradução automatizada entre idiomas, dentre outras.

Em (Chopra et al., 2013) os autores mencionam algumas das mais importantes tarefas a se alcançar na área de NLP, sendo elas: sumarização de textos de forma automática, análise de discursos, tradução automática, reconhecimento de entidades nomeadas, dentre outras.

2.1.1 Classificação de textos

No artigo (Minaee et al., 2020) os autores definem classificação de texto como sendo o processo de categorização de textos (*tweets*, artigos de notícia, análise de clientes) em grupos organizados. E dentro dos tipos de classificação de texto, temos: análise de sentimentos, categorização de notícias, análise de tópicos, resposta para perguntas, inferência de palavras, dentre outros.

Para este trabalho, foram utilizadas técnicas de NLP para tentar analisar e classificar um conjunto de textos, de forma que essa classificação reflita informações sobre a autoria dos textos.

2.2 REDES NEURAIS

Redes neurais ou redes neurais artificiais, são um conjunto de algoritmos de aprendizado de máquina que possuem a capacidade de aprender informações a partir de um conjunto de dados fornecido como entrada para a rede. Esses algoritmos foram construídos usando de inspiração a estrutura do cérebro humano (Education, 2020), e por isso também são conhecidos como algoritmos bioinspirados.

A inspiração é refletida na topologia de construção da rede neural, onde estão presentes nodos ou neurônios artificiais (referência aos neurônios humanos) que são conectados uns aos outros (referência a sinapse[1]) para envio de informações (pesos) para os outros nodos presentes no modelo. Organizando essas informações, temos a estrutura da rede neural, de forma que existe

[1]https://pt.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse

uma camada de entrada ligada a uma ou mais camadas escondidas e eventualmente ligada a uma camada de saída. Todas as camadas são compostas de neurônios artificiais. Uma exemplificação pode ser vista na imagem 2.1.

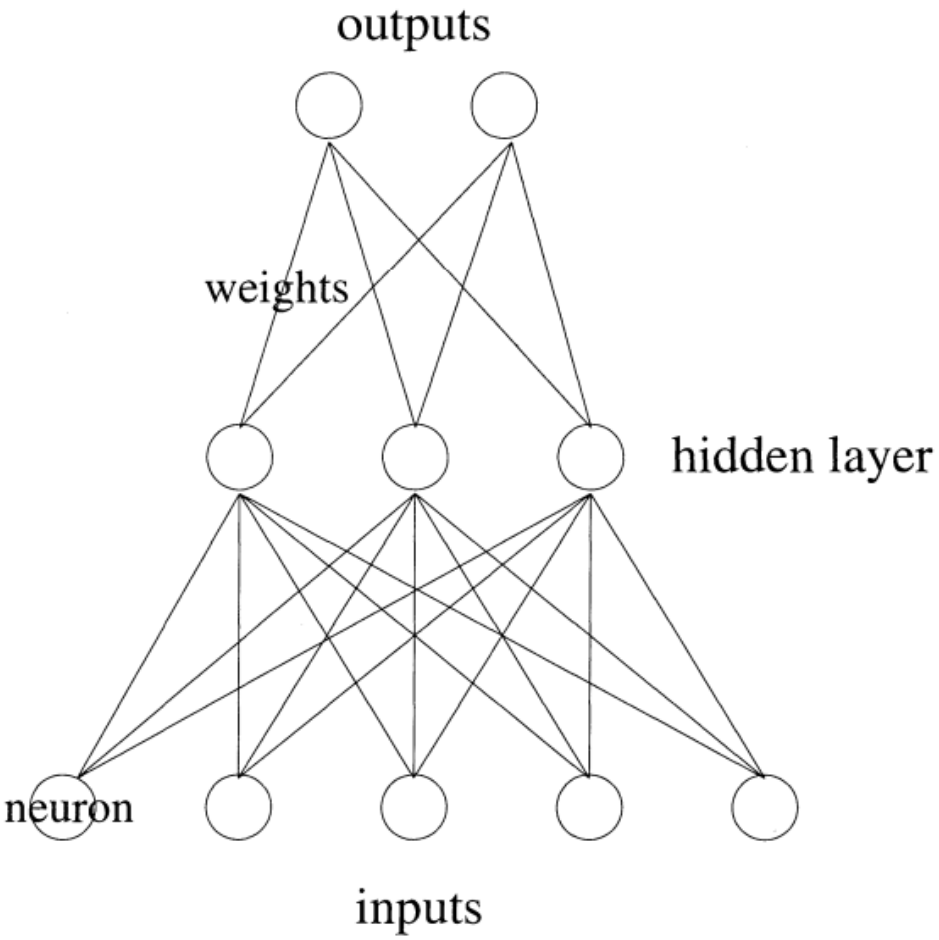


Figura 2.1: Arquitetura de uma Rede Neural. (Wang (2003))

O aprendizado é feito via um longo treinamento que é alimentado via utilização de uma grande quantidade de dados (textos, imagens, vídeos, etc.). Essa grande quantidade de dados vão servir para melhorar a capacidade do modelo de realizar um determinado tipo de tarefa com mais eficiência, por exemplo: classificação de textos.

2.2.1 Redes Neurais Recorrentes

Redes Neurais Recorrentes ou *Recurrent Neural Networks* (RNN) são redes neurais baseadas no modelo tradicional explicado na seção 2.2 mas com um diferencial, esse modelo de rede possui uma estrutura que permite a existência de uma memória.

A RNN não é uma única arquitetura de rede, mas sim uma coleção de arquiteturas que são aplicadas para diferentes problemas, dentre essas variações de arquitetura temos: rede de Hopfield[2], rede de Elman[3], e rede de Jordan.

[2]https://en.wikipedia.org/wiki/Hopfield_network

[3]https://en.wikipedia.org/wiki/Recurrent_neural_network#Elman_networks_and_Jordan_networks

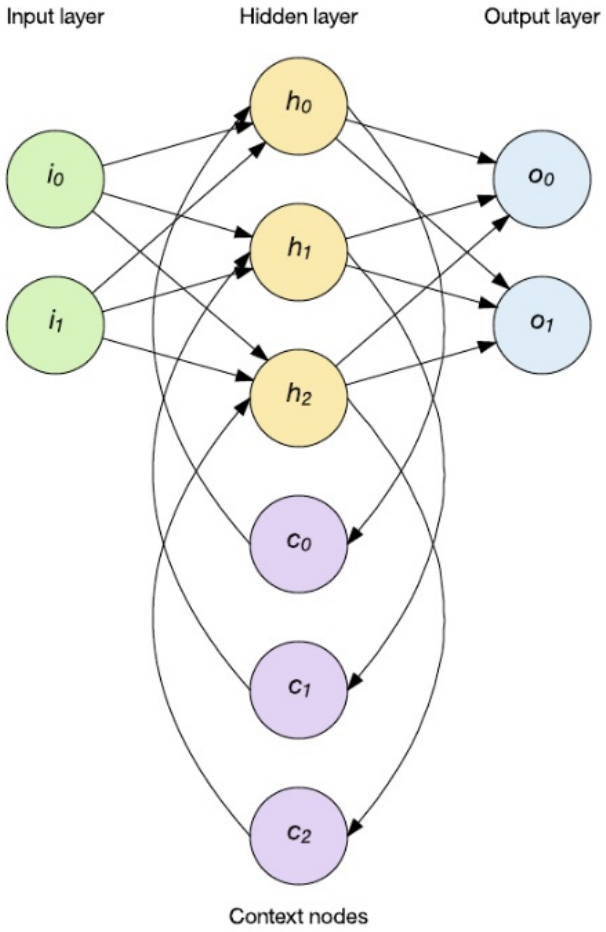


Figura 2.2: Arquitetura de uma rede neural recorrente. (Jones, 2017)

Usando a imagem 2.2 como base da explicação, temos:

1. São fornecidos dados para alimentar a *Input Layer*.

2. Os dados saem da *Input Layer* e em direção a *Hidden Layer*. Cada uma das flechas representadas no desenho saindo de i_0 e i_1 e indo de h_0 até h_2 carregam informações sobre parâmetros do modelo e os valores de entrada para processamento.

3. Dentro da *Hidden Layer* são processados as novas entradas pelos nodos h_0 até h_2. Os dados são processados levando em consideração os dados contidos dentro dos nodos C_0 até C_2, esses dados representam estados anteriores da rede, de forma que isso age como uma memória da rede. Logo, para o estado atual ser atualizado, são levados em consideração informações sobre o estado anterior do aprendizado.

4. Se os estados desejados para valores do modelo forem alcançados, os dados vão para a camada de saída (*Output layer*). Caso contrário, os valores de saída h_0 até h_2 que foram processados levando em consideração o estado anterior de aprendizado, vão substituir os valores presentes em C_0 até C_2, como uma forma de atualização do estado atual.

5. O processo 2 acontece novamente com um novo lote de dados para serem processados.

Apesar da adição de memória fazer com que as redes neurais ficassem mais interessantes, essa adição não veio livre de problema, e o problema é referente ao problema que leva o nome de

Vanishing gradient problem(Arbel, 2018). Esse é um problema que dificulta o aprendizado de dados com uma sequência consideravelmente longa. O método de propagação dos dados leva em consideração informações de pesos utilizados em estados anteriores da RNN, e devido a este fato, cada vez que os pesos são atualizados, o valor pode ir ficando cada vez menor, e portanto as atualizações dos parâmetros começam a ficar insignificantes e fazem com que o modelo não aprenda realmente algo novo. Para contornar esse problema, foi proposto o que veio a se chamar de *Long Short-Term Memory* ou simplesmente LSTM.

2.2.2 Modelo LSTM

O modelo *Long Short-Term Memory* (LSTM) aparece como uma melhoria da rede RNN padrão, de forma que, com as modificações propostas pela LSTM resolvem o problema relacionado a processamento de sequências longas de dados. Essas modificações envolvem ações para melhoria do sistema de memória da RNN, de forma que o modelo passe a ter uma memória de longo prazo usando células de memória que lembram de momentos de tempo arbitrários do modelo (Minaee et al., 2020) e utiliza de gates para ajudar a regular o fluxo de informação que entra e sai da célula.

Os *gates* são estruturas que agem como tipos de memória seletoras, essas estruturas levam o nome de *Input gate, Output gate* e *forget gate*. A presença desses *gates* permitem que o modelo "treine"neurônios de forma individual, isto é, em uma nova rodada de treinamento do modelo, pode ser que um determinado neurônio receba um sinal que os dados que já conhecidos não foram modificados, permitindo então que um determinado grupo de informações permaneça por quanto tempo o modelo as considere importantes.

O *input gate* age como a estrutura que vai ditar o que precisa ser lembrado, por exemplo, lembrar que “roupa” e “roeu” possuem uma ligação importante que precisa ser mantida dentro da frase “O rato roeu a roupa do rei de Roma”.

O *Output gate* age como a estrutura que vai ditar o que deve sair do neurônio para a próxima etapa, já que nem tudo que está salvo na memória precisa ser levado para o próximo neurônio.

O *forget gate* age como a estrutura que dita o que deve ser esquecido pelo neurônio, por exemplo o contexto do treinamento fazem com que “roupa” e “roeu” não tenham mais uma ligação importante, então essas informações podem ser esquecidas pelo neurônio.

Na imagem 2.3 é possível ver a *sigmoid layer*, essa camada vai ter como importância o valor de uma *sigmoid* (entre 0 e 1), que vai agir como representação da porcentagem de dados que devem ocorrer pro input, output e forget. Na imagem 2.3 é possível ver como uma unidade de LSTM funciona.

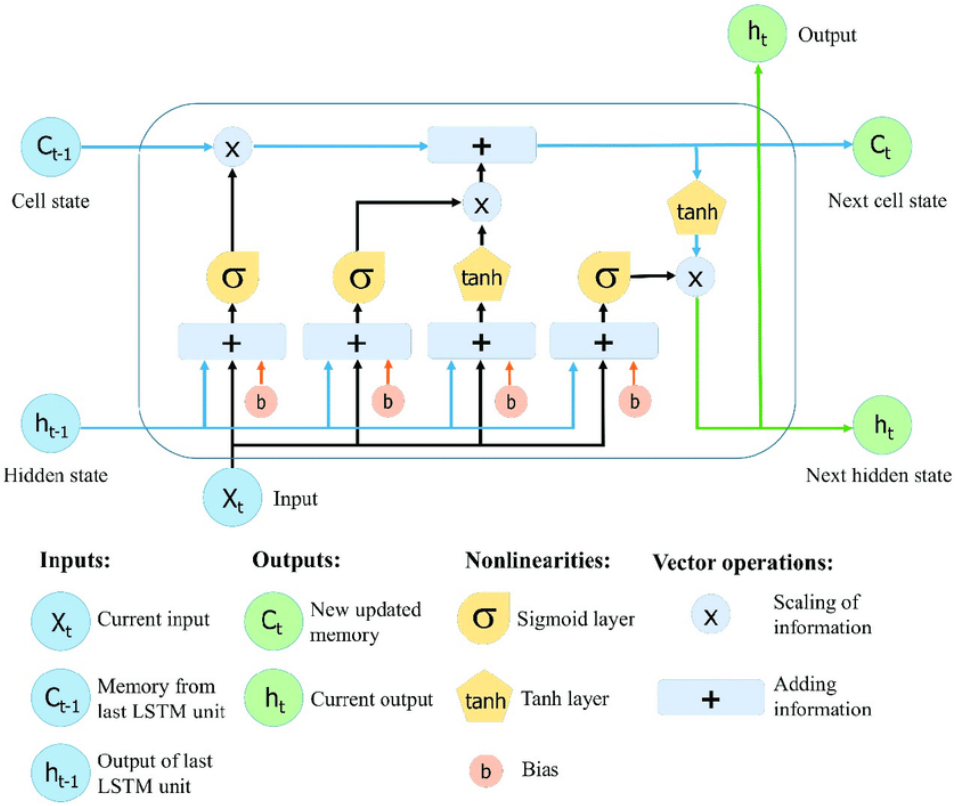


Figura 2.3: Arquitetura de uma unidade LSTM. (Le et al., 2019)

2.3 REPRESENTAÇÃO DE PALAVRAS

Essa parte do texto visa tentar explicar conceitos para compreender a forma que palavras são representadas para serem utilizadas por máquinas.

