UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Fernando Battisti

Detecção de Notícias Falsas Utilizando Aprendizado de Máquina

Florianópolis
2020

Fernando Battisti

Detecção de Notícias Falsas Utilizando Aprendizado de Máquina

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Eric Aislan Antonelo, Dr.
Supervisor: Prof. Eric Aislan Antonelo, Dr.

Florianópolis
2020

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Fernando Battisti

Detecção de Notícias Falsas Utilizando Aprendizado de Máquina

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 20 de Outubro de 2020.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Eric Aislan Antonelo, Dr.
Orientador e Supervisor
UFSC/CTC/DAS

Prof. Ubirajara Franco Moreno, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Fabio Luis Baldissera, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

Agradeço ao meu pai, Osmar, o qual trabalhou muito para dar o melhor para mim e minha irmã, além de minha mãe, Lourdes, a qual sempre me apoiou e deu todo o suporte emocional aos meus sonhos.

À minha companheira, Rafaela, e sua família por toda ajuda, conselhos, companhia, apoio e carinho.

Aos meus amigos Giovanni, Ígor, Jardel e Luis Felipe, pela parceria e por todas horas de estudo e compartilhamento de conhecimento.

À Universidade Federal de Santa Catarina (UFSC) pelo seu comprometimento com a construção de uma sociedade justa e democrática e também pela sua política de cotas, oferecendo uma formação de ensino de qualidade para alunos de diferentes realidades sociais.

Ao Departamento de Automação e Sistemas (DAS-UFSC) e seus professores, especialmente ao Prof. Hector, por todas conversas e ensinamentos.

Ao meu orientador, Prof. Eric Aislan Antonelo, pela paciência e pelos conselhos ao longo deste trabalho.

Ao Laboratório de Engenharia de Processos de Conversão e Tecnologia de Energia (LEPTEN/Boiling) e à empresa Atlantic Energias Renováveis S.A., pelo projeto de P&D e pela visita técnica ao parque eólico. Em especial gostaria de agradecer ao João, que conheci durante esse período, por todo incentivo e amizade.

Aos membros da Céu Azul Aeronaves pelos bons momentos durante a competição de AeroDesign de 2018.

Ao Laboratório de Sistemas Hidráulicos e Pneumáticos (LASHIP), em especial ao Diego.

Aos demais familiares, amigos, colegas e todos aqueles que contribuíram para esse trabalho.

Muito obrigado!

RESUMO

O PFC foi realizado no DAS/UFSC. Atualmente, devido à popularização da internet e a facilidade de acesso às informações, as notícias falsas propagam-se rapidamente, tornando-se uma preocupação para a sociedade. O objetivo deste PFC é obter um modelo de aprendizado de máquina para detecção de notícias falsas. Este estudo contempla o desenvolvimento de dois modelos de classificação, um utilizando regressão logística e outro redes neurais recorrentes do tipo LSTM. Ambos foram treinados e avaliados utilizando uma base de dados de notícias falsas e verdadeiras contendo 7200 amostras em Língua Portuguesa. O pré-processamento de dados foi realizado levando em consideração o modelo de aprendizado de máquina em questão. Para melhorar o desempenho, diversos experimentos foram elaborados com variações nos valores dos hiperparâmetros do modelo. Os resultados foram comparados através das métricas acurácia, precisão, *recall* e *F1-score*. O algoritmo regressão logística mostrou-se melhor que rede neural do tipo LSTM, obtendo um valor de *F1-score* de $92,8\%$. Por fim, foi desenvolvido uma interface web, a qual é possível realizar a checagem de uma notícia através da utilização do modelo treinado no *back-end*.

Palavras-chave: Regressão logística. Detecção de notícia falsa. Processamento de linguagem natural. LSTM.

ABSTRACT

This PFC was developed at DAS/UFSC. Currently, due to the popularization of the internet and the ease of access to information, fake news is spreading rapidly, becoming a concern for society. The purpose of this work is to obtain a machine learning model for fake news detection. This study contemplates the development of two classification models, one using logistic regression and the other recurrent neural networks of type LSTM. Both were trained and evaluated using data of fake and true news containing 7200 samples in Portuguese. Data pre-processing was performed taking into account the machine learning model in question. To improve performance, several experiments were carried out with variations in the values of the model's hyperparameters. Results were compared using the metrics accuracy, precision, recall and F1-score. Logistic regression algorithm proved to be better than LSTM neural network, obtaining a value of $92,8\%$ in F1-score. Finally, a web interface was developed using the trained model on the back-end where it is possible to check a news.

Keywords: Logistic regression. Fake news detection. Natural language Processing. LSTM.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Atualmente, devido ao desenvolvimento da tecnologia e internet, as redes sociais e os aplicativos de mensagens instantâneas permitem que conteúdos enganosos alcancem um maior número de pessoas.

Em razão da sua natureza apelativa, as *fake news* são disseminadas rapidamente, influenciando a percepção das pessoas em diversos assuntos, desde temas de supostos estudos científicos que confirmam meias-verdades a declarações de políticos e celebridades. Desse modo, as *fake news* não somente influenciaram nas eleições políticas ao redor do mundo, como também causaram problemas de saúde pública (p. ex: através de conspiração de campanhas de vacinação) e tragédias humanas (p. ex: linchamentos públicos e pessoas fazendo justiça com as próprias mãos) (SILVA *et al.*, 2020).

Segundo (MESQUITA *et al.*, 2020), até o término de fevereiro de 2020, durante o enfrentamento da pandemia da COVID-19, mais de 2100 iranianos foram envenenados por ingestão oral de metanol. Os pacientes relataram que mensagens em mídias sociais sugeriram que eles bebessem álcool para prever a infecção do SARS-CoV-2. Quase 900 pacientes intoxicados foram levados para a Unidade de Tratamento Intensivo (UTI), sendo que 296 morreram.

(SILVA *et al.*, 2020) diz que o ser humano tem uma grande dificuldade não somente para checagem de *fake news*, mas também para conteúdos enganosos em geral. Segundo (RUBIN; CONROY, 2011), os indivíduos possuem entre 50 e 63% de acurácia para detectar fraudes. Por esse motivo, é fundamental que existam ferramentas disponíveis com o objetivo de auxiliarem a população.

Atualmente, no Brasil, existem algumas entidades com o objetivo de analisar conteúdos divulgados na internet, como por exemplo: Agência Lupa, Aos Fatos e Estadão Verifica. Estas instituições recebem solicitações de checagem de notícias dos usuários da internet e as classificam seguindo suas metodologias. Acrescenta-se também que essas agências são membros do International Fact-checking Network (IFCN), cumprindo os princípios estabelecidos pela rede de checadores e passando por auditorias anualmente.

Para checagem manual de fatos e notícias falsas, o Aos Fatos (FATOS, s.d.) elenca 6 diretrizes básicas para auxiliar quem questiona o que é distribuído nas redes:

1. Buscar fontes confiáveis;
2. Questionar;
3. Certificar de que no texto há referências;
4. Observar a linguagem;

5. Ver se o texto está assinado ou se é possível contatar o veículo de divulgação;
6. Lembrar que as redes sociais são um começo, mas não a melhor fonte.

Considerando a crescente circulação das *fake news* e a dificuldade da sociedade na percepção da veracidade de uma informação, esforços envolvendo a checagem automática de notícias são muito relevantes.

De acordo com (SILVA *et al.*, 2020), a detecção automática de inverdades é atrativa por duas razões:

1. Estes sistemas são mais objetivos que os julgamentos humanos, os quais são propensos a vieses;
2. As pessoas estão mais propensas a atrasos e erros pela questão da sobrecarga de sua capacidade de julgar múltiplas interpretações de vídeos e áudios.

A partir desse conjunto de constatações, esse trabalho trata de criar um modelo de aprendizado de máquina para detecção automática de *fake news*. Para isso, dois modelos foram testados: regressão logística e rede neural recorrente. Ambos os modelos foram treinados, validados e testados com dados do *Fake.Br corpus*, o qual é um corpus de notícias falsas em Língua Portuguesa. Após a seleção do melhor modelo, foi construída uma interface web com o mesmo rodando em *back-end*, que um usuário pode checar em instantes a veracidade de uma notícia. A partir disso, define-se os objetivos desse projeto.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um modelo de aprendizado de máquina para classificar notícias como falsas ou verdadeiras.

1.1.2 Objetivos Específicos

- Buscar e analisar a base de dados de *fake news*;
- Aplicar e ajustar o modelo de regressão logística;
- Aplicar e ajustar o modelo de rede neural recorrente;
- Avaliar e comparar os resultados dos modelos;
- Implementar uma interface web responsável por permitir a checagem de uma notícia.

1.2 ESTRUTURA DO DOCUMENTO

Este documento é estruturado da seguinte forma:

No Capítulo 2 é apresentado conceitos básicos utilizados no desenvolvimento deste trabalho. São expostos conceitos de notícias falsas, pré-processamento de dados e aprendizado de máquina.

No Capítulo 3 é esclarecida a solução proposta e a metodologia.

No Capítulo 4 é apresentada a implementação da solução proposta. Aqui são abordadas as ferramentas utilizadas, trechos de código referentes ao pré-processamento e a modelagem, bem como a implementação da interface web.

No Capítulo 5 os resultados são expostos através de gráficos e tabelas. Além disso, também apresenta-se o resultado final da interface web.

Por fim, no Capítulo 6 são apontadas as conclusões e perspectivas sobre o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 NOTÍCIA FALSA

Atualmente, a palavra *fake news* vem sendo utilizada de forma ampla. Segundo (JR. *et al.*, 2018), esta palavra se tornou comum especialmente após as eleições presidenciais de 2016 nos EUA, um exercício democrático marcado por cargas de desinformações e notícias falsas. Consequentemente, há um maior poder de persuasão e alcance, como afirma (ZHOU *et al.*, 2020) sobre as teorias fundamentais de ciências sociais e psicológias que indicam que quanto mais as *fake news* são disseminadas, maior é a possibilidade dos usuários de redes sociais acreditarem e disseminá-las pelo motivo da exposição repetitiva e/ou por pressão de grupo.

(JR. *et al.*, 2018) fez uma revisão de estudos acadêmicos que constavam o termo *fake news* com o objetivo de identificar as diferentes formas que a palavra era utilizada e definida. Como resultado dessa revisão foram identificados seis maneiras que os estudos apresentavam a palavra: sátira, paródia, fabricação, manipulação, propaganda, e publicidade.

A Tabela 1 apresenta essas tipologias com base em duas dimensões: nível de facticidade e a intenção imediata do autor de enganar. De acordo com o mesmo autor, as definições atuais parecem focar no terceiro quadrante, o qual centra na fabricação que possui baixo nível de factilidade e alta intenção imediata de enganar.

Tabela 1 – Tipologia de definições de *fake news*.

Nível de factibilidade	**Intenção imediata do autor em enganar**	
	Alta	**Baixa**
Alta	Publicidade Propaganda	Sátira
Baixa	Manipulação Fabricação	Paródia

Fonte – Adaptado de Jr. *et al.* (2018).

2.2 PRÉ-PROCESSAMENTO DE DADOS

2.2.1 Introdução

O pré-processamento consiste em transformar a base de dados original em um formato mais conveniente para o processamento computacional. Ele consiste em combinações de diferentes técnicas, que variam dependendo do domínio do problema.

Segundo (MICELI *et al.*, 2018), textos são os dados sequenciais mais difundidos. Eles podem ser entendidos como sequências de palavras ou de caracteres, porém é

mais comum ser trabalhado a primeira opção.

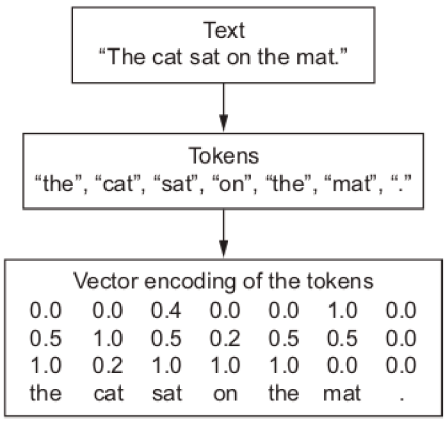
De acordo com (MICELI *et al.*, 2018), as diferentes unidades que um texto pode ser separado (palavra, caractere, n-grama) são chamadas de *tokens*, sendo que a separação do texto em *tokens* é denominada *tokenização*.

Há algumas formas de vetorizar dados textuais para poderem ser processados por modelos de aprendizado de máquina:

- Segmentar texto em palavras e transformar cada uma em vetor;
- Segmentar texto em caracteres e transformar cada um em vetor;
- Extrair n-gramas de palavras ou caracteres e transformar cada n-grama em vetor, sendo que ela representa um grupo de palavras ou caracteres que se sobrepõem.

Todo processo de vetorização de texto consiste em aplicar algum tipo de *tokenização* seguido pela associação de vetores numéricos para os *tokens* gerados, conforme exemplificado na Figura 1.

Figura 1 – Texto, *tokens* e vetores.



Fonte – Adaptado de (MICELI *et al.*, 2018)

Uma etapa comum em tarefas de processamento textual é a conversão de dados textuais para letra minúscula. Ela pode ajudar na redução da esparsidade dos dados em que as mesmas palavras escritas de maneiras diferentes são mapeadas para um mesmo padrão quando convertidas para letra minúscula. Por exemplo, as palavras *Brasil, brasil e brasiL* são convertidas para uma única, *brasil*. Esta conversão também auxilia no tempo de processamento, visto que há uma redução da quantidade de dados.

Alguns *tokens* não trazem muita importância da forma que se apresentam e podem ser substituídos por outros, de modo a apresentar somente seu significado.

Por exemplo, e-mails podem ser substituidos pelo *token 'EMAIL'*, números podem ser substituídos pelo *token '0'* e URLs podem ser substituídas pelo *token 'URL'*.

2.2.2 Vetor de palavra

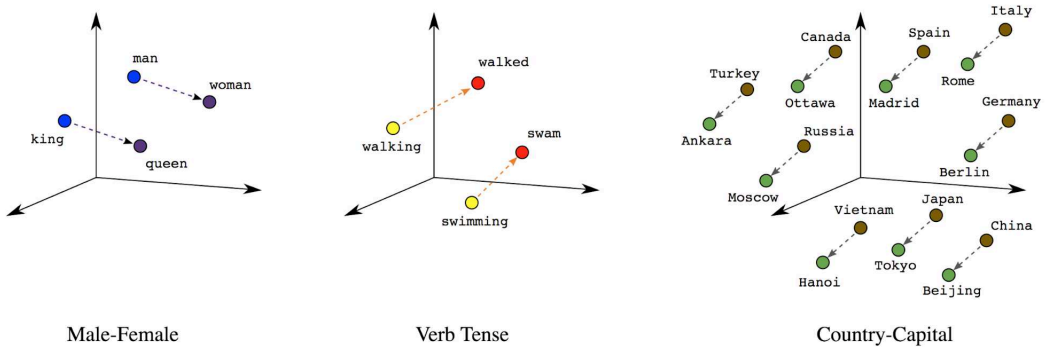
Um vetor de palavras (*word embedding*) é um vetor denso, contínuo e de tamanho fixo que representa uma palavra.

Segundo (MICELI *et al.*, 2018), o *word embedding* faz um mapeamento da linguagem humana para um espaço geométrico. Palavras com significados diferentes são localizadas em pontos distantes umas das outras, enquanto que as sinônimas se encontram próximas.

