UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Héderson Pereira dos Santos

Classificação multicategórica de notícias em língua portuguesa utilizando o Processamento de Linguagem Natural

São Carlos

2021

Héderson Pereira dos Santos

Classificação multicategórica de notícias em língua portuguesa utilizando o Processamento de Linguagem Natural

Trabalho de conclusão de curso apresentado ao Centro de Ciências Matemáticas Aplicadas à Indústria do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, como parte dos requisitos para conclusão do MBA em Ciências de Dados.

Área de concentração: Ciências de Dados

Orientador: Prof. Dr. Moacir Antonelli Ponti

Versão original

São Carlos

2021

AGRADECIMENTOS

Para que eu tivesse esta jornada de um ano de estudo e muito aprendizado, o apoio dos meus familiares, dos professores e dos monitores do MBA em Ciência de Dados foram essenciais. Desta forma agradeço:

- Inicialmente, aos meus pais que sempre me incentivaram na busca do conhecimento.
- A minha querida esposa Valéria, pelo apoio e por compreender a minha ausência por conta das várias horas de estudo.
- Ao meu filho Henrique, fonte da minha força de querer sempre melhorar.
- A cada um dos professores por abordarem temas fundamentais para que possamos trilhar o caminho na área de Ciência de Dados.
- Aos monitores por estarem sempre disponíveis em tirar dúvidas, trazerem uma visão mais prática dos assuntos abordados em aula e apresentarem temas importantes em monitorias extras.
- Ao meu orientador, Professor Moacir, pelas dicas e apontamentos que foram essenciais para a elaboração deste trabalho.

RESUMO

dos Santos, H. P. **Classificação multicategórica de notícias em língua portuguesa utilizando o Processamento de Linguagem Natural**. 2021. 96p. Monografia (MBA em Ciências de Dados) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2021.

O Processamento de Linguagem Natural (**PLN**) tem como objetivo modelar em computadores digitais a linguagem humana. Os modelos *autoencoders* pré-treinados em conjuntos de dados massivos e compostos por textos diversificados têm possibilitado a execução de tarefas de PLN com bom desempenho e com baixo consumo de recurso computacional. O **BERT** é um destes modelos que transformaram o PLN, pois além de atingir o estado-da-arte, a equipe de criação disponibilizou os modelos pré-treinados para *download*. Para a língua portuguesa há o **BERTimbau** que foi disponibilizado na plataforma Hugginface e que possibilita a realização de ajuste fino para execução de tarefas de PLN.

Neste trabalho utilizamos o BERTimbau para treinar um classificador de notícias jornalísticas. Construímos um conjunto de dados de notícias que foi resultado da integração entre uma solução construída por nós para raspagem de textos de alguns dos principais sítios de notícias do Brasil e um conjunto de dados de notícias da Folha de São Paulo disponibilizado na plataforma Kaggle. Utilizamos a sessão onde a notícia foi publicada no sítio como sendo a classe. Treinamos quatro classificadores utilizando tamanho de sequências diferentes: 128, 256, 512 e 394. Com o objetivo de termos uma linha de base para o nosso problema, treinamos dois classificadores utilizando a abordagem *bag of words* e os algoritmos clássicos Multinomial Naive-Bayes e *Random Forest*. Os dois classificadores BERTimbau com maior tamanho de sequência obtiveram os melhores desempenho com valores para acurácia balanceada e precisão acima de 90%.

Palavras-chave: PLN, BERT, BERTimbau

ABSTRACT

dos Santos, H. P. **Model for TCC in LaTeX using the USPSC class to the ICMC**. 2021. 96p. Monografia (MBA em Ciências de Dados) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2021.

Natural Language Processing (**NLP**) aims teaching computers to process the human language. Pre-trained autoencoders in massive datasets and composed of diversified texts have enabled the execution of NLP tasks with good performance and with low computational resource consumption. **BERT** is one of these models that transformed the NLP, as in addition to achieving the state-fo-the-art, the creation team made the pre-trained models available for download. **BERTimbau** is the BERT model for Portuguese language that is available on the Hugginface platform and which enables the realization of fine tuning for the execution of NLP tasks.

In this work, we used BERTimbau to train a journalistic news classifier. We built a news dataset that was the result of the integration between a solution built by us to scrape texts from some of the main news sites in Brazil and a news dataset from Folha de São Paulo made available on the Kaggle platform. We used the section where the news was published on the site as the class. We trained four classifiers using different size sequences: 128, 256, 512 and 394. In order to have a baseline for our problem, we trained two classifiers using the Bag of Word approach and the classical Multinomial Naive-Bayes and Random Forest algorithms. The two BERTimbau classifiers with the largest sequence size had the best performance with values for balanced accuracy and precision above 90%.

Keywords: BERT, BERTimbau, NLP

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ASSIN2	Segunda Avaliação de Similaridade Semântica e Inferência Textual
BERT	Bidirectional Encoder Representation from Transformers
biLM	Deep bidirection Language Model
BLEU	Bilingual Evalution Understudy
CNN	Convolutional Neural Network
CoLA	The Corpus of Linguistic Acceptability
DPCNN	Deep Pyramid Convolutional Neural Network
ELMo	Embeddings from Language Model
EOS	End of Sentence
GLUE	General Language Understanding Evaluation
GPT	Generative Pretrained Transformer
GPU	Graphics Processsing Unit
	HTML HyperText Markup Language
LM	Language Model
LSTM	Long Short-Term Memory
MNLI	Multi-Genre Natural Language Inference
MRPC	Microsoft Research Paraphrase Corpus
MT	Machine Translation
NMT	Neural Machine Translation
PLN	Processamento de Linguagem Natural
QNLI	Question Natural Language Inference
QQP	Quora Question Pairs
RNN	Recurrent Neural Network

	ROC Receiver operating characteristics
RTE	Recognizing Textual Entailment
SST-2	The Stanford Sentiment Treebank
STS-B	The Semantic Textual Similarity Benchmark
tf-idf	Term Frequency– Inverse Document Frequency
TPU	Tensor Processing Unit
ULMFiT	Universal Language Model Fine-tunning

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização e Motivação

As corporações, o governo e as pessoas geram diariamente uma grande quantidade de textos na forma de documentos, notícias, artigos e mensagens[1]. Para entender, classificar e gerar respostas manualmente aos textos produzidos, sem uma ferramenta computacional, gasta-se muito tempo e esforço humano.

O estudo da Linguística em conjunto com a Informática pode contribuir para resolver este problema (SCHIESSL, 2007). A Linguística busca caracterizar e explicar a diversidade de observações linguísticas que nos cerca, sejam em diálogos, seja na escrita, seja em qualquer outro meio. A Linguística computacional é a área que examina as relações entre a Linguística e a Informática e objetiva a construção de sistemas especialistas em reconhecer e produzir informação em linguagem natural. O Processamento de Linguagem Natural (PLN) está incluído no contexto da Linguística Computacional (VIEIRA; LIMA, 2001).

Processamento de Linguagem Natural tem como objetivo modelar em computadores digitais a linguagem humana (RUDER, 2019). Esta área nos últimos anos tem desenvolvido técnicas que possibilitam a criação de soluções que independem da intervenção humana. Algumas destas técnicas são classificação, respostas a perguntas, sumarização, análise de sentimentos e tradução (HOWARD; GUGGER, 2020).

Atualmente, os exemplos de estado-da-arte em Processamento de Linguagem Natural são redes neurais pré-treinadas que podem capturar padrões semânticos a partir de textos em linguagem natural e serem finamente ajustadas para melhorar a execução de várias tarefas em PLN (ZHANG et al., 2019). Exemplos destas redes neurais pré-treinadas são *Bidirectional Enconder Representation from Transformers* (BERT) (DEVLIN et al., 2018), *Universal Language Model Fine-tuning for Text Classification* (ULMFit) (HOWARD; RUDER, 2018) e *Generative Pre-Training* (GPT): GPT-1 de (RADFORD et al., 2018).

Uma das maiores dificuldades em treinar um destes modelos em estado-da-arte é o alto custo computacional que pode chegar a centenas de horas de uso de várias GPUs ou TPUs em paralelo e consequentemente em um alto orçamento com equipamento e eletricidade, sendo que tal obstáculo inviabilizaria a aplicação deles, principalmente, em empresas de pequeno e médio porte[2]. Outro desafio está na modelagem de problemas em linguagens com poucos recursos, como é o caso do Português (CASANOVA et al., 2021). Desta forma, a disponibilização destes modelos pré-treinados possibilitam que eles

[1] https://impacting.digital/processamento-de-linguagem-natural/acessado em 20/04/2021
[2] https://medium.com/ensina-ai/ensinando-português-ao-gpt-2-d4aa4aa29e1d

sejam utilizados através das técnicas de transferência de aprendizado e ajuste fino com bons resultados e com uma quantidade de processamento bem menor que o treinamento inicial (PONTI et al., 2021).

1.2 Objetivos

O objetivo principal deste trabalho é projetar um classificador para categorizar textos jornalísticos publicados em sítios eletrônicos brasileiros de notícias analisando dados anteriormente disponíveis e obtidos durante o trabalho.

Em específico, efetuaremos o ajuste fino no modelo BERTimbau disponibilizado na plataforma Hugginface(WOLF et al., 2019) para que seja possível categorizar as noticias em dez classes, sendo que cada classe é definida conforme a sessão do jornal que foi publicada a notícia. Para que tenhamos uma linha de base para comparação dos resultados obtidos, treinaremos também um classificador utilizando a abordagem de *bag of words* com os classificadores clássicos Multinomial Naives-Bayes e *Random-Forest*.

Dentre os vários modelos pré-treinados disponíveis, e com diferentes abordagens, a escolha pelo BERTimbau se deu, principalmente, por ser um modelo treinado em língua portuguesa utilizando os modelos BERT (DEVLIN et al., 2018) para textos curtos ($BERT_{base}$) e para textos longos ($BERT_{large}$), sendo que eles estão entre os modelos com melhores resultados nas plataformas de avaliação de PLN. Por exemplo, o *General Language Understanding Evaluation* (GLUE) [3]. Outro ponto positivo é o vasto vocabulário em Português com 30.000 unidades de subpalavras. Nas tarefas de PLN em que foi avaliado, ele obteve ótimos resultados. Por fim, ele foi implementado utilizando a biblioteca *Transformers* (WOLF et al., 2019), o que torna mais fácil a codificação e o treinamento do classificador.

No Capítulo 2 descrevemos a revisão bibliográfica contendo alguns dos principais trabalhos recentes em estado-da-arte de PLN . No Capítulo 3 abordamos a construção da base de dados de notícias, a contextualização e características desta base de dados e os tratamentos efetuado. Mostramos ainda as configurações dos experimentos feitos. Finalmente, no Capítulo 5 concluímos e sugerimos caminhos para trabalhos futuros.

[3] https://gluebenchmark.com/, acessado em 21/04/2021

2 REFERÊNCIAS BIBLIOGRÁFICAS

2.1 Critérios de Seleção da Pesquisa

Os três últimos marcos[1] para o processamento de linguagem natural são redes neurais com arquitetura *sequence-to-sequence* como proposto por (JOHNSON; ZHANG, 2017), mecanismos de atenção como proposto por (VASWANI et al., 2017) e modelos de linguagens pré-treinadas como proposto por (LE et al., 2019). Os artigos escolhidos foram e são o estado-da-arte nestas arquiteturas (cada um em seu tempo) e a escolha deles objetiva listar a evolução recente das redes neurais para processamento de linguagem natural. Consideramos como recente os últimos cinco anos.

Os modelos de processamento de linguagem natural possuem milhões de parâmetros, necessitam de grandes conjuntos de dados para serem treinados e demandam muito recurso computacional: (HOWARD; RUDER, 2018), (PETERS et al., 2018), (EISENSCHLOS et al., 2019), (DEVLIN et al., 2018), (RADFORD et al., 2018). Os modelos pré-treinados para execução de tarefas finais de processamento de linguagem natural, como classificação de textos, permitem que o desenvolvimento de soluções para execução destas tarefas seja mais ágil e demande menos recurso, pois partem de um modelo já treinado com o objetivo de realizar ajustes finos para tarefa final[2]. Recursos computacionais e um banco de dados grande de notícias também são limitações do nosso trabalho e isto serviu como parâmetro para buscarmos na literatura resultados estado-da-arte que possibilitem, sem muitos recursos, desenvolvermos um classificador eficaz.

Dentre as diversas publicações de modelos pré-treinados para a língua portuguesa, o BERTimbau de (SOUZA et al., 2020) é um dos mais recentes e disponibiliza o *tokenizador* e o modelo pré-treinado para construção de tarefas finais de processamento de linguagem natural. Desta forma, utilizamos também como parâmetro de pesquisa, os artigos que explicam teoricamente os elementos que formam a arquitetura do BERTimbau.

2.2 Modelos para Processamento de Linguagem Natural

Wu et al (WU et al., 2016) usando *Neural Machine Translation* (NMT) apresentaram a *Google Neural Machine Learning Translation* com o objetivo de deixar mais rápido e mais eficiente o processamento de linguagem natural para a tarefa de tradução.

Para resolver este problema, propuseram uma rede neural com arquitetura *Long*

[1] https://medium.com/@Capeai/natural-language-processing-a-brief-history-7811f0727f44 - acessado em 08-07-2021

[2] https://towardsdatascience.com/major-trends-in-nlp-a-review-of-20-years-of-acl-research-56f5520d473 - acessado em 08-07-2021

Short-Term Memory (LSTM) de (HOCHREITER; SCHMIDHUBER, 1997) com 8 camadas e com conexões residuais entre elas para estimular a descida do gradiente. Para paralelizar, conectaram a atenção com a camada mais baixa do codificador e com a camada mais alta do decodificador. Para melhorar o tempo de inferência, empregaram aritmética de baixa precisão que executa mais rápido quando processada com *Tensor Processing Unit* - TPU. Tratamento de palavras raras em tradução é um problema, para lidar com isto usaram como unidade de entrada e saída sub palavras com o objetivo de obter um bom equilíbrio para decodificação de caracteres únicos e palavras completas, além de contornar a necessidade de tratamento especial de palavras desconhecidas.

O modelo proposto utiliza o *framework* de aprendizado *sequence-to-sequence* com a camada atenção e é composto por uma rede para codificação, uma rede para decodificação e uma rede atenção, conforme Figura 1. O codificador transforma a sentença de origem em uma lista de vetores, um vetor por símbolo de entrada. Dada esta lista de vetores, o decodificador produz um símbolo por vez até que o símbolo que marca o final da sentença, EOS (*end-of-sentence*), é produzido. O codificador e o decodificador são conectados através do módulo atenção que permite o decodificador focar em diferentes regiões da sentença de origem durante o processamento da decodificação.

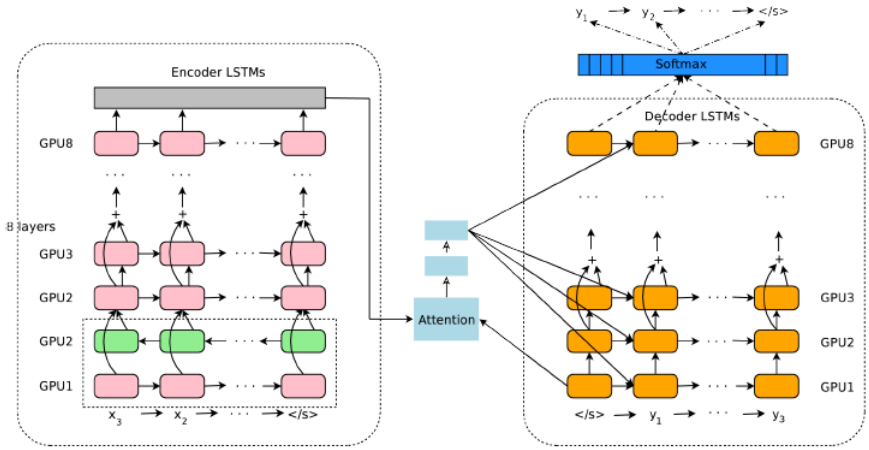
A camada inferior do codificador é bidirecional: os nós rosas coletam informações da esquerda para a direita enquanto os nós verdes coletam informações da direita para a esquerda. As outras camadas do codificador são unidirecionais. Conexões residuais iniciam nas três camadas mais baixas no codificador e decodificador. O modelo é particionado em múltiplas GPU para aumentar a velocidade de treinamento. São apresentadas 8 camadas LSTM (1 camada bi-direcional e 7 camadas uni-direcionais) e 8 camadas no decodificador. Com esta configuração, uma réplica do modelo é particionada em 8 partes e cada uma alocada em 8 GPUs diferentes. Durante o treinamento, as camadas bidirecionais da base do codificador computam em paralelo primeiro. Quando o processamento de todas as camadas finalizarem, as camadas unidirecionais do codificador iniciam o processamento, sendo que cada uma em uma GPU separada. Para manter o maior nível possível de paralelismo durante a execução das camadas do decodificador, a saída da camada mais baixa é usada somente para obter o contexto atenção que é enviado diretamente para todas as camadas do decodificador. A camada *softmax* é também particionada e alocada em multiplas GPUs.

Seja (X, Y) um par de sentenças que é composto por uma sentença de origem e uma sentença alvo. Seja $X = x_1, x_2, x_3, \cdots, x_M$ uma sequência de M símbolos na sentença de origem e $Y = y_1, y_2, y_3, \cdots, y_N$ uma sequência de N símbolos na sentença alvo. O codificador é uma função:

$$x_1, x_2, x_3, \cdots, x_M = EncoderRNN(x_1, x_2, x_3, \cdots, x_M)$$

Nesta equação $x_1, x_2, x_3, \cdots, x_M$ é uma lista de vetores de tamanho fixo. O número de

Figura 1 – Arquitetura do modelo *Google Neural Machine Learning Translation*. A esquerda está o codificador, a direita o decodificador e no meio está o módulo atenção. Nós rosas coletam informações da direita para esquerda e nós verdes da direita para esquerda. O modelo é particionado em 8 GPUs.



Fonte:(WU et al., 2016)

elementos na lista é o mesmo número de símbolos na sentença de origem. Usando a regra da cadeia, a probabilidade condicional da sequência $P(Y|X)$ pode ser decomposta como:

$$P(Y|X) = P(Y|x_1, x_2, x_3, \cdots, x_M) = \prod_{i=1}^{N} P(y_i|y_0, y_1, \cdots, y_{i-1}; x_1, x_2, x_3, \cdots, x_M)$$

onde y_0 é um símbolo especial que marca o início da sentença e é prefixo em toda sentença alvo.

Durante a inferência é calculada a probabilidade do próximo símbolo dada a codificação da sentença de origem e a decodificação da sentença alvo:

$$P(y_i|y_0, y_1, \cdots, y_{i-1}; x_1, x_2, x_3, \cdots, x_M)$$

O decodificador é implementado como uma combinação de uma rede RNN e uma camada *softmax*. O decodificador RNN produz um estado oculto y_i para o próximo símbolo a ser predito que através da camada *softmax* gera a probabilidade de distribuição sobre os símbolos de saída candidatos.

A saída a_i da camada Attention é definida como:

$$s_t = AttentionFunction(y_{i-1}, x_t) \forall t, 1 \leq t \leq M$$

$$p_t = \frac{exp(s_t)}{\sum_{t=1}^{M} exp(s_t)} \forall t, 1 \leq t \leq M$$

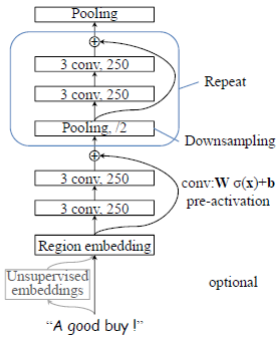
$$a_i = \sum_{t=1}^{M} p_t \times x_t$$

onde y_{i-1} é a saída da última camada de decodificação da rede RNN e $AttentionFunction$ é uma rede *feed forward* com uma camada oculta.

A implementação desta solução foi avaliada em vários conjuntos de dados e submetidas a pontuação BLEU (do inglês Bilingual Evaluation Understudy) (PAPINENI et al., 2002). A pontuação BLEU é um algoritmo de correspondência de sentenças que fornece métricas básicas de qualidade para pesquisadores e desenvolvedores de *Machine Translation* - MT. Para o conjunto de dados WMT'14 Englist-to-French obteve pontuação de 38.95 BLEU que é superior a 7.5 BLEU reportado por (LUONG et al., 2014) e superior a 1.2 BLEU reportado por (ZHOU et al., 2016). Para o conjunto de dados WMT'14 English-to-German obteve pontuação de 24.17 BLEU que é 3.4 BLEU melhor que (BUCK; HEAFIELD; OOYEN, 2014).

Jonhson e Zhang (JOHNSON; ZHANG, 2017) propuseram uma arquitetura de rede neural convolucional profunda (CNN) de baixa complexidade, no nível de palavras, para categorização de textos. Esta rede pode representar de forma eficiente associações existentes em textos longos. Nesta arquitetura a melhor acurácia pode ser obtida aumentando a profundidade e não o tempo de computação que é limitado por uma constante. Esta arquitetura foi nomeada de *Deep Pyramid Convolutional Neural Network* - DPCNN.

Figura 2 – Arquitetura do modelo DPCNN.



Fonte:(JOHNSON; ZHANG, 2017)

A arquitetura DPCNN é ilustrada na Figura 2. A primeira camada *Region embedding* generaliza a *word embedding* comumente usadas para a *embedding* de palavras em regiões de texto que abrangem uma ou mais palavras. Segue-se, então, por um empilhamento de blocos convolucionais, duas camadas de convolução e uma *shortcut* intercalada com camadas de agrupamento com passo 2 para redução da dimensão de entrada. A camada do agrupamento final agrega dados internos para cada documento em um vetor. Os pontos chaves da DPCNN são:

- *Downsampling* (reduz as dimensões espaciais do texto usando operações matemáticas)

sem aumentar o número de mapeamentos das camadas de saída. Este método habilita uma representação eficiente de associações de grande partes do texto. Por manter o mesmo número de mapeamentos, a computação por bloco é reduzida pela metade a cada dois passos e o tempo de computação é limitado por uma constante.

- Conexões de atalho com pré-ativação e mapeamento identidade para habilitar o treinamento em multicamadas.

- *Embedding* de regiões do texto melhorada com *embedding* não supervisionada (*embeddings* são treinadas de forma não supervisionada) para melhorar a acurácia.

Uma rede CNN para categorização de texto se inicia convertendo cada palavra no texto para um vetor de palavras (do inglês *word embedding*). *Embedding* de regiões de texto são vetores de regiões do texto que englobam uma ou mais palavras.

Na camada *Region Embedding* é computada para cada palavra do documento $Wx+b$, onde x representa a $k-word$ região em torno da palavra e o pesos w e o *bias* b são treinados com parâmetros de outras camadas. Seja v o tamanho do vocabulário, os tipos de representação de uma $k-word$ região para x são: entrada sequencial (concatenação kv-dimensional de k vetores *one-hot*), *bow input* (vetor tipo *bag of words* (bow) v-dimensional) e entrada *bag-of-n-gram* (*bag of word* uni, bi e trigrams contidos na região).

A acurácia é substancialmente melhorada quando se estende uma rede ShallowCNN com *embeddings* não supervisionados obtidos por *two views embedding training*, conforme mostrado por (JOHNSON; ZHANG, 2016). Este mesmo método foi aplicado a DPCNN. Para categorização de texto, definiu-se uma região de texto como *view*-1 e suas regiões adjacentes como *view*-2. Então, usando dados sem rótulos, treinou-se uma rede neural de uma camada oculta com uma tarefa artificial de predizer *view*-2 a partir de *view*-1. A camada oculta, que é uma função que recebe *view*-1 como entrada, serve como função não supervisionada no modelo para categorização de texto.