2.3.1 One-hot-encodding

Devido ao fato que as máquinas não conseguem compreender palavras, então, para realizar processamento de textos, é necessário realizar uma representação que máquinas consigam compreender, e essa representação é feita através de atribuição de um código numérico para cada palavra presente no texto. Essa codificação é feita utilizando base binária (DEY, 2021).

Um exemplo seriam as palavras tomate e salada. Realizando a codificação de tomate teríamos [1 0] e a codificação para salada [0 1].

Uma vez que tenham sido codificadas as palavras em formato de vetor numérico, é possível fornecer como entrada esses vetores para um modelo computacional processar.

2.3.2 Continuous bag of words

O *Continuous bag of words* (CBOW) é um modelo focado em tentar compreender o contexto das palavras fornecidas como entrada, e usando desse contexto, tentar predizer qual seria a palavra que de interesse que complementa a entrada.

Esse modelo funciona seguindo os seguintes passos: tokenização da sentença, transformação dos *tokens* em representação de *One-hot-encoding*, projeção da entrada, e por fim, predição da palavra que deveria estar ali presente.

Usando como exemplo a frase: “Na salada tem tomate vermelho”.

1. Aplicando o processo de tokenização da sentença, teríamos então **["Na", "salada", "tem", "tomate", "vermelho"]**

2. Aplicando o processo de *One-hot-encoding* 2.3.1 , teríamos então as seguintes representações:

   - Na = [1 0 0 0 0]
   - salada = [0 1 0 0 0]
   - tem = [0 0 1 0 0]
   - tomate = [0 0 0 1 0]
   - vermelho = [0 0 0 0 1]

3. Dada a frase, é passado uma janela de tamanho N (no caso desse exemplo, N = 3), para selecionar os pares de palavras e alvo a ser predito como entrada pro CBOW.

4. A janela seleciona os elementos e o formato fica: ([par de palavras do contexto], palavra que se deseja predizer). Exemplo: ([na, tem], salada).

5. Esses dados de entrada codificados são enviados à *hidden layer* junto com suas respectivas matrizes de contexto. Essas matrizes possuem tamanho **MxN**, onde M é o tamanho da janela de palavras, no caso de exemplo 3, e N é a quantidade de palavras do dicionário gerado durante a tokenização. No desenho essas matrizes estão representadas como W.

6. Usando os dados fornecidos, é aplicada uma função *softmax*[4] para pegar as probabilidades das palavras e determinar qual delas acredita ser a que falta.

7. O resultado da predição é comparado com a palavra alvo verdadeira, e os erros são utilizados para atualizar a matriz de contexto.

8. A janela de seleção desliza pela frase e o processo de 3 até 7 são repetidos até que todas as palavras tenham sido atualizadas.

   Na imagem 2.4 é possível ver um exemplo da arquitetura do CBOW.

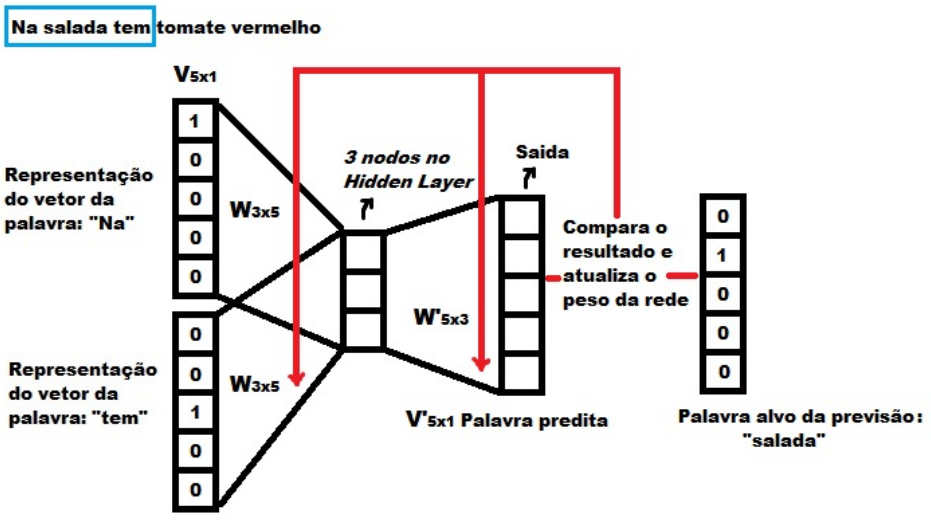


Figura 2.4: CBOW funcionando. Imagem adaptada de (Semicolon, 2018)

[4]https://en.wikipedia.org/wiki/Softmax_function

2.3.3 Word2Vec

O modelo *Word2Vec* utilizado neste trabalho utiliza a topologia do CBOW.

O *Word2Vec* recebe um corpo de dados como entrada e produz vetores de palavras como saída. Inicialmente é construído um vocabulário (ou dicionário) a partir dos dados obtidos do texto de treinamento, e ao utilizar o CBOW, são aprendidas informações sobre as palavras, como contexto, distância entre termos, dentre outras coisas (Google, 2013).

No modelo *Word2Vec*, os contextos das palavras são catalogados de acordo com as suas posições em relação a uma palavra alvo (Eisenstein, 2019b), esse processo é descrito na seção do CBOW 2.3.2. Como consequência direta do formato de extração da técnica CBOW, temos uma representação sensível à sintaxe. O que torna a diferenciação de palavras, como por exemplo, réis e reis seja possível mesmo que réis esteja sem acentuação. Considere a frase "A moeda réis estava em vigência no Brasil durante o século XIX", normalmente o treinamento do *Word2Vec* leva palavras sem sua forma tradicional, ou seja, sem acentuação alguma. Então nessa situação, réis estaria escrito como rei, e baseado na sua vizinhança, o contexto de representação de réis seriam relacionados a moeda. Mas agora pegamos a frase "Hoje vai acontecer uma conferência entre reis de reinos aliados". Nessa situação, o contexto da palavra reis estaria relacionado a reinos, devido a informações que são fornecidas pelas palavras próximas da frase.

2.3.4 Word Embedding

O termo *Word Embedding* é dado para a técnica de representação de palavras em formato de vetores de números reais de dimensão inteira k. (Eisenstein, 2019c) Essas representações são formas de se passar informações para redes neurais, de forma que sirvam como parâmetros de entrada para essas redes.

Os valores fornecidos pela técnica de *Word Embedding* contém informações para serem utilizadas pela máquina sobre o contexto individual da palavra, e caso a palavra tenha diversos significados que variam de acordo com o contexto, então o modelo deve conter vários *embeddings* para representar cada uma dessas situações. (Eisenstein, 2019c). Logo, o valor da técnica de *Word Embedding* se encontra nessa capacidade de fornecer informações que são relacionadas aos contextos sintáticos e semânticos das palavras.

O contexto de uma palavra pode ser definido de várias formas, mas para a topologia utilizada neste trabalho, o contexto é definido pela estrutura do *Word2Vec*.

Podemos dar como exemplo uma operação utilizando as representações das palavras: "homem", "rei", "mulher". Se a operação for realizada de forma que: rei - homem + mulher, provavelmente teremos como resultado o vetor de representação da palavra rainha. Isso acontece devido à proximidade da palavra mulher com homem e rei com rainha.

2.3.5 Tokenização

A tokenização foi realizada utilizando a função Tokenizer[5] da biblioteca Keras. Essa função tem como *default* um argumento de filtro que realiza a remoção de pontuação, mas por decisão de projeto, esse argumento foi passado como vazio.

O processo de tokenização pela função segue o seguinte fluxo: (i) Inicialização do tokenizador, por exemplo: passagem de argumento como quantidade máxima de palavras e filtros; (ii) construção do dicionário de palavras, com o índice sendo um valor numérico inteiro e o elemento uma palavra; (iii) gera um vetor com elementos de tamanho 100, em que cada elemento é uma versão numérica de uma sentença do *dataset*. Caso o vetor fique com tamanho maior que

[5]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

100, os elementos do final são removidos até ficar ajustado a 100; (iv) gera a codificação para as classes presentes na característica autores. Um exemplo do processo pode ser visto a seguir.

Damos como exemplo essas duas frases:

1. "O dia amanheceu ensolarado."
2. "O dia foi cansativo."

Após o passo (ii) é formato o dicionario da tabela 2.1.

1	2	3	4	5	6	7
"O"	"dia"	"amanheceu"	"ensolarado"	"."	"foi"	"cansativo"

Tabela 2.1: Dicionário de palavras únicas

Durante o passo (iii) a representação das frases ficaria da forma em que aparecem na tabela 2.2:

Frase	Representação numérica
"O dia amanheceu ensolarado."	[1 2 3 4 5]
"O dia foi cansativo."	[1 2 6 7 5]

Tabela 2.2: Representação numérica das frases

Durante o passo (iv) os *labels* dos autores são codificados da seguinte forma.

- Para a situação com 2 classes:

  jose = [1 0] que é representado no modelo como classe 0.

  machado = [0 1] que é representado no modelo como classe 1.

- Para a situação com 3 classes:

  jose = [1 0 0] que é representado no modelo como classe 0.

  machado = [0 1 0] que é representado no modelo como classe 1.

  outro = [0 0 1] que é representado no modelo como classe 2.

3 METODOLOGIA

Neste capítulo serão apresentados informações sobre as ferramentas utilizadas no trabalho, e sobre os passos tomados ao decorrer do desenvolvimento.

3.1 ETAPAS

Para a realização desse trabalho, foi optado pela utilização da linguagem de programação Python, pelos motivos de: (i) facilidade de utilização; (ii) possibilidade de utilização de um ambiente de execução em nuvem; (iii) existência de uma grande variedade de bibliotecas gratuitas com métodos necessários para o funcionamento da rede neural LSTM. Este trabalho começou com o intuito de verificar se era possível uma rede neural LSTM identificar características de escrita de obras literárias brasileiras. Com essa ideia em mente, foi então feita a escolha de utilizar autores com características similares de tipo de texto e com obras disponíveis no domínio público brasileiro, os escolhidos foram José de Alencar e Machado de Assis.

Após escolha dos autores e das tecnologias que seriam utilizadas, se deu início ao processo de experimentação para criação de um *dataset* base sem ruído.

Como mencionado anteriormente, a segmentação das sentenças tinha como ideia inicial tentar seguir os moldes de *datasets* construídos a partir de textos curtos (Twitter), e portanto a segmentação das sentenças inicialmente foi feita por contagem de caracteres. Foi decidido pelo valor arbitrário de 500 caracteres por sentença, incluindo a contagem do caractere de representação de espaço em branco. Essa versão foi descartada pois, após o processo de tokenização das sentenças, o tamanho dos vetores ficavam muito discrepantes, tendo em média o tamanho de 81 *tokens*, obrigando então a necessidade de utilização de preenchimento para que todos os vetores ficassem com o mesmo tamanho.

A segunda versão, que é a atual, manteve o princípio de molde de textos curtos, mas dessa vez a segmentação se deve por palavras, ignorando então os espaços em branco e facilitando na obtenção de sentenças com exatamente 100 palavras. Os detalhes sobre o processo de construção serão mencionados mais à frente no texto.

Com o *dataset* base construído, se deu então a etapa de experimentação verificar se era possível o utilizar para o propósito de atribuição de autoria. Verificado que existia possibilidade de levar o projeto adiante, foi então tomada a decisão da adição de um terceiro autor ao *dataset*, de forma com que fosse possível verificar se os valores obtidos durante os experimentos com dois autores eram reais, ou apenas valores inflados. O terceiro autor foi inserido como uma forma de ruído, e portanto, serviu como uma forma de verificar se o modelo estava realmente classificando algo e não apenas classificando aleatoriamente, dessa forma, seria possível tirar algumas conclusões sobre os valores obtidos nos experimentos com duas classes de autor.

Após os modelos serem treinados, foi decidido pela criação de dois novos *datasets* para realização de alguns testes de generalização dos modelos.