O *word embedding* pode codificar o conceito de gênero, cidade e capital, dentre outros. Ainda, é possível realizar operações com os vetores, por exemplo, ao fazer *Rei - Homem + Mulher* obtém-se um vetor próximo a *Rainha*.

É possível visualizar os vetores utilizando alguma técnica de redução de dimensionalidade, por exemplo, a t-SNE, conforme pode ser visto na Figura 2.

Figura 2 – Visualização de vetores de palavras.



Fonte – (DEVELOPERS, s.d.)

(MICELI *et al.*, 2018) afirma que há dois modos de obter um *word embedding*:

- Treinar o *word embedding* juntamente com a tarefa que está sendo executada. Nesta configuração, os vetores de palavras são inicializados de modo randômico e após, são aprendidos de maneira análoga aos pesos de uma rede neural;
- Carregar no modelo *word embeddings* que foram já treinados para outras finalidades de aprendizado de máquina.

2.2.3 TF-IDF

TF-IDF (Term Frequency - Inverse Document Frequency) é uma técnica utilizada para transformar palavras em números. Ela é usada para indicar a importância de uma palavra em documento em relação a uma coleção de documentos. Seu valor pode ser calculado através da multiplicação dos termos TF e IDF.

Segundo (CASALI, 2018), o cálculo pode ser feito da seguinte forma:

- TF (Term Frequency): Procura valorizar *tokens* que aparecem em abundância em um documento. Ele pode ser calculado fazendo uma contagem ou de maneira relativa:

$$TF = \frac{f_{ij}}{\sum_j f_i} \tag{1}$$

  onde f_{ij} é o número de vezes que o termo T_j aparece no documento D_i e $\sum_j f_i$ é a quantidade de termos no mesmo.

- IDF (Inverse Document Frequency): Utilizado para valorizar os termos que aparecem raramente no conjuntos de documentos e, portanto, são importantes:

$$IDF = log\frac{N}{DF} \tag{2}$$

  onde N é o número total de documentos e DF (Document Frequency) é a quantidade de documentos em que um termo T_j aparece.

Então, o valor TF-IDF de uma palavra aumenta proporcionalmente com a frequência dela no documento, no entanto, quanto maior a ocorrência dessa palavra em todos os documentos, menor é o valor TF-IDF.

2.3 APRENDIZADO DE MÁQUINA

De acordo com (GÉRON, 2019), aprendizado de máquina é a ciência (e arte) de programar computadores para que eles possam aprender com os dados.

(MITCHELL, 1997) fornece uma definição mais orientada à engenharia: "Diz-se que um programa de computador aprende pela experiência E, com respeito a algum tipo de tarefa T e performance P, se sua performance P nas tarefas em T, na forma medida por P, melhoram com a experiência E."

(MOHAMMED *et al.*, 2017) diz que sistemas de aprendizado de máquina podem ser separados de acordo com 4 técnicas de aprendizado:

- Aprendizado supervisionado: o conjunto de treinamento possui rótulo (resposta desejada). Aqui, pode-se ainda dividir os algoritmos em dois grupos:

  - Classificação: quando os rótulos são definidos por uma quantidade limitada de valores discretos. Por exemplo: detectar se uma notícia é falsa ou verídica.
  - Regressão: quando os rótulos possuem valores contínuos.

- Aprendizado não supervisionado: o conjunto de treinamento não possui rótulo, ou seja, o sistema tenta aprender sem um professor;
- Aprendizado semi-supervisionado: há uma combinação de dados rotulados e não rotulados;
- Aprendizado por reforço: um agente observa o ambiente, seleciona e realiza ações, obtendo em troca recompensas ou penalidades. Dessa forma, o agente aprende com ele mesmo qual a melhor estratégia, chamada de política, para conseguir a melhor recompensa ao longo do tempo. A política define qual ação o agente deveria escolher quando ele está em determinada situação.

Segundo (GÉRON, 2019), outro critério utilizado para classificar sistemas de aprendizado de máquina é se o aprendizado ocorre em lote ou de forma incremental:

- Aprendizado em lote: o sistema é incapaz de aprender incrementalmente. Ele deve ser treinado com todos os dados disponíveis. Inicialmente, o sistema é treinado, avaliado e lançado para a produção, na qual não executa novos aprendizados. Entretanto, se for preciso o aprendizado do sistema com novos dados, será necessário um novo treinamento com todo o conjunto de dados.
- Aprendizado online: o sistema é treinado incrementalmente pelas entradas de dados sequencialmente, seja individualmente ou em pequenos grupos, chamados mini-lotes. Acrescenta-se também que o aprendizado online é bom para sistemas que recebem dados em fluxo contínuo, como o mercado de ações e que necessitam se adaptar rapidamente e automaticamente.

2.3.1 Validação Cruzada

Uma prática comum no desenvolvimento de um projeto de aprendizado de máquina é a separação dos dados em três conjuntos: treino, validação e teste.

Usa-se o conjunto de treino para treinar o modelo e o de validação para avaliar o mesmo. Repete-se este ciclo até que o modelo alcance resultados satisfatórios. Após, realiza-se uma avaliação final do modelo no conjunto de teste.

Contudo, quando não temos um conjunto de dados suficientemente grande, podemos acabar escolhendo amostras não representativas ao realizar a separação dos conjuntos.

Para contornar o problema do conjunto de dados pequenos e aumentar a confiabilidade dos resultados, pode-se utilizar a validação cruzada, a qual o modelo é treinado e avaliado diversas vezes em diferentes porções dos dados, e o conjunto de teste ainda deve ser mantido para uma avaliação final.

Na abordagem básica da validação cruzada, chamada *k-fold* (Figura 3), o conjunto de treinamento é dividido em k subconjuntos mutualmente exclusivos e o seguinte procedimento é aplicado k vezes:

- Um modelo é treinado utilizando os $k-1$ subconjuntos (verde claro);
- O modelo é avaliado no subconjunto restante (azul claro).

A performance da validação cruzada é medida calculando-se a média dos valores resultantes das k avaliações.

Figura 3 – Validação cruzada *k-fold* com k=5

Fonte – Adaptado de (SCIKIT-LEARN, s.d.)

2.3.2 Regressão Logística

Regressão Logística é um algoritmo que lida com problemas de classificação e pode ser utilizado para prever a probabilidade de uma amostra pertencer a determinada classe, por exemplo: qual a probabilidade de uma notícia ser falsa?

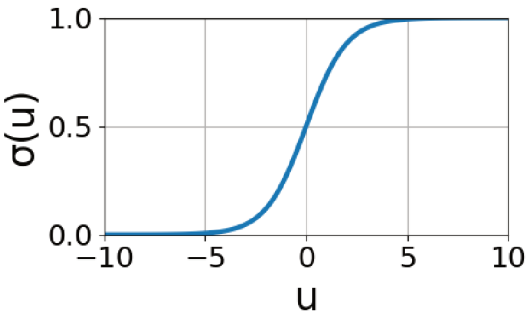
Este modelo é apresentado na Equação (3), em que $\mathbf{x} = (x_0, x_1, ..., x_n)$ é o vetor de variáveis, $\theta = (\theta_0, \theta_1, ..., \theta_n)$ é o vetor de parâmetros e σ, a função logística.

$$h_\theta(\mathbf{x}) = \sigma(\mathbf{x}^T\theta) \tag{3}$$

A função logística σ, conforme Equação (4) e Figura 4, tem por objetivo converter o valor produzido por $\mathbf{x}^T\theta$ em uma probabilidade.

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{4}$$

Figura 4 – Função logística.



Fonte – Autor

Logo, $h_\theta(\mathbf{x})$ pode ser interpretado como a probabilidade que a saída do modelo seja 1, dado o vetor de variáveis $\mathbf{x}$, parametrizado por θ (Equação (5)).

$$h_\theta(\mathbf{x}) = P(y = 1|\mathbf{x}; \theta) \tag{5}$$

Como a função logística varia entre 0 e 1, a predição pode ser feita definindo-se um limiar de $0,5$, conforme Equação (6).

$$\hat{y} = \begin{cases} 0, \ se\ h_\theta(\mathbf{x}) < 0,5 \\ 1, \ se\ h_\theta(\mathbf{x}) \geq 0,5 \end{cases} \tag{6}$$

Para obter o modelo treinado, deve-se minimizar uma função custo por meio de algum algoritmo de otimização, resultando, portanto, no vetor de parâmetros θ desejado.

A função custo entropia cruzada binária (*binary cross-entropy*), apresentada na Equação (7), mede como o modelo está performando durante o processo de treinamento e é amplamente utilizada para problemas de classificação. Na equação, $h_\theta(\mathbf{x})$ é a predição do modelo, y é a resposta desejada e m o número de amostras.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(\mathbf{x})^{(i)}) + (1 - y^{(i)})log(1 - h_\theta(\mathbf{x})^{(i)})] \tag{7}$$

Para minimizar a função custo, usa-se o algoritmo gradiente descendente, que de forma iterativa atualiza cada parâmetro através do uso do próprio valor do parâme-

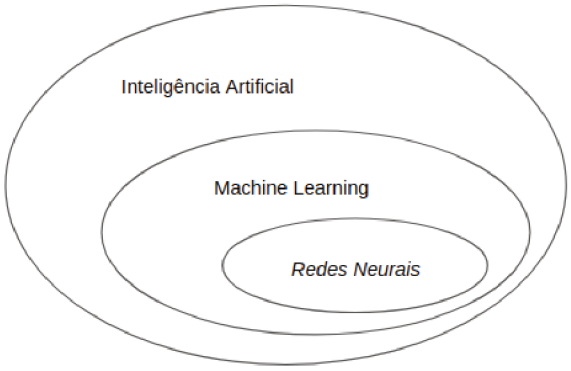
tro, da taxa de aprendizado α e da derivada da função custo. Para isso, repete-se a Equação (8), atualizando os valores de θ_j de forma simultânea a cada iteração.

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{8}$$

2.3.3 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) são modelos de aprendizado de máquina, como demonstra a Figura 5, e são inspiradas nas redes neurais biológicas encontradas no cérebro humano.

Figura 5 – Diagrama de Venn: relações entre inteligência artificial, aprendizado de máquina e redes neurais.



Fonte – Autor

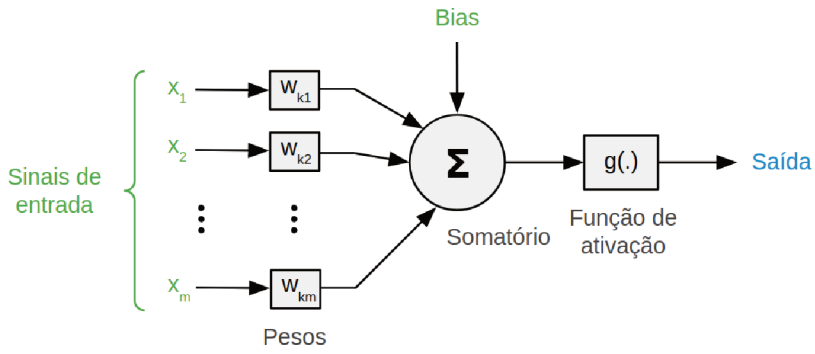
As RNAs existem há bastante tempo, segundo (GÉRON, 2019), elas foram introduzidas pela primeira vez, em 1943, pelo neurofisiologista, Warren McCulloch, e o matemático, Walter Pitts. Eles propuseram um modelo de neurônio biológico, que posteriormente foi chamado de neurônio artificial. Esse modelo possui uma ou mais entradas binárias e uma saída binária. Além disso, mostraram que, mesmo com esse modelo simplificado, era possível construir redes de neurônios artificiais que computassem qualquer proposição lógica. Posteriormente, esse modelo foi aperfeiçoado, conforme será apresentado adiante.

As RNAs são compostas por neurônios organizados em três tipos de camadas: camada de entrada, de saída e escondida. Na camada de entrada, na qual localizam-se os neurônios de entrada, tem a função de inserir os dados na rede neural. Nas camadas escondidas estão presentes os neurônios escondidos e elas têm a funçáo de ajudar a processar a informação. Finalmente, na camada de saída, localizam-se os neurônios de saída e possui a finalidade de retirar a informação da rede neural.

Uma rede neural com várias camadas escondidas é chamada de rede neural profunda. Existem várias definições de quantas camadas escondidas são necessárias para uma rede ser considerada profunda.

Um neurônio é uma unidade que possui, geralmente, mais de uma entrada e uma saída. O neurônio computa uma soma ponderada dos sinais de entrada $\mathbf{x} = (x_1, x_2, ..., x_n)$ com os pesos $\mathbf{w} = (w_1, w_2, ..., w_n)$, produzindo um potencial de ativação. Ao pontencial de ativação é aplicado uma função de ativação, produzindo, então, o resultado. Com o intuito de aumentar o grau de liberdade e consequentemente, a capacidade de representação da rede, o elemento Bias também é incluído ao somatório da função de ativação.

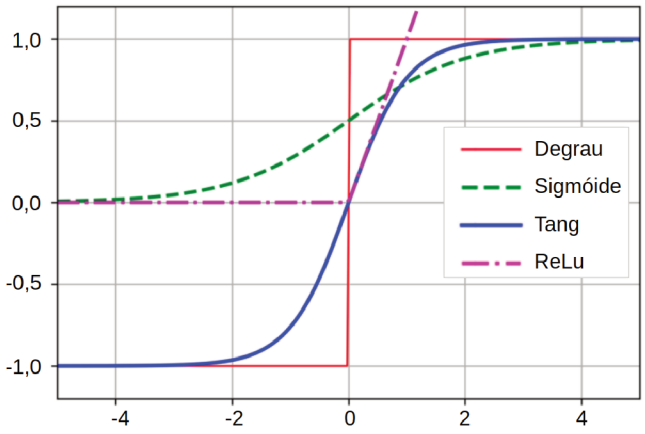
Figura 6 – Modelo matemático de um neurônio artificial.



Fonte – Autor

As funções de ativação são fundamentais para dar capacidade representativa às RNAs, introduzindo um componente de não linearidade. Alguns tipos de funções de ativação são mais populares e podem ser vistas na Figura 7, representando as funções degrau, sigmoide, tangente hiperbólica e ReLu (*Rectifield Linear Unit*).

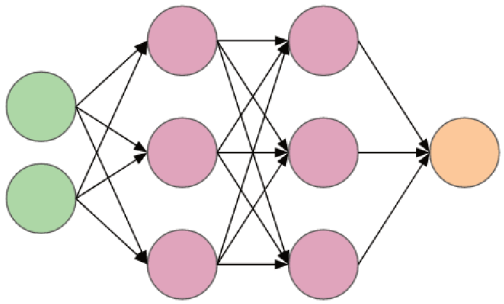
Figura 7 – Exemplos de funções de ativação.



Fonte – Adaptado de (GÉRON, 2019)

Nas RNAs do tipo *feedforward*, o fluxo do sinal sempre parte da camada de entrada em direção a camada de saída, conforme apresentado na Figura 8, a qual apresenta uma camada de entrada com dois neurônios, duas camadas escondidas com 3 neurônios cada e por fim, uma camada de saída com um neurônio.

Figura 8 – Exemplo de rede neural *feedforward*, com as setas indicando o sentido da informação.