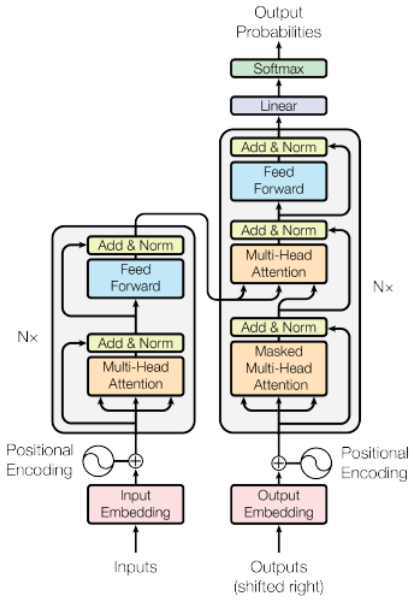
Vaswani et al (VASWANI et al., 2017) apresentaram a arquitetura *Transformer*, que transformam uma sequência de elementos em outra sequência utilizando mecanismos de atenção e evitando a utilização de redes neurais recorrentes.

A arquitetura *Transformer* é composta pelas estruturas Codificador e Decodificador. A Figura 3 apresenta a arquitetura desta rede neural, sendo que o Codificador está a esquerda e o Decodificador a direita. Ambos são compostos por uma pilha de camadas idênticas (representadas por Nx) e estas camadas são compostas por duas sub-camadas: um mecanismo *multi-head self-attention* e uma rede *position-wise fully connected feed-forward*. Todas as camadas e sub-camadas produzem saídas dimensionais d_{model}.

Nas saídas das sub-camadas do Codificador e do Decodificador há uma camada de normalização, isto é, a saída de cada sub-camada é dada por $LayerNorm(x+Sublayer(x))$,

tal que $Sublayer(x)$ é implementada pela própria sub-camada. No Codificador foi adicionada uma terceira sub-camada que executa *multi-head-attention* sobre as saídas da pilha de camadas do Decodificador. Também foi modificado no Decodificador a sub-camada auto-atenção para prevenir que posições sejam mapeadas para posições subsequentes.

Figura 3 – Arquitetura do Transformer.



Fonte:(VASWANI et al., 2017)

Uma função auto-atenção mapeia uma consulta (consulta é a representação vetorial de uma palavra na sequência) a um conjunto de pares chave-valor (chave-valor são representações vetoriais de todas as palavras na sequência) para uma saída, sendo que consulta, chaves e valores são todos vetores. A saída é computada como a soma ponderada dos valores tal que o peso atribuído a cada valor é computado por uma função de compatibilidade da consulta com a chave correspondente.

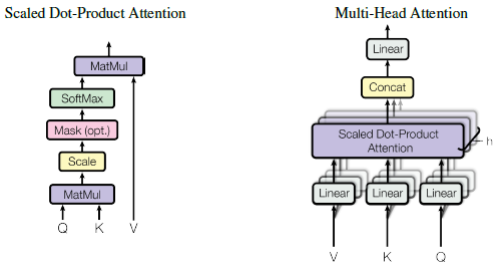
Na Figura 4, o diagrama a esquerda representa a função Atenção utilizada e que é definida como:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

K, Q e V são matrizes, sendo que Q representa valores para consulta com dimensão d_k, K representa valores para as chaves com dimensão d_k e V representa os valores com dimensão d_v.

O diagrama a direita da Figura 4 representa a *Multi-Head Attention*. Para implementá-la, em vez de executar uma simples função atenção com chaves, valores e consultas com dimensões d_{model}, K, V e Q foram projetados linearmente h vezes com diferentes projeções

Figura 4 – (esquerda) *Scaled Dot-Product Attention*. (direita) *Multi-Head Attention* consiste de várias camadas Atenção rodando em paralelo.



Fonte:(VASWANI et al., 2017)

lineares de aprendizado para dimensões d_k e d_v. Em cada uma destas projeções foram executadas a funções atenção em paralelo retornando valores com dimensões d_v. Estes são concatenados para resultar nos valores finais:

$$MultiHead(Q, K, V) = Concat(head_1, \cdots, head_h)W^O \text{ onde}$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

As projeções são matrizes de parâmetros $W_i^Q \in R^{d_{model}.d_k}$,$W_i^K \in R^{d_{model}.d_k}$, $W_i^V \in R^{d_{model}.d_v}$ e $W^O \in R^{hd_v \times d_{model}}$.

Tabela 1 – O *Transformer* atingiu a melhor pontuação BLEU em relação aos modelos estado-da-arte para tradução e reduziu o custo de treinamento.

Modelo	BLEU		Custo Treinamento (FLOPs)	
	ING-ALM	ING-FRA	ING-ALM	ING-FRA
ByteNet	23, 75			
Deep-Att + PosUnk		39, 2		1.0×10^{20}
GNMT + RL	24, 6	39, 92	2.3×10^{19}	1.4×10^{20}
ConvS2S	25, 16	40, 46	9.6×10^{18}	1.5×10^{20}
MoE	26, 03	40, 56	2.0×10^{19}	1.2×10^{20}
Deep-Att + PosUnk Ensemble		40, 4		8.0×10^{20}
GNMT + RL Ensemble	26, 30	41, 16	1.8×10^{20}	1.1×10^{21}
ConvS2S Ensemble	26, 36	41, 29	7.7×10^{19}	1.2×10^{21}
Transformer (base model)	27, 3	38, 1	3.3×10^{18}	
Transformer (big)	28, 4	41, 8	2.3×10^{19}	

Fonte: (VASWANI et al., 2017)

Foram realizados experimentos de tradução utilizando o conjunto de dados WMT 2014 contendo 4,5 milhões de pares de sentenças inglês-alemão e também o WMT 2014

larger que contém 36 milhões de sentenças em inglês-francês. O modelo *Transfomer* atingiu os melhores índices BLEU e com um desempenho de tempo bem mais rápido que os outros modelos, conforme mostrado na Tabela 1.

Peters et al (PETERS et al., 2018) apresentaram um novo tipo de representação para contextualização profunda de palavras (do inglês *deep contextualized word*). Esta representação modela o uso das características sintáticas e semânticas das palavras e também a variação delas de acordo com o contexto linguístico, além de ser facilmente integrada aos modelos de processamento de linguagem natural existentes.

As representações tradicionais até aquele momento para *word embeddings* consideram que cada *token* é atribuído a uma função que leva em conta toda a sentença (frase) de entrada. Esta nova proposta usa vetores derivados de uma LSTM bidirecional ((MA; HOVY, 2016), (MELAMUD; GOLDBERGER; DAGAN, 2016)) treinada em um modelo objetivo de linguagem associado e usando um grande *corpus*. Assim, ela foi batizada de ELMo (do inglês *Embeddings from Language Models*).

Elmo é considerada profunda, pois são funções de todas as camadas internas de um biLM (do inglês *Deep bidirection Language Model*). biLM é uma combinação linear de vetores empilhados acima de cada entrada de palavra para o fim de cada tarefa que melhora acentuadamente o desempenho em relação à simples utilização da camada superior do LSTM.

biLM é formalmente definido como: dada uma sequência de N *tokens*, $t_1, \cdots, t_N$, um modelo de linguagem no sentido de avançar (do inglês *forward model language*) computa a probabilidade da sequência ser modelada pela probabilidade de *tokens* t_k, dado o histórico $(t_1, \cdots, t_{k-1})$:

$$p(t_1, \cdots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, \cdots, t_{k-1})$$

Para predizer o próximo *token* t_{k+1}, é computada uma representação x_k^{LM} de *tokens* em um contexto independente — através de *token embedding* ou carácteres sobre uma CNN — que é passada adiante através de camadas LSTM, sendo que em cada posição k há uma saída de representação de contexto independente, $\overrightarrow{h}_{kj}^{LM}$ em cada camada LSTM, onde $j = 1, \cdots, L$. A saída do topo da camada LSTM, $\overrightarrow{h}_{kj}^{LM}$, é usada para predizer o próximo token t_{k+1} com uma camada Softmax.

O modelo de linguagem no sentido de voltar (do inglês *backward model language*) é similar ao *forward model language*, exceto que ele roda no sentido reverso predizendo o *token* prévio dado o contexto futuro:

$$p(t_1, \cdots, t_N) = \prod_{k=1}^{N} p(t_k | tk + 1, tk + 2, \cdots, t_N)$$

Similarmente ao *forward model language*, cada camada j *backward LSTM* em um modelo multicamadas L produz representações $\overleftarrow{h}_{kj}^{LM}$ de t_k, dado $(tk+1, tk+2, \cdots, t_N)$.

biLM combina *forward* e *backward* LM (do inglês *Language Model*). Peters et al maximizaram o log da verossimilhança das direções *forward* e *backward*:

$$\sum_{k=1} N(logp(t_k|t_1, \cdots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + logp(t_k|t_{k+1}, \cdots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

Os parâmetros para representações dos *tokens* (Θ_x) e da camada Softmax (Θ_s) em ambas as direções foram mantidos fixos e os parâmetros para as LSTMs em cada direção foram mantidos separados.

Elmo então é definido como uma combinação de tarefas específicas das camadas intermediárias representadas na biLM. Para cada *token* t_k, uma camada L computa um conjunto de $2L+1$ representações:

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, \cdots, L\} = \{h_{k,j}^{LM} | j = 0, \cdots, L\}$$

onde $h_{k,0}^{LM}$ é o *token* da camada e $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM}]$ para cada camada biLSTM.

Para as tarefas de PLN, ELMo unifica todas as camadas em R para um único vetor: $ELMo_k = E(R_k; \Theta_e)$. No caso mais simples, ELMo obtém a saída do topo da camada: $E(R_K) = h_{k,j}^{LM}$. De forma mais geral, foi calculada uma ponderação específica de tarefa de todas as camadas biLM:

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

onde s^{task} são pesos softmax-normalizada e o parâmetro escalar γ^{task} permite que o modelo de tarefa escale todo o vetor ELMo.

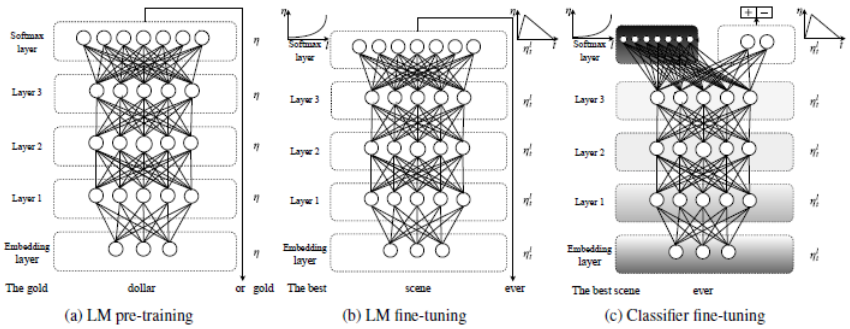
Foram efetuados experimentos em tarefas de processamento de linguagem natural. Em tarefas que foram trabalhadas somente as representações ELMo, houve melhora no estado-da-arte da época obtendo redução em até 20% nos erros relativos. Para tarefas onde foi possível obter comparações, ELMo superou CoVe (MCCANN et al., 2017).

Howard e Ruder (HOWARD; RUDER, 2018) propuseram um modelo de linguagem que utiliza um método de transferência de aprendizado indutiva para qualquer tarefa de PLN: ULMFiT (do inglês *Universal Language Model Fine-tunning*). Este modelo passa por um ajuste fino discriminativo usando taxas de aprendizagem triangulares inclinadas e graduais. Também é apresentada a técnica de descongelamento que objetiva manter o conhecimento e evitar o esquecimento catastrófico durante a afinação.

O ULMFiT pré-treina um modelo de linguagem em um corpus de domínio geral e então são feitos ajustes finos para a tarefa alvo de processamento de linguagem natural

utilizando técnicas de redes neurais. Howard e Ruder utilizaram o modelo AWD-LSTM para o pré-treinamento através do método transferência de aprendizado. Por fim, o modelo é ajustado para a tarefa de classificação de texto. A Figura 5 ilustra os passos que compõem o método.

Figura 5 – ULMFiT consiste de três estágios: pré-treinamento do modelo de linguagem, depois este modelo passa por ajustes finos utilizando os dados específicos da tarefa alvo e usando a técnica ajuste fino discriminativo e taxas de aprendizado triangular para aprender características específicas da tarefa. Por fim, o classificador passa por ajustes finos usando a técnica descongelamento gradual (cor cinza significa que o estágio está descongelado e preto o estágio está congelado)



Fonte:(HOWARD; RUDER, 2018)

O método proposto foi avaliado para tarefas de classificação de textos: análise de sentimento, classificação de questões e classificação de tópicos. Foram usados conjuntos de dados com quantidades variadas de documentos e que foram utilizados em abordagens de classificação de textos e transferência de conhecimento por (JOHNSON; ZHANG, 2017) e (MCCANN et al., 2017).

Tabela 2 – Taxas de erro (%) nos testes efetuados nos conjuntos de dados usados por (MCCANN et al., 2017)

IMDb		TREC-6	
Model	Test	Model	Test
Cove	8, 2	Cove	4, 2
oh-LSTM	5, 9	TBCNN	4, 0
Virtual	5, 9	LSTM-CNN	3, 9
ULMFiT	4, 6	ULMFiT	3, 6

Fonte:(HOWARD; RUDER, 2018)

Na Tabela 2 são apresentadas as taxas de erro dos testes para os conjuntos de dados IMDb e TREC-6 usados por (MCCANN et al., 2017). O ULMFiT obteve a menor taxa de

Tabela 3 – Taxas de erro (%) nos testes efetuados nos conjuntos de dados usados por (JOHNSON; ZHANG, 2017)

	AG	DBpedia	Yelp-bi	Yelp-full
Char-level CNN	9,51	1,55	4,88	37,95
CNN	6,57	0,84	2,90	32,39
DPCNN	6,87	0,88	2,64	30,58
ULMFiT	5,01	*O*,80	2,16	29,98

Fonte:(HOWARD; RUDER, 2018)

erro quando comparado com outros modelos que eram considerados o estado-da-arte no período: CoVe de (MCCANN et al., 2017), oh-LSTM de (JOHNSON; ZHANG, 2016) e Virtual de (MIYATO; DAI; GOODFELLOW, 2016).

Na Tabela 3 são apresentadas as taxas de erro dos testes para os conjuntos de dados utilizados por (JOHNSON; ZHANG, 2017). O ULMFit obteve a menor taxa de erros quando comparado com resultados obtidos pelos modelos *Char-level* CNN de (MCCANN et al., 2017), CNN de (JOHNSON; ZHANG, 2016) e DPCNN de (JOHNSON; ZHANG, 2017). .

Radford et al (RADFORD et al., 2018) exploraram uma abordagem semi-supervisionada para tarefas de modelos de linguagem usando uma combinação de pré-treinamento não supervisionadas e ajuste fino supervisionado. Desta forma, é possível aprender uma representação universal que pode ser transferida com pequenas adaptações para uma gama de tarefas.

Para a arquitetura foi usado Transformer (VASWANI et al., 2017) que fornece uma memória mais estruturada para lidar com dependências de longo prazo em texto quando comparadas com alternativas como rede recorrentes.

A abordagem consiste de dois estágios: pré-treinamento não supervisionado, aplica-se os dados sem rótulos em um modelo de linguagem para aprender os parâmetros iniciais de um modelo de rede neural. Em seguida este parâmetros são otimizados para uma tarefa de processamento de linguagem natural usando um modelo supervisionado, é o ajuste fino supervisionado.

O pré-treinamento não supervisionado é definido da seguinte forma, dado um conjunto de *tokens* de um *corpus* não supervisionado: $\mathcal{U} = u_1, \cdots, u_n$, utiliza-se um modelo de linguagem padrão para maximizar a verossimilhança:

$$\mathcal{L}_1(\mathcal{U}) = \sum_i logP(u_i|u_{i-k}, \cdots, u_{i-1}; \Theta)$$

onde k é o tamanho da janela contexto e P é a probabilidade condicional modelada usando Θ. Estes parâmetros são treinados usando a descida do gradiente estocástica.

Para o modelo de linguagem, foi utilizada uma variante do *Transformer: multi-layer decoder* de (LIU et al., 2018). Este modelo aplica operações *multi-headed self-attention* sobre os vetores de entrada seguidas por camadas *feedforward* para produzir uma distribuição de saída sobre o *token* alvo:

$$h_0 = UW_e + Wp$$
$$h_l = transformer\_block(h_{l-1} \forall i \in [1, n]$$
$$P(u) = softmax(h_n W_e^T)$$

onde $U = (u\_k, \cdots, u\_1)$ é o vetor de *tokens*, n é o número de camadas, W_e é a matriz de *tokens embedding* e W_p é a posição da matriz *embedding*.

No estágio supervisionado foram otimizados os parâmetros para a tarefa alvo supervisionada. Esta operação é definida como a seguir.

Seja um conjunto de dados etiquetados $\mathcal{C}$ onde cada instância consiste uma sequência de tokens de entrada $x^1, \cdots, x^m$ correspondentes a uma etiqueta y. As entradas são passadas através do modelo pré-treinado para obter os blocos de ativação *Transformer* h_l^m que é então enviado para uma camada de saída linear com parâmetros W_y para predizer y:

$$P(y|x^1, \cdots, x^m) = softmax(h_l^m W_y)$$

maximizando:

$$\mathcal{L}_2(\mathcal{C}) = \sum_{x,y} logP(y|x^1, \cdots, x^m)$$

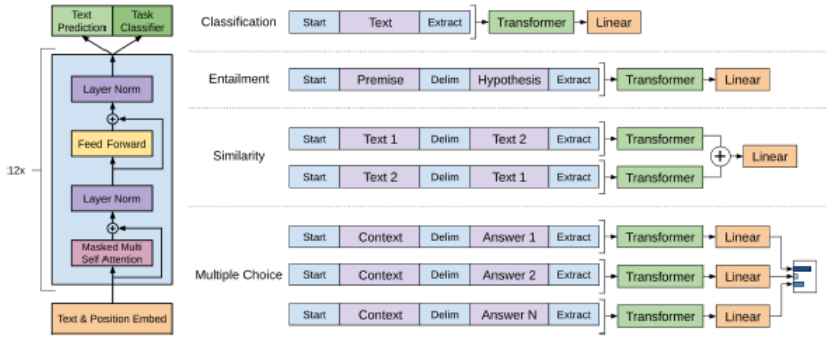
Com o objetivo de melhorar a generalização do modelo supervisionado e acelerar a convergência, foi adicionado um termo auxiliar $\mathcal{L}_1(\mathcal{C})$ com peso λ:

$$\mathcal{L}_3(\mathcal{C}) = \mathcal{L}_2(\mathcal{C}) + \lambda * \mathcal{L}_1(\mathcal{C})$$

A Figura 6 mostra a esquerda a arquitetura do modelo *Transformer* que foi utilizado. A direita estão listadas as transformações aplicadas nos *tokens* antes de submeter ao *Transformer* em cada uma das tarefas de processamento de linguagem natural que foram abordadas. Para a classificação de textos é possível submeter diretamente o *token* das sentenças ao modelo para ajuste fino, para as outras é necessário tratar as particularidades de cada tarefa ao preparar o *token* das sentenças.

Nos experimentos foram abordadas as seguintes tarefas: inferência em linguagem natural, resposta de perguntas, similaridade de sentenças e classificação. A Tabela 4 lista os conjuntos de dados utilizados para cada tarefa.

Figura 6 – (esquerda) Arquitetura do *Transformer* e treinamento objetivo utilizado (direita) Transformações para ajuste fino em diferentes tarefas.



Fonte:(VASWANI et al., 2017)

Tabela 4 – Lista de diferentes tarefas e conjuntos de dados usados nos experimentos por (RADFORD et al., 2018)

Tarefa	Conjunto de Dados
Inferência de linguagem natural	SNLI, MultiNLI, Question NLI, RTE, SciTail
Resposta de perguntas	RACE, Story Cloze
Similaridade de Sentenças	MSR Paraphrase Corpus, Quora Question Pairs, STS Benchmark
Classificação	Stanford Sentiment Treebank-2

Fonte: (RADFORD et al., 2018)

Para a tarefa de inferência de linguagem natural, os resultados do experimento foram comparados a outras abordagens consideradas estado-da-arte e estão listados na Tabela 5. O modelo *Finetuned Transformers* obteve resultados superiores em quatro dos cinco conjunto de dados. Os resultados usados para comparação foram dos modelos ESIM + ELMo de (PETERS et al., 2018), CAFE de (TAY; TUAN; HUI, 2017), Stochastic Answer Network de (LIU; DUH; GAO, 2018), GenSen de (WANG et al., 2018) e Multi-task BiLSTM + Attn de (WANG et al., 2018).

Para a tarefa resposta de perguntas, os resultados dos experimentos foram comparados a outras abordagens consideradas estado-da-arte (val-LS-skip de (SRINIVASAN; ARORA; RIEDL, 2018), Hidden Coherence Model de (CHATURVEDI; PENG; ROTH, 2017), Dynamic Fusion Net de (XU et al., 2017) e BiAttention MRU de (TAY; TUAN; HUI, 2018)) e estão listados na Tabela 6. O modelo *Finetuned Transformer* obteve resultado superior em todos os conjuntos de dados.

Para a tarefa similaridade de sentenças o resultado é apresentado na Tabela 7.

Tabela 5 – Resultados do *Finetuned Transformer* para a tarefa de inferência de linguagem natural comparados aos métodos estado-da-arte a época. Todos usaram acurácia como método de avaliação. 5x indica um conjunto de 5 modelos.

Método	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo (5x)	-	-	89,3	-	-	-
CAFE (5x)	80,2	79,0	89,3	-	-	-
Stochastic Answer Network (3x)	80,6	80,1	-	-	-	-
CAFE	78,7	77,9	88,5	83,3	-	-
GenSen	71,4	71,3	-	-	82,3	59,2
Multi-task BiLSTM + Attn	72,2	72,1	-	-	82,1	61,7
Finetuned Transformer LM	82,1	81,4	89,9	88,3	88,1	56,0

Fonte: (RADFORD et al., 2018)

Tabela 6 – Resultados do Finetuned Transformer para a tarefa de resposta de perguntas comparados aos métodos estado-da-arte a época. 9x indica um conjunto de 9 modelos.

Método	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip	76.5	-	-	-
Hidden Coherence Model	77.6	-	-	-
Dynamic Fusion Net (9x)	-	55.6	49.4	51.2
BiAttention MRU (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM	86.5	62.9	57.4	59.0

Fonte: (RADFORD et al., 2018)

Os experimentos foram feitos para classificação, similaridade semântica e todas foram avaliadas através do GLUE. Para similaridade semântica foi obtidos valores superiores em dois dos conjuntos de dados avaliados. Os modelos utilizados foram Sparse byte mLSTM de (GRAY; RADFORD; KINGMA, 2017), TF-KLD de (JI; EISENSTEIN, 2013), ECNU (mixed ensemble) de (TIAN et al., 2017) e modelos de BiLSTM + ELMo + Attn de (WANG et al., 2018).

Devlin et al (DEVLIN et al., 2018) apresentaram o BERT (do inglês *Bidirectional Encoder Representation from Transformers*). BERT é uma rede neural profunda pré-treinada com representações bidirecionais de textos não rotulados. Estes modelos BERT pré-treinados podem ser finamente otimizados para executar tarefas de processamento de linguagem natural, como por exemplo, responder questões, efetuar inferência de linguagem e classificar textos.

Para implementar um modelo BERT há dois passos: pré-treinamento e ajuste fino.