O *dataset* de teste simples possui um conjunto de 10 elementos de cada classe. As sentenças providas pelos livros possuem ligação com o *dataset* de treinamento, pois são do mesmo conjunto de livros que foram utilizados para o *dataset* de treinamento. No caso da versão com três classes, no caso dos dados fornecidos pelo Wikipédia, não se é possível garantir a existência de uma ligação entre os dados de treinamento e de teste, pois cada texto utilizado foi pego de páginas escolhidas aleatoriamente no site do Wikipédia.

O segundo *dataset* de teste possui um subconjunto de 37 elementos de cada classe. Os dados de construção deste *dataset* foram pegos de obras não utilizadas no *dataset* de treino, e portanto, esse *dataset* tem como intuito verificar se os modelos conseguem atribuir corretamente a autoria dessas sentenças que não possuem uma ligação direta às que foram utilizadas no treinamento.

Após criados os *datasets* de teste, se deu início a fase de experimentos voltados à verificação da capacidade de utilização do modelo para dados familiares (primeiro *dataset*, teste simples) e não familiares (segundo *dataset*, teste generalizado).

3.2 FERRAMENTAS

Nessa seção serão mencionados de forma simplificada a funcionalidade da biblioteca e o porque foi utilizado no trabalho.

3.2.1 Pandas

Pandas é uma biblioteca de código aberto voltada a fornecer ferramentas de suporte para análise e manipulação de dados. Essa biblioteca é uma ferramenta poderosa principalmente pelo tipo de estrutura de dados que ela fornece, o formato *Dataframe*[1].

Para este trabalho, o formato *Dataframe* possibilitou que fossem realizadas todas as operações relacionadas a manipulação e preparação dos dados para a criação do modelo. E devido a este fator, essa biblioteca foi escolhida para ser utilizada.

3.2.2 Pdfplumber

Essa biblioteca foi utilizada para extrair o conteúdo textual das páginas em PDF dos livros utilizados. Dentre suas utilidades, consta a capacidade de delimitar uma região para extração de informações das páginas, extrair informações sobre dimensões das páginas, dentre outras funcionalidades[2].

Para este trabalho foram utilizadas as funcionalidades de delimitação de região para extração de texto e extração de informações sobre dimensões das páginas. Essas funcionalidades foram utilizadas com o intuito de conseguir obter apenas o texto das páginas dos livros e ignorar qualquer tipo de informação que tivesse em rodapé ou cabeçalho. Foi necessário realizar essa ação, pois, foram encontrados diversos modelos de PDF para as obras, onde por exemplo, alguns possuíam paginação da obra no canto superior direito e outras possuíam paginação da obra no canto inferior esquerdo.

3.2.3 Keras

Keras é uma biblioteca de aprendizado profundo escrita em Python que utiliza de métodos de aprendizado de máquina fornecidos pela biblioteca TensorFlow. A própria página do Keras se refere a biblioteca como sendo simples, flexível e poderosa (Keras, 2022).

Essas características foram levadas em consideração para a escolha de utilização do Keras. Dessa biblioteca foram utilizados os métodos que implementam os elementos presentes nas camadas da rede neural LSTM.

[1] https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframel

[2] https://github.com/jsvine/pdfplumber#python-library

3.2.4 Google Colab

Google Colab[3] é uma plataforma online para execução de códigos computacionais. Essa plataforma fornece um ambiente computacional pré-configurado, de fácil acesso (utilização via navegador) e com algumas bibliotecas populares pré-instaladas, por exemplo: Keras, Pandas, etc. O Colab fornece planos gratuitos para execução de código e possibilita acesso a GPU. Existem planos pagos para utilização, mas para este trabalho a versão gratuita foi o suficiente.

O ambiente funciona fornecendo um ambiente interativo em nuvem chamado notebook Colab. Neste ambiente é possível escrever códigos segmentados por células de código, ou textos em células de texto. Cada célula de código pode ser executada de forma independente dentro do *notebook* Colab, isto faz com que seja possível executar segmentos de código de forma isolada, sem a necessidade de execução de todo o programa novamente.

A utilização do Colab neste trabalho se deve a necessidade de um ambiente computacional poderoso para treinamento dos modelos, e como mencionado anteriormente, a possibilidade de utilização de um ambiente com recursos computacionais limitados, mas poderosos de forma gratuita, foi essencial para poder realizar os experimentos do projeto.

3.2.5 Biblioteca Wikipedia para Python

Foi a biblioteca utilizada para obter os dados provenientes do Wikipédia. Essa biblioteca teve como finalidade agir como uma ponte de fácil acesso para utilização da API do site Wikipédia.

Os detalhes sobre a utilização são tratados com mais detalhes na seção de construção do *Dataset*.

3.2.6 Biblioteca RE

A biblioteca de expressões regulares[4] (re) nativa da linguagem Python foi a base para construção dos algoritmos de limpeza e padronização automática deste trabalho. Foi principalmente utilizada para separação de pontuação de palavras do texto, substituição de quebras de linha por espaços em branco, remoção de espaçamento excessivo, padronização de pontuação, dentre outras coisas que serão melhor descritas mais à frente no texto.

Essa biblioteca usa de expressões regulares, para encontrar e substituir em um dado texto, informações específicas que são descritas na forma de expressão regular. Por exemplo a frase: “Brasília é a capital do Brasil”, se for passado na expressão regular que desejamos remover todas as ocorrências da letra B dentro dessa frase, o resultado final seria: “rasília é a capital do rasil”.

3.2.7 Numpy

A biblioteca Numpy[5] é uma biblioteca *Open Source* voltada para o ambiente de computação científica. Numpy é conhecida por fornecer implementações otimizadas para o ambiente Python de operações matemáticas, manipulação de vetores, dentre outras coisas. Devido a estas características essa biblioteca é utilizada como suporte para trabalhos envolvendo aprendizado de máquina e redes neurais.

Durante este trabalho, os métodos do Numpy foram utilizados principalmente para manipulação da matriz de *embeddings* pré-treinados.

[3] https://colab.research.google.com/

[4] https://docs.python.org/3/library/re.html

[5] https://numpy.org/

3.2.8 CuDNN

CUDA Deep Neural Network ou cuDNN é uma biblioteca da NVidia com foco em acelerar o processamento de redes neurais de aprendizado profundo. Tem como compromisso fornecer uma melhor utilização de GPU de alto desempenho. Esse compromisso é realizado ao fornecer implementações para rotinas de execução presentes em alguns modelos de redes neurais, como por exemplo: rotina de normalização, ativação de camadas, dentre outros.

Essa biblioteca está presente dentro das implementações de diferentes tipos de bibliotecas de aprendizado de redes profundas, como por exemplo a Keras. Devido a pouca necessidade de modificação do modelo para a utilização dessa otimização, foram realizados experimentos levando em consideração um modelo LSTM com essa funcionalidade ativada, a fim de verificar se ocorrem diferenças de resultados em relação a um modelo LSTM “padrão”.

4 DATASET

Foram criadas duas variações de *dataset* para esse trabalho: uma versão com duas classes de autor, “Jose” e “Machado”; e outra versão com a adição de uma terceira classe de autor “Outro”, que são dados providos pelo Wikipédia. Para ambas versões, foram criados 3 *datasets* cada: *dataset* para treinamento, *dataset* para validação e testes “simples”, *dataset* para validação e testes “generalizada”.

Os *dataset* foram criados utilizando dados disponíveis de forma pública na Wikipédia e livros disponíveis no acesso público de obras brasileiras. As obras escolhidas são romances de José de Alencar e Machado de Assis. Para os *dataset* de treinamento e validação simples, os livros utilizados para extração de dados foram:

- Machado de Assis: "Casa Velha", "Dom Casmurro"e "Helena".
- José de Alencar : "A pata da gazela", "Lucíola"e "A viuvinha".

Para os *dataset* de validação generalizada, os livros utilizados para extração de dados foram:

- Machado de Assis: "Quincas Borba"e "Esaú e Jacó".
- José de Alencar: "Iracema"e "O gaúcho".

Como mencionado anteriormente no texto, os dados extraídos do Wikipédia possuem em sua maioria pouca relação entre si, então não foi necessário fazer uma distinção rigorosa da mesma forma com que foi feita para os livros. O cuidado tomado foi direcionado em não utilizar os mesmos dados presentes no *dataset* de treino para os de validação. Para a classe de autoria referente ao Wikipédia, foram usados aproximadamente 450 páginas em português, que foram selecionadas após um processo de filtragem manual que será descrito com mais detalhes mais à frente no texto.

O *dataset* de treinamento possui 1132 elementos de cada classe, totalizando 2264 entradas para o *dataset* em sua versão binária e 3369 elementos em sua versão multi-classes.

O *dataset* de teste e validação simples possui 10 elementos de cada classe, totalizando 20 elementos em sua versão binária e 30 elementos em sua versão multi-classes. As sentenças utilizadas neste *dataset* foram retiradas das mesmas obras utilizadas para o *dataset* de treino, de tal modo que foram tomadas medidas para as entradas presentes em um não estivessem presentes no outro.

O *dataset* de teste e validação generalizada possui 37 elementos de cada classe, totalizando 74 elementos em sua versão binária e 111 elementos em sua versão multi-classes. As sentenças utilizadas neste *dataset* foram retiradas de obras diferentes das que foram utilizadas durante o treinamento do modelo. Dessa forma, é possível verificar se a proposta do modelo de reconhecer padrões de escrita para determinado grupo de autores, foi atingida ou não.

O processo de criação de todos os *dataset* seguiram os mesmos passos que serão descritos futuramente no texto, e portanto, não será tratado de forma individual o processo de criação.

4.1 EXTRAÇÃO DE DADOS

Para conseguir realizar a obtenção e extração dos dados, foi utilizado duas técnicas diferentes devido a diferença do formato dos dados base. Para livros uma biblioteca para extrair texto de arquivos PDF, e para o Wikipédia uma biblioteca para escolha e extração de dados das páginas.

4.1.1 Livros

Os livros foram obtidos no formato PDF pelo site do Domínio Público[1]. Cada livro possui algum tipo de peculiaridade que foi necessário tratar de uma forma individual para cada obra. A ferramenta utilizada para extração dos textos foi a biblioteca *pdfplumber*.

O ajuste manual se deve principalmente à criação de um retângulo dentro de cada página dos livros para delimitar o que seria retirado de informação do texto. Essa delimitação foi necessária para evitar que as paginações fizessem parte do texto extraído, logo, foi necessário diminuir o ruído proveniente da extração. Foi observada duas versões de localização da paginação dentre as obras, um modelo tinha a numeração referente a página no canto superior direito de suas páginas, e o outro modelo tinha sua numeração no canto inferior direito da página.

Após a extração das informações das páginas em PDF, os livros foram salvos sem nenhum tipo de modificação em formato TXT e codificação “UTF-8”, que eventualmente seriam utilizado como base para os eventuais tratamentos textuais utilizados para os experimentos.

4.1.2 Wikipédia

As páginas do Wikipédia foram obtidas utilizando a biblioteca wikipedia[2] para linguagem Python. Para esse trabalho, foi configurado que as páginas que seriam selecionadas teriam que obrigatoriamente estarem em português brasileiro e teriam que ter no mínimo 600 caracteres. Junto a essas regras, foi escolhido um valor arbitrário de 2500 páginas para parâmetro de quantas páginas seriam buscadas de forma aleatória na função de busca da biblioteca, como resultado final apenas 750 páginas viraram candidatas a serem utilizadas no trabalho. Os textos foram extraídos da seção "conteúdos"de cada página e salvos em arquivos de texto com codificação "UTF-8"e formato TXT.

4.2 TRATAMENTO DOS DADOS

O tratamento de dados foi feito de forma manual e individual para cada conjunto de dados utilizado. Esse trecho do trabalho tem como intenção informar o processo utilizado na criação do *dataset* utilizado.

4.2.1 Pontuação

Como uma regra global no tratamento dos dados deste trabalho, foi tomada a decisão de considerar pontuação como sendo uma palavra, e portanto, contaria como sendo um *token* dentro da representação dos dados, a única exceção de pontuação que não foi contada como sendo uma palavra única, foi os hífens, isto é, palavras que estavam representadas como utilização de hífen se mantiveram completas e tratadas como um *token* próprio.

[1] http://www.dominiopublico.gov.br/pesquisa/PesquisaObraForm.jsp

[2] https://pypi.org/project/wikipedia/

Outra decisão tomada a respeito da pontuação foi referente a representação de reticências. Devido a erros de digitalização, foi considerado que qualquer trecho que possuíssem dois pontos finais ou mais seguidos, seriam considerados como representações de reticências. A exceção a essa regra foi aplicada a trechos onde tinham uma palavra abreviada com ponto e também seguida de mais um ponto final, nessa situação é possível se compreender que não foi um erro de digitalização. Exemplos de aplicação da regra podem nos casos a seguir:

1. Situação em que tem apenas dois pontos:
    - Frase Original: "Para D.. Foram feitas algumas solicitações"
    - Frase com a regra aplicada: "Para D. . Foram feitas algumas solicitações"”
2. Situação em que tem dois pontos adicionais ao de abreviação:
    - Frase Original: "Para D.. . Foram feitas algumas solicitações"
    - Frase com a regra aplicada: "Para D. ... Foram feitas algumas solicitações"

4.2.2 Padronização

Um dos processos utilizados durante a criação do *dataset* foi o de padronização. Esse processo foi aplicado tanto para os dados extraídos dos livros quanto para os dados extraídos do Wikipédia.