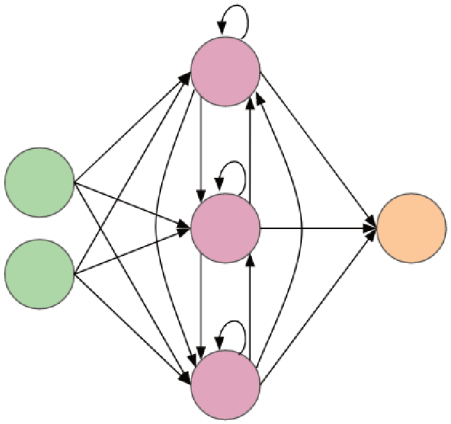


Fonte – Autor

Os pesos de uma RNA são alterados através de um algoritmo durante a fase de treinamento com o objetivo de atingir a generalização desejada. O algoritmo de treinamento mais utilizado é o *backpropagation*. Seu funcionamento pode ser entendido em duas etapas. Na primeira, o algoritmo faz a propagação da informação de entrada até a camada de saída, armazenando os valores dos pesos calculados. Na segunda, ocorre o cálculo do erro através de uma função custo e, em seguida, a retropropagação do mesmo que é calculado através da regra da cadeia. Por fim, o algoritmo ajusta os pesos utilizando os gradientes dos erros resultantes.

A Figura 9 mostra um exemplo de rede neural recorrente (RNR). Elas são caracterizadas por haver algum tipo de retroalimentação, a qual o sinal de saída de um neurônio volta para a entrada do mesmo ou de outros neurônios. Devido à retroalimentação dos neurônios, pode-se dizer que as RNRs possuem células de memórias e, por conseguinte, podem ser especialmente úteis para o processamento de dados sequenciais, como áudios, séries temporais ou textos.

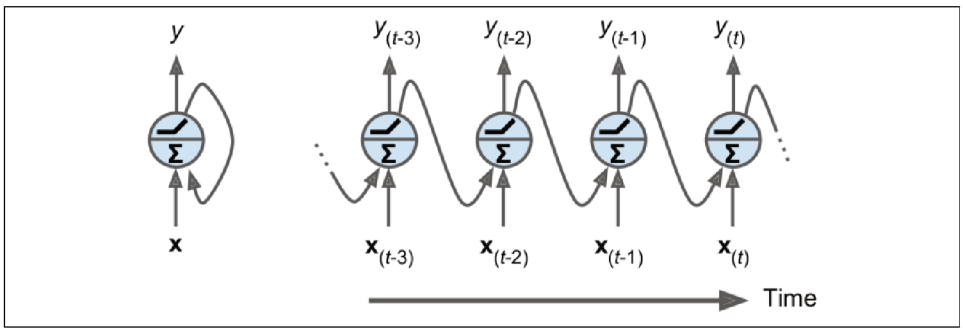
Figura 9 – Exemplo de uma rede neural recorrente, caracterizada por haver sinais de retroalimentação.



Fonte – Autor

A Figura 10 (esquerda) mostra a estrutura de um neurônio recorrente, na qual em cada *time step* t, ele recebe as entradas $x(t)$ e também sua própria saída do *time step* anterior, $y(t-1)$. Esse neurônio pode ser representado ao longo do eixo do tempo, como mostrado na Figura 10 (direita). Isso é conhecido como desenrolar a rede através do tempo.

Figura 10 – Um neurônio recorrente (esquerda) desenrolado ao longo do tempo (direita)



Fonte – (GÉRON, 2019)

Para treinar uma RNR, usa-se um algoritmo chamado *backpropagation through time* (BPTT). Essa estratégia consiste em desenrolar a RNR através do tempo e então utilizar o *backpropagation*, conforme já explicado.

Segundo (CHOLLET, 2017), embora o neurônio artificial seja teoricamente capaz de reter informações sobre as entradas vistas em muitos *timesteps* anteriores, na prática, essas dependências de longo prazo são difíceis de serem aprendidas devido ao problema da dissipação do gradiente *vanishing gradient problem*, que é um efeito que é similar ao que acontece em redes *feedforwards* com várias camadas profundas: conforme você adiciona camadas na rede, a rede eventualmente se torna difícil de treinar.

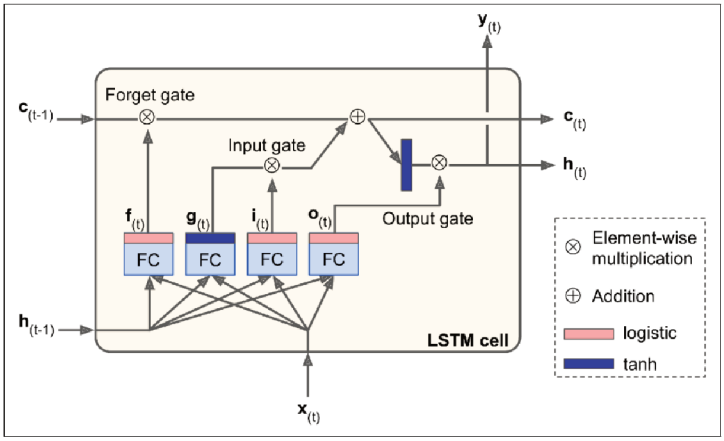
O desenvolvimento das células LSTM (*Long Short-Term Memory*) teve como motivação solucionar as dificuldades de processamento de longas sequências das redes neurais recorrentes.

Segundo (GÉRON, 2019), se olharmos uma célula LSTM como uma caixa preta, podemos usá-la como uma célula básica, exceto que a primeira apresentará uma melhor perfomance; o treinamento convergirá mais rápido e ela será capaz de detectar dependências de longo termo.

A Figura 11 apresenta a estrutura de uma célula LSTM. Esta estrutura possui um estado que pode ser separado em duas partes:

- $h(t)$: estado oculto
- $c(t)$: célula de memória

Figura 11 – Célula LSTM



Fonte – (GÉRON, 2019)

Quando $c(t-1)$ atravessa a rede, inicialmente passa pelo *forget gate*, esquecendo algumas memórias e a seguir, passa pela operação de adição, recebendo novas

memórias (selecionadas pelo *input gate*), então tem-se $c(t)$. Portanto, além do $c(t)$ ser enviado para fora da célula, também serve como entrada para uma função tangente hiperbólica e após, passa pelo *output gate*, resultando, assim, em $h(t)$ (que é igual a $y(t)$).

O vetor de entrada $x(t)$ e o estado oculto anterior $h(t-1)$ são alimentados em 4 camadas totalmente conectadas:

- A camada que possui $g(t)$ como saída é a principal. Essa tem o papel de analisar as entradas atuais $x(t)$ e o estado oculto anterior $h(t-1)$. As partes mais importantes dessa camada são armazenadas em $c(t)$ e as demais partes são esquecidas;
- As outras três camadas são portões controladores (*gates)*. Cada portão possui uma função logística em sua saída, que é utilizada na operação de multiplicação.
  - *Forget gate*: É controlado por $f(t)$ e controla qual parte de $c(t)$ deve ser apagada;
  - *Input gate*: É controlado por $i(t)$ e controla qual parte de $g(t)$ deve ser adicionada a $c(t)$;
  - *Output gate*: É controlado por $o(t)$ e controla quais partes de $c(t)$ devem ser lidas e apresentadas na saída desse *timestep*, tanto para $h(t)$ quanto para $y(t)$.

De modo a evitar *overfitting*, que é quando um modelo se ajusta muito bem aos dados de treinamentos, mas se mostra ineficaz para prever novos resultados, pode-se utilizar técnicas de regularização.

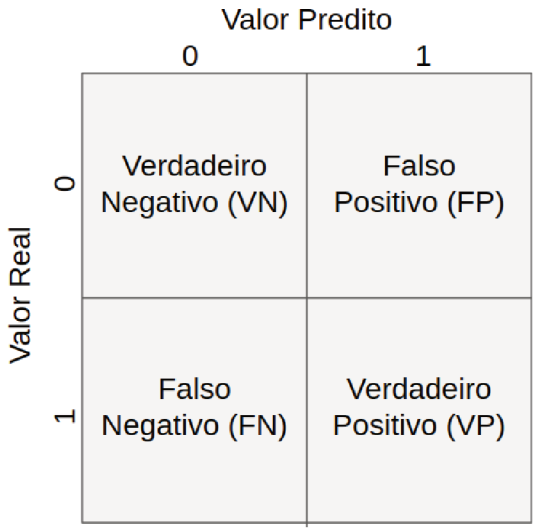
Uma técnica utilizada para a regularização de redes neurais é o *dropout*. Ao utilizar o *dropout*, em cada passo de treinamento, cada neurônio tem uma probabilidade de ser temporariamente desativado e ignorado durante esse passo, porém o mesmo pode estar ativo durante o próximo passo de treinamento. Após o treinamento, os neurônios não são mais desativados.

Em redes neurais recorrentes, além do *dropout*, pode-se utilizar o *recurrent dropout*, que desativa conexões entre as unidades recorrentes.

2.4 MÉTRICAS DE AVALIAÇÃO

As métricas de avaliação são utilizadas para avaliar o desempenho dos modelos de aprendizado de máquina. As mais tradicionais são calculadas através da matriz de confusão, mostrada na Figura 12.

Figura 12 – Matriz de confusão e seus conceitos.



Fonte – Autor

A Acurácia corresponde a taxa de acertos e é calculada através da Equação (9).

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{9}$$

A Precisão é calculada através da Equação (10) e avalia proporção de positivos classificados corretamente, ou seja, dos classificados como positivo quanto são positivos realmente. A precisão será máxima se não existirem falsos positivos.

$$Precisão = \frac{VP}{VP + FP} \tag{10}$$

Recall (sensibilidade) é calculado através da Equação (11) e representa o quão frequente as predições são positivas quando o valor real é positivo. O *Recall* será máximo se não existirem falsos negativos.

$$Recall = \frac{VP}{VP + FN} \tag{11}$$

O *F1-score*, calculado através de Equação (11), é uma média harmônica entre a Precisão e o *Recall*, trazendo uma métrica que contempla estas outras duas. O *F1-score* acusa valores pequenos em alguma das métricas anteriores, tendo seu valor decaído caso a Precisão ou o *Recall* forem baixos.

$$F1 - score = 2.\frac{Precisão.Recall}{Precisão + Recall} \tag{12}$$

3 SOLUÇÃO PROPOSTA

O problema abordado nesse trabalho se trata do desenvolvimento de um modelo de aprendizado de máquina para classificação automática de notícias como falsas ou verídicas.

Um dos primeiros passos é ter uma visão geral de qual é o problema e a obter uma base de dados de *fake news*. Com os dados a disposição, realiza-se uma exploração de forma a entendê-los e obter insights.

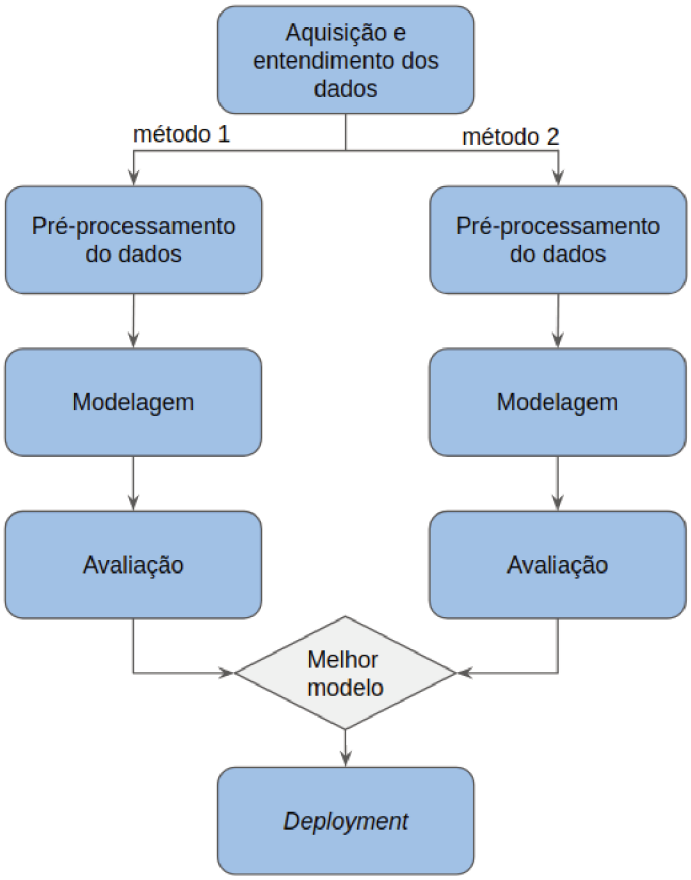
Foram testados dois métodos, o primeiro utilizando o algoritmo de regressão logística e o segundo, redes neurais recorrentes do tipo LSTM. Cada algoritmo exige que os dados tenham um tipo de representação espefícia na entrada. Para a regressão logística, os dados de entrada terão uma representação estatística e para a rede neural do tipo LSTM, os dados terão uma representação sequencial.

Para avaliação dos modelos, serão utilizadas as métricas Acurácia, Precisão, *Recall* e *F1-score*. Será considerada principalmente a métrica *F1-score*, que é uma maneira de visualizar as métricas Precisão e *Recall* juntas.

Posteriormente a escolha do melhor modelo, será desenvolvida uma interface web, que um usuário poderá inserir uma notícia a ser verificada e o melhor modelo, rodando no *back-end* da aplicação, fará a checagem da notícia. Assim, a interface web apresentará ao usuário uma indicação de notícia falsa ou verdadeira.

3.1 VISÃO GERAL DA METODOLOGIA

Figura 13 – Metodologia proposta



Fonte – Autor

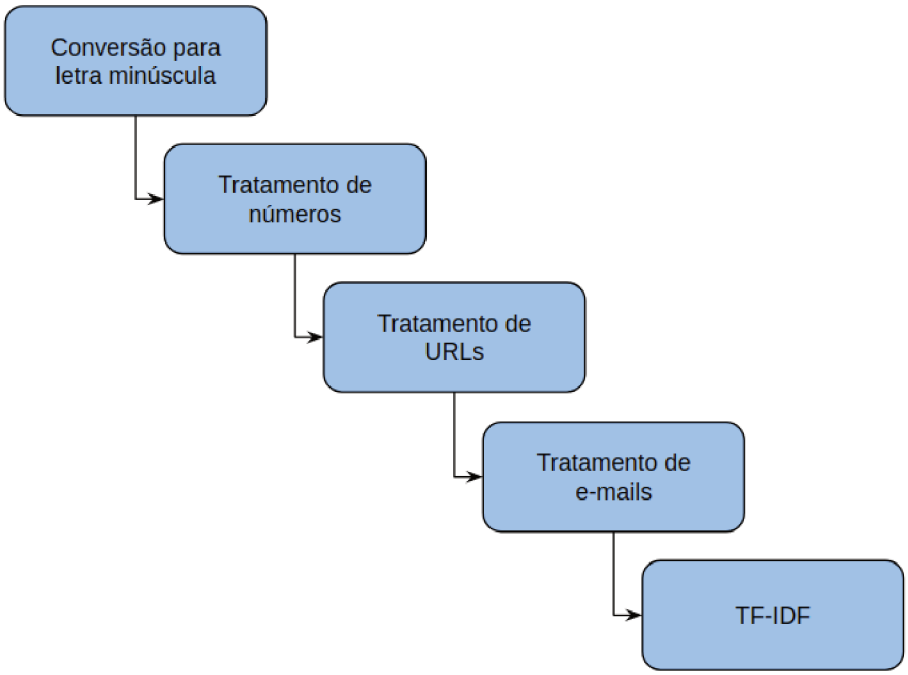
3.2 MÉTODO 1 - REGRESSÃO LOGÍSTICA

O método 1 tem por objetivo a obtenção de um modelo utilizando o algoritmo regressão logística.