Tabela 7 – Resultados para classificação e similaridade semântica comparados com modelos estados-da-arte a época. Todas as avaliações foram feitas usando o GLUE. (mc= Mathews correlation, acc=Accuracy, pc=Pearson correlation)

Método	Classificação		Similaridade Semântica			GLUE
	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	
Sparse byte mLSTM	-	93,2	-	-	-	-
TF-KLD	-	-	86,0	-	-	-
ECNU (mixed ensemble)	-	-	-	81,0	-	-
Single-task BiLSTM+ELMo+Attn	35,0	90,2	80,2	55,5	66,1	64,8
Multi-task BiLSTM+ELMo+Attn	18,9	91,6	83,5	72,8	63,3	68,9
Finetuned Transformer LM	45,4	91,3	82,3	82,0	70,3	72,8

Fonte: (RADFORD et al., 2018)

Na primeira parte, o modelo é treinado em dados não rotulados em diferentes tarefas de pré-treinamento. Nos ajustes finos, primeiro o modelo BERT é inicializado com os parâmetros pré-treinados e então todos os parâmetros são finamente ajustados usando dados rotulados para execução de tarefas de linguagem natural. Para cada tarefa de PLN é treinado um modelo com ajustes finos específicos.

A arquitetura do modelo BERT é um codificador Transformer (VASWANI et al., 2017) de multicamadas bidirecional. Foram implementados dois modelos: BERT Base (layes=12, hidde size = 768, self-attention heads=12, total parameters-110M) e BERT Large (layes=24, hidde size = 1024, self-attention heads=16, total parameters-340M). O objetivo de desenvolver o BERT Base foi para poder comparar os resultados com outros modelos de mesmo tamanho como o $GPTTransformers$ (OpenAI GPT) de (RADFORD et al., 2018).

Para avaliação, os modelos foram submetidos as tarefas de processamento de linguagem natural do GLUE (do inglês *General Language Understanding Evaluation*). GLUE é uma ferramenta para avaliar o desempenho de modelos em tarefas de compreensão de linguagem natural. Os modelos são avaliados com base na precisão média em todas as tarefas de cada ferramenta.

O modelo foi avaliado para as seguintes tarefas GLUE:

- MNLI (do inglês *Multi-Genre Natural Language Inference*): dado um par de sentenças, a meta é predizer se a segunda sentença é uma implicação, uma contradição ou um termo neutro em relação à primeira sentença.
- QQP (do inglês *Quora Question Pairs*): tarefa de classificação binária cujo objetivo é determinar se duas questões são semanticamente equivalentes.

- QNLI (do inglês *Question Natural Language Inference*): tarefa de classificação binária para resposta de questões. Dado um par de sentenças, formado por uma questão e por uma resposta, a tarefa responde de forma positiva quando o par contém a resposta correta ou responde de forma negativa quando o par não contém a resposta correta.
- SST-2 (do inglês *The Stanford Sentiment Treebank*): tarefa de classificação binária de sentenças simples a respeito de sentimentos sobre filmes.
- CoLA (do inglês *The Corpus of Linguistic Acceptability*): tarefa de classificação binária cuja meta é predizer se uma sentença na língua inglesa é aceitável ou não.
- STS-B (do inglês *The Semantic Textual Similarity Benchmark*): é um conjunto de pares de sentenças que possuem notas de 1 a 5 que representam quão similar as duas sentenças são em termos semânticos.
- MRPC (do inglês *Microsoft Research Paraphrase Corpus*): são pares de sentenças extraídas de sítios de notícias cujo objetivo é determinar se eles são equivalentes semanticamente.
- RTE (do inglês *Recognizing Textual Entailment*): é um tarefa para determinar se uma sentença está vinculada a outra, semelhante ao MNLI, mas com um muito menos dados de treinamento.

Tabela 8 – Apresentação dos resultado da avaliação dos modelos $BERT_{BASE}$ e $BERT_{LARGE}$ aplicados as tarefas de processamento de linguagem natural MNLI-(m/mm), QQP, QNLI, SST-2, CoLA, STS-B, MRPC, RTE e comparados com os modelos Pre-OpenAI SOTA, BiLSTM+ELMo+Attn e OpenAI GPT

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	$392k$	$363k$	$108k$	$67k$	$8,5k$	$5,7k$	$3,5k$	$2,5k$	-
Pre-OpenAI SOTA	80,6/80,1	66,1	82,3	93,2	35,0	81,0	86,0	61,7	74,0
BiLSTM+ELMo+Attn	76,4/76,1	64,8	79,8	90,4	36,0	73,3v	84,9	56,8	71,0
OpenAI GPT	82,1/81,4	70,3	87,4	91,3	45,4	80,0	82,3	56,0	75,1
$BERT_{BASE}$	84,6/83,4	71,2	90,5	93,5	52,1	85,8	88,9	66,4	79,6
$BERT_{LARGE}$	86,7/85,9	72,1	92,7	94,9	60,5	86,5	89,3	70,1	82,1

Fonte:(DEVLIN et al., 2018)

Os resultados da avaliação dos modelos BERT aplicados ao GLUE foram comparados com a avaliação dos modelos Pre-OpenAI SOTA, BiLSTM+ELMo+Attn de (PETERS

et al., 2018) e OpenAI GPT de (RADFORD et al., 2018). A Tabela 8 mostra os resultados obtidos por todos os modelos em cada tarefa. O BERT$_{BASE}$ e o BERT$_{LARGE}$ obtiveram resultados superiores ao estado da arte em 4.5% e 7.5% nas respectivas médias de acurácia. BERT$_{LARGE}$ superou o BERT$_{BASE}$ em todas as tarefas, especialmente as que possuem poucos dados para treinamento. A coluna *Average* lista a média das métricas de todas as bases de dados avaliadas.

Wolf et al [(WOLF et al., 2019)], baseados na arquitetura *Transformers* de (VASWANI et al., 2017), criaram biblioteca *Transformers* dedicada a suportar e facilitar a distribuição de modelos pré-treinados com o objetivo de facilitar o processo de desenvolvimento das empresas que implementam soluções utilizando os modelos pré-treinados mais populares. A biblioteca suporta a distribuição e o uso de uma variedade de modelos pré-treinados em um *hub* centralizado que permite aos usuários comparar diferentes modelos e experimentar modelos compartilhados que executam diferentes tarefas.

Transformers é mantida pela plataforma *Hugging Face* [3] e é distribuída sob a licença Apache 2.0 e está disponível no GitHub [4].

Transformers é pensada no modelo padrão de pipeline para processamento de linguagem natural: processar os dados, treinar modelo e fazer predições. Todo modelo nesta biblioteca é totalmente definido em três blocos de construção:

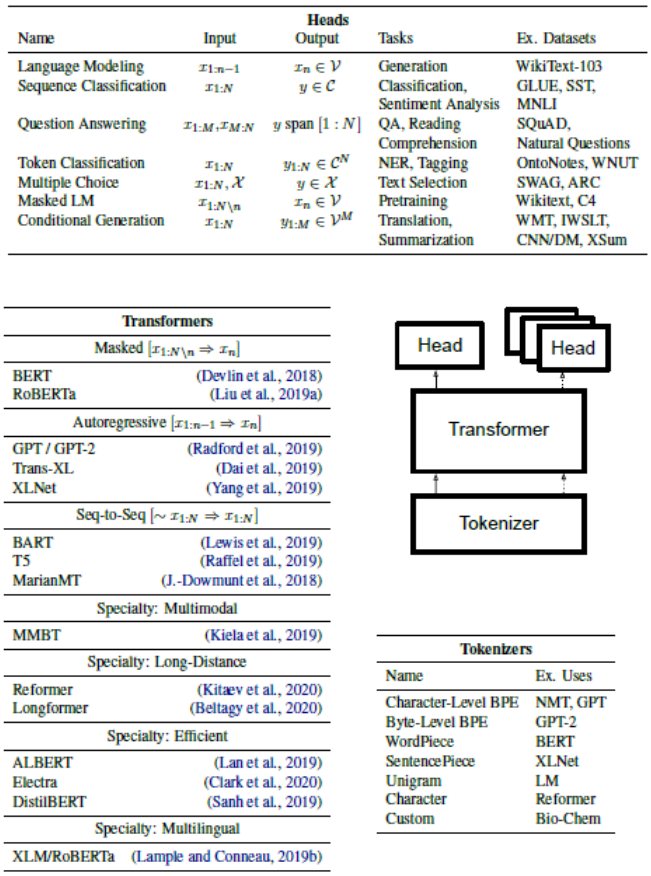
- *tokenizer*: converte linhas de texto em codificações de índices esparsos.
- *transformer*: transforma índices esparsos em *embeddings* contextuais.
- *head*: usa *embedding* contextuais para predizer tarefas específicas de PLN.

A Figura 7 detalha estes blocos. A direita é mostrada a interação entre eles. Um vocabulário amplo e fixo pode ser *tokenizado*, transformado em *embeddings* contextuais para pré-treinar um modelo que posteriormente pode ser utilizado como base para treinar outros modelos especialistas em realizar tarefas de PLN que utilizarão vocabulários específicos ao domínio do problema.

Cada modelo pré-treinado utiliza *tokenizadores* específicos. Eles possuem semelhanças, mas são específicos para cada modelo. O diagrama inferior lista os tipos de *tokenizadores* e os modelos que os utilizam. O diagrama a esquerda lista os modelos especialistas em cada tarefa de PLN. Heads permitem que os *Transformers* podem ser utilizados para executarem diferentes tarefas. No topo são listadas estas tarefas e assumido que dada uma sequência de *tokens* $x_{1:N}$ pertencentes a um vocabulário V e y representa diferentes possíveis saídas que possivelmente pertencerá a uma classe C.

[3] https://huggingface.co/transformers/
[4] https://github.com/huggingface/ transformers

Figura 7 – Estrutura da biblioteca Transformers: Tokenizer, Transformer e Head.



Heads

Name	Input	Output	Tasks	Ex. Datasets
Language Modeling	$x_{1:n-1}$	$x_n \in \mathcal{V}$	Generation	WikiText-103
Sequence Classification	$x_{1:N}$	$y \in \mathcal{C}$	Classification, Sentiment Analysis	GLUE, SST, MNLI
Question Answering	$x_{1:M}, x_{M:N}$	y span $[1:N]$	QA, Reading Comprehension	SQuAD, Natural Questions
Token Classification	$x_{1:N}$	$y_{1:N} \in \mathcal{C}^N$	NER, Tagging	OntoNotes, WNUT
Multiple Choice	$x_{1:N}, \mathcal{X}$	$y \in \mathcal{X}$	Text Selection	SWAG, ARC
Masked LM	$x_{1:N \setminus n}$	$x_n \in \mathcal{V}$	Pretraining	Wikitext, C4
Conditional Generation	$x_{1:N}$	$y_{1:M} \in \mathcal{V}^M$	Translation, Summarization	WMT, IWSLT, CNN/DM, XSum

Transformers

Masked $[x_{1:N \setminus n} \Rightarrow x_n]$	
BERT	(Devlin et al., 2018)
RoBERTa	(Liu et al., 2019a)
Autoregressive $[x_{1:n-1} \Rightarrow x_n]$	
GPT / GPT-2	(Radford et al., 2019)
Trans-XL	(Dai et al., 2019)
XLNet	(Yang et al., 2019)
Seq-to-Seq $[\sim x_{1:N} \Rightarrow x_{1:N}]$	
BART	(Lewis et al., 2019)
T5	(Raffel et al., 2019)
MarianMT	(J.-Dowmunt et al., 2018)
Specialty: Multimodal	
MMBT	(Kiela et al., 2019)
Specialty: Long-Distance	
Reformer	(Kitaev et al., 2020)
Longformer	(Beltagy et al., 2020)
Specialty: Efficient	
ALBERT	(Lan et al., 2019)
Electra	(Clark et al., 2020)
DistilBERT	(Sanh et al., 2019)
Specialty: Multilingual	
XLM/RoBERTa	(Lample and Conneau, 2019b)

Tokenizers

Name	Ex. Uses
Character-Level BPE	NMT, GPT
Byte-Level BPE	GPT-2
WordPiece	BERT
SentencePiece	XLNet
Unigram	LM
Character	Reformer
Custom	Bio-Chem

Fonte:(WOLF et al., 2019)

Le et al (LE et al., 2019) desenvolveram o FlauBERT, modelo pré-treinado BERT para a língua francesa. Foram usados os seguintes conjuntos de dados com vocabulário francês: WMT19; corpus de textos em francês disponíveis na OPUS collection e base de dados disponíveis no Wikimedia. Estes conjuntos de dados foram pré-processados para retirar sentenças pequenas e repetitivas e expressões sem significado como telefone, e-mail e endereços. O tamanho do conjunto de dados para treinamento foi de $43,4GB$.

Foram pré-treinados dois modelos: Base (12 *layers*, 768 *Hidden dimensions* e 12 *attention heads*) e Large (*24 layers*, 1024 *Hidden dimensions* e 16 *attention heads*).

O FLUE é uma ferramenta para avaliação de tarefas em linguagem natural para a língua francesa, equivalente ao GLUE. O FlauBERT foi finamente ajustado e avaliado para as seguintes tarefas: Classificação de Texto, Parafrasear e Inferência de Linguagem Natural.

Para realizar o ajuste fino para a tarefa de classificação de texto, foi utilizada um conjunto de dados com textos de avaliações em francês dos clientes da Amazon

paras os produtos livros (do inglês *books*), DVD e música (do inglês *music*). As acurácias das avaliações são mostradas na Tabela 9. O resultado é comparado com os modelos MultiFit (EISENSCHLOS et al., 2019), mBERT (multiLingual BERT (DEVLIN et al., 2018) e CamemBERT (MARTIN et al., 2019) cujas acurácias foram apresentadas em (EISENSCHLOS et al., 2019).

Tabela 9 – Apresentação dos resultado da avaliação dos modelos $FlauBERT_{BASE}$ e $FlauBERT_{LARGE}$ aplicados à tarefa de classificação de textos na língua francesa e comparados com os modelos MultiFit, mBERT e CamemBERT.

Modelo	Books	DVD	Music
MultiFit	91, 25	89, 55	93, 40
mBERT	86, 15	86, 90	86, 65
CamemBERT	92, 30	93, 0	94, 85
FlauBERT_{BASE}	93, 10	92, 45	94, 10
FlauBERT_{LARGE}	95, 00	94, 10	95, 85

Fonte: (LE et al., 2019)

Para a língua francesa, tanto o FlauBERT quanto o CamemBERT obtiveram acurácia superior ao mBERT. $FlauBERT_{BASE}$ teve acurácia superior ao CamemBERT somente para a categoria de produtos livros e foi superado por CamemBERT nas outras categorias. $FlauBERT_{LARGE}$ foi melhor em todas as categorias.

Para a atividade de parafrasear, foi utilizado o conjunto de dados PAWS-X (Paraphrase Adversaries fromWord Scrambling) em francês que é composto de pares de sentenças parafraseadas, extraídas dos sítios Wikipedia e Quora e foram julgadas por humanos. Parafrasear significa identificar quando duas sentenças são semanticamente equivalentes ou não. A Tabela 10 contém a acurácia das avaliações comparadas com os modelos ESIM((CHEN et al., 2016)), mBERT e CamemBERT. Os resultados destes modelos foram reportados por (YANG et al., 2019).

Para a língua francesa, o modelo CamemBERT obteve melhores resultados que o FlauBERT. O FlauBERT teve resultados muito próximos com o mBERT.

Para inferência de linguagem natural o resultado da avaliação foi comparado com os modelos $XLM - R_{LARGE}$ e $XLM - R_{BASE}$ (CONNEAU; LAMPLE, 2019), mBERT e CamemBERT.

A Tabela 11 apresenta a acurácia do FlauBERT e dos modelos que são usados na comparação. Os modelos franceses FlauBERT e CamemBERT obtiveram acurácia superior ao mBERT.

Souza et al (SOUZA et al., 2020) implementaram o BERTimbau, modelo pré-

Tabela 10 – Apresentação dos resultado da avaliação dos modelos $FlauBERT_{BASE}$ e $FlauBERT_{LARGE}$ aplicados a tarefa de parafrasear textos na língua francesa e comparados com os modelos ESIM, mBERT e CamemBERT.

Modelo	Acurácia
ESIM	66, 20
mBERT	89, 30
CamemBERT	90, 14
FlauBERT$_{BASE}$	89, 49
FlauBERT$_{LARGE}$	89, 34

Fonte: (LE et al., 2019)

Tabela 11 – Apresentação dos resultado da avaliação dos modelos $FlauBERT_{BASE}$ e $FlauBERT_{LARGE}$ aplicados a tarefa de inferência de textos na língua francesa e comparados com os modelos $XLM - R_{LARGE}$ e $XLM - R_{BASE}$, mBERT e CamemBERT

Modelo	Acurácia
$XLM - R_{LARGE}$	85, 2
$XLM - R_{BASE}$	80, 1
mBERT	76, 9
CamemBERT	81, 2
FlauBERT$_{BASE}$	80, 6
FlauBERT$_{LARGE}$	83, 4

Fonte: (LE et al., 2019)

treinado BERT para a língua portuguesa. Eles oferecerem um modelo com arquitetura BERT pré-treinado e que possibilita que os estados internos aprendidos previamente possam ser finamente otimizados para o aprendizado de novas tarefas finais de processamento na língua portuguesa.

Foram treinados dois modelos: Base (12 *layers*, 768 *hidden dimension*, 12 *attention heads* e 110M *parameters*) e Large (*24 layers*, 1024 *hidden dimension*, 16 *attention heads* and 330M *parameters*). Para o pré-treino foi usado o corpus brWaC (do inglês *Brazilian Web as Corpus*) que contém 2.68 bilhões de *tokens* extraídos de 3.53 milhões de documentos. Foram removidos *mojibakes* e *tags* HTML. O *corpus* processado tem 17.5GB de linhas de texto.

Os modelos pré-treinados foram avaliados nas seguintes atividades de processamento de linguagem natural: *Sentence Textual Similarity* (STS), *Recognizing Textual Entailment* (RTE), e *Named Entity* (NER).

Para o ajuste fino das atividades STS e RTE foi utilizado o conjunto de dados

ASSIN2. Os resultados BERTimbau foram comparados com os trabalhos que obtiveram melhores resultados na competição ASSIN2: mBERT + RoBERTa-Large-en (Averaging), mBERT + RoBERTa-Large-en (Stacking), mBERT (STS) and mBERT-PT (RTE) , USE+Features (STS) and mBERT+Features (RTE) e mBERT+Features e mBert que foi finamente ajustados por (SOUZA et al., 2020).

Tabela 12 – Apresentação dos resultado da avaliação dos modelos $BERTimbau_{Base}$ e $BERTimbau_{Large}$ aplicados às tarefas STS e RTE. Os resultados foram comparados aos modelos com abordagem BERT.

Modelo	STS		RTE	
	Pearson	MSE	F1	Acurácia
mBERT + RoBERTa-Large-en (Averaging)	0,83	0,91	84	84,8
mBERT + RoBERTa-Large-en (Stacking)	0,785	0,59	88,3	88,3
mBERT (STS) and mBERT-PT (RTE)	0,826	0,52	87,6	87,6
USE+Features (STS) and mBERT+Features (RTE)	0,800	0,39	86,6	86,6
mBERT+Features	0,817	0,47	86,6	86,6
mBERT (Souza)	0,809	0,58	86,8	86,8
$BERTimbau_{Base}$	0,836	0,58	89,2	89,2
$BERTimbau_{Large}$	0,852	0,50	90,0	90,0

Fonte: (SOUZA et al., 2020)

A Tabela 12 mostra os resultados obtidos. Os modelos BERTimbau obtiveram resultados superiores para a correlação de Pearson. Também obtiveram resultados superiores na tarefa STS para a acurácia e para F1 na tarefa RTE. O menor erro médio quadrático (MSE) para a tarefa STS foi alcançado pelo modelo USE+Features (STS) and mBERT+Features (RTE).

Para a tarefa *Named Entity*, Souza implementou uma variação nos exemplos de entrada. Em vez de usar um contexto por sentença, ele usou um contexto por documento para obter as vantagens de longos contextos no momento da codificar os *tokens* de acordo com a representação BERT. Este modelo ele chamou de $BERTimbau + CRF$.

Foram utilizados os conjuntos de dados HAREM e Mini HAREM ((SANTOS et al.,)) que possui anotações em diversos domínios e estão classificadas como Person, Organization, Location, Value, Date, Title, Thing, Event, Abstraction, and Other. Foram criados dois cenários: um primeiro com todas as classes, *Total Scenario*, e um segundo, *Selective Scenario*, com cinco classes: Person, Organization, Location, Value, and Date.

Para a tarefa *Named Entity* os resultados foram comparados aos modelos CharWNN, LSTM-CRF, BiLSTM-CRF+FlairBBP, mBERT e mBERT+CRF. A Tabela 13 mostra os resultado obtidos. O modelo $BERTimbau_{Large} + CRF$ obteve resultados superiores para as métricas *precision* e F1 tanto no *Total Scenario* quanto no *Selective Scenario*.

Tabela 13 – Apresentação dos resultado da avaliação dos modelos *BERTimbau* e *BERTimbau + CRF* aplicados a NER. Os resultados foram comparados aos modelos que foram avaliados com o HAREM.

Arquitetura	Total Scenario			Selective Scenario		
	Prec.	Rec.	F1	Prec.	Rec.	F1
CharWNN	67,24	63,7	65,4	74,0	68,7	71,2
LSTM-CRF	72,8	68,0	70,3	78,3	74,4	76,3
BiLSTM-CRF+FlairBBP	74,9	74,4	74,6	83,4	81,2	82,3
mBERT	71,6	72,7	72,2	77,0	78,8	77,9
mBERT + CRF	74,1	72,2	73,1	80,1	78,3	79,2
BERTimbau Base	76,8	77,1	77,2	81,9	82,7	82,2
BERTimbau Base + CRF	78,5	76,8	77,6	84,6	81,6	83,1
BERTimbau Large	77,9	78,0	77,9	81,3	82,2	81,7
BERTimbau Large + CRF	79,6	77,4	78,5	84,9	82,5	83,7

Fonte: (SOUZA et al., 2020)

2.3 Considerações Finais

Listamos neste capítulo exemplos de arquiteturas recentes de redes neurais para processamento de linguagem natural com mecanismos diferentes para pré-treinar modelos que podem ser otimizados através de ajustes finos para execução de tarefas finais como classificador de texto, inferência de linguagem e responder questões.

Para o desenvolvimento do nosso classificador de notícias utilizaremos o modelo BERTimbau de (SOUZA et al., 2020), modelo com arquitetura BERT de (DEVLIN et al., 2018) e pré-treinado na língua portuguesa. BERT possui multicamadas bidirecionais e é um codificador Transformer que foi proposto por (VASWANI et al., 2017).

Os modelos pré-treinados baseados na arquitetura *Transformer* podem ser disponibilizados, junto com o *tokenizador*, na plataforma Hugginface. O modelo BERTimbau está disponível nesta plataforma, desta forma, isto permitirá que apliquemos ajustes finos para treinar o modelo do nosso classificador de textos.

3 CONCEITOS FUNDAMENTAIS

Neste capítulo apresentamos o formalismo teórico dos algoritmos de classificação utilizados nos experimentos a título de comparação ou linha de base com relação aos métodos de aprendizado profundo. Mostramos a matriz de representação que utiliza a abordagem *bag of words* e é utilizada como entrada para classificação de textos pelos algoritmos clássicos. Dentre os diversos algoritmos com diferentes abordagens, mostramos o Multinomial Naive-Bayes e o *Random Forest*. Também mostramos os vetores de representação utilizados pelo BERT e BERTimbau, e explicamos o Ajuste Fino. Por fim, mostramos a técnica de redução de dimensionalidade t-SNE.

3.1 *Bag of Words*

Para o Processamento de Linguagem Natural utilizando algoritmos clássicos, uma das abordagens mais utilizadas é a *bag of words*, na qual cada documento é representado como um vetor de palavras e cada palavra é representada por pontuação calculada por TF-IDF (Term Frequency(TF) — Inverse Dense Frequency(IDF)).

(MATSUBARA; MARTINS; MONARD, 2003) descreve esta estrutura como uma matriz de representação do vocabulário de um corpo (conjunto de textos). Esta matriz de representação é definida assim:

Dada uma coleção de N documentos $D = d_1, \cdots, d_N$ e um conjunto C de y categorias $C = c_1, \cdots, c_y$ associadas à coleção de documentos D. Seja T o conjunto de palavras (termos) existentes no conjunto de documentos D, $T = t_1, \cdots, t_z$. Na matriz de representação, cada documento d_i pertence a uma linha da matriz que possui número de colunas igual ao número de termos z existentes em T e está associada a uma das c_y classes. Cada elemento a_{ij} da matriz é um termo t_j que pode ou não pertencer ao documento d_i.