O processo de padronização do texto consiste em aplicar uma série de filtros para aplicar um espaço em branco entre as palavras e as pontuações existentes nas sentenças (?, !, $, ..., etc.) e, simultaneamente ajudar a padronizar acentuações que tinham diferentes representações ao longo dos textos, como exemplo a ser mencionado é a representação do caractere de aspas duplas, que em alguns trechos aparecia com o símbolo de “menor que” (>). Nesse momento também era levado em consideração os casos especiais que foram construídos a partir de uma leitura das obras. Estes casos especiais possuem informações sobre quais palavras não devem ter sua pontuação separada, como no caso de abreviações. Palavras que podem ser dadas como exemplo desses casos especiais seriam: d’água e Sr. .

4.2.3 Tratamentos para os Livros

Para cada dado tratado proveniente de livros, foi utilizado uma sequência de passos que consistiram em: leitura superficial do livro, registro de casos “especiais” presentes na obra, limpeza e padronização de escrita, e por último a segmentação do livro em trechos com tamanho de 100 palavras.

Cada livro possui um conjunto de características que foram divididas em dois.

A primeira característica foi decidido que seria referente aos aspectos peculiares de escrita de cada autor, esses foram considerados casos especiais. As características que compõem os casos especiais são:

- Abreviação de palavras: "Sr.", "Sr.ª", "etc.", "D.", "Dr.", "Ex.ª", "cap.", "vers.", "Rev."
- Representação monetária: 1:070$000 e 80rs. para Réis.[3]
- Utilização de travessão.

[3]https://www.bcb.gov.br/content/acessoinformacao/museudocs/pub/SintesePadroesMonetariosBrasileiros.pdf

A outra característica foi definida como sendo erro proveniente da ferramenta utilizada para criação da versão digital das obras. Alguns dos erros observados foram tratados de forma manual, enquanto outros foram possíveis extrair um padrão e arrumar de forma automatizada com uma função computacional. Um exemplo desse problema poderia ser dado pela falha na captação de reticências. Foi comum encontrar trechos onde era possível visualizar a representação da reticência de forma quebrada, com dois pontos finais seguidos de um espaço em branco e mais um ponto final (como demonstrado em um exemplo na seção anterior). Outros casos que foram tratados como erro de digitalização:

- Troca de letras por números: A palavra enviá-los teve o carácter "L"representado com o numeral 1. Exemplo: "envia-1os".
- Padronização na escrita da mesma palavra. Um exemplo foi a palavra d'água que podia ser encontrada no mesmo livro sendo representada como: d' água e d'água.
- Padronização na codificação do caractere: aspas simples, aspas duplas.
- Correção na codificação usada para representação de caracteres. Exemplo: em alguns trechos aspas duplas estavam representadas pelo símbolo » (maior, maior) ou « (menor, menor).

A obtenção das características de cada livro foi realizada de forma manual, através de leitura superficial das obras. Durante a leitura era observado quais elementos necessitavam de ajustes e anotados para serem trabalhos individualmente. Uma parte dos elementos foram citados anteriormente nos casos especiais e características de erros de digitalização, a outra parte foi referente a verificação do padrão de título dado para cada novo capítulo de cada um dos livros utilizados.

Após registrado as características da obra a ser processada, o passo de limpeza era aplicado.

O processo de limpeza automatizada consiste em transformar quebras de linha, indentação, e espaçamentos excessivos em um único espaço em branco. Dessa forma é possível trabalhar o texto com o padrão de separação sendo espaços em branco.

Como passo final do processo de limpeza, acontece a remoção dos títulos dos capítulos do livro. Esse processo foi realizado levando em consideração o formato do título do capítulo, que poderia ser em formato de apenas numeral romano ou no formato **"CAPÍTULO + numeral romano + NOME DO CAPÍTULO (Opcional)"**, por exemplo: "CAPÍTULO XIII CAPITU". Caso o capítulo tivesse o formato de apenas numeral romano, era verificado se no corpo do texto existem casos que poderiam causar problemas em uma remoção automatizada, caso existissem, o capítulo era removido de forma manual do texto, caso contrário, era removido de forma automatizada usando critério de substituição do numeral romano por uma *string* vazia.

No caso em que se tinha o formato **"capítulo + numeral romano + nome do capítulo"**, foi passado um arquivo texto com o nome e formato de todos os capítulos do livro em questão, de forma que fosse possível buscar pelo termo exato no texto e o substituir por uma *string* vazia. A maioria dos títulos foram removidos com sucesso usando essa tática, raras exceções que não funcionaram, então esses foram removidos manualmente do texto.

Após aplicação do processo de padronização do texto mencionado na seção 4.2.2, os dados foram salvos em um arquivo TXT para que ocorresse validação humana e correções manuais para eventuais problemas encontrados.

Com o texto limpo, padronizado e validado, é aplicado um pré passo final de quebra do texto para poder realizar a criação das sentenças que vão compor o *dataset*. A quebra é realizada

usando como critério de separação o caractere de espaço em branco. O texto completamente quebrado é armazenado em um vetor onde cada elemento do vetor é uma “palavra”. Um exemplo pode ser visto a seguir:

1. Frase Original: "O gato subiu em cima do telhado, dormiu e, após algumas horas, desapareceu."

2. Aplicado o processo de separação: [**"O"**, **"gato"**, **"subiu"**, **"em"**, **"cima"**, **"do"**, **"telhado"**, **",** **"**, **"dormiu"**, **"e"**, **",** **"** , "**após"**, **"algumas"**, **"horas"** , **",** **"** , **"desapareceu"**, **".** **"**].

Importante ressaltar que mesmo que tenha ocorrido a quebra do texto, os elementos do vetor de palavras estão exatamente na ordem em que se encontravam dentro do texto, dessa forma é possível reconstruir o texto juntando todas as palavras e as separadas pelo caractere de espaço em branco.

E por fim, utilizando o vetor de palavras que foi gerado no passo anterior, são gerados os elementos que vão compor o *dataset*. Cada sentença é gerada juntando um conjunto de 100 palavras seguidas do vetor de palavras. Esse conjunto é ligado usando como critério o caractere espaço em branco, e vai ser salvo em um novo vetor onde cada elemento vai ser uma sentença com exatamente 100 palavras.

4.2.4 Tratamentos para os dados do Wikipédia

Para os dados provenientes da Wikipédia, foram realizados os seguintes passos: verificação manual de tamanho e conteúdo, limpeza manual, filtro automatizado, e por último, segmentação do conteúdo em blocos de 100 palavras.

Por decisão de padronização, foi optado por utilizar os mesmos casos especiais encontrados nos livros, de forma que, não tivessem versões duplicadas dos casos especiais providos pelos livros durante a criação do dicionário de *tokens* únicos.

O processo de verificação consistiu em análise manual e na limpeza inicial de cada uma das páginas que passaram no processo de extração e obtenção de dados. A análise consistiu em: leitura superficial do texto, remoção de quebras de linhas exageradas (exemplo: 10 quebras de linhas seguidas sem nenhum texto no meio), remoção de caracteres não presentes no alfabeto do português brasileiro (exemplo: *katakanas, hiragana, cirilico*, etc.), remoção de links e exclusão de conteúdo considerado inapropriado.

Todas as modificações realizadas durante o processo de seleção foram salvas nos próprios arquivos originais.

Foram desconsideradas páginas que o conteúdo fosse referente a temas muito polêmicos (cenário político atual), páginas com fórmulas matemáticas que após remoção das fórmulas não tivessem o suficiente para 100 palavras, páginas com apenas registros de datas ou referências (exemplo: lista de títulos futebolísticos), páginas que após remoção de quebras de linha ficassem com menos que 650 caracteres.

Após a validação do processo manual, são aplicados filtros de substituição de quebras de linha por espaços em branco, remoção de espaços em branco que aparecem de forma exagerada e remoção de indentação do texto. Dessa forma é possível trabalhar o texto com o padrão de separação sendo espaços em branco.

Após a aplicação dos filtros, os dados passaram pelo processo de padronização descrito em 4.2.2. E com os textos limpos, é dado início ao processo de segmentação das sentenças.

O processo de segmentação das sentenças é feito de forma similar ao relatado na seção dos livros, com a diferença que pelos dados do Wikipédia serem independentes entre si (estarem em arquivos TXT diferentes), a segmentação é feita em cada arquivo de forma individual.

Por fim, da mesma forma que foi aplicado no processo dos livros, é lembrado ao leitor que mesmo que tenha ocorrido a quebra do texto, os elementos do vetor de palavras estão exatamente na ordem em que se encontravam dentro do texto, dessa forma é possível reconstruir o texto juntando todas as palavras e as separadas pelo caractere de espaço em branco.

5 EXPERIMENTOS

Os experimentos foram realizados na plataforma online Google Colab com a opção de GPU ativada para ajudar na aceleração da execução quando possível.

As métricas e testes realizados nos experimentos foram construídos de forma a serem iguais para ambos os ambientes. As únicas alterações realizadas foram as que influenciavam na execução devido a aparição de um terceiro elemento na característica “autores” do *dataset*.

No demais, este capítulo tem como intuito relatar os passos realizados durante os experimentos, além de dar uma ideia sobre os caminhos tomados até o resultado final.

5.1 MÉTRICAS

Devido a natureza desse trabalho, confiar única e exclusivamente na métrica de Acurácia pode não refletir realmente a eficácia do modelo, e portanto as métricas escolhidas para análise foram: Acurácia, Matriz de confusão, *F1-score, Precision* e *Recall*.

5.1.1 Acurácia

A acurácia é um método simples e amplamente utilizado para verificação da taxa de acerto de uma determinada ação. O cálculo é realizado levando em consideração a quantidade de elementos considerados como corretos (m) dividindo pela quantidade de elementos totais testados (n). O resultado dessa divisão fica na escala entre os valores 0 e 1, onde 0 seria nenhum acerto, e 1 todos os elementos testados tiveram o acerto positivo. O resultado também pode ser multiplicado por 100 para se obter o valor em porcentagem.

$$Acurcia = \frac{m}{n} * 100 \tag{5.1}$$

5.1.2 Matriz de confusão

Matriz de confusão é uma tabela de tamanho n x n, onde n é a quantidade de classes presentes nos dados. Essa tabela tem como intuito permitir a visualização da distribuição de acertos e erros da classificação de um grupo de elementos. De forma com que, é possível verificar qual é o verdadeiro comportamento de classificação da porcentagem fornecida pelo cálculo da acurácia, isto é, vai ajudar a validar a capacidade de classificação de cada tipo de classe presente no modelo testado. Um exemplo de matriz de confusão para duas classes pode ser visto na tabela a seguir.

Matriz de confusão é uma tabela de tamanho n x n, onde n é a quantidade de classes presentes nos dados. Essa tabela tem como intuito permitir a visualização da distribuição de acertos e erros da classificação de um grupo de elementos. De forma com que, é possível verificar qual é o verdadeiro comportamento de classificação da porcentagem fornecida pelo cálculo da acurácia, isto é, vai ajudar a validar a capacidade de classificação de cada tipo de classe presente no modelo testado. Um exemplo de matriz de confusão para duas classes pode ser visto na tabela a seguir.

No caso do exemplo acima, é possível ver que o modelo recebeu 70 elementos (35 para cada classe) para realizar classificações. Dos 35 elementos que são verdadeiros, o modelo classificou erroneamente 5 elementos verdadeiros como sendo da classe falso, de forma com que

	Verdadeiro	Falso
Verdadeiro	30	5
Falso	3	32

Tabela 5.1: Exemplo de uma matriz de confusão

o modelo obteve uma taxa de acerto de 85% na classificação para elementos da classe verdadeiro. De forma similar podemos verificar que dos 35 elementos da classe falso, 3 elementos foram erroneamente classificados como sendo pertencentes a classe verdadeira, sendo assim, o modelo obteve uma taxa de acerto de 91% para a classificação dos elementos da classe falso. Se fosse analisada apenas a métrica de acurácia, teríamos o valor de 92% de acerto do modelo, entretanto não teríamos nenhuma informação sobre a eficácia de acerto para cada tipo de classe presente no teste.

5.1.3 Recall

Recall ou sensibilidade é o nome dado ao cálculo que leva em consideração a relação de elementos de uma determinada classe (P), que foram corretamente classificados. Em outras palavras, essa métrica nos informa a relação numérica de quantos elementos o modelo utilizado está classificando de forma correta. Pegando como exemplo a matriz de confusão do da seção anterior, o cálculo do *recall* é feito da seguinte forma:

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

Onde TP é referente a classificação *True Positive* (elementos corretamente classificado para a classe em questão) e FN é referente a classificação *False Negative* (elementos que foram erroneamente classificados da classe verificada).