3.2.1 Pré-processamento dos dados

Para o pré-processamento, serão seguidas as etapas realizadas por (SILVA *et al.*, 2020), em que inicialmente os textos serão convertidos para letra minúscula, números, URLs os e-mails serão tratados. Por último, os textos serão vetorizados através da abordagem TF-IDF.

Figura 14 – Pré-processamento para o método 1



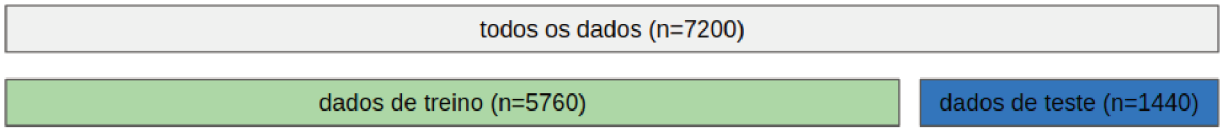
Fonte – Autor

3.2.2 Modelagem

Utilizando-se a regressão logística, o modelo será obtido por meio da testagem de diferentes valores de hiperparâmetros através de uma busca exaustiva com validação cruzada.

Os dados serão separados conforme apresentado na Figura 15, o conjunto de dados de teste será reservado para avaliação final, enquanto que o modelo será desenvolvido utilizando o conjunto de treino.

Figura 15 – Separação dos conjuntos de treino e teste



Fonte – Autor

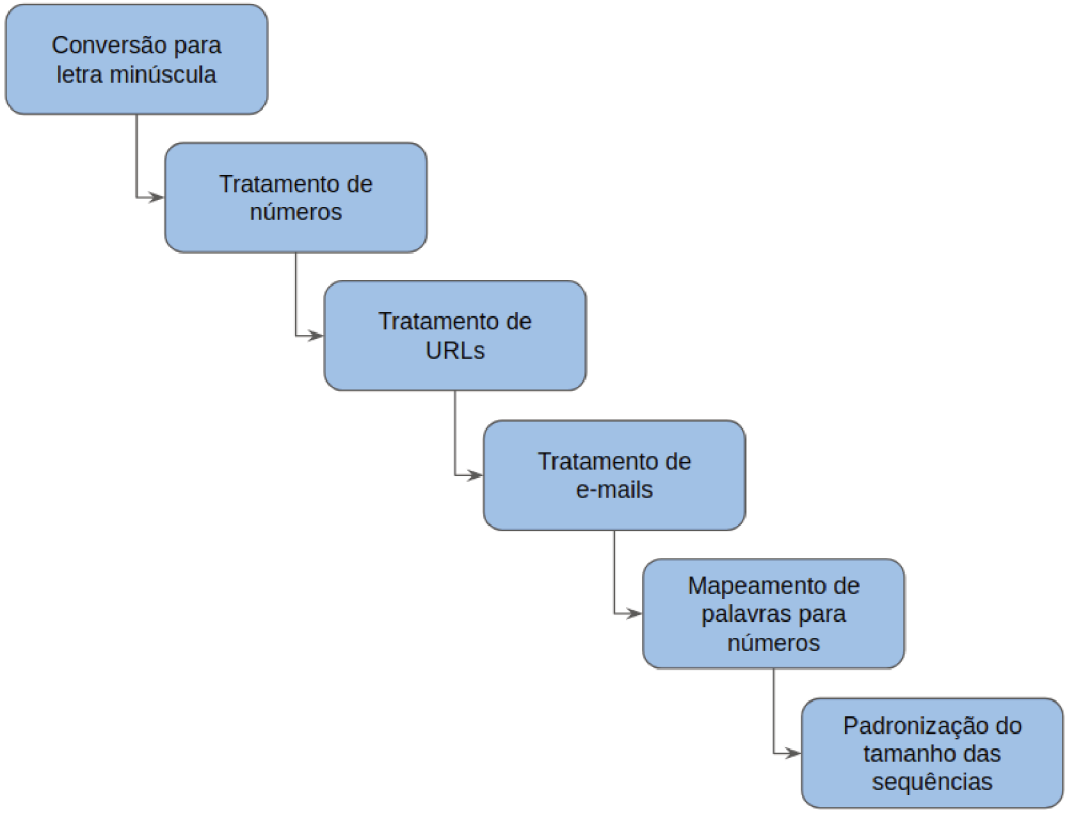
3.3 MÉTODO 2 - REDE NEURAL DO TIPO LSTM

O método 2 tem por objetivo a obtenção de um modelo utilizando a arquitetura de rede neural recorrente do tipo LSTM.

3.3.1 Pré-processamento dos dados

Em todos os três experimentos do método 2 será realizado o pré-processamento dos dados, como demonstrado na Figura 16. Os textos serão convertidos para letra minúscula, números, URLs e e-mails serão tratados, palavras serão mapeadas para inteiros através de um dicionário e, por fim, os textos serão padronizados para um mesmo tamanho.

Figura 16 – Pré-processamento para o método 2



Fonte – Autor

3.3.2 Modelagem

Devido ao alto custo computacional de treinamento, os hiperparâmetros serão ajustados através de três diferentes experimentos, em que cada um diz respeito a uma tentativa de aprimoramento dos resultados do experimento anterior.

Após os três experimentos será obtido o modelo final. Este modelo, portanto, será avaliado no conjunto de teste, o qual possui dados nunca vistos pelo modelo.

3.3.2.1 Experimento 1

O experimento 1 tem por objetivo definir o melhor método de regularização, que será fixado para os próximos experimentos.

Este experimento consiste em testes em uma rede neural com uma camada escondida e com o *word embedding* sendo aprendido durante o treinamento. Serão testadas variações no número de neurônios da camada escondida e no tipo da regularização, conforme Tabela 2.

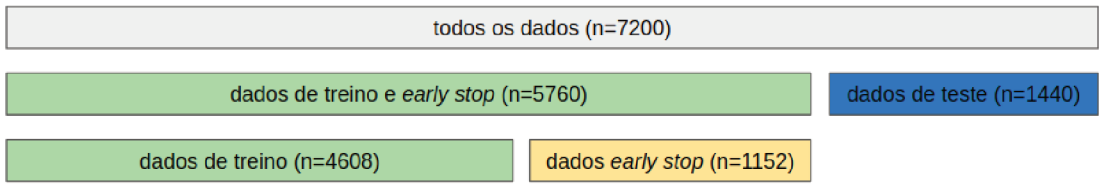
Tabela 2 – Experimento 1

Validação cruzada	Sim
Word embedding	Aprendido durante o treinamento
Num. camadas escondidas	1
Num. neurônios	10, 20, 30, 40, 50, 60, 70
Tipo de regularização	Sem regularização Dropout Early Stop Dropout e early stop

Fonte – Autor

Os conjuntos de dados para validação cruzada foram separados conforme a Figura 17.

Figura 17 – Separação dos dados

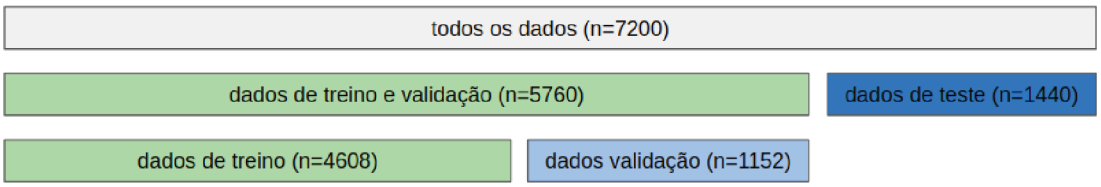


Fonte – Autor

Os *dados de treino* serão utilizado para realizar o treinamento através da validação cruzada e o conjunto *dados early stop* será utilizado para fazer o *early stop* nos treinamentos que utilizam este tipo de regularização. Deste modo, para cada etapa da validação cruzada, o modelo será treinado sempre observando os mesmos dados para realizar o *early stop*.

Após a obtenção do melhor tipo de regularização a partir da validação cruzada, um novo modelo será treinado utilizando os conjuntos de dados apresentados na Figura 18.

Figura 18 – Separação dos dados



Fonte – Autor

3.3.2.2 Experimento 2

O experimento 2 tem como objetivo testar o desempenho da rede neural ao se utilizar um *word embedding* já treinado. Será utilizada uma camada profunda do tipo LSTM e serão testadas variações no número de neurônios da camada LSTM e na dimensão do *word embedding*, conforme Tabela 3.

Tabela 3 – Experimento 2

Validação cruzada	Não
Word embedding	Já treinado
Dimensão do word embedding	50, 100, 300, 600, 1000
Num. camadas escondidas	1
Número de neurônios	10, 20, 30, 40, 50, 60, 70
Regularização	Obtida no experimento 1

Fonte – Autor

A separação dos dados para o experimento 2 é a mesma apresentada na Figura 18.

3.3.2.3 Experimento 3

O terceiro experimento possui como propósito testar o efeito da utilização de duas camadas escondidas LSTM, com variações do número de neurônios de ambas, conforme apresentado na Tabela 4.

Tabela 4 – Experimento 3

Validação cruzada	Não
Word embedding	Obtido nos experimentos 1 e 2
Num. camadas escondidas	2
Num. neurônios na primeira camada	10, 20, 30, 40
Num. neurônios na segunda camada	10, 20, 30, 40
Tipo de regularização	Obtida no experimento 1

Fonte – Autor

A separação dos dados para o experimento 3 é a mesma apresentada na Figura 18.

4 IMPLEMENTAÇÃO

A Tabela 5 apresenta as configurações do notebook utilizado para o desenvolvimento deste trabalho.

Tabela 5 – Configurações da máquina

Sistema Operacional	Ubuntu 18.04.4 LTS
Memoria RAM	8G
Tipo do sistema	64 bits
Processador	Intel® Core™ i7-3632QM CPU @ 2.20GHz x 8

Fonte – Autor

4.1 AQUISIÇÃO E ENTENDIMENTO DOS DADOS

A procura pela base de dados de *fake news* mostra que embora existam algumas opções disponíveis, ao se filtrar pelo idioma Português, percebe-se uma escassez de base de dados. Foi encontrado uma base no idioma Português disponível publicamente e com boas características.

O corpus denominado “Fake.Br Corpus” está no idioma Português e é disponibilizado por (MONTEIRO *et al.*, 2018) abertamente em (FAKE.BR-CORPUS, s.d.). Ele é composto por 7200 notícias: 3600 verdadeiras e 3600 falsas. As notícias estão situadas dentro de um intervalo temporal de 2 anos: 2016/Jan até 2018/Jan. Elas foram coletadas, resumidamente, da seguinte forma: Inicialmente foram obtidas as notícias falsas de modo manual e, para encontrar as notícias verdadeiras correspondentes, foi feito o uso de *web crawler*, medição de similaridade e por fim, verificação manual.

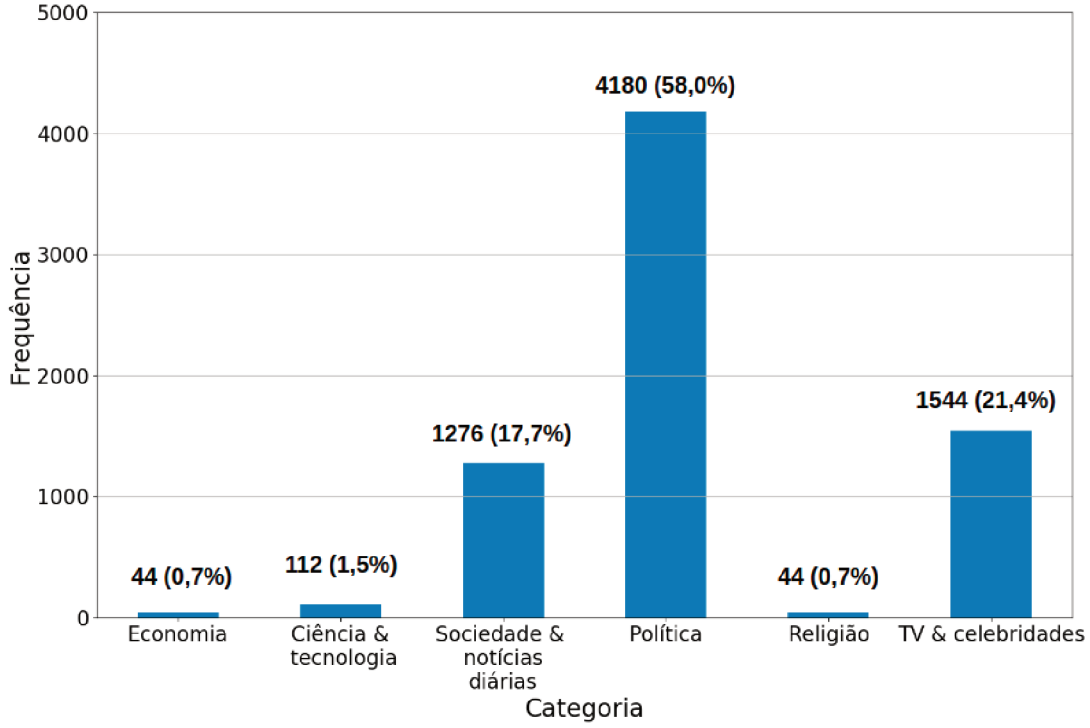
Tabela 6 – Exemplos de notícias falsas e verdadeiras alinhadas.

Falsa	Verdadeira
Michel Temer propõe fim do carnaval por 20 anos, “PEC dos gastos”. Michel Temer afirmou que não deve haver gastos com aparatos supérfluos sem pensar primeiramente na educação do Brasil. A medida pretende calcelar o carnaval de 2018.	Michel Temer não quer o fim do Carnaval por 20 anos. Notícias falsas misturam proximidade dos festejos, crise econômica e medidas impopulares do governo do peemedebista.
Acabou a mordomia ! Ingresso mais barato pra mulher é ilegal. Baladas que davam meia entrada para mulher, ou até mesmo gratuidade, esto na ilegalidade agora. Acabou o preconceito com os homens nas casas de show de todo o Brasil.	Ingresso feminino barato como marketing ‘não inferioriza mulher’, diz a juíza do DF. Afirmação consta em decisão sobre preços diferentes para homens e mulheres em festa no Lago Paranoá. ‘Prática permite que mulher possa optar por participar de tais eventos sociais’, diz texto.

Fonte – Adaptado de (MONTEIRO *et al.*, 2018)

O corpus possui notícias de 6 grandes categorias, as quais foram manualmente etiquetadas com os seguintes rótulos: política, TV e celebridades, sociedade e notícias diárias, ciência e tecnologia, economia e finalmente, religião.

Figura 19 – Frequência das notícias por categoria no Fake.Br corpus



Fonte – Adaptado de (SILVA *et al.*, 2020)

O corpus ainda é disponibilizado em duas versões: a primeira contém os textos completos, enquanto que a segunda, que foi utilizada nesse trabalho, contém os textos truncados. Conforme (SILVA *et al.*, 2020), a utilização do corpus normalizado evita a introdução de viés em modelos de aprendizado de máquina.