A Tabela 14 ilustra a matriz de representação. Nela estão representados N documentos que são formados por Z palavras.

Tabela 14 – Matriz de representação da *bag of words* que é utilizada para representar o vocabulário de um corpus. $D = \{d_1, \cdots, d_N\}$ é o conjunto de N documentos, $C = \{c_1, \cdots, c_y\}$ é o conjunto de y classes e $T = \{t_1, \cdots, t_z\}$ é o conjunto de z palavras (termos) existentes em D.

	t_1	t_2	$\cdots$	t_z	C
d_1	a_{11}	a_{12}	$\cdots$	a_{1z}	C_1
d_2	a_{21}	a_{22}	$\cdots$	a_{2z}	C_2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
d_N	a_{N1}	a_{N2}	$\cdots$	a_{Nz}	C_y

TF-IDF determina a frequência relativa das palavras em um documento específico comparada a proporção inversa da palavra sobre todo o vocabulário do documento. Intuitivamente este cálculo determina a relevância de uma dada palavra em um documento. (RAMOS et al., 2003)

Formalmente, TF-IDF é definida: dado um conjunto de documentos D, seja w uma palavra e tomemos um documento d tal que $d \epsilon D$: $w_d = f_{w,d} \cdot log(|D|/f_{w,D})$, tais que $f_{w,d}$ é a frequência que w aparece em d, $|D|$ é o tamanho do conjunto de dados (corpus) e $f_{w,D}$ é o número de documentos em que w aparece em D.

3.2 Algoritmos de classificação

3.2.1 Multinomial Naive-Bayes

O Multinomial Naive-Bayes é um classificador probabilístico que calcula - utilizando o Teorema de Bayes - a probabilidade de um documento pertencer a cada uma das categorias, sendo que a categoria com maior probabilidade é a eleita. Ele é um classificador simples e apresenta bons resultados em PLN (SINGH et al., 2019).

Para explicar o formalismo deste classificador, como feito por (ABBAS et al., 2019), tomemos a fração de documentos em cada classe como sendo:

$\pi_c = \frac{class_c}{\sum_{n=1}^{N} class_n}$ t.q c é uma classe, N é o número total de palavras e n representa cada palavra.

Para calcular a probabilidade de cada palavra (w) por classe (c), obtém-se pela média de cada palavra para uma dada classe:

$P(w|c) = \frac{word_w c}{word_c}$ t.q $word_w c$ é a frequência de w na classe em c e $word_c$ é a quantidade total de palavras na classe c.

Para as palavras que não existem na classe a frequência será zero, então, faz-se necessário aplicar a Transformada de Laplace (do inglês Laplace Smoothing) com valor baixo de α, sendo que α representa o valor de suavização para as palavras que não aparecem no vocabulário do treinamento e $|V|$ o tamanho do vocabulário:

$$P(w|c) = \frac{word_w c + \alpha}{word_c + |V| + 1}$$

Combina-se, a distribuição de probabilidade de P com a fração de documentos pertencentes a cada classe:

$$Pr(c) \propto \pi_c \Pi_{w=1}^{|V|} Pr(w|c)^{f_w}$$

Para evitar estouro negativo, aplica-se a soma dos logaritmos:

$$Pr(c) \propto \log(\pi_c \Pi_{w=1}^{|V|} Pr(w|c)^{f_w})$$
$$Pr(c) \propto \log \pi_c + \Pi_{w=1}^{|V|} f_w \log(Pr(w|c))$$

Aplica-se o logaritmo também na frequência para suavizá-la no caso de uma palavra aparecer repetidas vezes.

$$Pr(c) \propto \log \pi_c + \Pi_{w=1}^{|V|} \log(1 + f_w) \log(Pr(w|c))$$

Aplica-se o idf (t_w) para generalizar a função, ele a suaviza e ajuda o modelo a melhorar o desempenho. Assim, a equação do modelo é:

$$t_w = \log(\frac{\sigma_{n=1}^{N} doc_n}{doc_W}) \; Pr(c) \propto \log \pi_c + \Pi_{w=1}^{|V|} \log(1 + f_w) \log(t_w Pr(w|c))$$

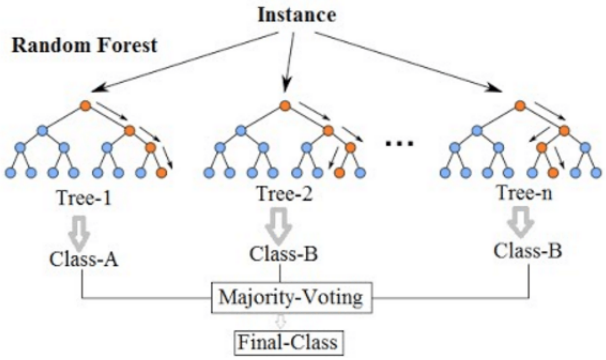
3.2.2 Random Forest

Random Forest é um algoritmo de classificação do tipo Comitê (também chamado de Ensembled em inglês) e apresenta maior acurácia e é mais robustos a ruídos que simples algoritmos de classificação (RODRIGUEZ-GALIANO et al., 2012). Ele apresenta algumas vantagens, por exemplo: rodar eficientemente em bases de dados grandes, gera uma estimativa de quais variáveis são importantes para a classificação - no nosso caso quais são as palavras mais influentes -, ser relativamente robusto a outliers e ruídos e é relativamente mais leve que outros métodos *ensembled* baseado em árvores, como o *boosting*.

Um classificador *Random Forest* consiste de uma combinação de classificadores do tipo árvores de decisão, sendo que cada classificador contribui com um simples voto para atribuição da classe mais frequente para um dado vetor X: $\hat{C}_{rf}^{B} = majorityvote\{\hat{C}_b(X)\}_1^B$,

sendo que $\hat{C}_b(X)$ é a classe predita da $b-$ésima árvore *random forest*. A Figura 8 ilustra a estrutura de classificação do *random forest*.

Figura 8 – Estrutura de classificação Random Forest.



Fonte: https://medium.com/swlh/random-forest-classification-and-its-implementation-d5d840dbead0 acessado em 01/12/2021.

Random Forest é um algoritmo de *bagging* ou *bootstrap aggregating*. *Bootstrap aggregating* é uma técnica que cria subconjuntos de amostras repetidamente e de forma aleatória a partir do conjunto de dados de treinamento e cada uma destes subconjuntos de amostra é submetida a b classificadores do tipo árvore de decisão que são treinadas em paralelo. Os exemplos do conjunto de dados de treinamento recebem pesos iguais e alguns dados podem ser usados mais de uma vez enquanto outros podem nunca ser usados.

3.3 BERTimbau

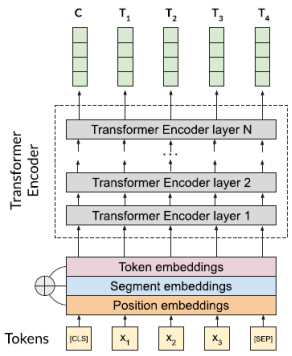
Nesta subseção explicamos como é representado o vocabulário do modelo BERT. O BERTimbau replica a arquitetura BERT e os procedimentos de treinamento com poucas mudanças (SOUZA et al., 2020). Mostramos como sequências de palavras dispostas em sentenças são representadas em vetores de palavras para entrada no modelo.

3.3.1 Representação do *embedding* de Entrada BERT

O modelo BERT é um modelo pré-treinado e necessita que as sentenças de entrada estejam de acordo com o padrão pré-estabelecido. Sentença pode ser qualquer texto contíguo, sendo que nos nossos experimentos tratamos como sendo o texto das notícias do nosso conjunto de dados.

O BERT pode receber como entrada uma ou duas sentenças e utiliza os *tokens* [CLS] para marcar o início da senteça e [SEP] para marcar o fim da sentença e também para separá-las quando a entrada contém um par delas.

Figura 9 – Representação do modelo BERT para uma sequência de entrada de 5 tokens.



Fonte:(SOUZA et al., 2020)

A Figura 9 mostra um diagrama do modelo BERT para uma sequência de entrada de 5 tokens. O modelo BERT recebe como entrada uma representação vetorial das sequências de subpalavras de cada texto. Para construir esta representação há três camadas: *Token Embeddings*, *Segment Embeddings* e *Position Embeddings*.

A camada *Token Embeddings* transforma as sequências de palavras das sentenças em representações vetoriais de dimensões fixas. Inicialmente, o texto é vetorizado e são adicionados nos vetores os *tokens* [CLS] e [SEP]. Também é aplicada a técnica de dividir a palavra em palavras menores, são as subpalavras (do inglês *wordpieces*). (WU et al., 2016) explica que nesta técnica as palavras são divididas em blocos, dado um modelo de bloco de palavras treinado. Então, símbolos especiais de limite de palavras (como estes ##) são adicionados antes do treinamento do modelo. No momento da decodificação, o modelo primeiro produz uma sequência de palavras, que é então convertida na sequência de *tokens* correspondentes.

Por exemplo: a sentença "Os pássaros cantam lindamente."é tokenizada da seguinte forma:

$$['[CLS]','Os','pássaros','canta','\#\#m','lin','\#\#damente','.','[SEP]']$$

As palavras "cantam"e "lindamente"foram subdividas em pedaços menores e nos pedaços cujos símbolos $'\#\#'$ os precedem, o BERT entende que eles fazem parte uma palavra maior e é precedido por outra sub-palavra [1].

A camada *Segment Embeddings* é responsável por distinguir duas sentenças A e B dentro de uma sequência. (SOUZA et al., 2020) explica que cada *token* da sentença A é vetorizado com identificadores a partir do vetor de *tokens* da senteça A, assim como

[1] https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#why-bert-embeddings acessado em 29/09/2021

cada *token* da sentença B é vetorizado com identificadores a partir do vetor de *tokens* da sentença B.

Por exemplo: Dadas duas sentenças A e B: "Os pássaros cantam lindamente".(A) e "Eles voaram."(B). *Segment Embedding* irá gerar um *embedding* identificando em qual sentença cada *token* está posicionado. No exemplo abaixo os valores 0 indicam que os *tokens* estão posicionados na sentença A e os valores 1 que eles estão posicionados na sentença B:

$$Token\ Embedding = [\prime[CLS]\prime,\prime Os\prime,\prime pássaros\prime,\prime canta\prime,\prime \#\#m\prime,\prime lin\prime,\prime \#\#damente\prime,\prime .\prime,$$
$$\prime[SEP]\prime,\prime Eles\prime,\prime voar\prime,\prime \#\#am\prime,\prime .\prime,\prime [SEP]\prime]$$
$$Segment\ Embedding = [0,0,0,0,0,0,0,0,0,0,1,1,1,1,1]$$

Position Embeddings cria uma propriedade temporal sobre as sequências de palavras, sendo que a posição das palavras na sentença tem influência no resultado da tokenização. Por exemplo: nas sentenças, "Ela viajou ontem"e "Ele foi falar com ela", o ela terá *embedding* de posições diferentes no resultado da tokenização.

Como explicado por nós em 2, o BERT foi implementado com base na arquitetura *Transformers* de (VASWANI et al., 2017) que diferentemente dos modelos com abordagem em LSTM, que codifica a natureza sequencial de suas entradas, aquela arquitetura utiliza a camada *Attention* para fazer com que o BERT aprenda uma representação vetorial para cada posição. Em outras palavras, *Position Embeddings* cria uma tabela de pesquisa onde a primeira linha é a representação vetorial de qualquer palavra na primeira posição, a segunda linha é a representação vetorial de qualquer palavra na segunda posição e assim por diante. Como BERT é limitado em sentenças de tamanho máximo de 512, esta tabela terá no máximo 512 linhas.

Por fim, as estruturas *embeddings* geradas em cada uma das camadas são combinadas elemento a elemento quando do processamento em cada uma delas para produzir uma única *embedding* de entrada que é passada para o classificador BERT.

3.3.2 Vocabulário do BERTimbau

O vocabulário do BERTimbau é composto de 30.000 unidades de subpalavras e segue a codificação padrão do modelo BERT, descrito em 3.3.1.

O tokenizador disponibilizado por (SOUZA et al., 2020) recebe como entrada um conjunto de documentos, efetua o processamento conforme o modelo BERT nas três camadas de tokenização (*Token Embeddings*, *Segment Embeddings* e *Position Embeddings*) e retorna o *embedding* do tipo *tensorflow* ou *torch*, contendo o vetor de representação dos índices de subpalavras de cada documento do conjunto de dados. Os índices das

subpalavras são obtidos do vocabulário do BERTimbau e estão na mesma ordem que as subpalavras nos documentos de entrada.

O tamanho de cada vetor de representação no *tensorflow* ou no *torch* é limitado ao tamanho definido para a sentença. Este valor é um hiperâmetro do modelo e, assim como no BERT, no BERTimbau este hiperparâmetro está limitado em 512 ((SOUZA et al., 2020) e (DEVLIN et al., 2018)). A quantidade de linhas em cada *tensorflow* também é um hiperparâmetro, o *batch size*. Para as subpalavras que não são reconhecidas pelo vocabulário do BERTimbau, o tokenizador atribuirá um índice referente à palavra desconhecida (UNK).

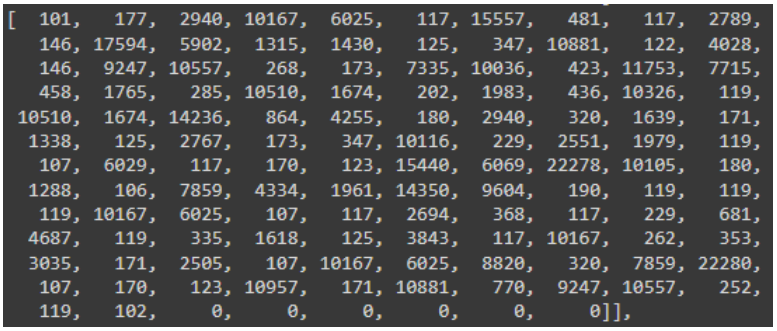
Na Tabela 15 é apresentado um exemplo de texto de notícias da nossa base de dados e as subpalavras geradas pelo tokenizador BERTimbau. Há algumas palavras que foram divididas de forma que as subpalavras não têm significado semântico para o Português, provavelmente são heranças do modelo BERT que foi herdado pelo BERTimbau, pois a palavra em inglês *Ladies* ele subdividiu corretamente.

Tabela 15 – Exemplo de geração de subpalavras pelo tokenizador BERTimbau.

Texto da Notícia
A cantora Rita Lee, 67 anos, deixou o característico tom ruivo de seu cabelo e assumiu o grisalho em fotos publicadas pelo cabeleireiro Duda Molinos no Instragram. Molinos divulgou três imagens da cantora ao longo do fim de semana em seu perfil na rede social. "Hoje,com a ovelha negra da família! Ladies and gentleman... Rita Lee", escreveu ele, na primeira delas. Em julho de 2014, Rita foi à estreia do musical "Rita Lee Mora ao Lado"com a raiz do cabelo já grisalha.
***Embedding* de subpalavras gerado pelo Tokenizador BERTimbau**
['[CLS]', 'A', 'cantora', 'Rita', 'Lee', ',', '67', 'anos', ',', 'deixou', 'o', 'característico', 'tom', 'ru', '##ivo', 'de', 'seu', 'cabelo', 'e', 'assumiu', 'o', 'gri', '##sal', '##ho', 'em', 'fotos', 'publicadas', 'pelo', 'cabel', '##eir', '##eiro', 'Du', '##da', 'Mol', '##inos', 'no', 'Ins', '##tra', '##gram', '.', 'Mol', '##inos', 'divulgou', 'três', 'imagens', 'da', 'cantora', 'ao', 'longo', 'do', 'fim', 'de', 'semana', 'em', 'seu', 'perfil', 'na', 'rede', 'social', '.', '"', 'Hoje', ',', 'com', 'a', 'ov', '##elh', '##a', 'negra', 'da', 'família', '!', 'Lad', '##ies', 'and', 'gent', '##lem', '##an', '.', '.', '.', 'Rita', 'Lee', '"', ',', 'escreveu', 'ele', ',', 'na', 'primeira', 'delas', '.', 'Em', 'julho', 'de', '2014', ',', 'Rita', 'foi', 'à', 'estreia', 'do', 'musical', '"', 'Rita', 'Lee', 'Mora', 'ao', 'Lad', '##o', '"', 'com', 'a', 'raiz', 'do', 'cabelo', 'já', 'gri', '##sal', '##ha', '.', '[SEP]']

A Figura 10 apresenta o *embedding* de representação do tokenizador BERTimbau que será submetido como entrada ao modelo para classificação. Esta representação se refere ao texto listado na Tabela 15. O hiperparâmetro de tamanho da sequência, neste exemplo, foi configurado para tamanho 128, porém o tamanho do embedding de subpalavras gerado foi de 122. Neste caso as posições faltantes do embedding é completado com índice 0, cujo significado é para dado faltante.

Figura 10 – Embedding de representação do texto da notícia descrito no Exemplo 15



3.3.3 Ajuste fino BERT

(XU et al., 2020) explica que para efetuar o ajuste fino, o BERT utiliza a saída da última camada (h) do primeiro token [CLS] como representação da entrada de sentença ou pares de sentença. Um classificador simples é adicionado ao topo do modelo para prever a probabilidade do rótulo y: $p(j|h) = softmax(Wh)$, sendo que W é a matriz de parâmetros específicos da tarefa de Processamento de Linguagem Natural.

BERT utiliza os mecanismos *self-attention* da arquitetura *Transfomers*. (CUI et al., 2019) explica que redes que possuem este mecanismo são capazes de diretamente relacionar *tokens* em diferentes posições a partir da sequência efetuada pela computação do *score attention* (pontuação de relevância) entre cada par de *tokens*.

No capítulo 2 citamos os trabalhos de (WU et al., 2016) que define os mecanismos de *self-attention* e (VASWANI et al., 2017) que utiliza estes mecanismos na arquitetura *Transformer*.

Reproduzimos a seguir a formalização apresentada por (CUI et al., 2019) para a computação da pontuação de relevância efetuada na camada de Atenção do BERT: dada a sequência de entrada $X = (x_1, \cdots, x_L)$, a representação de saída $y = (y_1, \cdots, y_L)$ é construída aplicando a soma dos pesos da transformação dos elementos de entrada x baseado na relevância, sendo que os elementos $\{x_i,\ y_i\}\ \epsilon\ R^d$. A $i-ésima$ saída y_i que é computada como:

$$y_i = \sum_{j=1}^{L} \alpha_{ij}(x_j W_v)$$
$$e_{ij} = \frac{(x_i W_q)(x_j W_k)^T}{\sqrt{d}}$$
$$\alpha_{ij} = \rho(e_{ij})$$

$W_q\ \epsilon\ R^{d\ x\ d}$, $W_k\ \epsilon\ R^{d\ x\ d}$, $W_v\ \epsilon\ R^{d\ x\ d}$ são as matrizes de treinamento obtidas do pré-treinamento, ρ é a probabilidade da função de mapeamento (probabilidade de acontecer próximo símbolo, dada a codificação da sentença de origem e a decodificação

da sentença alvo). O peso da *attention* α_{ij} representa a relevância entre o $i-$ésimo e o $j-$ésimo elemento.

A escolha para $\rho(e_{ij}) = softmax(e_{ij} = \frac{exp(e_{ij})}{\sum_{t=1}^{L} exp(e_{it})})$

Entretanto, $softmax$ é uma função estritamente positiva e produz a distribuição de probabilidade da atenção com total suporte. Isto significa que não há um valor de probabilidade zero para os relacionamentos menos significativos entre pares de palavras (*tokens*). Isto é, definindo pesos com valores diferentes de zero em todos os relacionamentos poderia também degradar a interpretabilidade. Porém, com a transformação $softmax$, as redes *self-attention* não dão mais atenção a estas importantes conexões enquanto são facilmente distribuídas por muitas palavras não relacionadas.

3.3.4 t-Distributed Stochastic Neighbor Embedding (tSNE)

t-SNE é uma técnica de redução de dimensionalidade usada para representar conjunto de dados com um número alto de dimensões em um espaço dimensional baixo, de duas ou três dimensões para que possam ser visualizadas.

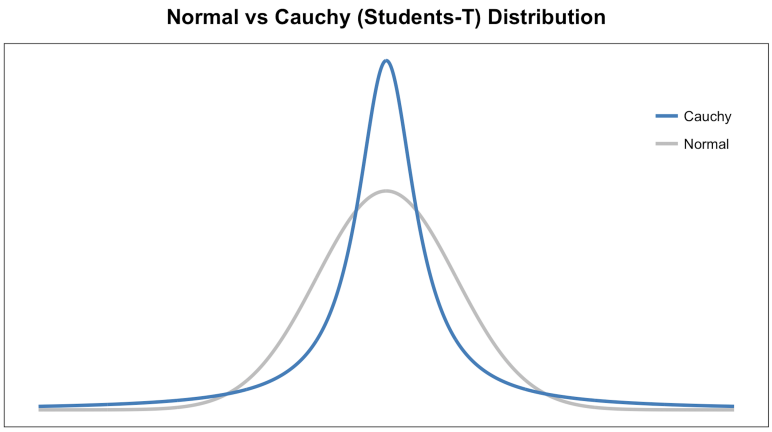
(MAATEN; HINTON, 2008) mostra que o algoritmo t-SNE calcula a medida de similaridade entre pares de instâncias no espaço dimensional superior e no espaço dimensional inferior. Em seguida, tenta otimizar essas duas medidas de similaridade usando uma função de custo.

Para medir a similaridade entre os pontos no espaço dimensional elevado, cada ponto é centralizado em uma distribuição Gaussiana, então é medida a densidade de todos os outros pontos sob essa distribuição Gaussiana. Estas probabilidades são proporcionais as semelhanças, isto é, se dois pontos quaisquer têm valores iguais sobre a curva Gaussiana, então suas proporções e semelhanças são iguais e, portanto, há semelhanças locais na estrutura deste espaço de alta dimensão. A distribuição Gaussiana pode ser manipulada usando através da perplexidade que influência a variância da distribuição e essencialmente o números de vizinhos mais próximos.

Um segundo conjunto de probabilidades no espaço de baixa dimensão é obtido usando a distribuição t-Student com um grau de liberdade, também chamada de distribuição Cauchy [2]. A Figura 11 mostra que a distribuição t-student tem caldas mais longas que a distribuição normal permitindo melhor modelagem para pontos distantes.

[2] https://medium.com/@violante.andre/an-introduction-to-t-sne-with-python-example-47e6ae7dc58f acessado em 05/12/2021

Figura 11 – comparação entre as distribuições Gaussiana e a t-student.



Fonte:https://medium.com/@violante.andre/an-introduction-to-t-sne-with-python-example-47e6ae7dc58f acessado em 05/12/2021

Para medir a semelhança entre estes dois conjuntos de probabilidade, é calculada a diferença entre as distribuições de probabilidades dos espaços bidimensionais usando a divergência de Kullback-Liebler (KL) que compara de forma eficiente grandes valores em espaços de alta dimensionalidade e de baixa dimensionalidade (PONTI et al., 2017). Para minimizar a função de custo KL, é utilizado a descida do gradiente.

4 MÉTODO

Neste capítulo descrevemos os procedimentos para construção da base de dados com textos jornalísticos de alguns dos principais sítios eletrônicos de notícias brasileiros. Apresentamos os tratamentos iniciais que foram aplicados sobre os dados e posteriormente efetuamos uma análise descritiva. Mostramos também os procedimentos para o treinamento do nosso classificador utilizando a abordagem *bag of words* com a métrica TF-IDF (*Term Frequency – Inverse Document Frequency*) que é submetida como entrada para os algoritmos de classificação *Naive-Bayes* e *Random-Forest*. Por fim, mostramos também o treinamento do classificador utilizando o modelo pré-treinado BERTimbau.