5.1.4 Precision

A métrica *Precision* ou precisão é relacionada a verificação de todas as classificações positivas, isto é, inclui previsões *True Positive* (TP) e *False Positive* (FP). Onde FP é o valor de elementos de outras classes que foram classificados erroneamente como a classe "positiva". O cálculo é feito da seguinte forma:

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

Esse resultado ajuda a verificar a escala de falsos positivos. Como neste trabalho desejamos possuir a menor quantidade possível de False Positive. Essa métrica é algo interessante a ser olhada também.

5.1.5 F1-score

E por último a métrica F1-Score, essa métrica é uma forma de visualizar de forma conjunta os valores obtidos pelas métricas Precision e Recall. Essa função permite com um valor apenas que existem problemas de valores baixos ou na Precision ou no Recall. A forma de calcular essa métrica é realizada com uma média harmônica simples 5.4.

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.4}$$

5.2 DESCRIÇÃO DOS EXPERIMENTOS

Os experimentos foram feitos de duas formas, uma etapa com o *dataset* composto de apenas duas classes: José e Machado, e a outra etapa com o *dataset* completo, composto das três classes: Jose, Machado e Outro.

As etapas dos experimentos consistiram em: (i) utilizar o *dataset* simplificado com 2 classes e verificar como os resultados se comportaram; (ii) adição de uma classe de ruído ao *dataset* simplificado, e verificação dos resultados obtidos; (iii) testagem dos modelos com diferentes configurações para a rede neural LSTM, (iv) com os modelos treinados, utilização do *dataset* de testagem simples e generalizada para verificar como o modelo se comporta com dados externos aos que existem no *dataset* de treino.

5.3 MODELOS

Os modelos mencionados nesta seção são utilizados para ambas as versões de experimentação (*dataset* com duas classes e três classes) com modificações realizadas apenas em alguns parâmetros para funcionamento com uma classe adicional.

Como anteriormente mencionado, os modelos utilizados neste trabalho foram construídos usando como base a rede neural LSTM, entretanto, como a maior parte dos textos utilizados no *dataset* possuem uma versão da língua portuguesa posterior a reforma ortográfica atualmente em vigor, foi levado isso em consideração, e portanto, foi decidido realizar uma variação de modelo com a utilização de uma matriz de *embeddings* pré-treinadas com o corpus do português na reforma ortográfica atual.

Os *embeddings* pré-treinados foram obtidos na plataforma do NILC[1] na opção de treinamento usando Word2Vec com o modelo CBOW de 1000 dimensões.

5.3.1 Configurações dos modelos

Para este trabalho, foi escolhido utilizar alguns valores de argumentos em comum entre os modelos, as configurações com valores diferentes serão mencionadas mais a frente no texto.

Para a situação de testagem e aprendizado para duas classes, as configurações foram: função de ativação *sigmoid* para camada Dense; rate de 0.2 na camada *SpartialDropout1D*; na camada de *embedding* o argumento de *trainable* foi deixado como *True*; na camada do LSTM o *dropout* foi deixado com o valor de 0.2; na função de *loss* foi utilizado *binary_crossentropy*; para *optimizer* foi utilizado o algoritmo 'Adam'. Para a situação de testagem e aprendizado para três classes, as configurações foram: função de ativação *softmax* para camada Dense; rate de 0.2 na camada *SpartialDropout1D*; na camada de *embedding* o argumento de *trainable* foi deixado como *True*; na camada do LSTM o *dropout* foi deixado com o valor de 0.2; na função de *loss* foi utilizado *categorical_crossentropy* para os modelos com 2 classes; para *optimizer* foi utilizado o algoritmo 'Adam'.

A quantidade máxima de palavras únicas presentes no trabalho foi colocada com o valor arbitrário inicial de 50.000. Esse valor é atualizado após a tokenização do texto, de forma com que é feita uma comparação com o tamanho do dicionário de palavras (*tokens*) únicos obtido pela função de tokenização, e então é realizado a atualização da variável que representa esse valor.

O valor da maior sentença do trabalho ficou definido como exatamente 100.

O tamanho da dimensão de *embedding* foi escolhido arbitrariamente com um valor de 222.

[1] http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc

Ambas versões dos *datasets* de treinamento foram divididos de forma igualitária. Utilizando da ferramenta do *sklearn*[2], foi realizado a divisão do dataset em treino e teste, utilizando a proporção de 70% para treino e 30% para teste. Então as partições ficaram:

	Dataset 2 Classes	Dataset 3 Classes
Treino	1584	2377
Teste	680	1019

Tabela 5.2: elementos de treinamento

Os valores finais após divisão variam devido a variação de elementos presentes no *dataset* devido a adição da terceira classe de autor.

Para os ajustes/treinamento do modelo, foram utilizado de 50 épocas, com lotes de tamanho 32, foi optado por utilizar os dados 30% dos dados do *dataset* como elementos de validação durante o treinamento.

5.3.2 Modelo com Embedding pré-treinado

O modelo com *embeddings* pré-treinados passou por alguns passos adicionais em relação aos outros modelos. Os passos adicionais foram: a inicialização de um dicionário com as palavras presentes no modelo pré-treinado junto aos seus coeficientes, criação de uma matriz de pesos para os *embeddings*.

O arquivo do modelo pré-treinado tem apenas duas colunas de características: palavra, vetor de *embedding* relacionados a palavra. Como é possível ver no exemplo a seguir:

Palavra	Embedding
sua	[-0.102091 -0.083437 (...) 0.014560 0.125399 -0.279959]

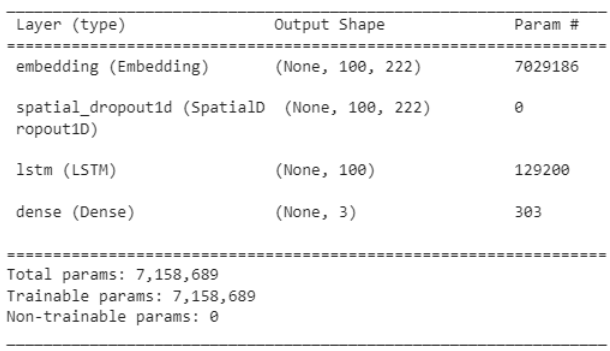
Tabela 5.3: Representação da palavra

O processo de criação do dicionário é realizado durante a leitura do arquivo do modelo utilizado, cada chave do dicionário vai ser referente a um termo presente no modelo, e o valor correspondente a chave vai ser o vetor de pesos daquela palavra.

O processo de carregamento da matriz de *embeddings* é realizado logo após a criação do dicionário. Nesta etapa é levado em consideração os *tokens* obtidos dos textos utilizados no *dataset*. A matriz é carregada usando o seguinte conjunto de passos: (i) laço de iteração para pegar cada elemento do dicionário de palavras providos do *dataset*; (ii) verifica se a palavra está presente no modelo pré-treinado, caso esteja, é adicionado os pesos da palavra na matriz de *embeddings*, caso contrário, os pesos são inicializados de forma aleatória por uma função do Numpy e então adicionados na matriz; (iii) passo (i) e (ii) se repetem até que todos os elementos tenham sido carregados. A matriz final ficou com a dimensão: 31633 x 222.

Com a matriz pronta, é possível passar ela como parâmetro de pesos iniciais para a camada de *Embeddings* do modelo. Um exemplo do sumario gerado pelo modelo com *embedding* pode ser visto na figura 5.1.

[2]https://scikit-learn.org/stable/



Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 222)	7029186
spatial_dropout1d (SpatialD ropout1D)	(None, 100, 222)	0
lstm (LSTM)	(None, 100)	129200
dense (Dense)	(None, 3)	303

Total params: 7,158,689
Trainable params: 7,158,689
Non-trainable params: 0

Figura 5.1: Sumário do modelo com *embedding* para 3 classes

5.3.3 Modelo sem Embedding pré-treinado

Para esses modelos não foi utilizado uma matriz pré-carregada de *embedding*. Portanto, foi utilizado a versão padrão da camada de *Embedding* do Keras, em que, caso não seja passado uma matriz com pesos iniciais, é realizado um processo de criação por conta própria, com todos os valores providos por uma distribuição uniforme.

Foram feitos dois modelos para essa versão, onde um dos modelos é uma versão simples com os valores de configuração sendo uma combinação dos valores padrões e das configurações generalizadas mencionadas anteriormente. O outro modelo utiliza de configurações padrões na maior parte das camadas, entretanto recebeu modificações nos valores da camada do LSTM com o intuito de poder utilizar a tecnologia cuDNN[3].

No modelo que utiliza da tecnologia cuDNN as modificações realizadas na camada LSTM foram: *recurrent_dropout* igual a 0, *activation* com função *tanh*, *recurrent_activation* com a função *sigmoid*, *unroll* deixado como *False*, e *use_bias* deixado como *True*. Essas configurações foram pegas diretamente da documentação do Keras[4].

A motivação de escolha para utilização de cuDNN surgiu pela ideia de se verificar se um modelo com configurações específicas a performance de execução teriam um resultado diferente de um modelo simples e despretensioso.

[3]https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[4]https://keras.io/api/layers/recurrent_layers/lstm/

6 RESULTADOS

Essa seção tem como objetivo principal demonstrar os resultados obtidos após a realização dos experimentos propostos na seção 5. Os resultados vão ser divididos por classes.

6.1 MODELO COM DUAS CLASSES

Como mencionado anteriormente, para este trabalho não era possível se chegar a qualquer conclusão analisando apenas a acurácia do modelo, e portanto foram utilizadas combinações de métricas para tentar se chegar a algum tipo de conclusão.

A classificação de cada elemento pelo modelo tem o formato de um vetor com dois elementos, onde a posição 1 do vetor representa o autor Machado de Assis, e a posição 0 do vetor representa o autor José de Alencar. O vetor resposta então fornece uma probabilidade daquela entrada ser de determinado autor, um exemplo desse vetor seria [0.95678 0.0263]. No caso deste exemplo, então a frase seria atribuída ao autor José de Alencar.

6.1.1 Modelo com Embeddings pré-treinados

O modelo com *embeddings* pré-treinados obteve durante o treinamento o valor de acurácia total de 0.95882 e o valor de *loss* de 0.27987. Esses valores por si só não nos fornecem muitas informações, então ao analisar o gráfico de *loss* 6.1, é possível ver que durante o treino o valor tem uma aparência de ser quase constante ao longo das épocas (eixo x do gráfico), enquanto durante o teste o valor de *loss* sofre com grandes oscilações ao decorrer das épocas. Isso pode indicar que a quantidade de dados para treinamento é insuficiente para que o modelo, de forma que durante o treino o modelo aparenta ter decorado alguns aspectos dos dados e isso fez com que durante o teste a classificação não possuísse boas capacidades de classificação. Esse argumento pode ser reforçado analisando o gráfico de acurácia 6.2 durante o treinamento do modelo. Como podemos ver, após uma certa quantidade de épocas, o treinamento apresenta acurácia fixa em 1.0, algo que não acontece durante o teste dos dados.

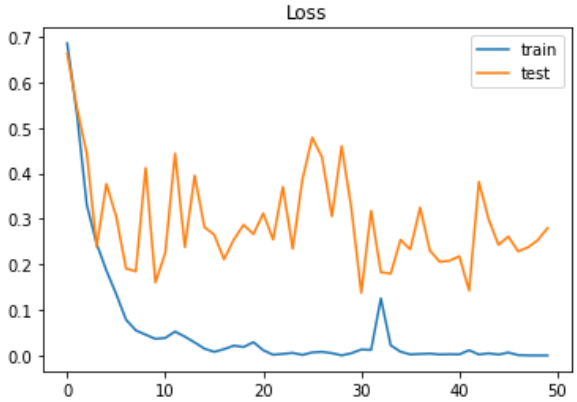


Figura 6.1: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo com *embeddings* pré-treinados 2 Classes

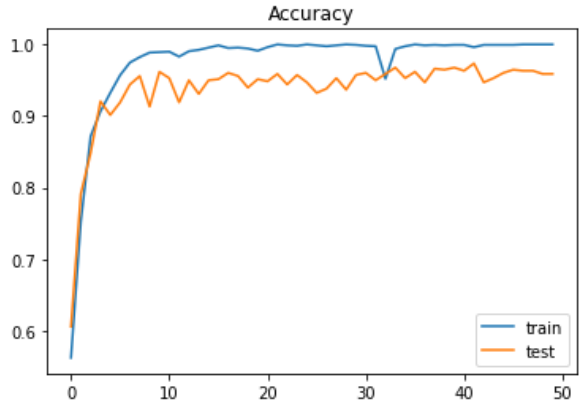


Figura 6.2: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo com *embeddings* pré-treinados 2 Classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos na tabela 6.1. Na tabela 6.2, é possível ver que dos 660 elementos 28 foram erroneamente classificados, podem parecer poucos elementos, mas em uma situação de atribuição de autoria, isso não poderia acontecer.