4.2 MÉTODO 1

4.2.1 Pré-processamento dos dados

Foram desenvolvidas funções atômicas utilizando expressões regulares de forma a realizar todas as etapas do pré-processamento.

4.2.1.1 Conversão para letra minúscula

Apresenta-se na Figura 20 a função responsável pela conversão dos textos para letra minúscula e na Figura 21, um exemplo do resultado da aplicação da função.

Figura 20 – Função responsável pela conversão para letra minúscula

```
# lower case all data
def lowerize_text(df):
    df = df.copy()
    df['text'] = df['text'].apply(lambda x: x.lower())
    return df
```

Fonte – Autor

Figura 21 – Exemplo de conversão para letras minúsculas

(a) Texto antes da conversão para letra minúscula

```
Avião com 97 passageiros faz pouso de
emergência no aeroporto de Brasília.  Um
avião acabou de fazer um pouso de emergência
no aeroporto JK.  A aeronave precisou
executar um pouso forçado devido a uma pane.
O voo da Avianca partiu de Guarulhos (SP) e
tinha como destino a cidade de Juazeiro do
Norte (CE). De acordo com informações
preliminares da Globo News,  havia fumaça a
bordo e os passageiros entraram em pânico.
Não há vítimas e o pouso ocorreu com
segurança. A aeronave está sendo rebocada
neste exato momento. 97 passageiros estão a
bordo. Nota da Avianca enviada para o Diário
do Brasil às 19:32
```

(b) Texto após conversão para letra minúscula

```
avião com 97 passageiros faz pouso de
emergência no aeroporto de brasília.  um
avião acabou de fazer um pouso de emergência
no aeroporto jk.  a aeronave precisou
executar um pouso forçado devido a uma pane.
o voo da avianca partiu de guarulhos (sp) e
tinha como destino a cidade de juazeiro do
norte (ce). de acordo com informações
preliminares da globo news,  havia fumaça a
bordo e os passageiros entraram em pânico.
não há vítimas e o pouso ocorreu com
segurança. a aeronave está sendo rebocada
neste exato momento. 97 passageiros estão a
bordo. nota da avianca enviada para o diário
do brasil às 19:32
```

Fonte – Autor

4.2.1.2 Tratamento de números

A Figura 22 mostra a função responsável pela conversão dos números para um único *token*. Um exemplo mostrando o resultado da aplicação desta função pode ser visualizado na Figura 23.

Figura 22 – Função responsável pelo tratamento de números

```
# Dummy feature: numerals
def numeral_to_dummy(df):
    df = df.copy()
    df['text'] = df['text'].apply(lambda x: re.sub(r'(\d+[\d\.\,]{1,}\d|\d{1,})', '0', x))
    return df
```

Fonte – Autor

Figura 23 – Exemplo de tratamento de números

(a) Antes do tratamento de números

```
avião com 97 passageiros faz pouso de
emergência no aeroporto de brasília.  um
avião acabou de fazer um pouso de emergência
no aeroporto jk.  a aeronave precisou
executar um pouso forçado devido a uma pane.
o voo da avianca partiu de guarulhos (sp) e
tinha como destino a cidade de juazeiro do
norte (ce). de acordo com informações
preliminares da globo news,  havia fumaça a
bordo e os passageiros entraram em pânico.
não há vítimas e o pouso ocorreu com
segurança. a aeronave está sendo rebocada
neste exato momento. 97 passageiros estão a
bordo. nota da avianca enviada para o diário
do brasil às 19:32
```

(b) Após o tratamento de números

```
avião com 0 passageiros faz pouso de
emergência no aeroporto de brasília.  um
avião acabou de fazer um pouso de emergência
no aeroporto jk.  a aeronave precisou
executar um pouso forçado devido a uma pane.
o voo da avianca partiu de guarulhos (sp) e
tinha como destino a cidade de juazeiro do
norte (ce). de acordo com informações
preliminares da globo news,  havia fumaça a
bordo e os passageiros entraram em pânico.
não há vítimas e o pouso ocorreu com
segurança. a aeronave está sendo rebocada
neste exato momento. 0 passageiros estão a
bordo. nota da avianca enviada para o diário
do brasil às 0:0
```

Fonte – Autor

4.2.1.3 Tratamento de URLs

A função responsável pelo tratamento de URLs pode ser vista na Figura 24 e um exemplo de sua aplicação na Figura 25.

Figura 24 – Função responsável pelo tratamento de URLs

```
# Dummy feature: URLs
def url_to_dummy(df):
    df = df.copy()
    pattern = """(?:(?:https?|ftp):\/\/|\b(?:[a-z\d]+\.))(?:(?:[^\s()<>]+|
                \((?:[^\s()<>]+|(?:\([^\s()<>]+\)))?\))+(?:\((?:[^\s()<>]+|
                (?:\(?:[^\s()<>]+\)))?\)|[^\s`!()\[\]{};:'".,<>?«»“”‘’]))?"""
    df['text'] = df['text'].apply(lambda x: re.sub(pattern, 'URL', x))
    return df
```

Fonte – Autor

Figura 25 – Exemplo de tratamento de URLs

(a) Antes do tratamento de URLs

```
a sexta-feira passada foi o dia do solto,
com a volta de aécio ao senado e a
libertação de rocha loures, para alívio de
temer, acossado por janot. ontem, segunda-
feira, foi o dia do opa. a prisão de geddel
deixa temer de calças na mão. loures solto
alivia. geddel preso preocupa. basta lembrar
a conversa com joesley e a forma desastrada
com que temer cuidou da denúncia do então
ministro da cultura, calero. e geddel também
está na mira da comissão de ética pública da
presidência, ao lado de kassab, marcos
pereira e guido mantega, os quatro
cavaleiros do apocalipse. kassab ainda deve
explicações sobre seu empenho em doar
patrimônio público às teles privatizadas e
perdoar 0 bi de dívidas da oi. (comentário
no jornal eldorado da rádio eldorado - fm 0
- na terça-feira 0 de julho de 0) para ouvir
clique no link abaixo e, em seguida, no
play: https://soundcloud.com/jose-neumanne-
pinto/neumanne-0-direto-ao-assunto para
ouvir quem
```

(b) Após o tratamento de URLs

```
a sexta-feira passada foi o dia do solto,
com a volta de aécio ao senado e a
libertação de rocha loures, para alívio de
temer, acossado por janot. ontem, segunda-
feira, foi o dia do opa. a prisão de geddel
deixa temer de calças na mão. loures solto
alivia. geddel preso preocupa. basta lembrar
a conversa com joesley e a forma desastrada
com que temer cuidou da denúncia do então
ministro da cultura, calero. e geddel também
está na mira da comissão de ética pública da
presidência, ao lado de kassab, marcos
pereira e guido mantega, os quatro
cavaleiros do apocalipse. kassab ainda deve
explicações sobre seu empenho em doar
patrimônio público às teles privatizadas e
perdoar 0 bi de dívidas da oi. (comentário
no jornal eldorado da rádio eldorado - fm 0
- na terça-feira 0 de julho de 0) para ouvir
clique no link abaixo e, em seguida, no
play: URL para ouvir quem
```

Fonte – Autor

4.2.1.4 Tratamento de e-mails

A Figura 26 mostra a função responsável pelo tratamento de e-mails para um único *token*. Um exemplo mostrando o resultado da aplicação desta função pode ser visualizado na Figura 27.

Figura 26 – Função responsável pelo tratamento de e-mails

```python
# Dummy feature: emails
def email_to_dummy(df):
    df = df.copy()
    df['text'] = df['text'].apply(lambda x: re.sub('[\w\+\.\-\~]+@[\w\.\-]+\.\w+', 'EMAIL', x))
    return df
```

Fonte – Autor

Figura 27 – Exemplo de tratamento de e-mails

(a) Antes do tratamento de e-mails

```
veja aqui o telefone e o e-mail de gilmar
mendes. demonstre sua indignação! gilmar
mendes conseguiu livrar michel temer e dilma
rousseff da condenação. em seu juízo a chapa
dilma/temer é honesta e estes dois
brasileiros merecem o respeito da nação.

eles estão de posse de seus direitos
políticos e poderão voltar em 0. dilma já
anunciou que se candidatará a senadora da
república. michel temer pode voltar até como
presidente. não permita que este julgamento
vergonhoso passe em branco. envie mensagens
pro e-mail de gilmar mendes ou até mesmo
telefone para ele manifestando toda sua
indignação. seguem aqui os contatos do
ministro que envergonhou o brasil.

ministro gilmar mendes
telefone: (0) 0-0
e-mail: mgilmar@stf.jus.br
e-mail: audienciasgilmarmendes@stf.jus.br
```

(b) Após do tratamento de e-mails

```
veja aqui o telefone e o e-mail de gilmar
mendes. demonstre sua indignação! gilmar
mendes conseguiu livrar michel temer e dilma
rousseff da condenação. em seu juízo a chapa
dilma/temer é honesta e estes dois
brasileiros merecem o respeito da nação.

eles estão de posse de seus direitos
políticos e poderão voltar em 0. dilma já
anunciou que se candidatará a senadora da
república. michel temer pode voltar até como
presidente. não permita que este julgamento
vergonhoso passe em branco. envie mensagens
pro e-mail de gilmar mendes ou até mesmo
telefone para ele manifestando toda sua
indignação. seguem aqui os contatos do
ministro que envergonhou o brasil.

ministro gilmar mendes
telefone: (0) 0-0
e-mail: EMAIL
e-mail: EMAIL
```

Fonte – Autor

4.2.1.5 TF-IDF

O trecho de código responsável pela construção da matriz TF-IDF é apresentado na Figura 28 e uma amostra da matriz final obtida na Figura 29.

Figura 28 – Trecho de código responsável pelo TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vect = vectorizer.fit(df_all_data['text'])
X = vectorizer.transform(df_all_data['text'])
```

Fonte – Autor

Figura 29 – Matriz TF-IDF

	0a	0am	0anos	0b	0bilhões	...	único	únicos	úteis	útero	útil
0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
5755	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
5756	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
5757	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0
5758	0.0	0.0	0.0	0.0	0.0	...	0.016483	0.0	0.0	0.0	0.0
5759	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0

[5760 rows x 41158 columns]

Fonte – Autor

4.2.2 Modelagem

Para encontrar os valores dos hiperparâmetros, foi realizada uma busca exaustiva utilizando a função *GridSearchCV*, a qual realiza validação cruzada para cada combinação dos hiperparâmetros definidos na *grid*.

Os hiperparâmetros e seus respectivos valores a serem testados são apresentados na Tabela 7.

Tabela 7 – Hiperparâmetros e seus valores a serem testados

Solver	**Penalty**	**C**	**l1_ratio**
lbfgs	none		
lbfgs	l2	5, 10, 15	
liblinear	l1, l2	5, 10, 15	
saga	none		
saga	elasticnet	5, 10, 15	0, 0.5, 1

Fonte – Autor

A Figura 30 mostra o trecho de código responsável pela busca exaustiva dos hiperparâmetros.

Figura 30 – Trecho de código da implementação do método 1

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

clf = LogisticRegression(max_iter=200, random_state=42)

grid_values = [{'solver': ['lbfgs'],
                'penalty': ['none']},
               {'solver': ['lbfgs'],
                'penalty': ['l2'],
                'C': [5, 10, 15]},
               {'solver': ['liblinear'],
                'penalty': ['l1', 'l2'],
                'C': [5, 10, 15]},
               {'solver': ['saga'],
                'penalty': ['none']},
               {'solver': ['saga'],
                'penalty': ['elasticnet'],
                'C':[5, 10, 15],
                'l1_ratio': [0, 0.5, 1]}]

grid_clf = GridSearchCV(clf,
                        cv=5,
                        param_grid=grid_values,
                        scoring=['accuracy', 'precision', 'recall', 'f1'],
                        return_train_score=True,
                        refit='f1',
                        verbose=2,
                        n_jobs=-1)

grid_clf.fit(df_X, df_y)
```

Fonte – Autor

4.3 MÉTODO 2

4.3.1 Pré-processamento dos dados

4.3.1.1 Conversão para letra minúscula e tratamento de números, URLS e e-mails

Essas etapas de pré-processamento são as mesmas já apresentadas nas seções 4.2.1.1, 4.2.1.2, 4.2.1.3 e 4.2.1.4.

4.3.1.2 Mapeamento de palavras para números

Esta etapa tem por objetivo transformar sequências de *tokens* (palavras) para sequências de números inteiros. Para realização deste mapeamento, foi utilizado o objeto *Tokenizer* do *Keras*, em que todas as palavras foram mapeadas para inteiros, os quais representam as palavras ordenadas pela frequência.

A Figura 31 mostra o trecho de código responsável pelo mapeamento de palavras para números. Nele, o objeto *Tokenizer* recebe o valor 40000 no argumento *num_words*. Isso significa que serão consideradas as 40000 palavras mais frequentes dos textos no mapeamento. Após o objeto *Tokenizer* ser treinado com os dados de

treinamento, ele é utilizado para a conversão das palavras para números, os quais são armazenados em *encoded_docs*.

Figura 31 – Trecho de código responsável pelo mapeamento de palavras para números

```
from tensorflow.keras.preprocessing.text import Tokenizer

TOKENIZER__NUM_WORDS =  40000

tokenizer = Tokenizer(num_words=TOKENIZER__NUM_WORDS)
tokenizer.fit_on_texts(texts)

encoded_docs = tokenizer.texts_to_sequences(texts)
```

Fonte – Autor

A Figura 32 apresenta o resultado do mapeamento de palavras para números.

Figura 32 – Exemplo de mapeamento de palavras para números

(a) Sequências de palavras

```
In [24]: print(texts[0])
avião com 0 passageiros faz pouso de emergência no
aeroporto de brasília.  um avião acabou de fazer um
pouso de emergência no aeroporto jk.  a aeronave
precisou executar um pouso forçado devido a uma pane.
o voo da avianca partiu de guarulhos (sp) e tinha como
destino a cidade de juazeiro do norte (ce). de acordo
com informações preliminares da globo news,  havia
fumaça a bordo e os passageiros entraram em pânico.
não há vítimas e o pouso ocorreu com segurança. a
aeronave está sendo rebocada neste exato momento. 0
passageiros estão a bordo. nota da avianca enviada
para o diário do brasil às 0:0
```

(b) Sequências de números

```
In [25]: print(encoded_docs[0])
[717, 13, 8, 2071, 226, 3502, 1, 1810, 12, 1010, 1,
162, 11, 717, 647, 1, 148, 11, 3502, 1, 1810, 12,
1010, 15276, 3, 1417, 2290, 6963, 11, 3502, 4620,
1102, 3, 16, 8593, 2, 1342, 7, 18647, 3160, 1, 6235,
204, 5, 229, 29, 1811, 3, 224, 1, 24783, 6, 68, 1792,
1, 78, 13, 241, 5660, 7, 240, 1103, 250, 5940, 3,
4811, 5, 18, 2071, 3032, 9, 2333, 15, 75, 937, 5, 2,
3502, 623, 13, 164, 3, 1417, 36, 203, 24784, 273,
2971, 209, 8, 2071, 82, 3, 4811, 288, 7, 18647, 2334,
10, 2, 755, 6, 51, 143, 8, 8]
```

Fonte – Autor

4.3.1.3 Padronização do tamanho das sequências

A etapa de padronização do tamanho das sequências se faz necessária, uma vez que o modelo sequencial exige que todas as amostras possuam o mesmo comprimento. Dessa forma, será definido um comprimento padrão para todas as sequências.