4.1 Obtenção dos Dados

A base de dados dos experimentos é composta por dois conjuntos de notícias: o primeiro contempla textos extraídos de sítios de jornais eletrônicos, sendo que foi desenvolvido por nós um algoritmo - em python e com o uso da biblioteca BeautifulSoup - para esta finalidade. O segundo conjunto de dados é uma base de notícias do jornal Folha de São Paulo disponibilizada no sítio da Kaggle [1].

O nosso algoritmo coletou diariamente durante o período de 17/04/2021 a 24/07/2021 as notícias dos sítios eletrônicos UOL, google-news-br, Poder360 e Notícias da TV. O total de notícias únicas coletadas em três meses totalizou 8.776 diferentes notícias. A Tabela 16 lista a quantidade de notícias extraídas de cada fonte.

Tabela 16 – Quantitativo de notícias por fonte extraídas em três meses pelo nosso algoritmo de *web scrapping*.

Fonte	Qtd Notícias
UOL	3.977
google-news-br	2.089
Poder360	1.448
Notícias da TV	1.262
Total	8.776

Os atributos coletados, além do texto da notícia, foram o título da notícia, a data de publicação, a sessão no sítio eletrônico em que foi publicada, a qual nominamos de categoria e o link (a URL) da notícia.

O outro conjunto de dados com notícias da Folha de São Paulo contempla publicações no período de janeiro de 2015 a setembro de 2017. No total são 166.288 notícias

[1] https://www.kaggle.com/marlesson/news-of-the-site-folhauol acessado em 31/07/2021

contendo título, texto, data da publicação, categoria, subcategoria e link da notícia. A subcategoria optamos por não utilizar, pois as notícias coletadas não possuem este atributo.

A dificuldade de utilizar somente os dados coletados nesse trabalho é sua baixa volumetria, pois seria necessário um tempo bem maior que três meses coletando textos diariamente para se ter um volume de notícias que permitisse treinar um classificador de bom desempenho. Porém, a vantagem deles sobre os dados do segundo conjunto é que estas notícias são mais atuais e por isto enriquecem o vocabulário do classificador com temas que não havia nenhuma referência em textos jornalísticos do passado, como a pandemia de COVID. Contudo, os dados da Folha de São Paulo trazem como vantagem, além de maior quantidade, diversidade de vocabulário para temas que se repetem sazonalmente, como jogos olímpicos, campeonatos esportivos anuais, eleições, corrupção nos governos, problemas na economia, no clima, entre vários outros. Desta forma, o vocabulário das notícias do passado pode ser usado para classificar as notícias da atualidade.

Outro ponto positivo para o conjunto da Folha de São Paulo é que o texto das notícias é limpo, isto é, contém somente o texto escrito pelo jornalista, e também a categoria que aponta para a sessão na qual a notícia foi publicada está corretamente preenchida. Nos dados coletado, o texto das notícias tem propaganda e links para outras notícias do sítio eletrônico hospedeiro e um breve resumo desta notícia que será lida neste link, isto é, contém dados que não contemplam somente o texto do jornalista. Uma outra dificuldade encontrada foi a categorização correta destas notícias, pois os sítios eletrônicos não necessariamente as disponibilizam no HTML da página, sendo necessário capturar o dado na sessão indicada no link da notícia.

4.2 Tratamento e Análise Descritiva

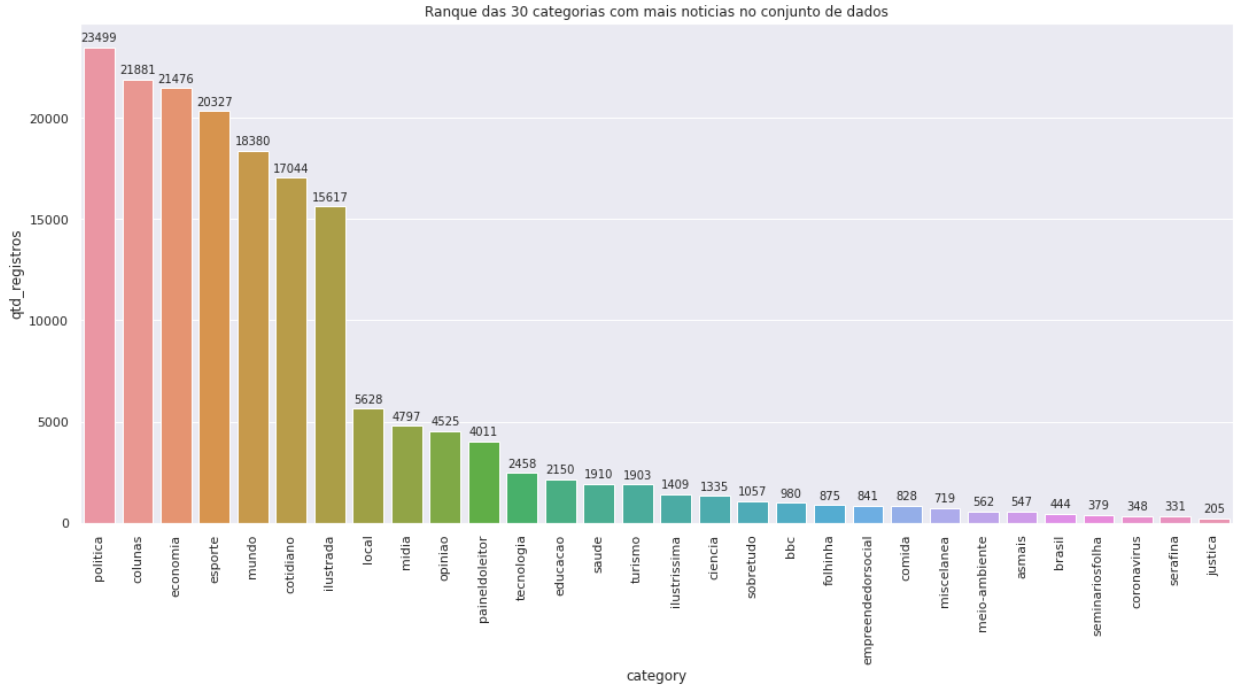
Para poder integrar os dois conjuntos de dados, o primeiro passo foi efetuar a limpeza e a correta categorização das notícias do nosso conjunto de dados. Cada um dos sítios eletrônicos possui um padrão de html para publicação, por exemplo: na primeira linha contém o nome do jornalista, na segunda linha contém a data da publicação e a terceira linha contém os registros de alteração de modificação na publicação com a nova data, depois vem o texto da notícia e no final alguma frase de propaganda ou chamada de link para algum vídeo no *youtube* ou links de outras reportagens do sítio. Desta forma, foi possível extrair somente o texto do jornalista usando estes padrões em combinações de expressões regulares.

Para efetuar a categorização dos registros que não contêm o valor deste atributo diretamente no html de publicação da notícia, utilizamos, inicialmente, o nome da sessão do sítio eletrônico que compõe o link da notícia. O resultado deste tratamento foi a obtenção de muitas categorias e com poucos registros para cada uma delas.

Para resolver esta granularidade fina, agrupamos as categorias cujos nomes das sessões representam assuntos específicos para nomes de sessões com semântica mais genérica. Por exemplo: sessões que falam especificamente de futebol, vôlei e nomes de times de futebol foram agrupadas com a categoria esporte; sessões de mercado e finanças foram agrupadas com a categoria economia; sessões sobre coronavírus, covid e pandemia foram agrupadas na categoria coronavírus; cinema, oscar, vídeos, música, lives foram agrupadas na categoria cultura e sessões com notícias locais de cada estado foram nomeadas para local.

Por fim, efetuamos a união dos registros, excluímos registros com texto jornalístico ausente, colocamos nomes únicos para as categorias com mesmo significado e criamos uma coluna numerada para a categoria. O resultado final foi um conjunto de dados com 177.407 notícias. A Figura 12 ilustra o ranque das trinta categorias com mais notícias, sendo que as sete primeiras categorias representam aproximadamente 78% dos registros da base de dados.

Figura 12 – Ranque das trinta categorias com mais registros na base de dados.



Como o recurso computacional disponível para treinar o nosso classificador é limitado, extraímos uma amostra da nossa base de dados. Para compor esta amostra, escolhemos dez categorias que representam semanticamente os assuntos abordados em outras categorias.

Para a escolha destas categorias, observamos graficamente que algumas das sessões

dos jornais possuem textos jornalísticos que tratam de assuntos que ficariam bem posicionados também em outras sessões. A Figura 13 mostra a nuvem de palavras mais repetidas no conjunto de dados de cada uma das 30 categorias com mais notícias.

Além de analisar graficamente as palavras mais repetidas, lemos algumas notícias escolhidas de forma aleatória em cada categoria, então observamos a existência de associações no vocabulário e na semântica entre as categorias. Por exemplo: as notícias da categoria "coluna"estão associadas à política e a economia, assim como as categorias cotidiano, opinião e local se referem a temas político, econômico e esportivo. Desta forma, escolhemos as categorias que pudessem indicar o assunto que é tratado nas notícias.

Também foi critério de seleção de uma determinada categoria a quantidade de textos jornalísticos, com preferência para maiores quantidades de forma a construir um vocabulário representativo. As dez categorias finais são: política, economia, esporte, mundo, ilustrada, mídia, tecnologia, educação, saúde e ciência. Não foram inseridas nestas categorias textos jornalísticos das categorias preteridas.

Figura 13 – Nuvem de palavras para as 30 categorias com mais registros na base de dados.

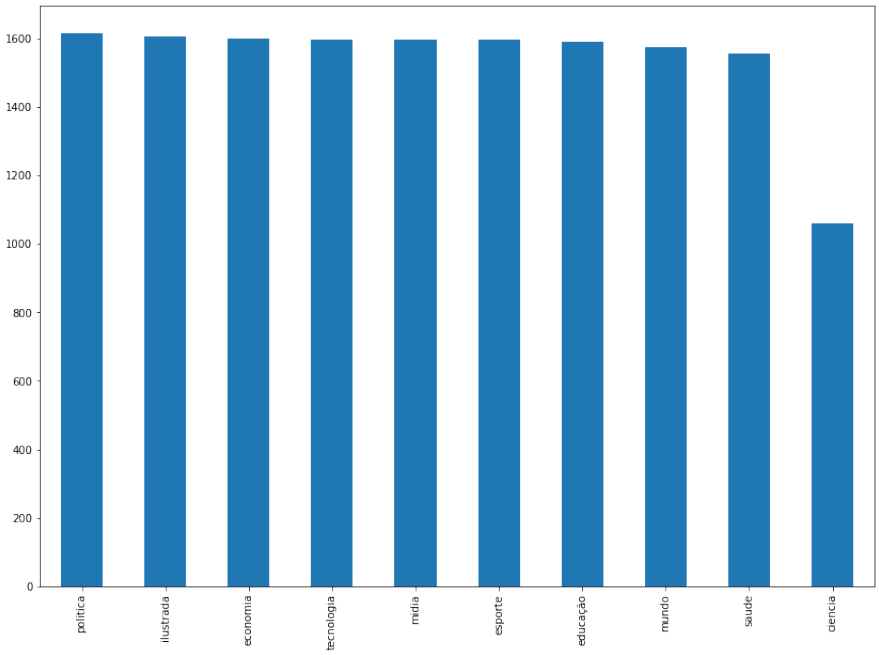
Para obtermos a amostra, construímos um subconjunto de dados selecionando

2.000 notícias de forma aleatória para cada uma das seguintes categorias: política, economia, esporte, mundo, ilustrada, mídia, tecnologia e educação. Saúde e ciência possuem, respectivamente, 1.910 e 1.335 notícias na base de dados, selecionamos todos os registros.

Dividimos este subconjunto de dados em treinamento, validação e teste, sendo que a proporção foi de 80% para o treinamento, 10% para a validação e 10% para o teste. O subconjunto de validação é utilizado na validação da convergência da função de perda para o menor erro durante o treinamento do classificador com o modelo BERTimbau. Na abordagem clássica, o subconjunto de validação foi adicionado ao subconjunto de teste para validar o desempenho do treinamento do *Naive-Bayes* e do *Random-Forest*.

A Figura 14 mostra o quantitativo de registros para cada categoria no treinamento. Devido ao critério adotado para a quantidade de registros da amostra em cada categoria, o conjunto de treinamento possui uma distribuição tendente a ser balanceada, sendo que a categoria ciência possui uma quantidade de registros um pouco inferior em relação as outras.

Figura 14 – Quantitativo de notícias para cada categoria no subconjunto de treinamento.



A Tabela 17 mostra que a metade dos textos possuem tamanho de até 394 palavras e que 75% dos textos possuem tamanho de até 596 palavras, sendo 464 palavras em média com desvio padrão 350. O desvio padrão é considerado elevado quando comparado aos valores da media e mediana. Para computar a estatística descritiva não foi feito nenhum tratamento nos dados.

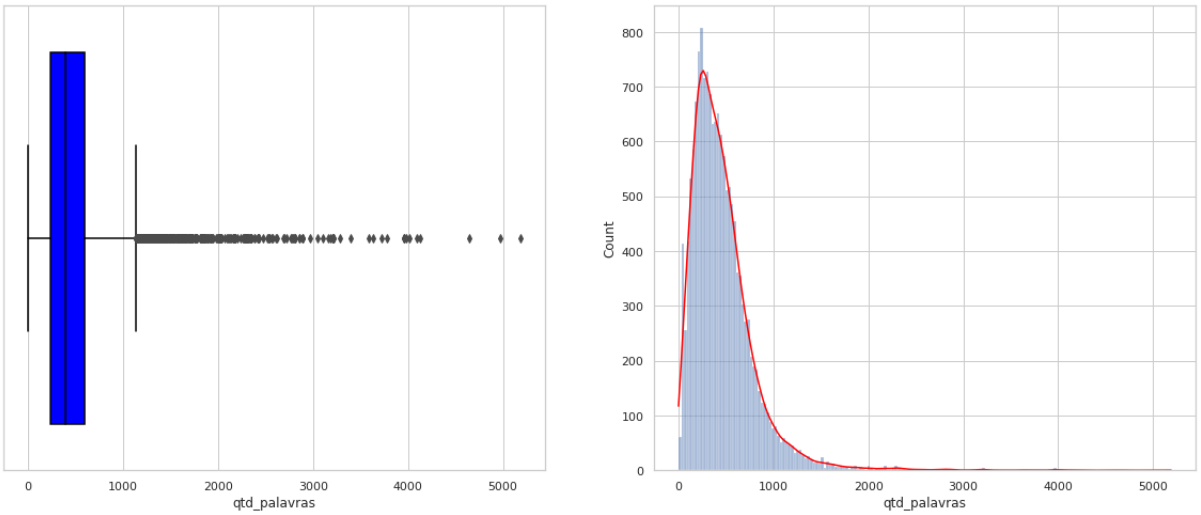
O boxplot da Figura 15 registra os *outliers* da amostra de treinamento, sendo que há muitos textos entre 1.000 e 2.000 palavras e textos com mais de 3.000 palavras. Há um

Tabela 17 – Estatística descritiva do tamanho dos textos em quantidade de palavras na amostra de treinamento.

métrica	qtd palavras
média	463
desvio padrão	350
mínimo	1
25%	236
50%	394
75%	596
máximo	5.190
tamanho da amostra	15.396 notícias

texto com mais de 5.000 palavras (o maior texto do treinamento possui 5.190 palavras).

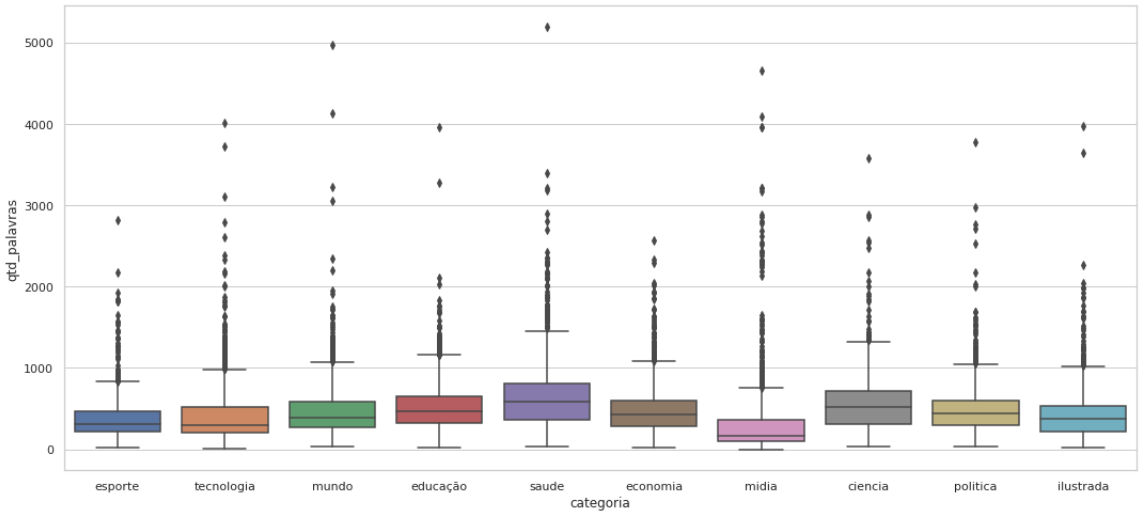
Figura 15 – Distribuição do tamanho dos textos em quantidade de palavras na amostra de treinamento. À esquerda o *boxplot* mostra a existência de textos grandes com mais de mil palavras e à direita o gráfico de densidade mostra que a distribuição da quantidade de palavras é assimétrica a esquerda.



O menor texto possui uma palavra e há no treinamento 25 registros com textos que possuem entre uma palavra e dez palavras, sendo que 24 registros são da categoria mídia e se referem a uma chamada de reportagem seguida de um link. Como estes textos não agregam valor ao classificador, optamos por retirá-los do treinamento.

A mediana e a dispersão da quantidade de palavras na amostra de treinamento é diferente em cada uma as categorias. A partir da análise do gráfico na Figura 16, percebemos que "mídia"é a categoria com textos mais curtos seguidos por "tecnologia"e "esporte". Por outro lado, "saúde"possui textos mais longos, seguidos por "ciência"e "educação". Todas as categorias possuem textos com mais de mil palavras.

Figura 16 – Boxplots para a quantidade de palavras em cada uma das categorias



Após efetuar a limpeza do texto através da remoção de URLs, números, pontuação, palavras comuns (*stop words*), extrair os radicais das palavras e excluir os registros cujos textos possuem menos de 10 palavra, obtemos a média de 254 palavras e a mediana de 218 palavras. Obtemos também uma redução na dispersão, sendo que agora o maior texto possui 2.908 palavras. A Tabela 18 mostra a estatística descritiva para o treinamento após o processamento de limpeza dos textos e extração dos radicais das palavras.

Tabela 18 – Estatística descritiva do tamanho dos textos em quantidade de palavras após o processo de limpeza dos textos e extração dos radicais das palavras.

métrica	qtd palavras
média	254, 66
desvio padrão	188, 30
mínimo	11
25%	131
50%	218
75%	327
máximo	2.908
tamanho da amostra	15.361 notícias

4.3 Implementação

4.4 Classificação das Notícias

Os modelos BERT são um dos exemplos estado-da-arte em Processamento de Linguagem existentes na atualidade, sendo que o BERTimbau é um modelo BERT pré-treinado para o Português brasileiro. Estes modelos pré-treinados são avaliados em base de

dados disponibilizadas especialmente para esta finalidade, porém o escopo destas avaliações são diferentes do objetivo do nosso classificador. Desta forma, é necessário comparar o desempenho do nosso classificador treinado a partir do BERTimbau com o desempenho de outro classificador que aborda uma solução diferente e que resolve o mesmo problema.

Escolhemos treinar dois classificadores adicionais utilizando os algoritmos Multinomial Naive-Bayes e *Random Forest* e efetuamos a comparação dos resultados destes diferentes classificadores. Descrevemos nesta seção os procedimentos para treinamento dos diferentes classificadores.

4.4.1 Abordagem clássica: Multinomial Naive-Bayes e *Random Forest*

Nesta subseção vamos explicar como geramos a *bag of words* ou matriz de representação das notícias que armazena o valor do TF-IDF de cada palavra. Esta matriz é utilizada como entrada para os algoritmos Multinomial Naive-Bayes e *Random Forest*. Explicamos também como efetuamos o treinamento destes dois classificadores.

4.4.1.1 Construção da Matriz de Representação

Para o treinamento do nosso classificador utilizando os algoritmos clássicos, adotaremos a abordagem *bag of words* (3.1) para a construção do vocabulário e pontuaremos cada palavra com o valor do TF-IDF (*Term Frequency(TF) — Inverse Dense Frequency(IDF)*). Vocabulário é o conjunto de palavras únicas utilizadas nos textos jornalísticos do nosso conjunto de treinamento.

Nesta abordagem, cada texto jornalístico é representado por um vetor de palavras que compõem a notícia. Utilizamos a representação da bag of words no formato de matriz conforme descrito em 3.1.

Para construir a matriz de representação, no Python, utilizamos o método CountVectorizer da biblioteca scikit-learn versão 1.0.1.

Na matriz de representação calculamos o TF-IDF de cada palavra representada nas colunas. O conjunto de dados de treinamento do nosso classificador possui 15.396 notícias com um vocabulário de 52.300 termos. A matriz de representação possui dimensão de 15.396 x 52.300. Provavelmente os classificadores clássicos terão problemas com a alta dimensionalidade causada pelo elevado número de termos em relação a quantidade de exemplos de notícias. Este tipo de problema é conhecido como a maldição da dimensionalidade em que a quantidade de dados necessários para alcançar o conhecimento desejado impacta exponencialmente o número de atributos necessários, sendo que o desempenho do classificador tende a se degradar a partir de um determinado número de atributos, mesmo que eles sejam úteis (CAMARGO, 2010).

Não são todos os termos do vocabulário de notícias que são úteis para que os

classificadores Naives-Bayes e *Random Forest* aprendam a identificar a categoria alvo. Existem termos que podem ser removidos do corpus como links de acesso a páginas e vídeos no *youtube*, números e pontuações.

As palavras comuns que aparecem em quase todos os textos jornalísticos como artigos, conjunções, preposições entre outras também não permitem que o classificador possa aprender termos que caracterizem uma determinada classe. Estes termos são conhecidos em PLN como *stop words*.

Outra técnica muito comum para reduzir a dimensionalidade em PLN é considerar somente o radical das palavras, pois todas a derivações de uma palavra são reduzidas ao radical. Esta técnica é conhecida como *stemming*. Então, antes de gerar a matriz de representação efetuamos um pré-processamento para remover links de páginas e vídeos no youtube, números, pontuações, as *stop words*, colocamos o texto em letras minúsculas e aplicamos a técnica de *stemming*.

Para remoção das *stop-words* e aplicação da técnica *stemming*, no Python, utilizamos a biblioteca do nltk versão 3.2.5.

Após o pré-processamento a quantidade de palavras do vocabulário reduziu para 40.966 palavras. Ainda é uma quantidade alta para compor a matriz de representação. Porém, podemos calcular a frequência que cada palavra do dicionário aparece em cada uma das notícias e então podemos excluir as palavras que não são *stop words*, mas que aparecem em um percentual alto de notícias do conjunto de treinamento e causam o mesmo efeito que as *stop words*. Por outro lado, as palavras que aparecem somente em alguns textos podem induzir erroneamente o classificador a uma determinada categoria. Então podemos excluir também as palavras que aparecem em poucos textos.

Definimos um percentual de 1% de documentos como a quantidade mínima de documentos que uma palavra deveria estar presente, pelo menos uma vez, para poder compor a matriz de representação. Também definimos o percentual de 98% de documentos como a quantidade máxima de documentos, isto é, uma palavra deverá estar presente em mais de 1% e em menos de 98% dos documentos do conjunto de treinamento para que esteja na matriz de representação.