	Precision	Recall	F1-score	Support
José	0.96	0.96	0.96	337
Machado	0.96	0.96	0.96	343
Accuracy			0.96	680
Macro avg	0.96	0.96	0.96	680
Weighted avg	0.96	0.96	0.96	680

Tabela 6.1: Detalhes da classificação do treinamento do modelo com *embeddings* pré-treinados 2 classes

	José	Machado
José	322	15
Machado	13	330

Tabela 6.2: Matriz de confusão do treinamento do modelo com *embeddings* pré-treinados 2 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.3. Analisando a tabela 6.4 é possível ver que para a classificação de Machado, ocorreu um falso negativo, reforçando ainda mais a possibilidade do problema de falta de dados de treinamento.

	Precision	Recall	F1-score	Support
José	0.91	1.00	0.95	10
Machado	1.00	0.90	0.95	10
Accuracy			0.95	20
Macro avg	0.95	0.95	0.95	20
Weighted avg	0.95	0.95	0.95	20

Tabela 6.3: Detalhes da classificação do teste simples do modelo com *embeddings* pré-treinados 2 classes

	José	Machado
José	10	0
Machado	1	9

Tabela 6.4: Matriz de confusão do teste simples do modelo com *embeddings* pré-treinados 2 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.5. Analisando a matriz de confusão 6.6 é possível perceber a tendência do modelo classificar textos do Machado de Assis como sendo de autoria do José de Alencar.

	Precision	Recall	F1-score	Support
José	0.83	0.95	0.89	37
Machado	0.94	0.81	0.87	37
Accuracy			0.88	74
Macro avg	0.89	0.88	0.88	74
Weighted avg	0.89	0.88	0.88	74

Tabela 6.5: Detalhes da classificação do teste generalizado do modelo com *embeddings* pré-treinados 2 classes

	José	Machado
José	35	2
Machado	7	30

Tabela 6.6: Matriz de confusão do teste generalizado do modelo com *embeddings* pré-treinados 2 classes

6.1.2 Modelo simples

O modelo simples não utiliza a matriz de *embeddings* pré treinados. Para o treinamento desta versão, foram obtidos os resultados de 0.96617 para acurácia total e um valor de 0.30377 para o *loss*. Como mencionado anteriormente, esses valores não nos fornecem informações detalhadas e muito menos sobre a qualidade de classificação do modelo, portanto analisando o gráfico de *loss* 6.3 é possível ver que ocorre a mesma tendência relatada anteriormente, durante o treinamento o *loss* fica com valores quase que constantes, algo que não acontece durante o teste. Verificando agora o gráfico de acurácia 6.4, podemos visualizar uma acurácia quase constante durante o treino. Essas características adicionam mais um pouco de reforço à hipótese que o modelo está com poucos dados para treinamento.

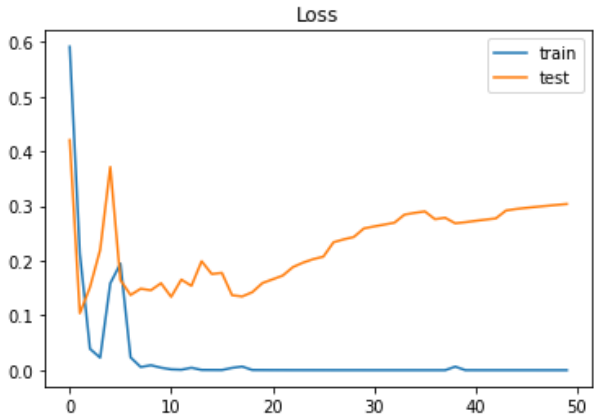


Figura 6.3: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo simples 2 Classes

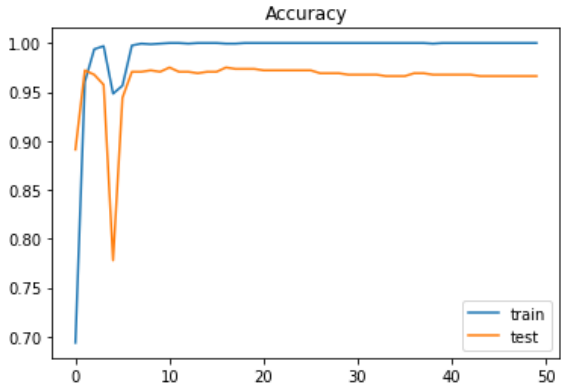


Figura 6.4: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo simples 2 Classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos em detalhes na tabela 6.7. Na tabela 6.8, é possível ver que dos 660 elementos 23 foram erroneamente classificados, quando comparamos com a matriz de confusão 6.2 os valores foram um pouco menores para o modelo simples, mas com o diferencial de agora podermos ver de forma mais clara uma tendência do modelo em classificar os textos do Machado de Assis como sendo de autoria do José de Alencar.

	Precision	Recall	F1-score	Support
José	0.95	0.98	0.97	337
Machado	0.98	0.95	0.97	343
Accuracy			0.97	680
Macro avg	0.97	0.97	0.97	680
Weighted avg	0.97	0.97	0.97	680

Tabela 6.7: Detalhes da classificação do treinamento do modelo simples de 2 classes

	José	Machado
José	330	7
Machado	16	327

Tabela 6.8: Matriz de confusão do treinamento do modelo simples com 2 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.9. Na tabela 6.10 vemos uma classificação perfeita para ambos autores, o que demonstra um cenário perfeito. Entretanto, esse resultado pode levantar uma pergunta de grande importância para a qualidade do modelo. Devido a origem dos dados do teste simples, podemos levantar a questão referente à: "Esses resultados indicam que características presentes nos livros de treinamento foram perfeitamente decoradas ou que o modelo aprendeu realmente a generalizar a classificação?". Para essa resposta precisamos analisar os resultados referentes ao teste generalizado.

	Precision	Recall	F1-score	Support
José	1.00	1.00	1.00	10
Machado	1.00	1.00	1.00	10
Accuracy			1.00	20
Macro avg	1.00	1.00	1.00	20
Weighted avg	1.00	1.00	1.00	20

Tabela 6.9: Detalhes da classificação do teste simples do modelo simples de 2 classes

	José	Machado
José	10	0
Machado	0	10

Tabela 6.10: Matriz de confusão do teste simples do modelo simples com 2 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.11. Na tabela 6.12 podemos responder a pergunta levantada a pouco, aqui vemos que o modelo aprendeu algumas características que descrevem o estilo de escrita de cada um dos autores, mas não aprendeu de forma perfeita, tanto que 8 elementos foram classificados de forma errada.

	Precision	Recall	F1-score	Support
José	0.89	0.89	0.89	37
Machado	0.89	0.89	0.89	37
Accuracy			0.89	74
Macro avg	0.89	0.89	0.89	74
Weighted avg	0.89	0.89	0.89	74

Tabela 6.11: Detalhes da classificação do teste generalizado do modelo simples de 2 classes

	José	Machado
José	33	4
Machado	4	33

Tabela 6.12: Matriz de confusão do teste generalizado do modelo simples com 2 classes

6.1.3 Modelo com cuDNN

O modelo que utiliza a tecnologia *cuDNN* não utiliza a matriz de *embeddings* pré-treinados. No treinamento desta versão foram obtidos os seguintes resultados: acurácia total com valor de 0.9750, e o valor de *loss* total como sendo 0.1061. Olhando com mais atenção o comportamento do modelo durante o treinamento, podemos ver no gráfico de *loss* 6.5 a característica de pouca variação durante o treinamento, mas picos muito ruins durante o teste de classificação. No gráfico de acurácia 6.6, podemos ver que o maior pico de *loss* é refletido em uma queda enorme de acurácia na classificação do modelo durante o teste, e que de forma similar aos outros dois modelos apresentados anteriormente, vemos uma acurácia quase constante em 1.

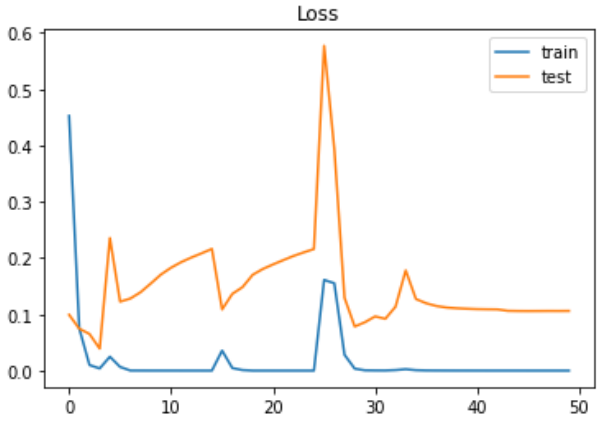


Figura 6.5: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo com cuDNN 2 Classes

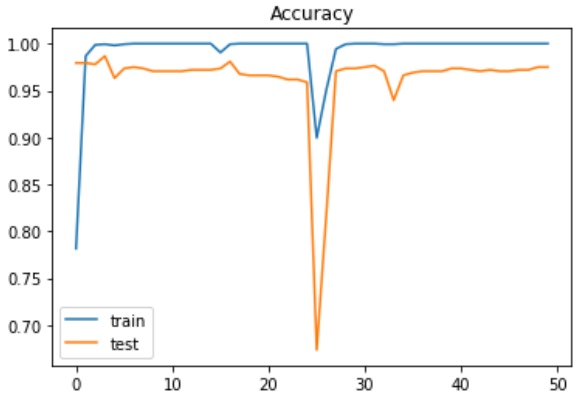


Figura 6.6: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo com cuDNN 2 Classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos em detalhes na tabela 6.13. Na tabela 6.14, é possível ver que dos 660 elementos 17 foram erroneamente classificados, quando comparamos com a matriz de confusão 6.2 e 6.8, os valores demonstram ser um pouco menores para este modelo "otimizado", mas com o diferencial que dessa vez o modelo tem uma tendência maior a classificar os textos do José de Alencar como sendo textos de autoria do Machado de Assis.

	Precision	Recall	F1-score	Support
José	0.98	0.96	0.97	337
Machado	0.97	0.99	0.98	343
Accuracy			0.97	680
Macro avg	0.98	0.97	0.97	680
Weighted avg	0.98	0.97	0.97	680

Tabela 6.13: Detalhes da classificação do treinamento do modelo com cuDNN de 2 classes

	José	Machado
José	325	12
Machado	5	338

Tabela 6.14: Matriz de confusão do treinamento do modelo com cuDNN de 2 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.15. Na tabela 6.16 vemos uma classificação perfeita para ambos autores, o que demonstra um cenário perfeito, de forma similar ao modelo simples. Entretanto, a mesma pergunta feita no modelo simples pode ser feita para essa versão do modelo. "Esses resultados indicam que características presentes nos livros de treinamento foram perfeitamente decoradas ou que o modelo aprendeu realmente a generalizar a classificação?". E novamente, para termos alguma noção da resposta, vai ser necessário analisar os resultados referentes ao teste generalizado.

	Precision	Recall	F1-score	Support
José	1.00	1.00	1.00	10
Machado	1.00	1.00	1.00	10
Accuracy			1.00	20
Macro avg	1.00	1.00	1.00	20
Weighted avg	1.00	1.00	1.00	20

Tabela 6.15: Detalhes da classificação do teste simples do modelo com cuDNN de 2 classes

	José	Machado
José	10	0
Machado	0	10

Tabela 6.16: Matriz de confusão do teste simples do modelo com cuDNN de 2 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.17. Na tabela 6.18 podemos responder a pergunta levantada a pouco, aqui vemos que o modelo aprendeu algumas características que descrevem o estilo de escrita de cada um dos autores, mas não aprendeu de forma perfeita, tanto que 7 elementos foram classificados de forma errada. Inclusive, dessa vez é possível ver novamente a presença de um vício do modelo em classificar os textos do José de Alencar como sendo textos de autoria do Machado de Assis.

	Precision	Recall	F1-score	Support
José	0.92	0.89	0.90	37
Machado	0.89	0.92	0.91	37
Accuracy			0.91	74
Macro avg	0.91	0.91	0.91	74
Weighted avg	0.91	0.91	0.91	74

Tabela 6.17: Detalhes da classificação do teste generalizado do modelo com cuDNN de 2 classes

	José	Machado
José	33	4
Machado	3	34

Tabela 6.18: Matriz de confusão do teste generalizado do modelo com cuDNN de 2 classes

6.1.4 Considerações para os modelos com 2 classes

Como visto nos resultados para as três versões de modelos LSTM utilizados, existe uma grande possibilidade de que a pouca quantidade de dados utilizados para treinamento tenham

influenciado para a grande quantidade de falsas classificações. Claro que, não podemos descartar a possibilidade dos modelos não terem sido calibrados bem o suficiente para essa tarefa, entretanto como o foco do trabalho não estava em obter as melhores versões possíveis de configuração para os modelos, essa hipótese não pode ser refutada com base apenas nos resultados apresentados neste trabalho.

De uma forma geral, vemos que os modelos tiveram a capacidade de aprender algum tipo de característica presente nos textos dos autores utilizados, o que é animador, tendo em vista a pouca quantidade de dados disponíveis para o treinamento. Pode se usar como argumento para justificar a performance ruim do modelo de *embeddings* pré-treinados em relação aos outros dois modelos, se encontra no fato que o corpus utilizado para o treinamento desses *embeddings* não possui um português exatamente igual ao dos textos retirados dos livros, isso se deve ao fato de que como mencionado anteriormente, o português presente nos livros é de uma reforma da língua anterior a dos dados presentes nesses corpus de treinamento, ou seja, a diferença de escrita de diversas palavras influenciou negativamente na capacidade de classificação do modelo que levava *embeddings* pré-treinados.