A Figura 33 apresenta a função responsável pela padronização do tamanho das sequências. A funcão *pad_sequences* recebe através do argumento *maxlen* o tamanho

final desejado, que foi configurado para 800. Foram realizados testes preliminares com valores maiores, porém resultavam em muito esforço computacional. O preenchimento e truncamento foram configurados através dos parâmetros *padding* e *truncating* para serem realizados na parte final das sequências. Sequências maiores que o tamanho padrão de 800 foram truncadas e sequências menores foram preenchidas com o valor 0 até atingir o tamanho padrão. O preenchimento é feito com o valor 0, pois este é o valor utilizado pela camada de mascaramento do modelo sequencial.

Figura 33 – Função responsável pela padronização do tamanho das sequências

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

padded_sequence = pad_sequences(encoded_docs,
                                padding='post',
                                truncating='post',
                                maxlen=800)
```

Fonte – Autor

A Figura 34 apresenta o resultado da padronização do tamanho de uma sequência.

Figura 34 – Exemplo de padronização do tamanho das sequências

(a) Sequências de números

```
In [43]: print(encoded_docs[0])
[717, 13, 8, 2071, 226, 3502, 1, 1810, 12, 1010, 1, 162, 11, 717, 647,
1, 148, 11, 3502, 1, 1810, 12, 1010, 15276, 3, 1417, 2290, 6963, 11,
3502, 4620, 1102, 3, 16, 8593, 2, 1342, 7, 18647, 3160, 1, 6235, 204, 5,
229, 29, 1811, 3, 224, 1, 24783, 6, 68, 1792, 1, 78, 13, 241, 5660, 7,
240, 1103, 250, 5940, 3, 4811, 5, 18, 2071, 3032, 9, 2333, 15, 75, 937,
5, 2, 3502, 623, 13, 164, 3, 1417, 36, 203, 24784, 273, 2971, 209, 8,
2071, 82, 3, 4811, 288, 7, 18647, 2334, 10, 2, 755, 6, 51, 143, 8, 8]

In [44]: len(encoded_docs[0])
Out[44]: 106
```

(b) Sequências de números com tamanho padronizado

```
In [48]: print(padded_sequence[0])
[  717    13     8  2071   226  3502     1  1810    12  1010     1   162
    11   717   647     1   148    11  3502     1  1810    12  1010 15276
     3  1417  2290  6963    11  3502  4620  1102     3    16  8593     2
  1342     7 18647  3160     1  6235   204     5   229    29  1811     3
   224     1 24783     6    68  1792     1    78    13   241  5660     7
   240  1103   250  5940     3  4811     5    18  2071  3032     9  2333
    15    75   937     5     2  3502   623    13   164     3  1417    36
   203 24784   273  2971   209     8  2071    82     3  4811   288     7
 18647  2334    10     2   755     6    51   143     8     8     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0

                                    ...

     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0]

In [49]: len(padded_sequence[0])
Out[49]: 800
```

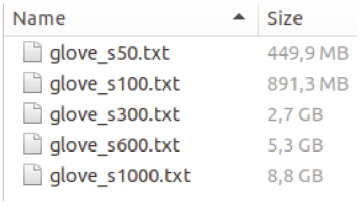
Fonte – Autor

4.3.2 Modelagem

Para simplificar a explanação, visto que foram realizados três experimentos em que as implementações seguem ideias parecidas, esta seção apresenta detalhes unicamente da implementação do experimento 2.

Os arquivos de *word embedding* GloVe já treinados para a Língua Portuguesa e utilizados neste trabalho foram disponibilizados por (HARTMANN *et al.*, 2017) no formato *txt*, conforme mostrado na Figura 35. Cada arquivo possui na linha inicial os metadados e para cada linha seguinte uma palavra com seu respectivo vetor.

Figura 35 – Arquivos de *word embedding* GloVe treinados.



Fonte – Autor

A Figura 36 apresenta a função responsável pelo carregamento dos arquivos *word embedding* já treinados, em que com o auxílio da biblioteca *Gensim*, retorna-se uma matriz, que mais a frente será utilizada na camada de *Embedding* do modelo.

Figura 36 – Função responsável pelo carregamento do *word embedding*

```
import gensim

def pretreined_embedding(file_path, embedding_dim, num_words, word_index):
    """
    This function creates a embedding_matrix to be used in the Embedding Layer
    of the sequential model

    Parameters
    ----------
    file_path : str
        Directory where the glove file are located
    embedding_dim : int
        The word embedding dimension
    num_words : int
        Number of words to load
    word_index : dict
        a dict containing words and the integers associated with them

    Returns
    -------
    embedding_matrix : numpy.ndarray
        An embedding matrix with size (num_words, embedding_dim) containing the
        word vectors associated

    """

    # load in pre-trained word vectors
    print(f'Loading word vectors with dim {embedding_dim}')
    embeddings_index = gensim.models.KeyedVectors.load_word2vec_format(file_path)

    # prepare embedding matrix
    print('Filling pre-trained embeddings...')

    embedding_matrix = np.zeros((num_words, embedding_dim))

    for word, i in word_index.items():
        if i < num_words:
            try:
                embedding_vector = embeddings_index[word]
                embedding_matrix[i] = embedding_vector
            except KeyError as e:
                print(e)

    return embedding_matrix
```

Fonte – Autor

Para a criação do modelo de rede neural do tipo LSTM foi utilizado a biblioteca *Keras*, mais especificamente a API Sequencial, que permite a criação do modelo através do empilhamento de camadas.

A Figura 37 apresenta a função responsável pela criação e compilação do modelo.

Figura 37 – Função responsável pela criação e compilação do modelo sequencial

```
def build_model(embedding_dim, embedding_matrix, lstm_units, dropout_rate):
    """
    This function create and compile a sequential model

    Parameters
    ----------
    embedding_dim : int
        The word embedding dimension.
    embedding_matrix : numpy.ndarray
        The trained GloVe embedding_matrix that to be load in the Embedding layer.
    lstm_units : int
        number of neurons in the LSTM layer.
    dropout_rate : float (between 0 and 1)
        Fraction of the units to drop for the linear transformation of the inputs
        and for linear transformation of recurrent state.

    Returns
    -------
    model : tensorflow.python.keras.engine.sequential.Sequential
        The compiled sequential model.

    """

    model = keras.models.Sequential()

    # Embedding layer
    model.add(
        keras.layers.Embedding(input_dim=TOKENIZER__NUM_WORDS,
                               output_dim=embedding_dim,
                               input_length=padded_sequence.shape[1],
                               mask_zero=True,
                               weights=[embedding_matrix],
                               trainable=False))

    # LSTM layer
    model.add(keras.layers.LSTM(units=lstm_units,
                                dropout=dropout_rate,
                                recurrent_dropout=dropout_rate))

    # Dense layer
    model.add(keras.layers.Dense(units=1, activation='sigmoid'))

    # compile the model
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=[
                      'accuracy',
                      keras.metrics.Precision(name='precision'),
                      keras.metrics.Recall(name='recall')])

    return model
```

Fonte – Autor

Abaixo, descreve-se as etapas da função *build_model*.

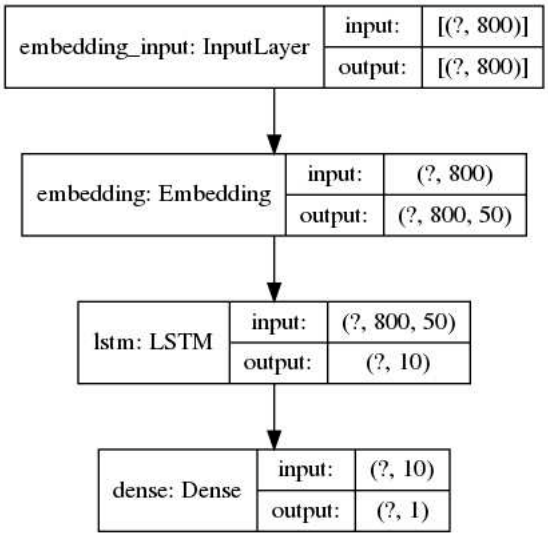
- Inicialmente o modelo sequencial é criado.
- A primeira camada adicionada ao modelo é a de *Embedding*. Os três principais argumentos são *input_dim*, *output_dim* e *input_lenght*, que são, respectivamente, o tamanho do dicionário, a dimensão da saída e o tamanho das sequências de entrada. Utiliza-se o argumento *mask_zero* = *True* para ativar o mascaramento

dos valores preenchidos na etapa de pré-processamento, *weights* para carregar a matriz treinada (GloVe) e por fim, *trainable* = *False* para desativar o treinamento desta camada, pois estamos importando uma matriz que já está treinada (GloVe).

- Em seguida é adicionada a camada LSTM. O argumento *units* é utilizado para configurar a dimensionalidade da saída, enquanto *dropout* e *recurrent_dropout* são utilizados para configurar a taxa de *dropout* na entrada e entre as unidades recorrentes, respectivamente.
- A camada de saída consiste em um único neurônio utilizando a função sigmóide, que permite calcular a probabilidade da notícia é falsa.
- Por fim, o modelo é compilado. Nesta etapa são especificados a função custo, o otimizador e as métricas.

Para confirmar a ordem das camadas e as dimensões, foram gerados imagens como a da Figura 38, que apresenta a visualização do modelo experimento 2 contendo *word embedding* com dimensão 50 e 10 células LSTM. A camada de *Embedding*, como esperado, recebe sequências de tamanho 800, e para cada valor da sequência, associa um vetor de palavras. Em seguida, a camada LSTM faz o processamento e expõe na saída vetores de dimensão 10. Por fim, uma camada de saída densa com 1 neurônio realiza predições entre 0 e 1 para as duas classes de notícias do problema.

Figura 38 – Visualização do modelo do experimento 2 com *word embedding* treinado de dimensão 50 e LSTM com 10 células.



Fonte – Autor

A Figura 39 apresenta o código de treinamento do experimento 2. Inicialmente é realizada a separação dos dados nos conjuntos de treino e validação. Em seguida,

foi selecionado a dimensão do *word embedding* e do número de neurônios através de dois laços *for* para o modelo ser treinado. Finalmente, são salvos o modelo treinado e seus erros.

Figura 39 – Código de treinamento do modelo do experimento 2

```
X_train, X_val, y_train, y_val = train_test_split(padded_sequence, y,
                                                  test_size=0.20, random_state=42)

for embedding_dim in LIST_EMBEDDING_OUTPUT_DIM: # [50, 100, 300, 600, 1000]

    # load embedding matrix
    file_path = os.path.join(path_dir, dict_embedding_dim[embedding_dim])
    embedding_matrix = pretreined_embedding(file_path, embedding_dim,
                                            TOKENIZER__NUM_WORDS, word_index)

    for lstm_units in LIST_LSTM_UNITS: # [10, 20, 30, 40, 50, 60, 70]

        # create and compile model
        tf.keras.backend.clear_session()
        model = build_model(embedding_dim, embedding_matrix, lstm_units, DROPOUT_RATE)

        # fit the model
        history = model.fit(X_train, y_train,
                            validation_data=(X_val, y_val),
                            epochs=NUM_EPOCHS,
                            batch_size=NUM_BATCH_SIZE,
                            callbacks=[time_callback, es_callback])

        # save history to csv
        filename = f'neur{lstm_units}_embdim{embedding_dim}'

        # save model and architecture to single file
        DIR_RESULT_H5 = os.path.join(BASE_DIR, '3_Resultados', 'lstm',
                                     f'{EXPERIMENT_NAME}', f'{filename}.h5')
        model.save(DIR_RESULT_H5)

        # save history to csv
        df_history = pd.DataFrame(history.history)
        df_history['val_f1_score'] = (2 * df_history['val_precision'] * df_history['val_recall']) /
                                     (df_history['val_precision'] + df_history['val_recall'])
        df_history['f1_score'] = (2 * df_history['precision'] * df_history['recall']) /
                                 (df_history['precision'] + df_history['recall'])
        df_history.insert(0, 'time', time_callback.times)
        df_history.insert(0, 'epoch', df_history.index)
        DIR_RESULT_CSV_HISTORY = os.path.join(BASE_DIR, '3_Resultados', 'lstm',
                                              f'{EXPERIMENT_NAME}', f'{filename}.csv')
        df_history.to_csv(DIR_RESULT_CSV_HISTORY, index=False)

        # save history to png
        df_history[['loss', 'val_loss', 'f1_score', 'val_f1_score']].plot(grid=True, figsize=(10,5))
        plt.title(EXPERIMENT_NAME)
        plt.xlabel('epoch')
        plt.tight_layout()
        DIR_RESULT_FIG = os.path.join(BASE_DIR, '3_Resultados', 'lstm',
                                      f'{EXPERIMENT_NAME}', f'{filename}.png')
        plt.savefig(DIR_RESULT_FIG)
        plt.close()