Uma outra abordagem que adotamos para a construção da nossa matriz de representação foi a utilização de sequência n-grama (do inglês *n-grams*) (ROOM, 2021). Além de considerarmos uma palavra isolada do vocabulário como característica, a combinação de sequências de duas ou mais palavras pode fazer com que o classificador aprenda melhor a identificar a categoria correta para cada vetor de palavras ou sequência de palavras.

Para os nossos experimentos utilizamos bigramas e para evitar que eles aumentem ainda mais o número de dimensões da matriz de representação, aplicamos também os percentuais de frequência mínima e máxima sobre eles. Após computar a matriz de

representação com estes critérios, reduzimos o tamanho do vocabulário para 3002 termos, sendo que 390 termos são bigramas.

Uma vez que já temos os termos que compõem a matriz de representação, o próximo passo é calcular o valor do TF-IDF para cada um deles em cada documento do nosso conjunto de dados. Para computar, no Python, este cálculo usamos o método TfidfTransformer da biblioteca sckit-learn versão 1.0.1.

4.4.1.2 **Experimentos com os algoritmos clássicos**

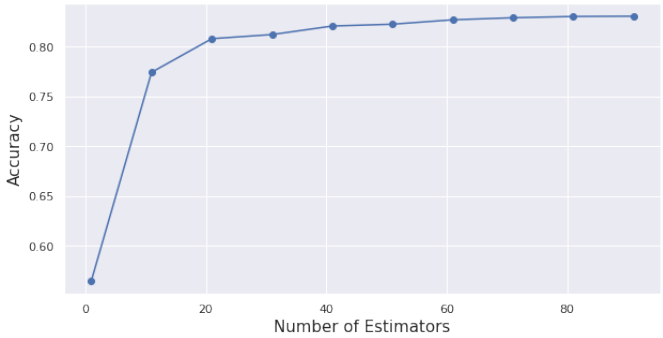
Para treinar o nosso classificador **Multinomial Naive-Bayes** no Python, utilizamos o classificador MultinomialNB contido na classe naive_bayes da biblioteca sklearn versão 1.0.1, sendo que passamos como parâmetro a matriz de representação.

Como este algoritmo utiliza todo o conjunto de dados para a calcular de uma só vez a probabilidade de um documento pertencer a cada uma das classes e sem fazer variações de amostras durante o processamento, o executamos somente uma vez.

Para treinar o nosso classificador de notícias ***Random Forest***, o primeiro passo foi definir o número de classificadores do tipo árvore. Para isto simulamos o treinamento com várias quantidades diferentes para obtermos o melhor valor para o hiperparâmetro *n_estimator*. Utilizamos o classificador RandomForestClassifer da classe ensemble da biblioteca sklearn versão 1.0.1, no Python, e passamos como entrada a matriz de representação.

Executamos o treinamento 10 vezes com a mesma semente e variando o valor do hiperparâmetro *n_estimator* em quantidade de 10 a cada execução, sendo que iniciamos em 1 e finalizamos em 91. A Figura 17 mostra que o melhor valor para *n_estimator* é 91 para o valor de acurácia $= 83,03\%$. Testamos também para valores maiores que 91 e menores que 200, porém o ganho em desempenho é muito pequeno e a complexidade do classificador aumenta tornando o treinamento mais lento e com consumo maior de recursos computacionais.

Figura 17 – Simulação do n_estimator para o classificador *Random Forest* de notícias.



O próximo passo foi efetuar o treinamento para 10 repetições utilizando o melhor valor do hiperparâmetro *n_estimator* obtido na simulação. Para cada treinamento utilizamos um número de semente diferente. Não fizemos nenhuma otimização nos outros hiperparâmetros.

4.4.2 **Experimentos BERTimbau**

Para executar os experimentos com o modelo BERTimbau, utilizamos os conjuntos de dados de treinamento e validação. Não aplicamos nenhum pré-processamento nos textos como fizemos na abordagem clássica. Os dados de treinamento foram submetidos ao ajuste fino do modelo BERTimbau para classificação das notícias e o dados de validação foram utilizados para avaliar o modelo treinado durante o treinamento do classificador. Utilizamos o modelo pré-treinado $BERTimbau_{BASE}$ e o tokenizador deste modelo disponibilizados na plataforma HuggingFace através da biblioteca, do Python, transformers versão 4.14.1, sendo que o acesso a estas publicações é feito por API e passando como parâmetro o nome 'neuralmind/bert-base-portuguese-cased' para o modelo e 'neuralmind/bert-base-portuguese-cased' para o tokenizador. Também foi utilizada a biblioteca torch versão 1.8.1.

Os hiperparâmetros utilizados são o de tamanho máximo da sequência (*seq_max_length*), o *batch size* (bs), a taxa de aprendizagem (lr), o número de épocas (n_epochs) e o número de classes do conjunto de treinamento.

O número de classes do nosso conjunto de treinamento é 10 (conforme explicado na subseção 4.2) e a função de perda que utilizamos foi a *crossEntropyLoss* da biblioteca *torch.nn* do python. Ela é utilizada para treinamento de classificadores com multiclasses.

Efetuamos experimentos para tamanhos diferentes de sequência: 128, 256, 512 e 394 de forma a observar se o classificador apresenta um bom desempenho com quantidades maiores de subpalavras. Para tal, tomamos como referência os valores da análise descritiva: sabemos que 25% da amostra possui sequências de tamanho limitado a 236 *tokens*, 50% possui sequências de tamanho limitado a 394 *tokens* e 75% possui tamanho limitado a 596 *tokens*. Adotamos então a mediana da amostra como um dos valores e como o tamanho máximo de sequência do $BERTimbau_{BASE}$ é 512, adotamos os outros três valores como potência de 2.

A Tabela 19 apresenta os valores destes hiperparâmetros para os quatro experimentos realizados. Para cada um destes experimentos, efetuamos dez execuções diferentes do ajuste fino, sendo que para cada execução adotamos um valor único para a semente.

Para a execução do ajuste fino, para cada texto jornalístico na base de dados de treinamento executamos o processo de tokenização para a geração do *embedding* de representação. Submetemos cada *embedding* de representação como entrada para a pilha

Tabela 19 – Hiperparâmetros utilizados para o ajuste fino do classificador BERTimbau em cada experimento.

Experimento	seq_max_length	bs	lr	n_epochs
1	128	8	$1e-6$	5
2	256	8	$1e-6$	5
3	512	8	$1e-6$	5
4	394	8	$1e-6$	5

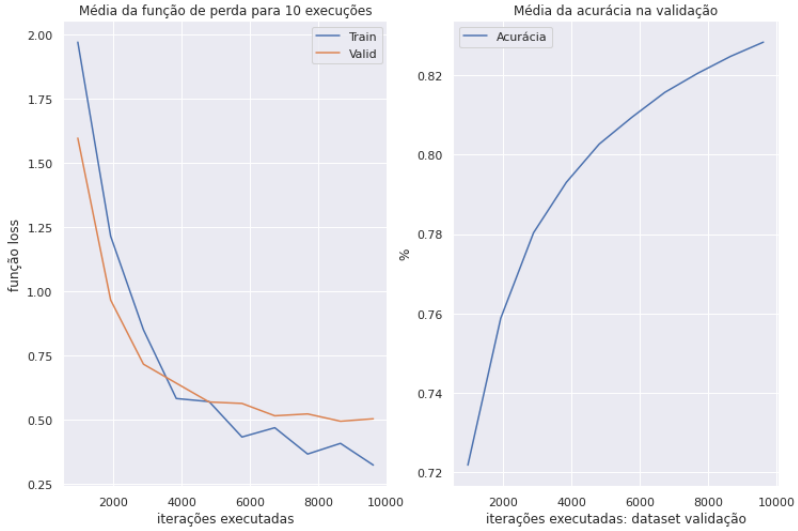
de *Transformers* no modelo $BERTimbau_{BASE}$, conforme representado na Figura 9. O modelo, então, nos fornece como saída da última camada a pontuação de relevância para cada uma das dez classes. Em cada época, avaliamos o modelo utilizando o conjunto de validação cada vez que ele é treinado com a metade do conjunto de treinamento e obtemos o valor médio da função de perda para o treinamento e para a validação. Também calculamos para o modelo de validação a acurácia.

4.4.2.1 **Análise do Treinamento**

Nesta subseção mostramos os resultados obtidos para a função de perda e a acurácia durante a fase de treinamento na execução dos experimentos. Estes resultados foram obtidos no conjunto de dados de validação.

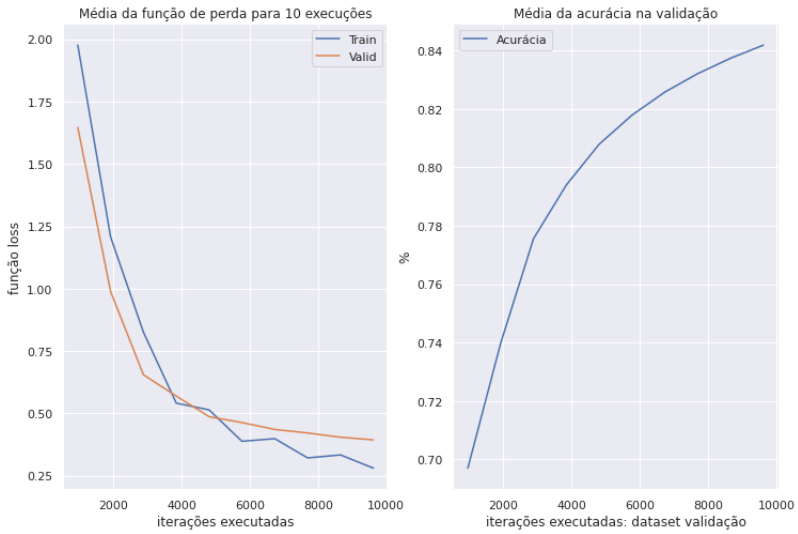
O gráfico na Figura 18 mostra à esquerda os valores do decaimento da função de perda para o treinamento e para a validação durante as dez execuções do experimento 1. À direita está a média da acurácia do modelo submetido a base de validação. Para o tamanho de sequência igual a 128 a função perda convergiu para um mínimo médio de $0,32$ para o treinamento e $0,50$ para a validação. A acurácia média do modelo na base de validação é de $0,83$.

Figura 18 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 1 da Tabela 19.



O gráfico na Figura 19 mostra à esquerda os valores do decaimento da função de perda para o treinamento e para a validação durante as dez execuções do experimento 2. À direita está a média da acurácia do modelo submetido a base de validação. Para o tamanho de sequência igual a 256 a função perda convergiu para um mínimo médio de $0,28$ para o treinamento e $0,39$ para a validação. A acurácia média do modelo na base de validação é de $0,84$.

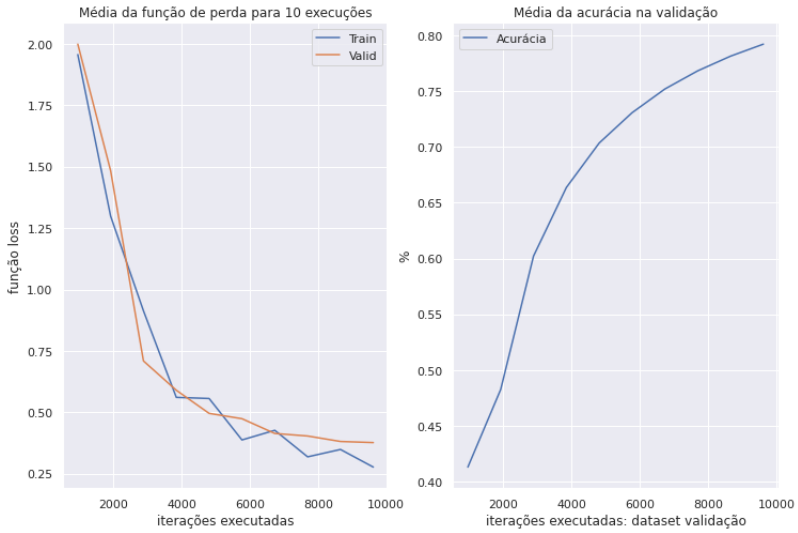
Figura 19 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 2 da Tabela 19.



O gráfico na Figura 20 mostra à esquerda os valores do decaimento da função de

perda para o treinamento e para a validação durante as dez execuções do experimento 3. À direita está a média da acurácia do modelo submetido a base de validação. Para o tamanho de sequência igual a 512 a função perda convergiu para um mínimo médio de $0,31$ para o treinamento e $0,40$ para a validação. A acurácia média do modelo na base de validação é de $0,77$.

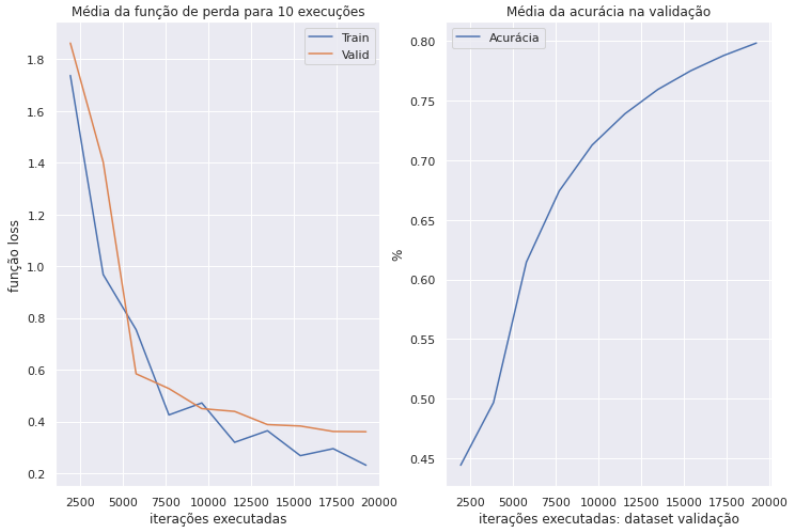
Figura 20 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 3 da Tabela 19.



A acurácia média do modelo na base de dados de validação para o tamanho de sequência 512 não ficou tão boa quando comparado à acurácia dos outros tamanhos. Com o objetivo de analisarmos se no tamanho de sequência igual a 512 há uma associação entre o tamanho do bach size e a acurácia, efetuamos dois experimentos para este tamanho de sequência e com os valores de *batch size* igual a 4 e igual a 2.

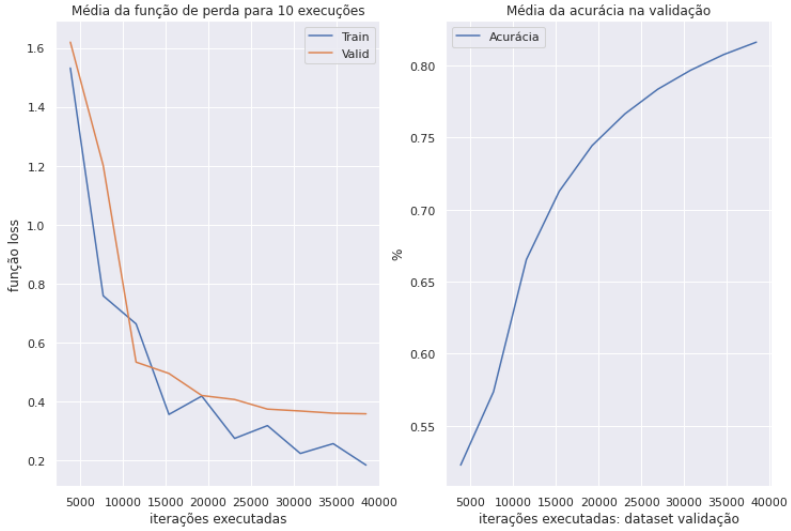
O gráfico na Figura 21 mostra à esquerda a média dos valores do decaimento da função de perda para o treinamento e para a validação durante as dez execuções do experimento 3 com valor de *batch size* igual a 4. À direita está a média da acurácia do modelo submetido a base de validação. Para este experimento a função perda convergiu para um mínimo médio de $0,23$ para o treinamento e $0,36$ para a validação. A acurácia média do modelo na base de validação é de $0,80$.

Figura 21 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 3 da Tabela 19 com valor de *batch size* = 4.



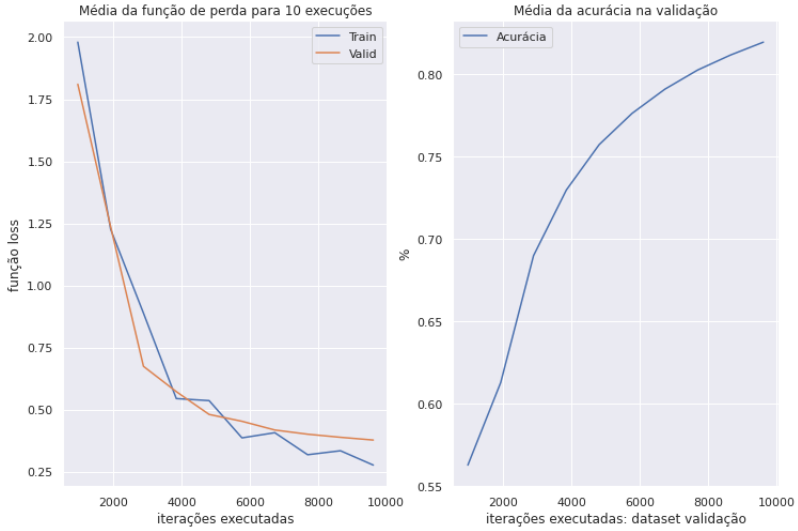
O gráfico na Figura 22 mostra à esquerda a média dos valores do decaimento da função de perda para o treinamento e para a validação durante as dez execuções do experimento 3 com valor de *batch size* igual a 2. À direita está a média da acurácia do modelo submetido a base de validação. Para este experimento a função perda convergiu para um mínimo médio de $0,18$ para o treinamento e $0,36$ para a validação. A acurácia média do modelo na base de validação é de $0,82$.

Figura 22 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 3 da Tabela 19 com valor de *batch size* = 2.



O gráfico na Figura 23 mostra à esquerda a média dos valores do decaimento da função de perda para o treinamento e para a validação durante as dez execuções do experimento 4. À direita está a média da acurácia do modelo submetido a base de validação. Para o tamanho de sequência igual a 394 a função perda convergiu para um mínimo médio de $0,28$ para o treinamento e $0,38$ para a validação. A acurácia média do modelo na base de validação é de $0,82$.

Figura 23 – Gráficos dos valores da função de perda e da acurácia durante a execução das dez repetições do ajuste fino para o experimento 4 da Tabela 19.



Em todos os experimentos a função de perda conseguiu convergir para um valor mínimo que variou entre as médias de 0,18 e 0,32 para o treinamento e 0,36 e 0,50 para a validação. A acurácia média variou entre 0,77 e 0,82, porém se considerarmos o melhor resultado para os experimentos de tamanho de sequência igual a 512 a acurácia média se mantém entre $0,82$ e $0,84$.

Para o tamanho de sequência igual 512 e somente analisando graficamente, há um indicativo de associação entre o tamanho do *batch size* e a acurácia do modelo de validação. Para *batch size* menores o valor da acurácia é maior. O melhor resultado de acurácia na validação foi de $0,82$ com *batch size* igual a 2, depois o de $0,80$ para *batch size* igual a 4 e $0,77$ para *batch size* igual a 8.

5 ANÁLISE DOS RESULTADOS

Neste capítulo analisamos os resultados dos experimentos efetuados para cada um dos classificadores aplicados ao conjunto de testes. Observamos o desempenho geral dos classificadores através das métricas: acurácia, acurácia balanceada, precisão, revocação e Cohen Kappa score (AKOSA, 2017).

Optamos pela acurácia balanceada, pois apesar de todas as classes terem quantidade próximas de exemplos, a classe ciência possui quantidade inferior as demais. A acurácia simples mede a proporção de classificações corretas feita por cada modelo, sendo que com classes desbalanceadas o resultado tenderá para a classe com mais exemplos. Os cálculos da acurácia balanceada incidem sobre a taxa de verdadeiro positivo e a taxa de verdadeiro negativo ($\frac{1}{2}\left(\frac{VP}{VP+FN}+\frac{VN}{VN+FP}\right)$).

A precisão nos informa a taxa de acerto de todas as classes e a revocação (taxa de verdadeiro positivo) nos mostra o quanto o modelo consegue diferenciar as classes, isto é, o quanto de cada classe foi identificada corretamente.

O Cohen Kappa informa quão melhor um classificador está se saindo em relação ao desempenho de um classificador que obtém as classes aleatoriamente de acordo com a frequência de cada classe, isto é, utiliza-se para para testar a confiabilidade dos modelos. (LANDIS; KOCH, 1977) interpreta estes índices com força moderada quando o valor é superior a 0,61 e perfeito quando o valor é superior a 0,81.

5.1 Classificadores

5.1.1 Multinomial Naive-Bayes

A Tabela 20 apresenta o valor das métricas de avaliação para o Multinomial Naive-Bayes. A métrica de acurácia balanceada é de $82,63$ e a de acurácia $82,70\%$. O modelo conseguiu ter bons índices gerais para identificação correta das categorias com $83,36\%$ de precisão e $82,70\%$ de revocação.

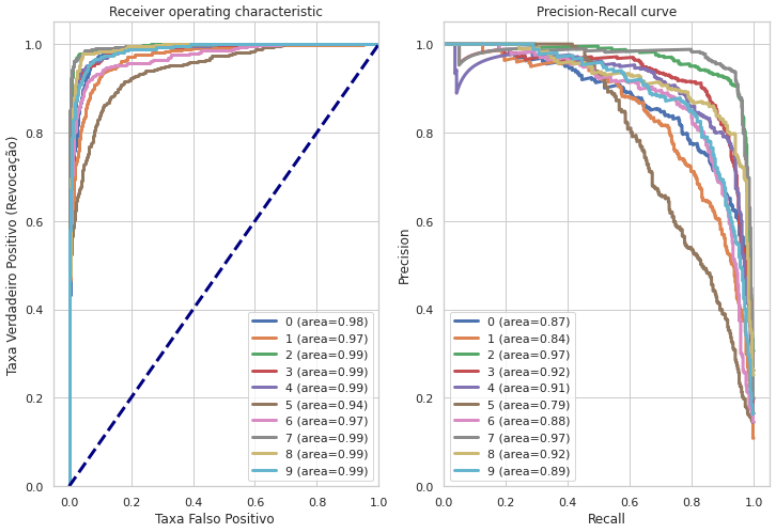
Tabela 20 – Métricas de Precisão, Revocação e F1-Score por categoria para o experimento com o classificador Multinomial Naive-Bayes. Mostramos também a acurácia, acurácia balanceada e o Cohen Kappa score deste modelo.

Classe	Precisão	Revocação	F1-score
política	$71,79\%$	$87,50\%$	$78,87\%$
economia	$77,37\%$	$73,50\%$	$75,38\%$
esporte	$92,35\%$	$89,60\%$	$90,95\%$
mundo	$85,98\%$	$86,79\%$	$86,38\%$
ilustrada	$75,64\%$	$90,84\%$	$82,54\%$
mídia	$91,77\%$	$55,33\%$	$69,04\%$
tecnologia	$79,42\%$	$81,39\%$	$80,39\%$
educação	$92,09\%$	$93,66\%$	$92,87\%$
saúde	$83,20\%$	$89,55\%$	$86,26\%$
ciência	$83,92\%$	$78,10\%$	$80,91\%$
Métricas Gerais			
Acurácia	$82,70\%$		
Acurácia balanceada	$82,63\%$		
Cohen Kappa Score	$80,75\%$		
Precisão	$83,36\%$		
Revocação	$82,70\%$		

Analisando o desempenho do modelo ao classificar individualmente cada categoria, a que o modelo melhor classificou corretamente foi esporte com precisão igual a $92,35\%$. Também obteve uma boa taxa de verdadeiro positivo (revocação) igual $89,60\%$. A categoria mídia também teve um bom desempenho na precisão, porém a taxa de verdadeiro positivo não foi tão boa, $55,33\%$. Analisando a métrica f1-score percebemos que o classificador consegue identificar bem as categorias, pois os valores são superiores a 75%, exceto mídia que possui um índice não tão bom, conforme observado na revocação.