6.2 MODELO COM TRÊS CLASSES

Como mencionado anteriormente, para este trabalho não era possível se chegar a qualquer conclusão analisando apenas a acurácia do modelo, e portanto foram utilizadas combinações de métricas para tentar se chegar a algum tipo de conclusão.

A classificação de cada elemento pelo modelo tem o formato de um vetor com três elementos: a posição 0 do vetor representa o autor José de Alencar, onde a posição 1 do vetor representa o autor Machado de Assis, e a posição 2 do vetor representa os dados outro (Wikipédia). O vetor resposta então fornece uma probabilidade daquela entrada ser de determinado autor, um exemplo desse vetor seria [0.95678 0.0263 0.4356]. No caso deste exemplo, então a frase seria atribuída ao autor José de Alencar.

6.2.1 Modelo com embeddings pré-treinados

O modelo com *embedding* pré-treinados obteve durante o treinamento o valor de acurácia total de 0.94897 e o valor de *loss* de 0.23295. Esses valores por si só não nos fornecem muitas informações, então ao analisar o gráfico de *loss* 6.7, é possível ver que durante o treino o valor tem uma aparência de ser quase constante ao longo das épocas (eixo x do gráfico), enquanto durante o teste o valor de *loss* sofre com grandes oscilações ao decorrer das épocas. Isso pode indicar que a quantidade de dados para treinamento é insuficiente para que o modelo, de forma que durante o treino o modelo aparenta ter decorado alguns aspectos dos dados e isso fez com que durante o teste a classificação não possuísse boas capacidades de classificação. Esse argumento pode ser reforçado analisando o gráfico de acurácia 6.8 durante o treinamento do modelo. Como podemos ver, após uma certa quantidade de épocas, o treinamento apresenta acurácia fixa em 1, algo que não acontece durante o teste dos dados.

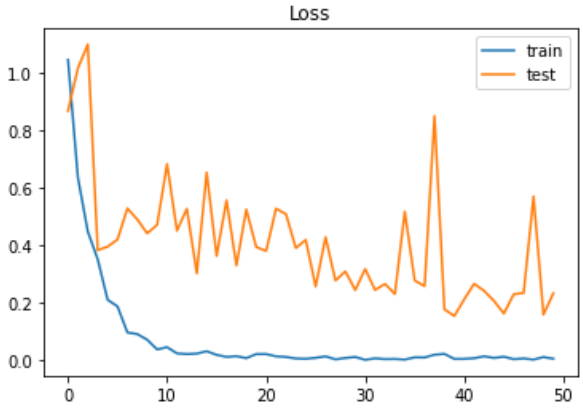


Figura 6.7: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo com *embeddings* pré-treinados 3 classes

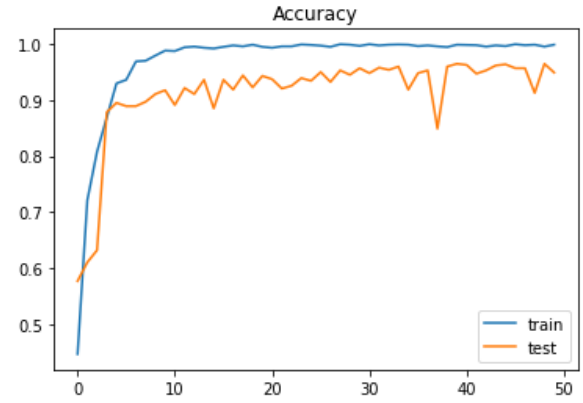


Figura 6.8: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo com *embeddings* pré-treinados 3 classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos na tabela 6.19. Na tabela 6.20, é possível ver que dos 1019 elementos 52 foram erroneamente classificados, podem parecer poucos elementos, mas em uma situação de atribuição de autoria, isso não poderia acontecer. Analisando a matriz de confusão 6.20 podemos ver dois potenciais motivos de preocupação: o primeiro fato preocupante é quando consideramos que ocorreram 3 classificações de falso positivo para a classe outro, isso é preocupante pois os dados da classe outro em sua maioria possuem grande diferença de estilo de escrita quando comparado aos textos dos outros dois autores, então esses falsos positivos podem não ser um bom sinal, e o segundo fato preocupante é referente a tendência do modelo estar viciado em classificar textos do Machado de Assis como sendo de autoria do José de Alencar.

	Precision	Recall	F1-score	Support
José	0.89	0.99	0.94	354
Machado	0.97	0.89	0.93	330
Outro	0.99	0.97	0.98	335
Accuracy			0.95	1019
Macro avg	0.95	0.95	0.95	1019
Weighted avg	0.95	0.95	0.95	1019

Tabela 6.19: Detalhes da classificação do treinamento do modelo com *embeddings* pré-treinados de 3 classes

	José	Machado	Outro
José	349	3	2
Machado	36	293	1
Outro	5	5	325

Tabela 6.20: Matriz de confusão do treinamento do modelo com *embeddings* pré-treinados de 3 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.21. Analisando a tabela 6.22 é possível ver que para a classificação de Machado, ocorreu um falso negativo, dando um pouco mais de reforço à hipótese de vício do modelo para a classe do José de Alencar.

	Precision	Recall	F1-score	Support
José	0.91	1.00	0.95	10
Machado	1.00	0.90	0.95	10
Outro	1.00	1.00	1.00	10
Accuracy			0.97	30
Macro avg	0.97	0.97	0.97	30
Weighted avg	0.97	0.97	0.97	30

Tabela 6.21: Detalhes da classificação do teste simples do modelo com *embeddings* pré-treinados de 3 classes

	José	Machado	Outro
José	10	0	0
Machado	1	9	0
Outro	0	0	10

Tabela 6.22: Matriz de confusão do teste simples do modelo com *embeddings* pré-treinados de 3 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.23. Analisando a tabela 6.24 é possível bater o martelo sobre a tendência do modelo classificar textos do Machado de Assis como sendo de autoria do José de Alencar. E novamente é possível ver duas classificações de falso positivo para a classe outro, isso demonstra que apesar de o modelo ter aprendido algumas características chave para classificação, essas características estão incompletas ou de alguma forma erradas. Isso pode ser atribuído a pouca quantidade de dados para treinamento dos modelos.

	Precision	Recall	F1-score	Support
José	0.81	0.92	0.86	37
Machado	0.97	0.78	0.87	37
Outro	0.95	1.00	0.97	37
Accuracy			0.90	111
Macro avg	0.91	0.90	0.90	111
Weighted avg	0.91	0.90	0.90	111

Tabela 6.23: Detalhes da classificação do teste generalizado do modelo com *embeddings* pré-treinados de 3 classes

	José	Machado	Outro
José	34	1	2
Machado	8	29	0
Outro	0	0	37

Tabela 6.24: Matriz de confusão do teste generalizado do modelo com *embeddings* pré-treinados de 3 classes

6.2.2 Modelo simples

O modelo simples não utiliza a matriz de *embeddings* pré treinados. Para o treinamento desta versão, foram obtidos os resultados de 0.97645 para acurácia total e um valor de 0.13024 para o *loss*. Como mencionado anteriormente, esses valores não nos fornecem informações detalhadas e muito menos sobre a qualidade de classificação do modelo, portanto analisando o gráfico de *loss* 6.9 é possível ver que ocorre a mesma tendência relatada anteriormente, durante o treinamento o *loss* fica com valores quase que constantes, algo que não acontece durante o teste. Verificando agora o gráfico de acurácia 6.10, podemos visualizar uma acurácia quase constante durante o treino. Essas características adicionam mais um pouco de reforço à hipótese que o modelo está com poucos dados para treinamento.

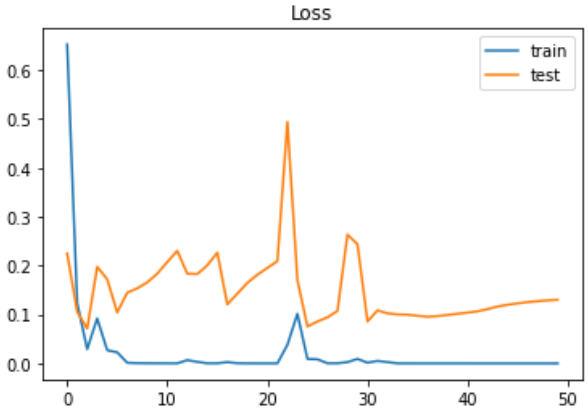


Figura 6.9: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo simples de 3 classes

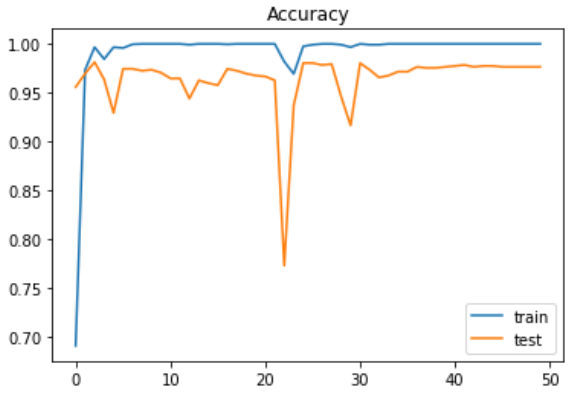


Figura 6.10: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo simples de 3 classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos em detalhes na tabela 6.25. Na tabela 6.26, é possível ver que dos 1019 elementos 24 foram erroneamente classificados, quando comparamos com a matriz de confusão 6.20 os valores foram um pouco maiores para o modelo simples, mas com o diferencial de agora podermos ver uma possível existência de um vício do modelo em classificar os textos do José de Alencar como sendo do Machado de Assis.

	Precision	Recall	F1-score	Support
José	0.98	0.96	0.97	354
Machado	0.96	0.98	0.97	330
Outro	0.99	1.00	0.99	335
Accuracy			0.98	1019
Macro avg	0.98	0.98	0.98	1019
Weighted avg	0.98	0.98	0.98	1019

Tabela 6.25: Detalhes da classificação do treinamento do modelo simples de 3 classes

	José	Machado	Outro
José	339	13	2
Machado	6	322	2
Outro	0	1	334

Tabela 6.26: Matriz de confusão do treinamento do modelo simples de 3 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.27. Na tabela 6.28 vemos uma classificação perfeita para as classes de machado e outro, mas com um falso negativo por parte da classe do jose, mostrando mais um pouco sobre a possibilidade do vício na classe do machado.

	Precision	Recall	F1-score	Support
José	1.00	0.90	0.95	10
Machado	0.91	1.00	0.95	10
Outro	1.00	1.00	1.00	10
Accuracy			0.97	30
Macro avg	0.97	0.97	0.97	30
Weighted avg	0.97	0.97	0.97	30

Tabela 6.27: Detalhes da classificação do teste simples do modelo simples de 3 classes

	José	Machado	Outro
José	9	1	0
Machado	0	10	0
Outro	0	0	10

Tabela 6.28: Matriz de confusão do teste simples do modelo simples de 3 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.29. Na tabela 6.30 podemos ver que o vício na classe do machado não existe, já que com dados generalizados jose não classificou nenhum machado, de forma contrária agora vemos problemas de classificação do machado em relação ao josé. E novamente vemos falsos positivos para outro, isso demonstra que por mais que os modelos tenham aprendido características para determinar o estilo de escrita do José de Alencar e do Machado de Assis, parece que essas características estão incompletas, de forma que o modelo apenas classifica de forma aleatória quando ele não consegue extrair alguma característica chave.

	Precision	Recall	F1-score	Support
José	0.85	0.89	0.87	37
Machado	1.00	0.81	0.90	37
Outro	0.88	1.00	0.94	37
Accuracy			0.90	111
Macro avg	0.91	0.90	0.90	111
Weighted avg	0.91	0.90	0.90	111

Tabela 6.29: Detalhes da classificação do teste generalizado do modelo simples de 3 classes

	José	Machado	Outro
José	33	0	4
Machado	6	30	1
Outro	0	0	37

Tabela 6.30: Matriz de confusão do teste generalizado do modelo simples de 3 classes

6.2.3 Modelo com cuDNN

O modelo que utiliza a tecnologia *cuDNN* não utiliza a matriz de *embeddings* pré treinados. No treinamento desta versão foram obtidos os seguintes resultados: acurácia total com valor de 0.97841, e o valor de *loss* total como sendo 0.14740. Olhando com mais atenção

o comportamento do modelo durante o treinamento, podemos ver no gráfico de *loss* 6.11 a característica de pouca variação durante o treinamento, mas alguns picos ruins durante a parte de teste. No gráfico de acurácia 6.12, podemos ver que por mais que ocorreram quedas no valor da acurácia durante o teste, o comportamento durante o teste se assemelha muito ao comportamento durante o treino.