        del model
```

Fonte – Autor

5 RESULTADOS

Esta seção apresenta os resultados para cada método e ao final é apresentado o resultado da interface web.

5.1 MÉTODO 1 - REGRESSÃO LOGÍSTICA

5.1.1 Modelagem

Após a busca exaustiva nos valores dos hiperparâmetros, chegou-se nos resultados apresentados na Tabela 8.

Tabela 8 – Resultados da otimização de hiperparâmetros

	Parâmetros				**Métricas**			
	penalty	solver	C	l1_ratio	Accuracy	Precision	Recall	F1-score
0	none	lbfgs			0,912	0,911	**0,913**	0,912
1	l2	lbfgs	5		0,911	0,913	0,909	0,911
2	l2	lbfgs	10		0,913	0,916	0,910	0,913
3	l2	lbfgs	15		0,913	0,916	0,909	0,912
4	l1	liblinear	5		0,903	0,903	0,904	0,903
5	l2	liblinear	5		0,911	0,913	0,909	0,911
6	l1	liblinear	10		0,902	0,904	0,900	0,902
7	l2	liblinear	10		0,913	0,916	0,910	0,913
8	l1	liblinear	15		0,900	0,902	0,898	0,900
9	l2	liblinear	15		0,913	0,916	0,909	0,912
10	none	saga			0,911	0,912	0,910	0,911
11	elasticnet	saga	5	0	0,911	0,913	0,909	0,911
12	elasticnet	saga	5	0,5	0,911	0,913	0,909	0,911
13	elasticnet	saga	5	1	0,904	0,904	0,904	0,904
14	elasticnet	saga	10	0	**0,913**	**0,916**	0,910	**0,913**
15	elasticnet	saga	10	0,5	0,911	0,913	0,907	0,910
16	elasticnet	saga	10	1	0,906	0,907	0,905	0,906
17	elasticnet	saga	15	0	0,913	0,916	0,909	0,912
18	elasticnet	saga	15	0,5	0,909	0,911	0,907	0,909
19	elasticnet	saga	15	1	0,908	0,911	0,905	0,907

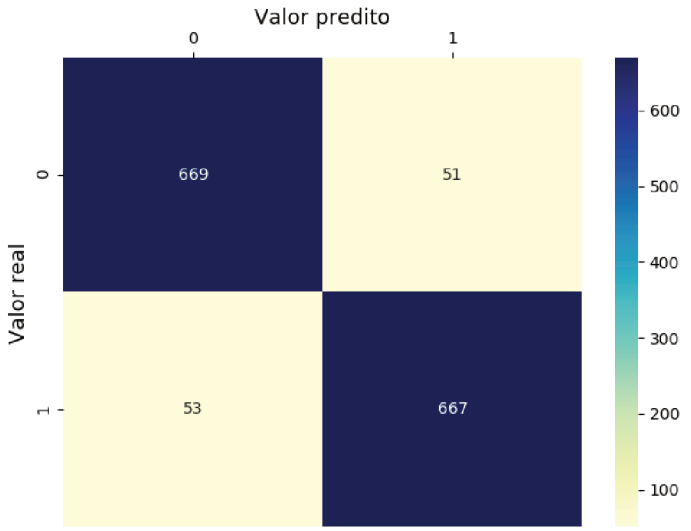
Fonte – Autor

O melhor conjunto de hiperparâmetros é o da linha 14.

5.1.2 Avaliação

Um modelo com o melhor conjunto de hiperparâmetros foi treinado utilizando todos os dados de treinamento e avaliado no conjunto de teste, obtendo a matriz de confusão da Figura 40 e as métricas da Tabela 9.

Figura 40 – Matriz de confusão nos dados de teste



Fonte – Autor

Tabela 9 – Resultados do método 1 nos dados de TESTE

Métricas	**Valor**
Acurácia	0,928
Precisão	0,929
Recall	0,926
F1-score	0,928

Fonte – Autor

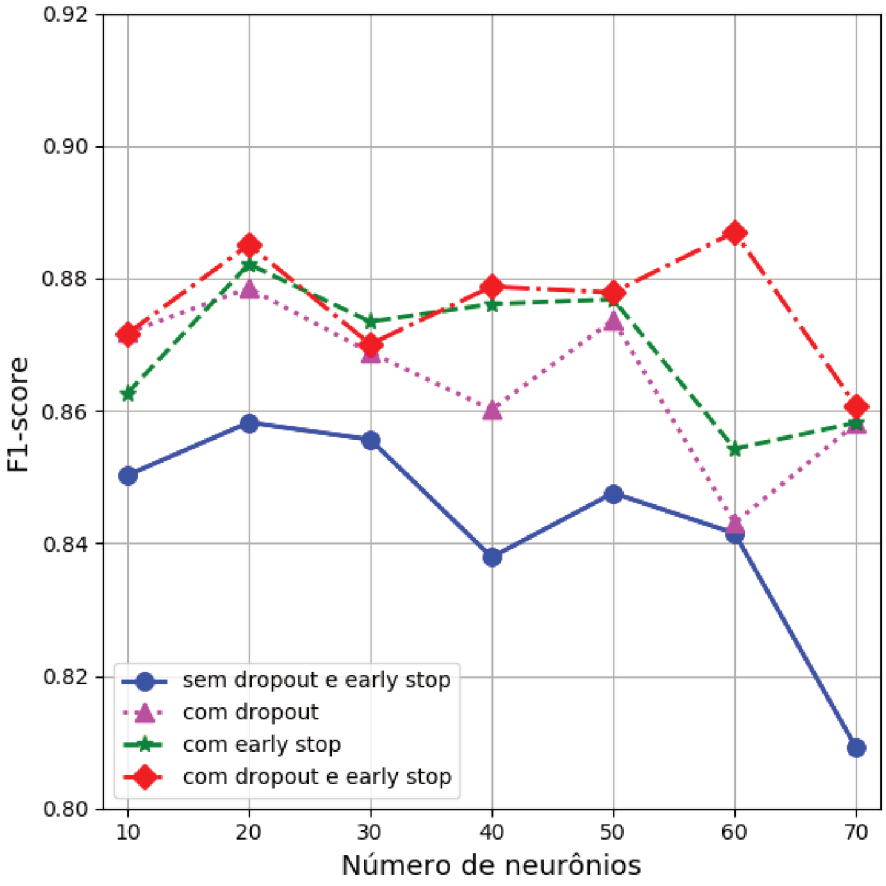
5.2 MÉTODO 2 - REDE NEURAL DO TIPO LSTM

5.2.1 Modelagem

5.2.1.1 Experimento 1

A Figura 41 apresenta o resultado da validação cruzada do experimento 1.

Figura 41 – Resultado da validação cruzada

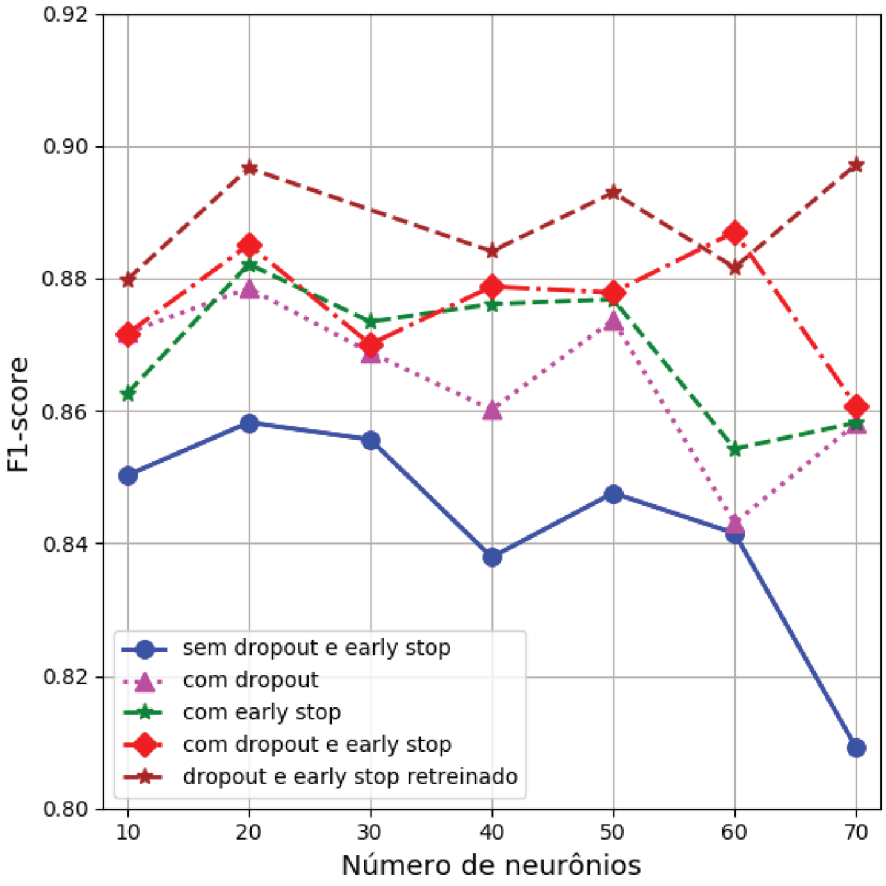


Fonte – Autor

Observa-se que a configuração com melhor desempenho é a que utiliza *dropout e early stop*.

Com esta configuração, um novo modelo foi treinado utilizando os *dados de treino* e avaliado nos *dados de validação* da Figura 18, obtendo a curva marrom da Figura 42. Como já esperado, obteve-se melhor resultado que as demais devido ao treinamento ser realizado com mais dados.

Figura 42 – Resultados do experimento 1



Fonte – Autor

Os valores para *dropout e early stop retreinado* são apresentados na Tabela 10

Tabela 10 – Resultado experimento 1

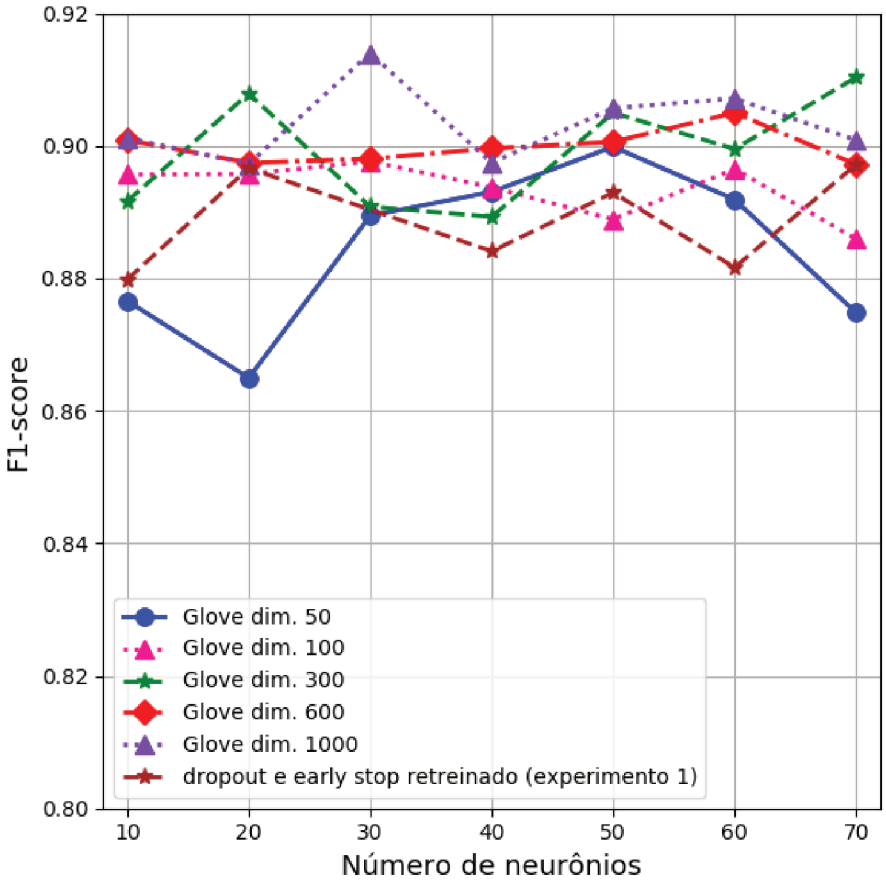
Nro. neurônios	**F1-score**
10	0,879
20	0,896
40	0,884
50	0,893
60	0,882
70	0,897

Fonte – Autor

5.2.1.2 Experimento 2

A Figura 43 apresenta os resultados ao utilizar o *word embedding* já treinado com a melhor configuração do experimento 1, que foi utilizar dropout e early stop.

Figura 43 – Resultados do experimento 2



Fonte – Autor

A Tabela 11 apresenta a média e o desvio padrão da métrica F1-score.

Tabela 11 – Resultado experimento 2

Curva	**F1-score**	
	Média	**Desvio padrão**
Word embedding dim. 50	0,884	0,012
Word embedding dim. 100	0,893	0,004
Word embedding dim. 300	0,899	0,008
Word embedding dim. 600	0,899	0,002
Word embedding dim. 1000	0,903	0,006
Dropout e early stop retreinado	0,888	0,007

Fonte – Autor

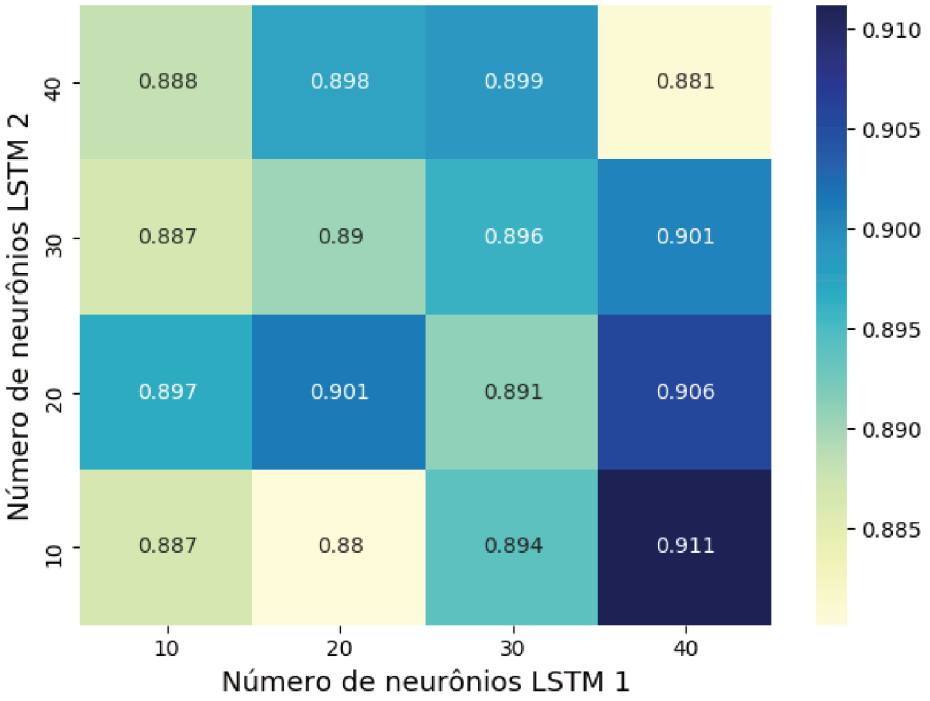
De acordo com a Tabela 11, será fixado a dimensão 600, que apresenta o segundo melhor valor médio e o menor desvio padrão.

5.2.1.3 Experimento 3

Os modelos treinados no experimento 3 foram configurados utilizando dropout e early stop, obtidos do experimento 1 e, word embedding GloVe com dimensão 600, obtido do experimento 2.

Os resultados alcançados são apresentados na Figura 44.

Figura 44 – Resultados do experimento 3



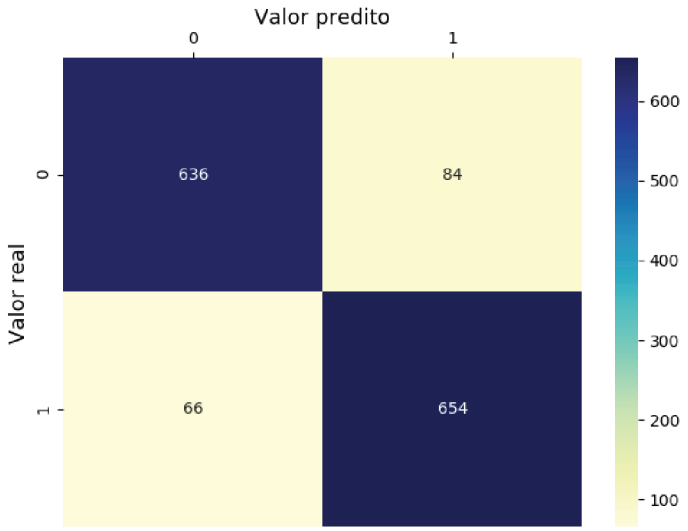
Fonte – Autor

5.2.2 Avaliação

Após todos experimentos, o melhor desempenho foi obtido no experimento 2, utilizando uma camada escondida com 30 neurônios e o word embedding com dimensão 1000.

Este modelo foi avaliado no conjunto de teste, obtendo a matriz de confusão da Figura 45 e as métricas da Tabela 12.

Figura 45 – Matriz de confusão nos dados de teste do método 2



Fonte – Autor

Tabela 12 – Resultados do método 2 nos dados de TESTE

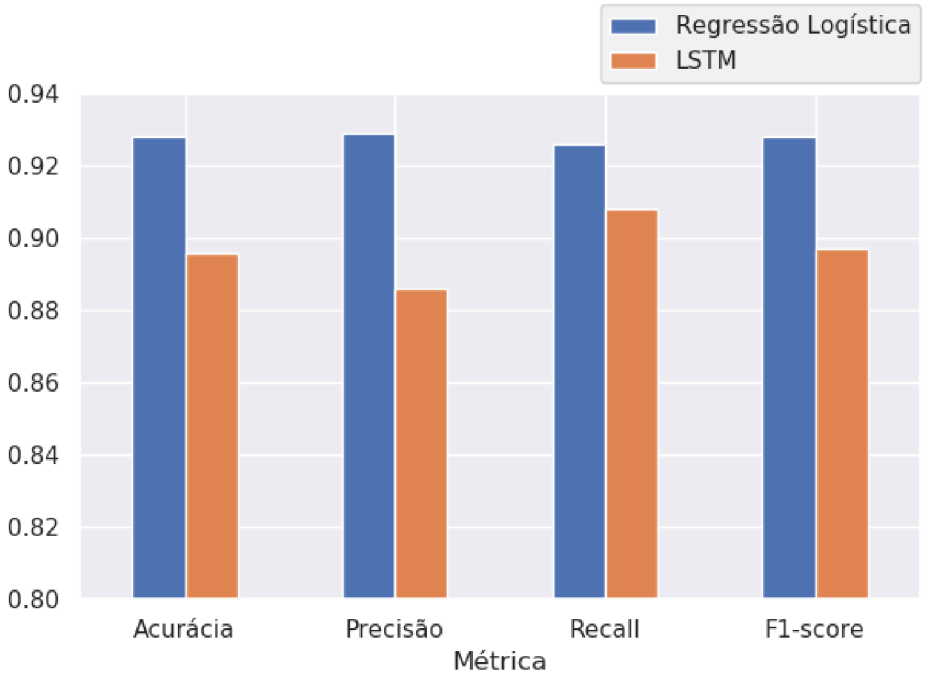
Métricas	**Valor**
Acurácia	0,896
Precisão	0,886
Recall	0,908
F1-score	0,897

Fonte – Autor

5.3 COMPARAÇÃO ENTRE OS DOIS MÉTODOS

A Figura 46 apresenta visualmente o desempenho dos dois métodos avaliados nos dados de TESTE, que tem seus valores apresentados nas Tabelas 9 e 12. Como pode ser observado, o método 1, que utilizou regressão logística, teve desempenho superior ao método 2, que utilizou redes neurais recorrentes do tipo LSTM.

Figura 46 – Comparação dos resultados do método 1 e do método 2 avaliados nos dados de TESTE

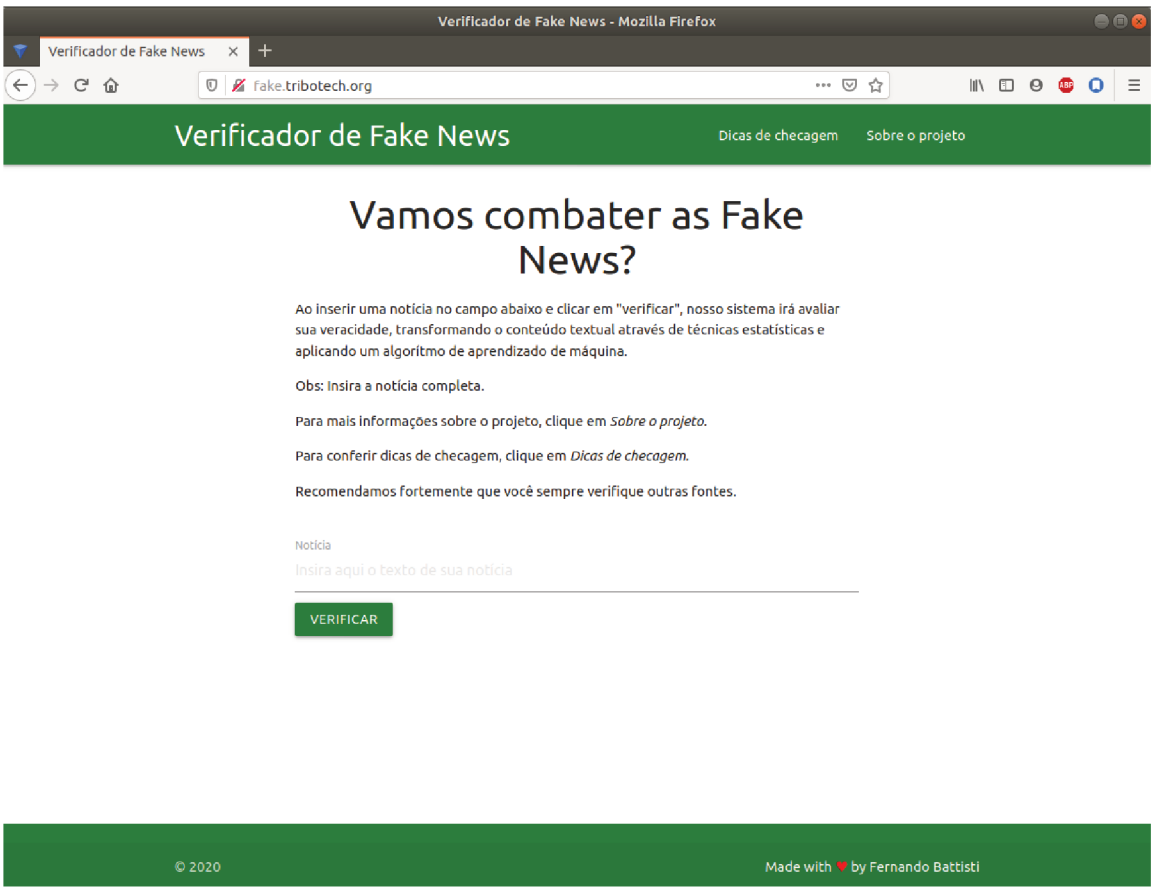


Fonte – Autor

5.4 INTERFACE WEB

A tela inicial da interface web é apresentada na Figura 47.

Figura 47 – Tela inicial da interface web



Fonte – Autor

Após o usuário inserir o conteúdo textual da notícia e clicar no botão *verificar*, os dados textuais serão processados e avaliados pelo modelo de aprendizado de máquina no *back-end* e, por conseguinte, o resultado da checagem é apresentado, conforme Figura 48.

Figura 48 – Resultado de uma verificação de notícia apresentado pela interface web

Fonte – Autor

6 CONCLUSÃO

O objetivo do projeto final de curso, o qual foi atingido com sucesso, foi o desenvolvimento de um modelo de aprendizado de máquina para classificar notícias como falsas ou verdadeiras.

Diferentes técnicas de programação e aprendizado de máquina foram utilizados ao longo deste trabalho, optando-se por opções modernas e de código aberto, como a linguagem Python.

O modelo de regressão logística teve o pré-processamento de dados levando em consideração a abordagem de representação estatística TF-IDF. Testou-se diferentes variações nos valores dos hiperparâmetros *solver*, *penalty*, *C* e *l1_ratio* através de uma busca exaustiva com validação cruzada. Por fim, o melhor modelo foi retreinado utilizando todos os dados de treino e avaliado nos dados de teste, o qual obteve-se um valor de *F1-score* de $92,8\%$.

O modelo de redes neurais recorrentes do tipo LSTM levou em consideração um pré-processamento para representação de dados sequenciais. Devido ao custo computacional para o treinamento deste modelo, variações do mesmo foram realizadas em três diferentes experimentos: o primeiro para avaliar o desempenho quanto ao uso de *dropout* e do *early stop*, o segundo para avaliar o desempenho ao utilizar o *word embedding* GloVe já treinado e o terceiro para avaliar a utilização de duas camadas escondidas. As características do melhor modelo contemplam a utilização de *dropout*, *early stop*, *word embedding* com dimensão 1000 e uma camada escondida de 30 neurônios. Esse modelo, quando avaliado nos dados de teste, obteve um valor de *F1-score* de $89,6\%$.