Na Figura 24 temos a curva da Precisão-Revocação - no lado direito - e curva ROC (Receiver operating characteristics) - no lado esquerdo -, sendo que na legenda os números de 0 a 9 se referem, respectivamente, as categorias: política, economia, esporte, mundo, ilustrada, mídia, tecnologia, educação, saúde e ciência. Graficamente confirmamos que mídia não possui um bom desempenho, pois é a categoria com menor área sob a curva no gráfico Precisão-Revocação, devido ao baixo valor da revocação.

Figura 24 – Gráficos com resultado do classificador Multinomial Naive-Bayes. O da esquerda mostra a curva ROC para cada uma das categorias e o da direita mostra a curva de Precisão-Revocação também para cada uma das categorias.



No ROC, todas as categoria possuem área sob a curva superior ou igual a $0,94$ e o índice de Cohen Kappa possui valor de $80,75\%$. A partir destes valores temos boas evidências que o classificador Multinomial de Naive-Bayes é um bom classificador. Porém, o gráfico de Precisão-Revocação mostra que ainda há espaço para melhorarmos os resultados, pois as curvas de mídia, economia e política são puxadas um pouco para baixo indicando que ou a precisão ou a revocação podem ser melhoradas.

5.1.2 *Random Forest*

A Tabela 21 mostra o resultado da avaliação do classificador *Random Forest*. O classificador obteve $82,58\%$ para a acurácia balanceada, sendo que a acurácia obteve um valor superior, $83,03\%$. Obteve também bons índices gerais de precisão ($82,94\%$) e revocação ($82,88\%$).

Tabela 21 – Métricas de Precisão, Revocação e F1-Score por categoria para o experimento com o classificador *Random Forest*. Mostramos também a acurácia, acurácia balanceada e o Cohen Kappa score deste modelo,

Classe	Precisão	Revocação	F1-score
política	75, 81%	85, 68%	80, 44%
economia	79, 23%	72, 50%	75, 72%
esporte	89, 58%	95, 79%	92, 58%
mundo	85, 09%	82, 08%	83, 55%
ilustrada	81, 29%	86, 26%	83, 70%
mídia	84, 84%	72, 21%	78, 02%
tecnologia	81, 16%	80, 15%	80, 65%
educação	89, 95%	96, 10%	92, 92%
saúde	78, 68%	87, 57%	82, 89%
ciência	84, 86%	67, 52%	75, 20%
Métricas Gerais			
Acurácia	83, 03%		
Acurácia balanceada	82, 58%		
Cohen Kappa Score	81, 12%		
Precisão	82, 94%		
Revocação	82, 88%		

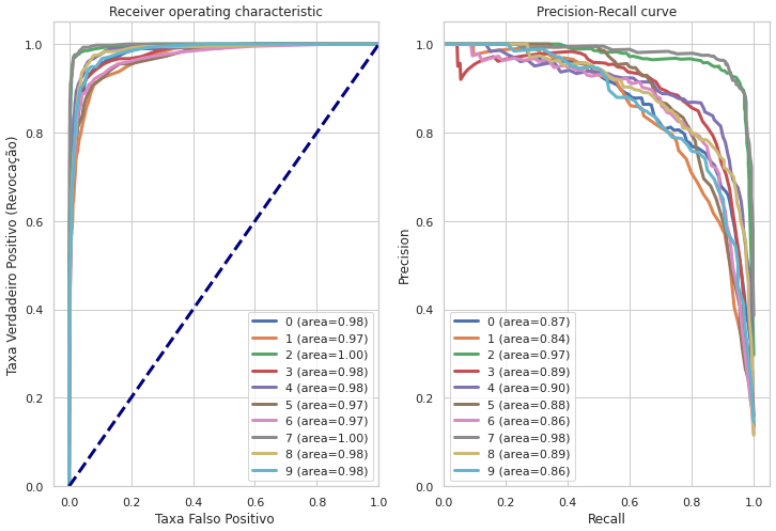
O classificador obteve precisão acima de 80% para todas as classes, exceto política, economia e saúde, que ficaram entre 75% e 80%. A revocação mostra bom indicador, exceto para ciência, que é a classe minoritária, que obteve 67, 07%. Os valores do índice F1-Score também mostram que este classificador consegue separar bem as categorias, sendo o menor valor 75, 20 (ciência).

Nos gráficos da Figura 25 os números de 0 a 9 se referem, respectivamente, as categorias: política, economia, esporte, mundo, ilustrada, mídia, tecnologia, educação, saúde e ciência.

Analisando a precisão e a revocação nas curvas de Precisão-Revocação e ROC na Figura 25, em conjunto com o índice de Cohen Kappa que é de 81, 12%, percebemos indícios que o classificador *Random Forest* é um bom classificador para as notícias. No gráfico Precisão-Revocação, todas as curvas estão bem postadas na parte superior esquerda, sendo que as categorias educação e esporte são as que o classificador obteve os melhores resultados.

Analisando os valores da área sob a curva para os dois gráficos, temos que tanto na curva ROC quanto na Precisão-Revocação os valores são muito bons. Na Precisão-Revocação a menor área possui valor de 0, 84 e na ROC as áreas estão próximas do valor ideal que é 1, isto é, o modelo consegue efetuar uma boa separabilidade das classes.

Figura 25 – Gráficos com resultado do classificador *Random Forest*. O da esquerda mostra a curva ROC para cada uma das categorias e o da direita mostra a curva de Precisão-Revocação também para cada uma das categorias.



5.1.3 **Comparação entre Multinomial Naive-Bayes e *Random Forest***

Considerando o resultado de todas as métricas em conjunto, o classificador *Random Forest* e o Multinomial Naive-Bayes obtiveram valores muito próximos.

Tabela 22 – Valores das métricas dos modelos Multinomial Naive-Bayes e *Random Forest*

Modelo	Acurácia	Acurácia Balanceada	Cohen Kappa	Precisao	Revocação
M. Naive-Bayes	82,70%	82,63%	80,75%	83,35%	82,70%
Random Forest	83,03%	82,58%	81,12%	82,94%	82,88%

A Tabela 22 mostra a acurácia, a acurácia balanceada, o Cohen Kappa, a precisão e revocação dos dois classificadores. O valor da acurácia balanceada está bem próxima para os dois classificadores, porém o Cohen Kappa do Random-Forest é superior.O *Random Forest* consegue classificar melhor novos textos em cada uma das classes do nosso problema. Esta distinção pode ser percebida graficamente quando olhamos para os gráficos da curva de Precisão-Revocação de cada modelo (24, 25), as curvas das categorias do *Random Forest* estão melhor posicionadas (possui maior área sob a curva) que as curvas das categorias do Multinomial Naive-Bayes.

Também observamos nos gráficos das curvas Precisão-Revocação e ROC que categorias que não foram tão bem percebidas pelo Multinomial Naive-Bayes (mídia, política e economia), o *Random Forest* as conseguiu identificar melhor.

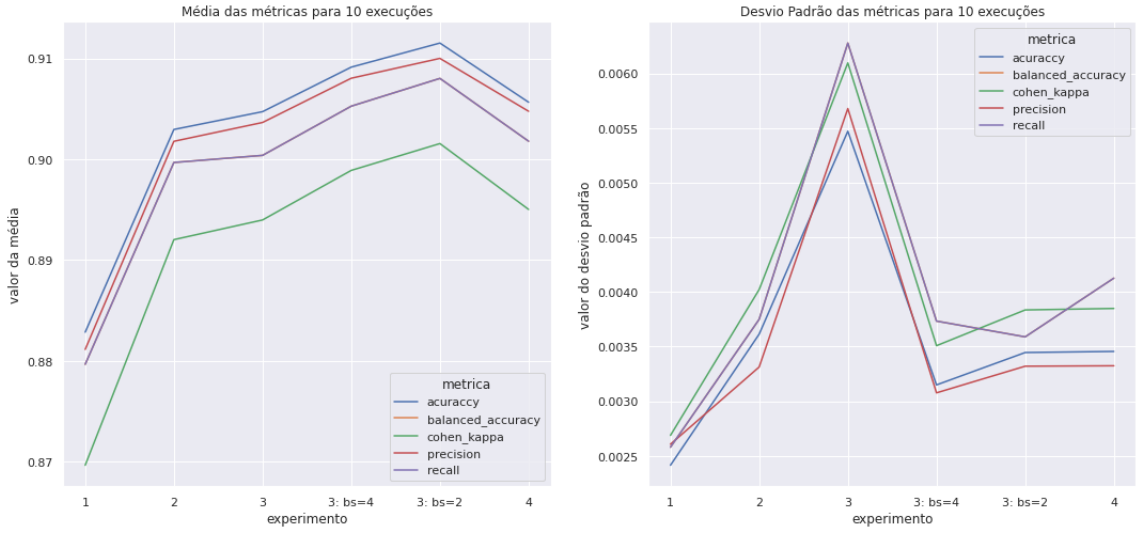
5.2 BERTimbau

Nesta seção mostramos os resultados para cada um dos experimentos efetuados com o classificador BERTimbau e descritos na subseção 4.4.2. Também comparamos os resultados obtidos entre os experimentos.

As tabelas com os resultados de cada uma das métricas para cada um dos experimentos executados estão listadas no anexo Tabelas com Resultados dos experimentos com BERTimbau em A.

A Figura 26 mostra o desempenho alcançado por cada experimento em relação aos índices de acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação. O gráfico à esquerda apresenta o valor médio obtido, no conjunto de testes, entre as dez execuções de cada um dos modelos treinados em cada experimento. O gráfico à direito mostra o desvio padrão dos valores obtidos nestas dez execuções de cada modelo. O valor da acurácia balanceada e da revocação possuem valores muito próximos, então a curva da revocação sobrepõe a da acurácia balanceada.

Figura 26 – Comparativo dos valores das métricas entre os experimentos realizados. À esquerda está o gráfico com a média dos índices obtidos entre as 10 execuções de cada experimento. À direita está o gráfico com o desvio padrão destes índices.Acurácia balanceada e revocação possuem valores muito próximos.



O melhor resultado foi obtido com o modelo treinado com tamanho de sequência 512 e *batch size* igual à 2, sendo que os outros modelos obtiveram índices bem próximos do melhor resultado. O modelo que obteve menor desempenho foi o de tamanho de sequência 128. Os índices de acurácia, acurácia balanceada, precisão e revocação foram menores que $89,00\%$.

Uma possível explicação para este cenário é que na base de dados de notícias, mais que 25% dos textos possuem tamanho maior que 236 palavras e ao efetuar o treinamento o BERTimbau considerou as 128 subpalavras obtidas do início de cada texto e isto pode afetar o desempenho do modelo, pois as palavras que compõem o início do texto de algumas notícias pode não determinar de forma clara o assunto, além de conter palavras ou frases que semanticamente pode representar outro assunto.

As métricas acurácia e acurácia balanceada obtiveram valores próximos em todos os experimentos, sendo que a acurácia obteve valores superiores a acurácia balanceada. Esta proximidade já era esperada, pois os conjuntos de dados de treinamento e teste possuem quantidades de exemplos aproximadamente iguais entre as categorias, com exceção da categoria ciência que possuem uma quantidade um pouco menor que as demais.

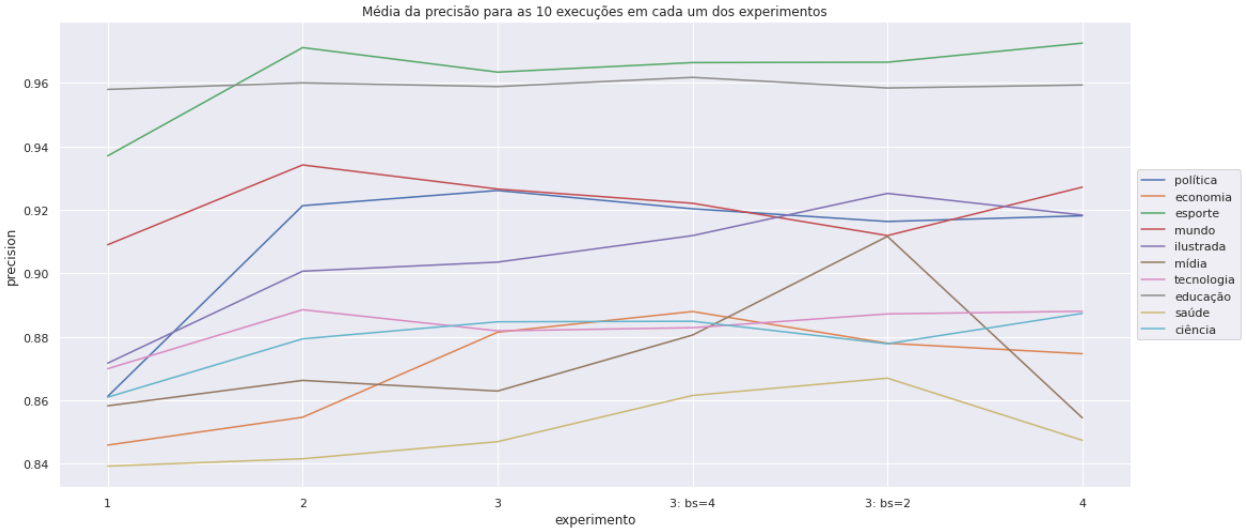
Para o índice Cohen Kappa, todos os modelos atingiram valores superiores a 0,87, sendo que o modelo que obteve melhor resultado foi o modelo com tamanho de sequência 512 e *batch size* igual à 2, cujo valor foi superior a $0,90$.

Os índices de precisão e revocação obtiveram valores altos para todos os experimentos indicando que, de forma geral, o modelo conseguiu separar bem as categorias. Faremos uma avaliação mais precisa destes índices comparando o resultado obtido de forma separada em cada uma das categorias.

O gráfico do desvio padrão nos mostra que os valores dos índices obtidos em cada uma das dez execuções para cada um dos experimentos foram aproximadamente iguais, isto é, com baixa dispersão. As maiores dispersões entre os valores dos índices foram obtidas para o experimento com tamanho de sequência 512 e *batch size* igual à 2

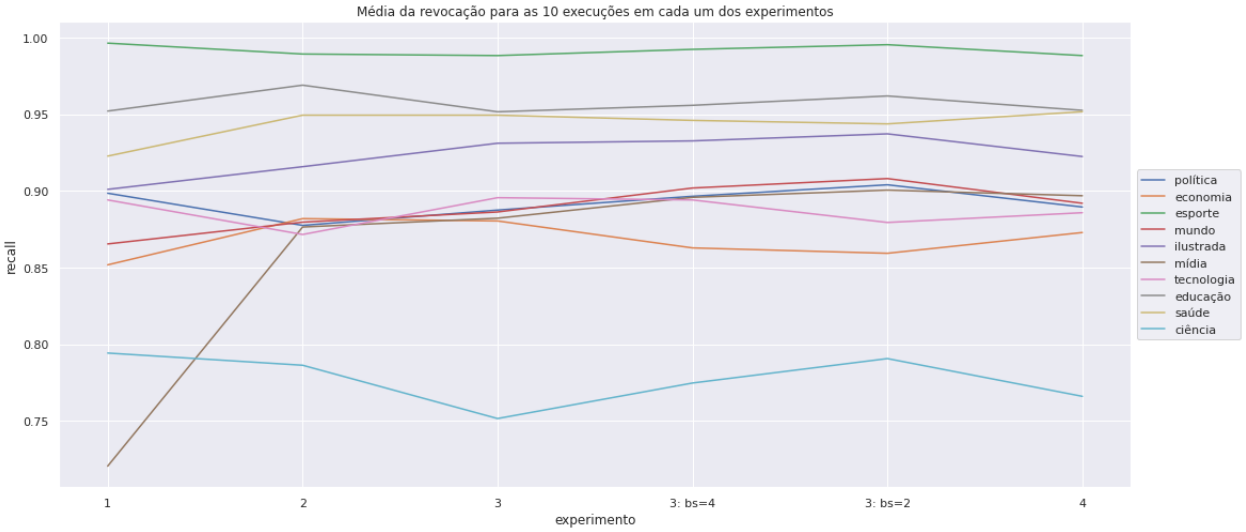
A Figura 27 mostra o valor médio da precisão em cada uma das categorias para cada experimento. Analisando graficamente, os modelos de todos os experimentos classificaram melhor a sequência de subpalavras das categorias esporte e educação. A categoria que obteve índices mais baixos foi saúde, porém estes índices foram superiores a 84% de acerto. A categoria mídia foi a que mais variou a precisão entre os experimentos. Ela obteve o melhor índice para tamanho de sequência igual a 512 com *batch size* igual a 2 e o pior índice para tamanho de sequência igual a 394.

Figura 27 – Comparativo da Precisão em cada categoria entre os experimentos realizados.



Analisando o gráfico comparativo da métrica revocação para cada experimento, na Figura 28, observamos que as categorias que tiveram índices superiores a 90% foram esporte, educação, saúde e ilustrada, sendo que esporte foi a que teve índices próximos a 100% de classificações corretas de verdadeiro positivo em todos os experimentos. A categoria mídia não obteve um bom índice no experimento 1, porém nos demais experimentos obteve bons índices, superiores a 85%. A categoria ciência não teve desempenho tão bom quanto as demais, ela obteve índices entre 75% e 80% em todos os modelos.

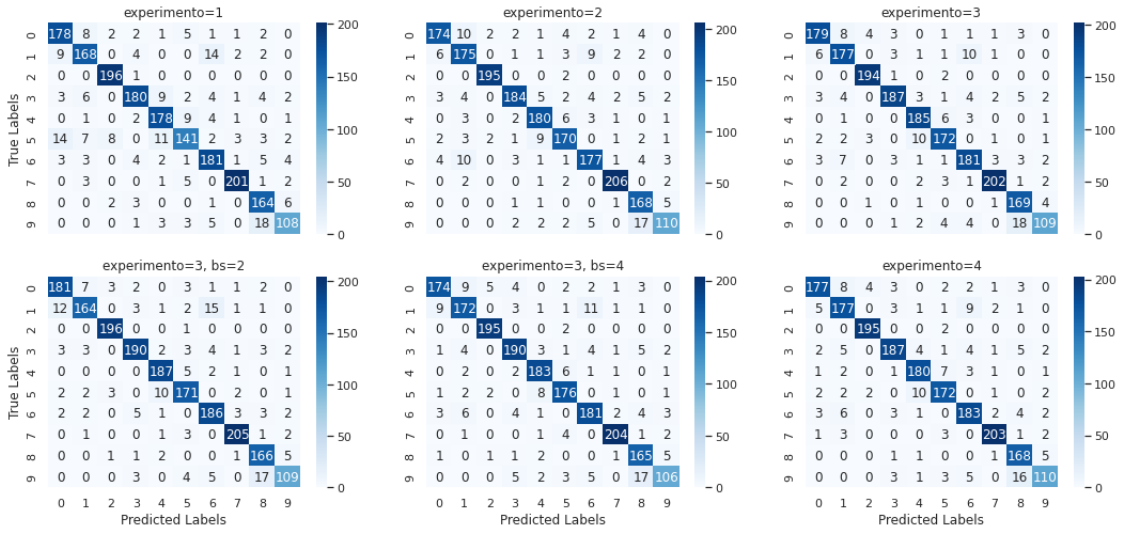
Figura 28 – Comparativo da Revocação em cada categoria entre os experimentos realizados.



Com o objetivo de verificar quantitativamente os valores de classificações feitas de

forma assertiva para cada uma das classes e os valores de classificações feitas em outras classes diferente da real, isto é, os falso negativos e os falso positivos, plotamos na Figura 29 a matriz de confusão do primeiro modelo gerado, entre os dez testados, para cada um dos experimentos efetuados.

Figura 29 – Matriz de confusão gerada para o primeiro modelo, entre os dez efetuados, para cada um dos experimentos.



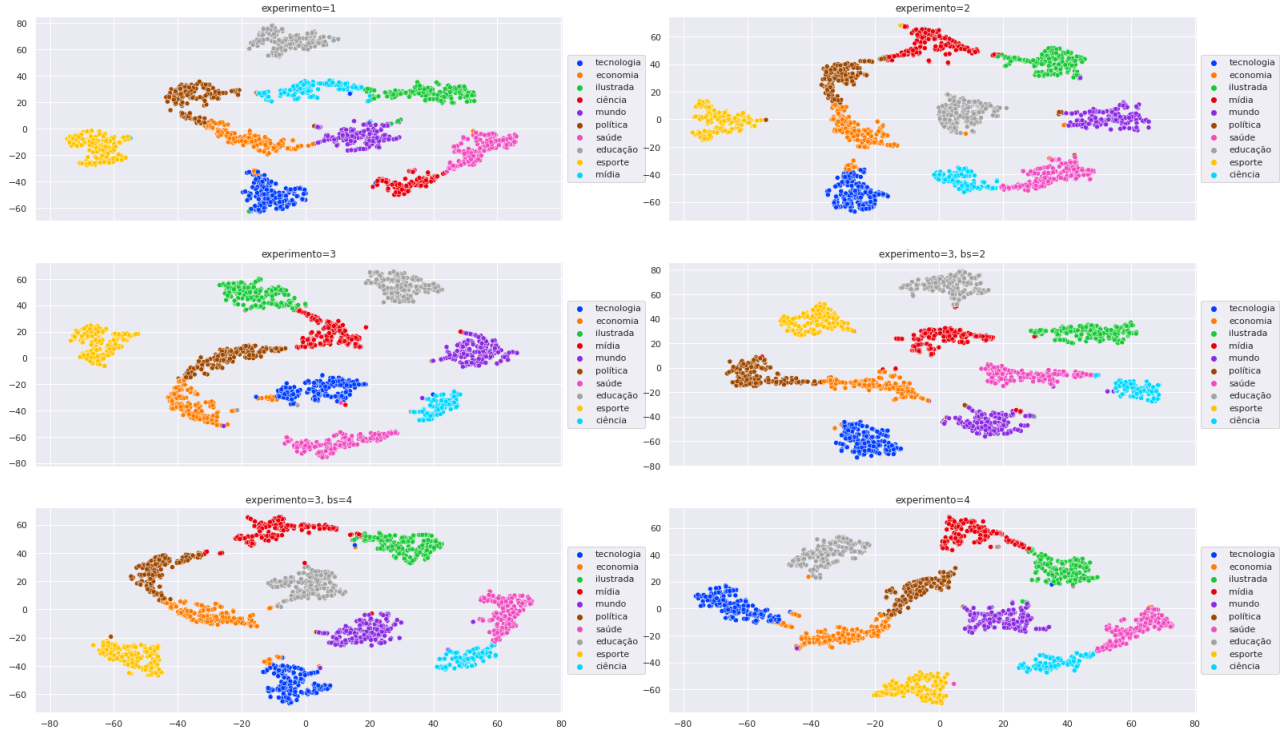
A escala de cores nas matrizes indica o número de exemplos classificados corretamente no conjunto de testes para cada categoria. Quanto mais escuro a cor na diagonal da matriz, indica que o número referenciado está próximo do número real da classe e quanto mais claro indica que o classificador errou mais para aquela classe. Os números de 0 a 9 em cada eixo da matriz indicam as categorias na ordem: política, economia, esporte, mundo, ilustrada, mídia, tecnologia, educação, saúde, ciência.

Podemos observar que o modelo errou mais exatamente na classe com menos exemplo, a de ciências, como observado nos resultados do índice de revocação. Também observamos que nas demais classes há um equilíbrio na taxa de verdadeiro positivo.

Com o objetivo de verificar as fronteiras de separação entre as categorias geradas pelo BERTimbau, recuperamos para o conjunto de teste as sequências geradas pela última camada do modelo BERTimbau, isto é, o *embedding* de pesos de relevância calculados pela última camada e que possui dimensão R^{10}. Obtemos estes valores para o primeiro modelo, entre os dez gerados, para cada um dos experimentos.