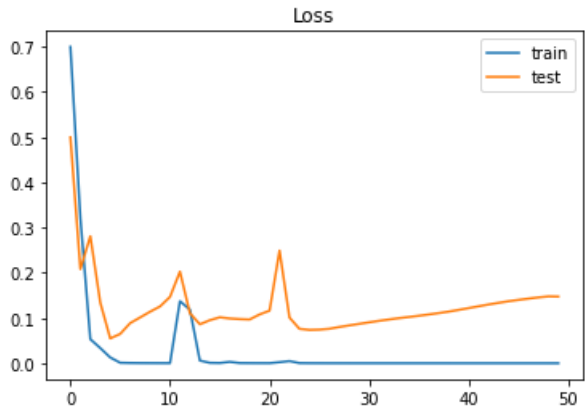


Figura 6.11: Gráfico de *loss* do modelo ao longo das épocas de treinamento. Modelo com cuDNN 3 classes

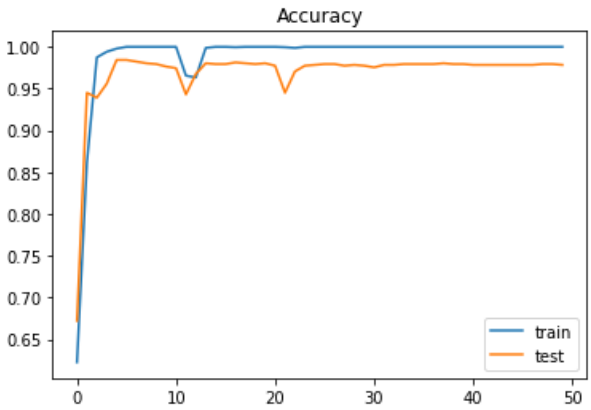


Figura 6.12: Gráfico de acurácia do modelo ao longo das épocas de treinamento. Modelo com cuDNN 3 classes

Os resultados de predição do modelo ao se utilizar 30% dos dados do *dataset* de treinamento podem ser vistos em detalhes na tabela 6.31. Na tabela 6.32, é possível ver que dos 1019 elementos 22 foram erroneamente classificados, quando comparamos com a matriz de confusão 6.20 e 6.26, os valores demonstram ser um pouco menores para este modelo "otimizado", mas com o diferencial que dessa vez o modelo tem uma tendência maior a classificar os textos do José de Alencar como sendo textos de autoria do Machado de Assis.

	Precision	Recall	F1-score	Support
José	0.98	0.96	0.97	354
Machado	0.96	0.98	0.97	330
Outro	0.99	1.00	0.99	335
Accuracy			0.98	1019
Macro avg	0.98	0.98	0.98	1019
Weighted avg	0.98	0.98	0.98	1019

Tabela 6.31: Detalhes da classificação do treinamento do modelo com cuDNN de 3 classes

	José	Machado	Outro
José	339	13	2
Machado	6	322	2
Outro	0	1	334

Tabela 6.32: Matriz de confusão do treinamento do modelo com cuDNN de 3 classes

Os resultados referentes ao teste simples podem ser vistos na tabela 6.33. Na tabela 6.34 vemos uma classificação perfeita para jose e outro, mas uma classificação errada por parte do machado. Com isso já podemos quase descartar a possibilidade de vício do modelo para a classe do machado.

	Precision	Recall	F1-score	Support
José	1.00	0.90	0.95	10
Machado	0.91	1.00	0.95	10
Outro	1.00	1.00	1.00	10
Accuracy			0.97	30
Macro avg	0.97	0.97	0.97	30
Weighted avg	0.97	0.97	0.97	30

Tabela 6.33: Detalhes da classificação do teste simples do modelo com cuDNN de 3 classes

	José	Machado	Outro
José	9	1	0
Machado	0	10	0
Outro	0	0	10

Tabela 6.34: Matriz de confusão do teste simples do modelo com cuDNN de 3 classes

Os resultados referentes ao teste generalizado podem ser vistos na tabela 6.35. Na tabela 6.36 podemos descartar completamente a possibilidade do vício no modelo para qualquer classe. Dessa vez observamos apenas 1 falso positivo por parte da classe outro, logo, podemos dizer que essa versão do modelo LSTM conseguiu captar de forma mais eficiente características de escrita de cada uma das classes presentes no dataset.

	Precision	Recall	F1-score	Support
José	0.85	0.89	0.87	37
Machado	1.00	0.81	0.90	37
Outro	0.88	1.00	0.94	37
Accuracy			0.90	111
Macro avg	0.91	0.90	0.90	111
Weighted avg	0.91	0.90	0.90	111

Tabela 6.35: Detalhes da classificação do teste generalizado do modelo com cuDNN de 3 classes

	José	Machado	Outro
José	9	1	0
Machado	0	10	0
Outro	0	0	10

Tabela 6.36: Matriz de confusão do teste generalizado do modelo com cuDNN de 3 classes

6.2.4 Considerações para os modelos com 3 classes

Como visto nos resultados para as três versões de modelos LSTM utilizados, existe uma grande possibilidade de que a pouca quantidade de dados utilizados para treinamento tenham influenciado para a grande quantidade de falsas classificações. Claro que, não podemos descartar a possibilidade dos modelos não terem sido calibrados bem o suficiente para essa tarefa, entretanto como o foco do trabalho não estava em obter as melhores versões possíveis de configuração para os modelos, essa hipótese não pode ser refutada com base apenas nos resultados apresentados neste trabalho.

De uma forma geral, vemos que os modelos tiveram a capacidade de aprender algum tipo de característica presente nos textos dos autores utilizados, o que é animador, principalmente quando se leva em consideração que o conjunto de dados para essa fase do treinamento levava uma classe de ruído. Logo, a finalidade de utilização de um terceiro elemento foi cumprida com sucesso, e foi possível determinar que sim, os modelos realmente estavam aprendendo algumas características para classificações corretas, e não apenas decorando dados e ou classificando de forma aleatória. Novamente se pode observar uma pior performance por parte do modelo que utiliza *embeddings* pré-treinados, o mesmo argumento utilizado para a versão do modelo com 2 classes pode ser utilizado aqui para o modelo com 3 classes. Logo, a diferença do português presente nos livros e do presente nos corpos utilizados para treinamento podem ter influenciado na capacidade de classificação da versão que utilizava *embeddings* pré-treinados.

6.3 CONSIDERAÇÕES SOBRE OS RESULTADOS

Por mais que os modelos tenham dado resultados animadores, foi possível notar o quanto uma base de dados pequena pode influenciar para a atividade de classificação de autoria. Os modelos se comportaram de forma similares tanto em suas versões com duas classes quanto em suas versões com três classes, o que é um bom sinal, já que com a adição de ruído ao *dataset* era esperado que acontecesse uma diminuição na performance dos modelos para três classes.

A técnica do LSTM demonstrou ser bem eficaz para a tarefa de classificação de dados, mesmo que neste trabalho o foco não estivesse na configuração e otimização dos modelos, a versão mais simples da LSTM conseguiu realizar a tarefa de forma satisfatória. Ao longo deste capítulo foi possível entender o motivo do porquê não se deve confiar apenas nos valores de acurácia total do modelo. Várias informações importantes são ocultadas quando não se leva em consideração os aspectos fornecidos pelas informações de *recall* e *precision*, informações essas que foram de grande ajuda para análise da possibilidade de existência de vício de classificação para uma determinada classe, e para verificação do comportamento de classificação de cada modelo, possibilitando a possibilidade de classificação feita na base da sorte.

7 SUGESTÕES DE TRABALHOS FUTUROS

Durante a execução deste trabalho, surgiram várias perguntas que não tiveram oportunidade de serem colocadas a prova e consequentemente não foram respondidas, seja por motivos devido a falta de tempo para as desenvolver, ou por falta de recursos computacionais. O ponto é que essas dúvidas ficaram e podem ser utilizadas como continuação deste trabalho. As sugestões que eu como autor deixo são:

- Adicionar mais elementos a classe de autor e verificar como ficaria a matriz de confusão da classificação com a adição de novos autores.
- Comparar o modelo da rede neural LSTM com uma rede neural CNN. Para verificar como uma rede de convolução se diferencia de uma rede de recorrência para esse tipo de classificação de texto.
- Reprodução desse trabalho usando a tecnologia do BERT e comparar os resultados com a LSTM, e caso tenha feito com a CNN, comparar entre as três.

8 CONCLUSÃO

Este trabalho focou nos aspectos de construção de um *dataset* em português e da verificação da capacidade de classificação de uma rede neural LSTM para o idioma português.

A construção do *dataset* resultou por ser uma tarefa de grande importância, pois, como mencionado anteriormente, a escassez de dados para processamento de linguagem natural fora do idioma inglês é algo que prejudica o desenvolvimento da área para outros idiomas. Logo, a criação do *dataset* proposto neste trabalho contribui de forma significativa para a área de processamento de linguagem natural para o português brasileiro.

Durante a construção do *dataset* percebeu-se a necessidade de buscar por aspectos especiais de cada texto, pois, dessa forma é possível visualizar aspectos e características únicas de escrita de cada autor e de cada obra daquela época. Nesta etapa também se percebeu a importância de se manter as pontuações das sentenças e a acentuação de cada palavra, até mesmo a abreviação, pois, determinadas palavras poderiam aparecer de forma abreviada quando escritas pelo autor "A", enquanto o autor "B" mantinha a forma extensa do termo. Todas essas características contaram como uma forma de verificação de estilo de escrita de cada autor, de forma que o aprendizado desses aspectos pelas redes neurais era o que se esperava conseguir para se obter algum resultado.

A adição de ruídos simples demonstrou que por mais que exista diferença ortográfica entre os dados do Wikipédia e os dados dos livros, os modelos ainda classificaram alguns elementos de forma errônea. Logo, é possível dizer que o objetivo de verificação foi alcançado, e como é possível ver na seção de resultados, a diferença ortográfica não necessariamente causou mais facilidade para classificação dos dados

Como mencionado nos resultados, a rede LSTM demonstrou ser bastante eficaz para a tarefa de identificação de autoria, e por ser de fácil utilização, essa rede foi essencial para validação do *dataset* proposto. Mesmo que os modelos tenham sido pouco modificados em relação às versões da biblioteca com as implementações utilizadas, foi possível ver como os dados se comportaram com diferentes tipos de configurações, de forma que foi possível observar semelhanças entre os comportamentos dos resultados, independentemente da forma do modelo utilizado.

Os resultados para este estudo foram satisfatórios e apesar de fornecerem resultados aparentemente bons, ficou claro que existe a necessidade de uma grande quantidade de dados para se obter modelos mais precisos, logo, é possível concluir que não existe viabilidade de utilização dessa proposta para cenários que se deseja obter resultados similares aos obtidos por especialistas humanos, ou que possuam poucos dados para alimentação do modelo.

REFERÊNCIAS

Arbel, N. (2018). How lstm networks solve the problem of vanishing gradients. https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. Acessado em 09/09/2022.

Chen Qian, Tianchang He, R. Z. (2017). Deep learning based authorship identification. Relatório técnico, Stanford University, Stanford, Californa.

Chopra, A., Prashar, A. e Sain, C. (2013). Natural language processing. *International Journal of Technology Enhancements and Emerging Engineering Research*, 1:131–134.

DEY, V. (2021). When to use one-hot encoding in deep learning? https://analyticsindiamag.com/when-to-use-one-hot-encoding-in-deep-learning/. Acessado em 09/09/2022.

Education, I. C. (2020). Neural networks. https://www.ibm.com/cloud/learn/neural-networks. Acessado em 09/09/2022.

Eisenstein, J. (2019a). *Introduction to natural language processing*, página 16. MIT Press.

Eisenstein, J. (2019b). *Introduction to natural language processing*, página 312. MIT Press.

Eisenstein, J. (2019c). *Introduction to natural language processing*, página 311. MIT Press.

Google (2013). word2vec. https://code.google.com/archive/p/word2vec/. Acessado em 06/09/2022.

Jones, M. T. (2017). Recurrent neural networks deep dive. https://developer.ibm.com/articles/cc-cognitive-recurrent-neural-networks/. Acessado em 09/09/2022.

Keras (2022). About keras. https://keras.io/about/. Acessado em 06/09/2022.

Khurana, D., Koli, A., Khatter, K. e Singh, S. (2022). Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*.

Koppel, M. e Winter, Y. (2014). Determining if two documents are written by the same author. *Journal of the Association for Information Science and Technology*, 65.

Le, X. H., Ho, H., Lee, G. e Jung, S. (2019). Application of long short-term memory (lstm) neural network for flood forecasting. *Water*, 11:1387.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M. e Gao, J. (2020). Deep learning based text classification: A comprehensive review.

Sanderson, C. e Guenter, S. (2006). Short text authorship attribution via sequence kernels, Markov chains and author unmasking: An investigation. Em *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, páginas 482–491, Sydney, Australia. Association for Computational Linguistics.

Semicolon, T. (2018). Word2vec - skipgram and cbow. https://www.youtube.com/watch?v=UqRCEmrv1gQ&t=321s. Acessado em 06/09/2022.

Sousa-Silva, R. (2020). *Plagiarism Across Languages and Cultures: A (Forensic) Linguistic Analysis*, páginas 2325–2345. Springer International Publishing, Cham.

Wang, S.-C. (2003). *Artificial Neural Network*, páginas 81–100. Springer US, Boston, MA.