Uma aplicação web monolítica foi desenvolvida para fazer a publicação do melhor modelo. Nela, um usuário consegue inserir o conteúdo textual de uma notícia através de uma interface e obter a veracidade da mesma.

Para melhoramento e trabalhos futuros, sugere-se alguns pontos citados a seguir:

- Variações do tamanho padrão das sequências durante a etapa de pré-processamento;
- Criação de variáveis linguísticas;
- Testar métodos de *ensamble learning*, combinando múltiplos modelos para produzir um modelo agrupado;
- Testar outros modelos de aprendizado de máquina, como o *Transformer*;
- Nas redes neurais recorrentes do tipo LSTM, sugere-se testar:
  - Treinamento bidirecional (Bi-LSTM);

  - Outros tipos de *word embedding* já treinados, por exemplo, os que utilizam *Word2Vec*, *Wang2Vec* e *FastText*;
  - Variações do batch de treinamento e na taxa de *dropout*.

- Para interface web, sugere-se:
  - Implementar um botão para o usuário enviar *feedback* da classificação da notícia;
  - Armazenar em um banco de dados os textos inseridos pelo usuário, bem como os resultados das classificações e seus feedbacks;
  - Implementar uma arquitetura de microserviços.

REFERÊNCIAS

CASALI, P. Aplicação de técnicas de data mining para previsibilidade eleitoral, 2018.

CHOLLET, François. **Deep Learning with Python**. [*S.l.*: *s.n.*], 2017. ISBN 9781617294433.

DEVELOPERS, GOOGLE. **Text classification guide - Step 3: Prepare Your Data**. Acessado em: Agosto de 2020. Disponível em: https://developers.google.com/machine-learning/guides/text-classification/step-3.

FAKE.BR-CORPUS. **Fake.Br Corpus**. Acessado em: Julho de 2020. Disponível em: https://github.com/roneysco/Fake.br-Corpus.

FATOS, AOS. **Como fazer sua própria checagem de fatos e detectar notícias falsas**. Acessado em: Julho de 2020. Disponível em: https://www.aosfatos.org/noticias/como-fazer-sua-propria-checagem-de-fatos-e-detectar-noticias-falsas/.

GÉRON, Aurélien. **Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. 2nd. [*S.l.*]: O'Reilly Media, Inc., 2019. ISBN 9781492032649.

HARTMANN, Nathan; FONSECA, Erick; SHULBY, Christopher; TREVISO, Marcos; RODRIGUES, Jessica; ALUISIO, Sandra. **Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks**. [*S.l.*: *s.n.*], 2017. arXiv: 1708.06025 [cs.CL].

JR., Edson C. Tandoc; LIM, Zheng Wei; LING, Richard. Defining "Fake News". **Digital Journalism**, Routledge, v. 6, n. 2, p. 137–153, 2018. DOI: 10.1080/21670811.2017.1360143. eprint: https://doi.org/10.1080/21670811.2017.1360143. Disponível em: https://doi.org/10.1080/21670811.2017.1360143.

MESQUITA, Claudio Tinoco; OLIVEIRA, Anderson; SEIXAS, FlÃ¡vio Luiz; PAES, Aline. Infodemia, Fake News and Medicine: Science and The Quest for Truth. en. **International Journal of Cardiovascular Sciences**, scielo, v. 33, p. 203–205, mai. 2020. ISSN 2359-5647. Disponível em:

http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2359-56472020000300203&nrm=iso.

MICELI, P. A.; BLAIR, W. Dale; BROWN, M. M. **Isolating Random and Bias Covariances in Tracks**. [*S.l.*: *s.n.*], 2018. P. 2437–2444. ISBN 9780996452762. DOI: 10.23919/ICIF.2018.8455530.

MITCHELL, Tom M. **Machine Learning**. New York: McGraw-Hill, 1997. ISBN 0070428077.

MOHAMMED, M.; KHAN, M.B.; BASHIER, E.B.M. **Machine Learning: Algorithms and Applications**. [*S.l.*]: CRC Press, 2017.

MONTEIRO, Rafael A.; SANTOS, Roney L.S.; PARDO, Thiago A.S.; ALMEIDA, Tiago A. de; RUIZ, Evandro E.S.; VALE, Oto A. Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, 11122 LNAI, p. 324–334, 2018. ISSN 16113349. DOI: 10.1007/978-3-319-99722-3_33.

RUBIN, Victoria L.; CONROY, Niall J. Challenges in automated deception detection in computer-mediated communication. **Proceedings of the ASIST Annual Meeting**, v. 48, 2011. ISSN 15508390. DOI: 10.1002/meet.2011.14504801098.

SCIKIT-LEARN. **Cross-validation: evaluating estimator performance**. Acessado em: Julho de 2020. Disponível em: https://scikit-learn.org/stable/modules/cross_validation.html.

SILVA, Renato M.; SANTOS, Roney L.S.; ALMEIDA, Tiago A.; PARDO, Thiago A.S. Towards automatically filtering fake news in Portuguese. **Expert Systems with Applications**, v. 146, p. 113199, 2020. ISSN 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113199. Disponível em: http://www.sciencedirect.com/science/article/pii/S0957417420300257.

ZHOU, Xinyi; WU, Jindi; ZAFARANI, Reza. **SAFE: Similarity-Aware Multi-Modal Fake News Detection**. [*S.l.*: *s.n.*], 2020. arXiv: 2003.04981 [cs.CL].

ANEXO A – FONTES E ESTATÍSTICAS DO CORPORA UTILIZADO PARA TREINAMENTO DO *WORD EMBEDDING* GLOVE

Corpus	Tokens	Types	Genre	Description
LX-Corpus [Rodrigues et al. 2016]	714,286,638	2,605,393	Mixed genres	A huge collection of texts from 19 sources. Most of them are written in European Portuguese.
Wikipedia	219,293,003	1,758,191	Encyclopedic	Wikipedia dump of 10/20/16
GoogleNews	160,396,456	664,320	Informative	News crawled from GoogleNews service
SubIMDB-PT	129,975,149	500,302	Spoken language	Subtitles crawled from IMDb website
G1	105,341,070	392,635	Informative	News crawled from G1 news portal between 2014 and 2015.
PLN-Br [Bruckschen et al. 2008]	31,196,395	259,762	Informative	Large corpus of the PLN-BR Project with texts sampled from 1994 to 2005. It was also used by [Hartmann 2016] to train word embeddings models
Literacy works of public domain	23,750,521	381,697	Prose	A collection of 138,268 literary works from the Domínio Público website
Lacio-web [Aluísio et al. 2003]	8,962,718	196,077	Mixed genres	Texts from various genres, e.g., literary and its subdivisions (prose, poetry and drama), informative, scientific, law, didactic technical
Portuguese e-books	1,299,008	66,706	Prose	Collection of classical fiction books written in Brazilian Portuguese crawled from Literatura Brasileira website
Mundo Estranho	1,047,108	55,000	Informative	Texts crawled from Mundo Estranho magazine
CHC	941,032	36,522	Informative	Texts crawled from Ciência Hoje das Crianças (CHC) website
FAPESP	499,008	31,746	Science Communication	Brazilian science divulgation texts from Pesquisa FAPESP magazine
Textbooks	96,209	11,597	Didactic	Texts for children between 3rd and 7th-grade years of elementary school
Folhinha	73,575	9,207	Informative	News written for children, crawled in 2015 from Folhinha issue of Folha de São Paulo newspaper
NILC subcorpus	32,868	4,064	Informative	Texts written for children of 3rd and 4th-years of elementary school
Para Seu Filho Ler	21,224	3,942	Informative	News written for children, from Zero Hora newspaper
SARESP	13,308	3,293	Didactic	Text questions of Mathematics, Human Sciences, Nature Sciences and essay writing to evaluate students
Total	1,395,926,282	3,827,725		

Fonte – (HARTMANN *et al.*, 2017)