Em seguida, utilizando o algoritmo de redução de dimensionalidade t-sne (t-Distributed Stochastic Neighbor Embedding) reduzimos a dimensão do *embedding* de saída do modelo para duas dimensões. Plotamos o resultado no gráfico da Figura 30.

Figura 30 – Redução de dimensionalidade com a técnica t-SNE no *embedding* de pesos da última camada do classificador de cada experimento.



Observamos que as fronteiras entre as categorias de todos os experimentos possuem pontos de sobreposição apesar de todas as categorias estarem bem definidas. Isto se dá, pois as notícias muitas das vezes tratam dos assuntos de forma integrada. Podemos observar isto em política e economia, ciência e saúde, ilustrada e mídia e tecnologia e economia que possuem fronteiras com sobreposição em todos os experimentos.

5.3 Comparação entre os modelos clássicos e os experimentos com BERTimbau

A Tabela 23 mostra os valores das métricas acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação de todos os modelos experimentados. O modelo do BERTimbau com tamanho de sequência igual a 512 obteve o melhor desempenho em todas as métricas. O segundo melhor modelo foi o BERTimbau com tamanho 394.

O $BERTimbau_{BASE}$ melhorou os resultados alcançados com os algoritmos clássicos Naives-Bayes e *Random Forest* e mostrou que consegue identificar todas as 10 classes com uma boa taxa de acerto tanto nos índices de acurácia quanto nos índices de precisão e revocação. A classe minoritária ciência também conseguiu ser identificada com uma boa taxa média de precisão, acima de 86% em todos os experimentos.

Tabela 23 – Valores das métricas dos experimentos realizados.

Modelo	Acurácia	Acurácia Balanceada	Cohen Kappa	Precisao	Revocação
M. Naive-Bayes	82,70%	82,63%	80,75%	83,35%	82,70%
Random Forest	83,03%	82,58%	81,12%	82,94%	82,88%
BERTimbau tamanho 128	88,29%	89,97%	86,97%	88,11%	87,97%
BERTimbau tamanho 256	90,30%	87,97%	89,20%	90,18%	89,97%
BERTimbau tamanho 512 bs=8	90,47%	90,03%	89,40%	90,37%	90,04%
BERTimbau tamanho 512 bs=4	91,00%	90,53%	89,90%	**90,80%**	90,53%
BERTimbau tamanho 512 bs=2	**91,15%**	**90,80%**	**90,16%**	90,00%	**90,80%**
BERTimbau tamanho 394	90,57%	90,18%	90,00%	90,48%	90,18%

5.4 Considerações Finais

Os modelos clássicos atingiram resultados razoáveis, apresentando bons indicadores de acurácia balanceada, Cohen Kappa, precisão e revocação. A categoria de notícias com menos exemplos, ciência, teve bons índices de precisão em ambos indicadores, porém o índice de revocação no *Random Forest* foi abaixo de 70%.

Todos os experimentos com o BERTimbau atingiram valores superiores aos algoritmos clássicos, sendo que o experimento com sequência de tamanho 128 foi o que teve menor desempenho. Este fato pode ser explicado pela estratégia adotada, pois ao considerarmos o texto para compor o vetor de representação, selecionamos a quantidade de subpalavras iniciais de cada texto e do tamanho da sequência escolhida. Como menos de 25% dos textos possuem sequencias menores que 128, é provável que, para muitos textos, as subpalavras iniciais não consigam representar sequências que permitem o modelo identificar as categorias com uma alta precisão.

Durante os experimentos com o tamanho de sequência igual a 512, observamos que quanto menor o tamanho do *batch size* melhor é o desempenho do modelo quanto aos indicadores considerados. A única variável que foi alterada entre os experimentos efetuados para este tamanho de sequência foi o tamanho do *batch size*. A explicação é que com taxas de *batch sizes* menores o treinamento é efetuado em mais iterações permitindo que o classificador aprenda melhor a separar as classes.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho treinamos os classificadores de textos jornalísticos utilizando os algoritmos de abordagem clássica e o modelo BERT pré-treinado para a língua portuguesa nomeado de BERTimbau. Para a primeira abordagem, foram realizados experimentos com o Multinomial Naive-Bayes e o *Random Forest* e com o $BERTimbau_{BASE}$ fizemos experimentos com tamanhos de sequências variadas: 128, 256, 512 e 394. Comparamos os resultados obtidos entre todos os classificadores. Os melhores resultados foram obtidos com o BERTimbau para as sequências de maior tamanho: 512 e 394. Estes classificadores obtiveram índices de acurácia balanceada e precisão superiores a 90%. A classe com menor representatividade na amostra obteve índices de precisão superiores a 88%.

O modelo com menor tamanho de sequência não obteve desempenho tão bom quanto aos modelos com tamanho de sequência maiores, sendo que quase 75% dos textos dos conjuntos de dados de treinamento e testes possuem textos com até 596 palavras. Observamos nos resultados que, para o nosso conjunto de dados, quanto maior a sequência de palavras e menor o *batch size*, melhor é o desempenho do classificador. O conjunto de dados possui muitos textos com quantidade de palavras superior a mil, porém o BERTimbau é limitado em tamanho de sequência igual a 512.

A partir destes resultados concluímos que os experimentos conseguiram atingir o objetivo, inicialmente proposto, em classificar textos jornalísticos de acordo com a sessão em que foram publicados. A metodologia utilizada neste trabalho pode contribuir para a classificação de textos em língua portuguesa e que possuem outros domínios de conhecimento.

Para melhorar o desempenho dos classificadores para textos com tamanho grande, poderíamos treinar um classificador utilizando parágrafos selecionados de partes específicas de cada um destes textos grandes, como o parágrafo inicial e final. Outro caminho interessante e que poderia gerar bons resultados seria utilizar outras abordagens de *autoenconders* que utiliza grandes bases de dados com textos não rotulados para pré-treinamento, como o ULMFit (HOWARD; RUDER, 2018), o GPT-1 (RADFORD et al., 2018) e o GPT-2 (FANG et al., 2021). Estes dois últimos são da família de *autoenconders* da Open-AI [1].

[1] https://openai.com/blog/better-language-models/ acessado em 07/12/2021

Anexos

ANEXO A – TABELAS COM RESULTADOS DOS EXPERIMENTOS COM BERTIMBAU

Mostramos neste Anexo as Tabelas 24,25, 26, 27, 28 e 29 com os valores médios das métricas de avaliação para os experimentos 1, 2, 3 e 4. Mostramos também os valores para as métricas dos experimentos 3 com variação do tamanho do *batch size*. Cada tabela contém o nome da classe, o valor médio e o desvio padrão médio das métricas precisão, revocação e f1-score obtidos nas dez execuções de cada experimento durante o processo de avaliação do modelo.

Mostramos também as médias e os desvio padrão calculados para as métricas gerais para cada um dos dez modelos testados em cada um dos experimentos.

Tabela 24 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 1 com o classificador *BERTimbau* com tamanho de texto igual a 128. Mostramos também a acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	86,13%	0,011	89,85%	0,015	87,93%	0,006
economia	84,59%	0,022	85,18%	0,017	84,84%	0,005
esporte	93,70%	0,006	99,64%	0,002	96,58%	0,004
mundo	90,90%	0,010	86,54%	0,010	88,66%	0,006
ilustrada	87,17%	0,007	90,10%	0,009	88,61%	0,005
mídia	85,83%	0,017	72,04%	0,019	78,31%	0,012
tecnologia	87,00%	0,010	89,41%	0,013	88,18%	0,007
educação	95,80%	0,003	95,21%	0,007	95,50%	0,004
saúde	83,93%	0,015	92,27%	0,016	87,88%	0,004
ciência	86,10%	0,008	79,42%	0,013	82,62%	0,008
Métricas Gerais						
Métrica	Média	DP				
Acurácia	88,29%	0,002				
Acurácia balanceada	87,97%	0,003				
Cohen Kappa Score	86,97%	0,003				
Precisão	88,11%	0,003				
Revocação	87,98%	0,003				

Tabela 25 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 2 com o classificador *BERTimbau* com tamanho de texto igual a 256. Mostramos também a acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	92,13%	0,009	87,75%	0,012	89,88%	0,008
economia	85,47%	0,014	88,19%	0,008	86,80%	0,007
esporte	97,11%	0,005	98,93%	0,002	98,01%	0,003
mundo	93,42%	0,007	87,96%	0,011	90,60%	0,005
ilustrada	90,07%	0,007	91,58%	0,007	90,82%	0,005
mídia	86,63%	0,019	87,64%	0,009	87,13%	0,012
tecnologia	88,86%	0,004	87,16%	0,014	87,99%	0,008
educação	96,00%	0,004	96,90%	0,003	96,45%	0,003
saúde	84,16%	0,011	94,94%	0,010	89,22%	0,004
ciência	87,94%	0,009	78,62%	0,012	83,01%	0,007
Métricas Gerais						
Métrica	Média	DP				
Acurácia	90,30%	0,004				
Acurácia balanceada	89,97%	0,004				
Cohen Kappa Score	89,20%	0,004				
Precisão	90,18%	0,003				
Revocação	89,97%	0,003				

Tabela 26 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 3 com o classificador *BERTimbau* com tamanho de texto igual a 512 e *batch size* igual a 8. Mostramos também a acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	92,61%	0,015	88,75%	0,009	90,63%	0,009
economia	88,15%	0,011	88,04%	0,018	88,08%	0,009
esporte	96,34%	0,005	98,83%	0,003	97,57%	0,003
mundo	92,66%	0,017	88,63%	0,011	90,58%	0,007
ilustrada	90,36%	0,010	93,11%	0,011	91,71%	0,007
mídia	86,29%	0,034	88,22%	0,023	87,23%	0,027
tecnologia	88,19%	0,008	89,56%	0,011	88,86%	0,004
educação	95,89%	0,003	95,16%	0,004	95,52%	0,003
saúde	84,70%	0,008	94,94%	0,006	89,53%	0,005
ciência	88,48%	0,009	75,14%	0,031	81,24%	0,020
Métricas Gerais						
Métrica	Média	DP				
Acurácia	90,47%	0,005				
Acurácia balanceada	90,03%	0,006				
Cohen Kappa Score	89,40%	0,006				
Precisão	90,37%	0,006				
Revocação	90,04%	0,006				

Tabela 27 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 3 com o classificador *BERTimbau* com tamanho de texto igual a 512 e *batch size* igual a 4. Mostramos também a acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	$92,03\%$	$0,016$	$89,65\%$	$0,016$	$90,81\%$	$0,007$
economia	$88,80\%$	$0,029$	$86,28\%$	$0,024$	$87,45\%$	$0,006$
esporte	$96,64\%$	$0,004$	$99,24\%$	$0,004$	$97,92\%$	$0,003$
mundo	$92,21\%$	$0,012$	$90,19\%$	$0,012$	$91,18\%$	$0,010$
ilustrada	$91,19\%$	$0,019$	$93,27\%$	$0,018$	$92,19\%$	$0,006$
mídia	$88,06\%$	$0,031$	$89,58\%$	$0,020$	$88,76\%$	$0,014$
tecnologia	$88,29\%$	$0,015$	$89,41\%$	$0,017$	$88,82\%$	$0,005$
educação	$96,18\%$	$0,003$	$95,59\%$	$0,006$	$95,88\%$	$0,003$
saúde	$86,15\%$	$0,014$	$94,60\%$	$0,006$	$90,18\%$	$0,009$
ciência	$88,49\%$	$0,006$	$77,46\%$	$0,029$	$82,59\%$	$0,017$
Métricas Gerais						
Métrica	Média	DP				
Acurácia	$91,00\%$	$0,003$				
Acurácia balanceada	$90,53\%$	$0,004$				
Cohen Kappa Score	$89,90\%$	$0,004$				
Precisão	$90,80\%$	$0,003$				
Revocação	$90,53\%$	$0,004$				

Tabela 28 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 3 com o classificador *BERTimbau* com tamanho de texto igual a 512 e *batch size* igual a 2. Mostramos também a acurácia, acurácia balanceada, Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	$91,63\%$	$0,020$	$90,40\%$	$0,008$	$91,00\%$	$0,009$
economia	$87,80\%$	$0,031$	$85,93\%$	$0,024$	$86,79\%$	$0,013$
esporte	$96,65\%$	$0,006$	$99,54\%$	$0,004$	$98,08\%$	$0,003$
mundo	$91,19\%$	$0,018$	$90,81\%$	$0,009$	$90,98\%$	$0,008$
ilustrada	$92,52\%$	$0,011$	$93,72\%$	$0,013$	$93,11\%$	$0,005$
mídia	$91,17\%$	$0,023$	$90,05\%$	$0,014$	$90,58\%$	$0,012$
tecnologia	$88,72\%$	$0,012$	$87,94\%$	$0,029$	$88,29\%$	$0,010$
educação	$95,84\%$	$0,006$	$96,20\%$	$0,004$	$96,02\%$	$0,003$
saúde	$86,70\%$	$0,009$	$94,38\%$	$0,005$	$90,37\%$	$0,005$
ciência	$87,78\%$	$0,009$	$79,06\%$	$0,017$	$83,18\%$	$0,009$
Métricas Gerais						
Métrica	Média	DP				
Acurácia	$91,15\%$	$0,003$				
Acurácia balanceada	$90,80\%$	$0,004$				
Cohen Kappa Score	$90,16\%$	$0,004$				
Precisão	$91,00\%$	$0,003$				
Revocação	$90,80\%$	$0,004$				

Tabela 29 – Valores médios das Métricas e do Desvio Padrão (DP) para Precisão, Revocação e F1-Score por categoria obtidos nas dez execuções do experimento 4 com o classificador *BERTimbau* com tamanho de texto igual a 394. Mostramos também a acurácia, acurácia balanceada, o Cohen Kappa, precisão e revocação deste modelo.

Classe	Precisão		Revocação		F1-score	
	Média	DP	Média	DP	Média	DP
política	91,81%	0,011	88,95%	0,012	90,35%	0,006
economia	87,47%	0,008	87,29%	0,008	87,37%	0,005
esporte	97,25%	0,003	98,83%	0,003	98,04%	0,002
mundo	92,72%	0,008	89,19%	0,007	90,92%	0,004
ilustrada	91,84%	0,011	92,24%	0,009	92,03%	0,006
mídia	85,45%	0,032	89,69%	0,010	87,49%	0,019
tecnologia	88,81%	0,009	88,58%	0,011	88,69%	0,006
educação	95,94%	0,003	95,26%	0,003	95,59%	0,002
saúde	84,74%	0,012	95,17%	0,004	89,65%	0,006
ciência	88,74%	0,009	76,59%	0,028	82,20%	0,019
Métricas Gerais						
Métrica	Média	DP				
Acurácia	90,57%	0,003				
Acurácia balanceada	90,18%	0,004				
Cohen Kappa Score	90,00%	0,003				
Precisão	90,48%	0,003				
Revocação	90,18%	0,004				

REFERÊNCIAS

ABBAS, M. et al. Multinomial naive bayes classification model for sentiment analysis. **IJCSNS Int. J. Comput. Sci. Netw. Secur**, v. 19, n. 3, p. 62–67, 2019.

AKOSA, J. Predictive accuracy: A misleading performance measure for highly imbalanced data. In: **Proceedings of the SAS Global Forum**. [S.l.: s.n.], 2017. v. 12.

BUCK, C.; HEAFIELD, K.; OOYEN, B. V. N-gram counts and language models from the common crawl. In: CITESEER. **LREC**. [S.l.], 2014. v. 2, p. 4.

CAMARGO, S. d. S. Um modelo neural de aprimoramento progressivo para redução de dimensionalidade. 2010.

CASANOVA, E. et al. Tts-portuguese corpus: a corpus for speech synthesis in brazilian portuguese. **Language Resources and Evaluation**, 2021.

CHATURVEDI, S.; PENG, H.; ROTH, D. Story comprehension for predicting what happens next. In: **Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing**. [S.l.: s.n.], 2017. p. 1603–1614.

CHEN, Q. et al. Enhanced lstm for natural language inference. **arXiv preprint arXiv:1609.06038**, 2016.

CONNEAU, A.; LAMPLE, G. Cross-lingual language model pretraining. 2019.

CUI, B. et al. Fine-tune bert with sparse self-attention mechanism. In: **Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)**. [S.l.: s.n.], 2019. p. 3548–3553.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018.

EISENSCHLOS, J. M. et al. Multifit: Efficient multi-lingual language model fine-tuning. **arXiv preprint arXiv:1909.04761**, 2019.

FANG, L. et al. Transformer-based conditional variational autoencoder for controllable story generation. **arXiv preprint arXiv:2101.00828**, 2021.

GRAY, S.; RADFORD, A.; KINGMA, D. P. Gpu kernels for block-sparse weights. **arXiv preprint arXiv:1711.09224**, v. 3, 2017.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HOWARD, J.; GUGGER, S. **Deep Learning for Coders with fastai and PyTorch**. [S.l.]: O'Reilly Media, 2020.

HOWARD, J.; RUDER, S. Universal language model fine-tuning for text classification. **arXiv preprint arXiv:1801.06146**, 2018.

JI, Y.; EISENSTEIN, J. Discriminative improvements to distributional sentence similarity. In: **Proceedings of the 2013 conference on empirical methods in natural language processing**. [S.l.: s.n.], 2013. p. 891–896.

JOHNSON, R.; ZHANG, T. Supervised and semi-supervised text categorization using lstm for region embeddings. In: PMLR. **International Conference on Machine Learning**. [S.l.], 2016. p. 526–534.

_____. Deep pyramid convolutional neural networks for text categorization. In: **Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. [S.l.: s.n.], 2017. p. 562–570.

LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. **biometrics**, JSTOR, p. 159–174, 1977.

LE, H. et al. Flaubert: Unsupervised language model pre-training for french. **arXiv preprint arXiv:1912.05372**, 2019.

LIU, P. J. et al. Generating wikipedia by summarizing long sequences. **arXiv preprint arXiv:1801.10198**, 2018.

LIU, X.; DUH, K.; GAO, J. Stochastic answer networks for natural language inference. **arXiv preprint arXiv:1804.07888**, 2018.

LUONG, M.-T. et al. Addressing the rare word problem in neural machine translation. **arXiv preprint arXiv:1410.8206**, 2014.

MA, X.; HOVY, E. End-to-end sequence labeling via bi-directional lstm-cnns-crf. **arXiv preprint arXiv:1603.01354**, 2016.

MAATEN, L. Van der; HINTON, G. Visualizing data using t-sne. **Journal of machine learning research**, v. 9, n. 11, 2008.

MARTIN, L. et al. Camembert: a tasty french language model. **arXiv preprint arXiv:1911.03894**, 2019.

MATSUBARA, E. T.; MARTINS, C. A.; MONARD, M. C. Pretext: Uma ferramenta para pré-processamento de textos utilizando a abordagem bag-of-words. **Techinical Report**, v. 209, n. 4, p. 10–11, 2003.

MCCANN, B. et al. Learned in translation: Contextualized word vectors. **arXiv preprint arXiv:1708.00107**, 2017.

MELAMUD, O.; GOLDBERGER, J.; DAGAN, I. context2vec: Learning generic context embedding with bidirectional lstm. In: **Proceedings of the 20th SIGNLL conference on computational natural language learning**. [S.l.: s.n.], 2016. p. 51–61.

MIYATO, T.; DAI, A. M.; GOODFELLOW, I. Adversarial training methods for semi-supervised text classification. **arXiv preprint arXiv:1605.07725**, 2016.

PAPINENI, K. et al. Bleu: a method for automatic evaluation of machine translation. In: **Proceedings of the 40th annual meeting of the Association for Computational Linguistics**. [S.l.: s.n.], 2002. p. 311–318.

PETERS, M. E. et al. Deep contextualized word representations. **arXiv preprint arXiv:1802.05365**, 2018.

PONTI, M. et al. A decision cognizant kullback–leibler divergence. **Pattern Recognition**, Elsevier, v. 61, p. 470–478, 2017.

PONTI, M. A. et al. Training deep networks from zero to hero: avoiding pitfalls and going beyond. **arXiv preprint arXiv:2109.02752**, 2021.

RADFORD, A. et al. Improving language understanding with unsupervised learning. Technical report, OpenAI, 2018.

RAMOS, J. et al. Using tf-idf to determine word relevance in document queries. In: CITESEER. **Proceedings of the first instructional conference on machine learning**. [S.l.], 2003. v. 242, n. 1, p. 29–48.

RODRIGUEZ-GALIANO, V. F. et al. An assessment of the effectiveness of a random forest classifier for land-cover classification. **ISPRS Journal of Photogrammetry and Remote Sensing**, Elsevier, v. 67, p. 93–104, 2012.

ROOM, C. N-gram model. **algorithms**, v. 3, n. 16, p. 46, 2021.

RUDER, S. **Neural transfer learning for natural language processing**. 2019. Tese (Doutorado) — NUI Galway, 2019.

SANTOS, D. et al. An advanced ner evaluation contest for portuguese.

SCHIESSL, J. M. Descoberta de conhecimento em texto aplicada a um sistema de atendimento ao consumidor. 2007.

SINGH, G. et al. Comparison between multinomial and bernoulli naïve bayes for text classification. In: IEEE. **2019 International Conference on Automation, Computational and Technology Management (ICACTM)**. [S.l.], 2019. p. 593–596.

SOUZA, F. C. d. et al. Bertimbau: pretrained bert models for brazilian portuguese= bertimbau: modelos bert pré-treinados para português brasileiro. [sn], 2020.

SRINIVASAN, S.; ARORA, R.; RIEDL, M. A simple and effective approach to the story cloze test. **arXiv preprint arXiv:1803.05547**, 2018.

TAY, Y.; TUAN, L. A.; HUI, S. C. A compare-propagate architecture with alignment factorization for natural language inference. **arXiv preprint arXiv:1801.00102**, v. 78, p. 154, 2017.

_____. Multi-range reasoning for machine comprehension. **arXiv preprint arXiv:1803.09074**, 2018.

TIAN, J. et al. Ecnu at semeval-2017 task 1: Leverage kernel-based traditional nlp features and neural networks to build a universal model for multilingual and cross-lingual semantic textual similarity. In: **Proceedings of the 11th international workshop on semantic evaluation (SemEval-2017)**. [S.l.: s.n.], 2017. p. 191–197.

VASWANI, A. et al. Attention is all you need. **arXiv preprint arXiv:1706.03762**, 2017.

VIEIRA, R.; LIMA, V. L. S. Lingüística computacional: princípios e aplicações. In: SN. **Anais do XXI Congresso da SBC. I Jornada de Atualização em Inteligência Artificial**. [S.l.], 2001. v. 3, p. 47–86.

WANG, A. et al. Glue: A multi-task benchmark and analysis platform for natural language understanding. **arXiv preprint arXiv:1804.07461**, 2018.

WOLF, T. et al. Huggingface's transformers: State-of-the-art natural language processing. **arXiv preprint arXiv:1910.03771**, 2019.

WU, Y. et al. Google's neural machine translation system: Bridging the gap between human and machine translation. **arXiv preprint arXiv:1609.08144**, 2016.

XU, Y. et al. Towards human-level machine reading comprehension: Reasoning and inference with multiple strategies. **arXiv preprint arXiv:1711.04964**, v. 105, 2017.

_____. Improving bert fine-tuning via self-ensemble and self-distillation. **arXiv preprint arXiv:2002.10345**, 2020.

YANG, Y. et al. Paws-x: A cross-lingual adversarial dataset for paraphrase identification. **arXiv preprint arXiv:1908.11828**, 2019.

ZHANG, Z. et al. Ernie: Enhanced language representation with informative entities. **arXiv preprint arXiv:1905.07129**, 2019.

ZHOU, J. et al. Deep recurrent models with fast-forward connections for neural machine translation. **Transactions of the Association for Computational Linguistics**, MIT Press, v. 4, p. 371–383, 2016.